An Efficient Deterministic Primality Test

Joseph M. Shunia

December 2023

[Draft]

Abstract

A deterministic primality test with a polynomial time complexity of $\tilde{O}(\log^3(n))$ is presented. The test posits that an integer n satisfying the conditions of the main theorem is prime. Combining elements of number theory and combinatorics, the proof operates on the basis of simultaneous modular congruences relating to binomial transforms of powers of two.

1 Introduction

Primality testing has seen remarkable advancements over the past few decades. A significant breakthrough in this field was the AKS primality test, introduced by Agrawal, Kayal, and Saxena (2002) [1]. The AKS test was the first to offer determinism and polynomial-time complexity, a monumental achievement that resolved a longstanding open question in computational number theory [2]. However, despite its theoretical importance, the AKS test has practical limitations due to its relatively high polynomial time complexity, rendering it inefficient for most applications. Agrawal, Kayal, and Saxena gave a time complexity of $\tilde{O}(\log^{12}(n))$ for the AKS test [1]. This bound was lowered significantly by Lenstra and Pomerance (2011) to $\tilde{O}(\log^6(n))$ [3]. Despite this reduction, AKS remains impractical and is mostly unused.

In the field of cryptography, the unique properties of prime numbers are widely exploited to create cryptographic primitives. It is often the case that many large primes must be generated in rapid succession [4]. To make these cryptographic operations practical, fast probabilistic primality tests such as the Baille-PSW primality test (BPSW) [5] or Miller-Rabin (MR) [6] [7] are used instead of AKS when searching for large primes. Probabilistic primality tests are by definition non-deterministic and may erroneously report a composite integer as being prime. Composite integers which pass a probabilistic primality test are relatively rare and are known as psuedoprimes (PSPs) for the respective test [8]. When generating primes for cryptographic purposes, probabilistic primality tests are often combined or repeated with different parameters in order to achieve an acceptable error-bound that makes it almost certain that no composite integer will pass. However, reducing the error-bound requires additional compute and increases running-time, creating a trade-off.

We present a new deterministic primality test that operates in polynomial time with a time complexity of $\tilde{O}(\log^3(n))$. This efficiency gain opens new avenues for practical applications, particularly in cryptography, where fast and reliable primality testing is desirable [9]. Our main theorem posits a condition for an odd integer n to be prime, based on specific modular congruences related to the binomial transforms of powers of 2. The basis for our test is the following main theorem: Let n be an odd integer satisfying $2^{n-1} \equiv 1 \pmod{n}$. Denote D as the least integer strictly greater than 2 and less than n which does not divide $n-1$. Then, n is prime if and only if a set of simultaneous modular congruences involving D, n, and binomial coefficients hold.

This paper is structured as follows: We begin by presenting the main theorem and its proof, substantiated by supporting lemmas. The proof of our main theorem demonstrates the test's validity for odd prime numbers

and its failure for odd composite numbers. Through this, we establish the deterministic nature of our test. We then describe the algorithm used to compute our test and analyze its computational complexity. We also give pseudocode for our test to show how it can be implemented.

2 Main Theorem

Theorem 1. Let n be an odd integer > 3 satisfying $2^{n-1} \equiv 1 \pmod{n}$. Denote D as the least integer greater than 2 and less than n which does not divide $n-1$. Then, n is prime if and only if the following congruence holds:

$$1 + 2^{\lfloor \frac{n-1}{D} \rfloor} \equiv \left(1 + 2^{\lfloor \frac{n-1}{D} \rfloor}\right)^n \equiv \sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n} \tag{1}$$

2.1 Supporting Lemmas

Lemma 1. Let n be an odd composite integer > 3 satisfying $2^{n-1} \equiv 1 \pmod{n}$. Denote D as the least integer greater than 2 and less than n which does not divide $n-1$. Then, $\lfloor \frac{n-1}{D} \rfloor \neq \mathrm{ord}_n(2)$.

Proof. We are given odd composite $n > 3$ with $2^{n-1} \equiv 1 \pmod{n}$. By the properties of the order of an integer modulo composite n, the smallest k such that $2^k \equiv 1 \pmod{n}$, that is $k = \mathrm{ord}_n(2)$, must be a divisor of $n-1$. Hence, $\mathrm{ord}_n(2) \mid n-1$. Since $\lfloor \frac{n-1}{D} \rfloor$ is strictly less than $n-1$ and D does not divide $n-1$, it follows that $2^{\lfloor \frac{n-1}{D} \rfloor} \not\equiv 1 \pmod{n}$ and therefore, $\lfloor \frac{n-1}{D} \rfloor \neq \mathrm{ord}_n(2)$. □

2.2 Proof of the Main Theorem

Proof of Theorem 1. Let n be an odd integer > 3 satisfying $2^{n-1} \equiv 1 \pmod{n}$.

Define $f(x) = 2^{\lfloor \frac{x-1}{D} \rfloor}$ and D as the least integer greater than 2 that does not divide $n-1$. We begin with the congruence:

$$(1 + f(n))^n \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n} \tag{2}$$

Using the binomial theorem, we expand the left-hand side:

$$\sum_{k=0}^{n} \binom{n}{k} f(n)^k \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \pmod{n} \tag{3}$$

Rewriting this, we must have:

$$\sum_{k=0}^{n} \binom{n}{k} \left(f(n)^k - f(k+1)\right) \equiv 0 \pmod{n}. \tag{4}$$

As a necessary condition of our test, we also require:

$$\sum_{k=0}^{n} \binom{n}{k} f(n)^k \equiv \sum_{k=0}^{n} \binom{n}{k} f(k+1) \equiv 1 + f(n) \pmod{n} \tag{5}$$

By the binomial theorem, we isolate the inner terms:

$$1 + f(n)^n + \sum_{k=1}^{n-1} \binom{n}{k} f(n)^k \equiv 1 + f(n) + \sum_{k=1}^{n-1} \binom{n}{k} f(k+1) \equiv 1 + f(n) \pmod{n} \tag{6}$$

Setting common terms to zero:

$$\sum_{k=1}^{n-1} \binom{n}{k} f(n)^k \equiv \sum_{k=1}^{n-1} \binom{n}{k} f(k+1) \equiv 0 \pmod{n} \tag{7}$$

We have defined n such that $2^{n-1} \equiv 1 \pmod{n}$, which implies $2^n - 2 \equiv \sum_{k=1}^{n-1} \binom{n}{k} \equiv 0 \pmod{n}$. Plugging it in:

$$\sum_{k=1}^{n-1} \binom{n}{k} \equiv \sum_{k=1}^{n-1} \binom{n}{k} f(n)^k \equiv \sum_{k=1}^{n-1} \binom{n}{k} f(k+1) \equiv 0 \pmod{n} \tag{8}$$

Subtracting $\sum_{k=1}^{n-1} \binom{n}{k}$ gives:

$$\left(\sum_{k=1}^{n-1} \binom{n}{k} f(n)^k \right) - \sum_{k=1}^{n-1} \binom{n}{k} \equiv \left(\sum_{k=1}^{n-1} \binom{n}{k} f(k+1) \right) - \sum_{k=1}^{n-1} \binom{n}{k} \equiv 0 \pmod{n} \tag{9}$$

Combining the summations:

$$\sum_{k=1}^{n-1} \left(\binom{n}{k} f(n)^k - \binom{n}{k} \right) \equiv \sum_{k=1}^{n-1} \left(\binom{n}{k} f(k+1) - \binom{n}{k} \right) \equiv 0 \pmod{n} \tag{10}$$

$$\sum_{k=1}^{n-1} \left(\binom{n}{k} f(n)^k - \binom{n}{k} f(k+1) - \binom{n}{k} \right) \equiv 0 \pmod{n} \tag{11}$$

Factoring out the common $\binom{n}{k}$ from the inner terms reveals:

$$\sum_{k=1}^{n-1} \binom{n}{k} \left(f(n)^k - f(k+1) - 1 \right) \equiv 0 \pmod{n} \tag{12}$$

For the above congruence, there are three potential cases we must examine.

Case 1: n is prime

In this case, the congruence always holds. By the binomial theorem, $\binom{n}{k} \equiv 0 \pmod{n}$ for $0 < k < n$ and the congruence simplifies trivially to zero.

Case 2: n is composite and $f(n) \equiv 1 \pmod{n}$

In this case, the congruence may or may not hold. With $f(n) \equiv 1 \pmod{n}$, the congruence simplifies significantly, making it possible:

$$\sum_{k=1}^{n-1} \binom{n}{k} \left(f(n)^k - f(k+1) - 1 \right) \equiv 0 \pmod{n} \tag{13}$$

$$\sum_{k=1}^{n-1} \binom{n}{k} \left(1^k - f(k+1) - 1 \right) \equiv 0 \pmod{n} \tag{14}$$

$$\sum_{k=1}^{n-1} \binom{n}{k} \cdot (-f(k+1)) \equiv 0 \pmod{n} \tag{15}$$

Case 3: n is composite and $f(n) \not\equiv 1 \pmod{n}$

In this case, the congruence cannot hold.

Since $\sum_{k=1}^{n-1} \binom{n}{k} \equiv 0 \pmod{n}$, we may add or subtract $\sum_{k=1}^{n-1} \binom{n}{k}$ from our congruence an arbitrary number of times and n must still divide the sum. The same applies to $\sum_{k=1}^{n-1} \binom{n}{k} f(n)^k$ and $\sum_{k=1}^{n-1} \binom{n}{k} f(k+1)$. Therefore, we surmise that the following must hold for all $W, X, Y \in \mathbb{Z}$:

$$W\left(\sum_{k=1}^{n-1} \binom{n}{k} f(n)^k\right) - X\left(\sum_{k=1}^{n-1} \binom{n}{k} f(k+1)\right) - Y\left(\sum_{k=1}^{n-1} \binom{n}{k}\right) \equiv 0 \pmod{n} \tag{16}$$

$$\left(\sum_{k=1}^{n-1} \binom{n}{k} f(n)^k W\right) - \left(\sum_{k=1}^{n-1} \binom{n}{k} f(k+1) X\right) - \left(\sum_{k=1}^{n-1} \binom{n}{k} Y\right) \equiv 0 \pmod{n} \tag{17}$$

$$\sum_{k=1}^{n-1} \binom{n}{k} \left(W \cdot f(n)^k - X \cdot f(k+1) - Y\right) \equiv 0 \pmod{n} \tag{18}$$

If $\binom{n}{k} \not\equiv 0 \pmod{n}$ for any k, as we know it must be for some k, and $f(n)^k \not\equiv f(k+1) \pmod{n}$, then the summation within our congruence may sum to any value by adjusting the values of W, X, and Y. For the congruence to hold, we would require all integers to be equivalent to 0 (mod n). However, this is a contradiction, as $n \neq 1$ and it is impossible. Therefore, to prove that the congruence cannot hold, it suffices to show that $f(n)^k \not\equiv f(k+1) \pmod{n}$ for some k with $\binom{n}{k} \not\equiv 0 \pmod{n}$.

For composite n, Kummer's theorem [10] states that $\binom{n}{k}$ is divisible by n if and only if at least one of the base-p digits of k is greater than the corresponding digit of n, where p is a prime dividing n. For these values of k, the congruence $f(n)^k \equiv f(k+1) \pmod{n}$ is unlikely because $f(n)^k$ and $f(k+1)$ represent different powers of 2 modulo n. It would necessarily imply $2^{\left\lfloor \frac{n-1}{D} \right\rfloor k} \equiv 2^{\left\lfloor \frac{k}{D} \right\rfloor} \pmod{n}$. This can only occur at values k in the range $0 < k < n$ which are solutions to the modular congruence:

$$\left\lfloor \frac{n-1}{D} \right\rfloor k - \left\lfloor \frac{k}{D} \right\rfloor \equiv 0 \pmod{\operatorname{ord}_n(2)} \tag{19}$$

- **Cycling of $f(n)^k = 2^{\left\lfloor \frac{n-1}{D} \right\rfloor k}$ Modulo n:** This term cycles through all possible residues of powers of 2 modulo n, as k varies from 1 to $n-1$. This is because $\left\lfloor \frac{n-1}{D} \right\rfloor$ is fixed for a given n, and as k increases, the exponent $\left\lfloor \frac{n-1}{D} \right\rfloor k$ effectively cycles through various powers.
- **Behavior of $f(k+1) = 2^{\left\lfloor \frac{k}{D} \right\rfloor}$ Modulo n:** In contrast, this term does not cycle through all residues in the same way. The exponent $\left\lfloor \frac{k}{D} \right\rfloor$ changes more slowly with k, as it's the floor of a fraction. Therefore, for multiple consecutive values of k, the value of $2^{\left\lfloor \frac{k}{D} \right\rfloor}$ remains the same.

Applying the Pigeonhole Principle [11], there must exist at least one value of k such that $\binom{n}{k}\left(f(n)^k - f(k+1)\right) \not\equiv 0 \pmod{n}$. This non-zero term in the sum, when multiplied by the non-zero binomial coefficient, ensures that the sum cannot be congruent to zero modulo n for all possible permutations of W, X, Y.

Conclusion:

When n is an odd prime integer > 3, the congruence holds. When n is an odd composite integer > 3, by Lemma 1 we have $f(n) \not\equiv 1 \pmod{n}$ and therefore, the congruence cannot hold. Hence, the theorem is proven. □

3 Algorithm

INPUT: An integer $n > 1$.

1. If $n \equiv 0 \mod 2$:
   (a) If n equals 2, output PRIME.
   (b) Otherwise, output COMPOSITE.
2. If n equals 3, output PRIME.
3. If $2^{n-1} \not\equiv 1 \pmod{n}$, output COMPOSITE.
4. Find the least integer D that is greater than 2 and less than n which does not divide $n-1$.
5. Set $A = 2^{\lfloor \frac{n-1}{D} \rfloor} \bmod n$.
6. Set $B = (1+A)^n \bmod n$.
7. If $B \not\equiv 1 + A \pmod{n}$, output COMPOSITE.
8. Set $C = \sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \bmod n$.
9. If $C \not\equiv 1 + A \pmod{n}$, output COMPOSITE.
10. Output PRIME;

3.1 Time Complexity Analysis

3.1.1 Algorithm Overview

The given algorithm is a primality test that involves several computational steps, including modular arithmetic and polynomial exponentiation in the ring $\mathbb{Z}/n\mathbb{Z}$.

3.1.2 Analysis of Individual Operations

1. **Check for Even** n:
   This step involves calculating $n \bmod 2$ and has a time complexity of $O(1)$.
2. **Modular Exponentiation** $2^{n-1} \mod n$:
   This step requires modular exponentiation with a $\log(n)$-digit base and a $\log(n)$-digit exponent. The time complexity of modular exponentiation is $O(\log(n)M(n))$.
3. **Finding** D:
   Finding the least integer $D > 2$ that does not divide $n-1$ takes at most $O(\log(n))$ steps, with each step requiring $O(1)$ time for the mod operation. Hence, the overall complexity is $O(\log(n))$.
4. **Computing** A **and** B:
   Each of these steps involves modular exponentiation similar to Step 2, and thus each has a time complexity of $O(\log(n)M(n))$.
5. **Computing** C:
   Computing C involves exponentiating a polynomial in the ring $\mathbb{Z}/n\mathbb{Z}$ with $O(\log(n))$ terms and summing the coefficients. The mathematics underlying the computation of C is described and proven in §4.
   (a) Summing the polynomial coefficients can be done in $O(\log^2(n))$ time.

(b) Exponentiation using repeated squaring takes $O(\log(n))$ steps, and each step requires $O(M(n)\log(n))$ time due to the multiplication of polynomials of size $O(\log(n))$.

Therefore, the overall complexity for computing C is $O(\log^2(n)M(n))$.

6. **Comparisons**:

The final steps involve comparisons which are $O(1)$ operations.

3.1.3 Overall Time Complexity

The dominant time complexity in the algorithm comes from computing C. Therefore, the overall time complexity of the algorithm is $T(n) = O(\log^2(n)M(n))$.

Harvey and van Der Heoven (2021) have given an algorithm for integer multiplication which has a time complexity $M(n) = O(\log(n)\log\log(n))$ [12]. This would give our algorithm an overall time complexity of:

$$T(n) = O(\log^2(n)M(n)) = O(\log^2(n)\log(n)\log\log(n)) = O(\log^3(n)\log\log(n)) = \tilde{O}(\log^3(n)) \tag{20}$$

3.1.4 Conclusion

The overall complexity is polynomial in the size of n when expressed in terms of bit operations, making the algorithm efficient for large values of n.

4 Efficient Calculations via Polynomial Rings

In this section, we define a special polynomial ring R with a modular variation M, and show how M can be used to efficiently calculate the value of $\sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n}$. The standard approach to calculating this value requires evaluating $\binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n}$ for each k and summing the results, which takes time exponential in n. Conversely, our approach offers an efficient polynomial time complexity of $\tilde{O}(\log(D) \cdot \log^2(n))$ (See: §3.1).

4.1 Ring Definition

Definition 1 (Polynomial ring R)**.** We construct a polynomial ring $R = \mathbb{Z}[x]$ in which addition is carried out as usual, and multiplication is followed by polynomial degree reduction via a special substitution function, denoted Φ [13]. The multiplication operation $*$ is defined as follows:

$$P(x) * Q(x) = \Phi(P(x) \cdot Q(x)) \text{ for all } P(x), Q(x) \in R, \tag{21}$$

Φ is an operation that enforces any defined substitution rules for the polynomial's terms upon multiplication. In our case, for the generator x, we have:

$$x * x^{d-1} = x^d = \Phi(x^d) = N, \text{ where } N \in R \tag{22}$$

For any polynomial in our ring R, the function Φ is distributed to the individual terms and implicitly applied to reduce terms. Whenever the term x^d appears, it is replaced with its defined mapping. The mappings in Φ are extended to terms of higher degrees, meaning $\Phi(x^{d+k}) = \Phi(x^d * x^k) = Nx^k$, for $k > 1$. Φ is applied recursively to the polynomial until and its terms until the degree of the polynomial is less than d. For terms which do not match a defined substitution rule, Φ returns them as they are.

Example 1. We take the ring R as defined in Definition 1 with $N = 2$, $d = 3$. Hence, $\Phi(x^3) = 2$. $P(x) := 1 + x + 3x^3 + x^4 + x^6, P(x) \in R$.

$$\begin{aligned}
\Phi(P(x)) &= \Phi(1 + x + 3x^3 + x^4 + x^6) && (23)\\
&= \Phi(1) + \Phi(x) + \Phi(3x^3) + \Phi(x^4) + \Phi(x^6) && (24)\\
&= 1 + x + \Phi(2 \cdot 3) + \Phi(2 \cdot x) + \Phi(2 \cdot x^3) && (25)\\
&= 1 + x + 6 + 2x + \Phi(2 \cdot 2) && (26)\\
&= 7 + 3x + 4 && (27)\\
&= 11 + 3x && (28)
\end{aligned}$$

4.2 Ring Axioms

We assert that the ring R with the modified operation $*$ still satisfies the standard ring axioms. Specifically, we show that $(R, +, *)$ is associative, commutative (with respect to addition), and has an additive identity and an additive inverse for every element. Additionally, the distributive property of multiplication over addition is preserved under the operation $*$.

4.2.1 Multiplicative Properties

Proposition 1 (Distributivity of $*$)**.** The operation $*$ is distributive over addition in the ring R.

Proof. The modified multiplication $*$ is distributive over addition because for any $P(x), Q(x), S(x) \in R$,

$$\begin{aligned}
P(x) * (Q(x) + S(x)) &= \Phi(P(x) \cdot (Q(x) + S(x))) && (29)\\
&= \Phi(P(x) \cdot Q(x) + P(x) \cdot S(x)) && (30)\\
&= \Phi(P(x) \cdot Q(x)) + \Phi(P(x) \cdot S(x)) && (31)\\
&= P(x) * Q(x) + P(x) * S(x) && (32)
\end{aligned}$$

Where the second equality uses the distributive property of the standard multiplication in $\mathbb{Z}[x]$ and the linearity of Φ with respect to polynomial addition. □

Proposition 2 (Associativity of $*$)**.** The multiplication operation $*$ in the ring R is associative.

Proof. To prove associativity, we need to show that for any $P(x), Q(x), S(x) \in R$:

$$(P(x) * Q(x)) * S(x) = P(x) * (Q(x) * S(x)) \quad (33)$$

Expanding the left-hand side:

$$\begin{aligned}
(P(x) * Q(x)) * S(x) &= \Phi(P(x) \cdot Q(x)) * S(x) && (34)\\
&= \Phi(\Phi(P(x) \cdot Q(x)) \cdot S(x)) && (35)
\end{aligned}$$

Similarly, for the right-hand side:

$$\begin{aligned}
P(x) * (Q(x) * S(x)) &= P(x) * \Phi(Q(x) \cdot S(x)) && (36)\\
&= \Phi(P(x) \cdot \Phi(Q(x) \cdot S(x))) && (37)
\end{aligned}$$

Since the standard multiplication in $\mathbb{Z}[x]$ is associative and Φ is a well-defined operation that respects this associativity, we have:

$$\Phi(\Phi(P(x) \cdot Q(x)) \cdot S(x)) = \Phi(P(x) \cdot \Phi(Q(x) \cdot S(x))) \quad (38)$$

Which shows that:

$$(P(x) * Q(x)) * S(x) = P(x) * (Q(x) * S(x)) \tag{39}$$

□

4.2.2 Additive Properties

Proposition 3 (Preservation of additive properties)**.** The commutative and associative properties of addition, and the existence of an additive identity and inverses, are maintained in the ring R.

Proof. The additive structure of R remains unchanged. Thus, the commutative and associative properties of addition, and the existence of an additive identity and inverses, are inherited directly from $\mathbb{Z}[x]$. □

4.3 Modular Ring Definition

Definition 2 (Modular polynomial ring M)**.** Let $R = \mathbb{Z}[x]$ be our polynomial ring as defined in Definition 1, where the multiplication is modified by a substitution function Φ as described previously. Let n be a positive integer. We define the modular variation of R, denoted as M, to be the ring of polynomials with coefficients in $\mathbb{Z}/n\mathbb{Z}$ and with multiplication modified by a corresponding substitution function Φ_M. Formally, $M = (\mathbb{Z}/n\mathbb{Z})[x]$, where the coefficients of the polynomials in M are taken modulo n, and the multiplication in M is given by

$$P(x) * Q(x) = \Phi_M(P(x) \cdot Q(x)) \text{ for all } P(x), Q(x) \in M, \tag{40}$$

Φ_M is defined analogously to Φ but operates within the context of the coefficients being in $\mathbb{Z}/n\mathbb{Z}$.

Proposition 4. The ring $M = (\mathbb{Z}/n\mathbb{Z})[x]$ with the modified multiplication operation $*$, as defined by the substitution function Φ_M, inherits the standard ring axioms from $R = \mathbb{Z}[x]$.

Proof. Since the ring M is structurally analogous to R with the only difference being the coefficient domain ($\mathbb{Z}/n\mathbb{Z}$ instead of $\mathbb{Z}$), and the modified multiplication operation $*$ in M is defined similarly to R using Φ_M, analogous to Φ, M inherits the ring properties of R. This includes the associativity and commutativity of addition, the existence of an additive identity and inverses, the distributivity of multiplication over addition, and the associativity of multiplication. These properties are preserved under the transition from $\mathbb{Z}$ to $\mathbb{Z}/n\mathbb{Z}$ coefficients and the analogous definition of $*$ in M. □

4.4 Ring Calculations

We now demonstrate how our modular polynomial ring M can be used to efficiently calculate the value of $\sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod n$.

In an earlier paper relating the central binomial coefficients and Gould's sequence to polynomial rings [13], we showed how a multivariate polynomial ring, analogous to the univariate polynomial ring we have defined herein, can be used generally to compute the binomial transforms of recursive integer sequences. We apply the same technique here.

Theorem 2. Let n, k be non-negative integers. Let D be an integer > 2. Define the ring M as in Definition 2 with $d = D - 1$, $N = 2$, hence $\Phi(x^{D-1}) = 2$. $P(x) := x, P(x) \in M$. Taking the sum of the coefficients modulo n in the polynomial expansion of $P(x)^k$, gives $2^{\lfloor \frac{k}{D} \rfloor} \pmod n$.

Proof. Examining $D = 2$, we can see:

$$\begin{aligned}
P(x)^0 &= 1 = 2^{\lfloor 0/2 \rfloor} \\
P(x)^1 &= x = 1 = 2^{\lfloor 1/2 \rfloor} \\
P(x)^2 &= x^2 = 2 = 2^{\lfloor 2/2 \rfloor} \\
P(x)^3 &= x^3 = 2x = 2 = 2^{\lfloor 3/2 \rfloor} \\
P(x)^4 &= x^4 = 2x^2 = 4 = 2^{\lfloor 4/2 \rfloor} \\
P(x)^5 &= x^5 = 4x = 4 = 2^{\lfloor 5/2 \rfloor} \\
P(x)^6 &= x^6 = 4x^2 = 8 = 2^{\lfloor 6/2 \rfloor} \\
&\vdots
\end{aligned}$$

We proceed by induction.

Base Case: For $k = 0$, we have $P(x)^0 = 1$, and the sum of coefficients is $2^{\lfloor \frac{0-1}{D} \rfloor} = 2^0 = 1$.

Inductive Step: Assume the theorem holds for some $k \geq 0$, i.e., the sum of coefficients modulo n in $P(x)^k$ is $2^{\lfloor \frac{k}{D} \rfloor} \pmod{n}$. We need to show it also holds for $k + 1$.

Consider $P(x)^{k+1} = P(x)^k * P(x)$. By the definition of $*$ and Φ, the degree of $P(x)^{k+1}$ gets reduced by Φ every time it reaches D. The number of times this reduction happens is $\lfloor \frac{k}{D} \rfloor$. Therefore, the sum of coefficients in $P(x)^{k+1}$ should be $2^{\lfloor \frac{k+1}{D} \rfloor}$, as each reduction by Φ doubles the sum of coefficients. Hence, the theorem holds for $k + 1$.

Conclusion: By the principle of mathematical induction, the theorem is proven. □

Theorem 3. Let n an integer > 0. Let D be an integer > 2. Define the ring M as in Definition 2 with $d = D - 1$, $N = 2$, hence $\Phi(x^{D-1}) = 2$. $P(x) := x, P(x) \in M$. Then taking the sum of the coefficients modulo n in the polynomial expansion of $(1 + P(x))^n$ is equivalent to $\sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \pmod{n}$.

Proof. By the binomial theorem, we have:

$$(1 + x)^n = \sum_{k=0}^{n} \binom{n}{k} P(x)^k \tag{41}$$

Hence, when we evaluate this polynomial at $x = 1$, we sum the coefficients of the polynomial to get:

$$\sum_{k=0}^{n} \binom{n}{k} 2^{\lfloor \frac{k}{D} \rfloor} \quad \text{(by Theorem 2)} \tag{42}$$

This completes the proof. □

5 Implementation Details

5.1 Reference Implementation

A sample open source .NET implementation, with test data, is available on the author's Github page [14].

5.2 Pseudocode Implementation

To demonstrate how our test may be implemented, we offer a pseudocode implementation of the key functions involved.

5.2.1 IsPrime Function

Require: An integer $n > 1$
  function IsPrime(n)
    if $n \bmod 2 = 0$ **then**
      if $n = 2$ **then**
        return true ▷ n is prime
      else
        return false ▷ n is composite
      end if
    end if
    if $n = 3$ **then**
      return true ▷ n is prime
    end if
    $fermat \leftarrow \text{Pow}(2, n-1, n)$ ▷ Fermat pseudoprime test to base 2
    if $fermat \neq 1$ **then**
      return false ▷ n is composite
    end if
    $D \leftarrow 2$
    $log \leftarrow \text{Log2}(n)$
    for $i \leftarrow 3$ **to** $\text{Max}(log, 3)$ **do**
      $D \leftarrow i$
      $m \leftarrow (n-1) \bmod D$
      if $m \neq 0$ **then**
        break
      end if
    end for
    $A \leftarrow \text{Pow}(2, \lfloor (n-1)/D \rfloor, n)$
    $expectedValue \leftarrow (A+1) \bmod n$
    $B \leftarrow \text{Pow}(A+1, n, n)$
    if $B \neq expectedValue$ **then**
      return false ▷ n is composite
    end if
    $polyDegree \leftarrow D - 1$ ▷ Subtract 1 to account for zero indexing in arrays
    $poly \leftarrow \text{PolyPow}([1,1], n, n, polyDegree, [2])$
    $C \leftarrow \text{PolyEval}(poly, 1) \bmod n$ ▷ Evaluate at x=1 to sum coefficients
    if $C \neq expectedValue$ **then**
      return false ▷ n is composite
    end if
    return true ▷ n is prime
  end function

5.2.2 PolyPow Function

  function PolyPow($poly_A$, k, n, d, $polyMapping$)
    $polyB \leftarrow [1]$ ▷ Initialize polyB as a polynomial with constant term 1

```
    while k > 0 do
        if k mod 2 = 1 then
            polyB ← PolyMul(polyA, polyB, n)                ▷ Multiply polynomials modulo n
            polyB ← PolyReduce(polyB, d, polyMapping, n)    ▷ Reduce degree of polyB
            if k = 1 then
                break
            end if
        end if
        polyA ← PolyMul(polyA, polyA, n)                    ▷ Square polyA modulo n
        polyA ← PolyReduce(polyA, d, polyMapping, n)        ▷ Reduce degree of polyA
        k ← k/2
    end while
    return polyB
end function
```

5.2.3 PolyReduce Function

```
function PolyReduce(polyA, d, mappingPoly, n)
    if Length(polyA) ≤ d then
        return polyA
    end if
    polyB ← Clone(polyA)                                    ▷ Copy the polynomial to a new array
    polyDegree ← Degree(mappingPoly)                        ▷ Find degree of the mapping polynomial
    degreeDelta ← d − polyDegree
    for i ← Length(polyB) − 1 downto d + 1 do
        if polyB[i] = 0 then
            continue
        end if
        for j ← i − 1 − degreeDelta, k ← polyDegree downto 0 do
            polyB[j] ← polyB[j] + polyB[i] × mappingPoly[k]  ▷ Degree reduction step
        end for
        polyB[i] ← 0
    end for
    polyC ← new array of size d + 1
    for i ← 0 to min(Length(polyC), Length(polyB)) − 1 do   ▷ Take coefficients modulo n
        polyC[i] ← polyB[i]
        if n ≠ 0 then
            polyC[i] ← polyC[i] mod n
        end if
    end for
    return polyC
end function
```

6 Acknowledgements

The author would like to thank the kind users at `mersenneforum.org` for their helpful feedback.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, pages 781–793, 2002.

[2] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[3] Hendrik W Lenstra and Carl Pomerance. Primality testing with gaussian periods. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1951): 3376–3390, 2011.

[4] Hendrik W Lenstra. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[5] Robert Baillie and Samuel S Jr Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152): 1391–1417, 1980.

[6] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1): 128–138, 1980.

[7] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[8] Samuel S Jr Wagstaff. Pseudoprimes and a generalization of artin's conjecture. *Acta Arithmetica*, 41 (2):141–150, 1983.

[9] Carl Pomerance. The use of elliptic curves in cryptography. *Advances in Cryptology*, pages 203–208, 1984.

[10] Ernst Eduard Kummer. Ueber die teilbarkeit der zahlen durch ihre grössten gemeinschaftlichen theiler. *Journal für die reine und angewandte Mathematik*, 54:313–321, 1857. doi: 10.1515/crll.1857.54.313. URL `https://eudml.org/doc/148450`.

[11] Kenneth H Rosen. *Discrete Mathematics and its Applications*. McGraw-Hill Education, 7 edition, 2012.

[12] Joris van Der Hoeven David Harvey. Integer multiplication in time o(n log n). *Annals of Mathematics*, 2021. doi: 10.4007/annals.2021.193.2.4.hal-02070778v2.

[13] Joseph M. Shunia. A polynomial ring connecting central binomial coefficients and gould's sequence, 2023. URL `https://arxiv.org/abs/2312.00302`.

[14] Joseph M. Shunia. A sample .net implementation of the primality test. `https://github.com/jshunia/Shunia.Primes`, 2023. Accessed: December 4 2023.