Randomized Primality Testing: Reinventing, Experiments, and Applications

Arshit Narang - 220209, Harsh Agrawal - 220425, Prem Kansagra - 220816 [Group 5, CS648]

11 April 2025

Abstract

Randomized algorithms play a pivotal role in computational number theory, often serving as the first practical—or even foundational—approach to problems that may later yield deterministic solutions. Primality testing exemplifies this paradigm, with randomized methods consistently offering superior real-world performance. This project investigates the Miller–Rabin primality test, a widely used probabilistic algorithm with time complexity $O(k \log^2 n)$, where k controls the trade-off between runtime and accuracy. In contrast, deterministic algorithms such as AKS, despite their theoretical guarantees and polynomial complexity of $O(\log^6 n)$, remain computationally prohibitive for large inputs. Our implementation of Miller–Rabin from first principles explores its probabilistic structure, modular arithmetic foundations, and efficiency in practice. We observe that with a modest number of iterations, the error probability decreases exponentially—approaching a level so low that the algorithm behaves as if deterministic. These findings reaffirm the practical advantages of randomized algorithms in primality testing and their critical role in scalable number-theoretic computations, including those found in cryptographic systems.

1 Project Tasks

- **First Task** - The initial idea focused on exploring the applicability of Fermat's Little Theorem in the domain of primality testing.

  `Fermat's Little Theorem:` For any integer **a** and a prime number **p**, the following property always holds:

  $$a^p \equiv a \pmod{p}$$

  Initial Hypothesis: For any composite number **b**, we hypothesized that there exists an upper bound on the proportion of integers in the range $[2, b-1]$ for which the following congruence holds:

  $$a^b \equiv a \pmod{b}$$

  where $a \in [2, b-1]$.

  If the hypothesis holds true, it would enable the design of a simple randomized primality testing algorithm. The proposed approach involves repeatedly selecting a random integer $u \in [2, b-1]$, and checking whether the following condition holds:

  $$u^{b-1} \equiv 1 \pmod{b}$$

  If this condition fails for any sampled value of u, the number b can be definitively declared composite. If the condition holds across all random samples, b is considered probably prime. Repeating this process multiple times with independent random values of u increases confidence in the result while reducing the likelihood of falsely classifying a composite number as prime.

  Testing: The hypothesis was empirically tested for all integers up to 40,000. However, the experimental results did not support the existence of a consistent upper bound on the proportion of integers satisfying the condition for composite values of b.

- **Second Task** - For any composite number n, the set of integers in the range $[2, n-1]$ can be partitioned into two subsets: one consisting of integers that are co-prime to n, and the other consisting of integers that are not co-prime to n.

  – **Sub-task 1:** The objective of this sub-task was to formally establish that for all integers $a < n$ such that $\gcd(a, n) > 1$, the following congruence does *not* hold:

  $$a^{n-1} \not\equiv 1 \pmod{n}$$

  Proof: We make use of the following fundamental number-theoretic property:

  · `Property 1:` For all integers $x, y \in \mathbb{Z}$ such that $ax + by \geq 0$, the inequality below holds:

  $$ax + by \geq \gcd(a, b)$$

  Now, assume $\gcd(a, n) > 1$. It follows that $\gcd(a^{n-1}, n) > 1$ as well.

  We proceed by contradiction. Suppose that:

  $$a^{n-1} \equiv 1 \pmod{n}$$

  Then there exists an integer $u \in \mathbb{Z}$ such that:

  $$a^{n-1} = un + 1$$

  Rewriting, we get:

  $$1 = a^{n-1} - un$$

  According to Property 1, this implies:

  $$1 \cdot a^{n-1} - u \cdot n \geq \gcd(a^{n-1}, n) > 1$$

  This yields a contradiction, since the left-hand side equals 1, but the right-hand side is strictly greater than 1. The contradiction arises from the false assumption that:

  $$a^{n-1} \equiv 1 \pmod{n}$$

  Therefore, the initial supposition is invalid. As a result, we conclude that for all integers $a < n$ such that $\gcd(a, n) > 1$, the following property always holds:

  $$a^{n-1} \not\equiv 1 \pmod{n}$$

  – **Sub-task 2:** Let S denote the set of integers in the range $[2, n-1]$ that are co-prime to n. The objective of this sub-task is to prove the following property: If there exists at least one element $a \in S$ such that

  $$a^{n-1} \not\equiv 1 \pmod{n},$$

  then at least half of the elements in S also satisfy this condition.

Proof: This statement will be proven through the application of the following two lemmas:

- **Lemma 1:** Let a and p be coprime integers. Then for any integers i, j such that $1 \leq i, j < p$ and $i \neq j$, the following inequality holds:

  $$(a \cdot i) \bmod p \neq (a \cdot j) \bmod p$$

  Proof: Assume, for contradiction, that:

  $$(a \cdot i) \bmod p = (a \cdot j) \bmod p.$$

  Then there exist integers u, v such that:

  $$a \cdot i = u \cdot p + \alpha,$$
  $$a \cdot j = v \cdot p + \beta,$$

  where $0 \leq \alpha, \beta < p$ and $\alpha = \beta$. Without loss of generality, suppose $i > j$. Subtracting the two equations yields:

  $$a \cdot (i - j) = (u - v) \cdot p.$$

  Since $\gcd(a, p) = 1$, by Bézout's identity, there exist integers x, y such that:

  $$ax + py = 1.$$

  Rewriting a using this identity:

  $$a = \frac{1 - py}{x}.$$

  Substituting into the previous equation gives:

  $$\left(\frac{1 - py}{x}\right) \cdot (i - j) = (u - v) \cdot p.$$

  Rearranging and taking modulo p on both sides results in:

  $$(i - j) \bmod p = 0,$$

  which implies:

  $$i = j \pmod p.$$

  Given that $1 \leq i, j < p$, this contradicts the assumption that $i \neq j$. Therefore, the original assumption must be false, proving the lemma:

  $$(a \cdot i) \bmod p \neq (a \cdot j) \bmod p.$$

  Thus, the lemma is proved.

- **Lemma 2:** Let $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$. Then it follows that:

  $$\gcd((ab) \bmod p, p) = 1.$$

  Proof: Since $\gcd(a, p) = 1$, there exist integers x and y such that:

  $$ax + py = 1.$$

  Similarly, since $\gcd(b, p) = 1$, there exist integers s and t such that:

  $$bs + pt = 1.$$

  Multiplying both equations yields:

  $$(ax + py)(bs + pt) = 1.$$

  Expanding the product:

  $$ab \cdot xs + p \cdot (pyt + axt + bsy) = 1.$$

  Define $V = xs$ and $W = pyt + axt + bsy$, so we obtain:

  $$ab \cdot V + p \cdot W = 1.$$

  Since 1 is a linear combination of ab and p, it follows that:

  $$\gcd(ab, p) = 1.$$

  To show $\gcd((ab \bmod p), p) = 1$, consider the congruence:

  $$ab \cdot V + p \cdot W \equiv 1 \pmod p.$$

  Since $p \cdot W \equiv 0 \pmod p$, this implies:

  $$ab \cdot V \equiv 1 \pmod p.$$

  Let $\gamma = V \bmod p$. Then:

  $$(ab \bmod p) \cdot \gamma \equiv 1 \pmod p.$$

  This implies that $ab \bmod p$ has a multiplicative inverse modulo p, hence:

  $$\gcd((ab \bmod p), p) = 1.$$

  Hence Proved.

– **Final Solution: Two Distinct Possibilities**
For every $x_i \in S$, exactly one of the following conditions must hold:

$$x_i^{a-1} \equiv 1 \pmod a \quad \text{or} \quad x_i^{a-1} \not\equiv 1 \pmod a.$$

By assumption, there exists at least one element $x_j \in S$ such that:

$$x_j^{a-1} \not\equiv 1 \pmod a.$$

Contradictory Assumption: Suppose, for the sake of contradiction, that at least half of the elements in S satisfy:

$$x_i^{a-1} \equiv 1 \pmod a, \quad \forall x_i \in S',$$

where

$$S' = \{x_1, x_2, \ldots, x_{\lceil N/2 \rceil}\} \subseteq S.$$

Now, construct a new set S'' by multiplying each element in S' by x_j, taken modulo a:

$$S'' = \{(x_j \cdot x_1) \bmod a, (x_j \cdot x_2) \bmod a, \ldots, (x_j \cdot x_{\lceil N/2 \rceil}) \bmod a\}.$$

- **By Lemma 1**, since $\gcd(x_j, a) = 1$, the mapping $x_i \mapsto (x_j \cdot x_i) \bmod a$ is a bijection on S', so all elements in S'' are distinct.
- **By Lemma 2**, each product $(x_j \cdot x_i) \bmod a$ remains coprime to a, i.e.,

  $$\gcd((x_j \cdot x_i) \bmod a, a) = 1 \quad \forall x_i \in S'.$$

  Hence, all elements of $S'' \subseteq S$.

Now, consider the expression $(x_j \cdot x_i)^{a-1} \mod a$ for any $x_i \in S'$. By properties of modular exponentiation:

$$(x_j \cdot x_i)^{a-1} \equiv x_j^{a-1} \cdot x_i^{a-1} \pmod a.$$

- Since $x_i \in S'$, we have $x_i^{a-1} \equiv 1 \pmod a$ by assumption.
- Therefore,

  $$(x_j \cdot x_i)^{a-1} \equiv x_j^{a-1} \pmod a.$$

- Given that $x_j^{a-1} \not\equiv 1 \pmod a$, it follows that:

  $$(x_j \cdot x_i)^{a-1} \not\equiv 1 \pmod a, \quad \forall x_i \in S'.$$

This implies:

- There are at least $\lceil N/2 \rceil$ distinct elements in $S'' \subseteq S$ such that:

  $$x^{a-1} \not\equiv 1 \pmod a.$$

- However, this contradicts the initial assumption that at least $\lceil N/2 \rceil$ elements in S satisfy:

$$x^{a-1} \equiv 1 \pmod{a}.$$

- Since each element of S'' corresponds to a distinct element in S', and all violate the condition, we arrive at a contradiction.

Conclusion: The initial assumption must therefore be false. Hence, it must be true that **at least half** of the elements in S satisfy:

$$x^{a-1} \not\equiv 1 \pmod{a}.$$

Thus, the claim is proved.

- **Third Task**:- In the previous task, we showed that for certain composite numbers n, at least half of the integers less than n do not pass Fermat's primality test. However, there exists a special class of composite numbers—called *Carmichael numbers*—that satisfy Fermat's condition for **all** bases a such that $\gcd(a, n) = 1$. This demonstrates a key weakness of `Fermat's test`: it is not foolproof.

  To address this shortcoming, we explore a more rigorous approach. Let n be an **odd integer** whose primality needs to be verified. We express $n - 1$ as:

$$n - 1 = 2^t \cdot s,$$

  where s is an odd number and $t \geq 1$. Let a be any integer such that $1 < a < n$.

  We now define j to be the **largest integer** such that $0 \leq j \leq t$ and

$$a^{2^j \cdot s} \not\equiv 1 \pmod{n}.$$

  Then, for any prime n, the following must hold:

$$a^{2^j \cdot s} \equiv -1 \pmod{n},$$

  which is equivalent to:

$$a^{2^j \cdot s} \equiv n - 1 \pmod{n}.$$

  This criterion is the foundation of the *Miller-Rabin primality test*, which offers a stronger and more dependable alternative to Fermat's test by filtering out many pseudoprimes that Fermat's test would accept. The third task is to prove the property stated above.

  Proof -

  – **Mini-Lemma**:-
  If a prime p divides a product ab, then p must divide at least one of a or b.

  Proof of Mini-Lemma:
  Assume we have the following:

$$ax + py = 1 \quad \text{and} \quad bz + pw = 1.$$

  Multiplying the two:

$$(ax + py)(bz + pw) = 1.$$

  Expanding:

$$abxz + axpw + pybz + p^2yw = 1.$$

  Reducing modulo p, we get:

$$abxz \equiv 1 \pmod{p}.$$

  If p divides ab, then $ab \equiv 0 \pmod{p}$, contradicting $abxz \equiv 1 \pmod{p}$.
  Therefore, p must divide either a or b.

  – **Lemma** -
  If n is an odd prime number and $x^2 \equiv 1 \pmod{n}$, then $x \equiv 1$ or $x \equiv n - 1 \pmod{n}$.

  Proof :
  Suppose $x \equiv k \pmod{n}$, i.e.,

$$x = \mu n + k, \quad \text{for some } \mu \in \mathbb{Z}.$$

  Squaring both sides:

$$x^2 = \mu^2 n^2 + 2\mu nk + k^2.$$

  Modulo n, we get:

$$x^2 \equiv k^2 \pmod{n}.$$

  Since $x^2 \equiv 1 \pmod{n}$, we obtain:

$$k^2 \equiv 1 \pmod{n}.$$

  Rewriting:

$$k^2 - 1 \equiv 0 \pmod{n}.$$

  Factoring:

$$(k-1)(k+1) \equiv 0 \pmod{n}.$$

  This implies:

$$(k-1)(k+1) = \gamma n, \quad \text{for some } \gamma \in \mathbb{Z}.$$

  By the **Mini-Lemma**, since n is prime, it must divide either $k - 1$ or $k + 1$.
  Case 1: If $n \mid (k - 1)$, then

$$k - 1 = \beta n \quad \Rightarrow \quad k \equiv 1 \pmod{n}.$$

  Case 2: If $n \mid (k + 1)$, then

$$k + 1 = \delta n \quad \Rightarrow \quad k \equiv -1 \equiv n - 1 \pmod{n}.$$

  As $0 \leq k < n$, the only possible values are:

$$k \equiv 1 \pmod{n} \quad \text{or} \quad k \equiv n - 1 \pmod{n}.$$

  Using the lemma above, we can construct a more dependable primality test. Unlike Fermat's method, which is vulnerable to specific composites like Carmichael numbers, the Miller-Rabin test leverages deeper arithmetic properties.

  Specifically, results from group theory and number theory show that for any odd composite number n, no more than one-fourth of the integers a with $1 < a < n$ and $\gcd(a, n) = 1$ will wrongly indicate that n is prime. Such misleading values of a are known as *liars*, because they falsely satisfy conditions typical of primes.

  This drastic drop in the proportion of false witnesses is what gives the **Miller–Rabin primality test** its strength. It is a probabilistic test whose error probability decreases exponentially with the number of rounds, making it a highly reliable and widely-used method in practice.

- **Final Algorithm** -

Algorithm 1 Miller–Rabin Primality Test

Input: $n > 2$, an odd integer to be tested for primality
Input: k, the number of rounds of testing to perform
Output: "composite" if n is found to be composite, "probably prime" otherwise

```
1: Let $s > 0$ and $d > 0$ such that $n - 1 = 2^s \cdot d$    ▷ Factor powers of 2 from $n - 1$
2: for $i = 1$ to $k$ do
3:     Choose random $a \in [2, n - 2]$
4:     $x \leftarrow a^d \bmod n$
5:     for $r = 1$ to $s$ do
6:         $y \leftarrow x^2 \bmod n$
7:         if $y = 1$ and $x \neq 1$ and $x \neq n - 1$ then
8:             return "composite"    ▷ Nontrivial square root of 1 mod $n$
9:         end if
10:        $x \leftarrow y$
11:    end for
12:    if $x \neq 1$ then
13:        return "composite"
14:    end if
15: end for
16: return "probably prime"
```

Time Complexity Analysis of the Miller–Rabin Algorithm

Let n be the number to test for primality, and k be the number of iterations (or rounds) used to reduce the error probability.

Per-Round Complexity

Each round of the Miller–Rabin test includes:

– Modular exponentiation $a^d \bmod n$, which requires $O(\log n)$ multiplications using exponentiation by squaring.
– Each multiplication of large integers (up to n) takes:
  · $O((\log n)^2)$ time with standard multiplication.
  · $O(\log n \log \log n)$ time using more advanced multiplication methods (e.g., Karatsuba or FFT).

Thus, the time complexity for a single round is:

$$O(\log^3 n)$$

using basic multiplication.

Total Time Complexity

Since the algorithm performs k independent rounds to ensure correctness, the total time complexity is:

$$O(k \cdot \log^3 n)$$

Error Probability

The probability of falsely identifying a composite number as prime decreases exponentially with k. For a composite number, the error probability is at most:

$$\left(\frac{1}{4}\right)^k$$

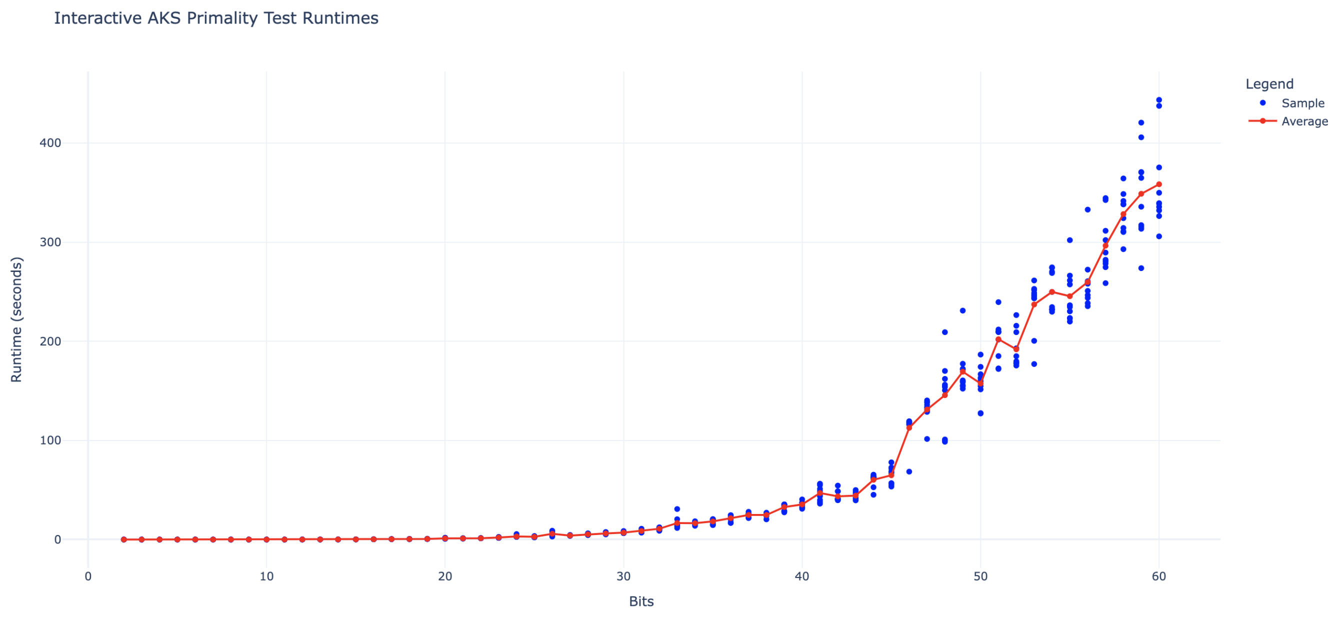


Figure 1. Runtime performance of the AKS algorithm for prime numbers of 2-60 bits. Each blue point represents a sample, and the red line indicates average runtime.

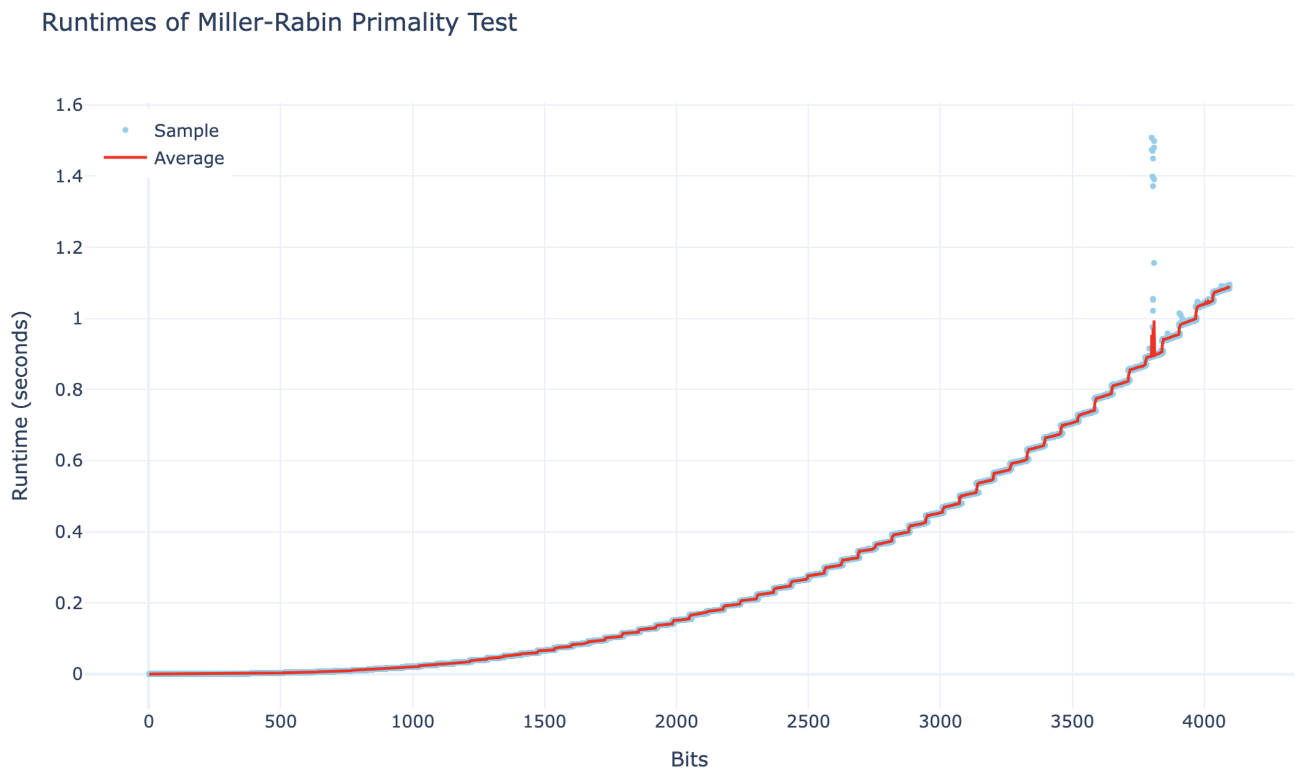


Figure 2. Runtime performance of the Miller-Rabin algorithm for prime numbers of 2-4096 bits. The light blue points represent individual samples, and the red line indicates average runtime.

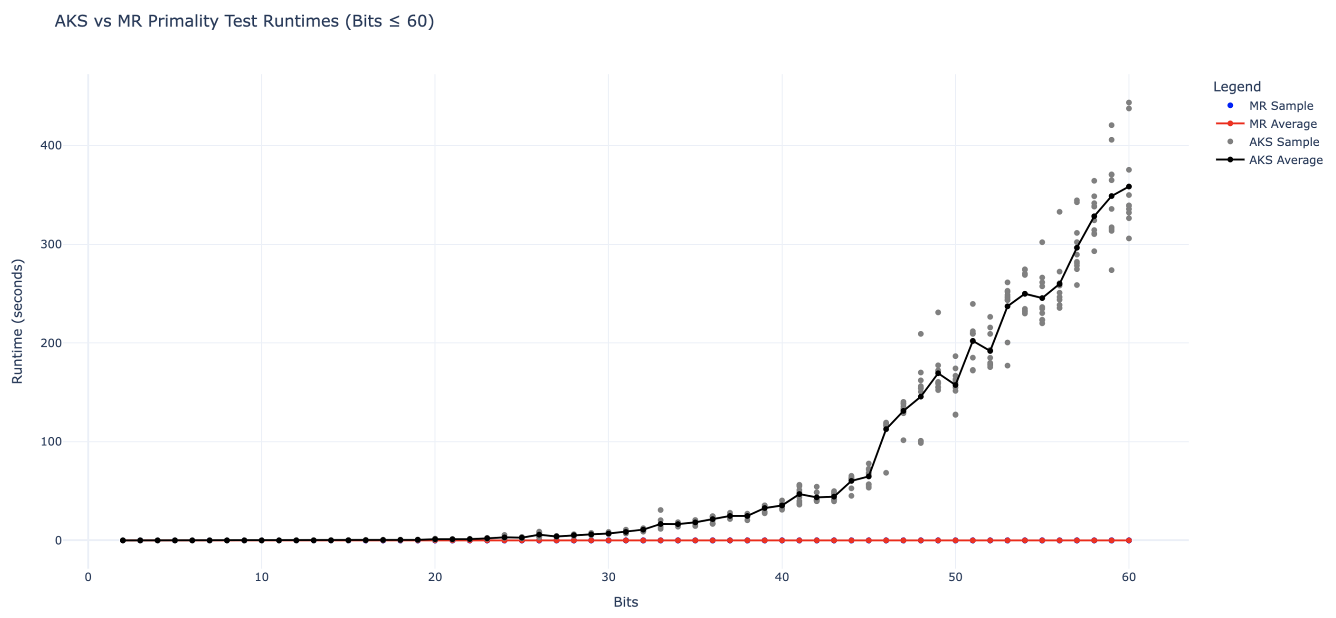


Figure 3. Comparison of runtime performance between AKS and Miller-Rabin algorithms for prime numbers of 2-60 bits. Gray points represent AKS samples, blue points represent MR samples, with their respective average runtimes shown by black and red lines.

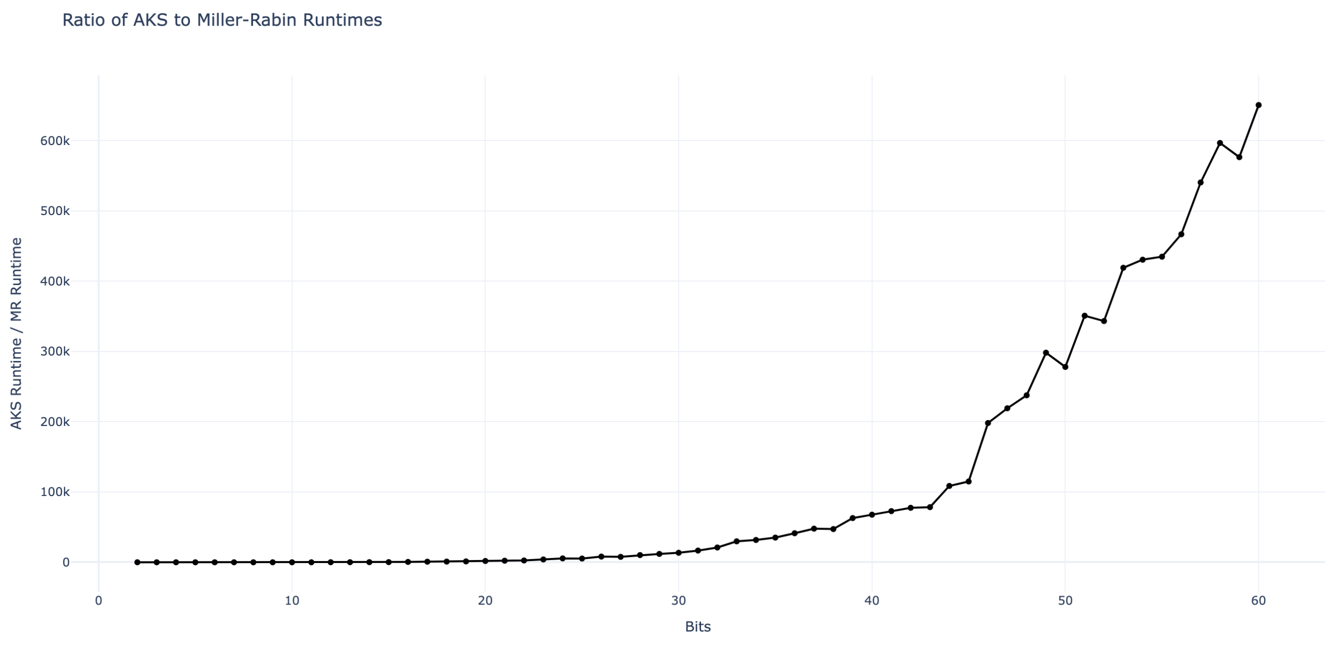


Figure 4. Ratio of AKS to Miller-Rabin average runtimes for prime numbers of 2-60 bits.

Experimental Analysis

In this section, we present a comprehensive analysis of the runtime performance of two primality testing algorithms: the deterministic AKS (Agrawal-Kayal-Saxena) algorithm and the probabilistic Miller-Rabin (MR) algorithm. Our experiments measure execution times for various bit-lengths of randomly generated prime numbers, highlighting the practical efficiency differences between these theoretically significant algorithms.

1.1 Experimental Setup

For our experiments, we generated uniformly random prime numbers of various bit lengths. For each bit length, multiple samples were tested to ensure statistical reliability. The AKS algorithm was tested with prime numbers of 2 to 60 bits, while the Miller-Rabin algorithm, due to its superior efficiency, was tested with prime numbers ranging from 2 to 4096 bits.

Experiments were conducted on the CSE department's `image`, `image1`, and `turing` servers with 20, 40, and 16 processors, respectively. We used MPI-based parallel programming to distribute workloads across multiple cores, significantly speeding up data generation. For instance, checking all numbers up to 10^8 with Miller-Rabin (for a fixed k) took 6.5 hours on a single core, but we parallelized this across 20–40 cores. Similarly, runtime experiments for both AKS and Miller-Rabin were parallelized to manage the long runtimes of AKS at higher bit lengths.

1.2 AKS Algorithm Runtime Analysis

Figure 1 illustrates the runtime behavior of the AKS primality test. For each bit length from 2 to 60, we tested 10 randomly generated prime numbers. The red line connects the average runtimes for each bit length. As visible from the graph, the AKS algorithm shows relatively efficient performance for small bit lengths (below 30 bits), with runtimes under 10 seconds. However, as bit length increases beyond 40 bits, we observe a dramatic increase in runtime, reaching approximately 400 seconds for 60-bit numbers. This exponential growth in computational time demonstrates why the AKS algorithm, despite its theoretical significance as the first deterministic polynomial-time primality test, faces practical limitations for large inputs.

1.3 Miller-Rabin Algorithm Runtime Analysis

Figure 2 presents the runtime performance of the Miller-Rabin primality test. Unlike the AKS algorithm, we were able to test the MR algorithm on much larger inputs, ranging from 2 bits to **4096** bits. For each bit length, 20 randomly generated prime numbers were tested. The graph clearly demonstrates the algorithm's poly-logarithmic time complexity, with runtimes staying remarkably low even for very large inputs. Even for 4096-bit prime numbers, the average runtime remains around 1 second, emphasizing the practical efficiency of this probabilistic approach.

1.4 Comparative Analysis

1.4.1 Direct Runtime Comparison

Figure 3 provides a direct comparison between the AKS and Miller-Rabin algorithms for bit lengths from 2 to 60. The contrast is striking: while the AKS algorithm (gray points, black line) shows an exponential increase in runtime as bit length increases, the Miller-Rabin algorithm (blue points, red line) maintains consistently low runtimes that are barely visible on the same scale. This visualization effectively demonstrates why the probabilistic Miller-Rabin algorithm is the preferred choice in practical applications despite lacking the deterministic guarantees of AKS.

1.4.2 Runtime Ratio Analysis

To quantify the efficiency gap between the two algorithms, Figure 4 plots the ratio of AKS runtime to Miller-Rabin runtime across different bit lengths. The ratio grows dramatically as bit length increases, reaching approximately 700,000 for 60-bit numbers. This means that for 60-bit prime numbers, the AKS algorithm takes around 700,000 times longer to execute than the Miller-Rabin algorithm, highlighting the enormous practical advantage of the probabilistic approach.

1.5 Extended Range Comparison

Figure 5 illustrates the maximum practical range for both algorithms. The AKS algorithm (shown in black) reaches runtimes of approximately 400 seconds at just 60 bits, while the Miller-Rabin algorithm (shown in red) extends to 4096-bit numbers with runtimes of only 1-2 seconds. This dramatic difference emphasizes that while the AKS algorithm provided a significant theoretical breakthrough, the Miller-Rabin algorithm remains the practical choice for primality testing of large numbers.

1.6 Time Complexity Visualization

Figure 6 represents a snapshot from an animation that incrementally shows the development of the Miller-Rabin runtime curve. By visualizing the algorithm's performance in 32-bit increments from 2 to 4096 bits, this animation clearly demonstrates the poly-logarithmic nature of the Miller-Rabin algorithm's time complexity, specifically $O(k \log^3 n)$, where k is the number of iterations and n is the input number.

1.7 Error Probability Analysis

To empirically evaluate the error probability of the Miller–Rabin primality test, we conducted experiments varying the number of iterations k from 1 to 20. Within the test range of integers from 2 to 10^8, the algorithm incorrectly classified 524 composite numbers as prime when $k = 1$, 25 such errors with $k = 2$, and only 2 errors with $k = 3$. Notably, for all $k \geq 4$, the test produced zero false positives as shown in Figure 7. These results underscore the rapid exponential decay of the error probability with increasing iterations and illustrate the practical reliability of the algorithm, even with a modest number of repetitions.

1.8 Summary of Findings

Our experimental analysis confirms the theoretical time complexity gap between the AKS and Miller–Rabin primality testing algorithms:

- The AKS algorithm, though deterministic, exhibits prohibitive runtime growth even at moderate bit lengths (40–60 bits).
- The Miller–Rabin algorithm is highly efficient, with runtimes remaining practical even for inputs up to 4096 bits.
- The performance gap between the algorithms grows exponentially with input size, reaching factors of hundreds of thousands.
- For practical applications involving large-number primality testing, the probabilistic Miller–Rabin algorithm is clearly preferable despite its non-deterministic nature.
- Even a few repetitions (e.g., $k = 4$) of the Miller–Rabin test yield consistently correct results. Increasing k to 40 reduces the error probability to a negligible level, as it decreases exponentially with k.

These findings underscore the trade-off between the theoretical rigor of deterministic algorithms and the practical efficiency of probabilistic methods in number-theoretic applications.

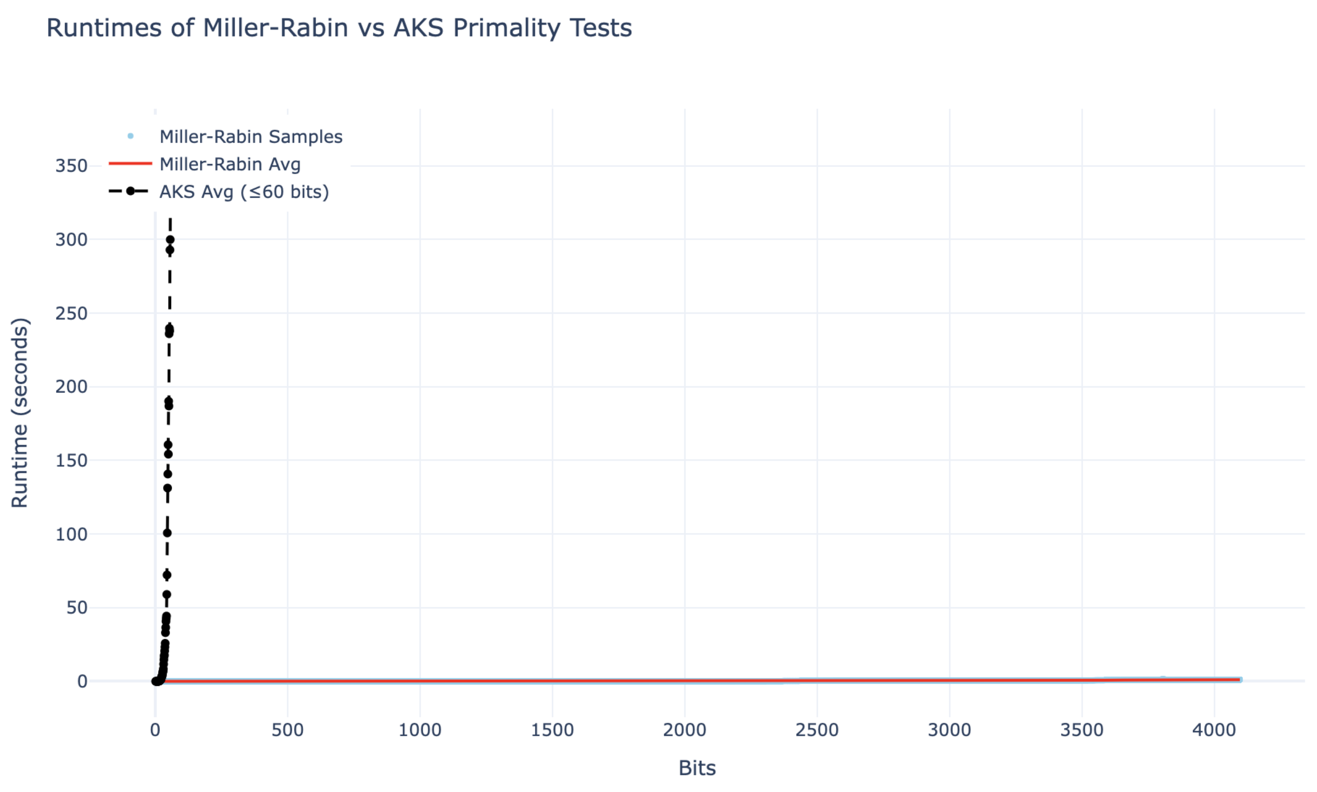


Figure 5. Extended comparison showing AKS (up to 60 bits) and Miller-Rabin (up to 4096 bits) runtimes on the same graph.

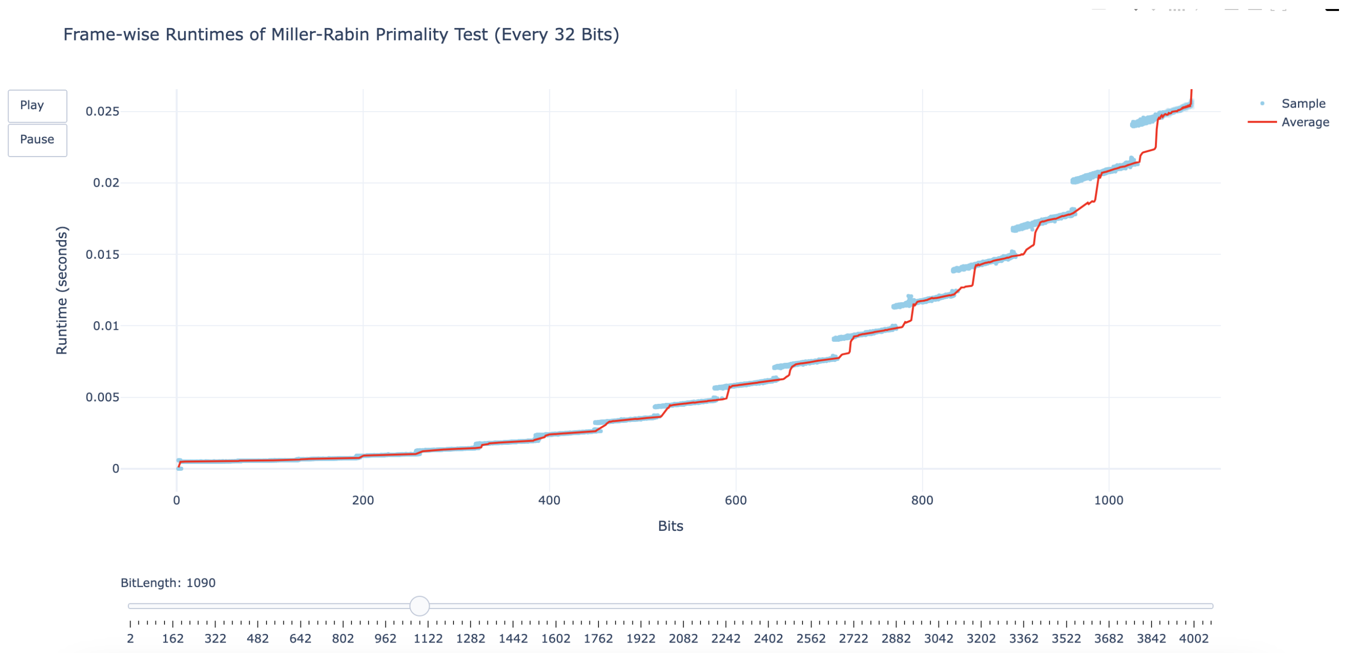


Figure 6. Frame-wise visualization of Miller-Rabin runtime growth, illustrating the $O(k \log^3 n)$ time complexity.

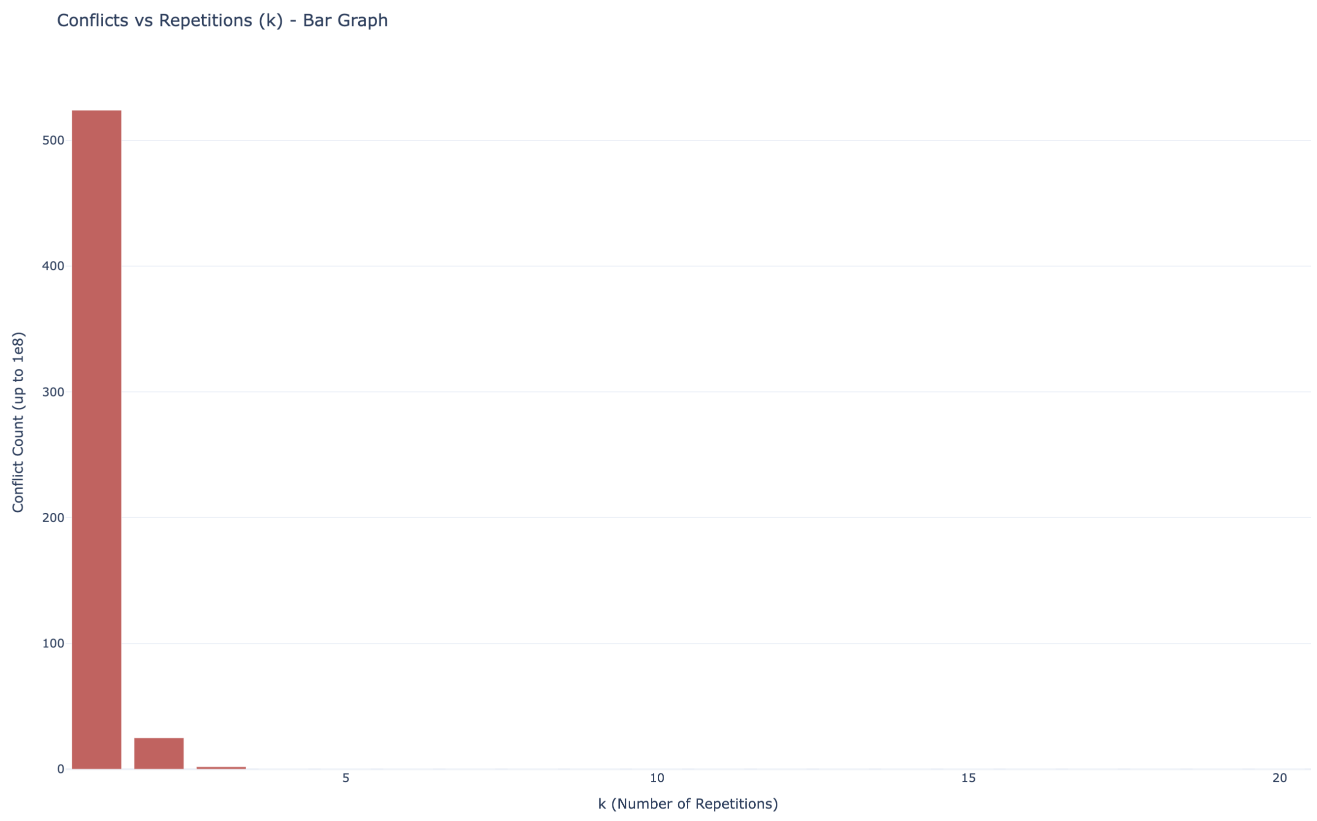
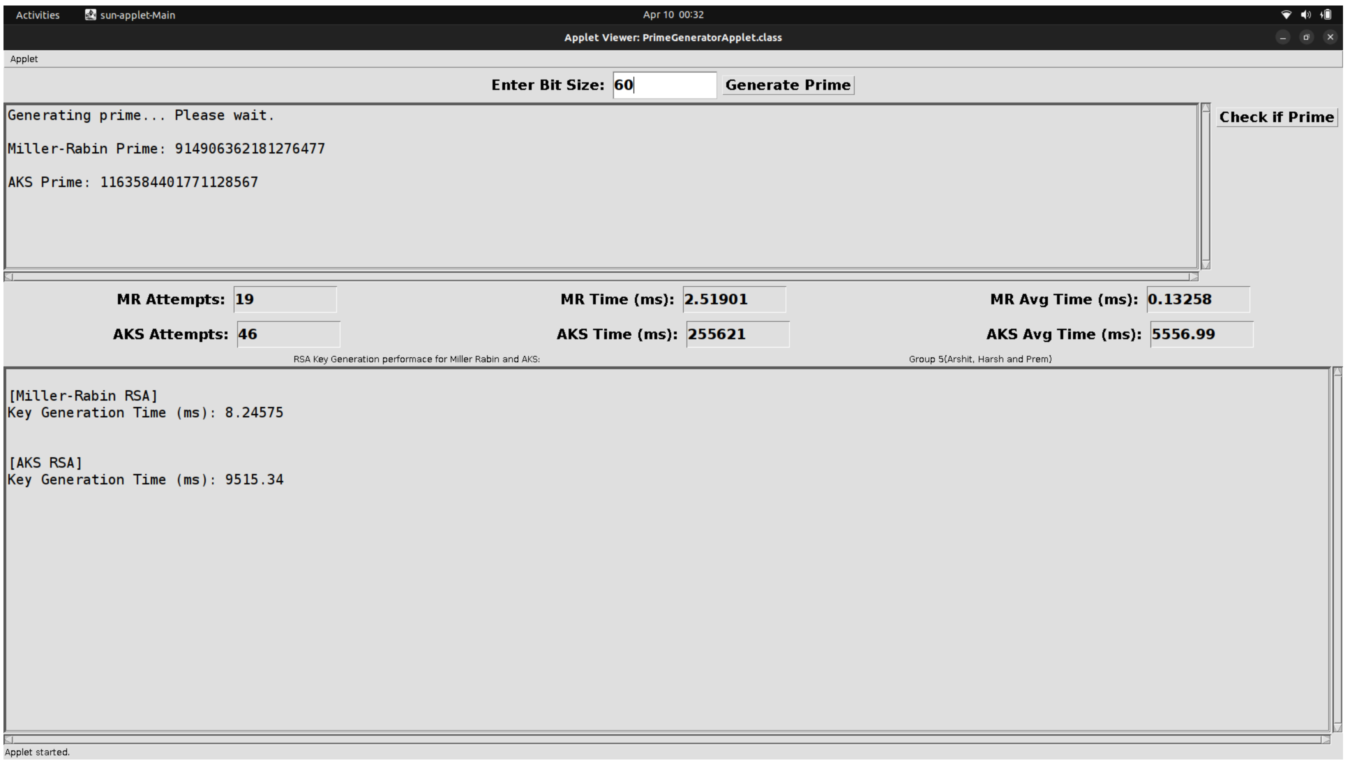


Figure 7. Error probability of Miller-Rabin algorithm with increasing number of iterations (k), tested on numbers from 2 to 10^8.


Activities
sun-applet-Main
Apr 10 00:32
Applet Viewer: PrimeGeneratorApplet.class
Applet
Generate Prime
Check if Prime
RSA Key Generation performace for Miller Rabin and AKS:
Group 5(Arshit, Harsh and Prem)
Applet started.

Enter Bit Size: 60

```
Generating prime... Please wait.

Miller-Rabin Prime: 914906362181276477

AKS Prime: 1163584401771128567
```

MR Attempts: 19   MR Time (ms): 2.51901   MR Avg Time (ms): 0.13258

AKS Attempts: 46   AKS Time (ms): 255621   AKS Avg Time (ms): 5556.99

```
[Miller-Rabin RSA]
Key Generation Time (ms): 8.24575


[AKS RSA]
Key Generation Time (ms): 9515.34
```

Figure 8. Applet showing AKS vs Miller-Rabin performance and RSA generation for $k = 60$ bits

Java Applet

1.9 Introduction

This project aims to evaluate the **performance and correctness** of two primality testing algorithms: the **deterministic AKS algorithm** and the **probabilistic Miller-Rabin test**, through a graphical Java Applet interface. The applet also supports **real-time RSA key generation** using the generated primes, offering both a visual and empirical comparison between the algorithms.

1.10 Applet Features and Workflow

The applet provides an intuitive and interactive interface for exploring the behavior of the two algorithms under different input sizes. The overall workflow is divided into two major parts: Prime Generation and RSA Key Generation.

1.10.1 Prime Generation

Given a bit-length parameter b, the applet generates two b-bit prime numbers:

- One using the **Miller-Rabin** primality test.
- One using the **AKS** primality test.

The process follows:

a. Select a random integer $n \in [2^{b-1}, 2^b - 1]$.
b. Apply the chosen primality test.
c. If n is prime, accept it; otherwise, repeat from step 1.

1.10.2 Performance Metrics

The applet displays the following metrics for both algorithms in the **upper half** of the interface:

- Number of attempts to find a prime.
- Total time taken to generate the prime.
- Average time per attempt.

1.10.3 Prime Verification

To ensure correctness, a **"Check Prime"** feature uses the AKS algorithm to verify the prime obtained via Miller-Rabin.

1.11 RSA Key Generation and Comparison

The **lower half** of the applet focuses on RSA key generation using the two primes p and q obtained above. The steps include:

a. Compute modulus: $n = p \times q$.
b. Set public exponent: $e = 65537$.
c. Compute private key d such that $d \equiv e^{-1} \mod \phi(n)$.

RSA key generation is performed separately for both algorithms, enabling a side-by-side visualization of speed and cryptographic feasibility.

1.12 Code Structure

The code is organized modularly across Java and C++ files for clarity and reusability:

- `main.cpp`: Backend entry point.
- `miller_rabin.cpp/.h`: Miller-Rabin implementation.
- `aks_prime.cpp/.h`, `aks.cpp/.h`: AKS primality test.
- `rsa_utils.cpp/.h`: RSA key generation utilities.
- `check_prime_with_aks.cpp`: AKS-based prime verification.
- `common_utils.cpp/.h`: Random number generation and formatting helpers.
- `PrimeGeneratorApplet.java`: Java Applet frontend.
- `PrimeGeneratorApplet.html`: HTML wrapper for applet execution.

1.13 Experimental Observations

- **Speed:** Miller-Rabin is significantly faster than AKS, especially for large bit-lengths.
- **Accuracy:** Despite its probabilistic nature, Miller-Rabin (with $k = 40$) never produced a composite in all trials. Even $k = 5$ leads to near-zero error rates.
- **Trade-offs:** While AKS offers deterministic guarantees, it is computationally expensive. Miller-Rabin is the preferred choice in practice, particularly for RSA.

1.14 Conclusion

This applet successfully illustrates the trade-offs between **theoretical guarantees** and **practical performance** in primality testing. It highlights how Miller-Rabin's speed and reliability make it ideal for cryptographic applications, whereas AKS, despite its correctness, lacks performance scalability.

Sophie Germain Prime Generator

1.15 Introduction

Sophie Germain numbers hold significant importance in number theory and cryptography due to their unique structure, where both the number p and 2p + 1 are prime. These primes enhance the security of cryptographic systems by supporting the construction of safe primes, which are resistant to certain attacks in protocols like Diffie–Hellman and ElGamal. To generate Sophie Germain primes efficiently, primality testing algorithms such as the Miller–Rabin or AKS algorithm are employed. These algorithms help verify whether both p and 2p + 1 are prime, ensuring the robustness and reliability of cryptographic systems built on such prime structures.

1.16 Experiments

To show the significance of **Miller-Rabin** algorithm, we did the following experiment:-

- We generated Sophie Germain prime pairs using both the AKS algorithm (deterministic) and the Miller–Rabin algorithm (probabilistic). To effectively simulate and visualize the generation process over time, we fast-forwarded the execution using a 60× time scale. This allowed us to observe and compare the rate at which each algorithm produces valid prime pairs. The results were displayed using a dynamic bar graph that updated in real-time, illustrating the number of Sophie Germain prime pairs generated by each algorithm within the same simulated time frame.

1.17 Results

While the AKS algorithm was able to generate approximately 2,000 Sophie Germain prime pairs, the Miller–Rabin algorithm produced nearly 200,000 pairs within the same simulated time frame. This stark contrast highlights the superior practical efficiency of the Miller–Rabin test, making it a more viable choice for large-scale prime generation tasks despite its probabilistic nature.

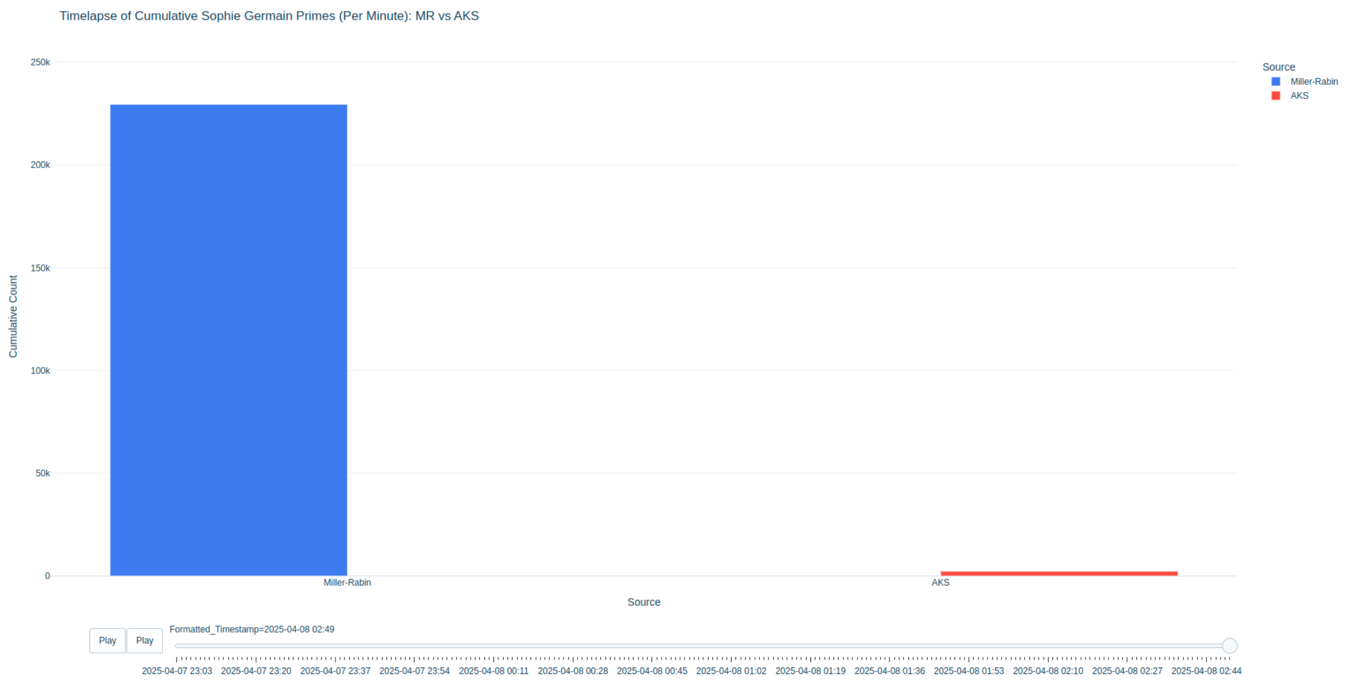


Figure 9. Sophie-Germain Prime Generation for numbers upto 1e8

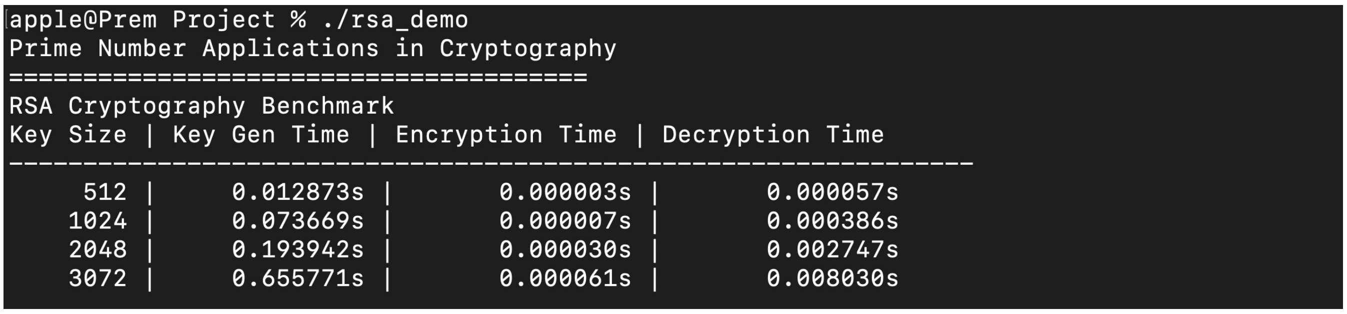
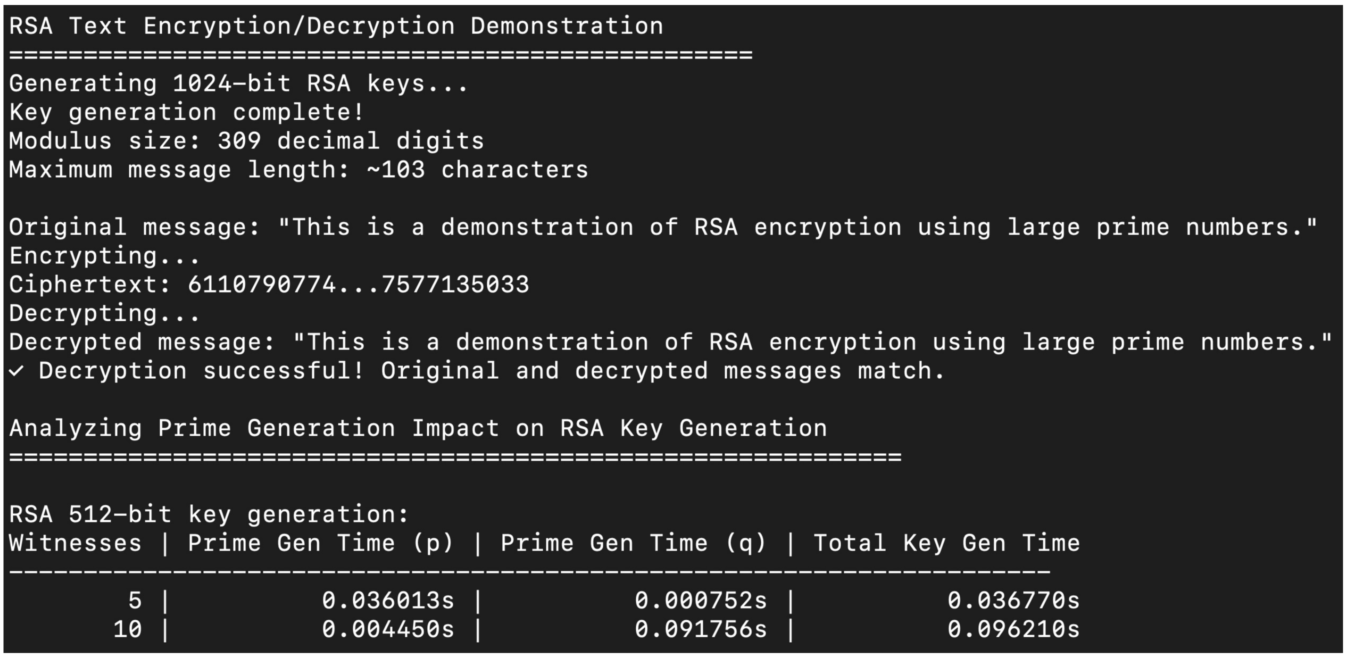

```
apple@Prem Project % ./rsa_demo
Prime Number Applications in Cryptography
=======================================
RSA Cryptography Benchmark
Key Size | Key Gen Time | Encryption Time | Decryption Time
-----------------------------------------------------------------
     512 |     0.012873s |       0.000003s |       0.000057s
    1024 |     0.073669s |       0.000007s |       0.000386s
    2048 |     0.193942s |       0.000030s |       0.002747s
    3072 |     0.655771s |       0.000061s |       0.008030s
```

Figure 10. Sophie-Germain Prime Generation for numbers upto 1e8

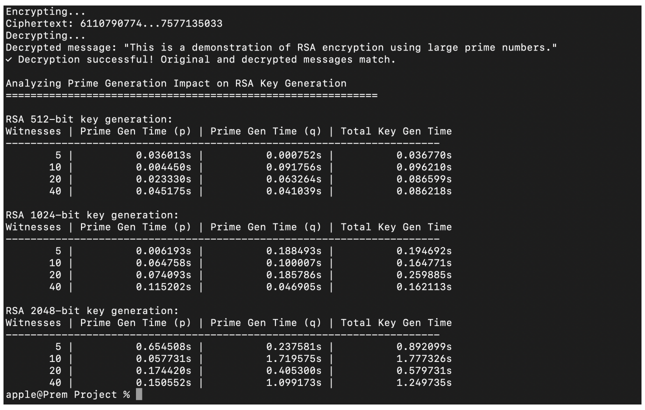

```
RSA Text Encryption/Decryption Demonstration
==================================================
Generating 1024-bit RSA keys...
Key generation complete!
Modulus size: 309 decimal digits
Maximum message length: ~103 characters

Original message: "This is a demonstration of RSA encryption using large prime numbers."
Encrypting...
Ciphertext: 6110790774...7577135033
Decrypting...
Decrypted message: "This is a demonstration of RSA encryption using large prime numbers."
✓ Decryption successful! Original and decrypted messages match.

Analyzing Prime Generation Impact on RSA Key Generation
============================================================

RSA 512-bit key generation:
Witnesses | Prime Gen Time (p) | Prime Gen Time (q) | Total Key Gen Time
----------------------------------------------------------------------
        5 |          0.036013s |          0.000752s |          0.036770s
       10 |          0.004450s |          0.091756s |          0.096210s
```

Figure 11. Sophie-Germain Prime Generation for numbers upto 1e8



```
Encrypting...
Ciphertext: 6110790774...7577135033
Decrypting...
Decrypted message: "This is a demonstration of RSA encryption using large prime numbers."
✓ Decryption successful! Original and decrypted messages match.

Analyzing Prime Generation Impact on RSA Key Generation
============================================================

RSA 512-bit key generation:
Witnesses | Prime Gen Time (p) | Prime Gen Time (q) | Total Key Gen Time
----------------------------------------------------------------------
        5 |          0.036013s |          0.000752s |          0.036770s
       10 |          0.004450s |          0.091756s |          0.096210s
       20 |          0.023330s |          0.063264s |          0.086599s
       40 |          0.045175s |          0.041039s |          0.086218s

RSA 1024-bit key generation:
Witnesses | Prime Gen Time (p) | Prime Gen Time (q) | Total Key Gen Time
----------------------------------------------------------------------
        5 |          0.006193s |          0.188493s |          0.194692s
       10 |          0.064758s |          0.100007s |          0.164771s
       20 |          0.074093s |          0.185786s |          0.259885s
       40 |          0.115202s |          0.046905s |          0.162113s

RSA 2048-bit key generation:
Witnesses | Prime Gen Time (p) | Prime Gen Time (q) | Total Key Gen Time
----------------------------------------------------------------------
        5 |          0.654508s |          0.237581s |          0.892099s
       10 |          0.057731s |          1.719575s |          1.777326s
       20 |          0.174420s |          0.405300s |          0.579731s
       40 |          0.150552s |          1.099173s |          1.249735s
apple@Prem Project % 
```

Figure 12. Sophie-Germain Prime Generation for numbers upto 1e8

Application of Rabin-Miller in Encryption-Decryption

To demonstrate a real-world application of the Miller-Rabin primality test, I built a simple RSA encryption-decryption model. This model simulates key generation, message encryption, and successful decryption using large primes.

Key Features of the Demonstration:

- RSA Key Generation: The program generates RSA keys of varying sizes: 512-bit, 1024-bit, 2048-bit.
- Encryption Decryption: A plaintext message is encrypted using the RSA public key and then decrypted using the private key. The decrypted message is verified against the original.
- Miller-Rabin Usage: The primality of the large integers used to form the RSA modulus is verified using the Miller-Rabin test. The number of witnesses (iterations) directly impacts the time and reliability of key generation.

Performance Evaluation: The performance analysis shows how RSA key generation time is affected by:

- Key size: Larger keys result in longer encryption, decryption, and especially key generation times.
- Witness count in Miller-Rabin: Increasing the number of witnesses improves confidence in primality but increases generation time. For example:
  For 2048-bit keys:
  With 5 witnesses: key generation time $\approx$ 0.89s
  With 40 witnesses: key generation time $\approx$ 1.25s
- Encryption/Decryption speed: Despite larger key sizes, encryption and decryption remain very fast due to the efficiency of modular exponentiation.

Practical Insight: This experiment validates how the Miller-Rabin test balances speed and accuracy in cryptographic applications. While increasing witnesses slows down generation slightly, it ensures the keys are reliably prime—crucial for secure RSA encryption.

Importance of Miller-Rabin in RSA Key Generation

The Miller-Rabin primality test is critical for RSA security due to:

- **Efficiency**: For 4096-bit primes, even 1 iteration achieves error probability $< 1/2^{80}$. This enables practical key generation while maintaining security.
- **Distributed Computation**: Modern protocols use Miller-Rabin for secure multiparty RSA modulus generation ($N = pq$) where no party learns p or q. This prevents single-point failures.

Table 1. RSA Use Cases Requiring Efficient Primality Testing

Application	Implementation Details
HTTPS/TLS	RSA-2048/4096 keys for server authentication
Digital Signatures	X.509 certificates use RSA with SHA-256 hashes
VPN Protocols	OpenVPN uses RSA for TLS handshakes
Secure Email	PGP/GPG encrypt messages using RSA-keys
Blockchain Wallets	RSA-3072 protects private key storage

Table 2. Comparison of Primality Tests in Practice

Test	Use Case	Error Rate	Speed
Miller-Rabin	RSA key generation	4^{-k} (per round)	Fastest
Solovay-Strassen	Legacy systems	2^{-k}	Slower
AKS	Academic/theoretical use	Deterministic	Very Slow

- **Standards Compliance**: ANSI X9.31 mandates 8 Miller-Rabin iterations per prime. Major libraries like OpenSSL implement this for TLS certificates and financial systems.

Applications of RSA Encryption (Implicitly Using Miller-Rabin)

Mathematical Foundation

Let $n = pq$ where Miller-Rabin verifies:

$$\Pr[\text{Composite passes } k \text{ tests}] \leq 4^{-k}$$

For $k = 8$, error probability $\leq 2^{-16}$, making false primes cryptographically negligible.

Performance Comparison

- **Miller-Rabin**: $O(k \log^3 n)$ time vs. **AKS**: $O(\log^6 n)$
- 4096-bit key generation: 18 sec (MR) vs. 8+ hours (AKS)

GitHub Repository

The complete source code is available at:
https://github.com/Harsh-Agrawal-425/Primality_testing

References

[1] Max Prime. *AKS Primality Test - GitHub Repository*. 2024. https://github.com/max-prime-math/aks.

[2] GNU Project and Victor Shoup. *GMP and NTL Libraries for Miller-Rabin Primality Test*. https://gmplib.org/ and https://libntl.org/.

[3] StackExchange User. *Do we still need probabilistic primality testing methods for practical applications?* 2014. https://math.stackexchange.com/questions/1064784/do-we-still-need-probabilistic-primality-testing-methods-for-prac

[4] Wikipedia contributors. *Miller–Rabin primality test — Wikipedia, The Free Encyclopedia*. 2024. https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test.

IBM
https://www.ibm.com/docs/en/linux-on-systems?topic=processes-rsa-key-pair-generation.