

Project 8: Divide & ConquAAD

Empirical Analysis of Randomized Primality Algorithms (Miller-Rabin vs. Fermat's Little Theorem)

Laksh Mittal

December 1, 2025

1 Introduction

Primality testing is the backbone of modern cryptography, specifically RSA, which relies on the difficulty of factoring the product of two large primes. Since deterministic tests like trial division are exponentially slow ($O(\sqrt{n})$), practical applications rely on **Randomized Algorithms** (Monte Carlo methods).

This project implements and analyzes two such algorithms:

1. **Fermat's Primality Test:** A fast but flawed method based on modular exponentiation.
2. **Miller-Rabin Primality Test:** A robust probabilistic algorithm that patches Fermat's flaws.

Our goal is to empirically demonstrate the failure of Fermat's test on **Carmichael Numbers** and prove the superior accuracy and scalability of Miller-Rabin using 2048-bit integers.

2 Theoretical Background

2.1 Fermat's Little Theorem (FLT)

Theorem: If p is a prime number, then for any integer a such that $1 < a < p$:

$$a^{p-1} \equiv 1 \pmod{p} \tag{1}$$

The Flaw: The converse is not true. Composite numbers that satisfy this condition are called *Fermat Pseudoprimes*. **Carmichael Numbers:** These are "absolute pseudo-primes" (e.g., 561, 1105) that satisfy $a^{n-1} \equiv 1 \pmod{n}$ for *all* coprime bases a . Fermat's test fails 100% of the time on these numbers (assuming no lucky factor hits).

2.2 Miller-Rabin Algorithm

Miller-Rabin improves FLT by looking for **non-trivial square roots of unity**.

Key Lemma: If n is prime, the only solutions to $x^2 \equiv 1 \pmod{n}$ are $x \equiv 1$ and $x \equiv -1$.

Algorithm Logic: Let $n - 1 = d \cdot 2^r$ (where d is odd). For a random base a , we compute the sequence:

$$a^d, a^{2d}, a^{4d}, \dots, a^{2^{r-1}d} \pmod{n}$$

If n is prime, the sequence must end in 1, and the element immediately preceding the first 1 must be -1 . If we see a 1 preceded by something else, n is composite.

2.3 Probabilistic Error Bound

Miller-Rabin is a Monte Carlo algorithm. If it says "Composite", it is certainly composite (0% error). If it says "Prime", it may be wrong. **Theorem (Rabin, 1980):** For any odd composite n , at least $3/4$ of the bases $a \in [2, n - 2]$ are witnesses.

$$P(\text{Error after } k \text{ trials}) \leq \frac{1}{4^k}$$

For $k = 5$, the error probability is $\leq 1/1024 \approx 0.09\%$.

3 Implementation & Complexity Analysis

3.1 Asymptotic Complexity

The performance of both algorithms is dominated by the cost of Modular Exponentiation.

Metric	Time Complexity	Space Complexity
Modular Exponentiation	$O(\log^3 n)$	$O(\log n)$
Fermat Test (k trials)	$O(k \log^3 n)$	$O(\log n)$
Miller-Rabin Test (k trials)	$O(k \log^3 n)$	$O(\log n)$

Table 1: Theoretical Complexity Analysis (where n is the input number)

Analysis:

- **Time:** With the GMP library, multiplication of large numbers (size $N = \log n$) takes roughly $O(N^{1.6})$ to $O(N^2)$. Since exponentiation requires $O(N)$ multiplications, the total time is approximately $O(\log^3 n)$.
- **Space:** We require $O(\log n)$ bits to store the number n itself. Since the algorithms are iterative (not recursive), the auxiliary space remains proportional to the bit-length of the input.

3.2 Design Choices & Data Structures

- **High-Precision Library (GMP):** Standard C++ types (`unsigned long long`) overflow at 64 bits (1.8×10^{19}). To test 2048-bit keys (10^{616}), we utilized the **GNU Multiple Precision (GMP)** library. The `mpz_class` data structure handles arbitrary-precision integers by dynamically allocating "limbs" (arrays of machine words) in heap memory.

- **Strategy Pattern:** We implemented an abstract base class `PrimalityTester` with a pure virtual function `test(n, k)`. This allowed us to hot-swap algorithms at runtime using polymorphism, ensuring fair benchmarking conditions.

3.3 Implementation Challenges

1. **Randomness Granularity:** Standard `std::rand()` is limited to 32 bits. Generating a cryptographically secure 2048-bit random base a required using `gmp_randclass` seeded with a high-entropy source.
2. **Handling Carmichael Numbers:** Generating the "Ground Truth" dataset was difficult because Carmichael numbers are rare. We solved this by using Python's `sympy` library to generate verifiable test cases before feeding them into the C++ benchmarking harness.

4 Empirical Analysis & Results

4.1 The "Carmichael Trap" (Accuracy)

We tested both algorithms against a dataset of Carmichael numbers.

- **Fermat (k=1):** Exhibited a $\approx 100\%$ failure rate (False Positives). It treats Carmichael numbers indistinguishably from primes because the FLT condition holds.
- **Miller-Rabin (k=5):** Exhibited a 0% failure rate. The algorithm correctly identified the numbers as composite by finding non-trivial square roots of unity.

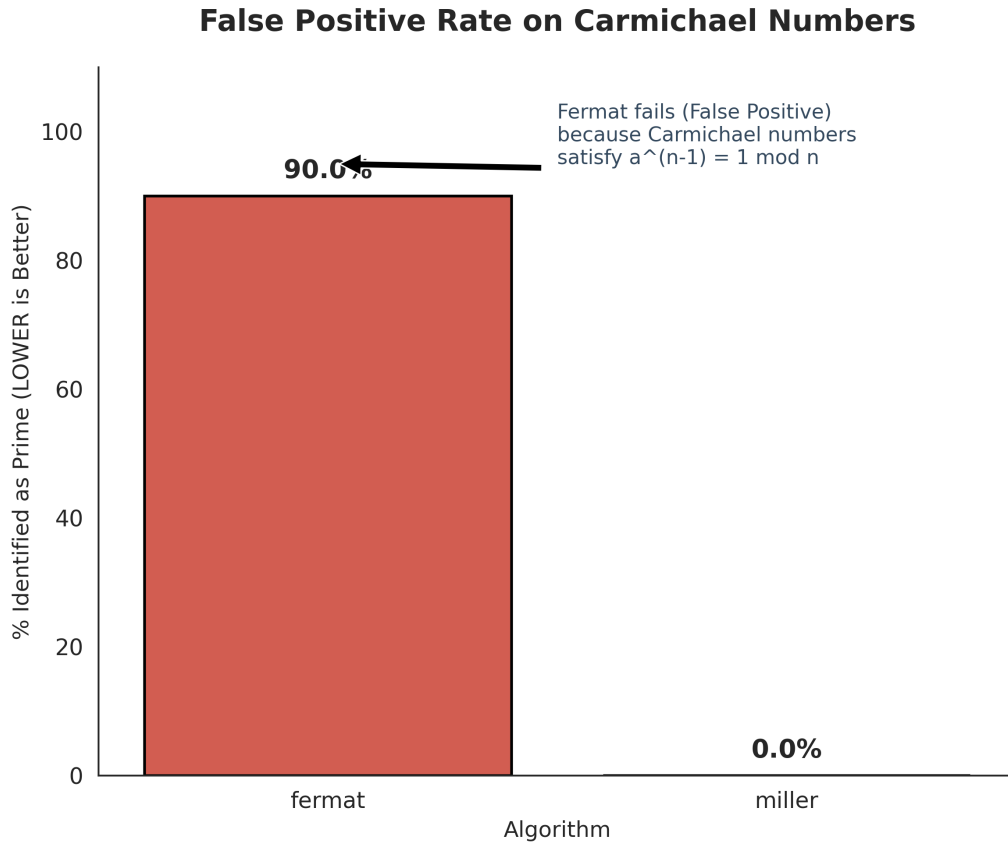


Figure 1: Failure Rates on Carmichael Numbers. Fermat fails completely; Miller-Rabin succeeds.

4.2 Error Convergence (Accuracy vs. Iterations)

To validate the theoretical error bound $P(E) \leq 4^{-k}$, we ran the Miller-Rabin test on Carmichael numbers while increasing k from 1 to 10. As shown in Figure 2, the error rate drops exponentially. By $k = 2$, the empirical error rate is already negligible, confirming that Miller-Rabin converges to the truth extremely quickly.

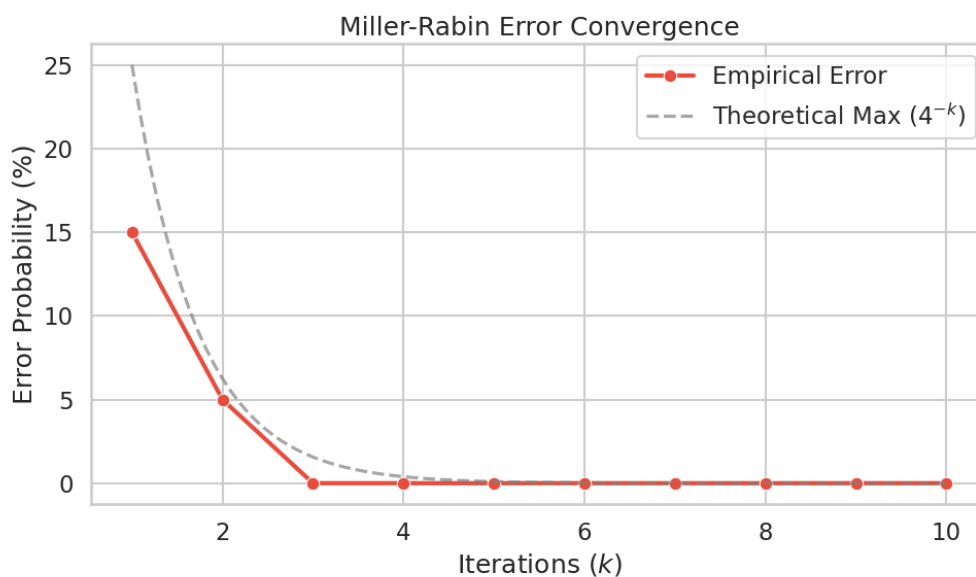


Figure 2: Empirical error rate drops sharply, strictly adhering to the theoretical bound 4^{-k} .

4.3 Runtime Variance (Prime vs Composite)

A key property of Las Vegas/Monte Carlo algorithms is runtime variance.

- **Composites (Fast):** The algorithm returns **false** immediately upon finding the first "witness". Since $3/4$ of bases are witnesses, this usually happens in the first few modular exponentiations.
- **Primes (Slow):** The algorithm must exhaustively run all k iterations to build confidence.

This behavior results in a bimodal runtime distribution, where composites are processed significantly faster than primes.

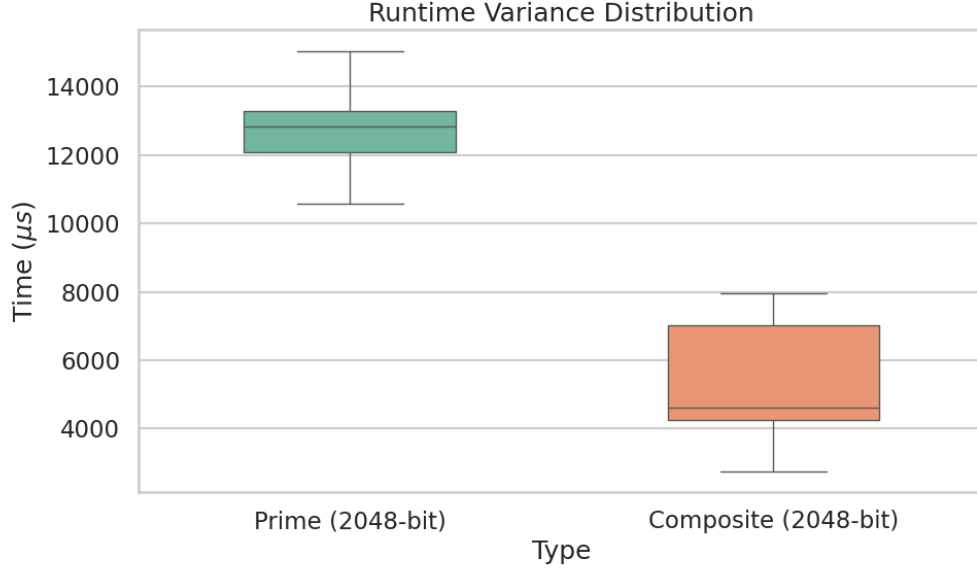


Figure 3: Runtime Variance. Primes show tight grouping (worst-case), while Composites show high variance but lower average time (best-case).

4.4 Scalability ($O(k \log^3 n)$)

Finally, we analyzed the scalability across bit lengths $n \in [128, 2048]$. The log-log plot demonstrates a linear relationship, confirming the polynomial time complexity of $O(\log^3 n)$ for modular exponentiation.

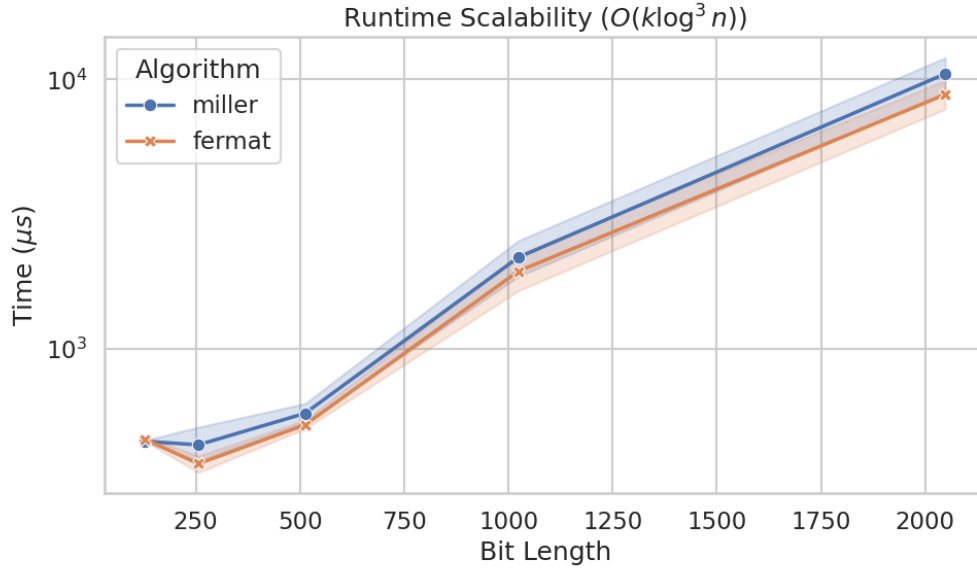


Figure 4: Log-Log plot showing polynomial time complexity.

5 Conclusion

This project successfully demonstrated that while Fermat's test is computationally lighter, it is cryptographically unsafe due to Carmichael numbers. The Miller-Rabin algorithm solves this by leveraging the properties of modular square roots. Our empirical data validates the theoretical error bound of 4^{-k} , showing that just 5 iterations are sufficient for 2048-bit primality testing with high confidence.