IRPW - Prime Heuristics and its Implications on RSA Cryptography

Sitar Eswar

July 15, 2024

Abstract

In this paper, we investigate prime heuristics and their applicability in computational number theory and cryptographic algorithms, specifically in RSA cryptography. We conducted a comparison between prime heuristics, taking into account time effeciency, computational effeciency, and overall performance in the number of primes that they each produce. We aim to result in a method by which generating primes is far easier than before, so that we can optimize RSA encryption and decryption processes. This paper will hopefully aid in finding a new method by which security and speed in cryptography may be superior to what it was before.

1 Introduction

Over thousands of years, heuristics that determine whether a number is prime have sparked the curiosity of countless mathematicians. In the third century BC, Greek Mathematician Erastothenes created one of the most widely known heuristics that we know today – the sieve of Eratosthenes [11]. Inspired by the abilities of Erastosthenes, Pierre de Fermat introduced *Fermat's Little Theorem*, a method to test for various prime numbers using modular exponentiation [1]. Later, in the 17th Century, Marin Mersenne expounded on Fermats work by creating a specific class of numbers called "Mersenne Numbers", with the property that they can be represented as $2^n - 1$, for some positive integer n [1].

Following this, in the early 1930's, Lucas and Lehmer collated the Lucas-Lehmer prime test, which was essential in the verification of Mersenne primes [10]. The Lucas-Lehmer prime test was the first check that used sequences to check if a pseudoprime is indeed prime. The later part of the 20th century saw further development with probabalistical methods, including the Miller-Rabin/Solovay-Strassen test, which introduced randomization to prime testing. Most recently, in the 1980's, Selfridge, Pomerance and Wagstaff came up with the PSW Selfridge prime check, which was revolutionary, as there is no room for error in their test -[4]. Over time, these heuristics have evolved from simple guess and check, to complicated functions and structured algorithms. Considering the vast improvement in our ability to find prime heuristics, it is clear that much more can be done.

Prime heuristics play an important role in modern cryptography, image compression algorithms useful in radiology, financial modeling, game theory, combinatorics, AI, Machine Learning, and much more. However, the scope of this research paper is largely on

the implications of prime heuristics in the security of encrypted messages (RSA Cryptography). Throughout this paper, we will explore the abilities of prime heuristics when selecting large prime numbers to secure cryptographic keys. We will also compare several popular primality tests to identify the most computationally efficient, time-effective, and accurate prime heuristic. Our goal in this project is to find an optimal heuristic that will enhance the security of RSA cryptography and aid in other fields of mathematics that require the use of prime numbers.

Roadmap: The remainder of this paper is organized as follows. In Section 2, we show how prime-checking heuristics play a crucial role in computational number theory, offering various methods to determine the primality of numbers efficiently. These tests range from historical methods like the Fermat test to more modern approaches such as the Miller-Rabin and Lucas-Lehmer tests. Each heuristic has its strengths and weaknesses, with some being highly efficient but having high numbers of false positives, while others are more reliable but computationally intensive.

In section 3, we evaluate prime number density using the Prime Number Theorem and the second Hardy-Littlewood conjecture. We evaluate various prime computation methods (Mersenne, Fermat, Selfridge, Miller-Rabin, Proth) for their application in number theory and RSA cryptography. We also use prime density graphs and test prime gaps at different bit scales to have a rough sampling of the number of primes around 256 bits (RSA private keys have 256-1024 bit primes). we also assess heuristics on prime generation, time efficiency, and false positives. Findings show Miller-Rabin as the most effective, with Fermat as a close second. Combining both could optimize RSA encryption.

Finally, in section 4, we discuss the findings and conclusions of this paper, analyzing the advantages of prime heuristics in RSA cryptography.

2 Background

2.1 Prime Number Heuristics and Checks

2.1.1 Selfridge, Pomerance, Wagstaff Heuristic

One of the most intriguing prime heuristics is the Selfridge Prime Heuristic, a conjecture that combines elements of the Fermat and Fibonacci tests. Proposed by John Selfridge, this heuristic states that an odd number p is prime if it meets certain modular constraints:

$$p \equiv \pm 2 \mod 5,$$

$$2^{p-1} \equiv 1 \mod p,$$

$$f_{p+1} \equiv 0 \mod p,$$

where f_k is the k'th Fibonacci Number.

This heuristic has yet to be provided with a counterexample. However, we must note that the Selfridge Prime conjecture is very specific upon the value of p. We would not be able to specifically discover a new prime, nor factor large composite numbers. The

Heuristics:	Summary:	Pros and Cons:
PSW-Selfridge - 1980	If p is an odd number, p is prime if the following hold: $p \equiv \pm 2 \mod 5$ $2^{p-1} \equiv 1 \mod p$ $f_{p+1} \equiv 0 \mod p$ where f_n denotes the nth fibonacci number.	**Pros:** This prime test does not fail, with a False-Positive rate of 0 (up until 100,000). **Cons:** The PSW-Selfridge primality heuristic is highly specific and serves primarily as a prime check rather than a general heuristic. It is also the least time efficient of all the heuristics.
Fermat - Mid 1600s	The Fermat Primality test is a probabalistic algorithm that checks if a number n is prime by verifying that $a^{n-1} \equiv 1 \mod n$. It is efficient, but it incorrectly identifies Carmichael numbers as prime as well.	**Pros:** The Fermat test correctly identifies all primes. The time required to perform the computations is very effecient, taking only 1 second to print primes from 1 through 100,000. **Cons:** The Fermat test considers Carmichael numbers to be prime, such as 341 and 9.
Proth - 1878	Proth's primality test is used to determine whether a Proth number is prime. A Proth number is of the form $k \cdot 2^n + 1$, where k is an odd integer and $2^n > k$. The test states that a Proth number p is prime if there exists an integer a such that $a^{(p-1)/2} \equiv -1 \pmod p$.	**Pros:** The Proth primality test is very time-efficient, taking only 0.9 seconds to compute all Proth primes from 1 through 100,000. **Cons:** The Proth primality test has a high False-Positive rate. Does not produce as many primes as Fermat, Miller-Rabin, and PSW-Selfridge. It is only applicable to proth numbers.
Lucas Lehmer Mersenne Check - 1930	The Lucas-Lehmer Prime check is a method to verify whether a Mersenne Number, of the form $M = 2^n - 1$, is prime. The check uses a sequence and modular arithmetic to prove that M is prime.	**Pros:** The prime check does not fail. There is a lot of leniency in selecting the function for the prime check. **Cons:** The prime check is computationally ineffective. It is not time-effecient.
Miller-Rabin - 1980	The Miller Rabin Test checks whether a specific property of primes holds for the number that is being tested. This test seperates a number n into $2^a * b + 1$, where $a \geq 1$, and b is odd. After a series of steps, one can say whether n is prime or composite.	**Pros:** Is very reliable in identifying primes and composites, if given sufficient number of witnesses. It is almost as reliable as the Fermat test. **Cons:** The prime test largely depends on what the witness (number generated at random) is.. i.e, 9 may be marked as prime if 8 is a witness. The time necessary to perform the computations is higher than Fermats.
Lucas-Lehmer-Riesel - Early 21st Century	The Lucas-Lehmer-Riesel primality check is a method to verify whether a certain number of the form $M = k * 2^n - 1$ is prime. This check, like the Lucas-Lehmer Check, uses a sequence to prove the primality of M.	**Pros:** The primality check does not fail in identifying whether the testing number is prime or composite. **Cons:** It is computationally ineffective, amd similar to the Lucas-Lehmer Test, it is not time-efficient. The time required to select a starting value s_0 of the sequence is also ineffecient.

Table 1: Overview of Prime Number Heuristics

conjecture is purely a method by which we can check whether a number that is $\pm$ 2 mod 5 is prime. Hence, we can do better.

2.1.2 Fermat Primality Test

In comparison to the Selfridge Primality test (a heuristic), the Fermat Primality test is a probabilistic (composite) test. A probabilistic test is a primality test that outputs a "possible prime", where the number that gets outputted has a high probability of being prime. Probabilistic tests are more rigorous than the prime heuristics that are discussed in the sense that they provide provable bounds on the probability of being misinterpreted as a prime (when the number is actually composite). However, one may reduce the error rate by inputting many different values of "a" from a sample set, as shown below.

The Fermat Primality test goes something like this:

Given an integer n, choose some integer a coprime to n and calculate a^{n-1} modulo n. If the result is different from 1, then n is composite. If it is 1, then n may be prime.

It is key to note that if it is 1 mod n, n *may* be prime. A common counterexample is the "pseudoprime", $n = 341$, with $a = 2$. $2^{340} \equiv 1 \mod 341$, but 341 is not prime (11 * 31).

On the other hand, Carmichael Numbers have the property that $a^{n-1} \equiv 1 \mod n$, for every a that is coprime to n. Aside from Carmichael Numbers and the vast amount of pseudoprimes, the Fermat test is used as a rapid test in RSA cryptography, especially when one is choosing the large prime numbers to multiply.

2.1.3 Miller-Rabin Primality Test

Similar to the Fermat primality Test, the Miller Rabin test checks whether a specific property of primes holds for the number that is being tested. Let us go into the details of the Miller-Rabin Test.

Starting off, we consider an odd integer $n > 1$, and we seperate it into $n = 2^a * b + 1$, where $a \geq 1$, and b is odd. Example: If $n = 17$, $a = 4$, $b = 1$.

With this construction of n, we can implement a Miller Rabin test, as follows:

- We pick a random $y \in \{1, 2, \ldots, n-1\}$.
- If $y^b \equiv \pm 1 \mod n$, then n is prime.
- If $y^{2^r * b} \equiv -1 \mod n$, where $0 \leq r \leq a-1$, n is prime
- If this is also false, then n is composite.

However, it is key to note that n is not guaranteed to be prime, and the Miller-Rabin Primality test is a "strong probable prime test".

2.1.4 Proth Primes

Proth's Primality Test is designed to check whether a specific property of Proth numbers holds for the number that is being tested. Let's go into the details of Proth's Primality Test. Let us start by considering a Proth number, which is an odd integer of the form

$n = k * 2^m + 1$, where k is an odd integer and $2^m > k$. Example: If $n = 17$, it can be expressed as $1 * 2^4 + 1$ with $k = 1$ and $m = 4$.

With this construction of nn, we can implement Proth's Primality Test, beginning by picking a random integer a such that $1 \leq a < n$. Next, we calculate $a^{(n-1)/2} \mod n$. If $a^{(n-1)/2} \equiv -1 \mod n$, n is prime. If this condition is not met, it is composite. However, it is important to note that while this test is quite efficient for Proth numbers, it is specifically tailored to this form and cannot be applied to other forms of numbers. However, note that the rate of False Positives for proth numbers is very high, and although it is a deterministic test, it is not very effective.

2.1.5 Mersenne Primes

Mersenne Primes are a special class of prime numbers in the form of $M = 2^n - 1$, where n itself is a prime number. Example: $n = 3$, thus $M = 2^3 - 1 = 7$, a prime number. However, not all inputs of n result in M being prime. One given example is $n = 11$, and $M = 2^11 - 1 = 2047$, which is indeed not prime. However, in order to verify that a Mersenne number is prime, we can use the Lucas-Lehmer primality check.

2.1.6 Lucas-Lehmer Test

The Lucas Lehmer Primality test is a prime-check to verify whether a Mersenne pseudoprime is prime. The Lucas Lehmer Primality Test begins by constructing a sequence, s_i, such that $s_0 = 4$, and $s_i = (s_{i-1}^2 - 2)$, for subsequent s_i's.

The first few values of the sequences are; 4, 14, 194, 17634, ...

According to the Lucas Lehmer Test, M is prime if and only if s_{n-2} is congruent to 0 mod M. However, the starting value of s_i does not necessarily have to be 4. For instance, starting values of 10 and 52 have been proved to be able to confirm the primality of Mersenne Primes. The Lucas Lehmer test is one of the only primality checks that have actually been proved. However, due to the large numbers involved, it is computationally ineffective, lowering time effeciency.

2.1.7 Lucas-Lehmer-Riesel Test

Similar to the Lucas-Lehmer Test, the Lucas-Lehmer-Riesel (LLR) test is used for number of the form $M = k \cdot 2^n - 1$, where k is an odd positive integer, and n is a positive integer. Keep in mind that $k < 2^n$. The Lucas-Lehmer-Riesel test proceeds as follows: We construct a sequence s_i such that $s_i = (s_{i-1}^2 - 2)$. However, the values of k depends on our value of s_0.

Case 1: $k \equiv 1$ or $5 \mod 6$. It is clear that if $k \equiv 1 \mod 6$ and n is even, or $k \equiv 5 \mod 6$ and n is odd, M is divisible by 3.

I will break this down: $k = 6m + 1$ for some integer m. For an even n, $2^n \equiv 1 \mod 3$, as $2^2 \equiv 1 \mod 3$. We also know that $k = 1 \mod 3$. Thus, $M = 1 \mod 3 * 1 \mod 3 - 1 = 1 * 1 - 1 = 0$. Thus, M is divisible by 3 if $k = 1 \mod 6$. Now, if $k \equiv 6m + 5$, for some integer m, and n is odd, then $2^n \equiv 2 \mod 3$, as 2^1, 2^3, 2^5 ... $\equiv 2 \mod 3$. Also note $k = 2 \mod 3$. Thus, $M = 2 \mod 3 * 2 \mod 3 - 1 = 4 \mod 3 - 1 \equiv 0 \mod 3$. Thus, if $k \equiv 5 \mod 6$, and n is odd, then M is divisible by 3.

However, if $k \equiv 1 \mod 6$ and n is odd, or $k \equiv 5 \mod 6$ and n is even, $M \equiv 7 \mod 24$. If this is the case, we take $s_0 = (2 + \sqrt{3})^k + (2 - \sqrt{3})^k$.

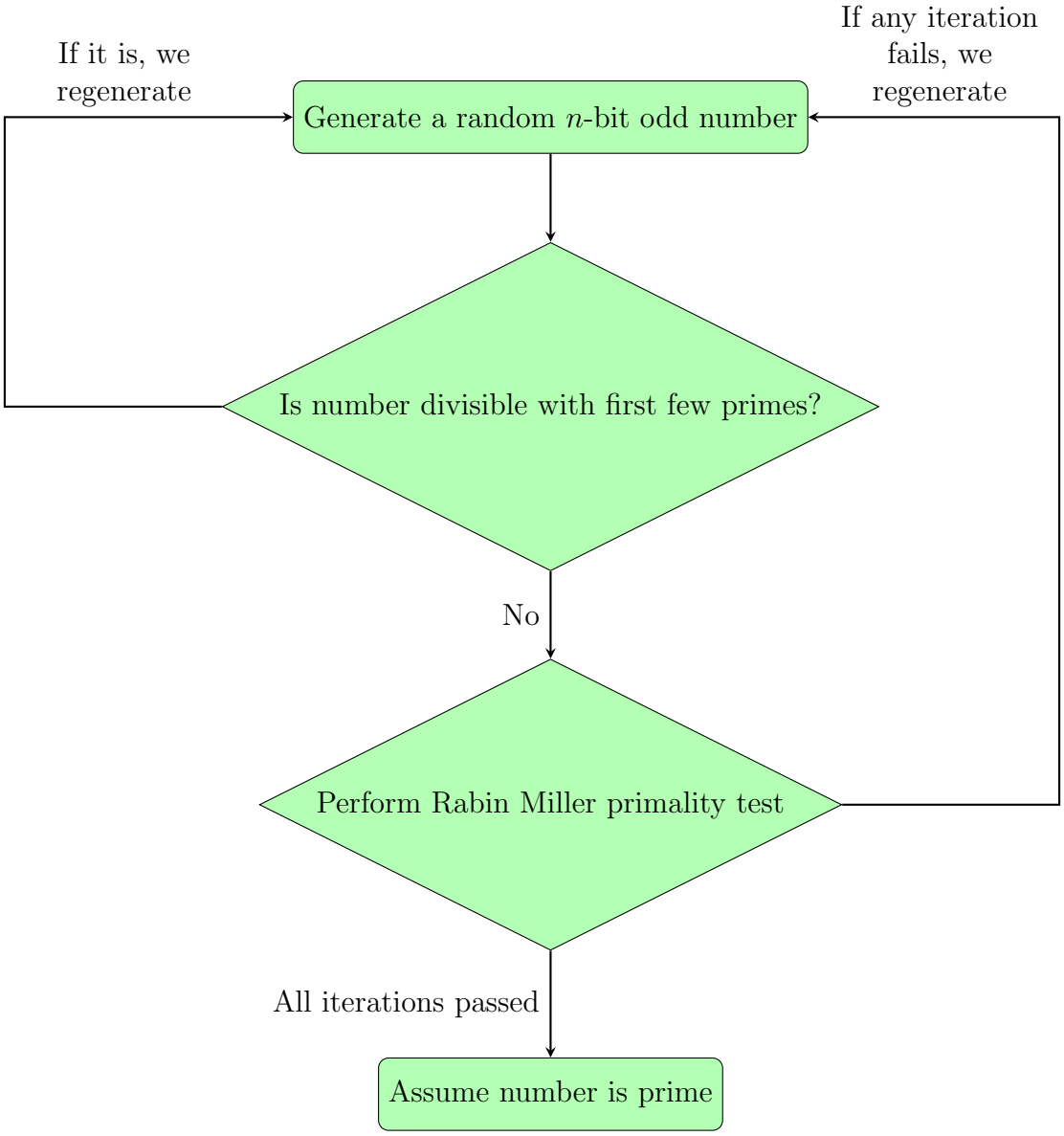

Figure 1: This figure shows the process by which key-holders choose their private key

Case 2: k is a multiple of 3. It is known that if $k = 3$ and $n \equiv 0$ or 3 (mod 4), we can take $s_0 = 5778$. The Lucas Lehmer Riesel test is primarily used as a check to identify whether numbers of the form $M = k * 2^n - 1$ are prime. However, similar to the Lucas Lehmer test, the LLR test is not computaionally effective.

2.2 Prime Numbers and Public Key Cryptography

There are many forms of public key cryptography including RSA [6], Elliptic curve cryptography [2], and Lattice-based cryptography [9]. Here, we focus on RSA cryptography and the implications of prime heuristics on its security. The key generation process for RSA cryptography begins by choosing two random primes numbers, denoted p and q. Keep in mind that p and q are kept confidential. To optimize the robust security of a private key, the prime numbers should be sufficiently large.

We compute n as a product of p and q, i.e. $n = p \cdot q$. The value n forms a part of

the public key. Following this, we calculate λn using Carmichael's totient function. Since $n = pq$, λn can be determined as $\lambda n = \text{lcm}\ (\lambda p, \lambda q)$. From Carmichael's totient function*, we derive that $\lambda p = \phi p$, where $\phi p = p - 1$ due to Euler's Totient Theorem. Similarly, $\lambda q = q - 1$ follows the same reasoning. Thus, $\lambda n = \text{lcm}\ (p - 1, q - 1)$ is computed and maintained as part of the private key. To show how the private key is usually computed in RSA cryptography, I have drawn a flowchart (Figure 1) from a geeksforgeeks site [3].

Next, we choose an integer e (commonly $2^{16} + 1$), such that $0 \leq e \leq \lambda n$ and e is coprime to λn. The value e is also released as part of the public key. After this, we determine a value d such that d is the modular multiplicative inverse of e, where $d \equiv e^{-1} \mod \lambda n$. Unlike e, we keep d confidential, as a component of the private key.

2.2.1 RSA Message Exchange Scenario

This is a scenario illustrating how Bob encrypts a message and transmits it to Alice using RSA cryptography: First, Alice must generate her public and private keys. She chooses two sufficiently large prime numbers, (let us say 61 and 53 for simplicity), and multiplies them. $n = 61 \cdot 53 = 3233$. Following this, she computes λn, which, by Carmichael's totient function, is equal to $\text{lcm}(p - 1, q - 1) = \text{lcm}(60, 52) = 780$.

Next, she chooses an arbitrary e such that e is coprime to λn (As said before, e is normally $2^{16}+1$, but for simplicity, let us say Alice picks the prime number 17). After this, Alice computes d, the modular multiplicative inverse of e, such that $d \equiv e^{-1} \mod \lambda n$. Now, using the extended Euclidean Algorithm Theorem*, we can calculate the value of $d = 413$. Alice's public key is (17,3233), and her private key is (413, 61, 53, 780).

Bob wants to send an encrypted message to Alice. Bob first needs to convert his message into a numerical value, let's say $M = 11$. Bob encrypts M using Alice's public key, 17 and 3233. Bob calculates the ciphertext, which is $C \equiv M^e \mod n$. Next, he performs modular exponentiation to find that $C \equiv 11^{17} \mod 3233 \equiv 3061$.

After this, Bob sends the ciphertext to Alice. Alice recieves the ciphertext from Bob and decrypts it using her private key, d. She calculates M as $M \equiv C^d \mod n$. This means that $M \equiv 3061^{413} \mod 3233 \equiv 11$, as desired. After Alice has recieved M successfully, she interprets M back into the original message Bob encrypted. This process makes sure that only Alice, with her private key, can decrypt the message that Bob encrypted with her private key, which ensures safety in their communication.

2.3 Selected Attacks on RSA-based Cryptography

Despite various attacks and vulnerabilities that have been discovered over the years, none have proven to be devastating. Most attacks highlight the improper use of RSA, such as using low values of e, the exponent, or common moduli across different users. In the following subsections, we describe some attacks that have been used against RSA.

2.3.1 Hastad's Broadcast Attack

In 1988, Hastad presented Hastad's Broadcast Attack, which involves using small private exponents, which can be exploited to recover the private key efficiently [13]. Let us represent this with the Alice and Bob scenario.

Bob wants to send the same encrypted message M, to multiple recipients (multiple Alice's), called $A_1, A_2, ...A_k$. Each recipient has a different RSA public key, say, (n_i, e_i), where n_i is the product of the two large primes, and e_i is the exponent. Bob encrypts

the message M using each recipients public key, producing the Ciphertext's, C_i, for every recipient A_i (i.e., $C_i \equiv M^{e_i} \mod n_i$).

Now, an attacker, Marvin, intercepts the Ciphertexts $C_1, C_2, ...C_k$, that is sent to the recipients. The public exponents are relatively small and common. They are most commonly 3, or $2^{16} + 1 = 65537$. They are ideally the same for all the A_i's. It is also safe to assume that the moduli n_i are different and pairwise coprime. We also know that M is smaller than each of the n_i's. Now, we can apply the Chinese Remainder Theorem. Since the moduli are all pairwise coprime, Marvin can use CRT to combine the Ciphertexts that he intercepted. His result is something like this:

$$C \equiv M^e \mod n_1 n_2 n_3 ... n_k$$

Note that C is a number such that $C \equiv C_i \mod n_i$ for every i. We know that $M < n_1, M < n_2 ... M < n_k$, so we can find $M^e < n_1 n_2 n_3 ... n_k$. This implies that M^e is in the range of $n_1 n_2 ... n_k$. With this, Marvin can compute the e-th root of C find M.

Example with $e = 3$: Let us assume tha $e = 3$ and Bob sends the message M with different moduli n_1, n_2, n_3. Thus, we can say that

1. $C_1 \equiv M^3 \mod n_1$
2. $C_2 \equiv M^3 \mod n_2$
3. $C_3 \equiv M^3 \mod n_3$

Marvin intercepts C_1, C_2, and C_3, and constructs a combined congruence using CRT as shown above. He results with $C \equiv M^3 \mod n_1 n_2 n_3$. Since $M < \min(n_1 n_2 n_3)$, we know that $M^3 < n_1 n_2 n_3$. Thus, M^3 is less than the product of the moduli. Now all Marvin needs to do is cube root C to find M.

2.3.2 Attacks based on weak PRNG Seeds

In 1996, Goldberg and Wagner proposed two attacks on Netscape's implementation of RSA public key encryption schemes that uses detailed knowledge of its internals to predict the encryption key. When a connection is established, an unencrypted challenge value is sent from the Netscape client to the secure server, which an attacker can intercept. While Unix browsers are more secure due to better randomization, they are still vulnerable.

The first attack assumes the adversary has access to the Unix system. The attacker is able to use the *ps* command to find the process ID (pid) and parent process ID (ppid) values. By using Ethernet sniffing tools, the attacker can estimate the time of day when the challenge was sent, which will narrow down the possible seed values used for generating the encryption key. Since the time is in seconds and the pid, and ppid known, only the microseconds value remains unknown. This means there are only one million possible values. The attacker then performs a brute force attack by generating and testing each possible seed to find the correct one that matches the intercepted challenge value. This process allows the attacker to recover the secret encryption key efficiently [12].

The second attack assumes the attacker does not have an account on the targeted UNIX machine, which means that the process ID (pid) and parent process ID (ppid) are unknown. However, Goldberg and Wagner propose that the values are predictable. The ppid is often 1 or slightly smaller than the pid, and process IDs can be inferred from other applications like sendmail [12]. Using these characteristics, the attacker can approximate the pid and the ppid. This allows the attacker to perform a more efficient brute force attack by only trying keys generated from plausible seed values. With this, it is possible for an attacker to break Netscape's encryption in minutes.

2.3.3 RSA Blinding Attack

Boneh describes the RSA blinding attack, which allows Marvin, the adversary, to obtain a valid signature on a chosen message by transforming the message before requesting the signature. I will show the process of the attack: Alice, as before, has a private key, $(\lambda n, d)$, and a public key, (n, e). Marvin, the attacker, wants Alice's signature to be on a message, M. Alice refuses to sign M. Now, in order to blind Alice and transfigure the message, Marvin selects a random $r \in \mathbb{Z}_n$, and computes

$$M' \equiv r^e M \mod n.$$

Once M' is found, Marvin asks Alice to sign M'. Alice agrees, and provides Marvin with the signature S. Note that

$$S \equiv (M')^d \mod n.$$

Next, Marvin computes

$$S' \equiv S * r^{-1} \mod n$$

which is Alice's signature on M. Now we can exponentiate both sides of the equation by e, and result in

$$(S')^e = S^e * r^{-e} \equiv M \mod n.$$

Thus, Marvin successfully obtains Alice's signature on M without her realizing it.

Example with Key Generation: Let's say Alice's two prime numbers are 3 and 11. $n = 3 * 11 = 33$. $\lambda n = \text{lcm}\ (p-1, q-1) = \text{lcm}\ (2, 10) = 10$. Next, we choose a public exponent such that $\text{lcm}(e, 10) = 1$. Let us say $e = 3$. Next, we need to choose the private exponent d such that $e * d \equiv 1 \mod \lambda n$. $3 * d \equiv 1 \mod 10$, so $d = 7$. Hence, alice's private key is $7, 10$, and her public key is $3, 33$. Now, to the steps of the RSA blinding attack. Let us say that the message M that Marvin wants Alice to sign is 5. Marvin needs to select a random r to compute a new message. Let us say that $r = 2$. We know that $e = 3$, so $M' \equiv 5 * 2^3 \mod 33 = 7$. Alice now signs M'. She provides Marvin with a signature $S = 28$, derived from $S \equiv (M')^d \mod n$. Next, Marvin computes $S' \equiv S * r^{-1} \mod n$, which is Alice's signature on M. $S' = 28 * 1/2 \mod 33 = 14 \mod 33$. After exponentiating, we get that $(14)^3 \equiv 5 \mod 33$.

2.3.4 Opportunistic Mining of "P's" and "Q's"

In 2012, Heninger et al. [14] delved into vulnerabilities of private keys due to incompetent random number generators. They harvested public keys by scanning the Internet for TLS and SSH services and were able to identify the private keys through a gcd attack. The team recognized consistencies in the random number generators, (urandom), and were able to calculate the gcd between two moduli, which led them to identifying the private keys. The method by which they efficiently computed the GCD between moduli is listed below. They note that 5.57 percent of TLS hosts, and 9.6 percent of SSH hosts use the same private keys as other hosts, in an apparently vulnerable manner (they were able to compute 64000 private keys of the TLS hosts, and 108000 private keys of the SSH hosts).

The paper leverages a method proposed by Bernstein et al. [5] to effeciently compute the GCD between several Moduli in parallel. By finding common factors between different RSA moduli, the corresponding factor can be identified as well, breaking the private key. As shown by Bernstein in his paper, we can use a product tree to calculate the gcd between moduli.

First, we construct a product tree by multiply the different moduli $P = n_1 n_2 n_3 ... n_k$. After computing P, we construct a remainder tree, to find the remainders of $P \mod n_i^2$, for each n_i. Let us denote q_i to $q_i = P \mod n_i^2$. The final output for each modulus n_i is the gcd of n_i and $\frac{q_i}{n_i}$. The moduli that result in a gcd $\neq 1$ are candidates for having shared primes. The gcd that results is one of the prime factors.

However, we should note that in some cases, the GCD might be the modulus itself. This happens when a modulus shares both of its prime factors with two other distinct moduli. When the gcd gives an output of the moduli, note that we do not know what the gcd of the moduli are.

3 Empirical Evaluation

In this section, we will delve into the evaluation of prime number density in relation to two fundamental theorems in number theory: the Prime Number Theorem and the second Hardy-Littlewood conjecture. The Prime Number Theorem [15] is an asymptotic distribution of prime numbers, stating that the number of primes less than a given number x is approximately

$$\pi(x) \approx \frac{x}{\ln x}$$

where $\pi(x)$ denotes the number of primes less than or equal to x. This theorem serves as a foundation for graphing the density of prime numbers up to large limits.

The second Hardy-Littlewood conjecture [16] also deals with the approximate number of primes in an interval, stating that for sufficiently large x and y, the following inequality holds:

$$\pi(x + y) - \pi(x) \leq \pi(y)$$

.

Here, we aim to empirically approximate the density of prime numbers for 32-bit, 64-bit, 128-bit, and 256-bit numbers. By calculating the approximate density of primes within these bit ranges, we will assess the consistency of our findings with the predictions made by the Prime Number Theorem and the second Hardy-Littlewood Conjecture. This enables us to analyze the effort required to brute-force an RSA private key and explore methods for implementing prime heuristics to reduce the computational work involved in such attempts.

3.1 Efficacy of Prime Heuristics

We shall begin by assessing the efficacy of various prime computation methods and their practical applications in number theory, RSA cryptography, and computational mathematics. Let us start by exploring Prime density, to give us a model to evaluate the accuracy of a prime heuristic. Figure 2 showcases a prime density graph, up to 100,000,000.

To estimate the gaps between consecutive primes with much larger numbers, we generated random odd numbers at the 255-256 bit scale and tested their primality using the Miller-Rabin and Fermat primality tests. We generate random odd numbers, check their primality, and incremented them by 2 if they are not prime. We also subtracted 2 to find the largest prime number less than the random number generated. We kept track of the number of additions required to find a prime. After executing the loop 1000 times, we logged the total number of additions and subtractions, and added them up. Now, since

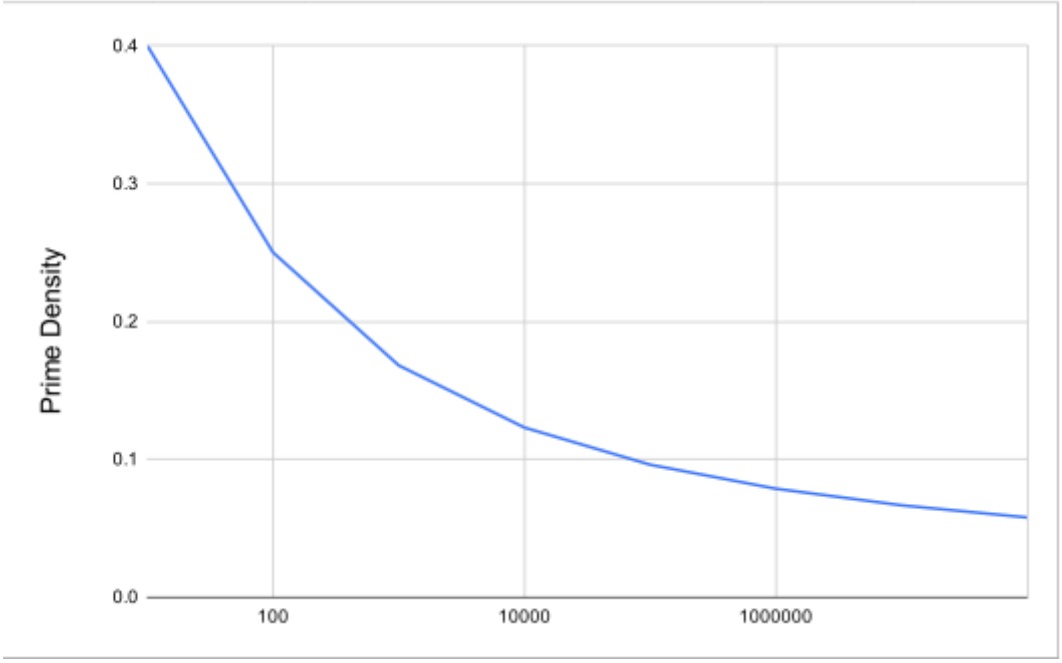

Figure 2: Prime Density graph displaying the density of primes up to 100,000,000

we increment by 2, we multiply the total addition by 2 and divide by 1000 to get the approximate gap between consecutive primes. We repeat these steps for numbers that are 32 bit, 64 bit, and 128 bit as well. Figure 6 depicts my findings.

Now, we must determine which heuristic can be most applicable in RSA cryptography. To determine the optimal heuristic, I have graphed five different approaches (Mersenne, Fermat, Selfridge, Miller-Rabin, and Proth) across three dimensions: the quantity of true primes generated, their production time efficiency, and the incidence of false positives.

Figure 3 illustrates the number of primes that are produced by each Prime Heuristic. Specifically, this graph shows that the Mersenne prime heuristic is extremely inefficient when calculating primes in strict bounds. Additionally, it reveals that Proth's heuristic is ineffective for generating primes below 100,000. It is evident that Fermat, Miller-Rabin, and Selfridge are the most effective in prime generation.

Next, we'll delve into the time-efficiency aspect of the prime heuristics. I have represented the time required to recieve outputs from the same 5 heuristics up until 100000 in Figure 4. This figure shows the time it takes to output values in seconds. With the PSW-selfridge test taking the longest, at 24 seconds, we do not need to consider it as a plausible prime heuristic for computing primes at a large 128 bit scale. However, Miller-Rabin and Fermat proved to be the most efficient, with the time only taking 4 seconds and 1 second respectively. Note that Proth and Mersenne have high time efficiency due to minimal outputs.

Now, we can discuss the 3rd dimension to these heuristics, the number of False-Positives. False Positives are the numbers that are marked as Prime by the heuristics, but are, in fact, composite. Figure 5 uses a bar graph to represent the number of False-Positives throughout the 5 heuristics. With this bar graph, we can see that Proth has the highest rate of False positives (with over 300 marked as prime), Fermat's with around 150, and Mersenne with 11. However, Selfridge and Miller-Rabin do not have any false positives. This is because the PSW-Selfridge test is an exceptionally specific primality check (for instance, it only takes into account numbers that are $\pm 2 \mod 5$. Miller-Rabin

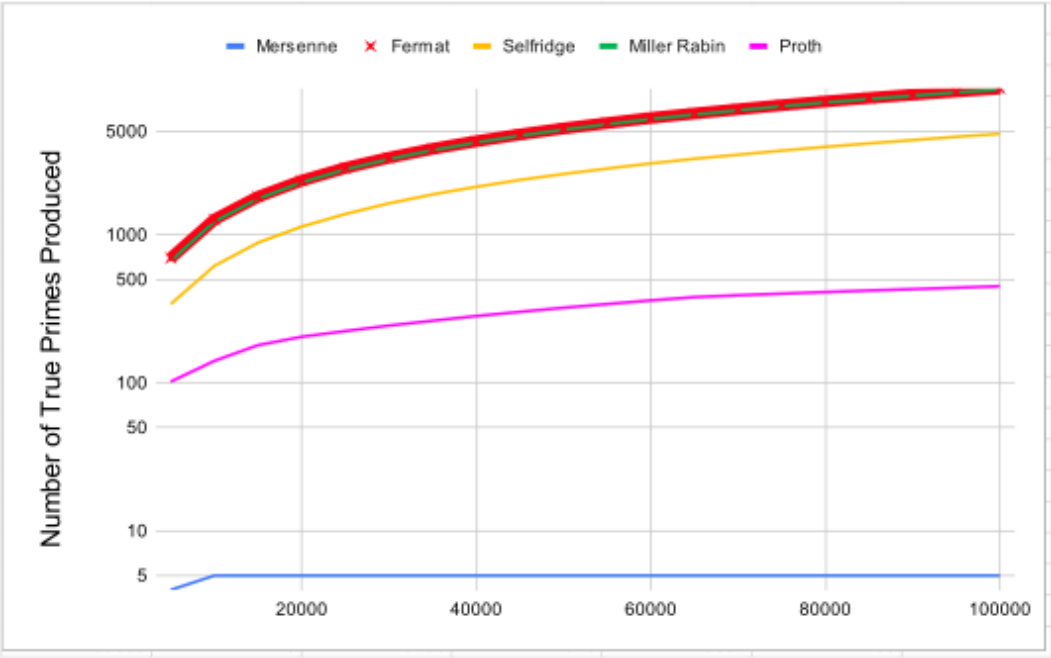

Figure 3: Miller Rabin, Fermat, and Selfridge produce the most number of primes within certain bounds

has 0 False-Positives with 20 iterations of witnesses for each number that it checks.

Combining the results from these 3 graphs, we can concur that Miller Rabin is the most effective prime heuristic, with Fermat's being a close second. Although Miller-Rabin is not as time-efficient as Fermat's, the number of False Positives is drastically smaller. Note that both of these are due to the number of iterations of witnesses that I used. With a smaller iteration, Miller-Rabin will be more time efficient, but it will most likely have False-Positives. With these factors into consideration, it is understandable why RSA cryptography uses the Miller-Rabin prime heuristic to verify if ones private key is indeed prime. However, it is important to note that, in my research, Fermat's proved to be more computationally effective, so it may be beneficial to combine the works of Fermat and Miller-Rabin to produce an optimal encryption.

3.2 Number Field Sieve's applications in RSA Cryptography and Factoring large prime numbers

The Number Field Sieve algorithm is currently the best known method for factoring composite numbers with more than 100 digits [8]. There are two main variations: the Special Number Field Sieve (SNFS) and the General Number Field Sieve (GNFS). he SNFS can quickly factor large numbers but is limited to numbers of a special form. The GNFS, on the other hand, can factor any composite number, although it is slightly slower than the SNFS due to its broader applicability.

3.2.1 Special Number Field Sieves

The Special Number Field Sieve (SNFS) is an algorithm used for factoring integers of a special form

$$n = r^e \pm s$$

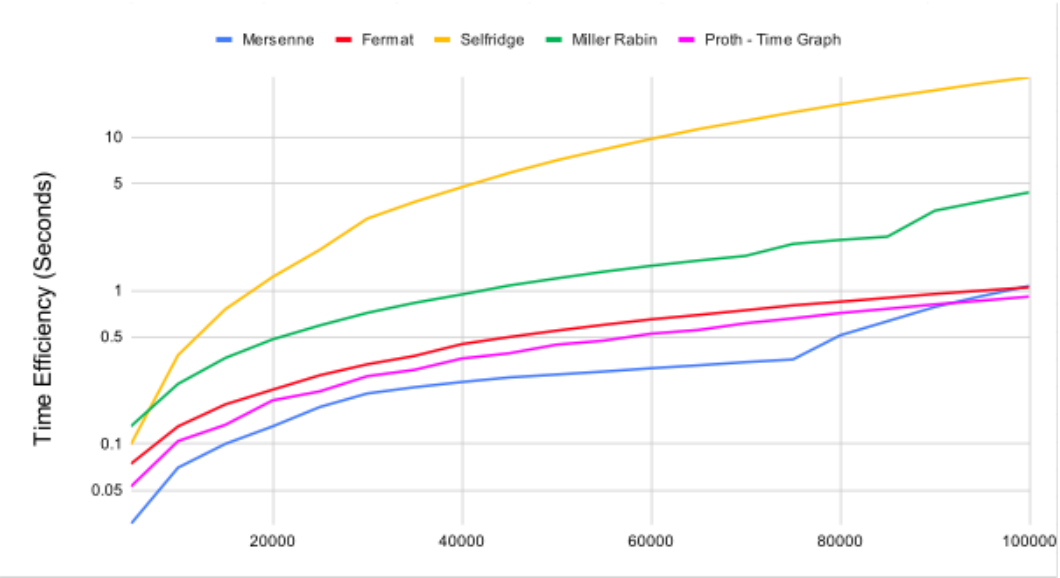

Figure 4: Illustration of the time required for heuristics to print prime numbers from 0 to 100000

, where r and s are relatively small integers. The SNFS is an optimized version of the general Number Field Sieve (NFS), taking advantage of the algebraic structure of the specific form of n. Heuristically, its complexity for factoring an integer n is of the form [7]:

$$\exp\left((1+o(1))\left(\frac{32}{9}\log n\right)^{1/3}(\log\log n)^{2/3}\right) = L_n\left[\frac{1}{3}, \left(\frac{32}{9}\right)^{1/3}\right]$$

Let us start with some definitions. First, we define two number fields; the rational number field $\mathbb{Q}(x)$ and an algebraic number field $\mathbb{Q}(\theta)$ where θ is a root of $f(x)$. Next, we will define a smooth relation. A smooth number is an integer that factors completely into small prime numbers i.e. an n-smooth number is a number whose prime factors are all less than or equal to n (This is similar to a factor base, which is the set of all primes less than or equal to n). For example, a 7-smooth number is a number who's prime factors are all primes less than or equal to 7. One example of a 7-smooth number is 30, as all the prime factors of 30 are less than 7 (2, 3, and 5). A relation is a pair (a, b) such that $a - b\theta$ (in the algebraic number field) and $a - br$ in the rational field are both smooth numbers. We shall also explain what a multiplicative relation is.

First, we identify a large number of multiplicative relations among a factor base in $\mathbb{Z}/n\mathbb{Z}$, ensuring the number of relations exceeds the number of factor base elements. Combine these relations so that all exponents are even, forming congruences of the type

$$a^2 \equiv b^2 \mod n$$

, which lead to factorizations of n.

To factor n, we follow these steps: We first choose an irreducible polynomial f with integer coefficients and an integer m such that $f(m) \equiv 0 \pmod{n}$. Let α be a root of f, forming the ring $\mathbb{Z}[\alpha]$. There is a unique ring homomorphism ϕ from $\mathbb{Z}[\alpha]$ to $\mathbb{Z}/n\mathbb{Z}$ mapping α to m. Next, we establish two factor bases: one in $\mathbb{Z}[\alpha]$ and one in $\mathbb{Z}$. The former consists of primes in $\mathbb{Z}[\alpha]$ with norms below a chosen bound $N_{\max}$, and the latter

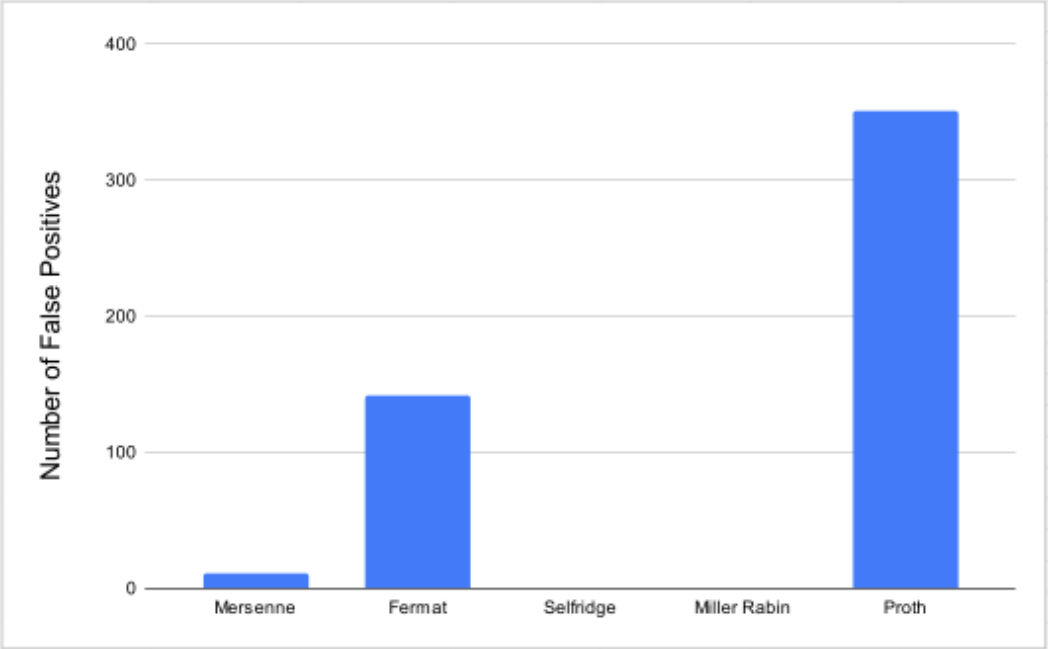

Figure 5: Plot illustrates False-Positive rates between Heuristics

consists of prime integers up to a certain limit. After this, we search for pairs of integers (a, b) such that $a+bm$ is smooth relative to the $\mathbb{Z}$ factor base and $a+b\alpha$ is smooth relative to the $\mathbb{Z}[\alpha]$ factor base. This process involves sieving, similar to the Sieve of Eratosthenes. For each pair, apply the ring homomorphism ϕ and the standard homomorphism from $\mathbb{Z}$ to $\mathbb{Z}/n\mathbb{Z}$. We set these equal to derive a multiplicative relation among the elements in $\mathbb{Z}/n\mathbb{Z}$.

By setting these equal, we establish a relationship among elements in $\mathbb{Z}/n\mathbb{Z}$ that involves products of primes (or prime ideals) from the respective factor bases. This relation, when combined with others obtained through similar processes (such as sieving), contributes to constructing congruences of the form $a^2 \equiv b^2 \pmod{n}$ or similar, which help in the factorization process [8].

3.2.2 General Number Field Sieves (GNFS)

The general number field sieve is one of the most efficient algorithms for factoring integers larger than 256 bits. It is commonly known as [7]:

$$\exp\left(\left(\frac{64}{9}\right)^{\frac{1}{3}}(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$$

The number field sieve generalizes the Special Number Field Sieve, which, as we know takes only numbers of special form into consideration. However, the General Number Field Sieve can factor any number except prime powers, but, of course, factoring a prime power is trivial, as one can just take its roots.

The number field sieve, just like the special number field sieve, is an upgrade of the quadratic and rational number field sieves, which look for smooth numbers of order $n^{\frac{1}{2}}$. The GNFS searches for smooth numbers that are subexponential in size, making them more likely to be smooth and improving efficiency. To expedite the process as described, one must perform the factorizations in number fields [8].

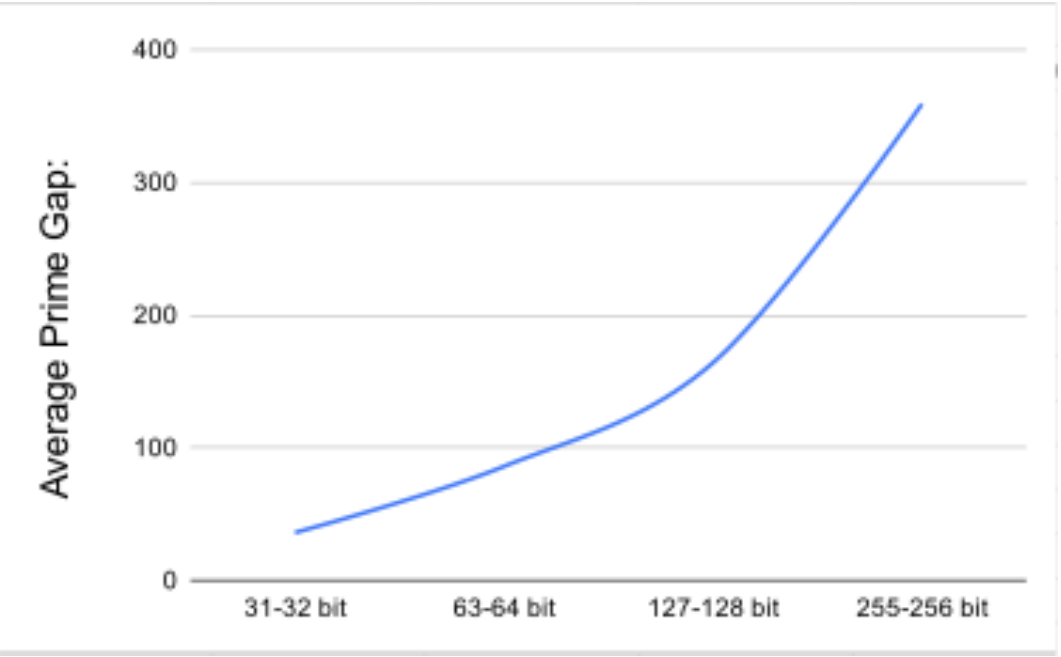

Figure 6: Average prime gap increases as bit length increases

Factorization:

To factor a number using the GNFS, two polynomials $f(x)$ and $g(x)$ with small degrees and integer coefficients are selected. These polynomials must be irreducible over the rationals and share a common root modulo n.Once the polynomials are chosen, the algorithm works with number field rings $\mathbb{Z}[r_1]$ and $\mathbb{Z}[r_2]$, where r_1 and r_2 are roots of f and g respectively. The goal is to find pairs of integers a and b such that the values r and s, derived from $f(a/b)$ and $g(a/b)$, are smooth relative to a chosen basis of primes. This is typically achieved through lattice sieving, requiring a large factor base for efficiency.

After gathering enough smooth pairs, Gaussian elimination is used to find products of certain r and s that are squares simultaneously. These products are then mapped through homomorphisms to the ring $\mathbb{Z}/n\mathbb{Z}$, allowing the final factorization of n by finding the greatest common divisor of n and the difference between two computed squares.

3.2.3 Implementation of NFS in MSieve

MSieve is a highly optimized C library designed for integer factorization, particularly effective for large composite integers like RSA moduli. It utilizes both the Quadratic Sieve (QS) for numbers up to around 100 digits and the Number Field Sieve (NFS) for larger numbers, using multiple polynomial selection, sieving strategies, parallel processing, and mathematical optimizations (including prime heuristics) to enhance performance. In cryptographic applications, MSieve is essential for evaluating the security of RSA keys by efficiently factoring the product of two large prime numbers. Figure 7 displays the time in seconds to factor a number (bit length is listed) made by multiplying two large prime numbers.

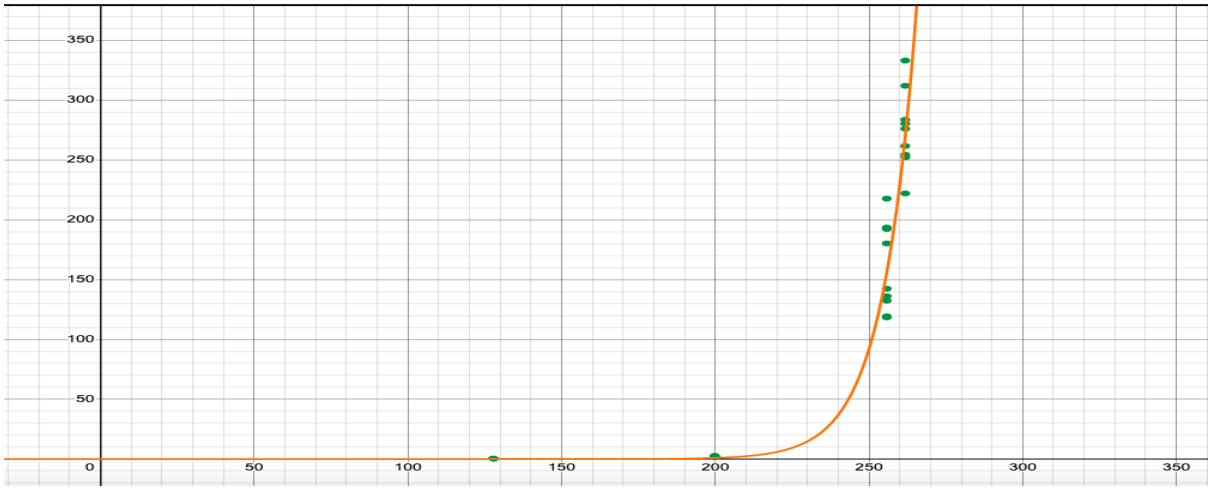

Figure 7: Experiments were run using the Msieve library on a 4-CPU 2.9 Ghz Ubuntu system. Approximately 3 mins to factor 256-bit number, but computation time rises exponentially! (x-axis: bit length; y-axis: time, in seconds)

4 Discussion

4.1 How do prime heuristics assist in brute forcing RSA keys?

Prime heuristics are essential tools for identifying prime numbers efficiently. This efficiency is crucial in two main areas: generating RSA keys and attacking RSA by finding plausible prime factors of the modulus n. For key generation, prime heuristics are essential in the process of finding large prime numbers, which are fundamental to creating secure RSA keys. In the context of attacks, these heuristics help narrow down the search for prime factors of n, thereby making the factorization process more feasible. The ability to quickly identify primes significantly helps both in the generation of secure keys and the vulnerability assessment of existing keys.

4.2 Importance of Targeting the Correct Range for Prime Factors

When attempting to factorize a very large composite number n as an RSA moduli, targeting the correct range for potential prime factors is crucial. Typically, the prime factors of an n-bit composite number are approximately $\frac{n}{2}$ bits in size. For example, if n is a 1024-bit number, each prime factor would likely be around 512 bits. Focusing the search within this range dramatically reduces the number of candidates that need to be checked. Without this targeted approach, one would have to consider a much broader range of possible factors, which increases the number of potential candidates exponentially. For instance, if one was to search for factors of a 1024-bit number without narrowing down the range, one might have to consider every possible prime number up to 1024 bits. This is an astronomically large number of possibilities.

However, by narrowing the search to around 512-bit numbers, the number of candidates is significantly reduced. The number of prime numbers less than a given number x can be approximated by $\frac{x}{\ln(x)}$ according to the Prime Number Theorem. For a 1024-bit number, this would be approximately $\frac{2^{1024}}{\ln(2^{1024})}$, which is vastly larger than the number of

primes in the range of 511-512 bit, approximately $\frac{2^{512}}{\ln(2^{512})}$ - $\frac{2^{511}}{\ln(2^{511})}$. This reduction from 1024 bits to 512 bits in the search range means going from considering around 10^{308} potential candidates to around 10^{153}. This dramatic decrease in the number of candidates makes the factorization process far more efficient and feasible.

Accurately identifying the correct range for potential prime factors ensures that computational resources are used effectively. By focusing on the more likely candidates (around $\frac{n}{2}$ bits), the factorization process becomes more manageable and less time-consuming. Instead of wasting computational power on an impractically large set of possibilities, it can be focused on a much smaller and more promising set of candidates. This approach is especially important in cryptograpy, where the efficiency of factorization is related to the security from cryptographic attacks.

4.3 Utilization of Prime Heuristics for Identifying Primes and Composites

Prime heuristics are invaluable for distinguishing between prime and composite numbers. Among the various primality tests available, the Fermat and Miller-Rabin tests stand out as particularly effective strategies. The Fermat primality test is based on Fermat's Little Theorem and can quickly identify non-prime numbers, though it may occasionally produce false positives (Carmichael numbers). The Miller-Rabin test, an enhancement of the Fermat test, is a probabilistic test that significantly reduces the likelihood of false positives. By using these tests in combination, one can achieve a high degree of confidence in the identification of prime numbers, thereby improving the efficiency of both key generation and factorization attacks. Note that this combination is what we used in our programs to find the average prime gap in large bit bounds.

4.4 Superior Performance of Number Sieve Implementations

Implementations of the Number Sieve, such as the Quadratic Sieve (QS) and the Number Field Sieve (NFS), greatly outperform naïve brute-force approaches to factoring RSA moduli. The QS is highly effective for numbers up to around 100 digits, while the NFS is the most efficient algorithm known for factoring very large integers, making it the best method for numbers beyond this range. These sophisticated algorithms leverage advanced mathematical techniques to decompose large composite numbers much faster than simple trial division or basic algorithms. Despite these advancements, 1024-bit RSA cryptography remains robust against contemporary attacks. Current factoring methods, even with the most optimized sieves, still require an impractical amount of time and computational power to break a 1024-bit RSA key, ensuring its security in the face of modern computational capabilities.

4.5 Conclusion

In summary, prime heuristics play a vital role in both the generation and assessment of RSA keys by efficiently identifying prime numbers. Targeting the appropriate range for potential prime factors streamlines the factorization process, while effective primality tests like Fermat and Miller-Rabin enhance the identification of primes. Advanced Number Sieve implementations significantly outperform basic brute-force methods, though

1024-bit RSA remains secure against current attack methodologies and system limitations.

References

[1] M. Agrawal. Primality tests based on fermat's little theorem. Unpublished work.

[2] N. Author. Elliptic curve cryptography (ecc). https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc.

[3] N. Author. How to generate large prime numbers for rsa algorithm, 2022. https://www.geeksforgeeks.org/how-to-generate-large-prime-numbers-for-rsa-algorithm.

[4] R. Baillie et al. Strengthening the baillie-psw primality test. *Mathematics of Computation*, page 1, Jan 2021. Accessed: 20 Feb. 2021.

[5] D. J. Bernstein. Papers. Cr.yp.to, 2024. Accessed: 2024-06-28.

[6] D. Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the American Mathematical Society*, 46:203–212, 1999.

[7] J. Buhler, H. Lenstra, and C. Pomerance. Factoring integers with the number field sieve.

[8] M. Case. A beginner's guide to the general number field sieve.

[9] D. P. Chi, J. W. Choi, J. S. Kim, and T. Kim. Lattice based cryptography for beginners. https://eprint.iacr.org/2015/938.pdf.

[10] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer, Berlin, 1st edition, 2001. chapter 4.2.1.

[11] A. Diab. Development of sieve of eratosthenes and sieve of sundaram's proof. *ArXiv.org*, May 2021. Accessed: 27 June 2024.

[12] I. Goldberg and D. Wagner. Randomness and the netscape browser, December 1996.

[13] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, 1988.

[14] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. *Rev*, 2, 2012.

[15] B. Peterson. Prime number theorem, 1996. https://www.math.ucdavis.edu/ tracy/courses/math205A/PNT_Petersen.pdf.

[16] M. Visser. The second hardy-littlewood conjecture is true, 01 2021.