The Quadratic Sieve Factoring Algorithm

Eric Landquist
MATH 488: Cryptographic Algorithms

December 14, 2001

1 Introduction

Mathematicians have been attempting to find better and faster ways to factor composite numbers since the beginning of time. Initially this involved dividing a number by larger and larger primes until you had the factorization. This trial division was not improved upon until Fermat applied the factorization of the difference of two squares: $a^2 - b^2 = (a - b)(a + b)$. In his method, we begin with the number to be factored: n. We find the smallest square larger than n, and test to see if the difference is square. If so, then we can apply the trick of factoring the difference of two squares to find the factors of n. If the difference is not a perfect square, then we find the next largest square, and repeat the process.

While Fermat's method is much faster than trial division, when it comes to the real world of factoring, for example factoring an RSA modulus several hundred digits long, the purely iterative method of Fermat is too slow. Several other methods have been presented, such as the Elliptic Curve Method discovered by H. Lenstra in 1987 and a pair of probabilistic methods by Pollard in the mid 70's, the $p - 1$ method and the ρ method. The fastest algorithms, however, utilize the same trick as Fermat, examples of which are the Continued Fraction Method, the Quadratic Sieve (and it variants), and the Number Field Sieve (and its variants). The exception to this is the Elliptic Curve Method, which runs almost as fast as the Quadratic Sieve. The remainder of this paper focuses on the Quadratic Sieve Method.

2 The Quadratic Sieve

The Quadratic Sieve, hereafter simply called the QS, was invented by Carl Pomerance in 1981, extending earlier ideas of Kraitchik and Dixon. The QS was the fastest known factoring algorithm until the Number Field Sieve was discovered in 1993. Still the QS is faster than the Number Field Sieve for numbers up to 110 digits long.

3 How it Works

If n is the number to be factored, the QS attemps to find two numbers x and y such that $x \not\equiv \pm y \pmod n$ and $x^2 \equiv y^2 \pmod n$. This would imply that $(x-y)(x+y) \equiv 0 \pmod n$, and we simply compute $(x-y, n)$ using the Euclidean Algorithm to see if this is a nontrivial divisor. There is at least a $\frac{1}{2}$ chance that the factor will be nontrivial. Our first step in doing so is to define

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

and compute $Q(x_1), Q(x_2), \ldots, Q(x_k)$. Determining the x_i will be explained below. From the evaluations of $Q(x)$, we want to pick a subset such that $Q(x_{i_1})Q(x_{i_2})\ldots Q(x_{i_r})$ is a square, y^2. Then note that for all x, $Q(x) \equiv \tilde{x}^2 \pmod n$. So what we have then is that

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

And if the conditions above hold, then we have factors of n.

3.1 Setting up a Factor Base and a Sieving Interval

With the basic outline of the QS in place, we need an efficient way to determine our x_i, and to get a product of the $Q(x_i)$ to be a square. Now to check to see if the product is a square, the exponents of the prime factors of the product need to all be even. So we will need to factor each of the $Q(x_i)$. Therefore, we want them to be small and to factor over a fixed set of small prime numbers (including -1), which we call our factor base. To make $Q(x)$ small, we need to select x close to 0, so we set a bound M and only consider values of x over the sieving interval $[-M, M]$. (Alternatively, we could have defined $Q(x) = x^2 - n$ and let the sieving interval be $[\lfloor\sqrt{n}\rfloor - M, \lfloor\sqrt{n}\rfloor + M]$.) Now if x is in this sieving interval, and if some prime p divides $Q(x)$, then

$$(x - \lfloor\sqrt{n}\rfloor)^2 \equiv n \pmod p,$$

so n is a quadratic residue (mod p). So the primes in our factor base must be primes such that the Legendre symbol

$$\left(\frac{n}{p}\right) = 1.$$

A second criterion for these primes is that they should be less than some bound B, which depends on the size of n. We will discuss this when we analyze the running time. We also say that this factor base is smooth: that every prime in the set is relatively small.

3.2 Sieving

Once we have a set of primes for our factor base, we begin to take numbers x from our sieving interval and calculate $Q(x)$, and check to see if it factors completely over our factor base. If it factors, it is said to have smoothness. If it does not, we throw it out, and we go on to the next element of our sieving interval. If we are dealing with a large factor base, though, it is incredibly inefficient to consider numbers one at a time and check all the primes in the factor base for divisibility. Instead, we will work with the entire sieving interval at once. If we are working in parallel, each processor would work over a different subinterval. Here is how it works. If p is a prime factor of $Q(x)$, then $p|Q(x+p)$. Conversely, if $x \equiv y \pmod p$, then $Q(x) \equiv Q(y) \pmod p$. So for each prime p in our factor base, we solve

$$Q(x) = s^2 \equiv 0 \pmod p, x \in \mathbf{Z}_p.$$

This can be solved using the Shanks-Tonelli Algorithm. We will obtain two solutions, which we call s_{1p} and $s_{2p} = p - s_{1p}$. Then those $Q(x_i)$ with the x_i in our sieving interval are divisible by p when $x_i = s_{1p}, s_{2p} + pk$ for some integer k.

There are a couple ways to do the sieving from here. One way is to take a subinterval (depending on the size of your memory), and put $Q(x_i)$ in an array for each x_i in the subinterval. For each p, start at s_{1p} and s_{2p} and divide out the highest power of p possible for each array element in arithmetic progression, recording the appropriate powers (mod 2) of p in a vector. You will have one vector for each of the factorable $Q(x_i)$ and each entry corresponds to a unique prime in the factor base. Once all the primes have had their turn sieving the interval, those array elements which are now 1 are those that factor completely over the factor base. The vector of powers of the primes can then be put into a matrix A. We repeat this process until we have enough entires in A to continue. This is explained below.

A second way is less exact, but is much quicker. Instead of working with the values of $Q(x)$ over some subinterval, record the number of bits of the $Q(x_i)$ in an array. For every element in the particular arithmetic progessions for p, subtract the number of bits of p. After every prime in the factor base has had their turn, those elements with remaining "bits" close to 0 are likely to be completely factorable over those primes. We need to take into account round-off error and the fact that many numbers are not square-free. For numbers that are not squre-free, we can sieve over the subinterval a second time picking out solutions to $Q(x) \equiv 0 \pmod{p^2}$ and so on. When all that is done, we set an upper bound on the number of bits we will consider. There will likely be fully factorable numbers that slip through at this point, but the time saved will more than make up for it. The numbers that meet this threshold condition will then be factored, by looking at the arithmetic progressions again so we can quickly nail down which primes divide which of the $Q(x_i)$.

Most implementations of the QS do not resieve the interval looking for powers of primes, so we will look at the sieving at a slightly deeper level. If we don't resieve with powers of primes, the threshold value becomes very important and powers of 2 becomes more significant. Fortunately we have a trick to deal with 2 to some extent. If $Q(x) = r^2 - n$, and we assume that r is odd, then $2|Q(x)$. We can work with n slightly so that a higher power of 2 always divides $Q(x)$. If we want 8 to always divide $Q(x)$ when it is even, we consider $n \pmod 8$. If $n \equiv 3, 7 \pmod 8$, then $2\|Q(x)$. If $n \equiv 5 \pmod 8$, then $4\|Q(x)$. Finally, if $n \equiv 1 \pmod 8$, then $8|Q(x)$. So to make 8 divide $Q(x)$ every time it is even, set $n := 5n$ if $n \equiv 3 \pmod 8$, set $n := 3n$ if $n \equiv 5 \pmod 8$, and $n := 7n$ if $n \equiv 7 \pmod 8$. Once the prime $p = 2$ is taken care of, sieve for the rest of the primes, subtracting the logarithms as above. Our threshold will then be

$$\frac{1}{2}ln(n) + ln(M) - Tln(p_{max})$$

where T is some value around 2 and p_{max} is the largest prime in the factor base. Silverman [8] suggested that $T = 1.5$ for factoring 30-digit numbers, $T = 2$ for 45-digit numbers, and $T = 2.6$ for 66-digit numbers, for example.

3.3 Building the Matrix

If $Q(x)$ does completely factor, then we put the exponents (mod 2) of the primes in the factor base into a vector as described above. We put all these vectors into the matrix A, so the rows represent the $Q(x_i)$, and the columns represent the exponents (mod 2) of the primes in the factor base. So, for example, if our factor base was $\{-1, 2, 3, 13, 17, 19, 29\}$ and $Q(x) = 2*3*17^2*19$, then the row corresponding to this $Q(x)$ would be $(0, 1, 1, 0, 0, 1, 0)$. Remember that we want the product of these $Q(x_i)$ to be a perfect square, so we want the sum of the exponents of every prime factor in the factor base to be even, and hence congruent to 0 (mod 2).

There may be several ways to obtain a perfect square from the $Q(x_i)$, which is good, since many of them will not give us a factor of n. So given $Q(x_1), Q(x_2), \ldots, Q(x_k)$, then we wish to find solutions to

$$Q(x_1)e_1 + Q(x_2)e_2 + \ldots + Q(x_k)e_k,$$

where the e_i are either 0 or 1. So if $\vec{a_i}$ is the row of A corresponding to $Q(x_i)$, then we want

$$\vec{a_1}e_1 + \vec{a_2}e_2 + \ldots + \vec{a_k}e_k \equiv \vec{0} \pmod 2.$$

This means that we need to solve

$$\vec{e}A = \vec{0} \pmod 2,$$

where

$$\vec{e} = (e_1, e_2, \ldots, e_k),$$

so via Gaussian elimination we find the spanning set of the solution space. Therefore we need to find at least as many $Q(x_i)$ as there are primes in the factor base. Each element of the spanning set corresponds to a subset of the $Q(x_i)$ whose product is a perfect square. Recall that at least half of the relations from the solution space will give us a proper factor. So if the factor base has B elements, and we have $B + 10$ values of $Q(x)$, then we have at least a $\frac{1023}{1024}$ probability of finding a proper factor. So we check solution vectors to see if the corresponding product of the $Q(x_i)$ and x_i yields a proper factor of n by doing a GCD calculation described at the beginning. If not, then check the next element in the spanning set. When a proper factor is found (you actually then have two factors), test those factors for primality.

If you are factoring an RSA modulus, then you know the factors are prime, so you are done.

4 Variant: The Multiple Polynomial Quadratic Sieve (MPQS)

As the name suggests, the MPQS uses several polynomials instead of $Q(x)$ in the algorithm, and was first suggested by Peter Montgomery. These polynomials are all of the form

$$Q(x) = ax^2 + 2bx + c,$$

where a, b, and c are chosen according to certain guidelines below. The motivation for this approach is that by using several polynomials, we can make the sieving interval much smaller, which makes $Q(x)$ smaller, which in turn will mean that a greater proportion of values of $Q(x)$ completely factor over the factor base.

In choosing our coefficients, let a be a square. Then choose $0 \leq b < a$ so that $b^2 \equiv n \pmod{a}$. This can only be true if n is a square mod q for every prime $q|a$. So we wish to choose a with a known factorization such that $\left(\frac{n}{q}\right) = 1$ for every $q|a$. Lastly, we choose c so that $b^2 - 4ac = n$. When we find an $Q(x)$ that factors well, notice that

$$aQ(x) = (ax)^2 + abx + ac = (ax + b)^2 - n.$$

So

$$(ax + b)^2 \equiv aQ(x) \pmod{n}.$$

Recall that a is square, so that $Q(x)$ must be.

Suppose that our sieving interval is $[-M, M]$. We wish to optimize M and the value of $Q(x)$ over this interval. One way to do this is to determine our coefficients so that the minimum and maximum values of $Q(x)$ on $[-M, M]$ have roughly the same magnitude, but be opposite in sign. Our minimum is at $x = -b/a$. Since we chose $0 \leq b < a$, $-1 < -b/a \leq 0$, and $Q(-b/a) =$

$-n/a$. So obviously our maximum is at $-M$ or M, and is roughly $\frac{a^2M^2-n}{a}$. We want this to be about n/a, so we choose

$$a \approx \frac{\sqrt{2n}}{M}.$$

One cause for concern with this method is the cost of switching polynomials. Pomerance [6] says that if the cost of switching polynomials is about 25-30% of the total cost, then it would be disadvantageous to use this method. When changing a polynomial, we obviously need new coefficients, but for each new polynomial we also need to solve $Q(x) \approx 0 \pmod p$ for each prime p in our factor base, which is the heaviest load in switching polynomials.

A scheme that Pomerance calls "self-initialization" can help to significantly reduce the cost of switching polynomials. The trick is to fix the constant a in several of the polynomials. We still want $a \approx \frac{\sqrt{2n}}{M}$, so let a be the product of k primes, p, which have a magnitude of about $\left(\frac{\sqrt{2n}}{M}\right)^{1/k}$ each, each of which satisfy $\left(\frac{n}{p}\right)$. We still need to find b such that $b^2 \equiv n \pmod a$. In fact since there are k prime factors of a, there are 2^{k-1} values for b. Then the initialization problem for the polynomials: finding solutions to $Q(x) \equiv 0 \pmod p$ for each polynomial and for each prime p in the factor base can be done all at once.

The main advantage to this variation is of course reducing the size of the factor base and sieving interval. Silverman [8] gives some suggestions for n up to 66 digits. The optimal size of the factor base varies with the machine being used, though a good estimate for the number of primes is about a tenth as many as the number of primes needed for the original QS. This also means that the sieving interval would be about a thousandth the size. Another advantage to this system is that it aids in parallel processing, with each processor working with a different polynomial. If each processor generates its own polynomial(s), then it can work fairly independently, only communicating with the central server when it has sieved the whole interval.

5 Variant: The Double Large Prime MPQS

This version of the QS was employed by Lenstra, Manasse, and several others in 1993 and 1994 to factor RSA-129 and reveal the message "The magic words are squeamish ossifrage." When RSA announced this challenge in 1977, they believed that it would take 23,000 years to factor the number and win the $100 prize. In actuality, the effort took only 8 months. What the Double Large Prime version does is that it considers partial factorizations of the $Q(x_i)$. In the sieving process, we hang onto $Q(x)$ and its partial factorization if we have:

$$Q(x) = \Pi {p_i}^{e_i} L, L > 1, L \leq {p_{max}}^2.$$

The factor L must be prime from its definition above. We can find these partial factorizations by incresing our theshold value after sieving by $2ln(p_{max})$. If we find another $Q(x)$ whose partial factorization contains L, then we can add L to our factor base and the product of the two $Q(x_i)$ will have the factor L^2. So we add these two factorizations to the matrix A. There may be other $Q(x_i)$ which factor over this larger factor base, so we add those in as well. It is estimated [7] that this cuts the sieving time by a sixth.

6 Gaussian Elimination

A critical step in the factoring process is the Gaussian elimination step. The matrix that is formed is huge, and almost every entry is a 0. Such a matrix is called sparse. Reducing this matrix using standard techniques from elementary linear algebra can be sped up considerably. A trivial consideration is that if we have a column with only one 1, we can eliminate the row associated with it. There is no possible way for that $Q(x)$ to be a factor in a square. There are two algorithms which do Guassian elimination of a matrix over a finite field: Wiedemann and Lanczos [10]. Of the two, Wiedemann [11] works better over $GF(2)$, which is the field we are in of course. The running time is approximately

$$O(B(w + Bln(B)ln(ln(B))).$$

where B is the number of primes in the factor base, and w is approximately the number of field operations required to multiply the matrix to a vector. If the matrix is sparse, as is the case here, w is very small. We also need $2B^2$ memory locations for storage.

7 Running Time

If the number of primes in our factor base was very small, we would not need very many factorizations of $Q(x)$ in order to obtain a possible factor of n. The problem with that though is finding every a few full factorizations would take a very long time, since a very small proportion of numbers factor over a small set of primes. If we were to create an enormous list of primes so that just about everything would factor of that factor base, then our problem would be getting all those numbers to create a large enough matrix to reduce. So the number of primes must be set to optimize performance. It turns out that this optimum value for the size of the factor base is roughly

$$B = \left(e^{\sqrt{ln(n)ln(ln(n))}}\right)^{\sqrt{2}/4}.$$

The sieving interval then turns out to be about the cube of this:

$$M = \left(e^{\sqrt{ln(n)ln(ln(n))}}\right)^{3\sqrt{2}/4}.$$

For example, when RSA-129 was factored in 1994, a factor base of 524,339 primes was used.

Some notes about components of the running time first. The optimum size of the factor base has been given, and further analysis tells us that the sieving time should be roughly three times the matrix reduction time. With B primes in the factor base, this step runs in less than $O(B^3)$ time, which gives us the size of the sieving interval. The heuristics estimate given previously says that the optimal umber of primes is

$$\left(e^{\sqrt{ln(n)ln(ln(n))}}\right)^{\sqrt{2}/4},$$

which determines the size of our sieving interval and so on. Put this all together and we have an asymptotic running time for the QS of

$$O\left(e^{\sqrt{1.125ln(n)ln(ln(n))}}\right).$$

With the improvements by Wiedemann in the Gaussian elimination, though, the running time comes down to being asymptotic to

$$O\left(e^{\sqrt{ln(n)ln(ln(n))}}\right).$$

The Number Field Sieve, by comparision, which is the fastest publicly known factoring algorithm has running time

$$O\left(e^{1.9223((ln(n))^{1/3}(ln(ln(n)))^{2/3})}\right).$$

The Number Field Sieve is the same as the QS from the sieving step on. It has a larger matrix to do the elimination step on, but the initial steps are much more efficient than the QS.

8 RSA

The security of the RSA cryptosystem relies on the difficulty of factoring integers. We have mentioned the successful factoring of a 129 digit RSA modulus. Currently RSA moduli of 512 bits, or about 155 digits would be feasible to factor, and in fact have been factored. In August, 1999, a team including Arjen Lenstra and Peter Montgomery factored a 512 bit RSA modulus using the Number Field Sieve in 8400 mips years (8400 million instructions per second-years) [2]. Current estimates say that a 768 bit modulus will be good until 2004, so for short term or personal use, such a key size is adequate. For corporate use, a 1024 bit modulua is suggested, and a 2048 bit modulus is suggested for much more permanent usage. These suggestions take into account possible advances in factoring techniques and for processor speed increases. Riesel [7] shows that it is possible to create an algorithm to factor integers in nearly polynomial time, so there is certainly room for improvements. However, if a quantum computer is ever built with a sufficient number of qubits, Peter Shor has discovered an algorithm to factor integers in polynomial time on it. Then RSA would have to be retired in favor of other encryption schemes as the moduli required to be secure would be much larger than what would be convenient.

References

[1] Bressoud, David. *Factorization and Primality Testing*. Springer-Verlag, New York, 1989.

[2] Cavallar, S., Lioen, W., te Riele, H., Dodson, B., Lenstra, A., Montgomery, P., Murphy, B., et al. "Factorization of a 512-bit RSA modulus, *Eurocrypt* (2000) (submitted).

[3] Cipra, Barry. *What's Happening in hte Mathematical Sciences, Volume 3*. AMS, Providence, RI, 1996.

[4] Gerver, J. "Factoring Large Numbers with a Quadratic Sieve," *Math. Comput.*, **41** (1983), 287-294.

[5] Kumanduri, Ramanujachary and Romero, Cristina. *Number Theory with Computer Applications*. Prentice hall, Upper Saddle River, NJ, 1998.

[6] Pomerance, Carl. *Cryptology and Computational Number Theory; Factoring*. AMS, Providence, RI, 1990.

[7] Riesel, Hans. *Prime Numbers and Computer Methods for Factorization 2nd Ed.* Birkhäuser, Boston, 1994.

[8] Silverman, Robert. "The Multiple Polynomial Quadratic Sieve Method of Computation," *Math. Comput.*, **48** (1987), 329-340.

[9] Song, Yan. *Number Theory for Computing*. Springer-Verlag, Berlin, 2000.

[10] Webster, J. "Linear Algebra Methods in Cryptography," unpublished paper, December, 2001.

[11] Wiedemann, D. "Solving Sparse Linear Equations over Finite Fields," *IEEE Trans. Inform. Theory*, **32** (1986), 54-62.