Modern Factoring Algorithms

Kostas Bimpikis and Ragesh Jaiswal
University of California, San Diego

... both Gauss and lesser mathematicians may be justified in rejoicing that there is one science [number theory] at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean.
- G.H. Hardy, *A Mathematician's Apology*, 1940

Unfortunately for Hardy, nowadays, number theoretic results form the basis of several applications in the field of cryptography. In this survey, we will present the most recent approaches on solving the factoring problem. In particular, we will study Pollard's ρ algorithm, the Quadratic and Number Field Sieve and finally, we will give a brief overview on factoring in quantum computers.

---

1. INTRODUCTION

The Integer Factorization problem is defined as follows: given a composite integer N, find a non-trivial factor e of N, i.e. find e such that $e|N$. The most straightforward method of factoring is *trial division*, which, essentially, checks for every prime number p, such that $p \leq \sqrt{N}$, if $p|N$. However, trial division may take more than $O(\sqrt{N})$ bit operations, which makes it extremely inefficient for the numbers of interest.

Before going on with describing the modern ideas for factoring, it would be interesting to give some facts about the history of the problem. Integer factorization is one of the oldest problems in mathematics. Many of the techniques used in modern factoring algorithms date back to ancient Greece (eg. the sieve of Eratosthenes, Euclid's algorithm for finding the *gcd*). The great mathematicians of the 17th and 18th century (Fermat, Euler Legendre, Gauss, Jacobi), also, contributed with a number of ideas. But the major breakthroughs happened during the last 30 years, especially after the introduction of public key cryptography, and in particular of the RSA cryptosystem. Specifically, it can be shown that if factoring is solved efficiently, then the RSA cryptosystem would become extremely insecure. That is why, RSA Security, the company behind RSA cryptosystem, provides funding for a factoring competition, the so-called RSA challenge. Interestingly enough, the most recent algorithmic improvements in factoring are closely associated with a RSA challenge number.

Factoring is also interesting from a complexity theoretic point of view. Its complexity status hasn't been resolved yet. Its decision version ("has N a factor less than M?") is known to belong to both NP and $coNP$. However, it is suspected that it isn't *NP-complete* or *coNP-complete*, since otherwise this would imply that $NP = coNP$, which is a rather surprising result. Moreover, it is known to be in BQP (bounded error quantum polynomial time) because of Shor's algorithm [Shor,

1994].

This paper is organized as follows: in section 2 we present *Pollard's ρ algorithm*, which was the first major improvement over the naive trial division algorithm. Then, in section 3 we describe the best two algorithms up to date: *quadratic sieve* and *number field sieve*. These two algorithms follow the same pattern and that is why we categorize them under the same name: *sieve algorithms*. In section 4 we present Shor's algorithm for factoring in quantum computers. Finally, in section 5 we conclude and provide the current status of the factoring problem.

2. POLLARD'S ρ ALGORITHM

In 1975 John M. Pollard proposed a very efficient Monte Carlo algorithm for factoring, now known as Pollard's ρ method. The algorithm is substantially faster than trial division for finding a small non-trivial factor of a large integer N, if such exists.

2.1 Basic ideas

The basic component of Pollard's ρ algorithm is a sequence of pseudo-random integers constructed in the following way:

$$x_0 = random(0, N-1)$$
$$x_i = f(x_{i-1}) \ (mod\ N), i = 1, 2, \cdots$$

where f is a polynomial with integer coefficients, eg. in most practical implementations of the algorithm $f(x) = x^2 + \alpha$, $\alpha \neq 0, -2$.

The intuition behind the algorithm is the following simple observation: suppose that N is the number we would like to factor and d is a non-trivial divisor of it. We further assume that d is small compared to N. Since, there is a smaller number of congruence classes modulo d than modulo N, it is quite probable, that there exist numbers x_i and x_j, such that: $x_i \equiv x_j \mod d$ and $x_i \not\equiv x_j \mod N$

From the above we conclude that $d|(x_i - x_j)$ and $N \not| (x_i - x_j)$, thus, it follows that $\gcd(x_i - x_j, N)$ is a non-trivial divisor of N.

Let's see how we can combine the previous observation with the sequence of integers described above to devise an efficient algorithm to compute a non-trivial factor d of N. First of all, note that, although we don't know d, we can use the x_i's to get a non-trivial factor of N by simply calculating $\alpha = gcd(x_i - x_j, N)$ at each step for every x_j with $j < i$. Most of the times α will be equal to 1, but we expect that it won't take too many steps to find d. The algorithm will become apparent with an example:

Let $N = 1079 = 13 * 83$, $f(x) = x^2 - 1$ and x_0=2. Then the sequence of pseudo-random integers generated by f is the following:

$$2, 3, 8, 63, 1194, 1186, 177, 814, 996, 310, 396, \cdots$$

At each step i of the algorithm we compute $\gcd(x_i - x_j, N)$ for every $j < i$. In our

case:

$$\begin{aligned}
gcd(x_1 - x_0, N) &= gcd(3 - 2, 1079) = 1 \\
gcd(x_2 - x_1, N) &= gcd(8 - 3, 1079) = 1 \\
gcd(x_2 - x_0, N) &= gcd(8 - 2, 1079) = 1 \\
gcd(x_3 - x_2, N) &= gcd(63 - 8, 1079) = 1 \\
gcd(x_3 - x_1, N) &= gcd(63 - 3, 1079) = 1 \\
gcd(x_3 - x_0, N) &= gcd(63 - 2, 1079) = 1 \\
gcd(x_3 - x_2, N) &= gcd(1194 - 63, 1079) = 13
\end{aligned}$$

The algorithm made only 7 calculations to find the divisor 13.

2.2 Improvements

Unfortunately, we cannot always guarantee that we will find d in such a small number of steps. Therefore, the cost of computing $gcd(x_i - x_j, N)$ for every $j < i$ becomes prohibitive as i increases. There are a few ways we could improve the algorithm:

—Pollard proposed using Floyd's cycle-finding algorithm to reduce the number of steps before two integers are found, such that $x_i \equiv x_j \ (mod\ d)$. To be more specific, the improved algorithm computes the following gcd's only:

$$gcd(x_{2i} - x_i, N) \qquad i = 1, 2, \cdots$$

To see why this works, suppose that $x_i \equiv x_j \ (mod\ d)$. Then, it is easy to see that $x_{i+1} \equiv x_{j+1} \mod d$, since $x_{i+1} = f(x_i)$ and $x_{j+1} = f(x_j)$. We can extend this fact to every $x_{i+k} \equiv x_{j+k} \ (mod\ d)$ (k=$1, 2, \cdots$), so eventually there will be an i, such that $x_{2i} = x_i \mod d$. So, we conclude that it suffices to compute for every i only one gcd ($gcd(x_{2i} - x_i, N)$) instead of $i - 1$. Also, note that we can exploit the following fact, in order to compute x_{2i}, without using the values $x_{i+1}, x_{i+2}, \cdots, x_{2i-1}$:

$$\begin{aligned}
y_0 &= x_0 \\
y_1 &= x_2 = f(x_1) = f(f(x_0)) = f(f(y_0)) \\
y_2 &= x_4 = f(x_3) = f(f(x_2)) = f(f(y_1)) \\
&\vdots \\
y_i &= x_{2i} = f(x_{2i-1}) = f(f(x_{2i-2})) = f(f(y_{i-1}))
\end{aligned}$$

So, this allows us to compute x_{2i} by storing only a single value, namely y_{i-1}.

—Another useful trick is instead of performing a gcd operation at every single step, to construct big products and perform a single gcd operation every, say, n steps. In other words, instead of computing gcd($x_{2i} - x_i, N$) for every i, we perform one gcd computation every n steps ($gcd(Q_n, N)$ by setting $Q_n = \Pi_{i=1}^{n}(x_{2i} - x_i)(mod\ N)$. It is easy to see that if gcd($x_{2i} - x_i, N$) $= d > 1$, then also $gcd(Q_n, N) > 1$.

—Brent proposed a slight modification to Pollard's ρ algorithm. Specifically, instead of computing $gcd(x_{2i} - x_i, N)$ he considers a different set of differences. This modification reduces the running time by 1/4.

2.3 Running time analysis

Suppose that p is a prime divisor of N. Then, we will show that it takes Pollard's algorithm an expected number of $O(\sqrt(p))$ iterations to find p. In order to prove this statement we will use the birthday paradox (see appendix in [Bellare et al., 2004]).

First of all, we assume that the numbers generated by f have a "random" behavior, i.e. are independently chosen from $\{0, 1, \cdots, p-1\}$, which is essential for the birthday paradox to apply in our setting. Suppose that we have the first k numbers of the sequence in hand $x_1, x_2, \cdots, x_k$. Then, according to the birthday paradox, the probability that all these k numbers are distinct $(mod\ p)$ is equal to the following:

$$(1 - \frac{1}{p})(1 - \frac{2}{p}) \cdots (1 - \frac{k-1}{p}) \approx \exp(\frac{-k^2}{2p})$$

From the above we conclude that if $k = O(\sqrt{p})$, then the probability of finding the values x_{2i}, x_i we are looking for is high. So, we conclude that after $O(\sqrt{p})$ we expect to find p.

Each iteration consists of computing $x_i, x_{2i} = y_i$ and $gcd(x_{2i} - x_i, N)$, so we need 3 modular multiplications, 4 modular additions (subtractions are counted as additions) and one gcd operation per iteration. This gives us a running time of $O(|N|^2)$ per iteration. We conclude that the total running time of Pollard's ρ method is $O(\sqrt{p} \cdot |N|^2) = O(N^{\frac{1}{4}} \cdot (\log N)^2)$, since p $\leq \sqrt{N}$.

Note, that the above proof depends on the assumption that f has a "random" behavior, which hasn't been proved formally yet. However, the algorithm seems to work well in practice with f equal to simple quadratic polynomials.

3. SIEVE ALGORITHMS

Before moving on with the main idea and the algorithm we need to set up some vocabulary.

DEFINITION 3.1. *A non-empty set, F of positive prime integers is called a* factor base. *An integer k is said to be smooth over a factor base F, if all primes occuring in the unique factorization of k into primes, are members of F.*

3.1 Difference of Squares

Most of the modern factoring algorithms use the basic technique of "difference of squares". The idea here is that given a number, N if we find two integers x and y such that $x \not\equiv \pm y\ (mod\ N)$ and:

$$x^2 \equiv y^2\ (mod\ N) \tag{1}$$

then $gcd(x - y, N)$ and $gcd(x + y, N)$ gives non-trivial factors of N. So, the first idea that comes to mind is why not just guess integers x, y which satisfy 1 and hope that $gcd(x - y, N)$ gives a non-trivial factor of N. If $N = pq$ for some primes p, q then it is easy to show that this happens with probability 2/3. This motivates us to develop factoring algorithms that try to find integers x, y such that the difference of their squares is divisible by N. Using these ideas, let us describe a simple factoring algorithm.

3.2 A Simple Factoring Algorithm

Consider a factor base $F = \{p_1, p_2, ..., p_m\}$. We construct a set $U = \{r_1, r_2, ..., r_n\}$ of integers such that $\forall i : f(r_i) = r_i^2 \ (mod\ N)$ is smooth over the factor base F, or:

$$\forall i : f(r_i) = r_i^2 \equiv p_1^{e_{i1}} p_2^{e_{i2}} ... p_m^{e_{im}} \ (mod\ N) \tag{2}$$

e_{ij} in the above equation is the exponent of p_j occuring in the factorization of $f(r_i)$. If U has the additional property that:

$$\forall j, 1 \leq j \leq m : \sum_{i=1}^{n} e_{ij} = 2 \cdot e_i \ \ (for\ some\ e_i \in \mathbb{Z}) \tag{3}$$

Then we have:

$$\prod_{i=1}^{n} f(r_i) = \prod_{i=1}^{n} r_i^2 \equiv (p_1^{e_1} \cdot p_2^{e_2} ... p_m^{e_m})^2 \ (mod\ N) \tag{4}$$

Now, the integers x, y, we were looking for are simply:

$$x = \prod_{i=1}^{n} r_i$$

$$y = \prod_{i=1}^{m} p_i^{e_i}$$

From equation 4 we can check that $x^2 \equiv y^2 \ (mod\ N)$. The main issue we are left with, is how to construct a set U as defined above. A way to do this, is by simply guessing $> m$ integers, the squares of which are smooth over F and then picking out a subset U of these, such that equation 3 is satisfied.

Picking the right subset U is done as follows: first, each integer r that is picked for consideration, is represented as an m dimensional vector, $\bar{v}$, where $\bar{v}_j = e_j$, where e_j is the exponent of the j-th element of the factor base in the integer decomposition of $f(r)$ into primes. The problem of finding U is now simplified to finding a set of vectors, such that their sum mod 2 is the zero vector. This can be easily achieved using linear algebraic techniques, which we do not discuss here.

Now, we are in a position to understand the quadratic sieve algorithm which is based more or less on the ideas presented above.

3.3 Quadratic Sieve Algorithm

The Quadratic Sieve algorithm was developed by Carl Pomerance and was a popular factoring algorithm in the 1980's and early 1990's. In particular, in 1994 the factorization of RSA-129 challenge number was completed using QS. This fact came as a surprise to the community, since RSA-129 was thought to be extremely difficult to factor.

The algorithm is mostly the same as the previous one. The difference lies in the modification of the function f from $f(r_i) = r_i^2 \ (mod\ N)$ to $f(r_i) = r_i^2 - N$. As we will see this change does not effect the logical flow of the previous algorithm, however it significantly improves the algorithm's running time, because of the fact that, it is a lot faster to check if for some r_i, $f(r_i)$ is smooth over a factor base.

The algorithm works as follows:

(1) Fix a factor base $F = \{p_1, p_2, ..., p_m\}$.
(2) Search for integers r such that $f(r) = r^2 - N$ is smooth over F.
(3) Find a subset $U = \{r_1, r_2, ..., r_n\}$ such that equations 2, 3 and 4 hold for the modified f. Now we can define:

$$x = \prod_{i=1}^{n} r_i$$

$$y = \prod_{i=1}^{m} p_i^{e_i}$$

We can check that x, y satisfy 1:

$$x^2 = \prod_{i=1}^{n} r_i^2 \equiv \prod_{i=1}^{n} (r_i^2 - N) \equiv \left(\prod_{i=1}^{m} p_i^{e_i} \right)^2 \equiv y^2 \ (mod\ N) \tag{5}$$

To see how the modified f benefited us, let us concentrate on the second step. Previously, in order to check the smoothness of $f(r)$ for a randomly selected r (in a particular range of interest), we would verify that $f(r)$ is divisible by all primes in the factor base F using trial division. Instead, in QS we use a much more efficient approach: we consider each prime p in F and find all the $r's$ (in the range of interest) for which p divides $f(r)$. This procedure is called the *sieving* process. Formally, for all integers, r in some range, say [a, b] (the range in which we expect to get more than m integers with smooth $f(r)$'s) we initialize an array with the corresponding $f(r)'s$. Next, for every prime p in the factor base, we divide all integers in the array with p (and its powers). We continue this process until we have more than m array locations containing 1. The $r's$ corresponding to these locations are the ones we are interested in.

Its still not very clear as to how the modified f would make the sieving procedure faster compared to the previous function (because the above procedure could have been applied in the simple algorithm too). To see this, let for some integer r, $f(r) = 0\ (mod\ p)$, where p is a prime in the factor base. This implies that $r^2 - N \equiv 0\ (mod\ p)$. Then, for any integer k we have

$$(r + kp)^2 - N \equiv r^2 + 2rkp + k^2p^2 - N \ \equiv r^2 - N \equiv 0\ (mod\ p)$$

or, $f(r + kp) \equiv 0\ (mod\ p)$.
So, in the sieving process, we first solve the quadratic congruence $r^2 \equiv N\ (mod\ p)$ (which can be done efficiently) and then divide p out from array locations corresponding to r and $r + kp$ for $k \in \mathbb{Z}$. This is better than blind trial division because we perform the division only with the array elements, which we apriori know would be divisible by the prime.

3.3.1 *Time Complexity.* In this subsection we will provide an estimation for the running time of quadratic sieve algorithm. Actually, we will bound the running time of the first algorithm we gave (the simple algorithm). Then, we will see how

much the sieving process of QS improves this bound. For simplicity, we may assume that our factor base consists of all primes $\leq y$ (the value of y will be determined later). The algorithm consists of three main steps as we have seen:

(1) find $(\pi(y)+1)$ integers $r_1, r_2 ... r_{\pi(y)+1}$ (where $\pi(y)$, the number of primes less that y), such that r_i^2 $(mod\ N)$ can be written as a product of primes (and their powers) $\leq y$.
(2) find a subset U of these numbers, such that the product of these number is a perfect square $(mod\ N)$.
(3) if integers x, y found in this way, are such that $x \equiv \pm y$ $(mod\ N)$ repeat the process.

The simple algorithm we presented in the previous subsection uses random guessing as a method to find the integers r_i. It can be shown that the probability to find an integer r_i, such that $r_i < N$ and r_i^2 can be written as the product of all primes $\leq y$ is approximately equal to u^{-u}, where u is defined as: $u = \frac{log\ N}{log\ y}$. Thus, we need to randomly guess $(\pi(y)+1)u^u$ integers, before we find the $(\pi(y)+1)$ r_i's. Moreover, the time to test for each such integer if it is smooth over the factor base, is upper bounded by the time to divide it with every prime p in the base, i.e. bounded by $O(y \cdot log\ N \cdot log\ y)$, since

—$\pi(y) \leq y$
—the primes in the factor base have at most $(log\ y)$ bits
—the randomly guessed have integers at most $(log\ N)$ bits.

So, using the fact that $\pi(y) \approx y/log\ y$ we get that step 1 requires $O(u^u \cdot logN \cdot y^2)$ bit operations.

As far as step 2 is concerned, it involves polynomial operations in y and $log\ N$. Therefore, step 2 adds a term $O(y^h \cdot (log\ N)^s)$ for integers h, s, to the complexity. Finally, there is a probability of at most $1/2$ for this process to fail, i.e. the resulting numbers x, y are such that: $x = \pm y \mod N$. So, if we repeat the process, say 100 times, then with high probability we will find a non-trivial factor of N. Therefore, we conclude that the time complexity has the following form:

$$O(100(u^u(log\ N)y^2 + y^h + (log\ N)^s)) = O(u^u y^h + (log\ N)^s)$$

Finally, we have to determine y. As we have claimed above the probability of finding a r_i such that $(r_i^2\ mod\ N)$ can be written as a product of primes $\leq y$ is u^u, where $u = \frac{log\ N}{log\ y}$. Intuitively, if y is large, then u^{-u} is large, so with fairly high probability we find the smooth r_i's. However, if y is large then it takes too much time to verify that r_i is actually smooth and, in step 2, to find the subset U. On the other hand, if y is small these operations are not so costly, but u^{-u} becomes small, so we would have to try a lot of numbers before we find a r_i, that satisfies our conditions. Therefore, we should choose the value of y from an intermediate range. It can be shown (after some simple algebraic calculations) that the appropriate value of y is:

$$y \approx e^{\sqrt{\frac{log\ N}{2k} log\ log\ N}}$$

Putting all these together, we get the following "rough" estimate for the time complexity:

$$O(e^{C\sqrt{log\ N log\ log\ N}})$$

where C is a constant.
The time complexity of QS has the same format. To be more specific, the expected running time of QS is asymptotically bounded above by:

$$O(e^{(1+\epsilon)\sqrt{logNloglogN}})$$

for any $\epsilon > 0$

3.4 General Number Field Sieve

We would like to use the idea of "difference of squares" to construct the algorithm but would like to make it faster than the previous 2 algorithms. The new idea here here is that we can work with rings other than $\mathbb{Z}$ which has the notion of smoothness similar to that defined for $\mathbb{Z}$ and hopefully has smooth values. If a mapping exists from such a ring to $\mathbb{Z}_N$ then we can possibly work out a way to construct a difference of squares. Here is how we might do it. We begin by considering the following fact from algebraic number theory.

FACT 3.1. *Given a monic, irreducible polynomial $f(x)$ with integer coefficients, a root $\theta \in C$ of $f(x)$, and an integer $m \in \mathbb{Z}_N$ for which $f(m) \equiv 0\ (mod\ N)$, the mapping $\phi : \mathbb{Z}[\theta] \to \mathbb{Z}_N$ with $\phi(1) = 1\ (mod\ N)$ which sends θ to m, is a surjective ring homomorphism. ($\mathbb{Z}[\theta]$ is a set of all polynomials in θ with integer coefficients)*

Now using the above fact we construct a difference of squares in the following manner. We find a set, U of integer pairs (a, b) such that:

$$\prod_{(a,b)\in U} (a + b\theta) = \beta^2 \text{ and } \prod_{(a+b)\in U} (a + bm) = z^2 \tag{6}$$

where $\beta \in \mathbb{Z}[\theta]$ and $z \in \mathbb{Z}$. Then using the homomorphism as defined in the fact above, the integer x, y can be defined as:

$$x = \phi(\beta) \text{ and } y = z$$

We will do a quick sanity check to confirm that this pair produces a difference of squares:

$$x^2 \equiv (\phi(\beta))^2 \equiv \phi(\beta^2) \equiv \phi(\prod_{a+b\theta\in U} (a+b\theta)) \equiv \prod_{a+b\theta\in U} \phi(a+b\theta) \equiv \prod_{a+b\theta\in U} (a+bm) \equiv y^2\ (mod\ N)$$

The algorithm has similar steps as we had outlined for the previous algorithms. Let us outline the steps and then discuss each step separately:

(1) Fix factor base I in the ring $\mathbb{Z}[\theta]$ and F in the ring $\mathbb{Z}$.
(2) Find integer pairs a, b such that $(a+b\theta)$ is smooth over I and $\phi(a+b\theta) \equiv (a+bm)$ is smooth over F.
(3) Find a subset, U of pairs found in the previous step such that equation 6 is satisfied.

Let us start with the first step of the algorithm. What do we mean by a factor base in $\mathbb{Z}[\theta]$? In $\mathbb{Z}$ we have defined it to be a set of prime numbers. It happens that in $\mathbb{Z}[\theta]$ we can define a factor base as a set of prime ideals of $\mathbb{Z}[\theta]$ of special form. We will derive this special form as we examine some interesting results from algebraic number theory.

3.4.1 *Algebraic Factor Base.* To understand the algorithm, we require a lot of results from algebraic number theory. Since our main interest is getting a good picture of the algorithm and the overall logical flow, we will not go forward to prove the results from number theory in this report. Instead we will dwell upon the results to build up the logical structure of the algorithm.

Let us begin by setting up some vocabulary.

DEFINITION 3.2. *Given a monic irreducible polynomial $f(x)$ of degree d with integer coefficients and a root $\theta \in C$ of $f(x)$, the set of all polynomials in θ with integer coefficients form a ring which is denoted by $\mathbb{Z}[\theta]$* [1]

DEFINITION 3.3. *Given an element $\alpha \in \mathbb{Z}[\theta]$, we denote the principle ideal generated by α as $< \alpha >$*

Now the very first fact from number theory, that we mention below should make the overall approach clear.

FACT 3.2. *Ideals of $\mathbb{Z}[\theta]$ can be uniquely factored, upto to order, into prime ideals of $\mathbb{Z}[\theta]$.*

Using this result we can start thinking of an algorithm in the following manner. We will fix a factor base consisting of prime ideals of $\mathbb{Z}[\theta]$. Then we pick up elements $a + b\theta \in \mathbb{Z}[\theta]$ for integers a, b such that $< a + b\theta >$ is smooth over the factor base.

$$< a + b\theta >= \rho_1^{e_1} \rho_2^{e_2} ... \rho_m^{e_m} \tag{7}$$

where ρ_i is a prime ideal in the factor base and e_i is the exponent which is basically a positive integer. Once we have enough such elements the we can possibly multiply a subset of such elements to produce a square in $\mathbb{Z}[\theta]$. Remember that we simutaneously need to produce a square in $\mathbb{Z}$ as described in the previous subsection (we will deal with this later).

Though the idea is more or less clear we still need to answer some of the following questions:

—What is the representation that we would be using for principle ideals and ideals? Is there an easy way to deal with these quantities?

—The fact mentioned above just states that a unique factorization exists, but gives no clue of actually getting the factorization i.e. the ideals and the exponents. So is there a way to get the factorization? Are there any special properties of the exponents that can be exploited?

Now it so happens that the answer to all the questions lies in the special form of the element $a + b\theta \in \mathbb{Z}[\theta]$, that we are considering. There are some nice results about elements of this type which we will exploit to answer the above questions.

[1] Definitions of various algebraic terms can be found at http://www.cs.ucsd.edu/users/rjaiswal/WebNotes/Files/algebra.pdf

Let us roughly sketch our goals at this point, so that we know where we are heading to. It would be good if we can express the factorization of an element $a + b\theta \in \mathbb{Z}[\theta]$ in terms of some *special* prime ideals which have easy representation and for which the exponents can be found easily. At this point of time we can only hope for such things and all our hopes are based on just the fact that we are considering elements of special form.

Let us continue by defining useful quantities and stating some interesting results from number theory.

DEFINITION 3.4. *Given an element $\alpha \in \mathbb{Z}[\theta]$, the norm of α denoted by $N(\alpha)$ is defined as:*

$$N(\alpha) = g(\theta_1) \cdot g(\theta_2)...g(\theta_d) \tag{8}$$

where $\alpha = g(\theta)$ ($g(\theta)$ is some polynomial in θ with integer coefficients) and $\theta_1, \theta_2...\theta_d$ are the roots of the polynomial $f(x)$.

Following is an interesting fact related to the norm of an element.

FACT 3.3. *The norm function as defined in 3.4 is a multiplicative function that maps elements of $\mathbb{Z}[\theta]$ to elements of $\mathbb{Z}$.*

Since we are also interested in ideals, let us define the norm for ideals.

DEFINITION 3.5. *Given a ring R and an ideal $\mathcal{I}$ of R, the* norm *of $\mathcal{I}$ is defined to be $[R : \mathcal{I}]$, the number of cosets of $\mathcal{I}$ in R.*

Having defined the norm function for an element of $\mathbb{Z}[\theta]$ and the norm of an ideal of a ring, consider the following result which relates these two quantities.

FACT 3.4. *Let $\alpha \in \mathbb{Z}[\theta]$. $N(< \alpha >) = |N(\alpha)|$.*

Now we define the *special* prime ideal that we hinted earlier.

DEFINITION 3.6. *A* first degree prime ideal *ρ of $\mathbb{Z}[\theta]$ is a prime ideal of $\mathbb{Z}[\theta]$ such that $N(\rho) = p$ for some prime integer p.*

The special prime ideal as defined above, exhibit the following interesting property.

THEOREM 3.1. *The set of pairs (r, p) where p is a prime integer and $r \in \mathbb{Z}_p$ with $f(r) \equiv 0 \pmod{p}$ is in bijective correspondence with the set of all first degree prime ideals of $\mathbb{Z}[\theta]$.*

This result hints at an efficient representation of first degree prime ideals which unfortunately can be made useful only if we show that $< a + b\theta >$ is uniquely factored into only the first degree prime ideals. We also do not have a clue right now as to how to get the exponents in the factorization even if the previous condition holds. The following two results should resolve most of the mystery.

FACT 3.5. *For every prime ideal ρ_i of $\mathbb{Z}[\theta]$, there exists a group homomorphism $l_{\rho_i} : \mathbb{Q}(\theta)^* \to \mathbb{Z}$ that possesses the following properties.*

(1) $l_{\rho_i}(\beta) \geq 0$ for all $\beta \in \mathbb{Q}(\theta)^*$

(2) $l_{\rho_i}(\beta) > 0$ if and only if the ideal ρ_i divides the principal ideal $< \beta >$

(3) $l_{\rho_i}(\beta) = 0$ *for all but a finite number of prime ideals* ρ_i *of* $\mathbb{Z}[\theta]$*, and* $|N(\beta)| = \prod N(\rho_i)^{l_{\rho_i}(\beta)}$ *for all prime ideals* ρ_i *of* $\mathbb{Z}[\theta]$

Here $\mathbb{Q}(\theta)^*$ *denotes the multiplicative group of non-zero elements in the field* $\mathbb{Q}(\theta)$ *(*$\mathbb{Q}(\theta)$ *is a ring consisting of all polynomials in* θ *with coefficients* $\in \mathbb{Q}$*).*

Note that if $\beta = a + b\theta \not\equiv 0$, then $\beta \in \mathbb{Q}(\theta)^*$ and the above applies.

FACT 3.6. *Given an element* $\beta \in \mathbb{Z}[\theta]$ *of the form* $\beta = a+b\theta$ *for coprime integers* a *and* b *and a prime ideal* p *of* $\mathbb{Z}[\theta]$*, then the homomorphism* l_ρ *of 3.5 corresponding to* ρ *has* $l_\rho(\beta) = 0$ *if* ρ *is not a first degree prime ideal of* $\mathbb{Z}[\theta]$*. Furthermore, if* ρ *is a first degree prime ideal of* $\mathbb{Z}[\theta]$ *corresponding to the pair* (r, p) *as in theorem 3.1, then*

$$l_\rho(\beta) = \begin{cases} ord_p(N(\beta) & if a \equiv -br \ (mod\ p), \\ 0 & otherwise. \end{cases}$$

where $ord_p(N(\beta))$ *denotes the exponent of the prime integer* p *occurring in the unique factorization of the integer* $N(\beta)$ *into distinct primes.*

Having studied all the results above, we are now in a position to put things together and understand the working of the algorithm. Consider an element $a+b\theta \in \mathbb{Z}[\theta]$ for co-prime a, b. From 3.2 we can write $< a + b\theta >= \rho_1^{e_1}\rho_2^{e_2}...$ for prime ideals $\rho_1, \rho_2....$ From facts 3.4 and 3.5 we have:

$$p_1^{e_1'}p_2^{e_2'}... = |N(a + b\theta)| = N(< a + b\theta >) = N(\rho_1)^{e_1}N(\rho_2)^{e_2}... \tag{9}$$

Now from fact 3.6 we get that $e_i = 0$ for ρ_i which is not a first degree prime ideal of $\mathbb{Z}[\theta]$. Also from theorem 3.1 for any first degree prime ideal ρ_i we have a corresponding pair (r, p) and again from fact 3.6 we get that $e_i = 0$ if $a + br \not\equiv 0 \ (mod\ p)$. Also, for the remaining first degree prime ideals, the exponent is the exponent of the correponding prime number occuring in the factorization of $|N(a + b\theta)|$. Let us now sum up the results:

(1) For co-prime a, b, $< a + b\theta >$ can be uniquely factored into only some first degree prime ideals of $\mathbb{Z}[\theta]$.
(2) The condition for some first degree prime ideal dividing $< a + b\theta >$ can be checked easily by considering the corresponding (r, p) pair and checking if $a + br \equiv 0 \ (mod\ p)$.
(3) The exponents in the above factorization can be obtained by looking at the prime factorization of $|N(a + b\theta)|$

3.4.2 *Sieving.* With this we come to the second step of the algorithm that we outlined initially. We fix our algebraic factor base I as some set of first degree prime ideals (this is simply represented by the set of pair (r, p) corresponding to each such ideal) and a rational factor base F as a set of primes. We have to find pairs (a, b) such that $a + b\theta$ is smooth over I and simultaneously $a + bm$ is smooth over F. When we have enough pairs we can construct the set U such that equation 6 is satisfied. Here is how we generate the pairs.

We fix b and consider all $a's$ in some range such that a and b are co-prime. We initialize an array with the values $a + bm$. Then we pick a prime, p from our factor base F and check if this divides an array location. This is done by checking if

$a \equiv -bm \ (mod\ p)$, but this also implies that $a = -bm + kp$ for integer k. So we can simply go to the array locations corresponsing to these $a's$ and divide out p from the value stored. We do this for all primes in our factor base F. Finally we pick out the useful pairs by looking for array locations that contain 1. Note that such a pair, to be useful, should also be smooth over I. So we need to sieve based on I simultaneously which is done as follows. As we had initialized the array with $a + bm$ in the previous case, here we initialize another array with $N(a + b\theta)$ for the same a, b values. We then pick an (r, p) pair corresponding to a first degree prime ideal $\rho \in I$, and check if $a \equiv -rb \ (mod\ p)$ (if so this implies that ρ divides $< a + b\theta >$). If the condition is satisfied then we also have $a = -br + kp$ for integer k. So we can go to the array locations corresponding to these $a's$ and divide out p from the value in the location. This is repeated for all the elements in I and finally the array location is searched for 1. So the array location which are simultaneously 1 for both the arrays give the required (a, b) pairs. Once we have $> max(|F|, |I|)$ such pairs we can construct a set U of pairs which satisfy equation 6 (using linear algebra techniques as discussed in the simple algorithm).

The only thing left now is to define the function f and m. We assume that we are given a number N to be factored in the form $w^e - s$. We pick up a small integer $d > 0$. Given d, let k be the minimum integer such that $kd \geq e$. So we have $w^{kd} \equiv sw^{kd-e} \ (mod\ N)$. Now we put $m = w^k$ and $c = sw^{kd-e}$ so that $m^d \equiv c \ (mod\ N)$. We define the function f as $f(x) = x^d - c$. We assume that $f(x)$ is irreducible for a reasonable choice of d (since it can be shown that for a non-trivial factor of f we can find a non-trivial factor of N).

3.4.3 *Time Complexity.* There is only a heuristic analysis for the time complexity of the algorithm. We avoid going into the complex details of the analysis and just mention the expected time. The algorithm runs in expected time $exp((c + o(1))(log\ N)^{1/3}((log\ log\ N)^{2/3})$ where c is a constant with value ≈ 1.526.

4. SHOR'S QUANTUM FACTORING ALGORITHM

A quantum computer is a device that uses quantum phenomena to perform computation. A classical system follows a single computation path while a quantum system uses superposition of states to move along all possible paths and finally end up in one particular state when measured. We begin by understanding some definitions and properties of a quantum system.

DEFINITION 4.1. *A* qubit *or a quantum bit is the analog of a bit for quantum computation. A qubit may assume continuum values of the form* $a|0> + b|1>$, *where* a, b *are arbitrary complex numbers (known as probability amplitudes) and* $|0>, |1>$ *are the basis states (analog of 0, 1 states in a classical system). Complex numbers* a *and* b *satisfies* $a^2 + b^2 = 1$.

Now we look an interesting property that a *system* of qubits exhibit. Consider a 2 qubit system $|\psi> = \alpha_{00}|00> + \alpha_{01}|01> + \alpha_{10}|10> + \alpha_{11}|11>$. Now when the first qubit of this system is observed, the post-measurement quantum state of the system would be the superposition of all the states(present in pre-measurement) which have the first bit same as measured, normalized to unit length. For example if the first qubit is measured to be 0, then the post-measurement quantum state

would be:

$$|\psi' >= \frac{\alpha_{00}|00> +\alpha_{01}|01>}{\sqrt{\alpha_{00}^2 + \alpha_{01}^2}}$$

DEFINITION 4.2. *A Quantum Fourier Transform is defined as:*

$$|a> \xrightarrow{FT_{\mathbb{Z}_q}} \frac{1}{q^{1/2}} \sum_{c=0}^{q-1} \omega^{ca}|c> \tag{10}$$

where $|a>$ is a basis state (the quantum system with a represented as binary), $0 \leq a < q$ for some q where the number of bits of q is polynomial, and ω is the q^{th} root of unity. Some observations about ω that will be helpful later are:

$$1 + \omega^j + \omega^{2j} + ... + \omega^{(q-1)j} = 0 \; if \; j \neq 0 \; (mod \; q) \tag{11}$$

$$1 + \omega^j + \omega^{2j} + ... + \omega^{(q-1)j} = q \; if \; j = 0 \; (mod \; q) \tag{12}$$

The other important thing to note about this transform is that there are efficient quantum circuit made of quantum gates *(analogous to classical gates and circuits) that can perform this transform.*

Now we can start discussing the quantum algorithm. We begin with some number theory basics. Given N (to be factored), consider the multiplicative group $< \mathbb{Z}_N^*, . >$. Let r be the order of any randomly chosen element $x \in \mathbb{Z}_N^*$. Then we have:

$$\begin{gathered} x^r - 1 \equiv 0 \pmod N \\ \Rightarrow (x^{r/2} - 1)(x^{r/2} + 1) \equiv 0 \pmod N \end{gathered}$$

Note that $x^{r/2} - 1 \not\equiv 0 \pmod N$ when r is even(else it becomes the order of x). So if **(A)** r is even and **(B)** $x^{r/2} + 1 \not\equiv 0 \pmod N$, we can compute a non-trivial factor of N by finding $gcd(x^{r/2} - 1, N)$ (can be done efficiently using the euclid's algorithm). Now we have reduced the original problem of facoring to the following problem.

(1) Find $x \in \mathbb{Z}_N^*$ such that properties (A) and (B) are satisfied.
(2) Find order r of x and compute $gcd(x^{r/2} - 1, N)$

The following result solves the first part of the problem.

FACT 4.1. *For randomly chosen $x \in \mathbb{Z}_N^*$ there is a high probability that properties (A) and (B) are satisfied.*

So we assume that we are given an x satisfying conditions (A) and (B), and we mainly concentrate on finding r.

In this work we will mainly try to understand how the quantum phenomenon helps in giving us r. We consider a simple case when we make the assumption that r divides q, where q is the quantity as in the definition of fourier transform. We start with two quantum registers in the initial state $|0>|0>$

We apply fourier transform to the first register to get a uniform superposition of states.

$$|0>|0> \xrightarrow{FT_{\mathbb{Z}_q}} 1/\sqrt{q} \sum_{a \in \mathbb{Z}_q} |a>|0> \tag{13}$$

We then apply modular exponentiation and raise x to value in the first register. Again this can be done using appropriate quantum circuits in polynomial time(the time here is proportional to one modular exponentiation in classical system). We get the state:

$$1/\sqrt{q} \sum_{a \in \mathbb{Z}_q} |a>|x^a \ (mod\ n)> \tag{14}$$

At this point we observe the second register. Suppose the state observed is $x^k \ (mod\, n)$. The post-measurement of the system according to the property we mentioned in the beginning of the system would be:

$$\frac{1}{\sqrt{q}} \frac{1}{\sqrt{q/r}} \sum_{\forall b \in \mathbb{Z}_q, st\ x^b = x^k \ mod\ n} |b>|x^k\ mod\ n> \tag{15}$$

since r is the order of $x\ mod\ n$, $b = lr + k\ 0 \leq l < q/r$. So we have:

$$\frac{\sqrt{r}}{q} \sum_{l=0}^{q/r-1} |lr+k>|x^k\ mod\ n> \tag{16}$$

Now we again apply fourier transform to the first register to get.

$$\frac{\sqrt{r}}{q} \sum_{l=0}^{q/r-1} \sum_{u \in \mathbb{Z}_q} \omega^{(lr+k)u} |u>|x^k\ mod\ n> \tag{17}$$

$$or, \frac{\sqrt{r}}{q} \sum_{u \in \mathbb{Z}_q} \omega^{ku} \sum_{l=0}^{q/r-1} (\omega^r)^{lu}\ |u>|x^k\ mod\ n> \tag{18}$$

Now ω^r is the q/rth root. From equations 11 and 12 we get that the amplitudes of all states except those where $u = 0\ mod\ q/r$, cancel out. Only terms with $u = 0\ mod\ q/r$ remain. So first register state is:

$$\frac{\sqrt{r}}{q} \frac{q}{r} \sum_{l=0}^{r-1} |l\frac{q}{r}> \tag{19}$$

We get uniform superposition of all states that are multiples of q/r. With high probability $gcd(l, \frac{q}{r}) = 1$, in such a case $gcd(q, l\frac{q}{r}) = \frac{q}{r}$. So with high probability we get r by dividing q by $gcd(q, l\frac{q}{r})$.

The case when $r \not| q$ is dealt with using a little more advanced algebraic techniques but the basic quantum mechanics are the same. To keep things simple we skip this case in our report.

5. CONCLUSION

In this survey, we looked into the most recent algorithmic advances in the integer factorization problem. We also stressed out the importance of the problem, by

relating it to the security of the RSA cryptosystem. Currently, the champion of these algorithms is, as mentioned before, the Number Field Sieve, which has been used to factor numbers up to 576 bits long. However, it seems quite improbable that a PC-based implementation of the algorithm can do much better. That is why, other, non-algorithmic methods, have been proposed to improve the efficiency of the existing algorithms.

The most popular of these new ideas is distributed computing. Specifically, the huge breakthrough of factoring the RSA-129 challenge number after 17 years, wouldn't be possible without the help of 600 volunteers, that contributed the idle time of their computers to the project.

The second idea is constructing special purpose factoring machines. Adi Shamir and others have shown that using custom hardware for the the sieving process, which is the most computationally intensive step of the Number Field Sieve, could make factoring of 1024-bit long integers tractable. The only drawback of such a machine would be its cost: it would be worth almost 10M $.

REFERENCES

Jr. M. S. Manasse A. K. Lenstra, H. W. Lenstra and J. M. Pollard. The number field sieve. In *Proceedings of the twenty second annual symposium on theory of computing*, pages 564–572. ACM Press, 1990.

Mihir Bellare and Phillip Rogaway. *CSE 207 Course Notes*. http://www.cs.ucsd.edu/users/mihir/cse207/index.html, 2004.

Matthew Briggs. *An Introduction to the General Number Field Sieve*. Master's Thesis, Virginia Tech, 1998.

Thomas H. Hungerford. *Algebra*. Springer-Verlag, 1980.

H.W. Lenstra J.P. Buhler and Carl Pomerance. *The Development of the Number Field Sieve*. Springer, 1994.

Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994.

RSA Laboratories. *The RSA Challenge Numbers*.

Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. Birkhaäuser, 1985.

Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.

Song Y. Yan. *Primality Testing and Integer Factorization in Public-Key Cryptography*. Kluwer Academic Publishers, 2004.