A Study of the Quadratic Sieve Algorithm with the Large Prime Variation for Factoring an Integer.

A Dissertation submitted in partial fulfillment of the requirement for the M. Tech (Computer Science) degree of the Indian Statistical Institute.

By
M. S. SRIDHAR.

Under the guidance of
Dr. Bimal Kumar Roy,
The Computer Science Unit.
Indian Statistical Institute,
203, Barrackpore Trunk Road,
Calcutta - 700 035.
India.

CERTIFICATE

This is to certify that the work described in the dissertation entitled **A Study of the Quadratic Sieve Algorithm with the Large Prime Variation for Factoring an Integer** has been undertaken by M.S. SRIDHAR under my guidance and supervision. The dissertation is found worthy of acceptance for the award of the Degree of *Master of Technology in Computer Science.*

Bimal Roy

(DR. BIMAL KUMAR ROY)

DATED :- July 17, 1995.

INDIAN STATISTICAL INSTITUTE,

CALCUTTA.

Acknowledgements

I am deeply grateful to Dr. Bimal Kumar Roy for his guidance and advice in this project, without which this could not have been possible. I am also deeply indebted to Mr. M. M. Sahoo, for his help and patience in reading the manuscript and for his valuable suggestions. I also thank Mr. S. Anburaj and Mr. V. Venugopal for helping me in preparing this report. I also thank my colleagues and friends for sharing their views regarding this subject. I also thank the Dean of Studies, the CSSC and the Computer Science Unit for providing their facilities.

(M. S. SRIDHAR)

Calcutta,
July, 1995.

Contents

Chapter 1

Introduction

1.1 History

The question of divisibility is arguably the oldest problem in Mathematics. Ancient peoples observed the cycles of nature: the day, lunar month, and the year, and assumed that each divided evenly into the next. Civilizations as separate as the Egyptians of ten thousand years ago and the Central American Mayans adopted a month of thirty days and a year of twelve months. Even when the inaccuracy of a 360-day year became apparent, they preferred to retain it and add five intercalary days. The number 360 retains its psychological appeal today because it is divisible by many small integers. The technical term for such a number reflects this appeal. It is called a "smooth" number.

1.2 Motivation

It is therefore surprising that a subject that is so very old should at the same time be so very new. Factorization and primality testing is a very hot area of current research. Among the factors creating interest in factorization, one cannot omit the advent of the RSA public key cryptosystem. The RSAC, as we shall henceforth address it, is an example of a public key or asymmetric encryption scheme.

In a public key scheme, the encryption and decryption keys are distinct. It is computationally infeasible or extremely difficult (though not impossible !) to obtain the decryption key from the encryption key. It is called a public key encryption scheme because not only does it eliminate the danger of the encryption scheme being stolen, but the encryption scheme can now be freely published so that anyone can send in a coded message.

Main Idea of RSAC:

Choose

$n = pq$, where p,q are large primes not close to one another.

e s.t $gcd(e, \phi(n)) = 1$ this is the encryption key.

d s.t $e \times d \equiv 1 \pmod{n}$ this is the decryption key.

(n,e) : public

(p,q) : concealed

d : private

Encoding Scheme :

M = message to be sent.

E = encrypted message.

$E \equiv M^e \pmod{n}$

Decoding Scheme :

$$M \equiv E^d \pmod{n}$$

It can be shown that once p,q are also made public then

$$\phi(n) = (p-1) \times (q-1)$$

and d can then be calculated by a method similar to the Euclidean algorithm. Thus the problem of finding d, the decryption key, has been reduced to finding the factorization of n. This gives us enough motivation why the subject of primality is a hot topic.

The success of the RSAC is based on the fact that it is very difficult to factor a number into its prime factors. For further information on the RSAC refer Köblitz[8].

Chapter 2

Factorization Techniques

2.1 Fermat's Algorithm

Fermat's idea is the following. If n is the number to be factored and if n can be written as a difference of two perfect squares:

$$n = x^2 - y^2,$$

then

$$n = (x - y) \times (x + y),$$

and we have succeeded in breaking n into two smaller factors. Furthermore, if we assume that the n we start with is odd (a safe assumption), then every representation of n as a product of two integers arises in this way.

To see this, let $n = a \times b$, where a and b are odd because n is odd. Let

$$x = \frac{a+b}{2},$$

and

$$y = \frac{a-b}{2}.$$

Then

$$x + y = a, \text{ and } x - y = b,$$

which implies

$$x^2 - y^2 = a \times b = n.$$

So, we start with $x = \lceil \sqrt{n} \rceil$ and $y = 0$ and try increasing y until $x^2 - y^2$ either equals or is less than n. In the first case we are done, in the second we increase x by one and iterate. We continue until we have success. If we set r equal to $x^2 - y^2 - n$, then we have success when $r = 0$.

This algorithm is further stream-lined by keeping track of $u = 2x + 1$ and $v = 2y + 1$ instead of x and y. All we are really interested in is keeping track of r. The variable u tracks the amount r, increases when x^2 is replaced by $(x+1)^2$, v tracks the amount r, decreases when y^2 is replaced by $(y+1)^2$. As x and y increase by one, u and v increase by 2.

For a formal description of the algorithm and analysis refer Knuth[3]. The major defect of this algorithm is that if let run long enough, it tends to prove the primality of n. In the worst case the algorithm takes $n - \sqrt{n}$ time, much worse than proving primality by trial division.

2.2 Kraitchik's Improvement

Maurice Kraitchik (1882-1957) realized that a major saving of time could be accomplished if instead of looking for x and y satisfying $x^2 - y^2 = n$, we would be satisfied if we obtained random x and y satisfying

$$x^2 \equiv y^2 \pmod{n}. \qquad (2.2.1)$$

Finding such a pair (x, y) no longer guarantees a factorization. But it does mean that n divides $x^2 - y^2 = (x - y) \times (x + y)$, and we now have at least a 50-50 chance that the prime divisors are distributed among the divisors of both of these factors so that the gcd of n and $x - y$ will be a nontrivial factor of n. That is to say, the gcd will be neither 1 nor n.

2.3 Factorization Techniques Using Kraitchik's Improvement

The two most effective methods using Kraitchik's improvement are the Continued Fractions Algorithm (CFRAC) by John Brillhart and Michael Morrison[7], and the Quadratic Sieve algorithm (QS) by Carl Pomerance[10] and its refinement called the Multiple Polynomial Quadratic Sieve (MPQS) by Peter Montgomery. We shall be concerned with only QS and a particular modification, called the Large Prime Variation of QS in this report. For further information regarding CFRAC refer [7].

Before we begin the discussion of the QS and the large prime variation of the QS we would like to make it clear that all the algorithms found so far for factoring a large number are probabilistic. So, success is not guaranteed. We also assume that the number to be factored is odd and not a perfect square. There are efficient tests to verify whether a given number is probably a perfect square. For further information on testing the primality of a number and to test whether it is a perfect square, we request the reader to refer Bressoud[1].

Chapter 3

Quadratic Sieve

3.1 Quadratic Residues

Definition: Given an integer n and a prime modulus p, if n is relatively prime to p and if there exits an integer t such that

$$n \equiv t^2 \pmod{p},$$

then we say that n is a quadratic residue modulo p.

Definition(LEGENDRE SYMBOL): Let p be an odd prime and n an integer. The Legendre symbol (n/p) is defined to be 0 if p divides n, $+1$ if n is a quadratic residue modulo p and -1 otherwise.

The Legendre symbol has certain properties which can be used to determine whether a given number is a quadratic residue of a given prime. For this we need to modify the definition of the Legendre symbol.

Definition(JACOBI SYMBOL): Let n be an integer and m any positive odd integer,

$$m = p_1 \times p_2 \times ... \times p_k,$$

where the p_i's are odd primes which may be repeated. The Jacobi symbol (n/m) has the value

$$(n/m) = (n/p_1) \times (n/p_2) \times ... \times (n/p_k)$$

where (n/p_i) is the usual Legendre symbol.

The Jacobi symbol (n/m) does not tell us whether n is a quadratic residue modulo m. It is rather a convenience that enables us to dispose of factorizations, except for pulling out powers of 2 which is easy, in the computation of the Legendre symbol. Note that if m is a prime, then the Legendre and Jacobi symbols are identical.

We state the following theorem without proof. For its proof refer Köblitz[8] and for a detailed description of the algorithm for computing the the Legendre symbol, refer Bressoud[1].

Theorem: *Let m and m' be odd positive integers, then*

(a) $(n/m) \times (n/m') = (n/(m \times m'))$,

(b) $(n/m) \times (n'/m) = ((n \times n')/m)$,

(c) $(n^2/m) = 1 = (n/m^2)$, *provided n and m are relatively prime,*

(d) *if* $n \equiv n' \pmod{m}$, *then* $(n/m) = (n'/m)$,

(e) $(-1/m) = 1$ *if* $m \equiv 1 \pmod{4}$, $= -1$ *if* $m \equiv -1 \pmod{4}$,

(f) $(2/m) = 1$ *if* $m \equiv 1$ *or* $-1 \pmod{8}$, $= -1$ *if* $m \equiv 3$ *or* $-3 \pmod{8}$,

(g) $(n/m) = (m/n)$ *if n and/or* $m \equiv 1 \pmod{4}$, $= -(m/n)$ *if n and* $m \equiv 3 \pmod{4}$.

For analysis of the algorithm refer Köblitz[8].

3.2 Dixon's Algorithm

The fundamental idea of trying to solve equation (2.2.1) was first proposed by John Dixon[2] in 1981.

We choose a random integer r and compute $g(r) \equiv r^2 \pmod{n}$. Now we factor $g(r)$. We are going to need lots of factored numbers, so we do not try too hard. We first do trial division up to 10,000, and if that does not work then we pick a different r. We keep doing this until we have more $g(r)$'s that are completely factored than primes below the limit of trial division. In this case we would need more than 1229 values of r for which we can factor $g(r)$.

Let $p_1, p_2, \ldots, p_{1229}$ be the first 1229 primes. If $g(r)$ factors completely then we can write it as

$$g(r) = {p_1}^{a_1} \times {p_2}^{a_2} \times \ldots \times {p_{1229}}^{a_{1229}},$$

where most of the a_i's will be 0. The factorization of $g(r)$ can be recorded by the vector

$$v(r) = (a_1, a_2, ..., a_{1229}).$$

If all of the entries of $v(r)$ are even, then $g(r)$ is a perfect square and we have solved equation (2.2.1) because

$$g(r) \equiv r^2 \pmod{n}.$$

Unfortunately, this is a very unlikely occurrence. But we have lots of these $v(r)$'s, more than the length of the vector. That means that we can find a sum of distinct $v(r)$'s which does have all even entries. This is accomplished by setting

$$w(r) = (b_1, b_2, ..., b_{1229}),$$

where

$$b_i = \begin{cases} 0, & \text{if } a_i \text{ is even.} \\ 1, & \text{if } a_i \text{ is odd.} \end{cases}$$

We then do Gaussian elimination modulo 2 on the resulting vectors to find a subset of the r's for which the sum of the corresponding $v(r)$'s has all even coordinates.

Since the sum of these selected $v(r)$'s is a vector with even entries, the product of the corresponding $g(r)$'s will be a perfect square. We get a congruence that looks like

$$g(r_1) \times g(r_2) \times ... \times g(r_t) \equiv r_1^2 \times r_2^2 \times ... \times r_t^2 \pmod{n},$$

both sides of which are perfect squares. We now have at least a 50-50 chance that this yields a factor of n. If not, we go back and find a different subset of the r's for which the $w(r)$'s are linearly dependent modulo 2. The defect with this method is that it is extremely costly in terms of computation time. To give some statistics, the largest prime factor of $g(r)$ is expected to be about $g(r)^{0.63}$. We can only factor $g(r)$ if the largest prime factor is less than 10,000. If n, and thus $g(r)$, is a 25-digit integer, we cannot factor it unless the largest prime divisor is roughly $g(r)^{0.16}$. The probability of that happening is only about 1/50,000, which means that we need to choose around 62 million values for r in order to get 1230 $g(r)$'s that we can factor.

3.3 Pomerance's Improvement

The crucial step of the algorithm is to recognize those $g(r)$'s which are completely factorizable over a set of primes chosen. Trial division was used to determine this. Given a prime p, from the set, is it possible to find those $g(r)$'s which are divisible by p without trial division ? It seems impossible. But, in 1981, Carl Pomerance suggested the possibility,and thus proved very effective in finding such $g(r)$'s.

His idea uses a sieve much similar to the sieve of Eratosthenes (refer [1]). We shall describe the algorithm below. Instead of choosing the r's at random, let

$$k = \lfloor \sqrt{n} \rfloor.$$

Take the values of $r : k+1, k+2, ...$ If we define $f(r)$ to be

$$f(r) = r^2 - n,$$

then $f(r) = g(r)$ as long as r lies between k and $\lfloor \sqrt{2n} \rfloor$.

We want to find the $f(r)$'s that factor into primes less than 10,000. Let p be any odd prime less than 10,000. We assume we have already done trial division on n up to 10,000 so we know that p does not divide n. If p divides $f(r)$, then

$$n \equiv r^2 \pmod{p};$$

the Legendre symbol $(n/p) = +1$. This means that we only need to consider those primes less than 10,000 for which $(n/p) = +1$, in other words about half of them. The set of primes which we try to divide into the $f(r)$'s is called a *factor base.*

If n is a quadratic residue modulo p, then it is the square of one of two residues modulo p:

$$n \equiv t^2 or (-t)^2 \pmod{p},$$

which means that r is congruent to either t or $-t$ modulo p. More importantly, if r is congruent to t or $-t$ modulo p, then p must divide $f(r)$.

We can now make two passes down our list of values of $f(r)$. Once we find the first r congruent to t modulo p, we know that $f(r)$ and every p^{th} $f(r)$ thereafter is divisible by p. Then we find the first r congruent ot $-t$ modulo p, and again run down the list. Since we know which $f(r)$'s are divisible by p without actually doing any division, we can just store the logarithm of $f(r)$ (to single precision) and subtract off the logarithm of p from the appropriate terms. When the remaining logarithm is sufficiently close to 0, we have found an r for which $f(r)$ factors.

But how does the algorithm take care of multiplicities ? The answer is it does not. For in order to take care of that we would first have to solve the congruence

$$x^2 \equiv n \pmod{p^a},$$

where p is an odd prime and where the exponent on p runs up to about

$$\frac{2 \times \log L}{\log p}$$

[1] where L is the largest prime in the factor base.

Consequently there are two problems. Firstly, the going is slow. When $p = 3$ we are subtracting $\log 3$ from every third entry. When $p = 311$, a single sieving run takes less than $(1/100)^{th}$ of the time because we are subtracting $\log 311$ from every 311th entry. Secondly, $\log 3 = 1.098...$ is much smaller than $\log 311 = 5.739...$. What this means that the convergence to 0's due to the smaller primes is slower.

We shall subsequently consider an idea suggested by Robert Silverman. We shall, for the time being make only a mention of it here. The proposition was to ignore the higher powers, but to be a little more generous on the cut-off for an entry that is accepted as probably completely factorable over the factor base. There will be few enough of these that trial division over the factor base can be used to decide if they really do factor completely.

In summary, there are three steps to the QS algorithm:

[1] All logarithms are to the base 2

1. Find a factor base and solve the congruences

$$x^2 \equiv n \pmod{p} \tag{3.3.1}$$

for each prime p in the factor base.

2. Perform the sieving operation to find sufficient $f(r)$'s which can be completely factored over the factor base.

3. Use Gaussian elimination to find a product of the $f(r)$'s which is a perfect square.

We shall examine the three steps in detail in the next three sections.

3.4 Solving Quadratic Congruences

Before we even begin the process of solving the quadratic congruences, we shall once again stress on our main goal - to increase the chance for $f(r) = r^2 - n$, to be divisible over the factor base. So we must choose our factor base in such a manner that it is neither too large (in which case we would end up spending most of our time in Gaussian elimination) nor too small (in which case the chance of a random $f(r)$ factoring is much less). In order to choose the size of the factor base refer Knuth and Trabb-Pardo[4]. So we can have an estimate of the number of primes that can be in the factor base.

Now we must increase the chance of a random $f(r)$ being factorizable over the factor base. For this we make use of a technique which uses a multiple of n, instead of n directly, in order to increase the odds that $f(r)$ will factor completely. We shall not elaborate this method here. We refer the interested reader to the references cited at the end of this report, for further details. We however, give an example here. We assume that the integer n, to be factored, is odd. If it is congruent to 1 modulo 8, then $r^2 - n$ is divisible by at least 8 whenever r is odd. If not what do we do?

Since the given number is odd and is not congruent to 1 modulo 8 it can be congruent to any of 3, 5, 7, modulo 8 only.

Case(a) If $n \equiv 3 \pmod{8}$, then multiply it by 5.

Case(b) If $n \equiv 5 \pmod{8}$, then multiply it by 3.

Case(c) If $n \equiv 7 \pmod{8}$, then multiply it by 7.

In all the three cases we have found a suitable multiplier of n, which increases the odds that $f(r)$ will be divisible by two. An important point to note is that the factor base should be found after the multiple of n has been chosen. We then set out to solve the quadratic congruence (3.3.1) for every prime in the factor base. The cases where $p \equiv 3 \pmod{4}$ or

$\equiv 5 \pmod 8$ are given in theorem 3.4.1. Theorem 3.4.2 is valid for any odd prime, but it is slightly slower than the procedures described in theorem 3.4.1. For the proofs we refer the reader to Bressoud[1].

Theorem 3.4.1: *If n is quadratic residue modulo the prime p and*

(1) *if $p = 4k + 3$, then*

$$x \equiv n^{k+1} \pmod p,$$

is a solution to equation (3.3.1).

(2) *if $p = 8k + 5$ and*

$$n^{2k+1} \equiv 1 \pmod p,$$

then

$$x \equiv n^{k+1} \pmod p,$$

is a solution to equation (3.3.1).

(3) *if $p = 8k + 5$ and $n^{2k+1} \equiv -1 \pmod p$, then*

$$x \equiv (4n)^{k+1} \times ((p+1)/2) \pmod p,$$

is a solution to equation (3.3.1).

Theorem 3.4.2: *Let n be a quadratic residue modulo an odd prime p and let h be chosen so that the Legendre symbol $((h^2 - 4n)/p)$ is -1. Define a sequence $v_1, v_2, \ldots$ by the recursion*

$$v_1 = h,$$
$$v_2 = h^2 - 2n \pmod p,$$
$$v_i = h \times v_{i-1} - n \times v_{i-2} \pmod p.$$

We then have that

$$v_{2i} = v_i^2 - 2n^i \pmod p, \tag{3.4.2}$$

and

$$v_{2i+1} = v_i \times v_{i+1} - n^i \times h \pmod p. \tag{3.4.3}$$

and a solution of equation (3.3.1) is given by

$$x \equiv v_{\frac{p+1}{2}} \times \frac{p+1}{2} \pmod p.$$

Theorem (3.4.2) was due to Lehmer[6] in 1969. The existence for h, in the above theorem is guaranteed. This algorithm can be very easily implemented. For any j we can compute v_j, given n, h and p in $\log j$ steps as follows:

(i) We first compute the binary expansion of j:

$$j = b_k b_{k-1} b_{k-2} ... b_1,$$

(ii) Use the following loop to compute the value of v_j

```
x : integer;    {       keeps track of v_(2k + 1).      }
v : integer;    {       keeps track of v_(2k).          }
w : integer;    {       keeps track of v_(2k + 2).      }

{ Initialize. }

m := n;
v := h;
w := (h * h - 2 * m) mod(p);

{ Compute. }

for i := (k - 1) downto 1 do
begin
x := (v * w - h * m) mod(p);
v := (v * v - 2 * m) mod(p);
w := (w * w - 2 * n * m) mod(p);
m := (m * m) mod(p);

if (b_k = 0) then w := x
else
begin
v := x;
m := (n * m) mod(p);
end;
end;

{ Terminate. }

write(v);
```

Analysis: The Compute loop is executed exactly $k - 1$ times which is equal to $\log j - 1$;

3.5 Sieving

We now move on to the next phase which is the most important phase as far as the time complexity is concerned. This is the step which takes the maximum time in the QS algorithm. We shall see how we can improve the performance of the algorithm.

In order to explain the techniques of the sieving process we choose a specific size of n, say 13. For 13 digit numbers we choose a factor base of size 30 and a sieving range of 10,000. Recall that the sieving process consists of making two passes down an array of 10,000 cells each initialized to $\log f(r)$, where the relation of r to the index value i is given by:

$$r = \lfloor \sqrt{n} \rfloor + i, \quad \text{where} \quad 1 \leq i \leq 10,000,$$

for every prime p in the factor base. During each pass the value of $\log p$ is subtracted from the cell whose index value corresponds to an r which satisfies eq. (3.3.1). Once the cell value is close enough to 0, we select the cell for trial division. Here we have not explained what the phrase "close enough" means. We also looked at the disadvantages of this approach in sec. (3.3).

We now give an elegant implementation of the same idea which was suggested by Robert D. Silverman. Firstly, instead of working with integers in the range

$$[\lfloor \sqrt{n} \rfloor + 1, \lfloor \sqrt{n} \rfloor + 10,000],$$

we work with integers in the range $[\lfloor \sqrt{n} \rfloor - 5000, \lfloor \sqrt{n} \rfloor + 5000]$. The idea is the closer the value of r is to $\lfloor \sqrt{n} \rfloor$ the smaller is the value of $f(r) = r^2 - n$ and hence, greater is the chance that it is factorable over the factor base. Of course, this gives rise to values of r which are negative. But this change can be easily accommodated by taking the first prime of the factor base to be -1. The new relation of r and i, the index variable of the sieve array is given by:

$$r = lower + i \; (1 \leq i \leq 10000),$$

where

$$lower = \lfloor \sqrt{n} \rfloor - 5000.$$

Secondly, rather than calculating the logarithm of the absolute value of $f(lower + i)$ 10,000 times, Silverman has suggested starting with a vector of zeros to which we add $\log p$ when

p divides the corresponding $f(r)$. If we are sieving over $2M$ [2], then the logarithm of the absolute value of $(\lfloor\sqrt{n}\rfloor - M + i)^2 - n$ will be approximately

$$target = \frac{\log n}{2} + \log M.$$

When the sieving is done, there will be few enough entries close to target that we can run trial division over the factor base on them to see exactly which ones factor completely.

How close is "close enough" ? If the remaining unfactored portion is less than the square of the largest prime in the factor base then it is prime. Even though we may not have a complete factorization over our factor base, we do get a complete factorization which, can still be used. This is called the Large Prime variation. Silverman's suggestion is therefore to set

$$close = target - T \times \log p_{max}$$

where p_{max} is the largest prime in the factor base and T is a constant near 2. For 13-digit numbers, $T = 1.5$ works nicely. T is called the prime tolerance factor.

In general, given an odd prime p and a solution t between 0 and p satisfying eq. (3.3.1), the first value of i to which we add $\log p$ is

$$p + (t - (\lfloor\sqrt{n}\rfloor - M) \pmod p).$$

If p divides the multiplier of n, in which case we would have only one solution viz. 0 satisfying eq.(3.3.1), eq.(3.3.1) has two distinct solutions t, $p - t$.

3.6 Gaussian Elimination

We now use Gaussian elimination to find a product of the $f(r)$'s which is a perfect square. Our running example is the same as above.

For each i for which we can factor $f(lower + i)$ we associate a string of thirty binary digits, each column corresponding to one of the thirty primes in the factor base. The digit is 1 if the corresponding prime appears to an odd power and 0 if it appears to an even power. It is convenient to represent the first prime -1, by the right-most digit and then read the columns from the right to left. Since the probability that a number is divisible by a large prime is small, most of the entries in this column will be zero so the elimination will be faster than for the smaller primes.

Thus we have a set of binary strings each of length L where L is the number of primes in the factor base. Let the number of such strings be p. Note that each of these strings corresponds to a vector in ${Z_2}^L$, which is, as we all know, a vector space over Z_2. Since

[2] in our example M = 5000

we have got $L + 1$ or more such binary vectors, they are linearly dependent. Several combinations of these vectors can sum upto 0.

We shall perform Gaussian elimination on these vectors, to obtain all such combinations. Firstly, we construct a binary matrix composed of these binary strings. We assume that the strings are represented rowwise. Next, we adjoin a $(p \times p)$ identity matrix to it. This is done to keep track of which rows are being added during the process. We then start looking for that row with a 1 in the first column. Once such a row is found every subsequent such row is added, modulo 2, with this row. Then the row located by the initial search is deleted. This process is repeated L times, at the end of which, there will be certain rows with their first L entries as 0. Their next p entries will tell us the history of which rows of the original matrix were added to get this. What this means is that the corresponding vectors add upto to zero sum. The implication of this is obvious - the powers of corresponding primes for all these vectors add upto to an even number, and thus we have a perfect square !

The process to find the solutions x, y for equation (2.2.1) then becomes very simple. Find the first row of the resultant matrix, having its first thirty entries 0. Read the right hand side starting with an index $i = 1$ and see if its entry is 1. If so, evaluate $(lower + i)$ (mod n). Initialize x to this value. Also, initialize a vector A, which keeps track of the powers of primes, to the vector corresponding to the prime factorization of $f(lower + i)$. The process of reading continues, incrementing the index at every step. For each such reading the same process described above is carried out, multiplying the value of $(lower+i)$ to x, modulo n, of course, and adding the prime representation vector to A.

After this, each entry of A is even. Divide each entry by 2, and evaluate the number with this a the prime representation vector, modulo n. This gives the value of y.

But all is not over. We still have to pass the acid test of testing whether the $gcd(n, x - y)$ is not 1 or n. If it so happens then we are successful and our process ends. If not we must continue looking for another vector with its first thirty entries as 0's and continue till success is achieved.

3.7 Large Prime variation

Before we start this section we would like to make certain remarks. Given a factor base $p_1, p_2, ..., p_L (p_1 = -1)$ and integers $v, a_1, a_2, ..., a_L$, satisfying,

$$v^2 \equiv \prod_{1 \leq j \leq L} {p_j}^{a_j} \pmod{n}, \tag{3.7.1}$$

we call eq. (3.7.1) a *small relation.*

Note that the entire process of QS is concerned with collecting sufficiently many such small relations, where v is $f(r = lower + i)$. As we have already remarked we may not always get complete factorizations over the factor base, but we often get complete factorizations.

So, all complete factorizations satisfy the following equation

$$v^2 \equiv q_v \times \prod_{1 \le j \le L} {p_j}^{a_j} \pmod{n}, \tag{3.7.2}$$

where

$$1 \le q_v < {p_L}^2,$$

which we shall refer to as a *partial relation* and call q_v as a *large prime.*

Can we some how make use of these relations ? The answer is in the affirmative. If u, v satisfy eq. (3.7.2) with $q_u = q_v$ and with powers $a_1, a_2, ...a_L$ and $b_1, b_2, ..., b_L$ respectively then

$$(u \times v)^2 = {q_u}^2 \times \prod_{1 \le j \le L} p_j^{a_j + b_j}, \tag{3.7.3}$$

which still satisfies (3.7.1) with

$$w = \frac{u \times v}{q_v}.$$

Note that w is an integer because q_u divides the right hand side of eq. (3.7.3) and so it must divide $(u \times v)^2$. So, it must divide $u \times v$.

So, during sieving we can modify our process to accept numbers with their unfactored portion less than ${p_{max}}^2$. The reader may wonder that we would then have to, in the Gaussian elimination step, work with a matrix of size $O(2L \times 2L)$ instead of $O(L \times L)$ (since we need at least $L + 1$ completely factorized $f(r)$'s) if this is to be accommodated. But this can easily be overcome.

What we shall do is to write down the binary vectors corresponding to each such factorization as usual at the beginning of the Gaussian step. We then sort these vectors in descending order of their large primes. We then start with the column corresponding to the largest large prime, and add a row with 1 in this position to a neighbouring row if and only if the neighbour also has a 1 in that position. If there is no such neighbour we simply delete this vector. If there are k such rows, add the first to each of the other $k - 1$ rows. We do this procedure for every subsequent large prime. Suppose we had started with vectors of size $L + t$, say, that at the end of this process, we would have these vectors having 0's in the last t positions. Note that all additions in this step are modulo 2.

The effect of this process is that we would then virtually work with binary vectors of size L, and continue with our Gaussian elimination. Note that, we must take care of the large primes while we calculate y, to obtain a solution to $x^2 \equiv y^2 \pmod{n}$.

This method is called the large prime variation method. With this modification. the algorithm runs about 2.5 times faster. Time estimates for this algorithm are given in section 3.9. For further discussion on the large prime variation refer C.Pomerance[9].

3.8 Time complexity of the QS algorithm with the large prime variation

$$t = e^{(1+c)\times\sqrt{\log n \times \log\log n}},$$

where $c = O(1)$. For further details, refer[8].

3.9 Conclusion and Some Implementation Results

In this report we have only discussed the large prime variation of the QS algorithm. Another improvement of this algorithm, called the Multiple Polynomial Quadratic Sieve [MPQS], was suggested by Peter Montgomery and its implementation is given in [11]. We refer the interested reader to it. Also, the choices for the prime tolerance factor T and the sieving range are given in [11].

We implemented the QS algorithm without and with the large prime variation on a 13-digit number, 499,94860,12441, with a factor base of size 30, a sieving range of size 10,000 and a prime tolerance factor of 1.5 and were successful in getting a factor, 999961. The time taken for sieving are 8 seconds and 5 seconds respectively.

Bibliography

1. David M. Bressoud, *Factorization and Primality Testing*, Undergraduate Texts in Mathematics, Springer-Verlag, 1989.

2. J. D. Dixon., "Asymptotically fast factorization of integers", *Math. Comput.* **36**(1981), pp. 255-260.

3. Donald E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2^{nd} ed., Addison-Wesley, Reading, MA, 1981.

4. Donald E. Knuth and Luis Trabb-Pardo, "Analysis of a simple factorization algorithm," *Theor. Comput. Sci.* **3**(1976), pp. 321-348.

5. James A. Davis, Diane B. Holdright, and Gustavus J Simmons, "Status Report on Factoring (at the Scandia National Laboratories)," pp. 183-215 in *Advances in Cryptology*, T. Beth, N. Cot, and I Ingemarsson, eds., Lecture Notes in Computer Science #209, Springer, Berlin, 1985.

6. D. H. Lehmer, "Computer technology applied to the theory of numbers," pp. 117-151 in *Studies in Number Theory*, W. J. LeVeque, ed., Mathematical Association of America Studies in Mathematics #6, 1969.

7. Michael A. Morrison and John Brillhart, "A Method of Factoring and the Factorization of $\mathcal{F}_7$," *Math. of Compt.*, **29**(1975), pp. 183-205.

8. Neal Köblitz, *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics, Springer-Verlag, 1987.

9. C. Pomerance, "Analysis and comparison of some integer factoring algorithms," pp. 89-139 in *Computational Methods in Number Theory, Part I,* H. W. Lenstra, Jr. and R. Tijdeman, eds., Mathematical Centre Tracts #154, Mathematisch Centrum, Amsterdam, 1982.

10. C. Pomerance, "The Quadratic Sieve Factoring Algorithm," pp. 169-182 in *Advances in Cryptology,* T. Beth, N. Cot, and I. Ingemarsson, eds., Lecture Notes in Computer Science #209, Springer, Berlin, 1985.

11. Robert D. Silverman, "The Multiple Polynomial Quadratic Sieve," *Math. of Comput.* **48**(1987), Number 177, 329-339.