PRIMALITY TESTING, FACTORING ALGORITHMS, AND THEIR APPLICATIONS IN CRYPTOGRAPHY

ARHAM RAKHECHA

Abstract

Confirming primality and factoring large values are two areas in mathematics with roots buried deep in history. This paper explores various primality tests, from basic probabilistic tests such as Fermat's primality test to deterministic algorithms such as the monumental Agrawal-Kayal-Saxena (AKS) test. The AKS primality test signified a large shift in the field, and the original theorem and lemmas are described within this paper. Following this, various factoring algorithms are observed, going from simple to complex theorems. The paper progresses by linking together these two crucial concepts to their cryptographic applications, and ending with a practical analysis of factoring algorithms. The algorithms included in this paper are the following: Fermat's primality test, Miller-Rabin primality test, Solovay-Strassen primality test, AKS primality test, Fermat's factorization, Trial Division, Generalized Trial Division, Quadratic Sieve and Pollard's Rho.

Contents

Introduction

The mystery of prime numbers has a profound heritage, with mathematicians investigating their properties for over twenty-three centuries.[1] This is most notably associated with Euclid's theorem on the infinitude of primes, circa 300 BCE.[2] Primality testing itself can be dated back to the third century BCE with the Sieve of Eratosthenes.[3] These cases illustrate the significance assigned to prime numbers even in ancient Egypt and Greece. Prime numbers emit an alluring aura, as they form part of the mathematical threads woven into the fabric of the universe. Research involving prime numbers is thoroughly exciting and has vast potential.

Carl Friedrich Gauss stated that "The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."[4] Primality testing is simply the determination of whether a given number n is prime or composite. The two types of tests considered in this paper are probabilistic and deterministic primality tests. Probabilistic algorithms are assigned an error probability, diminishing their accuracy, whilst deterministic algorithms can identify prime and composite numbers without fault. Factoring algorithms refer to processes that break down a given number into its prime factors. Various algorithms exist that conduct this process, at distinct speeds and ranges, many of which are developments on earlier techniques.

The focus of this paper stems from the applications of the aforementioned systems, namely in asymmetric cryptography. This form of cryptography encompasses two mathematically linked keys, of which one is freely shared, and the other is kept private. Primality testing is incorporated in the generation of these keys, whilst factorization is necessary to decipher them.

After beginning with a light overview of several primality tests, this paper more intensely examines the Agrawal-Kayal-Saxena primality test. Developing further upon this, factoring algorithms are presented. This is followed by applications in cryptography, which binds the two ideas. It concludes with an evaluation of practical tests conducted on a portion of algorithms.

1. Primality Testing

This section will cover several primality tests, including Fermat's test, the Miller-Rabin test, Solovay-strassen test and the basics of the Agrawal-Kayal-Saxena test.

1.1. **Fermat's Primality Test.**

Theorem 1.1 (Fermat's Little Theorem)**.** *Let p be a prime which does not divide the integer a, then:*

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

Proof. Begin by listing the first $p-1$ positive multiples of a:

$$a, 2a, 3a, \ldots, (p-1)a.$$

[1] [Me8]
[2] [Me8]
[3] [Pat21]
[4] [Gau86]

Suppose wa and va are the same modulo p, then we have

$$wa \equiv va \pmod{p},$$

so the $p-1$ multiples of a that are shown above are distinct and nonzero; they must have congruence with

$$1, 2, 3, \ldots, (p-1)$$

in some order.

If we multiply these congruences together, we will get:

$$a * (2a) * (3a) * \cdots * ((p-1)a) \equiv 1 * 2 * 3 * \cdots * (p-1) \pmod{p},$$

which is:

$$a^{(p-1)} * (p-1)! \equiv (p-1)! \pmod{p}.$$

Divide both sides by $(p-1)!$ to get Theorem 1.1 ■

Theorem 1.1 is not an "if, and only if" statement, as there exist Carmichael numbers, which are composite numbers for which the conditions are satisfied.

Definition 1 (Carmichael Numbers). The composite integer n is a Carmichael number if $x^{(n-1)} \equiv 1 \pmod{n}$ for all $x \in \mathbb{Z}_n^*$

Theorem 1.1 is the base of a very simple primality test, referred to as Fermat's test.

Algorithm 1 (Fermat's Test). (1) Choose $x \in 1, \ldots, (n-1)$ uniformly at random.
(2) If $x^{(n-1)} \equiv 1 \pmod{n}$ output **prime** else output **composite**.

For all prime numbers n, Theorem 1.1 guarantees that Fermat's test, Algorithm 1, will correctly output that n is prime. However, a flaw of this algorithm is that it will fail if the input n is a Carmichael number (Definition 1), to which it will still output that n is prime. Therefore, Fermat's primality test is a probabilistic algorithm, and only has applications in certain scenarios in which a probabilistic result is sufficient. The time complexity of this algorithm is $O(K * log(n))$ where K is the number of iterations, and n is the number being tested. This is drawn from the following: $log(n)$ is used for the time complexity for $a^(n-1)$ and the algorithm is repeated K times.

1.2. **Miller-Rabin Primality Test.**

Definition 2 (Witness). [Wei23b] A witness refers to a number which satisfies a given criterion, in this section it will be used as a determinant whether a number n is composite or prime, as a result of its number theoretic properties.

Theorem 1.2 (Monier-Rabin)**.** *[Mon80, Rab80] Let N be a composite integer which is odd. The probability that a random integer a for which $a \in [1, N-1]$ us a witness for N is at least* 0.75

For Theorem 1.2, the proofs of Monier and Rabin are too extensive for the purpose of the paper, so we will use a proof inspired by one written by Andrew Sutherland.

Proof. [Sut17] Consider n be an odd composite number where $n = 2^s t + 1$ with t odd, and let $n = q_1 \cdots q_r$ represent n factorized into powers of distinct primes. For this, we bear in mind that t is required to be coprime to all the q_j. Here we choose $a \in [1, (n-1)]$ at random, and set $b := a^t$. Therefore, if a is not a witness, then it must be true that either $b \equiv 1 \mod n$, for which $b \equiv 1 \mod q_j$ for all q_j, or it must be true that $b^{2^i} \equiv -1 \mod n$ for i

such that $0 \leq i < s$. In the second possibility, $b^{2^i} \equiv -1 \mod q_j$ for all q_j. If we set $i := -1$ in the first possibility, then $b \mod q_j$ is an element of order 2^{i+1} in the 2-Sylow subgroup S_j of $(\mathbb{Z}/p^k\mathbb{Z})^\times$ for $1 \leq j \leq r$. It can then be shown that the probability that every $b_j := b \mod q_j$ lies in S_j which also has order 2^i is at most $1/4$ through the following cases.

Case 1: N is not square-free. Then some q_j will be equal to p^k with $k > 1$. Due to p being odd, the group $\left(\mathrm{Z}/p^k Z\right)^\times$is cyclic with an order of $\phi\left(p^k\right) = p^{k-1}(p-1)$, and t is coprime to p, this means that the probability that b_j is within S_j is at most $1/p^{k-1}$, which is less than $1/4$ whilst $p^k > 0$. If $q_j = p^k = 9$ then $t \equiv \pm 1 \mod 6$ and only 2 of the 8 nonzero values of $a \mod 9$ lead to $b_j \in S_j$.

Case 2: This case considers N as a product of $r \geq 3$ distinct primes q_j. Each 2-Sylow subgroup S_j is a cyclic with an order of $2^{k,}$, for some $k_j > 1$, and, at most, half the elements in S_j can have any specific order. If we were to then assume that each b_j is within G_j then these are uniformly distributed due to the fact that t odd, and the probability they all have the same order is at most $1/4$.[3]

Case 3: The final case as done by Sutherland [Sut17], considers the chance that $N = q_1 q_2$ is a product of 2 distinct primes. We first begin with $q_1 = 2^{s_1} t_1 + 1$, and $q_2 = 2^{s_2} t_2 + 1$, with t_1 and t_2 having the quality of being odd. We then define the random variable X_j as -1 if b_i does not lie in S_i and otherwise we put $X_p = i$, where b_j has an order of 2^i in S_j. We then must show that

$$\Pr\left[X_1 = X_2 \geq 0\right] \leq 1/4.$$

To do this, assume $s_1 > s_2$. Since half the elements which are in S_1 have an order of $2^{s_1} > 2^{s_2}$, $\Pr\left[0 \leq X_1 \leq s_2\right] \leq 1/2$, and $\Pr\left[X_2 = X_1 \mid 0 \leq X_1 \leq s_2\right] \leq 1/2$; therefore $\Pr\left[X_1 = X_2 \geq 0\right] \leq 1/4$. We then assume that $s_2 = s_2$ so that we have

$$2^s t = N - 1 = q_1 q_2 - 1 = (q_1 - 1)(q_2 - 1) + (q_1 - 1) + (q_2 - 1) = 2^s t_1 t_2 + 2^{s_1} t_1 + 2^{s_2} t_2,$$

therefore if t is divisible by t_1 then t_1 also divides t_2, and the reverse. If t_1 and t_2 both divide t, then $t_1 = t_2$ and $q_1 = q_2$, which gives us a contradiction. So assume $t_1 \nmid t$. This means that $t_1 \neq 1$ must also be divisible by an odd prime $\ell \geq 3$ that does not divide t. This leads to:

$$\Pr\left[X_1 \geq 0\right] \leq 1/3,$$

and as we also have

$$\Pr\left[X_1 = X_2 \mid \bar{X}_1 \geq 0\right] \leq 1/2,$$

we end with

$$\Pr\left[X_1 = X_2 \geq 0\right] \leq 1/6 < 1/4.$$

■

Theorem 1.2, above, suggests that for a composite N, if we choose, say, 200 randomly selected integers $a \in [1, N-1]$, then it is very likely that we will find a witness for N. Theorem 1.2 also suggests that, if N is prime, then we shall not find a witness. Unless we try more than a quarter of all $a \in [1, N-1]$, this will not prove that N is prime, however it will support the possibility that is is prime.

The Miller-Rabin primality test is inspired by Theorem 1.2. This is another probabilistic primality test, depicted below.

Algorithm 2 (Miller-Rabin Test). [Rab80]

(1) Choose a random integer $a \in [1, N-1]$.

(2) Write $N = 2^s t + 1$, with t odd, and compute $b = a^t \bmod N$. If $b \equiv \pm 1 \bmod N$, return true (a is not a witness, N is possibly prime).
(3) For i from 1 to $s - 1$:
    Set $b \leftarrow b^2 \bmod N$.
    If $b \equiv -1 \bmod N$, return true (a is not a witness, N is possibly be prime).
(4) Return false (This is because if a is a witness this would mean that N is definitely not prime).

The Miller-Rabin test, Algorithm 2, has a time complexity of $O(K \times \log^3(n))$, where n is the number being tested for primality whilst K refers to the number of iterations of Algorithm 2 that are performed. This deems the algorithm highly efficient, and can be performed with many repeated iterations. In the case that all iterations return true, it can be concluded that n is probably prime. This algorithm is a probabilistic algorithm because if n is composite, the process will have the correct output of "false" with a probability of 75%, however if n is prime, the algorithm will always have the correct output of "true".

1.3. **Solovay-Strassen Primality test.**

Definition 3 (Relatively prime). Two integers a and b are referred to as relatively prime to each other if:

$$gcd(a, b) = 1.$$

Definition 4 (Legendre Symbol). We let a be an integer, and P be an odd prime with the greatest common factor of a and P being 1, then the Legendre symbol a and P is defined by:

$$\left(\frac{\mathbf{a}}{\mathbf{p}}\right) = \begin{cases} \mathbf{1};\ \text{If } a \text{ is quadratic residue of } P. \\ -\mathbf{1};\ \text{If } a \text{ is non quadratic residue of } P. \\ \mathbf{0};\ \text{If } a \equiv 0 (\bmod P) \end{cases}$$

Legendre symbol is a number theoretic function with values of $+1, -1$ and 0.

Definition 5 (Jacobi Symbol). [Wei23a] Jacobi symbol is a generalized Legendre symbol.

For any integer a with any positive integer n the Jacobi symbol for the two is defined as the product of the Legendre symbols, which correspond to the prime factor n. This is depicted below.

$$\left(\frac{a}{n}\right) = \left(\frac{a}{\mathrm{P}_1}\right) \cdot \left(\frac{a}{\mathrm{P}_2}\right) \cdot \left(\frac{a}{\mathrm{P}_3}\right) \cdot \cdots\cdots \left(\frac{a}{\mathrm{P}_k}\right)$$

Definition 6 (Euler Witness). [Con16] Suppose $n > 1$ is an odd integer, then an integer $a \in \{1, \ldots, n-1\}$ such that either

$$\text{(i) } (a, n) > 1$$

or

$$\text{(ii) } a, n) = 1 \text{ and } a^{(n-1)/2} \not\equiv \left(\tfrac{a}{n}\right) \bmod n,$$

is what we refer to as an Euler witness for n. All other integers in $\{1, \ldots, n-1\}$ are what we consider Euler nonwitnesses for n.

If n is an odd prime then neither (i) or (ii) hold for any $a \in \mathbb{Z}_n^*$ (from 1 to $(n-1)$), this means that n has no Euler witnesses. Therefore, if there is a single Euler witness for n, this would prove that n is composite. The following theorem is written on this basis.

Theorem 1.3 (Solovay-Strassen)**.** *[Con16] Consider an odd integer, which is composite: n. An integer a exists in $\mathbb{Z}_n^*$ which gives us*

$$(a, n) = 1$$

and

$$a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \bmod n.$$

Definition 7 (Square-free Integer)*.* We consider an integer n square-free if the number's prime decomposition does not have any repeated factors. If an integer is not divisible by any square number, other than 1, we refer to it as a square-free integer.

Theorem 1.3 has an essential facet. For a specific a which is in $\mathbb{Z}_n^*$, that is relatively prime to n, the following equation fails:

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

Proof. Within this proof, we must consider two separate cases. The first case we consider is that n is square-free and the second case is that n has a repeated prime factor.

To begin, we assume n is both composite as well as square-free, therefore $n = p_1 p_2 \cdots p_r$ where $r \geq 2$ and all p_i 's are odd primes, separate from each other. There exists $b \in \mathbf{Z}$ for which $\left(\frac{b}{p_1}\right) = -1$. Certain $a \in \{1, \ldots, n-1\}$ satisfy

$$a \equiv b \bmod p_1, \quad a \equiv 1 \bmod p_2 \cdots p_r,$$

according to the Chinese remainder theorem.

Given that $b \not\equiv 1 \bmod p$, we can state $a \neq 1$. This means that a is relatively prime to p_1 and therefore a is also relatively prime to $p_2 \cdots p_r$, thus $(a, n) = 1$.

Furthermore, we can say

$$\left(\frac{a}{p_1}\right) = \left(\frac{b}{p_1}\right) = -1$$

as well as:

$$\left(\frac{a}{p_i}\right) = \left(\frac{1}{p_i}\right) = 1$$

for $i > 1$, thus

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\cdots\left(\frac{a}{p_r}\right) = \left(\frac{a}{p_1}\right) = -1.$$

Suppose that

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

this would then result with

$$a^{(n-1)/2} \equiv -1 \bmod n.$$

Due to p_2 being able to divide n, it is possible for us to contract $a^{(n-1)/2} \equiv -1 \bmod n$ to modulus p_2, which leads us to the following:

$$1 \equiv -1 \bmod p_2,$$

due to the congruence $a \equiv 1 \bmod p_2$. This is a contradiction to our assumption due to 2 being less than the modulus p_2 . Continuing our proof, we complete the second case, where

n has a repeated prime factor, which we will refer to as p.[5] With this assumption, $n = p^k m$ for which $(p, m) = 1$ and $k \geq 2$. Chinese remainder theorem can be used to show that there exists an $a \in \{1, \ldots, n-1\}$ which is able to satisfy

$$a \equiv 1 + p \bmod p^2, \quad a \equiv 1 \bmod m.$$

This leads on to give the following:

$$a \neq 1,$$

and

$$a \text{ is not divisible by p,}$$

and

$$(a, m) = 1$$

Therefore we have $(a, n) = 1$.

If $a^{(n-1)/2}$ is congruent with $\left(\frac{a}{n}\right) \bmod n$ then by squaring this will present us with $a^{n-1} \equiv 1 \bmod n$. The next part of this proof will demonstrate that this is not possible.

To do this, we begin by reducing the congruence to modulus p^2, this is because it is a factor of the number being considered, n. This results in the following congruence:

$$a^{n-1} \equiv 1 \bmod p^2.$$

We can then bring $(1 + p)^{n-1} \equiv 1 \bmod p^2$ due to $a \equiv 1 + p \bmod p^2$.

$$(1 + p)^{n-1} \equiv 1 + (n - 1)p \bmod p^2,$$

by the Binomial theorem.

Therefore we can get to $1 + (n - 1)p \equiv 1 \bmod p^2$.

Next, we minus 1 from each side to get:

$$(n - 1)p \equiv 0 \bmod p^2,$$

thus $n-1 \equiv 0 \bmod p$. However, since n is a multiple of p, we are left with a contradiction. ■

The Solovay-Strassen primality test can also be represented as an algorithm, shown below:

Algorithm 3 (Solovay-strassen Algorithm). To test a number n:

(1) Pick a, where a is a random number > than 1 but also < n(1 < a < n).
(2) See if the following is true: $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2}(\text{modn})$.
(3) If the above equation is true, output "probably prime." If not, output "composite."

The Solovay-Strassen test in Algorithm 3 and Theorem 1.3 is a probabilistic test, however it can be made into a deterministic primality test if the truth of an unsolved problem, the Generalized Reimann Hypothesis, is assumed, as the Generalized Reimann Hypothesis implies that any odd composite positive integer n has an Euler witness that is at most $2(\log n)^2$ [Con16]. However, although many mathematicians believe the generalizations of the Reimann hypothesis to be true, it remains unsolved.

[5]The proof in the original paper of Solovay and Strassen did not cover the case where n is a perfect square, this was conducted later in [SS78].

1.4. **AKS Primality Test.** The Agrawal-Kayal-Saxena primality test is the first simultaneously general, polynomial-time, deterministic, and unconditionally correct primality test, and it is shown below in Algorithm 4. We will look into the Agrawal-Kayal-Saxena (AKS) primality test in more depth in the next section, which is entirely dedicated to the original theorem and Agrawal, Kayal and Saxena's subsequent proofs.

But, preceding the original theorem and the corresponding proof, we observe the algorithm for the AKS primality test.

Algorithm 4 (AKS Primality Test). [SS13] For $n \in \mathrm{N}, n > 1$

(1) If $b > 1$ and $n = a^b$ for $a \in \mathrm{N}$, return "composite."
(2) The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
(3) If it is true that $1 < \mathrm{GCD}(a, n) < n$ for every $a \leq r$, return "composite."
(4) If it remains true that $n \leq r$, return "prime."
(5) For $a = 1$ to $\lfloor 2\sqrt{\varphi(r)}\log n \rfloor$ do: if it is true that $\left((x+a)^n \neq x^n + a \left(\mathrm{mod} x^r - 1, n\right)\right)$, return "composite."
(6) Return "prime."

This AKS algorithm is executed in $\tilde{\mathrm{O}}\left(\log^{10.5} n\right)$ time, and therefore as n grows larger, the increase in time will be proportional to $(\log n)^{10.5}$. This polynomial time function is not as quick as the probabilistic tests in earlier sections, such as Fermat's test in Algorithm 1, however it is advantageous as it is fully deterministic, unlike Fermat's test which fails for Carmichael numbers (defined in Definition 1).

We will now go through the steps of the Agrawal-Kaya-Saxena primality test as shown in Algorithm 4. Step 1 tests if n is a perfect power, which, if true, would instantly determine n as composite. Step 2 involves Fermat's Little theorem, Theorem 1.1, this point is where r is found as this value binds the following steps in the algorithm. Step 2 finds the smallest r such that the order of $n \mod r$ is greater than $4log^2n$. Step 3 then determines the greatest common factor for all values less than or equal to r, and n. This makes sure that all values r and below are relatively prime, defined by Definition 3, to r. Step 4 is set to output prime if n is less than or equal to r. The fifth step is the step which takes the most time,[6] this step tests a relation for all values of a from 1 to $\lfloor 2\sqrt{\varphi(r)}\log n \rfloor$, if the tested relation holds true, then the number n is determined prime.

2. Agrawal-Kayal-Saxena Primality Test Original Theorem and Proofs

Within the previous section, in the final subsection, the Agrawal-Kayal-Saxena primality test was explained briefly, however this section did not refer to the original theorem written by Manindra Agrawal, Neraj Kayal and Nitin Saxena. The AKS primality test is one which is highly important in the field of mathematics concerning primality tests. This is the first deterministic algorithm for primality testing that we have examined in this paper, and deterministic algorithms differ from probabilistic algorithms as they ensure that the output is always accurate. This allows the AKS primality test to provide a definitive answer about the primality of a given number n, whilst probabilistic tests provide results which may be incorrect with a given probability. The AKS primality test is also very prominent for other reasons:

[6] [SS13]

- Polynomial Time Complexity: Unlike several other primality testing algorithms, such as the Miller-Rabin algorithm (shown in Algorithm 2), the AKS primality test runs in polynomial time. This essentially means that the time that the algorithm takes to determine the primality of the number is proportional to a polynomial funcion of the number of digits in the input. This polynomial time complexity makes the AKS algorithm very efficient in comparison to other primality tests.
- Theoretical Breakthrough: The AKS primality test represented a substantial shift in computational complexity, as it proved that primality testing can be performed in polynomial time, which contradicted the previous belief that primality testing was inherently a computationally difficult problem. The development of AKS led to several advancements in cryptography.
- Cryptographic Applications: Cryptographic algorithms such as Rivest-Shamir-Adleman require the use of prime numbers, these algorithms are used for secure communication, digital signatures and encryption. The Agrawal-Kayal-Saxena primality test provided a reliable and efficient method to confirm the primality of very large numbers that are used in cryptography, this enhanced the reliability of these systems.

We will now explore the original theorem and lemmas.

Algorithm 5 (Agrawal-Kayal-Saxena Primality Test). [SS13] For $\mathrm{n} \in \mathrm{N}, \mathrm{n} > 1$

(1) If $\mathrm{b} > 1$ and $\mathrm{n} = \mathrm{a}^{\mathrm{b}}$ for $\mathrm{a} \in \mathrm{N}$, return "composite."
(2) The smallest r for which $O_r(n) > 4\log^2 n$ is true, must be found.
(3) If it is true that $1 < \mathrm{GCD}(\mathrm{a}, \mathrm{n}) < \mathrm{n}$ for every $\mathrm{a} \leq \mathrm{r}$, return "composite."
(4) If it remains true that $\mathrm{n} \leq \mathrm{r}$, return "prime."
(5) For $\mathrm{a} = 1$ to $\lfloor 2\sqrt{\varphi(r)}\log n \rfloor$ do: if it is true that $\left((x+a)^n \neq x^n + a \left(\mathrm{mod} x^r - 1, n\right)\right)$, return "composite."
(6) Return "prime."

Theorem 2.1 (Theorem 4.1 in the original paper). *[AKS04] Algorithm 5 returns "prime" if and only if n is prime.*

To prove this theorem, we will break Theorem 2.1 down into several separate lemmas, which we will then prove individually.

Lemma 2.2 (Lemma 2.1 in the original paper). *[AKS04] Assume $a \in \mathcal{Z}, n \in \mathcal{N}, n \geq 2$, and $(a, n) = 1$. This then means that n is prime if and only if*

$$(X+a)^n = X^n + a(\mathrm{mod} n).$$

Proof. For $0 < i < n$, the coefficient of x^i in $((X+a)^n - (X^n + a))$ can also be represented by $\begin{pmatrix} n \\ i \end{pmatrix} a^{n-i}$.

Assume that n is prime. This means that $\begin{pmatrix} n \\ i \end{pmatrix} = 0 \pmod{n}$ leading to the conclusion that all of the coefficients are 0.

Then we assume that n is not prime. Consider a prime q which is a factor of n and let $q^k \| n$. This would bring us $\begin{pmatrix} n \\ q \end{pmatrix}$ is not divisible by q^k and q^k is coprime to a^{n-q} and therefore the coefficient of X^q is not $0 \ (\mathrm{mod} n)$. This then brings us to the fact that the following:

$$\left((X+a)^n - (X^n + a)\right),$$

is not identically zero over Z_n.

■

Lemma 2.3 (Lemma 4.2 in the original paper)**.** *[AKS04] In the case that n is prime, the output of the algorithm is "prime."*

Proof. In the case that n is actually prime then both step 1 and step 3 cannot possibly output "composite." Lemma 2.2 shows us that it is not possible for the loop to output "composite." This means that Algorithm 5 is able to correctly show that n is prime in either step 4 or 6. ■

The opposite of Lemma 2.3 demands a more lengthy proof. If in step 4 the output of the algorithm is "prime," then n cannot be composite, this is because otherwise the third step would have been able to deduce a nontrivial factor of n. This leaves the remaining case of when the output confirming primality is from step 6 in Algorithm 5. As in the work by the original authors, we shall also assume this to be the case.

The main steps left of the algorithm are 2 and 5.

The second step is utilized in order to find a suitable r whilst the fifth step confirms an equation for a range of a's. Continuing the proof, we will set a bound for the magnitude of the suitable r, but first a preliminary lemma must be proved.

Lemma 2.4 (Lemma 3.1 in the original paper)**.** *[AKS04] Let $LCM(m)$ represent*

$$\operatorname{lcm}(1, 2, 3, \ldots, m).$$

For $m \geq 7$:

$$LCM(m) \geq 2^m$$

In the initial proof of Theorem 2.1, Lemma 2.4 was not proven, however, for the comprehensiveness of this paper, we shall give a simple proof.

Proof. We begin by considering for $1 \leq n \leq m$:

$$I_{n,m} = \int_0^1 x^{n-1}(1-x)^{m-n} = \sum_{r=0}^{m-n} \frac{a_r}{n+r} \quad \text{for some } a_i \in \mathbb{Z}$$

From here on,

$$\ell_m = \operatorname{lcm}(1, 2, \ldots, n).$$

Note

$$\ell_m I_{n,m} \in \mathbb{Z}$$

for

$$1 \leq n \leq n.$$

This makes evident the following:

$$I_{n,m} = \frac{1}{n \begin{pmatrix} m \\ n \end{pmatrix}}.$$

This is identified through either reduction formulae or integration by parts. Following this, we are left with:

$$n \begin{pmatrix} m \\ n \end{pmatrix} \mid \ell_m \quad \forall 1 \leq n \leq m$$

Typically,

$$m\begin{pmatrix} 2m \\ m \end{pmatrix} \mid \ell_{2m} \quad \text{and} \quad (2m+1)\begin{pmatrix} 2m \\ m \end{pmatrix} = (m+1)\begin{pmatrix} 2m+1 \\ m+1 \end{pmatrix} \mid \ell_{2m+1}.$$

Now since $\ell_{2m} \mid \ell_{2m+1}$ we have:

$$m(2m+1)\begin{pmatrix} 2m \\ m \end{pmatrix} \mid \ell_{2m+1}$$

$$\Longrightarrow \ell_{2m+1} \geq m(2m+1)\begin{pmatrix} 2m \\ m \end{pmatrix} \geq m \cdot 4^m \geq 2^{2m+2} \quad \text{for} \quad m \geq 4.$$

The above penultimate inequality remains true due to $\begin{pmatrix} 2m \\ m \end{pmatrix}$ being the greatest co-efficient if we were to expand out $(1+x)^{2m}$. Thus, $(1+1)^{2m} \leq \begin{pmatrix} 2m \\ m \end{pmatrix}(2m+1)$. Also,

$$\ell_{2m+2} \geq \ell_{2m+1} \geq 2^{2n+2} \quad \text{for} \quad m \geq 4.$$

Thus completing the proof of

$$\ell_m \geq 2^m \quad \text{for} \quad m \geq 9.$$

■

Lemma 2.5 (Lemma 4.3 in the original paper)**.** *[AKS04] There is a case of*

$$r \leq \max\left\{3, \left\lceil \log^5 n \right\rceil\right\}$$

in which

$$o_r(n) > \log^2 n.$$

Proof. Lemma 2.5 can be readily proven true for the case of $n = 2$, due to the ability of $r = 3$ to satisfy every required condition. To conduct the proof, we will continue with the assumption that $n > 2$. This would mean that $\left\lceil \log^5 n \right\rceil > 10$, in which case Lemma 2.4 can be utilized. We shall now continue by considering $r_1, r_2, \ldots, r_t$ to be all of the numbers for which n is divisible by either either r_i or $o_{r_i}(n) \leq \log^2 n$. These numbers must all be able to divide the following:

$$n \cdot \prod_{i=1}^{\left\lfloor \log^2 n \right\rfloor} \left(n^i - 1\right) < n^{\log^4 n} \leq 2^{\log^5 n}.$$

The lcm of the first $\left\lceil \log^5 n \right\rceil$, according to Lemma 2.4, at the very least is $2^{\left\lceil \log^5 n \right\rceil}$. Thus, it must be true for some $s \leq \left\lceil \log^5 n \right\rceil$, that $s \notin \{r_1, r_2, \ldots, r_t\}$. In the case that $(s, n) = 1$, it can be seen that

$$o_s(n) > \log^2 n,$$

and the case would be complete.

In the case $(s, n) > 1$, due to the fact that $(s, n) \in \{r_1, r_2, \ldots, r_t\}$, and that n is not divisible by s, $r = \frac{s}{(s,n)} \notin \{r_1, r_2, \ldots, r_t\}$, therefore:

$$o_r(n) > \log^2 n.$$

■

There must be a prime factor p of n, due to $o_\tau(n) > 1$, for which $o_\tau(p) > 1$. The third step or the fourth step in Algorithm 5 would decide about the primality of n unless:

$$p > r.$$

Furthermore, the third or the fourth step would correctly identify n unless:

$$(n, r) = 1.$$

From $(n, r) = 1$, we can understand that

$$p, n \in Z_r^*.$$

For the purpose of this proof, both p and r shall remain fixed until the next section. To simplify further parts of this section, we shall also set

$$\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor.$$

The fifth step of the Agrawal-Kayal-Saxena primality test, given in Algorithm 5, is what confirms ℓ equations. As we are maintaining a similar format and sequence to Agrawal, Kayal and Saxena's work, we have assumed that the case we are going to consider is the case when the algorithm outputs "prime" in the sixth step. Therefore, the algorithm does not return "composite" in the fifth step, thus we can deduce the following:

$$(X + a)^n = X^n + a \, (\mathrm{mod} X^r - 1, n)$$

in the case of every a, given that $0 \leq a \leq \ell$. The following equation can then be deduced:

[1]

$$(X + a)^n = X^n + a \, (\mathrm{mod} X^r - 1, p)$$

with the range $0 \leq a \leq \ell$.

Due to Lemma 2.2, we can also deduce:

[2]

$$(X + a)^p = X^p + a \, (\mathrm{mod} X^r - 1, p)$$

with the same range $0 \leq a \leq \ell$.

From [1] and [2] above, we have:

[3]

$$(X + a)^{\frac{n}{p}} = X^{\frac{n}{p}} + a \, (\mathrm{mod} X^r - 1, p)$$

with the same range $0 \leq a \leq \ell$. Through the equation above, it can be seen that both $\frac{n}{p}$ and n behave similar to prime p. There has been a name given to this property:

Definition 8. [AKS04] Given a number m that is in $\mathcal{N}$ and a polynomial $f(X)$, it is said that m is *introspective* for $f(X)$ provided that

$$[f(X)]^m = f\left(X^m\right) (\mathrm{mod} X^r - 1, p).$$

We can see from equations [2] and [3] above, that both p as well as $\frac{n}{p}$ are introspective, when $0 \leq a \leq \ell$, for $X + a$.

We will use the lemma below to demonstrate that introspective numbers are closed under multiplication:

Lemma 2.6 (Lemma 4.5 in the original paper)**.** *[AKS04] $m \cdot m'$ is an introspective number for $f(X)$ if the numbers m and m' are introspective for $f(X)$.*

Proof. Considering that m is introspective for $f(X)$, it can be seen that:

$$[f(X)]^{m \cdot m'} = [f\left(X^m\right)]^{m'} \left(\mathrm{mod} X^r - 1, p\right).$$

Furthermore, we must also consider that m' has the same introspective property for $f(X)$. By replacing X with X^m in the introspection equation for m', we are able to reach:

$$\begin{aligned}[f\left(X^m\right)]^{m'} &= f\left(X^{m \cdot m'}\right) \left(\mathrm{mod} X^{m \cdot r} - 1, p\right) \\ &= f\left(X^{m \cdot m'}\right) \left(\mathrm{mod} X^r - 1, p\right) (\text{ since } X^r - 1 \text{ divides } X^{m \cdot r} - 1).\end{aligned}$$

Combining the 2 equations above, we finally reach:

$$[f(X)]^{m \cdot m'} = f\left(X^{m \cdot m'}\right) \left(\mathrm{mod} X^r - 1, p\right).$$

■

We can use this to show that, given a number m, the set of polynomials such that m is introspective is closed under multiplication.

Lemma 2.7 (Lemma 4.6 in original paper)**.** *[AKS04] It must be true that m is introspective for $f(X) \cdot g(X)$, on the condition that m is introspective for both $f(X)$ and $g(X)$.*

Proof. This will give us:

$$\begin{aligned}[f(X) \cdot g(X)]^m &= [f(X)]^m \cdot [g(X)]^m \\ &= f\left(X^m\right) \cdot g\left(X^m\right) \left(\mathrm{mod} X^r - 1, p\right).\end{aligned}$$

■

Both Lemma 2.7 and Lemma 2.6 can be used to demonstrate that all numbers within the below set

$$I = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \mid i, j \geq 0 \right\},$$

are introspective for all of the polynomials that belong to the set

$$P = \left\{ \prod_{a=0}^{\ell} (X + a)^{e_a} \mid e_a \geq 0 \right\}.$$

In order to carry on parallel to the original authors' proof,[7] we will create and define 2 groups that are required to complete the proof. These are derived from the above sets.

The first group defined in the paper by Agrawal, Kayal and Saxena is "the set of all residues of numbers in $I \mod r$." We have seen that:

$$(n, r) = (p, r) = 1,$$

and this can be used to determine that the first group we have defined is a subgroup of Z_r^*. We will refer to this group as G_1, and we will define v as $|G_1| = v$. We generate the group G by p modulo r and n, and because we know that

$$\mathrm{o}_r(n) > \log^2 n,$$

we can see

$$v > \log^2 n.$$

[7] [AKS04]

We will now define the next group as the set of all residues of polynomials in P modulo $h(X)$ and p. However, for us to define this group, we must give meaning to $h(X)$, and this is done by recalling some essential principles of cyclotomic polynomials over finite fields. Firstly, we will refer to the r^{th} cyclotomic polynomial over F_p using $Q_r(X)$. $X^r - 1$ is divisible by this polynomial, and $Q_r(X)$ also factors into irreducible factors with the degree $o_r(p)$[8]. We will now be able to define $h(X)$, and we will set this to be one such irreducible factor. The degree of $h(X)$ is larger than 1 due to $o_r(p) > 1$. This group is generated by the following elements:

$$X, X+1, X+2, \ldots, X+\ell$$

in the field

$$F = F_p[X]/(h(X)).$$

We will refer to this group as G_2. This G_2 is also a subgroup of the multiplicative group of F.

To progress with the proof, we must now prove, for the group G_2, a lower bound. The lower bound which we shall prove is a refined version of the bound which was presented in Agrawal, Kayal and Saxena's earlier papers.

Lemma 2.8 (Lemma 4.7 in original paper)**.** *[AKS04]*

$$|\mathcal{G}| \geq \binom{v+\ell}{v-1}.$$

Proof. We know that X is a primitive r^{th} root of unity within F, due to $h(X)$ being a factor of the cyclotomic polynomial $Q_r(X)$.

It can then be demonstrated that we can map two polynomials, which are distinct, of degree less than v in P to separate elements in the earlier defined group $\mathcal{G}$. From this point, we move forward by using $g(X)$ and $f(X)$ as two of these polynomials in P. We assume that $g(X)$ is equal to $f(X)$, in field F. We will also use m such that $m \in I$. Thus, in the field F, we also have:

$$[g(X)]^m = [f(X)]^m.$$

As $h(X)$ divides $X^r - 1$ and m is also introspective for g as well as f, we can state that in F:

$$g\left(X^m\right) = f\left(X^m\right).$$

The above suggests that, for each $m \in G$, X^m is a root of the polynomial $Q(Y) = f(Y) - g(Y)$. We can see that every such X^m is a primitive r^{th} root of unity, as we know that $(m, r) = 1$. This would then mean that there are $|G| = v$ distinct roots of $Q(Y)$ in F, but v is greater than the degree of $Q(Y)$ by the choice of g and f. As this is a contradiction we can state that, in F,

$$g(X) \neq f(X).$$

We then follow along with the original authors' proof to show that the following elements are distinct in F:

$$X, X+1, X+2, \ldots, X+\ell.$$

This is done as we can conclude that, because$\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor < \sqrt{r} \log n < r$ and $p > r$, $i \neq j$ in F_p for $1 \leq i \neq j \leq \ell$. Furthermore, $X + a \neq 0$ in F for every $a, 0 \leq a \leq \ell$, as we can know the degree of h is larger than 1. This means that at least $\ell + 1$ distinct polynomials

[8] [LN94]

exist that have degree one, in $\mathcal{G}$. Following this we can conclude our proof that at least $\begin{pmatrix} v+\ell \\ v-1 \end{pmatrix}$ with a degree less than v exist in $\mathcal{G}$. ■

We are also able to set an upper bound for the size of $\mathcal{G}$, which must be considered in the case where n is not actually a power of p.

Lemma 2.9 (Lemma 4.8 in original paper)**.** *[AKS04] $|\mathcal{G}| \leq n^{\sqrt{v}}$, considering that n is not a power of p.*

Proof. As in the original authors' proof, we will first look at the below subset of I,

$$\hat{I} = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor \right\}.$$

The set $\hat{I}$ should contain $(\lfloor \sqrt{v} \rfloor + 1)^2 > v$ distinct numbers, in the case we are considering. It must be true that a minimum of two numbers in $\hat{I}$ should be equal modulo r, as we know that $|G| = t$. We will denote these with m_1 and m_2 with $m_1 < m_2$. This would lead us to the following:

$$X^{m_1} = X^{m_2} \left(\mathrm{mod} X^r - 1\right).$$

From here, maintaining the essence of Agrawal, Kayal and Saxena's proof, we will set $f(X) \in P$ to lead us to

$$\begin{aligned} [f(X)]^{m_2} &= f\left(X^{m_2}\right) \left(\mathrm{mod} X^r - 1, p\right) \\ &= f\left(X^{m_1}\right) \left(\mathrm{mod} X^r - 1, p\right) \\ &= [f(X)]^{m_1} \left(\mathrm{mod} X^r - 1, p\right). \end{aligned}$$

Which would then suggest that in the field F

$$[f(X)]^{m_1} = [f(X)]^{m_2}.$$

This means that $f(X) \in \mathcal{G}$ is a root of the polynomial $Q'(Y) = Y^{m_2} - Y^{m_1}$ in the field F.[9] We can also state that the polynomial $Q'(Y)$ has a minimum $|\mathcal{G}|$ distinct roots in F, as we know that $f(X)$ is an arbitrtrary element of $\mathcal{G}$. This means that the degree of $Q'(Y)$ is

$$m_2 \leq \left(\frac{n}{p} \cdot p\right)^{\lfloor \sqrt{v} \rfloor} \leq n^{\sqrt{v}}.$$

Finally giving us:

$$|\mathcal{G}| \leq n^{\sqrt{t}}.$$

■

Giving that we have bounds for the size of $\mathcal{G}$, we can finally finish our proof to establish the AKS algorithm given in Algorithm 5. Our final lemma is below.

Lemma 2.10 (Lemma 4.9 in original paper)**.** *[AKS04] Algorithm 5 outputs "prime" if and only if n is prime.*

[9] [KSS02]

Proof. We will begin by assuming that Algorithm 5 outputs "prime." By Lemma 2.8 it can be suggested that for $v = |G|$ and $\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor$:

$$\begin{aligned}
|\mathcal{G}| &\geq \begin{pmatrix} v+\ell \\ v-1 \end{pmatrix} \\
&\geq \begin{pmatrix} \ell+1+\lfloor \sqrt{v} \log n \rfloor \\ \lfloor \sqrt{v} \log n \rfloor \end{pmatrix} \quad (\text{ since } v > \sqrt{v} \log n) \\
&\geq \begin{pmatrix} 2\lfloor \sqrt{v} \log n \rfloor + 1 \\ \lfloor \sqrt{v} \log n \rfloor \end{pmatrix} \quad (\text{ since } \ell = \lfloor \sqrt{\phi(r)} \log n \rfloor \geq \lfloor \sqrt{v} \log n \rfloor) \\
&> 2^{\lfloor \sqrt{v} \log n \rfloor + 1} \quad \left(\text{since } \lfloor \sqrt{v} \log n \rfloor > \left\lfloor \log^2 n \right\rfloor \geq 1\right) \\
&\geq n^{\sqrt{v}}.
\end{aligned}$$

We then consider Lemma 2.9 and use this to state that if n is not a power of p:

$$|\mathcal{G}| \leq n^{\sqrt{t}}.$$

This would mean that $n = p^k$ for a $k > 0$, and if $k > 1$ then this would mean that Algorithm 5 would output "composite" in the very first step. This completes our proof, as this means that $n = p$. ■

3. Factoring algorithms

3.1. **Fermat's Factorization.** Fermat's factorization algorithm is a very basic process, where to factor a number N, we express the number as a difference of two squares. Essentially, it is the principle that if we can interpret N as $a^2 - b^2$, where both a and b are positive integers, we are able to factor N into $(a+b)(a-b)$. This factorization is non-trivial if $a - b > 1$.

In order to be made into an algorithm, we use values of a from $\lceil \sqrt{N} \rceil$ to N as in the following:

$$a = \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil + 1, \lceil \sqrt{N} \rceil + 2, \ldots.$$

We then test to see if the value of $a^2 - N$ is square. In the chance that the determined value is square, b^2, then we will get $N = a^2 - b^2 = (a+b)(a-b)$. Fermat factorization is easier to use for values of N that have a factor which is near to $\sqrt{N}$ or are odd and the product of two close integers.

The time complexity of the Fermat factorization algorithm is exponential, but the general basis of most known algorithms which are below exponential complexity, such as the quadratic sieve and number field sieve algorithms, is dependent on Fermat factorization.

Algorithm 6 (Fermat Factorization). [Sah11] This algorithm is for an input of N to generate the factors of N.

(1) for a from $\lceil \sqrt{N} \rceil$ to N
(2) $bsqr = a \times a - N$
(3) if isSquare($bsqr$) then
    $b = \sqrt{bsqr}$
    $s = a - b$
    $v = a + b$
    if $s \neq 1$ and $s \neq N$ then output s, v

endif
(4) endif
(5) endfor

The "isSquare" function takes a square root, and then rounds the value received to an integer, then squares the outcome of that, and then checks if the result is the number that was first inputted.

3.2. **Trial Division.** Trial division is often regarded as factorization at its simplest. It is a fundamental algorithm which forms the basis of algorithms that we will consider.

This factorization algorithm simply divides N by each odd integer up until $\lfloor\sqrt{N}\rfloor$ and therefore if the number N being considered has a small prime factor p, then we are able to deduce the value of p very easily through the use of trial division.

Another way this can be used is to divide N by every odd integer down from $\lfloor\sqrt{N}\rfloor$ all the way to 2. This would be in situations where the prime p is close to $\sqrt{N}$, also making the process efficient. The algorithm is as follows below.

Algorithm 7. [Sah11] This algorithm is for an input of N to generate the factors of N.

(1) for x from 2 to $\lfloor\sqrt{N}\rfloor$
if x divides N then output $x, N/s$
endif
(2) endfor

This algorithm is a key fundamental required for a better understanding of the following subsection.

3.3. **Generalized Trial Division.** The factorization algorithm we will now examine can be considered a generalization of the trial division method in Algorithm 7, developed by Sahin.[10] Once again, slightly similar to how we did for Algorithm 6, we take numerous values of x, as in the following:

$$x = \lfloor\sqrt{N}\rfloor, \lfloor\sqrt{N}\rfloor \pm 1, \lfloor\sqrt{N}\rfloor \pm 2, \ldots.$$

We then see if the greatest common factor between x and N is a proper factor of N, where $\lfloor\sqrt{N}\rfloor$ is the largest integer which is either equal to or less than $\sqrt{N}$.

Now we continue by setting $N = pq$, where both p and q are prime numbers and q is less than p. The Generalized Trial Division factoring algorithm must take at least $\lfloor\sqrt{N}\rfloor - q$ and $p - \lfloor\sqrt{N}\rfloor$ steps and therefore it is effective when $\sqrt{N}$ is close to q. If we consider Algorithm 7, in a situation where we look for the prime divisor q from $\lfloor\sqrt{N}\rfloor$ down until 2 then trial division will be more efficient than the generalization that we are currently looking at. Also, if we consider Algorithm 6, we would see that it would take

$$\frac{q+p}{2} - \sqrt{N} = \frac{q+\frac{N}{q}}{2} - \sqrt{N} = \frac{(\sqrt{N}-q)^2}{2q}$$

steps, for Fermat factoring. This would mean that both are more efficient than this generalization. However, the Generalized Trial Division is effective in situations where there is a positive integer $d > 1$ such that d times any prime factor of N is close to $\sqrt{N}$, and not only if q is close to $\sqrt{N}$.

[10] [Sah11]

The generalization is effective in the cases where a positive integer d $d > 0$ that satisfies dp or dq being near $\sqrt{N}$. The algorithm below is able to find the prime factor q in c_0 steps because q is the greatest common factor between N and dq.

Algorithm 8 (Generalized Trial Division). [Sah11] This algorithm is for an input of N to generate the factors of N.

(1) for x from 1 to $\lfloor\sqrt{N}\rfloor$
    $a_1 = \lfloor\sqrt{N}\rfloor + 1$
    $a_2 = \lfloor\sqrt{N}\rfloor - 1$
    $A_1 = \gcd(N, a_1)$
    $A_2 = \gcd(N, a_2)$
    if A_k divides N then $(k = 1, 2)$ and output A_k, N/A_j
    endif
(2) endfor

3.4. **Quadratic Sieve.** We will also describe the Quadratic Sieve factoring algorithm, although in less detail than previous sections, in order to grasp the concept. For a given number N, the quadratic sieve factoring algorithm aims to reach two numbers a, b which satisfy the two following conditions:

$$a^2 \equiv b^2 \pmod{N}$$

and

$$a \not\equiv \pm b \pmod{N}.$$

If numbers are found that satisfy those conditions, it would suggest that $(a + b)(a - b) \equiv 0 \pmod{N}$. From here it is very straightforward to check if $(a - b,N)$ is a divisor, by using the Euclidean Algorithm.[11] There is at least a 50% chance that the factor is nontrivial.

In order to do this we first set:

$$Q(a) = (a + \lfloor\sqrt{n}\rfloor)^2 - n = \tilde{a}^2 - n,$$

then process:

$$Q(a_1), Q(a_2), \ldots, Q(a_k).$$

To reach a_i, continue with the following: from the calculated $Q(a)$, choose a subset for which $Q(a_{i_1}) Q(a_{i_2}) \ldots Q(a_{i_r})$ is the square, b^2. We then recall for each $a, Q(a) \equiv \tilde{a}^2 \pmod{N}$, giving us:

$$Q(a_{i_1}) Q(a_{i_2}) \ldots Q(a_{i_r}) \equiv (a_{i_1} a_{i_2} \ldots a_{i_r})^2 \pmod{N}.$$

In cases where the earlier defined conditions remain true, we are left with divisors of N. [Lan01]

[11] [Lan01]

3.5. **Pollard's Rho.**

Algorithm 9 (Pollard's Rho Algorithm). [WG23] For an input N, our assumption is that N has a small factor.

- Choose some x_0, often given the value of 2, and some equation, generally $f(x) = x^2 + 1$.
- Compute $x_1 = f(x_0) \mod N$, $x_2 = f(x_1) \mod N$, $x_3 = f(x_2) \mod N$ and so forth, with general equation $x_(n+1) = f(x_n) \mod N$.
- For subscripts which are even for x_{2y}, we continue with the following:
  Calculate greatest common factor of $(x_{2y} - x_y)$ and N
  Continue until greatest common factor is > 1.
- The greatest common factor is a factor, completing the algorithm.

We will now demonstrate this with an example from below: We will use the algorithm with $N = 1234$. We set $x_0 = 2$ and $f(x) = x^2 + 1$. Now we calculate:

$$\begin{aligned} x_1 &= 5 \\ x_2 &= 26 \qquad \gcd(26 - 5, 1234) = 1 \\ x_3 &= 677 \\ x_4 &= 516 \quad \gcd(516 - 26, 1234) = 2 \end{aligned}$$

This leads us to 2 as the factor, completing the algorithm.

4. Applications in Cryptography

Primality testing and factoring algorithms are two key components of modern cryptography. They are utilized primarily in asymmetric, or public-key, cryptography. The concept involves two mathematically related keys, one which is openly distributed and one which is held confidential.

Primality tests are simply used to determine whether a given number, n, is prime or composite. This is used to generate secure keys, as the security of the keys (in many forms of encryption) relies on the difficulty of factoring large numbers into prime factors. To maximize the protection of the system, the prime numbers must be chosen effectively and with care. The primality tests such as in Algorithm 5, ensure potential numbers are prime. This is why it is crucial to have accurate and reliable primality tests.

The most prominent application of factoring algorithms is to decipher encrypted messages. Factorizing the public-key can enable attackers to determine the prime factors used, which breaches the security of the encryption. This is known as factorization-based cryptanalysis. The ongoing research and development of factoring algorithms is crucial for defending against potential attacks and recognizing alternative techniques which can withstand these factoring algorithms. Factoring algorithms are mainly associated with the security of Rivest-Shamir-Adleman cryptography.

4.1. **Rivest-Shamir-Adleman Cryptography.**

RSA cryptography is a form of asymmetric encryption which relies on the difficulty of factoring large numbers into their prime factors as described above. To generate a key, two large prime numbers, p and q, are chosen. Primality testing is used to verify that p and q are indeed prime. The product of these numbers N is calculated, whilst keeping the values of p and q private.

The public key is made up of two values, the modulus, N, and the public exponent, denoted by e. The public key space K, can be defined as the set that describes all pairs of

integers (e, N) where N is as defined above and $1 < e < \phi(N)$ and the greatest common factor of the pair is 1. This public key is easily accessible, and can be used to send encrypted messages to the owner of the corresponding private key.

The private key is constructed using the prime factors of N, p and q, and also consists of the private exponent, denoted by d. For the public key (e, N), the corresponding private key is represented by (d, N), for which $ed \equiv 1 \mod \phi N$. This private key is used to decrypt messages. The assumption made is that factoring large numbers into prime factors is computationally infeasible, which is why key size is highly important to the level of security. If the key size is doubled, the difficulty of deciphering increases exponentially.

4.2. **Practical Applications.** There have been numerous tests conducted utilising factoring algorithms to compare efficiency in factorizing public keys. Here we will analyse data collected on the speed and success rate of Fermat's factorization (Algorithm 6), and Pollard's Rho algorithm (Algorithm 9).

Table 1 [ABC21]: Fermat's Factorization on Public Keys from 10/16 to 20/32 bytes

No.	Public Key n	Length of Public Key n (bytes)	p	q	Execution Time (ms)	Success Rate (%)
1.	291642541 1	10/16	65357	44623	561 ms	100%
2.	117527008142 59	14	343051 7	3425927	2 ms	100%
3.	1341849068550 433	16	393584 47	3409303 9	18497 ms	100%
4.	4172366223726 2923	17	209763 919	1989077 17	13207 ms	100%
5.	4325011719545 94013	18	779594 677	5547769 69	1640872 ms	100%
6.	8763301721976 902561	19	344668 3453	2542531 637	6688088 ms	100%
7.	4980853165476 5413631	20/32	707853 7649	7036556 719	6162 ms	100 %

Fermat's Factorization method was successful for all public key sizes within the range of Table 1. There was a 100% success rate, as the algorithm found both the values of p and q.

Table 2 [ABC21]: Pollard's Rho on Public Keys from 10/16 to 20/32 bytes

No.	Public Key n	Length of Public Key n (bytes)	p	q	Execution Time (ms)	Success Rate (%)
1.	2916425411	10/16	44623	65357	8892 ms	100%
2.	1175270081425 9	14	3425927	3430517	7394 ms	100%
3.	1341849068550 433	16	3935844 7	3409303 9	9843 ms	100%
4.	4172366223726 2923	17	1989077 17	2097639 19	8564 ms	100%
5.	4325011719545 94013	18	5547769 69	7795946 77	5148 ms	100%
6.	8763301721976 902561	19	2542531 637	3446683 453	8440 ms	100%
7.	4980853165476 5413631	20/32	7078537 649	7036556 719	28704 ms	100 %

Pollard's Rho was also successful in generating values for p and q, for all public key sizes within Table 2.

Comparing the two, we can see that Pollard's Rho had greater consistency in execution time. Although Fermat's Factorization was faster for specific sizes of the public key n, on average Pollard's Rho was 1184343.43 milliseconds faster.

Table 3 [ABC21]: Fermat's Factorization on Public Keys from 22 to 38/64 bytes

No.	Public Key n	Length of Public Key n (bytes)	p	q	Execution Time (ms)	Success Rate (%)
1.	293665345516073845 3027	22	-	-	28144000 ms	0%
2.	528910732087107271 20157	23	-	-	23014000 ms	0%
3.	147307994954025982 9229771	25	-	-	23140000 ms	0%
4.	12369352403768659453215077	26	-	-	31020000 ms	0%
5.	268889892902937863 375973328747	30	-	-	32405000 ms	0%
6.	568396900241882051 50194976305169	32	-	-	58058000 ms	0%
7.	20577799505369234093237916361495739654 9	38/64	-	-	30357000 ms	0 %

Examining Fermat's Factorization dealing with larger public keys, it is seen that their was a 0% success rate from 22 bytes to 38/64 bytes. This means that p and q were not able to be deduced, and the public key is considered secure against Fermat's Factorization.

Table 4 [ABC21]: Pollard's Rho on Public Keys from 22 to 38 bytes

No.	Public Key n	Length of Public Key n (bytes)	p	q	Execution Time (ms)	Success Rate (%)
1.	29366534551607384 53027	22	49865 64726 7	58891313 281	27737 ms	100%
2.	14730799495402598 29229771	25	11043 88782 851	13338418 24921	108280 ms	100%
3.	12369352403768659 453215077	26	34822 18272 409	35521473 48653	224082 ms	100%
4.	26888989290293786 3375973328747	30	53122 50579 49433	50616944 5283459	6003859 ms	100%
5.	56839690024188205 150194976305169	32	80319 78041 99680 9	70766739 80804041	24835270 ms	100%
6.	20577799505369234 09323791636149573 96549	38	-	-	71674000 ms/	0%

Successful from 22 bytes to 32 bytes, Pollard's Rho is able to factor larger public keys. However, Table 4 depicts the lack of effectiveness of Pollard's Rho for 38 bytes, illustrating the existence of a limit.

The greater range of efficacy of Pollard's Rho is evident through Table 3 and Table 4, as this algorithm's performance has a larger limit for successful factorization. Furthermore, it was seen that the average execution time for Pollard's Rho was 8831137.775 milliseconds faster than that of Fermat's Factorization. Fermat's Factorization was indeed faster for 10/16 bytes, 14 bytes and 20/32 bytes, however, with superior average speed and size range, Pollard's Rho algorithm is the preferable choice for overall success. Altogether, this practical analysis of the two factoring algorithms displays their utility, highlighting the significance of factoring algorithms.

Conclusion

Upon exploring several algorithms and theorems, including primality tests such as the Agrawal-Kayal-Saxena primality test, Algorithm 5, and factoring algorithms such as Trial Division, Algorithm 7, we conclude with possible routes for further learning. To pursue practical aspects of factoring algorithms, one can explore *Performance Analysis of Fermat Factorization Algorithms* by Bahig, Mahdi, Alutaibi, AlGhadhban and Bahig[12] and *A General Number Field Sieve Implementation* by Bernstein and Lenstra.[13] For an example of modern algorithms, *The Number Field Sieve* by Manasse, Pollard, Lenstra and Lenstra Jr.[14]

[12] [BMA+20]
[13] [BL93]
[14] [LLMP]

Finally, to study these concepts with their links to cryptography, *The Improvement of Elliptic Curve Factorization Method to Recover RSA's Prime Factors* by Somsuk[15] and *Modified trial division algorithm using KNJ-factorization method to factorize RSA public key encryption* by Lal, Singh and Kumar.[16] The author's future research will involve examining complexity theory and modifications of existing algorithms in order to provide a helpful contribution to research in the field.

Acknowledgments

The author would like to express their gratitude to Sawyer Anthony Dobson and Simon Rubinstein-Salzedo.

Bibliography

References

[ABC21] Aminudin Aminudin and Eko Budi Cahyono. A practical analysis of the fermat factorization and pollard rho method for factoring integers. *Lontar Komputer : Jurnal Ilmiah Teknologi Informasi*, 12(1):33, Mar 2021.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160(2):781–793, Sep 2004.

[BL93] Daniel J. Bernstein and A. K. Lenstra. A general number field sieve implementation. In Arjen K. Lenstra and Hendrik W. Lenstra, editors, *The development of the number field sieve*, pages 103–126, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[BMA+20] Hazem M. Bahig, Mohammed A. Mahdi, Khaled A. Alutaibi, Amer AlGhadhban, and Hatem M. Bahig. Performance analysis of fermat factorization algorithms. *International Journal of Advanced Computer Science and Applications*, 11(12), 2020.

[Con16] Keith Conrad. The solovay–strassen test. 2016.

[Gau86] Carl Friedrich Gauss. *Disquisitiones Arithmeticae*. Springer New York, New York, NY, 1986.

[KSS02] Adam Kalai, Amit Sahai, and Madhu Sudan. Notes on primality test and analysis of aks. *Private communication, August*, 2002.

[Lan01] Eric Landquist. *The Quadratic Sieve Factoring Algorithm*. 2001.

[LLMP] A.K Lenstra, H.W Lenstra, M.S Manasse, and J.M Pollard. *The number field sieve*.

[LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 2 edition, 1994.

[LSK15] Nidhi Lal, Anurag K Singh, and Shishupal Kumar. Modified trial division algorithm using knj-factorization method to factorize rsa public key encryption. *CoRR*, Jan 2015.

[Me8] Romeo Meštrović. Euclid's theorem on the infinitude of primes: a historical survey of its proofs (300 b.c.–2017) and another new proof, 2018.

[Mon80] Louis Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theor. Comput. Sci.*, 12:97–108, 1980.

[Pat21] Jaival Patel. Sieve of eratosthenes: One of the oldest algorithms still prevalent as if it were born yesterday, Sep 2021.

[Rab80] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[Sah11] Murat Sahin. Generalized trial division. *Int. J. Contemp. Math. Sciences*, 6(2):59–64, 2011.

[Som21] Kritsanapong Somsuk. The improvement of elliptic curve factorization method to recover rsa's prime factors. *Symmetry*, 13(8):1314, Jul 2021.

[SS78] R. Solovay and V. Strassen. Erratum: A fast monte-carlo test for primality. *SIAM Journal on Computing*, 7(1):118–118, 1978.

[SS13] Robert G. Salembier and Paul Southerington. An implementation of the aks primality test, 2013.

[15] [Som21]
[16] [LSK15]

[Sut17] Andrew Sutherland. *2 Primality proving*. 2017.
[Wei23a] Eric W. Weisstein. Jacobi symbol, 2023.
[Wei23b] Eric W Weisstein. Witness, 2023.
[WG23] Justin Wyss-Gallifent. Pollard's rho method, Feb 2023.

Email address: arrakhecha@gmail.com