Factoring $N = p^r q$ for Large r

(Extended Abstract)

Dan Boneh[*] dabo@cs.stanford.edu

Glenn Durfee[†] gdurf@cs.stanford.edu

Nick Howgrave-Graham nahg@maths.bath.ac.uk

Abstract

We present an algorithm for factoring integers of the form $N = p^r q$ for large r. Such integers were previously proposed for various cryptographic applications. When $r \approx \log p$ our algorithm runs in polynomial time (in $\log N$). Hence, we obtain a new class of integers that can be efficiently factored. When $r \approx \sqrt{\log p}$ the algorithm is asymptotically faster than the Elliptic Curve Method. Our results suggest that integers of the form $N = p^r q$ should be used with care. This is especially true when r is large, namely r greater than $\sqrt{\log p}$.

Keywords: RSA, Factoring, Lattice basis reduction, LLL.

1 Introduction

In recent years moduli of the form $N = p^r q$ have found many applications in cryptography. For example, Fujioke et al. [3] use a modulus $N = p^2 q$ in an electronic cash scheme. Okamoto and Uchiyama [10] use $N = p^2 q$ for an elegant public key system. Last year Takagai [16] observed that RSA decryption can be performed significantly faster by using a modulus of the form $N = p^r q$. In all these applications, the factors p and q are approximately the same size. The security of the system relies on the difficulty of factoring N.

We show that moduli of the form $N = p^r q$ should be used with care. In particular, let p and q be primes of a certain length, say 512 bits each. We show that factoring $N = p^r q$ becomes easier as r gets bigger. For example, when r is on the order of $\log p$ our algorithm factors N in *polynomial time*. This is a new class of moduli that can be factored efficiently. When $N = p^r q$ with r on the order of $\sqrt{\log p}$ our algorithm factors N *faster than the current best method* — the elliptic curve algorithm (ECM) [8]. Hence, if p and q are 512 bit primes, then $N = p^r q$ with $r \approx 23$ can be factored by our algorithm faster than with ECM. These results suggest that moduli of the form $N = p^r q$ with large r are inappropriate for cryptographic purposes. In particular, Takagai's proposal [16] should not be used with a large r.

Suppose p and q are k bit primes and $N = p^r q$. When $r = k^\epsilon$ our algorithm (asymptotically) runs in time $T(k) = 2^{(k^{1-\epsilon})+O(\log k)}$. Hence, when $\epsilon = 1$ the modulus N is roughly k^2 bits long and the algorithm will factor N in polynomial time in k. Already when $\epsilon = \frac{1}{2}$ the algorithm asymptotically performs better than ECM. The algorithm requires only polynomial space (in $\log N$).

[*]Supported by NSF contract #CCR-9732754.

[†]Supported by Certicom.

We implemented the algorithm to experiment and compare it to ECM factoring. It is most interesting to compare the algorithms when $\epsilon \approx 1/2$, namely $r \approx \sqrt{\log p}$. Unfortunately, since $N = p^r q$ rapidly becomes to large to handle we could only experiment with small p's. Our largest experiment involves 96 bit primes p and q and $r = 9$. In this case N is 960 bits long. Our results suggest that although our algorithm is asymptotically superior, for such small prime factors the ECM method is better. Our experimental results are described in Section 4.

The problem of factoring $N = p^r q$ is related to that of factoring moduli of the form $N = p^2 q$. Previous results due to Peralta and Okamato [11] and also due to Pollard and Bleichenbacher show that ECM factoring can be made more efficient when applied to $N = p^2 q$. Our results say very little about moduli with such a small r.

Our approach is based on techniques due to Coppersmith [2]. We use a simplification due to Howgrave-Graham [4]. This technique uses results from the theory of integer lattices. The next section provides a brief introduction to lattices. In Section 3, we describe the factoring algorithm for integers of the form $N = p^r q$ for large r. In Section 4 we discuss our implementation of the algorithm and provide examples of factorizations completed. In Section 5, we compare this approach to existing methods, and describe classes of integers for which this algorithm is the best known method.

2 Lattices

Let $u_1, \dots, u_d \in \mathbb{Z}^n$ be linearly independent vectors with $d \le n$. A lattice L spanned by $\langle u_1, \dots, u_d \rangle$ is the set of all integer linear combinations of $u_1, \dots, u_d$. We say that the lattice is full rank if $d = n$. We state a few basic results about lattices and refer to [9] for an introduction.

Let L be a lattice spanned by $\langle u_1, \dots, u_d \rangle$. We denote by $u_1^*, \dots, u_d^*$ the vectors obtained by applying the Gram-Schmidt orthogonalization process to the vectors $u_1, \dots, u_d$. We define the determinant of the lattice L as

$$\det(L) := \prod_{i=1}^{d} \|u_i^*\|$$

If L is a full rank lattice then the determinant of L is equal to the determinant of the $d \times d$ matrix whose rows are the basis vectors $u_1, \dots, u_d$.

Fact 2.1 (LLL) *Let L be a lattice spanned by $\langle u_1, \dots, u_d \rangle$. Then the LLL algorithm, given $\langle u_1, \dots, u_d \rangle$, will produce a vector v satisfying*

$$\|v\| \;\le\; 2^{d/2} \det(L)^{1/d}.$$

The algorithm runs in time quartic in the size of its input.

3 Factoring $N = p^r q$

Our goal in this section is to develop an algorithm to factor integers of the form $N = p^r q$. The main theorem of this section is given below. We use non-standard notation and write $\exp(n) = 2^n$. Similarly, throughout the paper all logarithms should be interpreted as logarithms to the base 2.

Theorem 3.1 *Let $N = p^r q$ where $q < p^c$ for some c. The factor p can be recovered from N, r, and c*

by an algorithm with a running time of:

$$\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot O(\gamma),$$

where γ is the time it takes to run LLL on a lattice of dimension $O(r^2)$ with entries of size $O(r \log N)$. The algorithm is deterministic, and runs in polynomial space.

Note that the factor γ is polynomial in $\log N$. It is worthwhile to consider a few examples using this theorem. For simplicity we assume $c = 1$, so that both p and q are roughly the same size. Taking c as any small constant gives similar results.

- When $c = 1$ we have that $\frac{c+1}{r+c} = O(\frac{1}{r})$. Hence, the larger r is, the easier the factoring problem becomes. When $r = \epsilon \log p$ for a fixed ϵ, the algorithm is polynomial time.
- When $r \approx \log^{1/2} p$, then the running time is approximately $\exp(\log^{1/2} p)$. Thus, the running time is (asymptotically) slightly better than the Elliptic Curve Method (ECM) [8]. A comparison between this algorithm and existing algorithms is given in Section 5.
- For small r, the algorithm is exponential time.
- When c is large (e.g. on the order of r) the algorithm becomes exponential time. Hence, the algorithm is most effective when p and q are approximately the same size. All cryptographic applications of $N = p^r q$ we are aware of use p and q of approximately the same size.

The proof of Theorem 3.1 is based on a technique due to Coppersmith [2] and Howgrave-Graham [4]. The basic idea is to guess a small number of the most significant bits of p and factor using the guess. As it turns out, we can show that the larger r is, the fewer bits of p we need to guess.

In [2] Coppersmith shows that given half the bits of p (the most significant bits) one can factor integers of the form $N = pq$ in polynomial time, provided p and q are about the same size. To do so Coppersmith proved an elegant result showing how to find small solutions of *bivariate* equations *over the integers*. Surprisingly, Theorem 3.1 does not follow from Coppersmith's results. Coppersmith's bivariate theorem does not seem to give an efficient algorithm for factoring $N = p^r q$. Recently, Howgrave-Graham [4] showed an alternate way of deriving Coppersmith's results for *univariate modular* polynomials. He then showed how the univariate modular results enable one to factor $N = pq$ given half the most significant bits of p assuming p and q are of the same size. In case p and q are of different size, the result is much weaker requiring many more bits of the smaller factor to be known.

We prove Theorem 3.1 by extending the univariate modular approach. Our results are an example in which the modular approach appears to be superior to the bivariate integer approach.

Note that for simplicity we assume r and c are given to the algorithm of Theorem 3.1. Clearly this is not essential since one can try all possible values for r and c until the correct values are found.

Lattice-based factoring

We are given $N = p^r q$. Suppose that in addition, we are also given an integer P that matches p on a few of p's most significant bits. In other words, $|P - p| < X$ for some large X. For now, our objective is to find p given N, r, and P. Consider the polynomial $f(x) = (P + x)^r$. Then the point $x_0 = p - P$ satisfies $f(x_0) \equiv 0$ modulo p^r. Hence, we are looking for a root of $f(x)$ modulo p^r satisfying $|x_0| < X$. Unfortunately, the modulus p^r is unknown. Instead, only a multiple of it, N, is known.

Given a polynomial $h(x) = \sum_i a_i x^i$ we define $\|h(x)\|^2 = \sum_i |a_i^2|$. The main tool we use to find x_0 is stated in the following simple fact.

Fact 3.2 (HG98) *Let $h(x) \in \mathbb{Z}[x]$ be a polynomial of degree d. Suppose that*

a. $h(x_0) = 0 \bmod p^{rm}$ for some positive integers r, m where $|x_0| < X$, and

b. $\|h(xX)\| < p^{rm}/\sqrt{d}$.

Then $h(x_0) = 0$ holds over the integers.

Proof Observe that

$$\begin{aligned} |h(x_0)| &= \left|\sum a_i x_0^i\right| = \left|\sum a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \\ & \sum \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i\right| \leq \sqrt{d}\,\|h(xX)\| < p^{rm} \end{aligned}$$

but since $h(x_0) \equiv 0$ modulo p^{rm} we have that $h(x_0) = 0$. $\square$

Fact 3.2 suggests that we should look for a polynomial $h(x)$ that has x_0 as a root modulo p^{rm}, for which $h(xX)$ has norm less than roughly p^{rm}. Let $m > 0$ be an integer to be determined later. For $k = 0, \ldots, m$ and any $i \geq 0$ define:

$$g_{i,k}(x) := N^{m-k} x^i f^k(x).$$

Observe that x_0 is a root of $g_{i,k}(x)$ modulo p^{rm} for all i and all $k = 0, \ldots, m$. We are looking for an integer linear combination of the $g_{i,k}$ of norm less than p^{rm}. To do so we form a lattice spanned by the $g_{i,k}(xX)$ and use LLL to find a short vector in this lattice. Once we find a "short enough" vector $h(xX)$ it will follow from Fact 3.2 that x_0 is a root of $h(x)$ over $\mathbb{Z}$. Then x_0 can be found using standard root finding methods over the reals.

Let L be the lattice spanned by the coefficients vectors of:

(1) $g_{i,k}(xX)$ for $k = 0, \ldots, m-1$ and $i = 0, \ldots, r-1$, and

(2) $g_{j,m}(xX)$ for $j = 0, \ldots, d - mr - 1$.

The values of m and d will be determined later. To use Fact 2.1, we must bound the determinant of the resulting lattice. Let M be a matrix whose rows are the basis vectors for L (see Figure 1). Notice that M is a triangular matrix, so the determinant of L is just the product of the diagonal entries of M. This is given by

$$\det(M) = \left(\prod_{k=0}^{m-1} \prod_{i=0}^{r-1} N^{m-k}\right) \left(\prod_{j=0}^{d-1} X^j\right) < N^{rm(m+1)/2} X^{d^2/2}.$$

Fact 2.1 guarantees that the LLL algorithm will find a short vector u in L satisfying

$$\|u\|^d \leq 2^{d^2/2} \det(L) \leq 2^{d^2/2} N^{rm(m+1)/2} X^{d^2/2}. \tag{1}$$

This vector u is the coefficients vector of some polynomial $h(xX)$ satisfying $\|h(xX)\| = \|u\|$. Furthermore, since $h(xX)$ is an integer linear combination of the polynomials $g_{i,k}(xX)$, we may write $h(x)$ as an integer linear combination of the $g_{i,k}(x)$. Therefore $h(x_0) \equiv 0 \bmod p^{rm}$. To apply Fact 3.2 to $h(x)$ we require that

$$\|h(xX)\| < p^{rm}/\sqrt{d}.$$

The factor of $\sqrt{d}$ in the denominator has little effect on the subsequent calculations, so for simplicity it is omitted. Plugging in the bound on $\|h(xX)\|$ from equation (1) and reordering terms, we see this condition is satisfied when:

$$(2X)^{d^2/2} < p^{rmd} N^{-rm(m+1)/2}.$$

Suppose $q < p^c$ for some c. Then $N < p^{r+c}$, so we need

$$(2X)^{d^2/2} < p^{rmd - r(r+c)m(m+1)/2}.$$

Larger values of X allow us to use weaker approximations P, so we wish to find the largest X satisfying the bound. The optimal value of m is attained at $m_0 = \lfloor \frac{d}{r+c} - \frac{1}{2} \rfloor$, and we may choose d_0 so that $\frac{d_0}{r+c} - \frac{1}{2}$ is within $\frac{1}{2r+c}$ of an integer. Plugging in $m = m_0$ and $d = d_0$ and working through tedious arithmetic results in the bound:

$$X < \frac{1}{2} p^{1 - \frac{c}{r+c} - \frac{r}{d}(1+\delta)} \qquad \text{where} \qquad \delta = \frac{1}{r+c} - \frac{r+c}{4d}$$

Since $\delta < 1$ we obtain the slightly weaker, but more appealing bound:

$$X < p^{1 - \frac{c}{r+c} - 2\frac{r}{d}} \tag{2}$$

When X satisfies the bound of equation (2), the LLL algorithm will find in L a vector $h(xX)$ satisfying $\|h(xX)\| < p^{rm}/\sqrt{d}$. This short vector will give rise to the polynomial equation $h(x)$, which is an integer linear combination of the $g_{i,k}(x)$ and thus has x_0 as a root modulo p^{rm}. But since $\|h(xX)\|$ is bounded, we have by Fact 3.2 that $h(x_0) = 0$ over $\mathbb{Z}$, and normal root-finding methods can extract the desired x_0. Given x_0 the factor $p = P + x_0$ is revealed.

We summarize this result in the following lemma.

$$\begin{array}{cc}
 & \begin{array}{ccccccccc} 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 \end{array} \\
\begin{array}{c} g_{0,0}(xX) \\ g_{1,0}(xX) \\ g_{0,1}(xX) \\ g_{1,1}(xX) \\ g_{0,2}(xX) \\ g_{1,2}(xX) \\ g_{0,3}(xX) \\ g_{1,3}(xX) \\ g_{2,3}(xX) \end{array} &
\begin{bmatrix}
N^3 & & & & & & & & \\
 & XN^3 & & & & & & & \\
* & * & X^2N^2 & & & & & & \\
 & * & * & X^3N^2 & & & & & \\
* & * & * & * & X^4N & & & & \\
 & * & * & * & * & X^5N & & & \\
* & * & * & * & * & * & X^6 & & \\
 & * & * & * & * & * & * & X^7 & \\
 & & * & * & * & * & * & * & X^8
\end{bmatrix}
\end{array}$$

Example lattice for $N = p^2 q$ when $m = 3$ and $d = 9$. The "$*$" entries correspond to non-zero entries whose value we ignore. The determinant of the lattice is the product of the elements on the diagonal. Elements on the diagonal are given explicitly.

Figure 1: Example of the lattice formed by the vectors $g_{i,k}(xX)$

Lemma 3.3 *Let $N = p^r q$ be given, and assume $q < p^c$ for some c. Furthermore assume that P is an integer satisfying:*

$$|P - p| < p^{1 - \frac{c}{r+c} - 2\frac{r}{d}}$$

Then the factor p may be computed from N, r, c, and P by an algorithm whose running time is dominated by the time it takes to run LLL on a lattice of dimension d.

Note that as d tends to infinity the bound on P becomes $|P - p| < p^{1 - \frac{c}{r+c}}$. When $c = 1$ taking $d = r^2$ provides a similar bound and is sufficient for practical purposes. We can now complete the proof of the main theorem.

Proof of Theorem 3.1 Suppose $N = p^r q$ with $q < p^c$ for some c. Let $d = 2r(r + c)$. Then, by Lemma 3.3 we know that given an integer P satisfying

$$|P - p| < p^{1 - \frac{c+1}{r+c}}$$

the factorization of N can be found in time $O((\log N)^2 d^4)$. Let $\epsilon = \frac{c+1}{r+c}$. We proceed as follows:

a. For all $k = 1, \ldots, (\log N)/r$ do:

b. For all $j = 0, \ldots, 2^{\epsilon k}$ do:

c. Set $P = 2^k + j \cdot 2^{(1-\epsilon)k}$.

d. Run the algorithm of Lemma 3.3 using the approximation P.

The outer most loop on the length of p is not necessary if the size of p is known. If p is k bits long then one of the P's generated in step (c) will satisfy $|P - p| < 2^{(1-\epsilon)k}$ and hence $|P - p| < p^{1-\epsilon}$ as required. Hence, the algorithm will factor N in the required time. $\square$

4 Implementation and Experiments

We implemented the lattice factoring method (LFM) using Maple version 5.0 and Victor Shoup's NTL (Number Theory Library) package [15]. The program operates in two phases. First, it guesses the most significant bits P of the factor p, then builds the lattice described in Section 3. Using NTL's implementation of LLL, it reduces the lattice from Section 3, looking for short vectors. Second, once a short vector is found, the corresponding polynomial is passed to Maple, which computes the roots for comparison to the factorization of N.

Several observations were made in the implementation of this algorithm. First of all, it was found that the order in which the basis vectors appear in the lattice given to LLL matters. In particular, since the final polynomial is almost always of degree equal to the dimension of the lattice, this means that a linear combination which yields a short vector must include those basis vectors corresponding to the last few $g_{i,k}$, say $k = m/2, \ldots, m$ and $i = 0, \ldots, r$. It turns out to be beneficial to place them at the "top" of the lattice, where LLL would perform row reduction first, as these would likely be enough to produce a short vector alone. We found the optimal ordering for the $g_{i,k}$ to be $i = r - 1, \ldots, 0$, $k = m, m - 1, \ldots, 0$; this resulted in greatly reduced running time compared to the natural ordering, in which LLL spent a large amount of time reducing basis vectors that would never be needed.

p	N	r	bits given	lattice dimension	running time
64 bits	576 bits	8	16 bits	49	20 minutes
80 bits	1280 bits	15	20 bits	72	21 hours
96 bits	768 bits	7	22 bits	60	7 hours
96 bits	960 bits	9	22 bits	65	10 hours
100 bits	600 bits	5	23 bits	69	11 hours

Figure 2: Example running times on a 400MhZ Pentium running Windows NT.

The reader may have noticed that in building the lattice in Section 3, we could have taken powers of $(P+x)$ instead of shifts and powers of $(P+x)^r$. The reason for performing the latter is mainly for a performance improvement. Although both methods yield a lattice with the same determinant, using shifts and powers of $(P+x)^r$ produces a matrix that appears "more orthogonal". That is, certain submatrices of the matrix from Section 3 are Toeplitz matrices, and heuristically this should make it easier for LLL to find a good basis. We compared both methods and found a speedup of about ten percent by working with $(P+x)^r$.

Lastly, recall that in an LLL-reduced lattice, the shortest vector u satisfies

$$\|u\| < 2^{d/2} \det(L)^{1/d}.$$

Implementations of LLL often try to improve on this "fudge factor" of $2^{d/2}$. However, as the analysis from Section 3 shows, its effect is negligible, requiring only an extra bit of p to be known. Therefore, the higher-quality reduction produced with a smaller fudge factor is not necessary, and running times can be greatly improved by "turning off" improvements such as Block Korkin-Zolotarev reduction.

To test the algorithm, we assumed that an approximation P of the desired quality was given; we model this by "giving" the appropriate number of bits to the algorithm before constructing the lattice. In general, these bits would be exhaustively searched, so k bits given would imply a running time of 2^k times what is shown. We ran several experiments and have listed the results in Figure 2. Needless to say, the results by themselves are not so impressive; for such small N ECM performs much better. However, we expect the running time to scale polynomially with the size of the input, quickly outpacing the running times of ECM and NFS, which scale much less favorably.

5 Comparison to other factoring methods

We restate the Theorem 3.1 so that it is easier to compare lattice factoring to existing algorithms. We first introduce some notation. Let $T_\alpha(p)$ be the function defined by:

$$T_\alpha(p) = \exp\left((\log p)^\alpha\right)$$

This function is analogous to the $L_{\alpha,\beta}(p)$ function commonly used to describe the running time of factoring algorithms [6]. Recall that

$$L_{\alpha,\beta}(p) = \exp\left(\beta(\log p)^\alpha (\log\log p)^{1-\alpha}\right)$$

One can easily see that $T_\alpha(p)$ is slightly smaller than $L_{\alpha,1}(p)$. We can now state a special case of Theorem 3.1.

Corollary 5.1 *Let $N = p^r q$ be given where p and q are both k bit integers. Suppose $r = (\log p)^\epsilon$ for some ϵ. Then given N and r, a non-trivial integer factor of N can be found in time*

$$\gamma \cdot T_{1-\epsilon}(p) = \exp\left[(\log p)^{1-\epsilon}\right] \cdot \gamma$$

where γ is polynomial in $\log N$.

Asymptotic Comparison

Let p, q be k-bit primes, and suppose we are given $N = p^r q$. We study the running time of various algorithms with respect to k and r, and analyze their behaviors as r goes to infinity. We write $r = (\log p)^\epsilon$. The standard running times of several algorithms are summarized in the following table, ignoring polynomial factors. Running times are taken from [1, 5].

Method	Asymptotic Running Time
Lattice Factoring Method	$\exp\left((\log p)^{1-\epsilon}\right)$
Elliptic Curve Method	$\exp\left(1.414 \cdot (\log p)^{1/2}(\log\log p)^{1/2}\right)$
Number Field Sieve	$\exp\left(1.902 \cdot (\log N)^{1/3}(\log\log N)^{2/3}\right)$

Since $N = p^r q$ and $r = k^\epsilon$, we know that

$$\log N = r \log p + \log q \geq rk = k^{1+\epsilon}.$$

Rewriting the above running times in terms of k yields the following list of asymptotic running times.

Method	Asymptotic Running Time
Lattice Factoring Method	$\exp\left(k^{1-\epsilon}\right) = T_{1-\epsilon}(p)$
Elliptic Curve Method	$\exp\left(1.414 \cdot k^{1/2}(\log k)^{1/2}\right) > (T_{1/2}(p))^{1.414}$
Number Field Sieve	$\exp\left(1.902 \cdot k^{(1+\epsilon)/3}((1+\epsilon)\log k)^{2/3}\right) > (T_{(1+\epsilon)/3}(p))^{1.902}$

Suppose the running time of a particular algorithm is $T_\alpha(p)^{O(1)}$. We are particularly interested in the behavior of the parameter α, which governs the exponential blowup of the running time of the algorithm, with respect to the value of $r = k^\epsilon$. Notice that when $\epsilon = \frac{1}{2}$, then $\alpha = \frac{1}{2}$ for all three algorithms. The behavior of α in terms of ϵ is illustrated in Figure 3.

Practical Comparison to ECM

Of particular interest in Figure 3 is the point at $r = \sqrt{\log p}$ (i.e. $\epsilon = \frac{1}{2}$), where ECM, LFM, and NFS have similar asymptotic running times. We refer the reader to Figure 2 for the sample running times with the lattice factoring method on similar inputs.

Since some of the larger integers that we are attempting to factor exceed 1000 bits, it is unlikely that current implementations of the Number Field Sieve will perform efficiently. This leaves only the Elliptic Curve Method for a practical comparison. We reproduce a table of some example running times for factorizations performed by ECM below. ([17]; see also [13].)

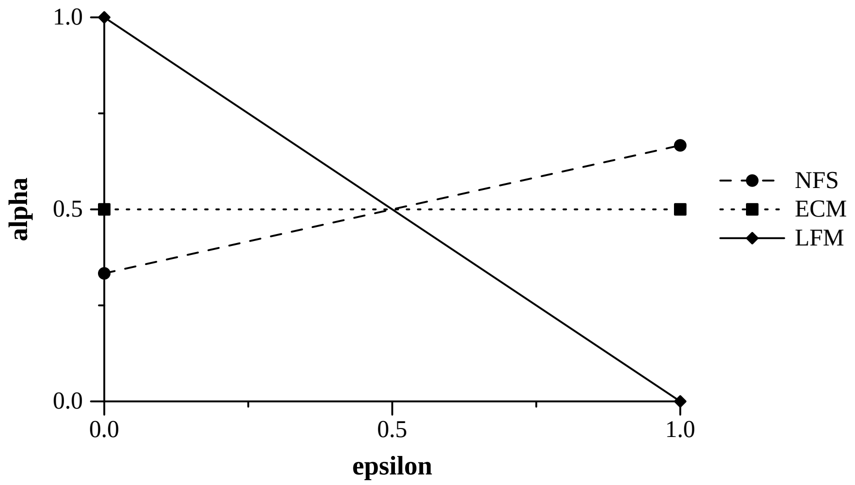

Figure 3: Asymptotic comparison of the lattice factoring method with ECM and NFS.

p	running time with $r = 1$	predicted run time for large r
64 bits	53 seconds	$r = 8$: 848 seconds
96 bits	2 hours	$r = 9$: 50 hours
128 bits	231 hours	$r = 10$: 7000 hours

Clearly, the Elliptic Curve Method easily beats the lattice factoring method for small integers. However, LFM scales polynomially while ECM scales exponentially. Based on the two tables above we conjecture that the point at which our method will be faster than ECM in practice is for $N = p^r q$ where p and q are somewhere around 400 bits and $r \approx 20$.

6 An application to integers of the form $N = p^2 q$

Throughout this section we assume that p and q are two primes of the same size. Lemma 3.3 can be used to obtain results on "factoring with a hint" for integers of the form $N = p^r q$ for *small* r. Coppersmith's results [2] show that when $N = pq$ a hint containing half the bits of p is sufficient to factor N. When $r = 1$ Lemma 3.3 reduces to the same result. However, when $r = 2$, i.e. $N = p^2 q$ the lemma shows that only a third of the bits of p are required to factor N. In other words, the hint need only be of the size of $N^{1/9}$. Hence, moduli of the form $N = p^2 q$ are more susceptible to attacks (or designs) that leak bits of p.

7 Conclusions

We showed that for cryptographic applications, integers of the form $N = p^r q$ should be used with care. In particular, we showed that the problem of factoring such N becomes easier as r get bigger. For example, when $r = \epsilon \log p$ for a fixed constant $\epsilon > 0$ the modulus N can be factored in polynomial time. Hence, if p and q are k bit primes, the modulus $N = p^k q$ can be factored by a polynomial time algorithm. Even when $r \approx \sqrt{\log p}$ such moduli can be factored in time that is asymptotically faster than the best current methods. Our results say very little about the case of small r, e.g. when

$N = p^2 q$.

Our experiments show that when the factors p and q are small (e.g. under 100 bits) the algorithm is impractical and cannot compete with the ECM. However, the algorithm scales better; we conjecture that as soon as p and q exceed 400 bits each, it performs better than ECM when r is sufficiently large.

Surprisingly, our results do not seem to follow directly from Coppersmith's results on finding small roots of bivariate polynomials over the integers. Instead, we extend an alternate technique due to Howgrave-Graham. It is instructive to compare our results to the case of unbalanced RSA where $N = pq$ is the product of two primes of different size, say p is much larger than q. Suppose p is a prime on the order of q^s. Then, the larger s is, the *more* bits of q are needed to efficiently factor N. In contrast, we showed that when $N = p^r q$, the larger r is, the *fewer* bits of p are needed.

One drawback of the lattice factoring method is that for each guess of the most significant bits of p, the LLL algorithm has to be used to reduce the resulting lattice. It is an interesting open problem to devise a method that will enable us to run LLL once and test multiple guesses for the MSBs of p. This will significantly improve the algorithm's running time. A solution will be analogous to techniques that enable one to try multiple elliptic curves at once in the ECM. Another question is to generalize the LFM to integers of the form $N = p^r q^s$ where r and s are approximately the same size.

Acknowledgments

We thank Paul Zimmermann, Peter Montgomery and Bob Silverman for supplying the running times for the ECM method.

References

[1] D. Coppersmith, "Modifications to the number field sieve", J. of Cryptology, Vol. 6, pp. 169–180, 1993.

[2] D. Coppersmith, "Small solutions to polynomial equations, and low exponent RSA vulnerabilities", J. of Cryptology, Vol. 10, pp. 233–260, 1997.

[3] A. Fujioke, T. Okamoto, S. Miyaguchi, "ESIGN: an efficient digital signature implementation for smartcards", In. proc. Eurocrypt '91, pp. 446–457, 1991.

[4] N. Howgrave-Graham, "Finding small roots of univariate modular equations revisited", Proc. of Cryptography and Coding, LNCS 1355, Springer-Verlag, 1997, pp. 131–142.

[5] A. Lenstra, H.W. Lenstra Jr., "Algorithms in Number Theory", in Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity), ch. 12, pp. 673–715, 1990.

[6] A. Lenstra, H.W. Lenstra Jr., "The development of the number field sieve", Lecture Notes in Mathematics, Vol. 1554, Springer-Verlag, 1994.

[7] A. Lenstra, H.W. Lenstra Jr., and L. Lovasz, "Factoring polynomial with rational coefficients", *Mathematiche Annalen*, 261:515–534, 1982.

[8] H.W. Lenstra Jr., "Factoring integers with elliptic curves", Annuals of Mathematics, 126:649-673, 1987.

[9] L. Lovasz, "An algorithmic theory of numbers, graphs and convexity", SIAM lecture series, Vol. 50, 1986.

[10] T. Okamoto, S. Uchiyama, "A new public key cryptosystem as secure as factoring", in Proc. Eurocrypt '98, pp. 310–318, 1998.

[11] R. Peralta, T. Okamoto, "Faster factoring of integers of special form", IEICE Trans. Fundamentals, Vol. E79-A, No. 4, pp. 489–493, 1996.

[12] J.J. Quisquater and C. Couvreur, "Fast deciperhment algorithm for RSA public-key cryptosystem", *Electronic Letters*, **18**(21), pp. 905–907, 1982.

[13] R. Silverman, Wagstaff Jr., "A Practical analysis of the elliptic curve factoring algorithm", Math. Comp. Vol 61, 1993.

[14] A. Shamir, "RSA for Paranoids", RSA Laboratories' CryptoBytes, vol. 1, no. 3, pp. 1–4, 1995.

[15] V. Shoup, Number Theory Library (NTL), `http://www.cs.wisc.edu/~shoup/ntl`.

[16] T. Takagai, "Fast RSA-type cryptosystem modulo $p^k q$", in Proc. Crypto '98, pp. 318–326, 1998.

[17] P. Zimmerman, private communications.