Factoring $N = p^2q$

Nathan Manohar Ben Fisch

Abstract

We discuss the problem of factoring $N = p^2q$ and survey some approaches. We then present a specialized factoring algorithm that runs in time $\tilde{O}(q^{0.31})$, which is comparable to the runtime $\tilde{O}(p^{1/3})$ of the factoring algorithm for integers of the form $N = p^rq$ presented in [1]. We then survey the factoring algorithm of [1] and discuss the number of advice bits needed for it to run in polynomial time. Furthermore, we discuss the possibility of constructing cryptographic primitives from the assumption that p^2q is hard to factor. We present our attempt at constructing key agreement and discuss the difficulties of building this primitive from the hardness of factoring p^2q.

1 Introduction and Problem Overview

The problem of factoring p^2q is of considerable interest in cryptography. Various encryption schemes, such as the EPOC cryptosystem [7] and the ESIGN digital signature scheme [4], are based on the assumption that this problem is hard to solve. Furthermore, Takagai [9] showed that RSA decryption can be made much faster when moduli of the form $N = p^rq$ or, in particular, $N = p^2q$ are used. A natural question to ask is whether factoring p^2q is as hard as factoring a general RSA modulus $N = pq$. In particular, is there a way to exploit the fact that N is of the form p^2q that could lead to an improvement in factoring moduli of this form?

One observation is that given $N = p^2q$, it is easy to learn if q is a quadratic residue modulo some prime ℓ. To see this, we first note that the Jacobi symbol $\left(\frac{\ell}{N}\right)$ is computable in polynomial time via repeated application of the law of quadratic reciprocity. Additionally, since the Jacobi symbol is a multiplicative function, it follows that

$$\left(\frac{\ell}{N}\right) = \left(\frac{\ell}{p^2q}\right) = \left(\frac{\ell}{p^2}\right)\left(\frac{\ell}{q}\right) = \left(\frac{\ell}{p}\right)^2\left(\frac{\ell}{q}\right).$$

Since $\gcd(\ell, p) = 1$ (for $\ell \neq p$), it follows that the Legendre symbol $\left(\frac{\ell}{p}\right) = \pm 1$ and therefore

$$\left(\frac{\ell}{N}\right) = \left(\frac{\ell}{q}\right).$$

By the law of quadratic reciprocity, it follows that

$$\left(\frac{\ell}{q}\right)\left(\frac{q}{\ell}\right) = (-1)^{\frac{\ell-1}{2}\frac{q-1}{2}},$$

and so $\left(\frac{q}{\ell}\right)$ can be computed in polynomial time.

One approach to factoring p^2q is to attempt to leverage the information about q's quadratic residuosity to recover q. In Section 3, we present a $\tilde{O}(q^{0.31})$ time factoring algorithm for p^2q that follows this approach. Furthermore, we discuss how additional constraints (the quadratic residuosity of q modulo additional primes ℓ_i) reduces the number of possible solutions for q in some range.

In Section 4, we survey a specialized factoring algorithm for numbers of the form p^rq due to Boneh, Durfee, and Howgrave-Graham [1]. They show that when given a good enough approximation P to p, it is possible to recover p in polynomial time by constructing a polynomial f with small coefficients that has $x = p - P$ as a root. If the coefficients of the polynomial are sufficiently small and the root x is also sufficiently small, they show that there will be no wrap-around and x will be a root of f over the integers, which can be recovered in polynomial time. The required approximation P to p is found via brute force and this process determines the running time of the algorithm since the rest of it runs in polynomial time. If the required approximation P to p can be found in polynomial time, then this algorithm runs in polynomial time, which [1] shows to be the case when $r = \Omega(\log p)$. For factoring p^2q, this algorithm has a running time of $\tilde{O}(p^{1/3})$, which is slightly worse than the time complexity of our factoring algorithm (assuming the factors p and q are of roughly the same size).

Additionally, this algorithm requires $\frac{\log p}{3} + O(1)$ advice bits to factor p^2q in polynomial time [1].

These two algorithms provide interesting approaches to factoring p^2q and could potentially be improved to reduce their running times. However, these two algorithms are currently eclipsed asymptotically by general purpose factoring algorithms such as the Elliptic Curve Method [6] and the Number Field Sieve [8].

Since factoring p^2q appears to be quite difficult, we also approach this problem in the other direction where we assume it is hard to solve and try to construct cryptographic primitives from this assumption. In Section 5, we show our attempt at constructing key agreement from this assumption and discuss the difficulties of making such a construction work.

We conclude by discussing the variety of open problems related to factoring p^2q and describe directions for future work.

2 One Approach

One approach to factoring p^2q is to attempt to use the information about q's quadratic residuosity modulo many primes ℓ_i to construct a polynomial that must have q as a small root and then determine this root using Coppersmith's method [2].

As discussed previously, it is possible given $N = p^2q$ and some prime ℓ to compute $\left(\frac{q}{\ell}\right)$ in polynomial time. Using this information, we can construct tables T_{ℓ_i} for primes ℓ_i that list the possible values of q mod ℓ_i. Since there are $\frac{\ell_i-1}{2}$ quadratic residues and nonresidues modulo ℓ_i, the size of T_{ℓ_i} will be $\frac{\ell_i-1}{2}$. Equivalently, let $b_i = \left(\frac{q}{\ell_i}\right)$. Then, the table T_{ℓ_i} contains the roots of the polynomial $(x^{(\ell_i-1)/2} - b_i)$ over $\mathbb{Z}_{\ell_i}$. The question of factoring p^2q has now been reduced to the question of whether q can be efficiently reconstructed given information about whether or not q is a quadratic residue modulo primes for a fixed sequence of primes.

Observe that for each prime ℓ_i for which we have a corresponding table T_{ℓ_i}, we can construct the polynomial

$$f_{\ell_i}(x) = \prod_{a \in T_{\ell_i}} (x-a) \bmod \ell_i.$$

Then, using the Chinese remainder theorem, we can construct the polynomial f mod $\prod_i \ell_i$ obtained by applying to the Chinese remainder theorem term-wise to the coefficients of each of the f_{ℓ_i}'s. Note that since q is a root of all the f_{ℓ_i}'s, it follows that q is a root of f. However, q is not a sufficiently small root of f for Coppersmith's algorithm to determine it. In particular, if f is a polynomial of degree d modulo L, Coppersmith's algorithm will only find roots that are $< L^{1/d}$. However, if the ℓ_i's are the first n primes, then $\prod_i \ell_i \approx e^n$. Since we must have $\prod_i \ell_i > q$, it follows that we need $n > \ln q$. Additionally, we note that the degree of f will be the size of the largest table, which will be $\frac{\ell_n-1}{2}$. Since the size of the largest prime is $\approx n \ln n$, it follows that $L^{1/d}$ will be approximately

$$e^{2n/n\ln n} = e^{2/\ln n} = e^{O(1/\log\log q)} << q,$$

and so Coppersmith's method will not be able to determine q.

An immediate observation is that if the degree of f was smaller (was $n^{1-\varepsilon}$ instead of $\approx n\ln n$), then Coppersmith's method would be sufficient to determine q. In this case, we would have that $L^{1/d}$ is $\approx e^{n^\varepsilon} > q$ for $n = O(\log^{1/\varepsilon} q)$. However, there does not seem to be any way to reduce the degree using these tables alone. In particular, since the size of the tables is $O(n\ln n)$, f must necessarily have degree $O(n\ln n)$ in order to capture all the possible values of q. If some extra information about q unrelated to the quadratic residuosity of q modulo primes could be gathered, this could potentially be leveraged to reduce the size of the tables.

3 Attacks Leveraging Quadratic Reciprocity

Given that Coppersmith's method will not work for finding sufficiently many roots (i.e., more than constant size roots) of a polynomial of degree $O(\log(M))$ over M, we need to search for other approaches of leveraging the information gained from the quadratic reciprocity of q mod small primes in order to factor p^2q. Define $K = p_1 \cdots p_k$ such that $q < K$. Define $M = p_1 \cdots p_m$ for $m > k$. As before, let F be the unique polynomial mod M that is equivalent to $f_i = (x^{(p_i-1)/2} - b_i)$ over $\mathbb{Z}_{p_i}$ where $b_i = \left(\frac{q}{p_i}\right)$. Let T_i be the table of roots of f_i. The first question is whether we can even hope that there will be a small (i.e. polynomial) number of roots of F that are less than K, otherwise there would probably be no hope of recovering q. When $n = k$ it is easy to see that there are an exponential number of roots in $[0,K)$ because any k-wise combination of the roots of f_i for each i corresponds to a unique integer in $\mathbb{Z}_k$ that is a root of F. This total number of roots is precisely $\Pi_{i=1}^k (p_i-1)/2 \approx K/2^k$. Although the number of solutions for q remaining is still exponential, the information about q that we got from examining the roots of $f_1, \ldots, f_k$ reduced the solution space by a factor 2^k. Heuristically, we would hope that each polynomial f_i for $i > k$ would continue to reduce the solution space by a factor 2 so that we would only need $n = O(k) = O(log(q))$ polynomials to reduce the number of solutions to a constant.

3.1 Analyzing number of solutions.

Despite the intuition outlined above, proving that the number of solutions is reduced to a constant given the constraints imposed by sufficiently many reduced polynomials seems tricky. As a first step we would want to show that the number of integers in $[0,K)$ that are in T_1,...,T_{k+1} is a constant fraction smaller than the number of integers in $[0,K)$ that are in $T_1,...,T_k$. However, this is not in general true for arbitrary sets T_i of size $p_i/2$. For example, consider $K = p_1$, i.e. k = 1, let T_1 contain $[0,p_1/2)$ and let T_2 contain $[0,p_2/2)$. Then T_2 doesn't eliminate any solutions that just satisfy T_1. The number of integers in $[0,K)$ that are in both T_1 and T_2 is precisely $p_1/2$.

Assuming even distribution of roots. Suppose that the roots of F are evenly distributed over $\mathbb{Z}_M$. Let $n = poly(k)$, so $n >> k$. The total number of roots of F is $M/2^m$. Assuming an even distribution of roots, the fraction of these roots that fall in the range $[0,K)$ would be approximately $(M/2^m)\cdot(K/N) = K/2^m \approx e^k/2^m = e^{k-log(e)m} = e^{-poly(k)}$, which is negligible for sufficiently large k. Therefore, if $q \in [0,K)$ is a root of F (by construction), it is likely to be the only root in this range. More precisely, if we assume that for a *randomly* chosen modulus of the form $N = p^2q$ the roots of F are *randomly* distributed then in expectation q is the unique root of F in the range $[0,K)$, and the probability that there are a non-polynomial (or even non-constant) number of additional roots in this range is negligible by Chernoff bounds.

3.2 An $\tilde{O}(q^{0.31})$ attack

Note that if we set $K \approx q$ as described above we already obtain an attack on factoring $N = p^2q$ in time $\tilde{O}(q^{0.31})$, which is comparable to the $\tilde{O}(q^{1/3})$ attack of [1]. We know that q can be efficiently derived using CRT from *some* k-wise combination of the roots of the polynomials $f_1,...,f_k$. The total number of combinations is $K/2^k \approx e^k/2^k = e^{k(1-ln(2))} = K^{1-ln(2)} \approx q^{0.30685}$. Thus we can try all combinations in time $\tilde{O}(q^{0.31})$.

4 Factoring $N = p^rq$

Due to prevalence of moduli of the form $N = p^rq$ in cryptography, researchers have studied the difficulty of factoring moduli of this special form. In [1], Boneh, Durfee, and Howgrave-Graham present a method for factoring integers of the form $N = p^rq$ that utilizes techniques introduced by Coppersmith in [3]. However, this method only runs in polynomial time (in $\log N$) when $r = \Omega(\log p)$. When $r = 2$, this method runs in time $\tilde{O}(p^{1/3})$, which is slower than the Elliptic Curve Method [6] or the Number Field Sieve [8]. When $r \approx \sqrt{\log p}$, this lattice-based factoring method begins to be asymptotically faster than the Elliptic Curve Method and the Number Field Sieve. This method is particularly interesting because it runs in polynomial time if it is provided with a good enough approximation to p. For the case of $N = p^2q$, this lattice-based method will factor N in polynomial time given the most significant third of the bits of p. Since this method has a similar asymptotic running time to our factoring algorithm, we present a sketch of the algorithm here for completeness.

For simplicity, we will only describe the algorithm for the case $r = 2$, but the process described below easily extends to larger r. Additionally, we will try to follow the notation used in [1]. At a high level, the algorithm is based on the following idea. Suppose that some rough approximation P to p is known. In particular, suppose $|P - p| < X$ for some X to be determined. Then, if one considers the polynomial

$$f(x) = (P+x)^2,$$

it follows that the point $x_0 = p - P$ is a root of $f(x)$ mod p^2. Furthermore, since P was a good approximation to p, it follows that $|x_0| < X$. So, the problem has been reduced to finding a small root of $f(x)$ mod p^2. It is important to point out here that the modulus in question, p^2 is unknown. However, knowing the modulus becomes unnecessary given the following fact due to Howgrave-Graham: Suppose that $f(x)$ is a degree d polynomial over the integers. Suppose further that $f(x_0) \equiv 0 \bmod p^s$ for some small root $|x_0| < X$ and that $||f(xX)|| < p^s/\sqrt{d}$, where $||f||$ is defined as the ℓ_2 norm of its coefficients vector. Then, $f(x_0) = 0$ in $\mathbb{Z}$.

The above essentially states that if we can construct a polynomial with sufficiently small coefficients, then x_0 will be a root over the integers, which can be found in polynomial time. We note that knowing the modulus p^2 or p^s is unnecessary. All that we need to be able to do is to construct a polynomial that has small coefficients when evaluated on xX while maintaining the fact that the polynomial has a root at x_0 modulo p^s.

To accomplish this, [1] sets $s = 2m$ for some parameter m to be determined and takes a series of polynomials $g_{i,k}(x)$ that are constructed so that $x_0 = p - P$ is a root modulo p^{2m}. They then construct a lattice $\mathscr{L}$ whose basis is the set of coefficients vectors of the $g_{i,k}(xX)$'s. Using the LLL algorithm [5], it is then possible to find a short vector $\mathbf{v} \in \mathscr{L}$. This short vector $\mathbf{v}$ can be thought of as the coefficients vector of a new polynomial $h(xX)$. However, since $\mathbf{v} \in \mathscr{L}$, $h(x)$ can be expressed as a linear combination of the $g_{i,k}$'s and since $g_{i,k}(x_0) \equiv 0 \bmod p^{2m}$ for all $g_{i,k}$, it follows that $h(x_0) \equiv 0 \bmod p^{2m}$. Therefore as stated previously (provided that $||h(xX)||$ is sufficiently small), x_0 is a root of h over the integers and can

therefore be found in polynomial time. Since $x_0 = p - P$ and P is public, it is then possible to recover p and factor $N = p^2 q$.

More specifically, the polynomials $g_{i,k}$ are defined as

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

where $f^k(x) = (P+x)^{2k}$. We note immediately that

$$\begin{aligned} g_{i,k}(x_0) &= (p^2 q)^{m-k} x^i (P + p - P)^{2k} \\ &= p^{2m} q^{m-k} x^i \\ &\equiv 0 \bmod p^{2m} \end{aligned}$$

and so x_0 is a root of $g_{i,k} \bmod p^{2m}$ for all i,k.

The LLL algorithm guarantees that when given a full rank lattice $\mathscr{L}$ of dimension d as input, it will output a lattice vector $\mathbf{v}$ such that

$$||\mathbf{v}|| \leq 2^{d/2} \det(L)^{1/d}$$

where L is the matrix whose column vectors form a basis of $\mathscr{L}$. The lattice $\mathscr{L}$ is instantiated to be the lattice whose basis vectors are the coefficients vectors of the polynomials $g_{i,k}(xX)$ for $i = 0,1$ and $k = 0,\ldots,m-1$ and the coefficient vectors of the polynomials $g_{j,m}(xX)$ for $j = 0,\ldots,d-2m-1$. $\mathscr{L}$ is a d-dimensional lattice and the matrix L corresponding to this lattice is triangular, so its determinant can be easily computed as the product of the diagonal entries. After some calculation, [1] determines that

$$\det(L) < N^{m(m+1)} X^{d^2/2}$$

and therefore, the LLL algorithm will output a vector $\mathbf{v}$ satisfying

$$||\mathbf{v}|| \leq 2^{d/2} N^{m(m+1)/d} X^{d/2}.$$

Viewing $\mathbf{v}$ as the coefficients vector of some polynomial $h(xX)$, we see that the algorithm can determine p provided

$$||h(xX)|| < \frac{p^{2m}}{\sqrt{d}}.$$

It remains to determine optimal values for the parameters X, m, d. It can be shown that the optimal value for m is

$$m = \lfloor \frac{d}{3} - \frac{1}{2} \rfloor.$$

The algorithm will then succeed provided that

$$X < p^{2/3 - 4/d}.$$

Since the LLL algorithm runs in time polynomial in the dimension d of the lattice [5], we can choose $d = O(\log p)$ and still have one iteration of the factoring algorithm run in polynomial time.

All that remains is to determine P. This can be done by iterating through all possible values of the most significant bits of p, where the number of bits we need to guess is determined by the precision of the approximation required. For factoring $N = p^2 q$, we need to correctly guess one third of the bits of p (plus an additional constant number of bits which can be brute forced at each step without increasing the asymptotic running time). There are $p^{1/3}$ possibilities for the most significant third of the bits of p, and so it follows that this algorithm factors $p^2 q$ in time $\tilde{O}(p^{1/3})$.

A interesting property of this algorithm is that it can be easily made to run in polynomial time given a sufficient number of advice bits. In particular, given a close enough approximation P to p, the algorithm can determine p in polynomial time. For factoring $p^2 q$, the most significant third of the bits of p or $\frac{\log p}{3}$ advice bits are needed for the algorithm to run in polynomial time.

5 Turning the Problem Around

Suppose that it is indeed (classically) hard to factor $p^2 q$. Does that imply the hardness of some problem related to finding small roots of polynomials of poly degree over a super smooth modulus N? Can we build some useful crypto primitives from this new problem? The hope is that this problem would be classically hard by reduction to factoring $p^2 q$, and also might be quantum hard. While there are known efficient quantum algorithms for factoring there might not be efficient quantum algorithms for finding roots of polynomials over composite moduli.

Candidate problem. Given $M = p_1 \cdots p_m$ where p_i is the ith prime, a bound $B = M^{1-\varepsilon}$ for some $\varepsilon < 1$, and a polynomial P of degree $O(log(M)$ over $\mathbb{Z}_M$, output a root of P in the range $[0,B)$ or $\perp$ if there is no root.

The hardness of this problem (for certain worst case choices of P, ε) is implied by the hardness of factoring $N = p^2 q$ for randomly chosen λ-bit p, q under the *evenly distributed roots* assumption described above. The reduction is as follows. Given a random instance of $N = p^2 q$ where $q < N^{1/3}$, let $k \approx ln(N^{1/3})$ and $m = ck$ for $c = O(poly(\lambda)$. Form the polynomial F as above, i.e. the unique polynomial mod M that is equivalent to $f_i = (x^{(p_i-1)/2} - b_i)$ over $\mathbb{Z}_{p_i}$ where $b_i = \left(\frac{q}{p_i}\right)$ for $M = p_1 \cdots p_m$. Let $B = e^k$, and note that $B \approx e^{m-k} = e^{m(1-1/c)} = M^{1-1/c} = M^{1-\varepsilon}$ for $\varepsilon = 1/poly(\lambda)$. F is a polynomial of degree $(m - 1/2) = O(log(M))$ over $\mathbb{Z}_M$. Thus we have constructed an instance of the candidate problem, from which we obtain a root x of F in the range $[0,B)$. We know that q is a root of F, and by the assumption on the distribution of roots and a simple application

of Markov's inequality, the probability that there is more than 1 root in $[0,B)$ is less than $e^{-poly(\lambda)}$, i.e. negligible in λ. Thus, $x = q$ with overwhelming probability.

Building key exchange? We were not able to construct key exchange from the candidate problem, but we sketch here an idea. The construction needs to leverage the hardness of finding *small* roots of a polynomial P over the exponential super-smooth modulus M even though it is obviously easy to find some (possibly large) root of P by finding a root of P mod p_i for each i and using CRT. Any construction where the shared secret can be derived from any root of a polynomial P will not work, as anyone would be able to obtain the shared secret.

Suppose there are two parties Alice and Bob. Alice can construct a polynomial P of degree $O(log(M)$ over $\mathbb{Z}_M$ with a small root $\alpha < B$ that is her secret, and Bob can construct a polynomial Q also of degree $O(log(M))$ over $\mathbb{Z}_M$ with a small root $\beta < B$ that is his secret. Now Bob and Alice both know small (yet distinct) roots of the polynomial $P \cdot Q$, which is also of degree $O(log(M))$ over $\mathbb{Z}_M$, and we assume it is hard for anyone else to compute a small root of $P \cdot Q$. Is there a way for Bob and Alice to derive a shared secret that can be computed just from knowledge of any *small* root of $P \cdot Q$? It is important that the derivation would fail with a large root.

A broken idea. Here is another simple idea that doesn't work, but may inspire more ideas. The secret to be agreed upon will lie in $[0,B)$, which is still exponentially large but *small* with respect to the modulus M. Alice selects P with small root $\alpha < B$ as above and sends P to Bob. Bob chooses a secret $s \in [0,B)$, samples a random masking polynomial R, samples a random masking value $D \in [0,B)$, and sends the polynomial $H(x) = P(x) \cdot Q(x) + Dx + sB^2$ back to Alice. Using α, Alice can compute $H(\alpha) = (D\alpha) + sB^2$. Since $D\alpha/B^2 < 1$ over the integers, Alice divides $(D\alpha) + sB^2/B^2 = s + D\alpha/B^2$ (over the rationals, not $\mathbb{Z}_M$!) and rounds down to the nearest integer to recover s

An adversary that has a large root $\alpha^* > B^2$ wouldn't be able to recover s naively because $H(\alpha^*)/B^2 = D(\alpha^*/B^2) + s > D + s$ for unknown $D \in [0,B)$, which hides s. However, there is an attack. Since P is public the attacker can reduce P mod p_i for each i. Since $\mathbb{Z}_{p_i}$ is a field for each p_i, by the division theorem for polynomials over an integral domain there is a unique Q,R in $\mathbb{Z}_{p_i}[x]$ such that $H = PQ + R$ with $deg(R) < P$, hence the attacker recovers $R = Dx + sB^2 \ mod \ p_i$ for each i, and can thus learn sB^2 mod p_i for each i and recover sB^2 mod M using CRT.

Connection to Short Integer Solution? SIS (*short integer solution*) is a well known lattice problem that is conjectured to be hard, and reduces to LWE. Given a lattice basis B, the SIS problem is to find a vector x of low norm such that $Bx = 0$. Roughly, SIS reduces to LWE because if it easy to find a small x such that $Bx = 0$ then one can distinguish $Bs + e$ from random r by computing $(Bs + e)x = e^T x$, which will have low norm as opposed to $r^T x$. We do not know of a key exchange constructed directly from SIS, but constructing key exchange from SIS and our candidate problem seem similar in nature.

6 Conclusions and Open Problems

We described our attempts at factoring p^2q and building cryptographic primitives from the hardness of factoring p^2q. However, this problem proved to be extremely difficult, and as a result, numerous open problems remain. The most obvious unresolved question is can p^2q be factored classically in polynomial time? However, showing this is extremely ambitious and it would be interesting to even construct a specialized factoring algorithm for $N = p^2q$ that performs better asymptotically than ECM or the Number Field Sieve. We attempted to use the information extracted from N about the quadratic residuosity of q modulo primes ℓ_i to speed up the running time of the Quadratic Sieve or the Number Field Sieve, but were unable to utilize this additional information in a meaningful manner. However, it seems like this information should be able to be leveraged in some way.

Additionally, it would be interesting to study the number of advice bits needed to factor p^2q in polynomial time and see if it is possible to do better than the $\frac{\log p}{3} + O(1)$ bits needed in [1]. Another direction would be to try to say something about how additional constraints reduce the possible number of solutions for q in a given range. Alternatively, one could try to prove something about the distribution of the roots of a polynomial f that has been derived from a modulus $N = p^2q$ in the manner described previously.

On the other hand, one could try to build cryptographic primitives assuming the hardness of factoring p^2q. In particular, is it possible to build key exchange from the candidate problem described above, namely that it is hard to find a small root of a polynomial over a smooth modulus. We can reduce this problem to the question of building key exchange between two parties, each of which knows some small root to a public polynomial. It seems like this should be enough to build key exchange, but as shown previously, our attempted construction failed because the smooth modulus is easily factored and the secret could be recovered by determining its value modulo the prime factors of the smooth modulus and then applying CRT.

References

[1] BONEH, D., DURFEE, G., AND HOWGRAVE-GRAHAM, N. *Factoring N = p r q for Large r*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 326–337.

[2] COPPERSMITH, D. Finding a small root of a univariate modular equation. In *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques* (Berlin, Heidelberg, 1996), EUROCRYPT'96, Springer-Verlag, pp. 155–165.

[3] COPPERSMITH, D. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology 10*, 4 (1997), 233–260.

[4] FUJIOKA, A., OKAMOTO, T., AND MIYAGUCHI, S. *ESIGN: An Efficient Digital Signature Implementation for Smart Cards*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 446–457.

[5] LENSTRA, A. K., LENSTRA, H. W., AND LOVÁSZ, L. Factoring polynomials with rational coefficients. *Mathematische Annalen 261*, 4 (1982), 515–534.

[6] LENSTRA, H. W. Factoring integers with elliptic curves. *Annals of Mathematics 126*, 3 (1987), 649–673.

[7] OKAMOTO, T., UCHIYAMA, S., AND FUJISAKI, E. Epoc: Efficient probabilistic public-key encryption (submission to p1363a).

[8] POMERANCE, C. A tale of two sieves. *Notices Amer. Math. Soc 43* (1996), 1473–1485.

[9] TAKAGI, T. Fast rsa-type cryptosystem modulo pkq. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, UK, 1998), CRYPTO '98, Springer-Verlag, pp. 318–326.