Development and Research of a Non-Periodic Pseudorandom Number Generator*

Andriy Horpenyuk[1,†], Mykola Horpenyuk[1,†], Nataliya Luzhetska[1,†], Oleksandr Horpenyuk[1,†] and Alona Desiatko[2,*,†]

[1] Lviv Polytechnic National University, 12 Stepan Bandera str., 79000 Lviv, Ukraine
[2] State University of Trade and Economics, 19 Kyoto str., 02156 Kyiv, Ukraine

Abstract

This paper presents the results of the development and research of a pseudorandom number generator based on computing the square root of a prime number. The square root of a prime number is one of the known irrational numbers. According to a well-known hypothesis in mathematics, the sequence of digits of such numbers is a pseudorandom non-periodic digit sequence. Therefore, calculating the square root of a prime number can serve as the basis for constructing a pseudorandom number generator. The key of such a generator can be a simple number and the digit position number from which the regeneration process starts. To efficiently compute the square root in a "bit by bit" style, a modified bisection method is proposed. The proposed method allows saving one multiplication operation in each iteration of the bisection method. At the same time, the complexity of one iteration of the proposed method is close to the complexity of a single addition operation.

Keywords

pseudorandom number generator, stream cipher, prime number, irrational number

1. Introduction

The problem of producing high-quality keys for cryptographic applications is an important and pressing issue [1]. It is better to use true random numbers for key generation. However, generating such numbers is a slow and costly process. Moreover, a true random number cannot be reproduced. Therefore, the problem of exchanging a long random key arises. For these reasons, not true random numbers but pseudorandom numbers are often used as cryptographic keys [2, 3]. Statistically, they are indistinguishable from random numbers, but they have a very important advantage. A pseudorandom number can be reproduced. Thanks to this, subscribers can exchange relatively short secret parameters of an identical pseudorandom number generator. Based on the same parameters, such a generator will produce an identical key for both subscribers [4–6].

One of the important characteristics of a pseudorandom number generator is the period of the generated bit sequence [7]. Only a part of the generated sequence up to the point of repetition can be used as a key. Otherwise, the cipher can be easily broken. Therefore, it is desirable for the period of the generated sequence to be as long as possible. Moreover, the requirements for the period are constantly increasing due to the growth of key sizes in modern cryptosystems. This is especially relevant for post-quantum asymmetric cryptosystems, which are currently being actively implemented [8, 9].

It is often stated that all pseudorandom numbers are periodic. However, this is not entirely true. Hypothetically, there exist non-periodic pseudorandom numbers. Among them are irrational numbers [10].

* CQPC 2025: Classic, Quantum, and Post-Quantum Cryptography, August 5, 2025, Kyiv, Ukraine
* Corresponding author.
† These authors contributed equally.
andrii.y.horpeniuk@lpnu.ua (A. Horpenyuk); mykola.horpeniuk.kb.2023@lpnu.ua (M. Horpenyuk); nataliia.m.luzhetska@lpnu.ua (N. Luzhetska); oleksandr.horpeniuk.kb.2024@lpnu.ua (O. Horpenyuk); desyatko@gmail.com (A. Desiatko)
0000-0001-5821-2186 (A. Horpenyuk); 0009-0001-1577-2068 (M. Horpenyuk); 0000-0002-5449-5825 (N. Luzhetska); 0009-0000-9478-6283 (O. Horpenyuk); 0000-0003-2860-2188 (A. Desiatko)

In mathematics, there is a well-known hypothesis regarding the normality of irrational and transcendental numbers, in particular, the square roots of prime numbers [10, 11]. This means that the sequence of digits of the square root of a prime number is pseudorandom digit sequence.The mentioned hypothesis has not been proven yet. It remains one of the most famous unsolved problems in mathematics. Interest in calculating the square root of a prime number arose a very long time ago [12]. In particular, among collections of Babylonian historical artifacts preserved at Yale University, there is a round clay tablet (Fig. 1). It is dated to 1750 BC. The tablet shows a square divided by diagonals. Three digits are clearly inscribed on it with cuneiform characters. When the inscription was deciphered, it became clear that almost 4000 years ago in Babylon they already knew how to determine the diagonal of a square based on its side length by multiplying the side by the square root of two. The markings on the tablet provide an approximate value of $\sqrt{2}$ in four sexagesimal digits, which corresponds to eight decimal digits $1+\frac{24}{60}+\frac{51}{60^2}+\frac{10}{60^3}=1.41421296$

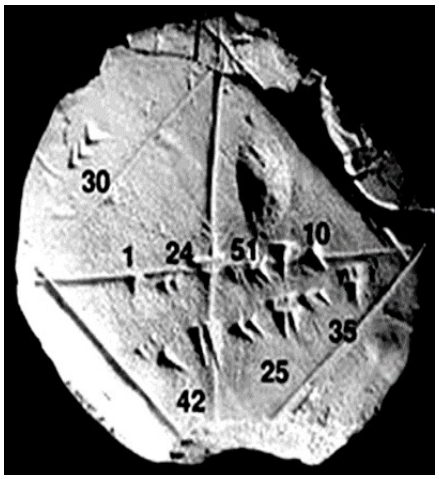

Figure 1: Babylonian clay tablet

The task of calculating as many digits as possible of the square root of a prime number remains relevant today. One of the notable modern achievements in this area is the work of Professor Jacques Dutka, a staff member of the Department of Mathematical Methods in Engineering at Columbia University. He developed an algorithm and calculated the value of the square root of two up to the one-million eighty-second decimal place. However, the record was soon surpassed multiple times. In particular, in 2010, Shigeru Kondo computed one trillion decimal digits of the square root of two [13]. At that time, only the number π had been computed with greater precision (five trillion decimal digits). The modern relevance of this computational task is explained both by the desire to empirically confirm the pseudorandomness of the digit sequence of the root and by the wide range of applications for pseudorandom numbers, particularly in cryptography.

In addition to pseudorandomness, the digit sequence of the square root of a prime number has two other important features that make such sequences valuable from the perspective of modern cryptography. The first feature is that the digit sequence of the square root of a prime number (according to the hypothesis) is non-periodic. Therefore, such sequences of digits (or bits) can be used in stream ciphers for encrypting large volumes of information, including images, video, and audio files. The second feature is that the chosen prime number, as well as the digit number from which the "refinement" of the root value starts, can serve as secret parameters (a key) for a pseudorandom number generator based on computing the square root of a prime number.

2. Literature review and problem statement

This work will demonstrate an efficient method for constructing a non-periodic pseudorandom number generator based on computing the square root of a prime number. We will review some

computational methods traditionally used for calculating square roots. We will also analyze these methods in terms of their suitability for use in a pseudorandom number generator (PRNG).

The Babylonian method. For the purpose of calculating the square root of two, the ancient Babylonian method for computing square roots is often used [10, 14, 15]. It is based on the following principle:

$$a_{n+1}=\frac{a_n+\frac{2}{a_n}}{2}=\frac{a_n}{2}+\frac{1}{a_n}. \quad (1)$$

The more repetitions in the algorithm (that is, the more iterations are performed and the larger "n" becomes), the better the approximation of the square root of two. Each iteration approximately doubles the number of correct digits.

The long division method for calculating the square root. This method used to be taught in schools. The advantage of this method is that at each step of the algorithm, one additional correct digit of the result is obtained [16], which also becomes the next digit of the pseudorandom sequence. The disadvantages of the method are that it is not systematic: at each step, the method's parameters must be selected manually. In addition, the remainder at each subsequent step of the algorithm can be twice as long as the previous remainder. Therefore, this method is poorly suited for the automated calculation of a large number of root digits, which is required for generating pseudorandom sequences.

Method of calculating the square root via Taylor series expansion:

$$\sqrt{1+x}=\sum_{n=0}^{\infty}\frac{(-1)^2(2n)!}{(1-2n)(n!)^2(4^n)}x^n=1+\frac{1}{2}x-\frac{1}{8}x^2+\frac{1}{16}x^3-\frac{5}{128}x^4+\ldots. \quad (2)$$

This method [15] provides good convergence, but it involves complex computations, which are inconvenient when calculating a large number of root digits.

Method of arithmetic extraction of the square root. For square numbers, the following rule applies [15] $1 = 1^2$, $1 + 3 = 2^2$, $1 + 3 + 5 = 3^2$, and so on.

That is, the integer part of the square root of a number can be determined by successively subtracting all odd numbers from it until the remainder becomes zero or is less than the next odd number to be subtracted. The result (the integer part of the root) is the number of performed actions (subtractions). For example, 9 – 1 = 8, 8 – 3 = 5, 5 – 5 = 0.

Three steps were performed, so the square root of 9 is 3.

The disadvantage of this method of calculation is that as a result of its application, only the integer part of the result can be obtained. At the same time, this method is useful for obtaining a rough initial approximation of the root.

The iterative analytical algorithm (Heron's iterative formula). This is a popular and very ancient iterative algorithm [17], which allows refining the solution using the formula:

$$\begin{cases} x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right), \\ x_0=a, \end{cases} \quad (3)$$
$$\lim_{n\to\infty} x_n=\sqrt{a}.$$

The method converges quickly: each iteration approximately doubles the number of correctly computed digits of the result. At the same time, the method involves complex operations, including division.

Numerical methods of approximate square root calculation. These methods [18] include, in particular, the bisection method, the secant method, the tangent method (Newton's method), and others. Most of these methods (except for the bisection method) do not allow for the computation of the square root "digit by digit."

To compute square roots, specialized function transformations can also be used, for example, number-to-impulse transformations based on classical digital converters [18–20].

For use in PRNGs (pseudorandom number generators), most of the listed methods are unsuitable. This is due to two reasons. First, a PRNG typically needs to generate a long sequence of pseudorandom bits. When constructing such a generator based on the calculation of the square root of a prime number, this means that a large number of digits of the square root need to be computed. Moreover, each subsequent digit (bit) of the result must be computed precisely. Otherwise, the pseudorandomness property of the generated sequence may be lost. Additionally, the digits of the square root, which become bits of the pseudorandom sequence and can be used further in fast stream ciphers, must be computed quickly.

Thus, for use in a PRNG based on the square root calculator of a prime number, a fast and accurate “digit-by-digit” calculator is suitable. Accordingly, various methods for rough estimation of the square root value are only suitable for calculating the initial approximation. Methods such as columnar calculation and most of the approximate numerical methods (Heron's method, secant, tangent, and others) are also poorly suited. The main reason for their unsuitability in building a PRNG is the high complexity of computational operations, which increases with the number of computed digits. At the same time, it is worth noting that the columnar calculation method allows for the "digit-by-digit" computation of the result.

Considering the aforementioned application characteristics, the bisection method, or the method of bisections, is convenient for constructing a PRNG calculator. This method involves relatively simple computational operations. It has slow convergence, but it is fast and, under certain conditions, can be converted into a “digit-by-digit” method. In terms of performance, for a PRNG, the speed of generating the exact value of the next bit in the sequence is more important than the overall speed of generating the entire sequence without guaranteeing the accuracy of individual bits.

3. Development of an improved square root calculation algorithm for a PRNG

Let us formulate an observation regarding the bisection method that makes it possible to transform this method into a “digit-by-digit” square root calculation method.

If the width of the initial approximation interval in the bisection method (the interval in which the root of the nonlinear equation $x^2 = p$ is localized) equals an exact power of two, then at each step of the bisection method we obtain the next correct bit of the square root result.

Thus, if the formulated requirements for the initial approximation of the root are met, we can construct an algorithm for calculating the square root using the “digit-by-digit” method.

Therefore, the bisection method can be used to construct a pseudorandom number generator based on computing the square root of a prime number. Compared to other square root calculation methods, the bisection method is characterized by lower computational complexity, although it often has slower convergence. At the same time, the bisection method allows for the selection of an initial approximation that enables “bit-by-bit” computation of the square root, with each step of the algorithm yielding the exact value of the next bit of the result – an essential requirement when generating a pseudorandom bit sequence.

Another important characteristic of a pseudorandom number generator is its speed. The low complexity of the basic operations in the bisection method can ensure this performance characteristic. This is why the bisection algorithm is well-suited for computing the square root of large numbers. At the same time, when using a square root calculator to construct a pseudorandom number generator, a number of improvements to the bisection method can be proposed, which would significantly reduce the computational effort required. The essence of the proposed improvements is as follows:

1 The width of the initial approximation interval must be an exact (positive or negative) power of two. This allows for the precise computation of the next bit of the result at each step of the algorithm. Furthermore, as will be shown later, this makes it possible in the bisection algorithm to eliminate the need for monitoring the right boundary of the root localization interval, as well as to compute the midpoint of the interval by simply appending a one to the next bit.

2 When generating a pseudorandom sequence, which in the case of using a square root calculator of a prime number, is non-periodic—we can begin the generation from any bit. In doing so, it is necessary to provide an appropriate initial approximation of the root, i.e., the position of point a within the root localization interval (see Fig. 2). The width of the root localization interval should be chosen according to the requirements of point 1. As will be shown below, this allows for a significant simplification of the computation of root localization conditions.

3 For clarity and unambiguity in the further formal exposition, we choose the initial approximation of the root (the position of point a in the localization interval (see Fig. 2)) to be equal to the integer part of the root value.

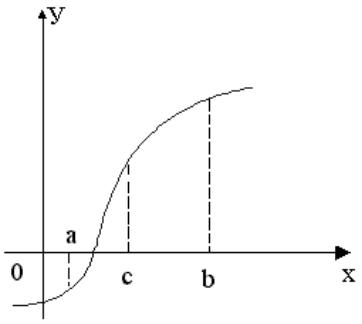

Figure 2: Illustration of the bisection method

In doing so, and in accordance with point 1, it is assumed that the right boundary of the root localization interval (point *b*) is located one unit away from point *a*.

4. Development of the operating equation of the PRNG calculator

We develop the generator's operating equation based on the bisection method, taking into account the proposed improvements to the method. We need to compute the value of x, the square root of a prime number p.

$$x = \sqrt{p}. \tag{4}$$

To do this, we apply the bisection method, which we use to refine the initial approximation of the root of a nonlinear quadratic equation.

$$x^2 - p = 0. \tag{5}$$

As the initial approximation x_0 (the left boundary of the root localization interval in the bisection method), we choose the integer part of x is the greatest integer whose square is less than the number p. Obviously, at the initial approximation point, the value of the function (5) is negative. In the formulated proposals, in addition to the initial approximation x_0, we also proposed the width of the localization interval one unit. Therefore, under these conditions, to find the midpoint of the localization interval, we need to add half of the localization interval to the initial approximation x_0 in our case, 1/2, that is, $x_0 + 2^{-1}$. Next, we need to compute the square of the midpoint and compare the result with the number p. The square of the midpoint, based on the initial approximation and considering the accepted assumptions, is determined by the following relation:

$$(x_c)^2 = y_c = (x_0 + 2^{-1})^2 = x_0^2 + 2x_0 \cdot 2^{-1} + 2^{-2} = x_0^2 + x_0 + 2^{-2} = y_0 + x_0 + 2^{-2}. \tag{6}$$

Next, we compare the computed y_c with p. If $y_c < p$, we set $x_1 = x_c$, $y_1 = y_c$.

Otherwise $x_1 = x_0$, $y_1 = y_0$. We then proceed further, and at the i^{th} step of the algorithm, we obtain:

$$(x_c)^2 = y_c = (x_{i-1} + 2^{-i})^2 = x_{i-1}^2 + 2^{-i+1} x_{i-1} + 2^{-2i} = y_{i-1} + 2^{-i+1} x_{i-1} + 2^{-2i}. \quad (7)$$

We again compare with p, and based on the result of the comparison, we determine the value of the next bit of the result. We continue in a similar manner.

Analyzing expression (7), we conclude that the addition of the term 2^{-2i} is practically implemented by writing a one in the corresponding bit position (since, before the i^{th} step of the algorithm, all lower bits of the number y, starting from the bit with weight 2^{-2i+2},are zero). The term $2^{-i+1}x_{i-1}$ is formed by shifting the number x_{i-1}. Thus, to compute the new y_i according to formula (7), we need to write a one in the next bit, shift the number x_{i-1}, and add it to y_{i-1}. In terms of complexity, this set of operations is close to a single addition.

Next, the calculated number is compared with p, and based on the comparison results, new values of x_i and y_i are established. The new value x_i is set by appending either a zero or a one (the newly calculated correct bit!) to the next lower bit of the result. Thus, if we do not take into account simple operations such as bit writing, shifting, and reassignment, the proposed algorithm requires only one addition operation per correct bit of the result.

5. Development of the PRNG algorithm based on a square root calculator from prime numbers

Based on the proposed operating principle of the generator and the derived operating equation of its calculator (3), an algorithm for the operation of a PRNG based on a square root calculator from prime numbers was developed.

The structure of the developed algorithm is presented in Fig. 3. The following notations are used in the structure of the algorithm: x_0 and y_0 are initial approximations of the result x and its square; *pp* is a given prime number; *s* is a number of bits in the integer part of the result; *yy* is a intermediate value of the square of the *result*, calculated according to (7); *p* is a carry value for the next bit during addition.

The developed algorithm (Fig. 3) provides for the generation of a 20,000-bit sequence for the purpose of further analysis of the statistical properties of the generated sequence in accordance with FIPS 140 standard. Therefore, the values *x*, *y*, and *yy*, which are large numbers, appear as bit arrays in the algorithm shown in Fig. 3. Taking into account the term 2^{-2i} added to y when calculating *yy* according to (7), as well as the need to shift *x* as per (7), the size of these arrays is increased to 40000 + 2*s*.

In the proposed algorithm (Fig. 3), the main loop generates 20,000 bits of the square root of the given prime number *pp*. To do this, according to (7), the square of the midpoint on the current localization interval is calculated. The addition of the term is implemented by writing a one into the *yy*[*b*] bit. The required shift of the number x according to (7) is implemented by modifying the index of the *x* array, followed by bitwise addition of **y** and the shifted *x* with carry to the next digit. After calculating *yy* according to (7), *yy* is compared with the given prime number. If the prime number is greater, then according to (7), the function is negative at the current midpoint of the localization interval (under the assumed conditions, the function at the right point of the interval is always positive). In this case, the left boundary of the localization interval should shift to the midpoint *x* is increased by appending a one in the corresponding digit, and the intermediate *yy* becomes the new *y*. If the intermediate *yy* in the current iteration is greater than the prime number *pp*, we stay at the previous left boundary of the localization interval (assuming the right boundary has moved to the midpoint).

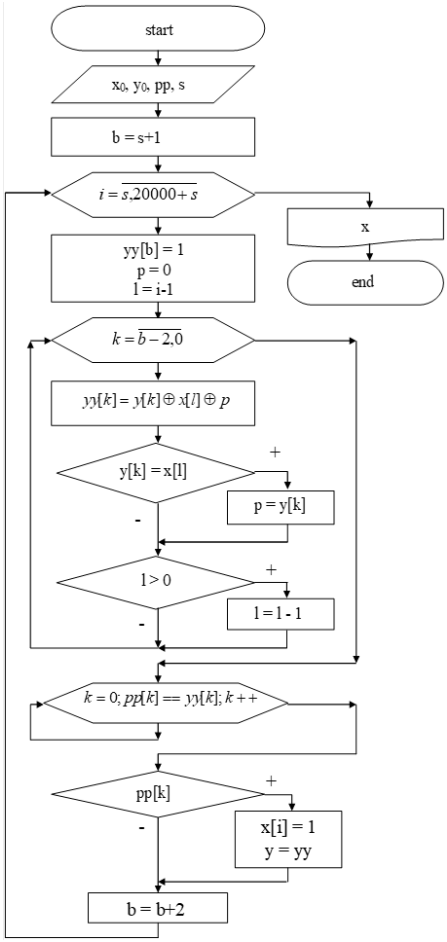

Figure 3: Structure of the operation algorithm of the PRNG on a square root calculator from a prime number

In both cases, the increment of x for computing the next bit is halved, and we proceed to the next iteration of the algorithm.

6. Development of algorithms for testing the generated bit sequence according to FIPS 140

The FIPS 140 standard defines four statistical randomness tests: the monobit test, the block test (poker test), the runs test, and the long runs test. Each test defines thresholds for acceptable statistical parameters. A separate bit sequence of length 20,000 generated by the PRNG is tested using all four tests. If any test fails, the generator is considered to have failed the entire test suite.

Monobit test. The essence of this test lies in counting the number of zeros and ones in the generated bit sequence of a given length (in this standard, the sequence length is 20,000 bits). Let n_1 and n_2 denote the number of zeros and ones in the sequence x, respectively. If the sequence is random, the values of n_1 and n_2 must satisfy the condition $9654 < n_1 (n_2) < 10346$.

The structure of the developed monobit test algorithm is shown in Fig. 4.

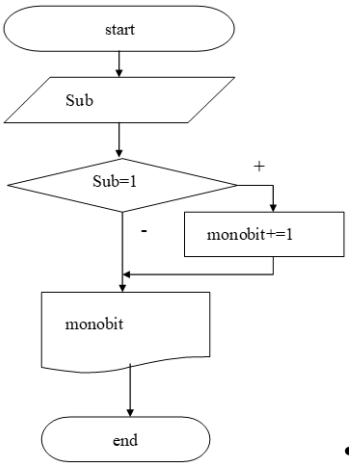

Figure 4: Structure of the monobit test algorithm

Block test. Let m be a positive integer such that:

$$k = \frac{n}{m} \geq 5 \cdot (2^m). \tag{8}$$

The sequence is divided into non-overlapping subsequences of length m = 2, 3, … . Let be the number of occurrences of the i^{th} type of subsequence of length m. The block test determines whether the subsequences of length m appear approximately the expected number of times in the sequence x that is, each subsequence should occur approximately equally often, as would be expected in a truly random sequence. To apply the criterion, the following parameter is computed:

$$X_3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k, \tag{9}$$

which follows a distribution close to the χ^2 with $2^m - 1$ degrees of freedom. The statistical parameter defined by the equation is calculated for m = 4. The statistic must satisfy the following condition $1.03 < X_3 < 57.4$.

The structure of the developed block test algorithm is shown in Fig. 5.

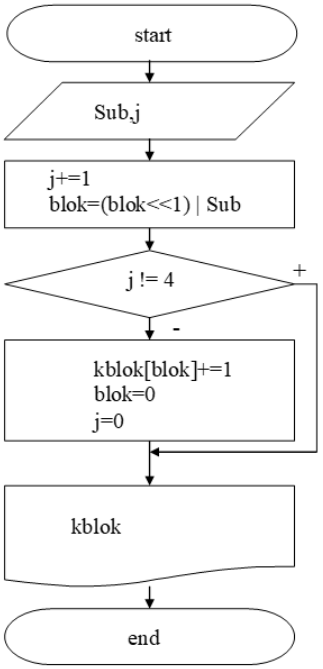

Figure 5: Structure of the block test algorithm

Here, *blok* is a four-bit block of the generated sequence, the array *kblok* accumulates the number of blocks of each type (by index), and *Sub* is the next generated bit.

Runs test. A *run* is defined as a sequence of identical symbols—either ones or zeros. The essence of the test is to count the number of runs of lengths 1, 2, 3, 4, 5, and 6 in the tested sequence of a given length (runs longer than 6 elements are treated as runs of length 6). If the sequence is random, the number of runs of each length should fall within the following intervals (Table 1).

Table 1
Number of runs

Run Length	Required Interval
1	2267–2733
2	1079–1421
3	502–748
4	223–402
5	90–223
6	90–223

That is, as the run length increases by one, the number of runs approximately halves.

Long run length test. The essence of this test lies in verifying the maximum length of a run of identical elements (ones or zeros). If the sequence is random, the maximum run length must not exceed 34.

The structure of the developed combined algorithm for the runs test and the long run length test is shown in Fig. 6.

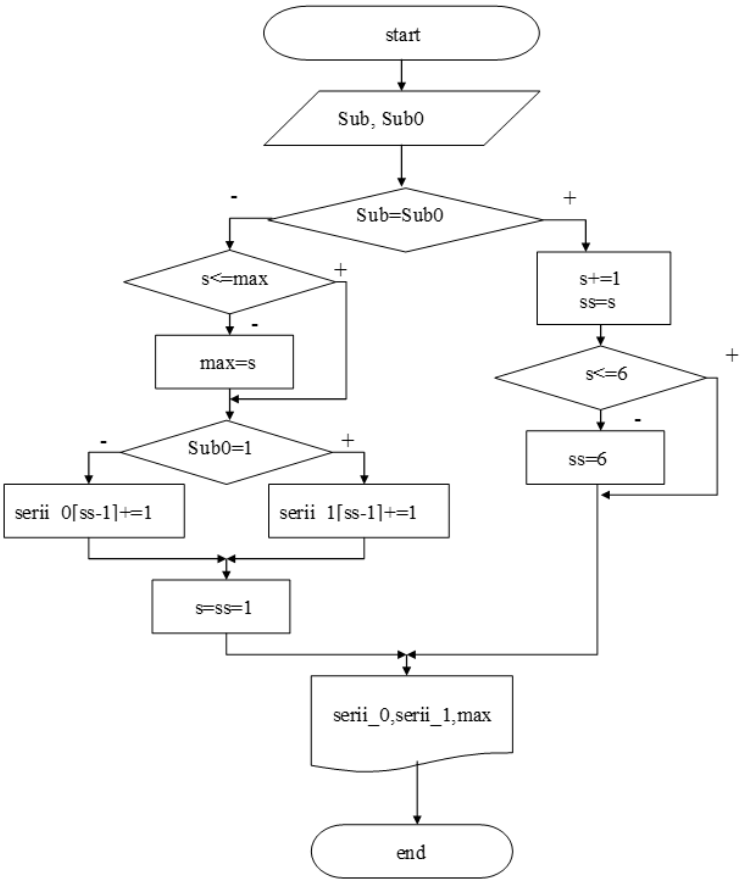

Figure 6: Structure of the algorithm for the runs and long run length tests

Here, *SS* denotes the run length (not exceeding 6), *S* is the absolute length of the run, *Sub* is the current generated bit, and *Sub0* is the previously generated bit. The arrays *serii_1* and *serii_0* accumulate the number of runs of the corresponding (indexed) length. *max* stores the maximum run length.

7. Comparison of the developed algorithm with the baseline

Analyzing the proposed algorithm (Fig. 3), we conclude that generating a single bit of the pseudorandom sequence x requires one addition, one comparison, and, optionally, one reassignment. In contrast, in the classical bisection algorithm applied to our function (5), computing each bit of the result requires an addition and shift to determine the midpoint of the localization interval, a multiplication to compute the square of the midpoint, a comparison, and two reassignments.

Thus, the proposed algorithm, compared to the classical one, saves one multiplication per bit of the result. Considering that the approximate complexity of the proposed algorithm is one addition per bit of the result, this gain is significant, especially in view of the lengths of bit sequences that must be generated in stream ciphers using PRNGs.

Based on the proposed algorithm for the operation of the PRNG on a square root calculator of prime numbers (Fig. 3), as well as the developed PRNG testing algorithms according to the FIPS 140 standard (Figs. 4-6), a software implementation of the generator based on the square root calculator of a prime number was developed. The implementation enables the generation of a pseudorandom sequence in accordance with the proposed method for computing the square root of a prime number. It also provides the capability to investigate the statistical properties of the generator.

In the process of studying the statistical properties of the developed generator, a sequence of 20,000 pseudorandom bits was generated using its software model. During this process, the generated sequence was tested for compliance with the requirements of the FIPS 140 standard. The results of the statistical testing of the generator for three small prime numbers are summarized in the Table 2.

Table 2
Results of the statistical characteristic evaluation of the generator

	Monobit test	Block test	Run test						Long run length test
			1	2	3	4	5	6	
$\sqrt{3}$	10036	10	2433	1261	632	350	157	143	14
			2465	1259	610	336	172	134	
$\sqrt{17}$	10014	18	2527	1240	614	310	163	158	12
			2526	1234	621	335	144	153	
$\sqrt{31}$	10005	24	2522	1225	630	339	181	127	11
			2531	1215	679	316	132	150	

The results of the runs test for ones are presented in the upper row, and for zeros in the lower row.

Conclusions

The paper presents the results of the development and investigation of the structure of a pseudorandom number generator based on a square root calculator of a prime number. The square root of a prime number is an irrational number. Therefore, the sequence of its digits hypothetically forms a non-periodic pseudorandom sequence. An improvement to the bisection method is proposed in order to enable its application in the PRNG based on the square root calculator of a

prime number. An algorithm and software implementation of the PRNG based on the square root calculator of a prime number have been developed. The statistical characteristics of the PRNG based on the improved square root calculator have been investigated. The results of the conducted research show that the application of the improved bisection method allows a significant enhancement in performance due to the reduction of computational workload per multiplication during each iteration. In the context of the PRNG built upon such a method, this means that computing a single bit of the pseudorandom sequence requires only one addition operation. The evaluations of the statistical characteristics of the developed generator confirm that the quality of the pseudorandom sequences it generates meets the requirements of the FIPS 140 standard.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] B. Schneier, Applied cryptography: Protocols, algorithms and source code, in: C. John Wiley and Sons, New York, 2nd ed., 1998. doi:10.1002/9781119183471

[2] S. Popereshnyak, Y. Novikov, Y. Zhdanova, Cryptographic system security approaches by monitoring the random numbers generation, in: Cybersecurity Providing in Information and Telecommunication Systems II (CPITS-II), 3826, 2024, 301–309.

[3] L. Blum, M. Blum, M. Shub, A simple unpredictable pseudo-random number generator, SIAM J. Comput. 15(2) (1986) 364–383. doi:10.1137/0215025

[4] A. Bessalov, et al., Computing of odd degree isogenies on supersingular twisted Edwards curves, in: Cybersecurity Information and Telecommunication Systems, 2923, 2021, 1–11.

[5] A. Bessalov, et al., CSIKE-ENC combined encryption scheme with optimized degrees of isogeny distribution, in: Cybersecurity Providing in Information and Telecommunication Systems, 3421, 2023, 36–45.

[6] A. Bessalov, V. Sokolov, S. Abramov, Efficient commutative PQC algorithms on isogenies of Edwards curves, Cryptography 8(3), iss. 38 (2024) 1–17. doi:10.3390/cryptography8030038

[7] O. Harasymchuk, et al., Modern methods of ensuring information protection in cybersecurity systems using artificial intelligence and blockchain technology, Technology Center PC., 2025. doi:10.15587/978-617-8360-12-2

[8] A. Horpenyuk, I. Opirskyy, P. Vorobets, Analysis of problems and prospects of implementation of post-quantum cryptographic algorithms, in: Classic, Quantum, and Post-Quantum Cryptography (CQPC), 3504, 2023, 39–49.

[9] P. Vorobets, et al., Implementing post-quantum KEMs: Practical challenges and solutions, in: Cybersecurity Providing in Information and Telecommunication Systems II, 3826, 2024, 212–219.

[10] D. Proskurin, et al., Hybrid RNN-CNN-based model for PRNG identification, in: Workshop on Classic, Quantum, and Post-Quantum Cryptography (CQPC), 3829, 2024, 47-53.

[11] A. Bessalov, et al., Multifunctional CRS encryption scheme on isogenies of nonsupersingular Edwards curves, in: Classic, Quantum, and Post-Quantum Cryptography, 3504, 2023, 12–25.

[12] J. L. Beery, F. J. Swetz, The best-known old Babylonian tablet? Convergence, 2012. doi:10.4169/loci003889

[13] Constants and records of computation, 2010. https://web.archive.org/web/20120301190937/http://numbers.computation.free.fr/Constants/Miscellaneous/Records.html

[14] D. E. Knuth, Ancient Babylonian algorithms, Commun. ACM 15(7) (1972) 671–677.

[15] A. Flores, The Babylonian method for approximating square roots: Why is it so efficient? The Mathematics Teacher, 108, 2014.

[16] C. Baumann, Playing with the square root. A practical study, ver. 1.5, 2024, 1–54. https://computarium.lcd.lu/literature/COMPUTARIUM_CREW/BAUMANN/report_squareroot_1.5.pdf
[17] A. Ralston, P. Rabinowitz, A first course in numerical analysis, 2nd ed., McGraw-Hill, 1978.
[18] A. Horpenyuk. V. Dudykevych, N. Luzhetska, Conveyor sine-cosine pulse-number functional converter, Autom. Meas. Control 639 (2009) 94–101. [In Ukrainian].
[19] P. Montuschi, M. Mezzalama, Survey of square rooting algorithms, IEE Proc. E Comput. Digit. Tech. 137(1) (1990) 31. doi:10.1049/ip-e.1990.0003
[20] A. Gorpeniuk, Fast algorithms and computing means of cryptological functions, Int. Sci. J. Comput. 4(2) (2005) 69–76. doi:10.47839/ijc.4.2.339