A Quadratic Sieve on the n-Dimensional Cube

René Peralta [*]

Electrical Engineering and Computer Science Department
University of Wisconsin - Milwaukee

Abstract. Let N be a large odd integer. We show how to produce a long sequence $\{(X_i, Y_i)\}_{i=1}^{2^n}$ of integers modulo N which satisfy $X_i^2 \equiv Y_i$ modulo N, where $X_i > N^{1/2}$ and $|Y_i| < cN^{1/2}$. Our sequence corresponds to a Hamiltonian path on the n-dimensional hypercube C_n, where n is $\Theta(\log N / \log\log N)$. One application of these techniques is that, at each vertex of the hypercube, it is possible to search for equations of the form $U^2 \equiv V$ modulo N with V a product of small primes. The search is as in the quadratic sieve algorithm and therefore very fast. This yields a faster way of changing polynomials in the Multiple Polynomial Quadratic Sieve algorithm, since moving along the hypercube turns out to be very cheap.

1 Introduction

Given a large odd integer N, there is no known way of efficiently generating random congruences of the form $X^2 \equiv Y$ modulo N with Y substantially smaller than $N^{1/2}$. One reason for wanting to generate such congruences is that they can be used to factor N. The Continued Fraction Algorithm [2] factors N by generating many such congruences, choosing the ones for which Y factors over a small prime factor base FB, and then solving a linear system of equations in order to create one congruence $X^2 \equiv Z^2$ modulo N which, if $X \not\equiv \pm Z$, yields a proper factor $p = GCD(X+Z, N)$ of N. An important bottleneck in the Continued Fraction Algorithm is the cost of testing whether Y factors over FB. This is done by trial division for each prime in the factor base. The Quadratic Sieve Algorithm [6] considers a sequence $\{(X_i, Y_i)\}_{i=1}^{M}$ of M pairs where $X_i = \lfloor\sqrt{N}\rfloor + i$ and $Y_i = X_i^2 - N$. Since Y_i is given by an integer quadratic polynomial, it is easy to predict which Y_i's will be divisible by a given prime p. The values of i which generate $Y_i \equiv 0 \bmod p$ lie on two arithmetic progressions $\alpha \pm kp$ and $\beta \pm kp$ ($k = 0, 1, \ldots$). The cost of avoiding trial division is that the Y_i's are of order $O(MN^{1/2})$ and therefore they are less likely to factor over FB than the Y's generated by the Continued Fraction Algorithm. However, avoiding trial division more than compensates for the increased size of the Y_i's. A variation on the Quadratic Sieve Algorithm is the Multiple Polynomial Quadratic Sieve [8] (MPQS), which uses several polynomials as a way to fight the increase in the size of the Y_i's. The latter is currently the algorithm of choice for factoring integers which are about one hundred digits long.

* Supported by National Science Foundation grant No. CCR-8909657

We show how to produce a long sequence $\{(X_i, Y_i)\}_{i=1}^{2^n}$ of integers modulo N which satisfy $X_i^2 \equiv Y_i$ modulo N, where $X_i > N^{1/2}$ and $| Y_i | < cN^{1/2}$. Our sequence corresponds to a Hamiltonian path on the n-dimensional hypercube C_n, where n is $\Theta(\log N / \log\log N)$. One application of these techniques is that, at each vertex of the hypercube, it is possible to search for equations of the form $U^2 \equiv V$ modulo N with V smooth. The search is as in the quadratic sieve algorithm and therefore very fast. This yields a factoring algorithm which is faster than the Multiple Polynomial Quadratic Sieve algorithm, since moving along the hypercube turns out to be very cheap. The asymptotics of the new algorithm are as in MPQS. Therefore it is not asymptotically as fast as the recently discovered Number Field Sieve algorithm [3, 1].

2 Generating "small" quadratic congruences

Let N be a large odd integer. We will use the symbol "$\equiv$" to denote modular congruence, and we will restrict the use of "$=$" to equality. Let s, t be such that

- $t = \prod_{j=1}^{n} p_j$, where the p_j's are distinct primes (n will be chosen later).
- The prime 2 may be among the p_j's if and only if $N \equiv 1$ modulo 4.
- N is a quadratic residue modulo each p_j.
- s satisfies $s^2 \equiv N$ modulo t^2 and $|s| < t^2$.

Lemma 1. *Let $c = t/N^{1/4}$. Let $x \equiv s/t$ modulo N and $y \boxminus x^2$ modulo N where y is the member of the residue class of x^2 with smallest absolute value. Then $|y| < c_1 N^{1/2}$ where $c_1 = Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\}$.*

Proof. Since $s^2 \equiv N$ modulo t^2, we have $s^2 = kt^2 + N$ for some (possibly negative) integer k. Then $y \equiv \frac{s^2}{t^2} \equiv \frac{s^2 - N}{t^2} = \frac{kt^2 + N - N}{t^2} = k$, where congruence is modulo N. Thus we may choose $y = k$. Since $s^2 < t^4$ and $t = cN^{1/4}$, we have

$$|y| = |k| = \left| \frac{s^2 - N}{t^2} \right| \leq Max\{t^2 - \frac{N}{t^2}, \frac{N}{t^2}\} = N^{1/2} Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\}. \square$$

Thus, if $c \in (1, \sqrt{\frac{1+\sqrt{5}}{2}})$, then $|y| < N^{1/2}$ (the golden mean strikes again!). Also note that $Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\}$ is minimized at $c = 2^{1/4}$. For $c < 2^{1/4}$, the bound is $\frac{1}{c^2}$. For $c > 2^{1/4}$, the bound is $c^2 - \frac{1}{c^2}$.

By construction, there are 2^n square roots of N modulo t^2. Given the p_j's, it is a simple matter to compute one such root s_1. By the Chinese Remainder Theorem, we may think of s_1 as an n-tuple $(\alpha_1, \ldots, \alpha_n)$, where $\alpha_j^2 \equiv N$ modulo p_j^2. Then the complete set of square roots of N modulo t^2 is given by $(\pm\alpha_1, \ldots, \pm\alpha_n)$ for all choices of signs $\pm$. Any member of this set can be easily calculated as a sum $\sum_{j=1}^{n} \delta_j \alpha_j b_j$ modulo t^2 where

- δ_j is the sign at the j-th coordinate.
- b_j is the unique element of Z_{t^2} which is 1 modulo p_j^2 and 0 modulo p_i^2 for $j \neq i$.

Note that there are two possible values for each α_j. *We will choose α_j such that $b_j\alpha_j$ modulo t^2 is less than $\frac{t^2}{2}$.*

The maximum size of n can be estimated from the familiar relation $\sum_{p\le x}\ln p \sim x$, where the sum is over all primes p less than or equal to x. Since N is typically a quadratic residue modulo half of the first $2n$ primes we can estimate the maximum n from $2n \sim \pi(x)$ where x satisfies $\prod_{p<x} p = N^{1/2}$ (in this way, the product of the approximately n primes for which N is a quadratic residue is approximately $N^{1/4}$). This implies $\sum_{p\le x}\ln p \sim \frac{1}{2}\ln N$, and so $x \sim \frac{1}{2}\ln N$. Thus $n \sim \frac{1}{2}\pi(x) \sim \frac{1}{2}\pi(\frac{1}{2}\ln N) \sim \frac{1}{4}\frac{\ln N}{\ln\ln N - \ln 2} = \Theta(\log N/\log\log N)$.

Example : the RSA modulus

The 129-digit RSA modulus

$$N_{RSA} = 114381625757888867669235779976146612010218296721242362562561$$

$$842935706935245733897830597123563958705058989075147599290026879543541$$

is a quadratic residue modulo the 20 primes

$$\{2,5,17,19,29,37,41,43,47,59,79,97,101,103,107,113,131,151,157,163\}.$$

Letting t be the product of all these primes except 79, we get $t \sim 1.01N_{RSA}^{1/4}$. Thus, for this case we get $n = 19$.

3 Traversing the hypercube

The set of square roots of N modulo t^2 can be thought of as the n-dimensional hypercube, where we connect two roots if and only if they differ at exactly one sign. A Hamiltonian path on the hypercube is defined by a starting point s_1 and the sequence $\{k_i\}_{i=1}^{2^n-1}$ of coordinate changes, e.g. $k_{130} = 8$ means the 130th move on the Hamiltonian path is a change of sign at coordinate 8. Let $\mu_i = +1$ if move i switches a $-$ sign for a $+$ sign and $\mu_i = -1$ if move i switches a $+$ sign for a $-$ sign. Let $\gamma_j \equiv \alpha_j b_j$ modulo t^2, where α_j, b_j are as defined in the previous section. Note that, by our choice of α_j, we have $0 < \gamma_j < \frac{t^2}{2}$ for all j. Then we may define the i-th square root of N modulo t^2 by

$$s_{i+1} = s_i + 2\mu_i\gamma_{k_i} - \omega_i t^2$$

where ω_i is a correction factor to make $s_i \in (0, t^2)$. Note that ω_i is always $-1, 0$, or $+1$. The sequence of ω_i's can be easily computed from the few most significant bits of the γ_j's, and if simply allowed to be 0, the values of s_i will remain in a small interval (as shown by our next example).

An n-dimensional cube C_n is composed of two $(n-1)$-dimensional cubes $C_{n-1}^{(1)}, C_{n-1}^{(2)}$ whose vertices are connected in a 1-1 fashion. Thus, a simple way to traverse the n-cube is

- traverse $C^{(1)}_{n-1}$;
- move to $C^{(2)}_{n-1}$;
- traverse $C^{(2)}_{n-1}$.

The recursive procedure works because on moving to $C^{(2)}_{n-1}$ the algorithm finds itself at a node which is, up to isomorphism, the same starting node as in $C^{(1)}_{n-1}$. *From now on we will assume the Hamiltonian path on the n-cube is generated by this procedure.*

Example : Traversing the 3-cube

A traversal of the 3-cube yields

i	1	2	3	4	5	6	7
k_i	1	2	1	3	1	2	1
μ_i	-1	-1	+1	-1	-1	+1	+1

Using $\omega_i = 0$ for all i, this table gives the following values for the s_i's.

$$\begin{aligned}
s_2 &= s_1 - 2\gamma_1 \\
s_3 &= s_2 - 2\gamma_2 \\
&= s_1 - 2(\gamma_1 + \gamma_2) \\
s_4 &= s_3 + 2\gamma_1 \\
&= s_1 - 2\gamma_2 \\
s_5 &= s_4 - 2\gamma_3 \\
&= s_1 - 2(\gamma_2 + \gamma_3) \\
s_6 &= s_5 - 2\gamma_1 \\
&= s_1 - 2(\gamma_1 + \gamma_2 + \gamma_3) \\
s_7 &= s_6 + 2\gamma_2 \\
&= s_1 - 2(\gamma_1 + \gamma_3) \\
s_8 &= s_7 + 2\gamma_1 \\
&= s_1 - 2\gamma_3
\end{aligned}$$

The value of s_6 could be larger in absolute value than $2t^2$, but no larger than $3t^2$. This illustrates the point that if the ω_i's are not used then s_i still remains in the interval $(-nt^2, nt^2)$. [2] Also note that the sequence of k_i's can be generated

[2] Different Hamiltonian paths on the hypercube might yield different bounds.

in linear time (to generate the sequence for C_n simply put n between two copies of the sequence for C_{n-1}).

Thus the integer recurrence

$$s_{i+1} = s_i + 2\mu_i \gamma_{k_i} - \omega_i t^2$$

together with

$$x_i \equiv (s_i/t) \text{ modulo } N \quad ; \quad y_i \equiv (x_i)^2 \text{ modulo } N$$

yield

$$|\, y_i| < Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\} N^{1/2}$$

if y_i is the member of the residue class of x_i^2 with smallest absolute value. We will now produce an integer recurrence for the y_i's.

Note that, modulo N,

$$\begin{aligned} y_{i+1} &\equiv (s_{i+1}/t)^2 \\ &\equiv \left(\frac{s_i + 2\mu_i\gamma_{k_i} - \omega_i t^2}{t} \right)^2 \\ &\equiv \left[(\frac{s_i}{t})^2 \text{ modulo } N \right] + \frac{(2\mu_i\gamma_{k_i} - \omega_i t^2)^2}{t^2} + \frac{2s_i(2\mu_i\gamma_{k_i} - \omega_i t^2)}{t^2} \\ &\equiv y_i + \frac{(2\mu_i\gamma_{k_i} - \omega_i t^2)^2}{t^2} + \frac{2s_i(2\mu_i\gamma_{k_i} - \omega_i t^2)}{t^2} \end{aligned}$$

Since $s_i^2 \equiv N$ modulo t^2 and $s_{i+1}^2 \equiv N$ modulo t^2 , we have $(2\mu_i\gamma_{k_i} - \omega_i t^2)^2 + 2s_i(2\mu_i\gamma_{k_i} - \omega_i t^2)$ is congruent to 0 modulo t^2. Thus

$$\frac{(2\mu_i\gamma_{k_i} - \omega_i t^2)^2}{t^2} + \frac{2s_i(2\mu_i\gamma_{k_i} - \omega_i t^2)}{t^2}$$

is an integer which is easily seen to be of order $N^{1/2}$. This means the integer recurrence

$$y_{i+1} = y_i + \frac{(2\mu_i\gamma_{k_i} - \omega_i t^2)^2}{t^2} + \frac{2s_i(2\mu_i\gamma_{k_i} - \omega_i t^2)}{t^2}$$

holds. By lemma 1, y_1 can be chosen so that $|y_1| < c_1 N^{1/2}$ where $c_1 = Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\}$. Again by lemma 1, the integer recurrence generates y_i's whose absolute value is less than $Max\{c^2 - \frac{1}{c^2}, \frac{1}{c^2}\}N^{1/2}$.

Now let us traverse the hypercube "modulo p", where p is a small prime. This simply means generating the sequence of y_i's modulo p. Assume p is not a factor of t. We may write

$$y_{i+1} = y_i + \Psi_i + s_i \Upsilon_i$$

$$s_{i+1} = s_i + \Delta_i$$

where

$$\Psi_i = \frac{(2\mu_i\gamma_{k_i} - \omega_i t^2)^2}{t^2} \quad ; \quad \Upsilon_i = \frac{2(2\mu_i\gamma_{k_i} - \omega_i t^2)}{t^2} \quad ; \quad \Delta_i = (2\mu_i\gamma_{k_i} - \omega_i t^2).$$

Note that Ψ_i, Υ_i and Δ_i can take on at most $6n$ values (μ_i can take on two values, ω_i can take on three values, and γ_{k_i} can take on n values). Thus, Ψ_i, Υ_i and Δ_i can be read from precomputed tables, of size $6n$ and indexed by μ_i, ω_i, k_i. Thus, computing (y_{i+1}, s_{i+1}) modulo p from (y_i, s_i) modulo p involves one multiplication and three additions modulo p. The cost of computing $s_{i+1} \bmod p$ from $s_i \bmod p$ is one addition modulo p. This fact will be used in section 4.

Note that precomputation is not possible if p divides t, since then Ψ_i and Υ_i may not be defined modulo p. More specifically, Ψ_i, Υ_i are not defined modulo p_{k_i}.

Also note that, if N is not a quadratic residue modulo p, then p does not divide y_i. This can be shown as follows: Suppose N is not a quadratic residue modulo p. Then p does not divide t^2 and therefore p divides $y_i = \frac{s_i^2 - N}{t^2}$ if and only if p divides $s_i^2 - N$. But if this was the case then $s_i^2 \equiv N$ modulo p, which would contradict the assumption that N is not a quadratic residue modulo p.

4 A factoring algorithm

The algorithm consists of visiting A vertices of the hypercube and, at each vertex s, finding the values of λ for which

$$z_\lambda = (s/t + \lambda t)^2 \bmod N \qquad\qquad (\lambda \in -M..M)$$

is B-smooth. The optimal values of A, B, and M will follow from the analysis of the algorithm. We actually do not compute the z_λ, but rather find the values of λ for which z_λ is B-smooth. So that the z_λ are "small", we will choose $t \sim \frac{N^{1/4}}{\sqrt{M}}$.

Notice that

$$z_\lambda \equiv (s/t)^2 + \lambda^2 t^2 + 2s\lambda \bmod N = \frac{s^2 - N}{t^2} + \lambda^2 t^2 + 2s\lambda.$$

Thus we set

$$\frac{s^2 - N}{t^2} + \lambda^2 t^2 + 2s\lambda \equiv 0 \bmod p.$$

This yields

$$\lambda \equiv (-s \pm \sqrt{N})t^{-2} \bmod p,$$

where $\sqrt{N}$ is a modular square root. Therefore z_λ is divisible by p for all $\lambda = kp + D_s$ and all $\lambda = kp + E_s$, where

- k is an integer;
- $D_s = (-s + \sqrt{N})t^{-2} \bmod p$;
- $E_s = (-s - \sqrt{N})t^{-2} \bmod p = D_s - 2\sqrt{N}t^{-2} \bmod p$.

Thus, the "good" λ modulo p are in arithmetic progressions. Therefore standard sieving techniques can be used to find those λ for which z_λ is B-smooth.

We may precompute $\sqrt{N} \bmod p$, $t^{-2} \bmod p$, and $-2\sqrt{N}t^{-2} \bmod p$. Therefore computing D_s and E_s involves

- one addition to compute $s \bmod p$ (see section 3);
- one addition and one multiplication to compute D_s;
- one addition to compute E_s.

Thus the total cost of moving from one vertex of the hypercube to another is, essentially, three additions and one multiplication modulo p for each prime in the factor base. This is much cheaper than the cost of changing polynomials in the Multiple Polynomial Quadratic Sieve.

We now show that the z_λ's are about $MN^{1/2}$ in absolute value. Recall that $t \sim \frac{N^{1/4}}{\sqrt{M}}$ and consider

$$z_\lambda \equiv (s/t + \lambda t)^2 \equiv (s/t)^2 + \lambda^2 t^2 + 2s\lambda \bmod N.$$

As in the proof of lemma 1 we have $(s/t)^2 \bmod N = k$ where $s^2 = kt^2 + N$. Since $s^2 < t^4 < N$, we have that k is negative. By lemma 1, $|(s/t)^2 \bmod N| = |k| \leq MN^{1/2}$. Assuming, for simplicity, that $t < \frac{N^{1/4}}{\sqrt{M}}$, we have $\lambda^2 t^2 \leq M^2 \frac{N^{1/2}}{M} = MN^{1/2}$. Since $s < t^2$, we have $|2s\lambda| < 2\frac{N^{1/2}}{M}M = 2N^{1/2} << MN^{1/2}$. Thus z_λ is, essentially, the difference of two numbers in the range $0..MN^{1/2}$. We conclude that our algorithm is faster than the Multiple Polynomial Quadratic Sieve because

- vertices in the hypercube correspond to polynomials in MPQS.
- for each vertex, the cost of sieving $2M$ locations is the same in our algorithm as in MPQS.
- the size of the quadratic residues considered is, as in MPQS, about $MN^{1/2}$.
- changing polynomials is much more expensive than changing vertices of the hypercube. Therefore the optimal value for the size of the hypercube path is bigger than the optimal number of polynomials in MPQS. This means that M will be smaller in our algorithm and therefore $MN^{1/2}$ will be smaller. Thus, our algorithm will generate smaller quadratic residues than MPQS.

In practice there are many speedups to be included in an implementation of this factoring algorithm. All the enhancements described in [5] can be used with this algorithm. A rough estimate of how much faster this algorithm is than MPQS can be obtained as follows:

Let T be the running time of the algorithm, in terms of arithmetic operations on single-precision numbers. Let V be the cost of moving from one vertex to another. Let S be the cost of sieving at each vertex. Suppose we sieve modulo all primes less than B for which N is a quadratic residue. There are about $\frac{\pi(B)}{2}$ such primes. It takes four operations per prime to make a move on the hypercube. Therefore we can estimate V by $2\pi(B)$. We can estimate S by $1/2 \sum_{p<B} 4M/p$ since for each prime p in the factor base about $4M/p$ locations of an accumulator

array need be updated. We can estimate this sum by $6M$. Thus our estimate for T is $A(S+V) = 2A\pi(B) + 6AM$.

Let $F(y,x)$ be the probability that a random number in Z_y factors over primes smaller than x. The number of quadratic residues considered by our algorithm is $2AM$, and each can be thought of as a random number in $Z_{\lfloor MN^{1/2} \rfloor}$. Thus about $2AMF(MN^{1/2}, B)$ of the quadratic residues will be B–smooth. Since we need about $\pi(B)/2$ smooth quadratic residues, we set

$$2AMF(MN^{1/2}, B) = \pi(B)/2.$$

Approximating $\pi(B)$ by $B/\ln B$ and $F(y,x)$ by $(\ln x/\ln y)^{\ln y/\ln x}$ (see [4]), our problem is to minimize

$$2AB/\ln B + 6AM$$

subject to

$$A = \frac{B}{4M \ln B}\left(\frac{\ln(MN^{1/2})}{\ln B}\right)^{\frac{\ln(MN^{1/2})}{\ln B}}$$

The solution to this optimization problem can be approximated numerically. For $N \sim 10^{100}$, optimal values are

$$A = 1.2 \cdot 10^5; B = 1.4 \cdot 10^8; M = 1.3 \cdot 10^7; T = 1.1 \cdot 10^{13}$$

Assuming the cost of changing polynomials in MPQS is $50\pi(B)$, [3] the numbers for MPQS are

$$A = 5.8 \cdot 10^3; B = 1.9 \cdot 10^8; M = 4.5 \cdot 10^8; T = 1.9 \cdot 10^{13}$$

Thus it appears that our techniques significantly improve on the running time of MPQS.

4.1 Remarks

1. The running-time predictions given above are very crude estimates. The true test of the running time of this algorithm will be its implementation.
2. In practice, the ω_i's defined in section 2 can be set to zero without a significant cost in the running time of the algorithm. Doing so has the advantage of diminishing the memory requirements of the algorithm.
3. In practice, the factors of t should not be small primes. This is because the numbers being sieved have a chance of $1/p$ of being divisible by p when p divides t (as opposed to $2/p$ when p is an odd prime in the factor base which does not divide t). The resulting loss of smoothness is significant for small p. Because of this, the t we use in practice may not have as many factors as the optimization of parameters requires.

[3] This is 25 times as expensive as in our algorithm. Changing polynomials in MPQS involves arithmetic with large numbers. Hence the cost will depend on the particular implementation of large number arithmetic. The number 25 was arrived at using "ln++", a c++ package developed at UWM.

4. Pomerance, Smith, and Tuler [7], and Montgomery (reported in [7]) propose ways of speeding up MPQS which are similar to the one proposed here. Their methods can be combined with the techniques being proposed here. It appears that doing so may further improve the running time of the algorithm.
5. The number of vertices of the hypercube to be visited by the factoring algorithm should be at most 2^{n-1}, where n is the number of factors of t. Otherwise duplication of polynomials occurs, since $(s/t+\lambda t)^2$ and $((t^2-s)/t+\lambda t)^2$ are essentially equivalent.

Acknowledgments

I am indebted to anonymous referees for numerous suggestions, including remarks 3 and 4 above.

References

1. L.M. Adleman. Factoring numbers using singular integers. In *Proceedings of the 23th Annual ACM Symposium on the Theory of Computing*, pages 64–71, 1991.
2. D.H. Lehmer and R.E. Powers. On factoring large numbers. *Bull. Amer. Math. Soc.*, 37:770–776, 1931.
3. A.K. Lenstra, H. W. Lenstra, M.S. Manasse, and J.M. Pollard. The number field sieve. In *Proceedings of the 22th Annual ACM Symposium on the Theory of Computing*, pages 564–572, 1990.
4. D. Knuth and L. Trabb Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3:321–348, 1976.
5. A.K. Lenstra and M.S. Manasse. Factoring by electronic mail. In *Advances in Cryptology - proceedings of EUROCRYPT 89*, volume 434, pages 355–371. Springer-Verlag, 1990.
6. C. Pomerance. Analysis and comparison of some integer factoring algorithms. 154:89–139, 1982.
7. C. Pomerance, J.W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM Journal on Computing*, 17(2):387–403, 1988.
8. R. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48(177):329–339, 1987.