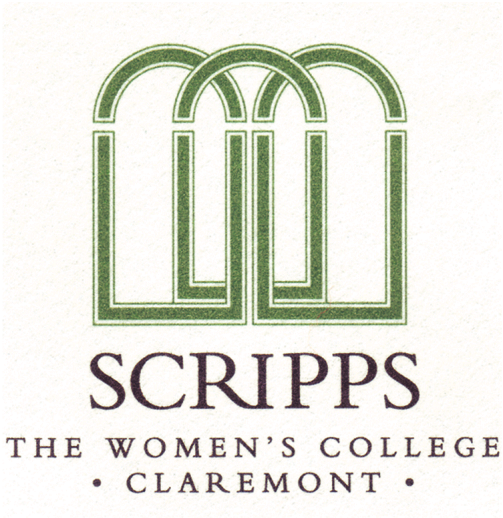

RSA Cryptography

Cracking the Code

Aya Furutani

Professor Yeşem Kurt, Advisor

Professor Christopher Towse, Reader

Submitted to Scripps College in Partial Fulfillment

of the Degree of Bachelor of Arts

April 5, 2006

Department of Mathematics

Abstract

RSA cryptosystems are generally considered secure unless the modulus component of its public key can be factored. This thesis will explore the security of the RSA cryptosystem by examining various factoring algorithms, both general-purpose and special purpose, ultimately building up to a generalized description of the workings of the Number Field Sieve, the fastest factoring algorithm known so far.

Contents

Acknowledgments

My thanks to Professor Towse for his great help and to Professor Kurt for her putting up with me and being a great advisor.

Chapter 1

Mechanics of the RSA Cryptosystem

The main idea behind public key cryptosystems is that the key to encipher a message is made public, so that anyone can send a message. However, only the person who developed the keys to the public cryptosystem will be able to calculate the key to decipher the incoming messages. For the purpose of this thesis, a message is encoded using a bijective function, and the resulting outcome is decoded using the inverse of the function.

The significance of public key cryptosystems is that they depend on an encryption function where the inverse function is very difficult to find. It is because of this that the encoding and decoding a message usually involves big keys that have giant numbers, making the system inefficient compared to, say, private key cryptography, where each pair of people exchanges their private keys and can therefore use a function that is far less time consuming

in its calculations than those of public key encryptions. However, private key cryptography can be cumbersome in the sense that one person has to keep track of everybody's private keys and must use different keys to communicate with different people. Also, in order to agree on a private key, the pair exchanging messages must have a private means of key exchange so that the public can not know the individual's keys. Public key cryptosystems conveniently avoid the necessity of having a private means of key exchange, making it a more secure system in this sense.

One of the most commonly used public key cryptosystems today was invented in 1978 by R.L. Rivest, A. Shamir, and L. Adleman. This system uses the Euler Phi function and Fermat's Little theorem to effectively encode and decode messages. As we will see, the security of the system is dependent on the ability to factor a number which is a product of two primes. Suppose Alice wants to develop an encryption method so that her friend Bob can send her an enciphered message without their nosy neighbor Eve, who is known for her tendency to eavesdrop, to overhear the conversation. So here is how Alice can use RSA cryptography:

1.1 Setting the Keys

Definition 1.1 *For any positive integer n, the* Euler Phi Function, *$\phi(n)$ is a function that maps n to the number of positive integers less than n which are relatively prime to n.*

Definition 1.2 *Two integers a and b are* relatively prime *if the only factors that a and b have in common are 1 and -1. (In other words, their greatest common*

divisor is 1.)

Theorem 1.3 *(Linear Equation Theorem) If two integers a and b have a greatest common divisor of 1, then there is one solution (x, y) to the equation $ax + by = 1$.*

1. First, Alice picks two large prime numbers p and q and multiplies them together: $pq = n$

2. Alice computes the value of the Euler Phi Function of n, which in this case is $\phi(n) = (p-1)(q-1)$. This is because p being a prime means that by definition all the positive integers less than p are relatively prime to p. Also, the same is true with the prime q.

3. Then Alice picks a number k such that $gcd\,(k, \phi(n)) = 1$. By the Linear Equation Theorem, there is one solution (u, v) to the equation

$$ku - \phi(n)v = 1.$$

   This means $ku \equiv 1 \pmod{\phi(n)}$, or u is the inverse of k in the group of invertible elements in $\mathbb{Z}_{\phi(n)}$.

4. Alice makes the information n and k public. These will be the public keys (k, n). Alice keeps the value u secret, since it is the deciphering key, or the private key, and is instrumental in the deciphering of the messsage.

1.2 Encrypting a Message

Definition 1.4 *The numerical representation of letters of a message is called* plaintext*, represented as P, because it has not been enciphered. Once it has been enciphered, it will be* ciphertext*, or C.*

The message that Bob sends Alice, or plaintext P, is enciphered when Bob raises it to the power k modulo n, so Bob calculates $P^k \pmod{n}$. Therefore, the enciphering function for message $P \leq n$ is the map

$$f : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$$

with

$$f(P) \equiv P^k \pmod{n} = C.$$

1.3 Decrypting a Message

Once Alice recieves Bob's enciphered message C, Alice takes it to the uth power modulo n to recover the plaintext. In other words,

$$f^{-1} : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$$

is

$$f^{-1}(C) = C^u \pmod{n} = P.$$

Notice that only Alice knows the value of u, the private key. The deciphering function works because

$$P^k = (C^u)^k = C^{ku} = C^{(1+\phi(n)v)} = C(C^{\phi(n)})^v$$

where $C^{\phi(n)} \equiv 1 \pmod{n}$ by Euler's Formula so

$$P^k \equiv C(1)^v \pmod{n} \equiv C \pmod{n}.$$

Example 1.5 *Suppose Alice says* $A = 11$, $B = 12$, $C = 13$, *and so on up to* $Z = 36$ *is her numerical representation of letters. Alice makes this method of numerical representation public, so Bob will know how to write a message in plaintext. If Bob's message was* CAT, *it would be* 131130 *in plaintext.*

Suppose Alice picked prime numbers $p = 503$ *and* $q = 509$ *so* $n = 503 \cdot 509 = 256027$ *so* $\phi(n) = (503-1)(509-1) = 255016$. *Then suppose Alice choose* $k = 11$ *since* $\phi(n)$ *does not have* 11 *as a factor (try dividing 255016 by 11 to check). Then Alice knows that the equation* $255016u + 11v = 1$ *has one solution for* (u, v), *so solving this by using the* ***Euclidean Algorithm***, *Alice finds that* $255016(4) + 11(-92733) = 1$, *so* $u = 4$ *and* $v = -92733$.

The Euclidean algorithm is an algorithm that runs in $O(\ln(k)\ln(\phi(n)))$ *bit operations when solving for* (u, v) *in* $ku + \phi(n)v = gcd\,(k, \phi(n)) = 1$. *Therefore, the Euclidean Algorithm runs in polynomial time.*

Now Bob can encipher the message by calculating 131130^{11} (*mod* 256027) $\equiv$ 201259 *(courtesy of the Big Number Calculator at http://world.std.com/ reinhold/BigNumCalc.html) So now Bob has the ciphertext*

$C = 201259$, *which he sends to Alice.*

To decipher the ciphertext, Alice simply calculates

$$C^u \ (mod\ n) = 201259^4 \ (mod\ 256027) \equiv 131130 = CAT!$$

1.4 So What Does it Take to Crack The Code?

Suppose that there are eavesdroppers (namely the nosy neighbor Eve) who happen to intercept Bob's message in its encrypted form, in other words, the Eve has somehow acquired the ciphertext. This means that Eve knows $C \equiv P^k \pmod{n}$. Also, because the public key (k, n) is made public, it is assumed that Eve knows the values of k and n. One way for Eve to decipher the message $P \equiv C^u \pmod{n}$ to read the plaintext, is that Eve can calculate the value of u to decipher the encoded message. However, the private key u is the solution to $ku \equiv 1 \pmod{\phi(n)}$.

The information $\phi(n)$ is not made public by Alice so it is unknown to Eve. Therefore Eve must find the values of $p - 1$ or $q - 1$. But in this case, finding $(p - 1)$ and $(q - 1)$ is equivalent to finding the factors p and q since $\phi(n) = (p - 1)(q - 1)$. Conversely, with some rearranging, $\phi(n) = (p - 1)(q - 1) = pq - p - q + 1$ so knowing that $n = pq$ Eve can figure out that $\phi(n) = n - p - q + 1$. However, this is not much help to Eve since Eve now must find the values of p and q in order to find $\phi(n)$. Therefore, finding the values of p and q are equivalent to finding the values $p - 1$ and $q - 1$. So now the security of the cryptosystem is dependent on Eve's ability to factor n into its prime components, p and q. As we shall see, this ability

to factor becomes very difficult for large values of n, especially when the two primes p and q are close in value.

Chapter 2

Some Factoring Algorithms and their Efficiency

The most obvious method of factoring the number n is to simply start dividing n by every prime that is less than $\sqrt{n}$. However, looking at the sheer number of primes between 2 and $\sqrt{n}$ will quickly show that this is a lot of time consuming work, even for numbers that are less than ten digits long.

Theorem 2.1 *(Prime Number Theorem) Let $\pi(m)$ be the number of primes less than or equal to the number m. Then the limit*

$$\lim_{m\to\infty} \frac{\pi(m)}{m/\ln(m)} = 1$$

so $\pi(m) \asymp \frac{m}{\ln(m)}$, or $\pi(m)$ is asymptotically equal to $\frac{m}{\ln(m)}$.

So in other words, if we let $m = \sqrt{n}$, then we can calculate the approximate number of primes that we will have to divide n by. Suppose

that we let $n = 1000000000000$, a number only 13 digits in length. Then $\sqrt{n} = \sqrt{1000000000000} = 1000000$ so $\pi(n) = \frac{n}{\ln(n)} \approx 72382$ meaning that we have to divide n by approximately the first 72382 primes in order to be sure that we find a factor of n. This is clearly not an effective way to factor some large value of n that could potentially be hundreds of digits in length.

2.1 A Note on Time Estimates

The "big O" notation, or $f = O(g)$, is a measure of the complexity of the algorithm. To be more precise, it gives an upper bound on the number of operations an algorithm takes to complete a problem. The formal definition is as follows:

Definition 2.2 *If for some value n_0, two functions $f(n)$ and $g(n)$ where $n \geq n_0$ are defined, take positive values, and for some constant C satisfy the inequality $f(n) \leq Cg(n)$, then $f = O(g)$.*

This is used to approximate the number of bit operations required to perform any given algorithm. An algorithm is said to run in polynomial time if the number of bit operations required is at most $O(k^c)$ where k is the number of bits in the input (which in the case of factoring algorithms is the length of the binary representation of the number to be factored, or in other words, $\log_2(n)$) and c is a constant.

None of the factoring algorithms that have been developed as of yet are in polynomial time. Most perform in exponential time, which means that

the number of bit operations which any given factoring algorithm performs to factor a number with k digits in binary will have a "big O" notation of $O(e^{ck})$ for some integer c. This means that with a big enough value of k, the algorithm could virtually take forever and use up unimaginable amounts of processing power. Currently, RSA Laboratories suggests using a key that is at minimum 1024 bits and for ultra-security, a 2048 bit key is recommended.

The following algorithms in this section are all algorithms in exponential time with the exception of H.W. Lenstra's Elliptic curve method, which factors a number in *subexponential time*. This can be explained as follows:

Definition 2.3 *Let n be a large positive integer which is input into an algorithm. Also, let γ be a variable such that $\gamma \in [0,1]$ and let c be a constant such that $c > 0$. Finally, let*

$$L_n(\gamma; c) = O(e^{c((\ln(n))^{\gamma}(\ln(\ln(n)))^{1-\gamma})}).$$

When this function is applied to an integer n, this is called a $L(\gamma) - algorithm$, which serves as a time estimate $L_n(\gamma; c)$ for a constant c.

If an algorithm runs in polynomial time, it is an $L(0) - algorithm$ which means that $L_n(o; c) = O(e^{c(\ln(\ln(n)))}) = O((\ln(n))^c)$. Also, if an algorithm runs in exponential time, then it is an $L(1) - algorithm$. This means that $L_n(1; c) = O(e^{c(\ln(n))}) = O(n^c)$ However, if $\gamma < 1$, then the algorithm is considered to run in *subexponential time*.

2.1.1 General-Purpose vs Special-Purpose Factoring Algorithms

General-purpose factoring algorithms have running times based only on the size of n, the number being factored. They are mainly built around ideas involving the congruence of squares and are more commonly used for RSA. The examples examined in this paper are:

- Dixon's Algorithm
- The Quadratic Sieve
- The Multipolynomial Quadratic Sieve
- The General Number Field Sieve

Special-purpose factoring algorithms have running times based on known properties of the factors, p and q of a number n. For instance, there are algorithms in which p must have a special form, such as in Pollard's $p-1$ method, where the factor p must be such that $p-1$ is divisible by small primes. Also, in a method such as trial division, the running time is based on the approximate size of a factor p of n. The special-purpose algorithms discussed in the rest of this thesis include:

- Trial Division
- Pollard's Rho Algorithm
- Pollard's $p-1$ Algorithm
- Fermat's Factorization Method
- Lenstra's Elliptic Curve Factorization Method

- The Special Number Field Sieve

2.2 Pollard's Rho Method

The Rho method, or the ρ method is a factoring algorithm which is best used when factoring composite numbers with small factors.

2.2.1 Pollard's Rho Algorithm

1. To factor a composite number n, choose a polynomial $f : \mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z}$ which is not one-to-one. This is usually a polynomial such as $f(x) = x^2 + 2$.

2. Choose an $x_0 = x$ for some integer x.

3. Let $x_{i+1} = f(x_i)$ for $i = 0, 1, 2, \ldots$.

4. Compute the $gcd\,(x_j - x_k, n)$ until there is a solution m such that $1 < m < n$. Then m is a factor of n and the algorithm can be terminated. Otherwise, choose a different polynomial f and point x_0 and repeat the algorithm.

   If the process encounters the problem of $gcd\,(x_j - x_k, n) = n$, then $x_j = x_k$, and, according to Floyd's cycle-finding algorithm, the process will repeat, so in other words, the values x_j and x_k will be ones that have already been encountered, therefore the algorithm should be restarted with a new function f.

In this method, the goal is to map $\mathbb{Z}/n\mathbb{Z}$ to itself. Supposing that p divides n, we find a value x_k such that for some $j < k$, the value $x_j \equiv x_k \pmod{p}$.

The ρ method has a high probability of finding a factor of n in

$$O(e^{4\sqrt{n}ln^3(n)})$$

bit operations. This method is much more efficient than trial division, but there are more efficient algorithms such as the $p-1$ method.

2.3 Pollard's $p-1$ Method

Definition 2.4 *An integer n is considered* smooth *if it has small factors. An integer n is $B-smooth$ if all the prime factors of n are less than some bound B. Then B is the* smoothness bound *of n.*

Although Pollard's $p-1$ method is faster than factoring with Pollard's Rho method, it does have the disadvantage that it will find a factor p of a composite number of the form $n = c^k - 1$ if k is divisible by $p-1$ and if $p-1$ is smooth, that is, if $p-1$ has small prime factors.

2.3.1 Pollard's $p-1$ Algorithm

1. To factor a composite number n, choose a smoothness bound B Calculate $B!$, the product of all the integers less than B.

2. compute $2^{B!} \equiv a \pmod{n}$.

3. If $gcd\,(a-1,n)=p$ where $1<p<n$, then p divides n.

4. If not, then use **successive squaring**, which is an algorithm with a running time of $O(\ln(n))$, to compute $a_i \equiv a_{i-1}^{i} \pmod{n}$ starting at $i=2$ and use this value to compute $gcd\,(a_i-1,n)$ until a factor is found. Because successive squaring runs in polynomial time, this step should not take very long to complete.

Theorem 2.5 *(Fermat's Little Theorem) If p is a prime, than for any integer n, $n^p \equiv n$ (mod p). In particular, if n is not divisible by p, then $n^{p-1} \equiv 1$ (mod p).*

Suppose $p|n$. If $(p-1)|B!$, then $B! = (p-1)k$ for some integer k. Because we have calculated $a \equiv 2^{B!} \pmod{n}$ and since we supposed that $p|n$, it follows that $a \equiv 2^{B!} \pmod{p}$. By Fermats Little Theorem, $2^{p-1} \equiv 1 \pmod{p}$, so $a \equiv 2^{B!} = 2^{(p-1)k} \equiv 1 \pmod{p}$. Then $p|a-1$ and $p|n$, which means $p|gcd\,(a-1,n)$. Note that this whole algorithm is dependant on the group $\mathbb{Z}/p\mathbb{Z}$ having an order divisible by a small prime q in the smoothness bound B. The next factoring algorithm will remedy the problem of being limited to one group.

Pollard's $p-1$ algorithm can be performed in

$$O(B\ln(B)(\ln(n))^2 + (\ln(n))^3)$$

bit operations.

2.4 Elliptic Curve

The Elliptic Curve method of factorization is one of the fastest methods aside from the number field sieve. Mathematician Hendrik Willem Lenstra, Jr. developed the elliptic curve method of factorization in 1985 when making a variation of Pollard's $p-1$ method. If a field K has a characteristic greater than 3, then an elliptic curve over K has the form $E : y^2 = x^3 + ax + b$. The rational points on an elliptic curve, together with the operation of "addition" form an abelian group. Lenstra took advantage of this fact to create his elliptic curve method.

2.4.1 Elliptic Curve Addition

Definition 2.6 *The "point at infinity", or O, is the additive identity of points on an elliptic curve, and any line perpendicular to the x-axis (which is therefore a tangent line with an undefined slope), is said to pass through O.*

If a, b, and c are integers, then for any two rational solutions $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ can be "added" in the sense that a third rational point on the elliptic curve can be found by using the equation of the line through the two known points to find a third point on the line which intersects with the curve. This point reflected across the x-axis will give a new rational coordinate $P_1 + P_2 = P_3 = (x_3, -y_3)$. The point P_3 can be calculated using the coordinates of P_1 and P_2 in the following manner:

$$P_3 = \left(\left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2 \, , -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1} \right)(x_1 - x_3) \right).$$

Furthermore, a rational point $P = (x, y)$ can be added to itself by using the derivative of the elliptic curve at $P = (x_0, y_0)$ to find the slope of the line which is tangent to the elliptic curve at that point, finding its intersection with E and, as above, reflecting the point of the intersection about the x-axis to get the new rational coordinate $2P = (x_1, y_1)$. This point $2P$ can be calculated from the coordinates of P as follows:

$$2P_0 = \left(\left(\frac{3x_0^2 + a}{2y_0}\right)^2 - 2x_0\ , -y_0 + \left(\frac{3x_0^2 + a}{2y_0}\right)(x_0 - x_1)\right).$$

Note that if the "point at infinity" is added to any point P, then $P+O = P$ since O is the additive identity.

2.4.2 Reducing an Elliptic Curve Modulo n

TThe equation of the elliptic curve $x^3 + ax + b$ when reduced modulo n will have coefficients a and b modulo n. A point $P = (\frac{x_i}{y_i}, \frac{x_j}{y_j})$ on the elliptic curve reduced modulo n is $(\frac{x_i}{y_i} \pmod n, \frac{x_j}{y_j} \pmod n)$

For any two rational numbers $\frac{x_i}{y_i}, \frac{x_j}{y_j}$, if the denominators x_j and y_j are relatively prime to n and if the numerator of the fraction $\frac{x_i}{y_i} - \frac{x_j}{y_j} = \frac{x_i y_j - x_j y_i}{y_i y_j}$ is a factor of n, then $\frac{x_i}{y_i} \equiv \frac{x_j}{y_j} \pmod n$.

This means that for any rational number $\frac{x_i}{y_i}$ with $gcd\,(y_i, n) = 1$, one can calculate a unique fraction $\frac{x_j}{y_j} \in [0, n-1]$ such that $\frac{x_i}{y_i} \equiv \frac{x_j}{y_j} \pmod n$ by solving for x_j and y_j in the equivalence $x_i y_j - x_j y_i \equiv 0 \pmod n$. Because it is assumed that any rational number $\frac{x_i}{y_i}$ is reduced to its lowest form, so the values of x_i and y_i are relatively prime. This can be solved because by the linear equation theorem, there is one solution (x_j, y_j) to the equation $x_i y_j -$

$x_j y_i = 1$, where (x_j, y_j) can be calculated using the Euclidean Algorithm.

For the factoring algorithm, Lenstra looks at the elliptic curve of the form $E : y^2 = x^3 + ax + b$ with $a, b \in \mathbb{Z}$. If we choose some random values for a, x, and y in some range, there is a high probability that the cubic will have distinct roots. In other words, we have a high probability of choosing an elliptic curve where the discriminant of the equation $x^3 + ax + b$, which is $4a^3 + 27b^2$ is not equal to 0. Therefore, we assume that $x^3 + ax + b$ has only distinct roots modulo p if p is a prime which divides n. This means that $4a^3 + 27b^2$ must be relatively prime to n. If not, then we have found a factor of n and the case is closed. So suppose we choose a, x, and y such that $gcd\,(4a^3 + 27b^2, n) \neq 1$. We will find the following proposition handy:

Proposition 2.7 *(Modular Group Law) Let E be an elliptic curve $E : y^2 = x^3 + ax + b$ with integer values a and b such that $gcd\,(4a^3 + 27b^2, n) = 1$. Let $P_1 = (\frac{x_i}{y_i}, \frac{x_j}{y_j})$ and $P_2(\frac{x_k}{y_k}, \frac{x_\ell}{y_\ell})$ be two rational points on E with*

$$gcd\,(y_i, n) = gcd\,(y_j, n) = gcd\,(y_k, n) = gcd\,(y_\ell, n) = 1.$$

Suppose that $P_1 + P_2 \neq O$.

Then $P_1 + P_2 = P_3 \in E$ where $P_3 = (\frac{x_c}{y_c}, \frac{x_d}{y_d})$ such that $gcd\,(y_c, n) = gcd\,(y_d, n) = 1$ if and only if there is no prime $p|n$ such that the sum $P_1 + P_2 \equiv O\ (mod\ p) \in E\ (mod\ p)$.

2.4.3 Lenstra's Elliptic Curve Algorithm

1. Choose random integers a, b, and y for the elliptic curve $E : y^2 = x^3 + ax + b\ (\text{mod}\ n)$. Check that $4a^3 + 27b^2 \neq 0$ and that $gcd\,(4a^3 +$

$27b^2, n) = 1$. If the greatest common divisor is $\geq n$, then choose a different elliptic curve. If the greatest common divisor is between 1 and n, then we have found our factor.

2. Choose a rational point $P = (x, y)$ on E so that we have selected our random pair (E, P)

3. Choose some positive integer $k \in \mathbb{N}$ such that for bounds $A, B \in \mathbb{N}$,

$$k = \prod_{i=1}^{\ell} p_i^{\alpha_{p_i}}$$

where p_i are small primes less than the bound B, so $p_1, p_2, p_3, \ldots, p_\ell \leq B$ and $\alpha_{p_i} = \left[\frac{\ln(A)}{\ln(p_i)} \right]$ is the largest exponent such that $p_i^{\alpha_{p_i}} \leq A$.

Note that the larger the bound B, the greater the chances are that $kP \pmod p = O \pmod p$ and we will find the prime $p|n$. However, the larger bound B also means that computing $kP \pmod p$ will take much longer. Therefore, find an optimal value for B based on computing time.

4. Compute kP. This is easier if **repeated doubling** is used, which means starting by computing $2P$, then $2(2P)$, $2(2^2P)$, and so on up to $2^{\alpha_2}P$ and then move on to $3(2^{\alpha_2}P)$, $3(3 \cdot 2^{\alpha_2}P)$, $\ldots$, $3^{\alpha_3}2^{\alpha_2}P$, and so on until we reach

$$\prod_{i=1}^{\ell} p_i^{\alpha_{p_i}} P = kP.$$

Repeated doubling is much like the successive squaring method discussed in Pollard's $p - 1$ algorithm and has the same running time

estimate, making these calculations fairly quick to solve.

During each step of computing kP, the new point k_iP is calculated modulo n. If a point $k_iP = (\frac{x_i}{y_i}, \frac{x_j}{y_j})$ has $gcd\,(y_i, n) = (y_i, n) = 1$, then it can be reduced modulo n. If the gcd is not 1, then by the modular group law, there is a prime $p|n$ such that $k_iP \equiv 0 \pmod{p}$.

In other words, we found a k_i that is a multiple of the order of the element $P \pmod{p}$ in the group $E \pmod{p}$. If no such k_i is found, then simply choose a new pair (E, P).

We set k to be the product of small primes less than bound B because we hope that this will increase the chance that k is a multiple of of the order of $P \pmod{p}$ in the group $E \pmod{p}$. We can add more factors to k by increasing the bound B since k is the product of all the primes less than B, however, this would increase the number of operations needed to compute kP.

Lenstra's method is classified as special-purpose factoring algorithm because the running time is substantially faster if a factor of n is $p < \sqrt{n}$. However, it is considered the third fastest method for general-purpose factoring.

The Elliptic curve method is generally used in conjunction with other factoring algorithms to factor auxiliary numbers, namely those with smaller prime factors for the reason mentioned above. The largest prime factor found using the ECM as of April in 2005 had 66 decimal digits. The factor of $3^{466} + 1$ was found by B. Dodson on Apr. 6, 2005.

Lenstra's Elliptic Curve Method of factorization factors a number in

$$O(e^{(1+o(1))\sqrt{(2\ln(p)\ln(\ln(p)))}})$$

bit operations, which means that it is an $L(\frac{1}{2}) - algorithm$.

Chapter 3

Developing the Number Field Sieve

The Number Field Sieve is a method of factoring which evolved from the following methods of factorization. Starting with Fermat factorization, or factoring based on the idea of quadratic residues, faster and more efficient methods were built on this idea. We start with Dixon and his method using factor bases, then move to the quadratic sieve and the concept of sieving added to the method of factor bases, and finally we add multiple polynomials to our quadratic sieve.

3.1 Fermat Factorization

Fermat factorization is efficient at factoring a composite number n if n is a product of two integers which are close in value. It is based on the idea that quadratic residues, or determining which numbers are squares modulo n,

will allow us to factor the sum of squares modulo n and end up with a divisor of n.

3.1.1 Fermat's Method

For any number n which can be written as the product of two integers a and b, we can manipulate the equation $ab = n$ in the following fashion:

$$\begin{aligned} n &= ab \\ &= \frac{(a^2 - a^2 + b^2 - b^2 + 2ab + 2ab)}{4} \\ &= \frac{\Big[(a^2 + 2ab + b^2)(a^2 - 2ab + b^2)\Big]}{4} \\ &= \left[\frac{(a+b)}{2}\right]^2 \left[\frac{(a-b)}{2}\right]^2 \end{aligned}$$

Let $t = \frac{(a+b)}{2}$ and $s = \frac{(a-b)}{2}$ rewriting a and b in terms of s and t, $a = t+s$ $b = ts$ This way $n = t^2 - s^2 = (t+s)(t-s) = ab$. Here is the main idea behind Fermat factorization:

Definition 3.1 *A number a is said to be a quadratic residue modulo n if there is a number b less than n such that $a \equiv b^2 \pmod{n}$.*

We know that $n = t^2 - s^2$ which means that n divides $t^2 - s^2$. In other words, $t^2 - s^2 \equiv 0 \pmod{n}$ or $t^2 \equiv s^2 \pmod{n}$

This means that if a square is congruent to another square modulo n, and if $t \not\equiv \pm s \pmod{n}$, then we have found factors of n!

If the factors a and b happen to be close together, then $s = \frac{a-b}{2}$ will be a small number and $t = \frac{a+b}{2}$ will be a number that is close to $\sqrt{n}$. Using this fact, the number n can be factored efficiently if the values for t are gathered close to the square root of n. So let $t = [\sqrt{n}\,] + 1, [\sqrt{n}\,] + 2, [\sqrt{n}\,] + 3, \ldots$ and so on until $t^2 - s^2 = n$. If this does not prove to be fruitful within a reasonable number of t, then perhaps the factors a and b are not so close together so the process can be modified so that $t = [\sqrt{kn}\,] + 1, [\sqrt{kn}\,] + 2, [\sqrt{kn}\,] + 3, \ldots$ and so on for $k = 0, 1, 2, \ldots$.

3.2 Dixon's Method

Dixon's method is a method based on Fermat Factorization, and uses factor bases to develop an algorithm on which the quadratic sieve is based.

Definition 3.2 *A factor base is a set $B = \{p_1, p_2, \ldots, p_r\}$ of distinct primes p_i, usually including $p_1 = -1$.*

For Dixon's method, the optimal primes in B will be less than or equal to the closest integer value of $e^{\sqrt{(\log(n)\log(\log(n)))}}$. In other words, they will be of the order of the approximate running time of an $L(\frac{1}{2}) - algorithm$.

Definition 3.3 *If the least absolute residue $b^2 \pmod n$ of a number b can be written as a product of the numbers from the factor base B, then b is a $B - number$.*

3.2.1 Dixon's Algorithm

1. Choose a bound P, a prime, and create factor base $B = \{p_1 = -1, p_2, p_3, \ldots, p_r\}$ containing distinct primes $p_j < P$ and the number -1. Note that if P

is large, then there will be a lot of primes p_j in the factor base which will make the process more tedious and more computationally expensive in the sense that the matrix in the following steps will be bigger and harder to work with, however, it will allow for a greater number of $B - numbers$ to be found. Therefore, find an optimal P for the computational power that is available to run the algorithm.

2. Find several $B - numbers$ by taking values $b_i = \lceil \sqrt{kn} \rceil + c$ for $c = 0, 1, 2, \ldots$ and $k = 0, 1, 2, \ldots$, squaring the values, and reducing modulo n. Test to make sure that these values of b_i are indeed $B-numbers$ by making sure that b_i^2 is the product of primes in the factor base B, and discard all the values of b_i that are not $B - numbers$.

3. Write the power of the prime factors of b_i^2 as vectors in $r-$dimensional space (Where r is the number of elements in B) and reduce them modulo 2 (or assign 1 to odd numbers and 0 to even numbers). In other words, a vector for a $B - number\ b_i$ where

$$b_i^2 \pmod{n} = p_1^{\alpha_{i_1}} p_2^{\alpha_{i_2}} p_3^{\alpha_{i_3}} \ldots p_r^{\alpha_{i_r}}$$

will have the form

$$\omega_i = (\alpha_{i_1}, \alpha_{i_2}, \alpha_{i_3}, \ldots, \alpha_{i_r}) \pmod{2}.$$

4. By adding these vectors ω_i in modulo 2, we can determine which $B - numbers$ multiplied together will give a product that is a square in modulo n. (They will be all zero vectors because summing the vec-

tors is equivalent to multiplying the primes in the basis since we have added their powers together.) This is more easily done if the vectors are written as a matrix. Since the matrix will be reduced, a system having linear dependance would guarentee a zero row, or in other words, there will be a linear combination of r-dimensional vectors which will produce a zero vector $(\text{mod } 2)$ if the matrix has m different $B - numbers$ represented where $m > r$. The matrix will be m by r dimensional and can be reduced using **Gaussian Elimination**. The Gaussian Elimination algorithm runs in $O(r^3)$ bit operations for a matrix that has dimention $r \times r$. For a relatively small r, it will not take long to find the linear combination of vectors that will produce a zero vector, giving the desired $B - numbers$ whose product is a square modulo n.

The matrix will be as follows:

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \vdots \\ \omega_m \end{pmatrix} = \begin{pmatrix} \alpha_{1_1} & \alpha_{1_2} & \alpha_{1_3} & \dots & \alpha_{1_r} \\ \alpha_{2_1} & \alpha_{2_2} & \alpha_{2_3} & \dots & \alpha_{2_r} \\ \alpha_{3_1} & \alpha_{3_2} & \alpha_{3_3} & \dots & \alpha_{3_r} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_{m_1} & \alpha_{m_2} & \alpha_{m_3} & \dots & \alpha_{m_r} \end{pmatrix}.$$

5. Once the combination of vectors is found, simply multiply their respective $B - numbers$ to get a square number c^2 and reduce modulo n to get a number k^2 so that $c^2 - k^2 \equiv 0 \ (\text{mod } n)$. Check to see that $c \not\equiv \pm k \ (\text{mod } n)$ so that the factors will not be trivial. This means that

$(c+k)(c-k) \equiv 0 \pmod{n}$ so $c+k$ and $c-k$ are both factors of n.

3.3 Quadratic Sieve

The quadratic sieve method of factorization was developed by Carl Pomerance in 1981. It combines the idea of a sieve with Dixon's method of factorization. There are several variations of the sieve, as we will later see with the multipolynomial quadratic sieve. The idea of a sieve is central to the development of the number field sieve, which is, in fact, a generalization of the quadratic sieve method.

3.3.1 Sieve of Eratosthenes

The concept of *sieving* was developed long ago and was famously used by Greek mathematician Eratosthenes to find all the prime numbers less than 1000. Eratosthenes started with 2, the smallest prime, and removed all of the multiples of 2 from the set $S = \{1, 2, 3, \ldots, 998, 999, 1000\}$. Next, Eratosthenes focused on the next prime, 3, removing all of its multiples from S. Continuing in this fashion, all multiples of the primes "fell through" the sieve and Eratosthenes was left with the set of all the primes less than 1000.

The quadratic sieve works best when the composite number n has factors that are close to $\sqrt{n}$. If the factors are significantly less than $\sqrt{n}$, then Lenstra's Elliptic Curve method works faster than the quadratic sieve.

The main idea behind the sieving in this algorithm deals with the inconvenience in Dixon's algorithm of finding $B - numbers$ by dividing the numbers b_i in the set $\left\{\sqrt{kn} + c \mid k, c = 0, 1, 2, \ldots\right\}$ by each prime in the fac-

tor base B. Rather than going through this tedious process, Pomerance suggests simultaneously testing all the numbers in the set S of possible $B-numbers$ for divisibility by the primes $p \in B$ using a sieve. In this case, the set of possible $B-numbers$ will be $S = \{t^2 - n \mid \lceil\sqrt{n}\rceil + 1 \leq t \leq \lceil\sqrt{n}\rceil + A\}$ for an optimally chosen bound A.

3.3.2 Pomerance's Quadratic Sieve Method

Definition 3.4 *If a is a quadratic residue modulo n, so in other words, if $a \equiv b^2 \pmod{n}$ for some number $b < n$, then the* Legendre Symbol $(\frac{a}{n}) = 1$.

1. Choose bounds A and P in the following fashion: Both the positive integer A and the prime P should approximately be of order of magnitude $e^{\log(n)\log(\log(n))}$ and A should be a number between P and a small power of P, so typically $P < A < P^2$. Then the factor base B will be the set of primes $B = \left\{p \leq P \mid (\frac{n}{p}) = 1\right\}$. It is important to pick primes such that n is a quadratic residue because the objective is to find a value t such that

$$t^2 \equiv p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_r^{k_r} \pmod{n}$$

for r number of primes in the factor base. This means that

$$t^2 - p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_r^{k_r} = cn$$

for some constant c. Rearranging this equation to

$$t^2 - cn = p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_r^{k_r}$$

and taking this equation modulo p_i for any $p_i \in B$ will mean that

$$p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_r^{k_r} \equiv 0 \pmod{p_i}$$

so the equivalence relation is $t^2 - n \equiv 0 (\text{mod } p_i)$. Because we will look at the divisibility of the difference of the terms $t^2 - n$ by the primes in the factor base B, it is important that n is a quadratic residue for each prime in the factor base B.

The $B - numbers$ will be chosen from the set

$$S = \left\{ t^2 - n \mid [\sqrt{n}\,] + 1 \leq t \leq [\sqrt{n}\,] + A \right\}.$$

2. Calculate $t^2 - n$ for

$$t = [\sqrt{n}\,] + 1, [\sqrt{n}\,] + 2, [\sqrt{n}\,] + 3, \ldots, [\sqrt{n}\,] + A$$

and list these values in a column.

3. Excluding the case where $p = 2$ (which will be dealt with later), calculate $t^2 \equiv n \pmod{p_i^{\beta}}$ for $p_i \in B$ where $\beta = 1, 2, 3, \ldots$ until there are no values of t for

$$[\sqrt{n}\,] + 1 \leq t \leq [\sqrt{n}\,] + A$$

such that there is a solution to $t^2 \equiv n \pmod{p_i^\beta}$. Let β_0 be the largest β value before $t^2 \equiv n \pmod{p_i^\beta}$ has no solutions.

4. Take two solutions t_1 and t_2 to $t^2 \equiv n \pmod{p_i^{\beta_0}}$ not necessarily in the range $\lceil\sqrt{n}\rceil + 1 \leq t \leq \lceil\sqrt{n}\rceil + A$. Let p_i be a row in the table where we already listed the values of $t^2 - n$ in the first column. Under p_i, write 1 if the t in $t^2 - n$ differs from t_1 by a multiple of p_i. If t differs from t_1 by a multiple of p_i^2, then change the 1 to a 2, and if the difference between t and t_1 is a multiple of p_i^3, then change the entry to a 3, and continue up to $p_i^{\beta_0}$. Note that the largest possible entry will be β_0. Then repeat the entire process using t_2 in place of t_1.

5. During the previous step, if a certain entry was changed from 1 to 2 to 3 up to k for some $k \leq \beta_0$, then for the corresponding t, divide $t^2 - n$ by p_i^k and record this value. When this process is repeated with the next prime $p_j \in B$, the entry for the same t was changed to $c \leq \beta_0$, then the quotient of $t^2 - n$ and p_i^k is divided by p_j^c. (This works because if $t^2 - n$ is divisible by p_i^k, then the quotient should be divisible by p_j^c since p_i and p_j must be relatively prime.)

6. Back to the case where $p = 2$, there are three possibilities to consider.

   (a) If $n \not\equiv 1 \pmod{8}$, then if t is odd, t^2 must also be odd, which would make the difference $t^2 - n$ even since it is assumed that n is the product of two odd primes, and must therefore be odd. Since $t^2 - n$ is even, it is divisible by 2 so the entry for the t column is 1 and $t^2 - n$ is divided by 2.

(b) In the case where $n \not\equiv 1 \pmod 8$ and t is even, the difference $t^2 - n$ must be odd, so the entry remains 0.

(c) If $n \equiv 1 \pmod 8$, then $n = 8k + 1$ for some integer k. The value $8k + 1$ can be substituted into the equation $t^2 \equiv n \pmod{2^{\beta_0}}$ so the equation then becomes $t^2 \equiv 8k + 1 \pmod{2^{\beta_0}}$. But this means that if β_0 is 1, then $t^2 \equiv 1 \pmod 2$ so the only possible solution for t is 1. If β_0 is 2, then the equation becomes $t^2 \equiv 1 \pmod 4$ so the solutions will be $t_1 = 1$ and $t_2 = -1$ (since $3 \equiv -1 \pmod 4$). However, if $\beta_0 = 3$, then the equation $t^2 \equiv 1 \pmod 8$ has solutions $t_1 = 1$, $t_2 = -1 = 7$, $t_3 = 3$, $t_4 = -3 = 5$. Suppose that $\beta_0 > 3$ and suppose that $t^2 \equiv n \pmod{2^{\beta_0}}$ where $n \equiv 1 \pmod 8$ has a solution t_0. Then $t_0^2 = n + c2^{\beta_0}$ for some integer c. Because n is odd and $c2^{\beta_0}$ is even, we know that t_0^2 is odd. Then there is a unique solution s_0 to the equation $t_0 s \equiv -c \pmod 2$. Let $t_1 = t_0 + s_0 2^{\beta_0 - 1}$. Then

$$t_1^2 = (t_0 + s_0 2^{\beta_0 - 1})^2 = t_0^2 + t_0 s_0 2^{\beta_0} + s_0^2 2^{2\beta_0 - 2}$$

And since $t_0^2 = n + c2^{\beta_0}$, we can substitute to get

$$t_1 = n + (c + t_0 s_0)2^{\beta_0} + s_0^2 2^{2\beta_0 - 2}.$$

Because s_0 satisfies $t_0 s \equiv -c \pmod 2$, we know that $2|(c + t_0 s_0)$, which means that $2^{\beta_0+1}|(c+t_0 s_0)2^{\beta_0}$ so therefore $t_1^2 \equiv n \pmod{2^{\beta_0+1}}$. Thus by induction, there are four solutions to the equation $t^2 \equiv$

$n \pmod{2^{\beta_0}}$ for $n \equiv 1 \pmod 8$ when $\beta_0 \geq 3$.

Therefore, solve for t in the equation $t^2 \equiv n \pmod{2^{\beta_0}}$. If $\beta_0 \geq 3$, then there are four solutions t_1, t_2, t_3, and t_4. Repeat the process performed with the odd primes in B with these solutions.

7. After the table is complete for all values $p_i \in B$, discard all of the entries t unless $t^2 - n$ divided by all the various powers of primes in the factor base is 1. Note that this means $t^2 - n$ are $B - numbers$ and the other columns show the values of $p \leq P$ such that n is a quadratic residue.

8. The rest of the procedure is exactly the same as Dixon's method; form a matrix from the vectors of the product of the powers of primes for each of the $B-numbers$ and use Gaussian Elimination to find a linear combination of vectors that will produce a zero vector. Then find the product of these $B - numbers$ raised to their respective powers modulo n and hope to get a congruence $c^2 \equiv k^2 \pmod{\text{n}}$ such that $c \not\equiv k \pmod n$. Note that the chances of finding such a combination of vectors is greatly increased from Dixon's method due to the fact that the process of selection of $B - numbers$ allows for a greater number of them to be found.

The Quadratic Sieve is an algorithm that can be performed in

$$O(e^{(1+o(1))\sqrt{(\ln(n)\ln(\ln(n)))}})$$

bit operations, where the "little o" notation, $o(1)$ is a function of n as $n \rightarrow$

∞, and p is the smallest prime factor of n. In other words, the Quadratic Sieve is an $L(\frac{1}{2})-algorithm$. The largest number factored using this method was done in April of 1994 when a 129 digit number, $RSA-129$, was factored into two primes, one with 64 digits, and the other with 65.

3.4 Multipolynomial Quadratic Sieve

This is a variation of Pomerance's Quadratic Sieve which, as the name suggests, uses multiple polynomials rather than the one polynomial, $f(t) = t^2 - n$ used in the previous method. The advantage in this method over the previous one is that by using several polynomials $W_k(x)$, the values of the input x (which we called t in the previous problem), can be taken from a smaller interval, so the average value of the polynomials used in this method is less than the value of the polynomial of the previous method. This is significant because the probability that these polynomials will factor over small primes is much greater than the values $t^2 - n$ from the previous method.

3.4.1 Multipolynomial Quadratic Sieve Method

1. To factor a composite number n, start by choosing a large smoothness bound B and a number $M \in \mathbb{N}$ such that $\left(\frac{\sqrt{2n}}{M}\right)^{\frac{1}{4}} > B$.

2. Choose a prime P and form a factor base in the same way as the previous method, so the base would consist of ℓ number of primes p_i in the set $F = \left\{p_i \leq P \mid (\frac{n}{p_i}) = 1 \text{ for } i = 1, 2, 3, \ldots, \ell\right\}$.

3. Let $q_i = p_i^{\alpha_i}$ for some α_i such that $q_i = p_i^{\alpha_i} < B$. Then solve the equation $t_{q_i}^2 \equiv n \pmod{q_i}$ for t_{q_i} such that $0 < t_{q_i} < \frac{q_i}{2}$.

4. Choose a polynomial $W_k(x) = a_k^2 x^2 + 2b_k x + c_k$ where the coefficients a_k, b_k, and c_k are chosen in the following manner:

$$a_k^2 \approx \frac{\sqrt{2n}}{M}, \quad b_k^2 - n = a_k^2 c_k, \quad |b_k| < \frac{a_k^2}{2}.$$

This can be done in the following fashion:

(a) Choose numbers $r, s \in \mathbb{N}$ such that $1 < s < r$. Usually in practice r is chosen to be around 30, but this is quite large if the objective is to test the algorithm on a small composite number.

(b) **Definition 3.5** *For primes $g_1, g_2, g_3, \ldots, g_r$, if*

i. $g_i \approx \left(\frac{\sqrt{2n}}{M}\right)^{\frac{1}{2s}}$,

ii. $(\frac{n}{g_i}) = 1$, *and*

iii. for each $i = 1, 2, 3, \ldots, r$, $gcd\,(g_i, q) = 1$ for all $q \in F$,

then they are known as g-primes.

Theorem 3.6 *(Chinese Remainder Theorem) If $gcd\,(m_i, m_j) = 1$ for $i \neq j$ in the congruences*

$$x \equiv a_1 (mod\ m_1),$$

$$x \equiv a_2 (mod\ m_2),$$

$$\ldots$$

$$x \equiv a_r (mod\ m_r),$$

then there is a simultaneous solution x to all r congruences.

Note that this theorem depends on solving the congruences using the Euclidean algorithm, an algorithm whose running time was discussed previously in the first chapter.

(c) For $1 \leq i_1 < i_2 < \ldots < i_s \leq r$, let $a = g_{i_1} g_{i_2} g_{i_3} \cdots g_{i_s}$.

(d) Solve the congruences $b_{k_i}^2 \equiv n \pmod{g_i^2}$ with $i = 1, 2, 3, \ldots, r$ for b_{k_i}. Then find the value b_k by setting $b_k \equiv \pm b_{k_{i_1}} \pmod{g_{i_1}^2}, b_k \equiv \pm b_{k_{i_3}} \pmod{g_{i_2}^2}, b_k \equiv \pm b_{k_{i_3}} \pmod{g_{i_3}^2}, \ldots, b_k \equiv \pm b_{k_{i_s}} \pmod{g_{i_s}^2}$. This system of equations can be solved for b_k by using the **Chinese Remainder Theorem**.

(e) Once the value for b_k is found, then simply set $c_k = \left(\frac{b_k^2 - n}{a_k^2}\right)$. This method guarantees that the conditions set earlier for selecting the constants a_k, b_k, and c_k are met.

5. **Definition 3.7** *Let $j \in [-M, M]$ and $q_i = p_i^{\alpha_i}$ for $p_i \in F$. If $q_i | W_k(j)$, then j is a sieve number.*

Note that if $q_i | W_k(j)$, then since $b_k^2 - n = a_k^2 c_k$, we know that

$$a_k^2 j^2 + 2 b_k j + c_k = a_k^2 j^2 + 2 b_k j + \frac{b_k^2 - n}{a_k^2}$$

$$= \frac{1}{a_k^2}(a_k^4 j^2 + 2 a_k^2 b_k j + b_k^2 - n)$$

$$= \frac{1}{a_k^2}((a_k^2 j + b_k)^2 - n).$$

Because a_k has already been defined as the product of primes g_i such that $gcd\,(g_i, q) = 1$, it follows that $gcd\,(a_k, q_i) = 1$. Therefore, knowing that q_i can not divide a_k^{-2}, it must be that $q_i|(a_k^2 j + b_k)^2 - n$.

This means that for some $d_1 \in \mathbb{N}$, $d_1 q_i = (a_k^2 j + b_k)^2 - n$. Rearranging this equation we get $\sqrt{d_1 q_i + n} = a_k^2 j + b_k$ and solving this for j, we get $j = \frac{\sqrt{d_1 q_i + n} - b_k}{a_k^2}$. However, since $t_{q_i}^2 \equiv n \pmod{q_i}$, then $t_{q_i}^2 = d_2 q_i + n$ for some constant $d_2 \in \mathbb{N}$. Because we desire to look at the equation in modulo q_i, we can assume that $t_{q_i}^2$ can be substituted for $d_1 q_i + n$, because constants d_1 and d_2 are multiplied by q_i and therefore the terms containing d_1 and d_2 become 0 modulo q_i. Therefore

$$j \equiv a_k^{-2}(\sqrt{t_{q_i}^2} - n) \pmod{q_i}$$

so $j \equiv a_k^{-2}(\pm t_{q_i} - n) \pmod{q_i}$.

Compute $a_k^{-2} \equiv g_{i_1}^{-2} g_{i_2}^{-2} g_{i_3}^{-2} \cdots g_{i_s}^{-2} \pmod{q_i}$. This step is easier ifthe values of $g_i^{-2} \pmod{q_i}$ are calculated when finding the g_i or $g-primes$ to get the value a, for $q_i = p_i^{\alpha_i}$ for $p_i \in F$, .

6. Let $h = (h(-M), h(-M+1), \ldots, h(j), \ldots, h(M-1), h(M))$ be an $(2M+1)$-tuple where initially all values $h_0(j)$ are set to 0 for all $j \in [-M, M]$. However, if a prime power $q_i = p_i^{\alpha_i} | W_k(j)$, then reset the value $h_i(j) = \ln(p_i) + h_{i-1}(j)$. Using the natural log means that there will be fewer entries to keep track of, and in the next step, the values for the prime power q_i which divides W_k(j) can be recalculated.

7. **Definition 3.8** *For $j \in [-M, M]$, the report threshold is the average value of* $\ln |W_k(j)|$.

   In this case, the report threshold will be $RT = \ln\left(\frac{1}{2}M\sqrt{n/2}\right)$. When the value $h(j) \geq RT$, then $W_k(j)$ is considered to be a candidate for factoring over the factor base.

   Therefore, take all values of $h(j)$ that are approximately equal to the report threshold and if W_k factors over F, then save the values a_k, b_k, c_k, and j. Go back to step 4 and repeat the process until there are $\ell + 2$ different polynomials $W_k(j)$.

8. Once there is a collection of $\ell + 2$ different polynomials, then we have $\ell + 2$ different sieve values j, so let

$$W(j) = (-1)^{b_{j_0}} \prod_{i=1}^{\ell} p_i^{b_{j_i}}$$

   and let $b_{j_i} \leq a_i$ for $j = 1, 2, \ldots, \ell + 2$. The exponents of the function $W(j)$ are then put into the vector $\omega_j = (b_{j_0}, b_{j_1}, b_{j_2}, \ldots, b_{j_\ell}) \pmod 2$. This means that there are $\ell + 2$ vectors of $\ell + 1$ dimensions, so there must be at least one case of linear dependence. Just as the former methods, we can make a matrix and, using **Gaussian Elimination**, we can find a linear combination of vectors such that the sum is a zero vector $\pmod 2$, which means that the associated product

$$\prod W(j) \equiv z^2 \pmod n$$

   for some $z \in [-n, n]$.

9. This means that $(a^2x + b)^2 \equiv a^2W(x)(\text{mod } n)$ so just as in Dixon's method and Pomerance's quadratic sieve,

$$X^2 \equiv \prod(a^2j + b)^2 \equiv z^2 \prod a^2 \equiv Y^2(\text{mod } n)$$

and if $1 < gcd\,(X - Y, n) < n$, then that is the non-trivial factor of n.

In August of 2001, a 135 digit number was factored by B. Dodson, A.K. Lenstra, P. Leyland, A. Muffet, and S. Wagstaff using the Multipolynomial Quadratic Sieve. This number was a factor of the Cunningham number $2^{1606} + 1$.

3.5 General Number Field Sieve

The General Number Field Sieve is a generalization of the Special Purpose Number Field Sieve, a factoring algorithm which can factor numbers $n = r^t - s$ for small numbers $r, |s| \in \mathbb{N}$ and large $t \in \mathbb{N}$ with a running time of $L(\frac{1}{3}, (\frac{32}{9})^{\frac{1}{3}})$. As previously mentioned, the general number field sieve is a general-purpose factoring algorithm, and it has the fastest running time of all factoring algorithms, factoring any number n in

$$L(\frac{1}{3}, \frac{64}{9}) = O(e^{(\frac{64}{9}n)^{\frac{1}{3}}(\ln(n))^{\frac{2}{3}}})$$

bit operations. As of yet, the General Number Field Sieve has factored a number with 158 digits, a number which happened to complete the factorization of the number $2^{953} + 1$ by factoring the last composite factor into

a 73-digit prime number and a 86-digit prime number. This was accomplished in 2002 by F. Bahr, J. Franke, and T. Kleinjunj.

3.5.1 The General Idea

In the late 1980's, Pollard started looking into factoring beyond methods dependent on quadratic residues, exploring cubic residues, fifth degree equations, and finally kth degree equations. Thus the concepts behind the Number Field Sieve were developed. The Number Field Sieve is said to be the most complicated factoring algorithm developed as of yet, so the following is only an overview of the basic concepts behind the sieve.

Selecting from experimental data, a monic polynomial f of degree d which is irreducible over $\mathbb{Z}$, alongside with a smoothness bound B, are choosen. Find $m \in \mathbb{N}$ such that $f(m) \equiv 0 \pmod{n}$. This can be done by taking $m = n^{1/d}$ where d is the degree of polynomial f. Then putting n over base m, we have

$$n = m^d + c_{d-1}m^{d-1} + \ldots + c_0$$

where $0 \leq c_i \leq m - 1$ for $i = 0, 1, \ldots, d$. Then if the polynomial f was set such that

$$f(x) = x^d + c_{d-1}x^{d-1} + \ldots + c_0 \in \mathbb{Z}[x],$$

it would mean that $f(m) = n$. Note that if f is reducible in $\mathbb{Z}$, then it factors into two polynomials $f(x) = g(x)h(x)$ such that $f(m) = g(m)h(m) = n$, giving us the factors $g(m)$ and $h(m)$ of n. However, we are assuming that

this is not the case, so we assume also that f is irreducible. Thus we are working with a number field $F = \mathbb{Q}(\alpha)$ of degree d over the field $\mathbb{Q}$. Let $\alpha \in \mathbb{C}$ be a root of f. Then there is a natural homomorphism

$$\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}/n\mathbb{Z}$$

where $\phi(\alpha) = m$. Then for any polynomial $g(x) \in \mathbb{Z}[x]$, we can map $\phi(g(\alpha)) \equiv g(m) (\text{mod } n)$.

The goal is to find a set S of polynomials g such that the product

$$\prod_{g \in S} g(\alpha) = \beta^2$$

where β^2 is an element in $\mathbb{Z}[\alpha]$ and furthermore,

$$\prod_{g \in S} g(m) = y^2$$

where y^2 is an element of $\mathbb{Z}$.

Then solving for x in the equation $\phi(\beta) \equiv x \ (\text{mod } n)$ gives

$$x^2 = \phi(\beta)^2 \equiv \phi(\beta^2).$$

This means that

$$x^2 \equiv \phi\Big(\prod_{g \in S} g(\alpha) \Big) \equiv \prod_{g \in S} g(m) \equiv y^2 \ (\text{mod } n).$$

Therefore, we find ourselves back at Fermat Factorization and the equiv-

alence of two squares modulo n.

In this thesis, we have examined the Integer Factorization Search Problem, (Find the factors of an integer n.) which is equivalent to the Integer Factorization Decision Problem (Given integers n and k such that $k < n$, is there a factor of n less than k?). A decision problem can be proven to be NP and co-NP in terms of the algorithm used to solve the problem. If a problem is NP, then given an unlimited amount of computing power, an answer can be found, and if the answer is "yes", then the answer can be proven to be true. For example, we can prove that any number n has a factor less than k trial division of n by every number less than k until we find a factor since we are allowed unlimited computing power. A decision problem is co-NP if, given unlimited computing power, the answer can also be found, and if the answer is a "no", it can be proven to be correct. For instance, if there aren't any factors of n less than a given integer k, then one can prove with trial division that n is not divisible by any number less than k. The Integer Factorization Decision Problem is said to be both NP and co-NP.

It is not yet known if the Integer Factorization Decision Problem is NP-complete. If a problem in NP is NP-complete, then all other problems in NP can be reduced to the NP-complete problem in polynomial time. This makes NP-complete problems the hardest class of problems to solve out of all problems in the class NP since it would imply that the class NP is equivalent to the problem that is NP-complete. On the plus side, if P is a problem in polynomial time, then P being NP-complete implies that all problems in NP can be reduced to problems in polynomial time. Due to the fact that the Integer Factorization Decision Problem is both NP and co-NP,

proving that the problem is NP-complete would mean proving that NP = co-NP, a result that would be very surprizing.

Therefore, it is highly unlikely that the problem of integer factorization is NP-complete, leading mathematicians to believe that factoring algorithms belong in a class outside NP-complete, and thus not part of the hardest class of problems. There have been many attepts to find an algorithm to solve the Integer Factorization Decision Problem in polynomial time but each has failed, leading mathematicians to believe that factoring algorithms that run in polynomial time do not exist. Perhaps there will be faster algorithms invented in the future, but it is considered extremely improbable that an algorithm would jeopardize the security of a system with an encryption key of significant length.

This is why all systems encoded by RSA-cryptography are considered secure for the time being. (Excluding the possibility of a quantum computer processor being invented in the near future!)

Bibliography

[1] R.L. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, 1997.

[2] J.P. Buhler, H.W. Lenstra, Jr., Carl Pomerance, *Factoring Integers with the Number Field Sieve*, 1991.

[3] Neal Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1987.

[4] Neal Koblitz, *Algebraic Aspects of Cryptography*, Springer, 1999.

[5] A.K. Lenstra, H.W. Lenstra, Jr., *The Development of the Number Field Sieve*, Lecture Notes in Mathematics, 1554, Springer-Verlag, 1993.

[6] A.K. Lenstra, H.W. Lenstra, Jr., M. S. Manasse, and J.M. Pollard, *The Factorization of the Ninth Fermat Number*, 1991.

[7] Richard A Mollin, *RSA and Public-Key Cryptography*, Chapman & Hall/CRC, New York, 2003.

[8] Joseph H. Silverman, *A Friendly Introduction to Number Theory*, Pearson Prentice Hall, 2006.

[9] Douglas R. Stinson, *Cryptography Theory and Practice*, CRC Press, 1995.