CS6105 Final Report

Nakkul Sreenivas
ns3kb

December 15, 2014

Prime numbers form the basis of all integer numbers. In a field where information is considered discrete, prime numbers are integral in cracking codes and generating keys. Take, for example, a polyalphabetic cipher. When the cipher was first cracked, it was done by indentifying the key length using patterns between repeated phrases. Factoring the set of lengths between these phrases allowed an adversary to break the cipher and identify the length of the key. Since then, prime generation has become more popular as a means of creating public keys. Multiplying two large prime numbers to create a public key lets the user decrypt an message using the two original factors. Due to so many modern day encryption schemes relying on this method, prime factorization is a powerful tool with many practical applications.

Prime factorization algorithms are seperated into two categories. Category 1 algorithms are dependent upon the size of the smallest prime. Since prime factorization can be applie recurzively or numbers with more than two prime factors, knowing the smallest prime can drastically reduce the size of the number n thereby making the problem easier to solve. Category 2 algorithms are more generaly purpose and are usually more applicable to cryptography. Most prime factorization is done on numbers that are the product of only two primes. More prime numbers increases the chance that a given public key can be factored and thereby decrypted. The one exception to this algorthm is Shor's algorithm which will not be covered in this report, but it is important to note that Shor's algorithm is the only polynomial run time algorithm known for factorizing prime numbers.

The most trivial way to test determine the prime factorization of a number n is to test divisibility by primes less than $\sqrt{n}$. Since algorithm run time is measured in terms of the size of the number n, the upper bound for worst-case run time becomes $O(\sqrt{n})$. This reduces to $O(\sqrt{n}/log(n))$ when using a table to store prime numbers so as to avoid testing composite numbers. Although this method has become obsolete, it introduces the theory of sieves which is a a precedent for the more current algorithms for prime factorization. A sieve is any algorithm that works by eliminating elements within a given set to until all that remains is a set of solutions. This idea was used to create a more efficient algorithm to factoring large primes called the **quadratic sieve**. The quadratic sieve relies on the priniciple of congruence of squares to make educated "guesses". The algorithm searches for number pairs of form

$$p^2 \equiv q^2 \mod n$$

When such a pair is found, the expression can be reduced to give

$$p^2 - q^2 \equiv 0 \mod n$$

. Such a pair guarantees that there are two factors of n such that neither factor $(p-q)$ nor $(p+q)$ is divisible by n. This produces two factors for n that aren't divisible by one another, but additionally, in a case of public key decryption, this produces both the prime factors as they are the only ones that exist. The quadratic sieve algorithm takes an extremely long amount of time to search for these pairs due to the sheer size of the domain for p and q. However, this method is still one of the fastest methods (fastest for integers approximately less than 10^{100} and second fastest method overall).

The fastest method currently in use is the genereic number field sieve (GNFS). The GNFS uses a similar theory to the quadratic sieve, but it has a different algorithm for searching for *smoothnumbers*. An n-smooth number is a number that, in fully prime factorized form, has n factors. The principles of both methods still rely on the congruence of squares method, but the GNFS uses a faster algorithm in its search for smooth numbers. As the name states, the algorithm uses number fields to get a sub-exponential search for smooth numbers. From here the GNFS writes the small numbers as n-vectors, n being the number of primes in its prime factorization, and then uses these vectos as a basis in searching for a linear combination to search for pairs of numbers p and q. However, the GNFS is a complicated algorithm relying on determining the proper coefficients for an properly chosen polynomial. For this reason, the algorithm is both hard to implement and not practical for integers that aren't extremely large. Without a size restriction or a restriction on memory, the GNFS is the most efficient factorization method, but the quadratic sieve is more efficient for smaller numbers and memory restrictions.

The two sieve algorithms discussed above rely solely on the size of the number being factored making them Category 2 general purpose algorithms. However, there exist special Category 1 algorithms that have practical use. Despite the fact that they are heavily reliant on the property of the number itself as not being the product of larger prime numbers, this same liability makes it worthwhile in eliminating large but smooth composite numbers in the sieve simultaneously with the quadratic/GNFS algorithms.One example is Shanks' square form factorization which makes use of a variant of the Fermat Method. The advantage of this method was that it could be performed on a calculator with minimal memory and relative ease. Unlike the other sieve algorithms, Shanks' algorithm could efficiently factor integers with smaller prime factors with a smaller memory requirement.

One of the more recent improvements to the prime factorization problem was related to the Fermat algorithm. Fermat realized that by putting the number n in the form

$$n = x^2 - y2 = (p+q)^2 - (p-q)^2$$

it would be easier to search for numbers p and q that satisfied the equation. However, searching for p and q is not an easy task which is why the new technique estimates these values so as to create a potential solution. Using certain properties of modulus and numerical distance, three mathematicians recently found a way to closely estimate values of p and q. It relies on calculating the MSB's in common between p and q to then see how smooth the two numbers are in relation. This results in an increased likelihood that the two numbers will overlap in a given vector space and thereby eliminates collisions and potential choices that won't work. The optimization has an 81.7

In conclusion, there exist multiple different algorithms to factorize primes based on the various constraints of number size, memory allocation, and properties of the number itself such as the distribution of its prime factors. However, there is no doubt that prime factorization is integral in cryptographic schemes and proving their computational secity. The paper doesn't explore more outdated and obscure methods such as Pollard's Rho algorithm, Dixon's algorithm, and Shor's algorithm for quantum computers, but sieve theory and Fermat's identity for primes are all tools that have been used to bring us closer to potentially cracking the foundation of modern day encrytion.

1. Wu, M., Tso, R., & Sun, H. (2014). On the improvement of Fermat factorization using a continued fraction technique. Future Generation Computer Systems, 30, 162-168.

2. Bahmann, H., & Schatte, P. (2001). Analysis of Some Elementary Algorithms for Prime Factorization. Computing, 66, 91-95.

3. Gries, D., & Misra, J. (1978). A Linear Sieve Algorithm for Finding Prime Numbers. Communications of the ACM, 21(12), 999-1003.