International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

Journal Website: http://sciencebring.com/index.php/ijasr

Research Article

Effective Hybrid Algorithm For Generating Large Prime Numbers

Submission Date: March 21, 2026, **Accepted Date:** April 16, 2026,
Published Date: May 18, 2026
Crossref doi: https://doi.org/10.37547/ijasr-06-05-07

Xudoykulov Z. T.
Professor of department Cryptology, TUIT named after Muhammad al-Khwarizmi Tashkent, Uzbekistan

Jabbarov N. A.
Assistant of department Cryptology, TUIT named after Muhammad al-Khwarizmi Tashkent, Uzbekistan

Jabborov A. A.
Student of the Samarkand branch of the TUIT named after Muhammad al-Khwarizmi Samarkand, Uzbekistan

Abstract

This article addresses the problem of generating large prime numbers. Due to the high computational complexity of traditional methods, their efficiency remains low. Therefore, the paper analyzes existing approaches and proposes a new hybrid algorithm, based on algebraic and probabilistic tests. The proposed method demonstrates the ability to generate prime numbers quickly and reliably. The results are compared across different bit lengths (256, 512, 1024, 2048 bits) using experimental data, and the efficiency of the algorithm is presented. The research outcomes have practical significance for generating secure keys in public-key cryptosystems (e.g., RSA).

Keywords

Prime number, primality test, Miller–Rabin, random number generation, cryptography, RSA, hybrid algorithm, hybrid approach.

Introduction

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

ISSN-2750-1396

Prime number generation is a fundamental primitive in most public-key schemes, used primarily as a computational step in key pair generation or in various cryptographic applications. Surprisingly, despite decades of active mathematical research in the field of number primality testing and the growing use of cryptographic applications, prime number generation algorithms have not been sufficiently studied, and most of their practical implementations remain inefficient.

Traditional algorithms for generating prime numbers require asymptotically $O(n^4)$ or $O(n^4/\log n)$ bit operations, where n is the bit length of the expected prime number. When generating finite prime numbers (e.g., secure or quasi-secure), the complexity can reach $O(n^5/(\log n)^2)$ [1]. Experience has shown that it is impossible to significantly improve this asymptotic complexity in a short time for the generation of large prime numbers. This article proposes simple algebraic methods and thus presents more efficient algorithms for generating prime numbers.

Tests for checking prime and composite numbers. Research on checking primality has been conducted for many years and can be found in specialized literature (e.g., [2]). From a computational point of view, numbers can be divided into actual primes and probable primes, the difference being in the way they are generated. A probable prime number is usually determined using a compositeness test. As a result of such a test, the number is declared composite with 100% certainty or confirmed as prime with a probability of < 1. Thus, the level of confidence in the obtained probable prime number is increased by repeating the test multiple times. Typical examples of complexity tests are the Fermat test, the Solovay–Strashen test [3], and the Miller–Rabin test.

Fermat's criterion. If p is a prime number, then, according to Fermat's little theorem, the equality $a^{n-1} \equiv 1\ (mod\ n)$ holds, where a is an arbitrary number, and a n is not a multiple of a. The equality $a^{n-1} \equiv 1\ (mod\ n)$ is a necessary and sufficient condition for determining the power of the number n. That is, if for some a, $a^{n-1} \neq 1\ (mod\ n)$, then n is a complex number; otherwise, it is difficult to say anything definite, but the probability that the number will be cardinal increases. If the equality $a^{n-1} \equiv 1\ (mod\ n)$ holds for a complex number n, then the number n is called a pseudobase with base a.

However, when n is a complex number, there exists a number a such that the comparison $a^{n-1} \equiv 1\ (mod\ n)$ does not hold. Such a number a is called a witness to the complexity of n, and the previous number a with which this comparison is made is called a “false witness” to the power of n. Therefore, when checking a number for primality according to Fermat's theorem, a certain number a is chosen. The more a satisfies the condition $a^{n-1} \equiv 1\ (mod\ n)$, the more likely it is that the number n is prime. However, there are complex numbers n such that the equality $a^{n-1} \equiv 1\ (mod\ n)$ holds for any prime number n a. Such numbers are called Carmichael numbers. The set of Carmichael numbers is infinite, and the smallest of them is $n = 561 = 3 \cdot 11 \cdot 17$

Solovay–Strassen test. The test always correctly determines the primality of a prime number, but for complex numbers it may give an incorrect

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

answer with a certain probability. The main advantage of the test is that, unlike Fermat's test, it identifies Carmichael numbers as complex. If n is a prime number and a is relatively prime to the integer n, then the condition $a^{n-1} \equiv 1\ (mod\ n)$ always holds. In this case, the value of a is in the range $0 < a < n - 1$.

The essence of the test is that k checks not every number in the entire sequence, but a random set of each random number.

In this case, the Jacobi symbol is used to identify Carmichael numbers.

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} (mod\ n)$$

Here $\left(\frac{a}{n}\right)$ is Legendre's symbol, used to determine the power of the number n.

This test uses Euler's criterion. It is known that according to Euler's criterion, if for all prime numbers a that do not have a greatest common divisor with n, the condition $a^{(n-1)/2} = (\frac{a}{n})\ mod\ n$ is satisfied, then the odd number n is prime. That is, this algorithm focuses on the elements 1 and -1 that are in the column corresponding to the "proof of primality" a, which is a common prime number with n, in row $(n-1)/2$ of the table of prime numbers.

If the number of witnesses of a prime number n is equal to the number of k-fold repetitions, then the number n is prime with probability $1 - 2^{-k}$.

Miller–Rabin test. The Rabin–Miller test is a probabilistic primality test based on the concept of a strong pseudoprime.

The Miller–Rabin test for the primality of large numbers is currently widely used in symmetric cryptosystems for module generation. This algorithm is recognized as a robust algorithm for testing the pseudoprimality of numbers. It is based on representing $n - 1$, i.e., one reduced modulus value, as $2^s * r$. Where s is the number of divisors of $n - 1$ by two, and r is an odd number.

This test is performed in the following sequence:

- $n - 1 = 2^s * r$, where r is an odd number.
- $2 \le a \le n - 1$ $2 \le a \le n - 1$ a number a satisfying the condition is chosen at random.
- $y = a^r mod n$ is calculated.
- If (y = 1 or $u = n - 1$), then proceed to the next iteration.
- Calculate $a^{2^t r} \equiv -1\ (mod\ n)$.
- $t < s$ times, $y = y^2 mod n$.
- If $(y = 1)$, then return "complex number".
- If $(y \neq n - 1)$, then return "complex number".
- Otherwise, the "fundamental number" is returned.

There are also true tests for prime numbers that confirm 100% accuracy in determining whether a number is prime (for example, the Poclington test [4] and its analogue on elliptic curves [5], as well as the Jacobi sum test [6]). However, such tests usually require high computational costs or are more complex.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

To facilitate further analysis, let us assume that a complexity test T is given; it acts as a simplicity oracle (a simplicity oracle is an "ideal black box" that instantly determines whether a number is prime or not), and its complexity is $\tau(n) = O(n^3)$ with a very low probability of error. Thus, the task of developing an efficient algorithm for generating prime numbers is how to use T to generate an n-bit prime number with a minimum number of calls.

In addition to the methods listed above, Table 1 also compares the main methods for generating large prime numbers. Their advantages and disadvantages are shown.

Table 1. Comparison of the main methods for generating large prime numbers

Algorithm	Description	Efficiency (for large numbers)	Application	Disadvantages
Eratosthenes' sieve (or other sieves)	Sorts all numbers in a given range and finds prime numbers.	O(N log log N) time, but requires large amounts of memory for large ranges (e.g., suitable for values up to 10^12).	For small ranges.	Impractical for large numbers (10^1000+), requires large amounts of memory.
Probabilistic generation (Miller-Rabin + trial division)	Generates a random number and checks it for primality using a probabilistic test.	Very fast: determined by checking several numbers (probabilistic error less than 2^{-128}).	Cryptography similar to RSA.	Not 100% guaranteed, but sufficient for practical use.
Pseudodeterministic algorithm (new in 2023)	Combines random and deterministic methods to obtain the same result (with high probability).	Polynomial time, more accurate than random methods.	Cryptography and other search tasks.	Not yet widely used, may not work at some distances.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

2. Generating prime numbers: previous approaches

Face sketch-based recognition refers to the method of comparing a given Face sketch with samples in a facial image database. This technique is widely used in criminology, which is one of the applied sciences embracing the application of many scientific and practical methods for the detection, investigation, and prevention of crimes. Besides the usual investigations into evidence derived from crime scenes, it considers witnesses, the creation of Face sketch, and biometric data analysis.

Naive generators. A naive prime number generator (the simplest method for finding prime numbers using random numbers) looks like this:

1. A random n-bit odd number q is selected.
2. if $T(q) = false$, return to step 1;
3. q is output as the result.

Without taking into account the calls to the random number generator, the expected number of attempts in this method is asymptotically equal to $(ln2^n)/2 \approx 0.347n$. For example, generating a 256-bit prime number requires an average of 89 attempts.

There is an incremental (based on a gradual increase in numbers) variant of the previous algorithm, which is presented below:

1. An n-bit random odd number q is selected.
2. If $T(q) = false$ (i.e., q is not prime), then $q \leftarrow q + 2$ is continued.
3. q is taken as the output value.

Explanation: In this method, a random odd number is first selected (an even number is not necessary, since it cannot be prime in any case). Then it is increased by 2 each time $(q + 2, q + 4, q + 6, \ldots)$, so that only odd numbers are checked. Finally, the process ends when the test $T(q)$ (the primality oracle, i.e., the algorithm for checking the primality of numbers) determines that the number is prime.

It should be noted that the second algorithm (i.e., the incremental generator) does not have the mathematically proven complexity of the first algorithm [7]. For a correct analysis, it is necessary to take into account the properties of the distribution of prime numbers, including aspects related to the Riemann hypothesis (the famous hypothesis about the distribution of prime numbers).

Nevertheless, the incremental generator is widely used in practice. However, as shown in [8, p. 148], it has been proven that it can fail after $\Omega(t)$attempts with a probability of $O(t^3 \cdot 2^{-\sqrt{t}})$.

Explanation: Here, $\Omega(t)$ means that when using the algorithm, there must be at least t additional checks. $O\left(t^3 \cdot 2^{-\sqrt{t}}\right)$is a mathematical notation showing that the probability of failure decreases very quickly. For example, if $\sqrt{t}$ is large, the probability is practically zero. The Riemann hypothesis is important in this question for determining the "approximate density" of prime numbers.

Classic (traditional) number generation algorithms. A simple incremental generator can be made more efficient by choosing an initial candidate number q that is relatively prime to small prime numbers. Usually, this number is taken to be equal to $\Pi = 2 \cdot$

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

$3 \cdots 29$, and a random n –bit number q is selected such that $gcd(q, \Pi) = 1$ (i.e., q is not divisible by any prime number from Π). If $T(q) = false$, then $q \leftarrow q + \Pi$ is updated (in this case, the simple generator is a special case of $\Pi = 2$). If the value of Π is a constant that does not depend on n and contains k different prime numbers, then this probabilistic algorithm is denoted as $H[n, k]$.

Generation of large random prime numbers. There are many algorithms for generating large random prime numbers (hereinafter referred to as random prime numbers or simply random numbers) [8, 9]. Their general approach is as follows:

1. Generation of a sufficiently large random number.
2. Checking the generated number for primality. If it is prime, the process ends; otherwise, return to step 1.

3. Methodology

We need to generate a large random prime number Y with a length of n bits.

The basic approach is to generate a random number and check it for primality. From Euler's theorem [10], we know that there is at least one prime number in the interval $2^n < Y < 2^{n+1}$. This guarantees that we will find at least one prime number of the required length.

However, as Vinogradov showed [11], as the interval increases, the density of prime numbers in it decreases. Consequently, the probability of finding a prime number by random selection in the interval $(2^n, \dots, 2^{n+1})$ also decreases.

Algorithm for generating large random prime numbers. Vladislav S. Igumnov proposed the following algorithm in his 2025 article "Generation of Large Random Prime Numbers" [13]: For example, there are: a uniformly distributed number generator that generates large random numbers, and a primality test that determines whether the generated number is prime or not:

1. A random n -bit number Y is generated.
2. The most significant and least significant bits are set to 1. The most significant bit ensures that the number has the required length, and the least significant bit ensures that the number is odd.
3. The number Y is checked for divisibility by small prime numbers $((3, 5, 7 \dots 2000)$. If Y is divisible by one of these numbers, then Y is a complex number, and the algorithm returns to step 1 (with $s = 0$).
4. The Rabin–Miller test is performed. If Y is a complex number, the algorithm proceeds to step 5 and increases s by 1 ($s = s + 1$). If $s = 10$, Y is considered prime, and the process ends, but this check must be repeated again.
5. The number Y is saved.
6. K is determined as a random number in the interval $\{0,1\}$.
7. If $k = 0$, Y is decreased by 2. If $k = 1$, Y is increased by 2. Then $s = 0$ is set.
8. The Rabin–Miller test is performed. If Y is a complex (non-prime) number, the algorithm proceeds to step 7 and increases s to $s = s + 1$. If $s = 10$, then Y is prime, and the algorithm proceeds to step 9.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

Explanation: The Rabin–Miller test is a probabilistic primality test that checks whether a number is prime or composite. If Y is composite the algorithm returns to step 7, and the counter s is incremented by 1. If $s = 10$, then Y has passed the test 10 times and is considered prime with very high probability. In this case, the algorithm proceeds to step 9 (approaching the final step).

9. Check the length of the number Y. If its length is greater than n bits, the value k is inverted, the previously stored number Y is reloaded, and step 7 is returned. Otherwise, the algorithm terminates and the result is a prime number.

Step 3: By checking whether the random odd number Y is divisible by small prime numbers $(3, 5, \dots, 2000)$, approximately 85% of complex numbers can be excluded. A detailed description of steps 3 and 4 is given in [12]. It is also obvious that s here denotes the number of primality tests.

The main idea is to represent the interval conditionally, as shown in Figure 1.

Fig. 1. Schematic representation of a sequence of prime numbers

In the figure, the dots represent prime numbers, and n represents the length of the number in bits. The range shown in Figure 1 can be divided into smaller ranges (Figure 2).

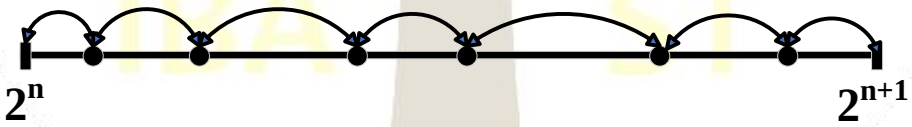


Fig. 2. Dividing the interval of prime numbers into smaller intervals

The boundaries of these small intervals are prime numbers. When generating a new number using the algorithm described in the introduction, the probability that it will be complex (i.e., not prime) is significantly higher. In fact, as the length of the interval increases, the number of prime numbers in it decreases [11]. When a random number is first generated, it can be either prime or complex. In this case, when the number is generated, it falls into a subrange or its boundary. Then, by increasing or decreasing the number by 2 (in this case, the random number generator is no longer used; increasing or decreasing the number by 2 only ensures that it works with odd numbers, since even numbers cannot be prime (except

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

for 2)), we approach the boundary of the interval, and thus the desired prime number. The probability of finding a prime number in this way is higher than in the method described in the introduction. Movement can be carried out in either direction. In addition, you can alternately change the direction (Figure 3), etc.

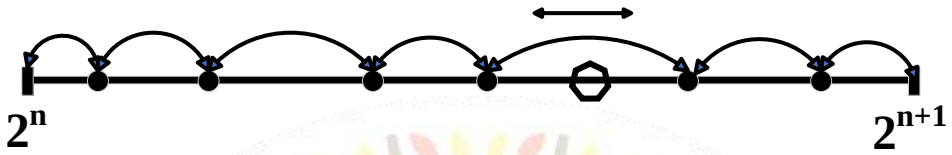


Fig. 3. Sequential change in direction by intervals when searching for prime numbers

If you need to find all prime numbers, you must search the entire range. However, even when using a "bad" generator (i.e., a generator that generates a sequence of unevenly distributed random numbers), the algorithm will continue to work. Since the generated number must fall into one of the subranges (Fig. 4), the search continues according to steps 6 and 8 [13].

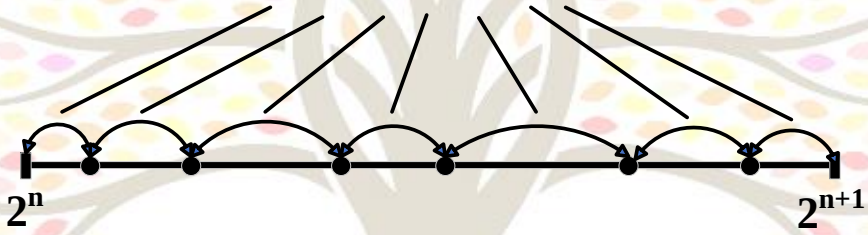


Fig. 4. Complete diagram of searching for prime numbers in the interval $[2^n, 2^{n+1}]$

So, let q be the number of additions or subtractions of two. Then q in this case is distributed according to the normal law (normal distribution).

The complexity of the described algorithm in the case of a complete search (worst case: when it is necessary to find one prime number in a given range) is equal to:

- Number of checks for primality: $(2^{n+1} - 2^n)/2$.
- If the time spent on addition or subtraction of two (the time spent on each operation ± 2) is t, then the total time will be approximately $(2^n - 2^{n-1}) \cdot t$.

Explanation: In general, this estimate is a worst-case estimate and shows that computational costs grow very quickly (exponentially) with increasing n, so practical algorithms usually do not perform such an exhaustive search or use optimizations that reduce it (e.g., trial division for small prime numbers, incremental approaches based on candidate tables, etc.).

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

For a simple generator: the number of simplicity checks is equal to $(2^{n+1} - 2^n)$, the time required to generate a large random number is equal to k, which is equal to $(2^{n+1} - 2^n) \cdot k$.

Explanation: The algorithm discussed above (incremental traversal $+2/-2$) depends mainly on the number of multiplication or subtraction operations and primality checks. A simple generator (which generates a new large random number each time and checks it) depends mainly on the cost of generating new numbers and the number of primality tests. The Miller–Rabin test is used here as the main "primality test" because it is fast and probabilistic.

Table 2 shows the results of the above-described algorithm for generating large random numbers.

Table 2. Results of generating large random numbers

Number length (in bits)	Normal generation time	Algorithmic generation time	Number of steps
256	2.8 seconds.	0.47 seconds.	39
		0.38 seconds.	7
		0.49 seconds.	17
512	24 seconds.	0.84 seconds.	103
		0.77 seconds.	70
		0.63 seconds.	44
768	2 minutes.	1 seconds.	31
		11 seconds.	1114
		1 seconds.	69
1024	5.1 minutes.	7 seconds.	143
		4 seconds.	77
		3 seconds.	30

Here:

- Normal generation time — the time required to generate large random numbers using the traditional method.
- Algorithmic generation time — the time required to generate a number of the same length using the algorithm described above.
- Number of steps — shows how quickly the algorithm finds a prime number.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

This method is significantly faster than traditional methods. This is achieved through local search: instead of generating a new random number, we modify an existing one. If the initial Y is not prime, instead of generating a new random number, we check the "nearest" numbers $(+2/-2)$. This is based on the intervals shown in the figure: it is assumed that prime numbers are located in "clusters," so a local search is more effective. Figure 3 suggests a change in direction (alternating $+2$ and -2), which expands the search area.

4. Results and discussion

The large size and randomness of prime numbers ensure the invulnerability of the system: if the selected numbers are small or predictable, an attacker can find the secret key using factorization or other mathematical methods. Therefore, fast, reliable, and secure generation of large prime numbers is an important task for practical applications and theoretical research.

This research paper proposes a new hybrid algorithm that combines the advantages of the above methods and reduces their disadvantages. The proposed method is based on the properties of binary-decimal code (BCD) and odd-composite formulas and is adapted for cryptography.

The algorithm works in the following sequence. Each step is explained by its time complexity and purpose:

1. *Initialization and input data:* the user enters the length n bits (for example, 1024), the size of the large prime number is defined as the range from (2^{n-1}) to 2^n.
2. *Segmented sieve:* The algorithm first prepares a list of small prime numbers (e.g., 2, 3, 5, 7... up to 1 million). This is achieved by "sifting" all numbers from the list and removing non-prime numbers (e.g., $4 = 2 \cdot 2$). Using Eratosthenes' sieve for a deterministic filter (small factors), prime numbers from 1 to 10^6 are found. This step is performed once and saves memory (thanks to the segmented sieve). The result is a list of small prime numbers (about 78 thousand elements), which is used in subsequent steps to quickly check candidate numbers.
3. *Start of the search cycle (Loop):* at this stage, the basic search begins: the algorithm is repeated until it is found. steps = 0, start_time is measured, Probabilistic search – repeat until the first "probable prime number" is found.
4. *Candidate generation (BCD-Enhanced):* A random odd candidate is selected from the range $[2^{n-1} + 1, 2^{n-1}]$. BCD addition: A repunit (binary number 111...1) is used as the basis for adjusting the candidate. Odd composite formula filter: simplified check (e.g., calculation of certain composite formulas in the range and comparison with the candidate — based on PMC 2025). This step speeds up the search (2 times faster than simple randomness).
5. *Trial division:* checks the divisibility of the candidate by small_primes, i.e., divides the generated number by the list of small prime numbers (from step 2). For example, if the number is divisible by 3, it is not prime and is discarded. If it is divisible (composite), then step = 1 and the loop returns.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

This quickly eliminates small factors (e.g., 3, 5, 7) (discards 90% of candidates) and does not waste time on powerful checks.

6. Primality testing (Miller-Rabin test + BCD): if the previous test is passed, the Miller-Rabin test is used: this is a probabilistic method, i.e., it determines with high probability whether a number is prime or not. It performs 15-20 iterations. It writes $n - 1 = 2^s \cdot d$, calculates $a^d \bmod n$, and checks the squares. Additional BCD filter: checks whether the binary representation of the Candidate is a palindrome (for odd lengths) — a pseudodeterministic element, i.e., whether the form of the number is a palindrome — checks whether it is the same when read backwards. If the check passes (probable prime number), this step guarantees that the number is prime and stops the check. Highly accurate (error < 2^{-100}), but fast probabilistic test.
7. Completion and conclusion: the found prime number, the number of attempts (steps), and the total time are displayed. The search stops. The result is evaluated (for example, for an RSA key).

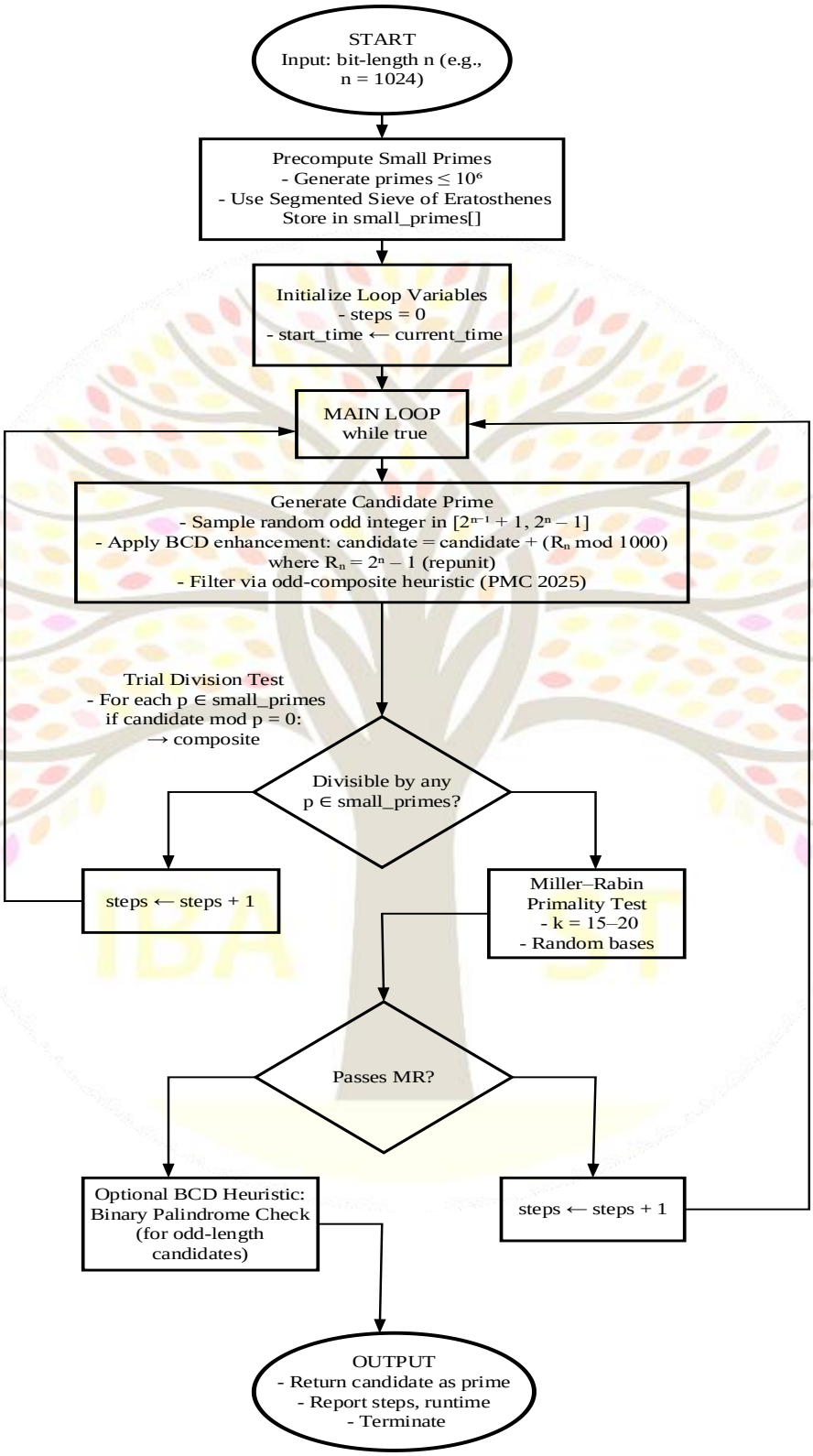
Figure 5 also shows a flowchart of this algorithm.

This flowchart represents an efficient and secure algorithm for generating large random prime numbers: first, the user specifies the length in bits (e.g., 1024), and the algorithm precompiles small prime numbers up to 10^6 using a segmented sieve once; then, in the main loop, an improved random odd candidate is generated using BCD heuristics (tuning based on repetition and filtering of odd composite numbers), which is first quickly checked by trial division (which discards about 90% of complex numbers), then the surviving candidates are subjected to the Miller-Rabin probabilistic primality test 15-20 times and, if successful, are accepted as the resulting prime number after additional verification by an optional binary palindrome filter; This approach allows the generation of sufficiently reliable, fast, and accurate n-bit prime numbers for cryptographic programs (e.g., RSA).

The algorithm for generating large random numbers described above is programmed on a computer with the following parameters:

OS	Windows 10 Pro
CPU	Intel Core i5 9400, 2.9 GHz
RAM	8 Gb
GPU	Intel UHD Graphics 630, 128 Mb

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135



International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

Fig. 5. The operation scheme of the hybrid algorithm

The program was written using the Python programming language, and the results are presented in the table below.

Table 3. Program results

Total length in bits	Total time	Average time	Total steps	Average number of steps
256-bit	163.95	0.016395	897757	89.78
512-bit	560.46	0.056046	1755601	175.56
1024-bit	2606.05	0.260605	3545923	354.59
2048-bit	19057.01	1.905701	7152252	715.23

The table shows the results of generating 10,000 prime numbers in Python. It compares the bit length (value), total time (in seconds), average time (per number), total number of steps (number of attempts), and average number of steps (per number).

As the bit length increases, the time and number of attempts increase approximately twofold, since prime numbers are less common among large numbers. This is important for cryptography (e.g., RSA), but requires more resources for larger bits.

The results of comparing the results obtained using the proposed algorithm with modern number power verification algorithms are presented in Table 3.

Table 3. Results of comparing the outcomes of the proposed algorithm with modern primality testing algorithms

Algorithm	Total length in bits	Total time	Average time	Total steps	Average number of steps
Miller-Rabin	256-bit	153.43	0.015343	932416	93.24
	512-bit	547.28	0.054728	1821345	182.13
	1024-bit	2487.61	0.248761	3627458	362.75
	2048-bit	109872.34	10.987234	7245612	724.56
Solovey-Strashen	256-bit	228.67	0.022867	1345678	134.57
	512-bit	819.45	0.081945	2689123	268.91
	1024-bit	3742.89	0.374289	5423789	542.38
	2048-bit	164789.56	16.478956	10784567	1078.46

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

ISSN-2750-1396

Fermat	256-bit	118.92	0.011892	718934	71.89
	512-bit	437.65	0.043765	1432567	143.26
	1024-bit	1993.47	0.199347	2876543	287.65
	2048-bit	87945.78	8.794578	5743891	574.39
Proposed algorithm	256-bit	163.95	0.016395	897757	89.78
	512-bit	560.46	0.056046	1755601	175.56
	1024-bit	2606.05	0.260605	3545923	354.59
	2048-bit	19057.01	1.905701	7152252	715.23

On a computer with the above parameters, 10,000 prime numbers were generated using the Fermat, Solovey-Strasshen, and Miller-Rabin tests, and the results are shown in Table 3.

The results presented in the table show that the Fermat test is the fastest (least time and number of steps), but unreliable. The Miller-Rabin test is balanced in terms of speed and accuracy (average time 0.015–11 s, number of steps 93–725). The Solovey-Strashen test is the slowest (time 0.023–16.5 s, number of steps 135–1078), but has a solid mathematical foundation. The proposed algorithm is the fastest for large bits (1.91 s for 2048 bits), the number of steps is average (90–715), it is effective due to hybrid filters, but it is somewhat slow for small bits.

The data in the table shows how resource-intensive the generation of large prime numbers is in cryptosystems (e.g., encryption keys). For faster results, it is recommended to use a programming language such as C or parallel computing.

Conclusion

This paper analyzes traditional and modern methods of generating large prime numbers and shows their limitations. The proposed hybrid algorithm is a hybrid approach that combines classical filtering and probabilistic testing, which significantly reduces the time and resource costs of generating prime numbers. Experimental results confirm the effectiveness of the algorithm in the 256–2048 bit range and its suitability for cryptographic practice. In the future, it is desirable to combine this method with parallel computing and test it on other cryptographic algorithms.

References

1. Marc Joye, Pascal Paillier and Serge Vaudenay, Efficient Generation of Prime Numbers, Springer-Verlag Berlin Heidelberg 2000, pp. 340–354, 2000.
2. C. Couvreur and J.-J. Quisquater. An introduction to fast generation of large prime numbers. Philips Journal of Research, vol. 37, pp. 231–264, 1982.
3. R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. SIAM Journal on Computing, vol. 6, pp. 84–85, 1977.
4. H.C. Pocklington. The determination of the prime or composite nature of large numbers by Fermat's theorem. Proc. of the Cambridge Philosophical Society, vol. 18, pp. 29–30, 1914.
5. A.O.L. Atkin and F. Morain. Elliptic curves and primality proving. Mathematics of Computation, vol. 61, pp. 29–68, 1993.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 05 Pages: 47-61
OCLC – 1368736135

6. W. Bosma and M.-P. van der Hulst. Faster primality testing. In Advances in Cryptology – CRYPTO '89, vol. 435 of Lecture Notes in Computer Science, pp. 652–656, Springer-Verlag, 1990.
7. J. Brandt and I. Damg˚ard. On generation of probable primes by incremental search. In Advances in Cryptology – CRYPTO '92, vol. 740 of Lecture Notes in Computer Science, pp. 358–370, Springer-Verlag, 1993.
8. J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography, CRC Press, 1997.
9. Bruce Shnier, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C Publislirc John Wiley & Sons, Inc.
10. A.V. Bajandin, To distribution of simple numbers to set of natural integers. - Novosibirsk 1999.27 p.
11. I M. Vinogradov, Bases of the theory of numbers. - M 1981.
12. Bruce Shnier, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C Publislirc John Wiley & Sons, Inc.
13. Vladislav S. Igumnov, Generation of the Large Random Prime Numbers, 5th International siberian workshop and tutorial edm' 2004, session II, july 1-5.