Faster Factoring of Integers of a Special Form

René Peralta *[1,2] Eiji Okamoto **[2]

[1] Electrical Engineering and Computer Science Department
University of Wisconsin-Milwaukee, USA
[2] School of Information Science
Japan Advanced Institute of Science and Technology

Abstract. A speedup of Lenstra's Elliptic Curve Method of factorization is presented. The speedup works for integers of the form $N = PQ^2$, where P is a prime sufficiently smaller than Q. The result is of interest to cryptographers, since integers with secret factorization of this form are being used in digital signatures. The algorithm makes use of what we call "Jacobi signatures". We believe these to be of independent interest.

1 Introduction

It is not known how to efficiently factor a large integer N. Currently, the algorithm with best asymptotic complexity is the Number Field Sieve (see [6]). For numbers below a certain size (currently believed to be about 120 integers), either the Quadratic Sieve [14] or the Elliptic Curve Method [7] are faster. Which of these algorithms to use depends on the size of N and of the smallest prime factor of N. When the size of the smallest factor is sufficiently smaller than $\sqrt{N}$, the Elliptic Curve Method is the fastest of the three. In this note we consider integers of the form $N = PQ^2$, where P is prime. These numbers are of importance in Cryptography. For example, they are used in Okamoto et. al.'s ESIGN scheme and associated crypto-algorithms (see [5, 11]). We show how to significantly speed up the Elliptic Curve Method when applied to this type of integers. Our method relies on a novel way to decide whether two numbers x, y modulo N are equal modulo P. The usual way to decide this is to compute $GCD(x - y, N)$. Our method is slower and is only probabilistic. However, it has the advantage that finding $i \neq j$ such that $x_i \equiv x_j \bmod P$ in a list $x_1, \ldots, x_m$ takes essentially linear time in m. This leads to a particularly efficient "large prime" variation of the elliptic curve method of factorization.

* Supported in part by NSF Grant CCR-9207204 and by Komatsu Endowed Chair of JAIST.

** Member IEICE.

2 Jacobi signatures

Consider the following problem:

given numbers $x_1 \neq x_2$ modulo $N = PQ^2$ determine whether $x_1 \equiv x_2 \bmod P$.

The natural way to do this is to calculate $GCD(x_1 - x_2, N)$. If $x_1 \equiv x_2 \bmod P$, then the GCD will be P or PQ. In either case we can obtain P, Q and solve the problem. But notice that a Jacobi symbol calculation may rule out equality mod P. i.e. if $(\frac{x_1}{N}) \neq (\frac{x_2}{N})$ then we know $x_1 \not\equiv x_2 \bmod P$. In fact, if r is any number mod N and $(\frac{x_1+r}{N}) \neq (\frac{x_2+r}{N})$, then $x_1 \not\equiv x_2 \bmod P$. This is because, for all x mutually prime with N, we have $(\frac{x}{N}) = (\frac{x}{PQ^2}) = (\frac{x}{P})$.

This suggests the following definition:

Definition 1. Given $x \in Z_N$ and a vector $\mathbf{r} = (r_1, \ldots, r_k)$ of numbers modulo N, the "Jacobi $\mathbf{r}$-signature" of x is the bit vector

$$J_{\mathbf{r}}(x) = ((\frac{x + r_1}{N}), \ldots, (\frac{x + r_k}{N})).$$

Our first algorithm tests whether $x_1 \equiv x_2 \bmod P$ by simply computing their Jacobi $\mathbf{r}$-signatures for a random vector $\mathbf{r}$.

Algorithm 1

1. For $i = 1$ to k let r_i be a random number modulo N.
2. If $J_{\mathbf{r}}(x_1) = J_{\mathbf{r}}(x_2)$ then return("probably equal"), else return("not equal").

If $x_1 \equiv x_2 \bmod P$ then the algorithm returns "probably equal" with probability 1. If $x_1 \not\equiv x_2 \bmod P$ then the algorithm returns "probably equal" if and only if the Jacobi symbol of $r_i^2 + (x_1 + x_2)r_i + x_1 x_2$ is 1 for all i. When viewed as a polynomial in r_i, this expression has non-zero discriminant modulo P. It is a standard assumption, supported by much empirical data, that such quadratic polynomials behave in a pseudo-random manner when sampled at random points r_i. Thus we expect the probability of error in Algorithm 1 to be proportional to $\frac{1}{2}^k$. This holds regardless of the distribution of x_1, x_2.

If x_1, x_2 are random numbers modulo N then we can set $r_i = i$ in Algorithm 1. We then use the heuristic assumption that the Jacobi symbols $(\frac{x+i}{P}) \quad (i = 1, \ldots, k)$ are approximately independent random variables for random x. Recent work [2, 4, 12, 1] on the distribution of Jacobi symbols lends much theoretical support to this assumption. In particular, these sequences are ϵ-biased, a type of distribution which can be proven to closely approximate the distribution of fair independent coins [10, 12]. Thus, we again expect the probability of error to be proportional to $\frac{1}{2}^k$. This motivates the following:

Definition 2. Given $x \in Z_N$ and an integer k, the "Jacobi signature" (of length k) of x is the bit vector

$$J^{(k)}(x) = ((\frac{x+1}{N}), \ldots, (\frac{x+k}{N})).$$

Algorithm 2

1. If $J^{(k)}(x_1) = J^{(k)}(x_2)$ then return("probably equal"), else return("not equal").

Algorithms 1 and 2 are mostly curiosities when the problem is just to compare two numbers, since it is less efficient than the straight-forward way. However, the idea of Jacobi signatures can be put to good use when the problem involves many x_i's. Consider the following problem:

given numbers $x_1, ..., x_m$ modulo $N = PQ^2$, find $i \neq j$ such that $x_i \equiv x_j \bmod P$.

The straight-forward method of doing this requires $O(m^2)$ GCD operations (one for each pair (x_i, x_j)). This requires $O(m^2 \lg^2 N)$ steps. But consider the following randomized algorithm:

Algorithm 3

1. For $i = 1$ to $\lfloor \lg m \rfloor$ let r_i be a random number modulo N.[1]
2. For $i = 1$ to m compute $J_{\mathbf{r}}(x_i)$.
3. For each pair x_i, x_j with $J_{\mathbf{r}}(x_i) = J_{\mathbf{r}}(x_j)$ compute $GCD(x_i - x_j, N)$ to see if $x_i = x_j \bmod P$.

The Jacobi signatures can be viewed as integers in the range $\{0, \ldots, m\}$. Therefore determining all collisions can be done in linear time in m and in the number of collisions. The number of expected collisions is

$$\frac{\binom{m}{2}}{2^{\lfloor \lg m \rfloor}} < m.$$

[1] Throughout this paper we use the notation $\lg x$ to denote the logarithm base 2 of x and $\log x$ to denote the natural logarithm of x.

Thus the running time of Algorithm 3 is dominated by the computation of m Jacobi signatures of length $\lfloor \lg m \rfloor$. This involves computing $m\lfloor \lg m \rfloor$ Jacobi symbols. Therefore the running time is $O(m \lg m \lg^2 N)$, a substantial improvement over the straight-forward method. Asymptotically, this is the same complexity of methods which use the Fast Fourier Transform [9]. But in practice this is much simpler and faster. As with algorithm 2, if the x_i's are random numbers, we may use a simpler algorithm:

Algorithm 4

1. For $i = 1$ to m compute $J^{(\lfloor \lg m \rfloor)}(x_i)$.
2. For each pair x_i, x_j with $J^{(\lfloor \lg m \rfloor)}(x_i) = J^{(\lfloor \lg m \rfloor)}(x_j)$ compute $GCD(x_i - x_j, N)$ to see if $x_i = x_j \bmod P$.

3 Speeding the Elliptic Curve Method

Consider the following simple factorization algorithm:

Algorithm 5

1. Generate $x_1, ..., x_m$ at random modulo N, where $m = \lfloor \sqrt{P} \rfloor$. Use Algorithm 4 to find a pair of x_i, x_j such that $GCD(x_i - x_j, N) = P$.

This algorithm finds P, with probability about $\frac{1}{2}$, using $\lfloor \sqrt{P} \rfloor \lfloor \lg \sqrt{P} \rfloor$ Jacobi symbol calculations.[2]

Algorithm 5 is mostly a curiosity, since Pollard's-ρ algorithm [13] would find P in $O(\sqrt{P})$ operations modulo N. However, things get much better if $P - 1$ happens to be "semi-smooth"[3]:

Definition 3. Let $z \leq y$ be positive integers. An integer n is "semi-smooth" with respect to y and z if all the prime factors of n are bounded above by z, with the possible exception of a prime factor p such that $p \leq y$ and $p^2 \nmid n$.

A large-prime variation of Pollard's "$p - 1$" method can be implemented as follows (the algorithm is parametrized by a bound B and a positive integer m). We assume we have some approximation $\hat{P}$ to P (e.g. we could know the number of decimal digits of P).

[2] The probability bound here is asymptotically $1 - 1/\sqrt{e} \approx 0.4$. Throughout this paper we replace probabilities such as this by the statement "about 1/2". This simplifies the exposition while avoiding the use of the "big O" notation when we want to make clear that there are no large hidden constants in our expressions.

Algorithm 6

1. Define $L_B = \prod p^i$, where the product is over all prime powers p^i such that $p < B$ and $p^i < \hat{P} < p^{i+1}$
2. Choose u at random from Z_N^*.
3. Calculate $\omega = u^{L_B} \bmod N$ (note that L_B need not be calculated explicitly).
4. Let $x_1, ..., x_m$ be random elements in the multiplicative subgroup of Z_N^* generated by ω.
5. Use Algorithm 4 to find a pair of x_i, x_j such that $GCD(x_i - x_j, N) = P$.

As a consequence of the Birthday Paradox, Algorithm 6 will find P with probability about $\frac{1}{2}$ if the order of the subgroup of Z_p^* generated by ω is no more than m^2. A sufficient condition for $card(<\omega>) < m^2$ is that $P-1$ be semi-smooth with respect to m^2 and B.

Generating the x_i's can be done in a variety of ways. The simplest is to let $x_i = \omega^{r_i}$ where r_i is a random number modulo N. A more efficient way is to let $x_i = \omega^{r_i}$ for $i = 1, \ldots, 2\sqrt{m}$, and then generate the remaining $x_i's$ by multiplying these in pairs. Note that more than m elements of $<\omega>$ will be generated in this case. This is necessary to keep the probability of success at about $\frac{1}{2}$. Nevertheless, step 4 takes a linear (in m) number of modular multiplications.

Tables of asymptotic semi-smoothness probabilities appear in [3]. As an example of the implications of Algorithm 6 consider the following: about 3% of primes P are such that $P-1$ is semi-smooth with respect to $P^{1/2}$ and $P^{1/5}$. Setting $B = \hat{P}^{1/5}$ and $m = \hat{P}^{1/4}$, we have that, for 3% of integers $N = PQ^2$, repeated executions of Algorithm 6 will find P in $O(P^{1/4} \lg P)$ operations modulo N. If $P \approx Q$, then Algorithm 6, is an $O(N^{1/12} \lg^2 N)$ algorithm for 3% of such integers N.[3]

We now show that these techniques can also be used in the context of Lenstra's elliptic curve algorithm [7]. The following algorithm is parametrized by a bound B and a positive integer m. The exponent L_B is as in Algorithm 6.

Algorithm 7

1. Pick a random elliptic curve $\mathcal{C}$ together with a point $(a, b) \in \mathcal{C}$.
2. Let $\omega = (a, b)^{L_B}$, where exponentiation is in $\mathcal{C}$.

[3] Setting $P \approx Q$ is actually recommended for some cryptographic applications, the security of which depends on the hardness of factoring N [11]

3. Let $(x_1, y_1), \ldots, (x_m, y_m)$ be random elements in the subgroup of $\mathcal{C}$ generated by ω.
4. Use Algorithm 4 to find a pair of x_i, x_j such that $GCD(x_i - x_j, N) = P$.

As in Algorithm 6, step 3 can be performed in a linear number of operations in $\mathcal{C}$ by generating $O(\sqrt{m})$ random points in $\mathcal{C}$ and using these to generate the remaining $O(m)$ points. Algorithm 7 finds P with probability about $\frac{1}{2}$ if the order of the subgroup of $\mathcal{C}$ generated by ω is no more than m^2. A sufficient condition for $card(< w >) < m^2$ is that the number of points in $\mathcal{C}$, when considered modulo P, be semi-smooth with respect to m^2 and B. The cardinality of $\mathcal{C}$ modulo P is in the range $(P - 2\sqrt{P}, P + 2\sqrt{P})$. It is a standard assumption that this cardinality is roughly uniformly distributed in this range. Recall that the probability that a random number around P is semi-smooth with respect to $P^{1/2}$ and $P^{1/5}$ is about 3%. Thus, setting $B = P^{1/5}$ and $m = P^{1/4}$, we have that about 33 iterations of Algorithm 7 are necessary to find P. Each iteration runs in time $O(P^{1/4})$ operations in $\mathcal{C}$. A more detailed analysis follows.

3.1 Analysis of running time

The expected running time of this and all known variations on the elliptic curve algorithm is of the order $e^{\sqrt{2 \log P \log\log P}}$. The exact expected running time depends on several implementation details. The main cost factors are:

- the cost of calculating x^t in $\mathcal{C}$. Express this cost as $E \lg t$, where E is an implementation dependent constant.
- the cost of calculating xy in $\mathcal{C}$ given x and y. Call this cost M (for "multiplication cost").
- the cost of calculating a Jacobi symbol modulo N. Call this cost J (for "Jacobi").

Using modular multiplication as the unit of measurement we estimate the following values for these constants

$$E = 15 \qquad\qquad M = 10 \qquad\qquad J = 8.$$

These are rough estimates. Actual values depend on the architecture of the machine, the size of N, and the parametrization of elliptic curves used.

Recall that Algorithm 7 is parametrized by B and m. Let $B = \hat{P}^{1/u}$ and $m^2 = \hat{P}^{1/v}$. Since we are assuming we have a reasonable estimate of P, setting $\hat{P} = P$ will not significantly affect our analysis. The running time of step 1 is not significant. Step 2 involves $\pi(B) \approx B/\log B$ exponentiations in $\mathcal{C}$, where the exponents are of order

P. Since $B \approx P^{1/u}$, step 2 will involve approximately $1.44uEP^{1/u}$ multiplications modulo N. Omitting lower-order quantities, step 3 involves approximately m multiplications in $\mathcal{C}$, i.e. $MP^{0.5/v}$ multiplications modulo N. Step 4 involves calculating $m \lg m = JP^{0.5/v} \lg P/(2v)$ Jacobi symbols modulo N. The expected number of times the algorithm will have to be iterated is $\frac{1}{\sigma(u,v)}$, where $\sigma(u,v)$ is the asymptotic probability that a random number modulo P is semi-smooth with respect to $P^{1/v}$ and $P^{1/u}$. Thus the expected cost, in number of multiplications modulo N, of Algorithm 7 is approximately

$$T(u,v,P) = \frac{1.44uEP^{1/u} + MP^{0.5/v} + (2v)^{-1}JP^{0.5/v}\lg P}{\sigma(u,v)}.$$

The analogous equation for the cost of the straight-forward implementation of the elliptic curve algorithm is

$$L(u,P) = \frac{1.44uEP^{1/u}}{\rho(u)}$$

where ρ is Dickman's rho function.

The minimum values of these functions are easily computable using the algorithm for estimating semi-smoothness probabilities given in [3]. For $P \approx 10^{40}$ the function $T(u,v,P)$ optimizes at $u = 5.8$ and $v = 3.1$. The cost with this parametrization is about 10^{12} modular multiplications. The function $L(u,P)$ optimizes at $u = 5.5$ with a cost of $2.6 \cdot 10^{13}$ modular multiplications. Thus the large-prime variation is expected to be about 25 times as fast as the straight-forward implementation. For $P \approx 10^{60}$, the speedup increases to a factor of 40. Of course, extracting 60-digit factors from numbers which are over 200 digits currently could only be accomplished using special-purpose hardware.

Thus our methods are a significant improvement over the Montgomery's speedups reported in [8]. Although, of course, his speedups are for general numbers and ours is for a special class of numbers only. [4].

At first sight, the ratio of the functions L over T might appear to represent the speedup due to our methods. This is not quite so, however, since the true speedup is obtained by evaluating the functions at their optimal values of u and v, given P. As shown in the (numerically estimated) figures given above, the functions do not optimize at the same value of u. Nevertheless, a crude estimate might be obtained by assuming common

[4] It is hard to gauge what is exactly the speedup in Montgomery's paper. Our reading of it puts it at perhaps a 4-fold improvement. As mentioned earlier, the techniques which might compete with ours asymptotically are those introduced by Pollard and based on the FFT. Montgomery writes that Pollard "expresses uncertainty as to when this becomes practical". In a later paper, Montgomery states that making practical use of the FFT in speeding up the ECM is an "open problem".[9]

values of u and setting v at $v = 0.5u + \epsilon$ for small positive ϵ. Then the second and third terms in the numerator for $T(u, v, P)$ are asymptotically dominated by the first term and hence we may ignore them. The ratio $L(u, P)/T(u, v, P)$ then becomes essentially $\frac{\sigma(u,v)}{\rho(u)}$. This in turn can be approximated by $\frac{\tau(u)=\sigma(u,0.5u)}{\rho(u)}$. There are no known closed formulas for this ratio. However, extremely accurate estimates can be obtained using the numerical methods of Bach and the first author [3]. Here is a table computed using these methods.

u	τ(u) $\sim$ speedup
3.0	9.202408e+00
3.5	1.379083e+01
4.0	1.962950e+01
4.5	2.669337e+01
5.0	3.499470e+01
5.5	4.464122e+01
6.0	5.558695e+01
6.5	6.791888e+01
7.0	8.163088e+01
7.5	9.675625e+01
8.0	1.133222e+02
8.5	1.313376e+02
9.0	1.508334e+02
9.5	1.718186e+02
10.0	1.943151e+02

The table is mostly of theoretical interest. It gives the reader a rough idea of the rate at which the speedup achieved by our methods increases as the function $T(u, v, P)$ optimizes at increasing values of u. The function $\tau(u)$ is an overestimate of actual speedups due to the two heuristic assumptions discussed above.

4 Acknowledgments

We would like to acknowledge the helpful suggestions of two anonymous referees.

References

1. Alon, N., Goldreich, O., Hastad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Structures and Algorithms **3** (1992) 289–304.
2. Bach, E.: Realistic analysis of some randomized algorithms. Journal of Computer and System Sciences **42** (1991) 30–53.
3. Bach, E., Peralta, R.: Asymptotic semismoothness probabilities. Mathematics of Computation **65** (1996) 1701–1715.
4. Damgård, I.: On the randomness of Legendre and Jacobi sequences. In Advances in Cryptology - Proceedings of CRYPTO 88 (1989) Lecture Notes in Computer Science, Springer-Verlag pp. 163–172.
5. Fujioka, A., Okamoto, T., Miyaguchi, S.: ESIGN: An efficient digital signature implementation for smart cards. In Advances in Cryptology - Proceedings of EUROCRYPT 91 (1991) pp. 446–457.
6. Lenstra, A., H.W. Lenstra, J., Eds. The development of the number field sieve vol. 1554 of Lecture Notes in Mathematics. Springer-Verlag 1993.
7. Lenstra, H. W.: Factoring integers with elliptic curves. Annals of mathematics **126** (1987) 649–673.
8. Montgomery, P. L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48** (1987) 243–264.
9. Montgomery, P. L., Silverman, R. D.: An FFT extension to the p-1 factoring algorithm. Mathematics of Computation **54** (1990) 839–854.
10. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM Journal on Computing (1993).
11. Okamoto, T., Fujioka, A., Fujisaki, E.: An efficient digital signature scheme based on an elliptic curve over the ring z_n. In Advances in Cryptology - Proceedings of CRYPTO 92 (1993) vol. 740 of Lecture Notes in Computer Science, pp. 54–65.
12. Peralta, R.: On the distribution of quadratic residues and nonresidues modulo a prime number. Mathematics of Computation **58** (1992) 433–440.
13. Pollard, J. M.: A Monte Carlo method for factorization. BIT **15** (1975) 331–334.
14. Pomerance, C.: The quadratic sieve factoring algorithm. In Advances in Cryptology - Proceedings of EUROCRYPT 84 (1985) vol. 209 of Lecture Notes in Computer Science, Springer-Verlag pp. 169–182.

This article was processed using the LaTeX macro package with LLNCS style