Journal of Mathematical Sciences: Advances and Applications
Volume 38, 2016, Pages 1-48
Available at http://scientificadvances.co.in
DOI: http://dx.doi.org/10.18642/jmsaa_7100121630

A STUDY OF SOME METHODS FOR FINDING SMALL ZEROS OF POLYNOMIAL CONGRUENCES APPLIED TO RSA

ALI H. HAKAMI and MOHAMMED H. HAKAMI

Department of Mathematics
Faculty of Science
Jazan University
P. O. Box 277, Jazan
Postal Code: 45142
Saudi Arabia
e-mail: aalhakami@jazanu.edu.sa

Department of Computer Science
Information Security
Faculty of Science and Engineering
Queensland University of Technology
Gardens Point GP, P Block Level 8801
Australia

Abstract

In this paper, we shall follow Håstad [28], Coppersmith [16, 17], and others to describe methods for finding small zeros of polynomial congruences. As an application, we study the security of public key cryptosystems. In particular, we study the RSA public key cryptosystem by making use of these methods.

2010 Mathematics Subject Classification: 11C08, 11D79.
Keywords and phrases: small solutions, small zeros, polynomial, congruences, applications of lattices, RSA.
Received February 4, 2016

1. Introduction and Notations

The purpose of the present note is to give a method for finding small roots to polynomial congruence in one or two variables. We shall follow technique of Coppersmith [16, 17] as a starting point for many of our result. We will introduce the general Coppersmith approach and provide a few simple examples. We use a simplified version due to Howgrave-Graham [30, 31, 32].

In our work, we shall use standard notion and write $\mathbb{Z}$ to denote the ring of integers, $\mathbb{Q}$ for the field of rationals, and $\mathbb{R}$ for the field of real numbers.

We denote by $\mathbb{Z}^n$ (resp., $\mathbb{Q}^n$ and $\mathbb{R}^n$) the vector space with elements that are n-tuples of elements of $\mathbb{Z}$ (resp., $\mathbb{Q}$ and $\mathbb{R}$). We endow these vector spaces with the Euclidean norm: if $a \in \mathbb{R}^n$, then $\|a\|^2 = \sum_i a_i^2$.

We also work with the rings of unvaried polynomials $\mathbb{Z}[x]$, $\mathbb{Q}[x]$, and $\mathbb{R}[x]$, as well as multivariate polynomials in $\mathbb{Z}[x, y]$, $\mathbb{Q}[x, y]$, and $\mathbb{R}[x, y]$.

Given a polynomial $g(x) = \sum_i a_i x^i$, we define the *norm* of $g(x)$ by $\|g(x)\|^2 = \sum_i a_i^2$. Given nonnegative $X \in \mathbb{R}$, we define the *weighted norm* of $g(x)$ by $\|g(xX)\|^2 = \sum_i (a_i X^i)^2$. Similarly, given a polynomial $h(x, y) = \sum_{i,j} b_{i,j} x^i y^j$, we define the norm of $h(x, y)$ by $\|g(x, y)\|^2 = \sum_{i,j} b_{i,j}^2$. Given nonnegative $X, Y \in \mathbb{R}$, we define the weighted norm of $h(x, y)$ by $\|h(xX, yY)\|^2 = \sum_i (a_i X^i)^2$.

A univariate polynomial is said to be *monic* when the coefficient of the leading term is equal to 1. Of primary importance in this work is the notion of a *lattice* (concept from the geometry of numbers), defined in the next section.

Let $u_1, u_2, \ldots, u_{\mathrm{w}}$ be linearly independent vectors in an normal n-dimensional vector space. The lattice L spanned by $(u_1, \ldots, u_2)$ is the set of all integer linear combinations of $u_1, \ldots, u_w$. We call w the *dimension* or *rank* of the lattice L, and that the lattice is full rank when $w = n$. We alternatively say that L is generated by $(u_1, \ldots, u_2)$ and that $(u_1, \ldots, u_2)$ *forms a basis of L*.

We work mostly with lattice in two vector spaces: the set $\mathbb{R}^n$ endowed with the Euclidean norm; and, the set of polynomials in $\mathbb{R}[x]$ of degree at most $n-1$, endowed with the norm given by $\left\|\sum_{i=0}^{n-1} a_i x^i\right\|^2 = \sum_{i=0}^{n-1} a_i^2$.

There is a natural isomorphism between these two vector spaces given by identifying a polynomial with its coefficients vector, and throughout this work switch freely between representations as convenient.

We denote by $(u_1^*, \ldots, u_w^*)$ the vectors obtained by applying the Gram-Schmidt orthogonalization process to the vectors $u_1, \ldots, u_w$. We define the *determinant* of the lattice L as $\det(L) := \prod_{i=1}^{w} \left\|u_i^*\right\|$. For example, if full rank lattice with basis in $\mathbb{R}^n$, then the determinant of L is equal to the determinant of the $w \times w$ matrix whose rows are the basis vectors $u_1, \ldots, u_w$.

We note that every nontrivial lattice in $\mathbb{R}$ has infinitely many bases. This gives rise to the notion of the *quality* of a basis for a lattice. The notion of quality applied to a lattice depends on the application, but usually includes some measure of the lengths or orthogonality of the vectors in a basis.

The goal of *lattice reduction* is to find good lattice bases. The theory of lattice reduction goes back to the work of Lagrange [36]; Gauss [24]; Hermite [29]; and Korkine and Zolotareff [35], who were interested in the reduction of quadratic forms. Lattice theory was given its geometric foundation in Minkowski's famous work Geometrie der Zahlen (The Geometry of Numbers) [40] in the late nineteenth century.

Minkowski (see Niven et al. [43]), proved that there is always a basis $u_1, \ldots, u_w$ for a lattice L satisfying $\prod \|u_i\| \leq \gamma_w \cdot \det(L)$ for some γ_w that depend only on w (and not on the entries of u_i). However, his proof is nonconstructive, and it would be almost a century before the first polynomial-time algorithm to compute reduced bases of lattices of arbitrary dimension was discovered. This is the celebrated Lenstra-Lenstra-lovaász lattice basis reduction algorithm. This algorithm is of primary importance to the results in this work, and is summarized in Lemma 2.1.

2. Lattices and it Representations

In order to discuss the running times of algorithms operating on lattices, we must describe the representation of lattices given as input to these algorithms. Suppose $u_1, \ldots, u_w$ are vectors in $\mathbb{Z}^n$. The running time of an algorithm with input $(u_1, \ldots, u_w)$ is parameterized by w, n, and by the largest element of the u_i, defined by $u_{largest} := \max_{i,j} u_{ij}$. For lattices in $\mathbb{Q}^n$, the situation is slightly more complex. Suppose we are given the vectors $u_1, \ldots, u_w$ in $\mathbb{Q}^n$. There is some least integer D such that $Du_1, \ldots, Du_w$ are all in $\mathbb{Z}^n$. Then we define the largest element by $u_{largest} := \max_{i,j}(Du_{ij})$.

We note that all lattices in $\mathbb{R}^n$ used in this work are in fact lattices in $\mathbb{Q}^n$ since they are represented using rational approximations.

Lemma 2.1 (The *LLL* algorithm)**.** *Let L be a lattice spanned by* $(u_1, \ldots, u_w)$. *The LLL algorithm, given* $(u_1, \ldots, u_w)$, *produces a new basis* $(b_1, \ldots, b_w)$ *of L satisfying*:

(1) $\left\|b_i^*\right\|^2 \le 2\left\|b_{i+1}^*\right\|^2$ *for all* $1 \le i \le w$.

(2) *For all i, if* $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_j b_j^*$, *then* $|\mu_j| \le \frac{1}{2}$ *for all* j.

The algorithm performs $O(w^4 l)$ *arithmetic operations, where* $l = \log u_{largest}$.

We note that an *LLL*-reduced basis satisfies some stronger properties, but those are not relevant to our discussion.

We denote by $T_{LLL}(w, n)$ the running time of the *LLL* algorithm on a basis $(u_1, \ldots, u_w)$ satisfying $\log_2 u_{largest} \le n$. When $(b_1, \ldots, b_w)$ is the output of the *LLL* algorithm on a basis for a lattice *L*, we say that it is an *LLL*-reduced basis. We will make heavy use of the following fundamental fact about the first vector in an *LLL*-reduced basis.

Lemma 2.2. *Let L be a lattice and* $b_1, \ldots, b_w$ *be an LLL-reduced basis of L. Then*

$$\|b_1\| \le 2^{(w-1)/4} \det(L)^{\frac{1}{w}}.$$

Proof. Since $b_1 = b_1^*$ the bound immediately follows from:

$$\det(L) = \prod_i \left\|b_i^*\right\| \ge \|b_1\|^w \cdot 2^{-\frac{w(w-1)}{4}}. \qquad \square$$

In sequence sections, we may also need to bound the length of the second vector in an *LLL*-basis. This is provided in the following lemma:

Lemma 2.3. *Let L be a lattice and* $b_1, \ldots, b_w$ *be an LLL-reduced basis of L. Then*

$$\|b_1\| \leq 2^{\frac{w}{4}} \left[\frac{\det(L)}{\|b_1\|}\right]^{\frac{1}{w-1}}.$$

Proof. Observe

$$\det(L) = \prod_i \left\|b_i^*\right\| \geq \|b_1\| \cdot \left\|b_2^*\right\|^{w-1} \cdot 2^{\frac{-(w-1)(w-2)}{4}},$$

giving us

$$\left\|b_2^*\right\| \leq 2^{\frac{w-2}{4}} \left[\frac{\det(L)}{\|b_1\|}\right]^{\frac{1}{w-1}}.$$

Then the bound follows from

$$\|b_2\| \leq \left\|b_2^*\right\| + \frac{1}{2}\|b_1\| \leq (1 + \frac{1}{2\sqrt{2}})\left\|b_2^*\right\| \leq \sqrt{2}\left\|b_2^*\right\|. \qquad \square$$

3. Finding Small Zeros to Univariate Polynomial Congruences

Much of the work described in this work was inspired by the seminal work of Coppersmith for finding small solutions to polynomial congruences [16]. We use this very effective technique as a starting point for many of our results. In subsequent sections, we will apply and extend this technique to solve a number of cryptanalytic problems, and discuss subtleties in its implementation and use. In this section, we will introduce the general Coppersmith approach and provide a few simple examples. We use a simplified version due to Howgrave-Graham [30, 31, 32].

Suppose we are giving a polynomial $f(x)$ and a real number M, and we wish to find a value $x_0 \in \mathbb{Z}$ for which $f(x_0) \equiv 0 (\text{mod } M)$. The main tool we use is stated in the following simple fact. This has been attributed to many authors. Its first use we are aware of is in the work of Håstad [28]; it was later used was used implicitly by Coppersmith [15, 16], and stated in nearly the following form by Howgrave-Graham [32].

Recall that if $h(x) = \sum a_i x^i$, then $\|h(xX)\|^2 = \sum_i (X^i a_i)^2$.

Theorem 3.1. *Let* $h(x) \in \mathbb{R}$ *be a polynomial of degree* w, *and let* $X \in \mathbb{R}$ *be given. Suppose there is some* $|x_0| < X$ *such that*

(1) $h(x_0) \in \mathbb{Z}$, *and for all* $1 \leq i \leq w$.

(2) $\|h(xX)\| < \dfrac{1}{\sqrt{w}}$.

Then $h(x_0) = 0$.

Proof. Observe

$$|h(x_0)| = \left|\sum a_i x_0^i\right| = \left|\sum a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i\right|$$

$$\leq \sqrt{w}\|h(xX)\| < 1,$$

but since $h(x_0) \in \mathbb{Z}$ we must have $h(x_0) = 0$. □

Theorem 3.1 suggest we should look for a polynomial $h(x)$ of small weighted norm satisfying $h(x_0) \in \mathbb{Z}$. To do this, we will build a lattice of polynomial related to f and use *LLL* to look for short vectors in that lattice.

Our first observation is that $\dfrac{f(x_0)}{M} \in \mathbb{Z}$ because $f(x_0) \equiv 0 (\text{mod } M)$. Define

$$g_{i,k}(x) := x^i \left(\frac{f(x)}{M}\right)^k.$$

Observe that $g_{i,k}(x_0) = x_0^i \left(\dfrac{f(x_0)}{M}\right)^k$ for all $i, k \geq 0$. Furthermore, this is true for all integer linear combinations of the $g_{i,k}(x)$. The idea behind the Coppersmith technique is to build a lattice L from $g_{i,k}(xX)$ and use LLL to find a short vector in this lattice. The first vector b_1 returned by the LLL algorithm will be a low-norm polynomial $h(xX)$ also satisfying $h(x_0) \in \mathbb{Z}$. If its norm is small enough, Theorem 3.1 would imply $h(x_0) = 0$. Traditional root-finding methods [45] such as Newton-Raphson would then find x_0.

To use Theorem 3.1 we must have $\|h(xX)\| < 1$. Fortunately, Lemma 2.2 allows us to compute a good bound on the norm of the first vector in an LLL-reduced basis. We see

$$\det(L) < 2^{\frac{-w(w-1)}{4}} w^{\frac{-w}{2}} \Rightarrow \|h(xX)\| < \frac{1}{\sqrt{w}}. \tag{3.1}$$

Usually, the "error term" of $2^{-\frac{w(w-1)}{4}} w^{-\frac{w}{2}}$ is insignificant compared to $\det(L)$ and this condition is simplified as

$$\det(L) \ll 1 \Rightarrow \|h(xX)\| < \frac{1}{\sqrt{w}}. \tag{3.2}$$

The determinant of L depends on the choice of polynomials $g_{i,k}(xX)$ defining the lattice. In general, it is a difficult problem to compute the determinant of a lattice when the basis has symbolic entries. However, a careful choice of basis polynomials $g_{i,k}(xX)$ may lead to a lattice with a determinant that can be easily computed. Ideally, we will be able to choose a basis so that the matrix whose rows are the coefficients vectors of $g_{i,k}(xX)$ is *full-rank* and *diagonal*, with an explicit formula for the entries on the diagonal.

We illustrate this technique with a few examples.

Example 3.1. A numerical example.

Suppose we wish to find a root x_0 of the polynomial

$$x^2 - 2849x + 5324 \equiv 0 \pmod{10001} \tag{3.3}$$

satisfying $|x_0| \le 17$. Define

$$f(x) := \frac{x^2 - 2849x + 5324}{10001}.$$

We build a lattice with polynomial

$$\{1, 17x, f(17x), 17xf(17), f^2(17x)\}.$$

Writing this as a matrix gives us

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 0 & 0 \\ 5324 \cdot 10001^{-1} & -2849 \cdot 17 \cdot 10001^{-1} & 17^2 \cdot 10001^{-1} & 0 & 0 \\ 0 & 5324 \cdot 17 \cdot 10001^{-1} & -2849 \cdot 17^2 \cdot 10001^{-1} & 17^3 \cdot 10001^{-1} & 0 \\ 28344976 \cdot 10001^{-2} & -30336152 \cdot 17 \cdot 10001^{-2} & 81227449 \cdot 17^2 \cdot 10001^{-2} & -5698 \cdot 17^3 \cdot 10001^{-2} & 17^4 \cdot 10001^{-2} \end{bmatrix}.$$

The determinant of this lattice is just the product of the entries on the diagonal

$$\det(L) = 17^{10} \cdot 10001^{-4} \approx 2.0 \cdot 10^{-4}.$$

Recall we required $\det(L) < \gamma$, where

$$\gamma = 2^{\frac{-5(5-1)}{4}} 5^{\frac{-5}{2}} \approx 5.6 \cdot 10^{-4}.$$

Hence, condition (3.1) is satisfied. We find that the *LLL* algorithm returns the polynomial

$$h(17x) = \frac{-417605x^4 - 7433369x^3 + 1970691x^2 - 2625174x + 7250016}{10001^2}.$$

This leads to

$$h(x) = \frac{-5x^4 - 1513x^3 + 6819x^2 - 154422x + 7250016}{10001^2}.$$

The roots of $h(x)$ over the reals are $\{16, -307.413\ldots\}$. We find the only integer solution x_0 to Equation (3.3) satisfying $|x_0| \le 17$ is $x_0 = 16$.

Example 3.2. Håstad's original result.

Suppose we are given a monic polynomial $f(x)$ of degree d with integer coefficients, along with integers X and M. We wish to find an integer x_0 such that $|x_0| < X$ and $f(x_0) \equiv 0 \pmod{M}$. An early attempt to solve this problem was given by Håstad [28].

We take as a basis for our lattice L the polynomials

$$\{1,\ Xx,\ (Xx)^2,\ \ldots,\ (Xx)^{d-1},\ \frac{f(Xx)}{M}\}.$$

For instance, when $d = 6$ this results in a lattice spanned by the rows of the matrix in Figure 3.1.

$$\begin{array}{cc} 1 & : \\ Xx & : \\ (Xx)^2 & : \\ (Xx)^3 & : \\ (Xx)^4 & : \\ (Xx)^5 & : \\ f(Xx)/M & : \end{array} \begin{bmatrix} 1 & & & & & & \\ & X & & & & & \\ & & X^2 & & & & \\ & & & X^3 & & & \\ & & & & X^4 & & \\ & & & & & X^5 & \\ - & - & - & - & - & - & X^6M^{-1} \end{bmatrix}.$$

Coppersmith's lattice for finding small solutions to a polynomial congruence. The "–" symbols denote nonzero off-diagonal entries whose values do not affect the determinant.

Figure 3.1. Example Håstad lattice.

The dimension of this lattice is $w = d + 1$ and its determinant is

$$\det(L) = X^{\frac{w(w-1)}{2}} M^{-1}.$$

To satisfy condition (3.1), we require $\det(L) < 2^{\frac{-w(w-1)}{4}} w^{\frac{-w}{2}}$. This leads to

$$X < \gamma_a \cdot M^{\frac{2}{d(d+1)}}, \tag{3.4}$$

where $\frac{1}{2\sqrt{2}} \leq \gamma_a < \frac{1}{\sqrt{2}}$ for all d. Hence, when X satisfies bound (3.4), the LLL algorithm will find a short vector $h(xX)$ satisfying $h(x_0) = 0$ and $\|h(xX)\| < \frac{1}{\sqrt{d}}$. Standard root-finding techniques will recover x_0 from h.

The running time of this method is dominated by the time to run LLL on a lattice of dimension $d + 1$ with entries of size at most $O(\log M)$.

Example 3.3. Coppersmith's generic result.

The first major improvement over Håstad's result came from Coppersmith [16]. Coppersmith suggested including powers and shifts of the original polynomial in the lattice; as we shall see, this is the reason for improved results. We use a presentation by Howgrave-Graham [31]. Again, suppose we are given a monic polynomial $f(x)$ of degree d with integer coefficients, along with integers X and M. We wish to find an integer x_0 such that $|x_0| < X$ and $f(x_0) \equiv 0 \pmod{M}$.

Let $m > 1$ be an integer to be determined later. Define

$$g_{i,k}(x) := x^i \left(\frac{f(x)}{M}\right)^k.$$

We use as a basis for our lattice L the polynomials $g_{i,k}(xX)$ for $i = 0, \ldots, d-1$ and $k = 0, \ldots, m-1$. For instance, when $d = 3$ and $m = 3$ this result in the lattice spanned by the rows of the matrix in Figure 3.2.

$$\begin{array}{r}
g_{0,0}(xX): \\
g_{1,0}(xX): \\
g_{2,0}(xX): \\
g_{0,1}(xX): \\
g_{1,1}(xX): \\
g_{2,1}(xX): \\
g_{0,2}(xX): \\
g_{1,2}(xX): \\
g_{2,2}(xX):
\end{array}
\begin{bmatrix}
1 & & & & & & & & \\
 & X & & & & & & & \\
 & & X^2 & & & & & & \\
- & - & - & X^3M^{-1} & & & & & \\
 & - & - & - & X^4M^{-1} & & & & \\
 & & - & - & - & X^5M^{-1} & & & \\
 & - & - & - & - & - & X^6M^{-2} & & \\
 & - & - & - & - & - & - & X^7M^{-2} & \\
 & & - & - & - & - & - & - & X^8M^{-2}
\end{bmatrix}.$$

Coppersmith's lattice for finding small solutions to a polynomial congruence. The "–" symbols denote nonzero off-diagonal entries whose values do not affect the determinant.

Figure 3.2. Example Coppersmith lattice.

The dimension of this lattice is $w = md$ and its determinant is

$$\det(L) = X^{\frac{w(w-1)}{2}} M^{\frac{-w(m-1)}{2}}.$$

We require $\det(L) < 2^{\frac{-w(w-1)}{4}} w^{\frac{-w}{2}}$; this leads to

$$X < M^{\frac{m-1}{w-1}} 2^{-\frac{1}{2}} w^{\frac{-1}{w-1}},$$

which can simplified to

$$X < \gamma_w \cdot M^{\frac{1}{d}-\varepsilon},$$

where $\varepsilon = \dfrac{d-1}{d(w-1)}$ and $\dfrac{1}{2\sqrt{2}} \le \gamma_w < \dfrac{1}{\sqrt{2}}$ for all w. As we take $m \to \infty$ we have $w \to \infty$ and therefore $\varepsilon \to 0$. In particular, if we wish to solve for up to $X < M^{\frac{1}{d}-\varepsilon_0}$ for arbitrary ε_0, it is sufficient to take $m = O\left(\dfrac{k}{d}\right)$, where $k = \min\{\dfrac{1}{\varepsilon_0}, \log M\}$.

We summarize this in the following theorem:

Theorem 3.2 (Univariate Coppersmith)**.** *Let a monic polynomial $f(x)$ of degree d with integer coefficients and integer X, M be given. Suppose $X < M^{\frac{1}{d}-\varepsilon}$ for some $\varepsilon > 0$. There is an algorithm to find all $x_0 \in \mathbb{Z}$ satisfying $|x_0| < X$ and $f(x_0) \equiv 0 \pmod{M}$. This algorithm runs in time $O(T_{LLL}(md, m \log M))$, where $m = O\left(\frac{k}{d}\right)$ for $k = \min\{\frac{1}{\varepsilon}$,* $\log M\}$.

We note that the generic result is in some sense "blind" to the actual polynomial being used (it takes into account only the degree, but not the coefficients), and that there may be a more optimal choice of polynomials $g_{i,k}$ to include in the lattice to solve a particular problem. By taking into account an optimized set of polynomials, one can improve over this generic (see, for example, [15, 37, 34]).

4. Finding Small Zeros to Bivariate Polynomial Congruences

In this section, we generalize the results of the previous section to finding solutions of bivariate polynomial congruences. We note that this is a different application of these techniques than the solution of bivariate polynomial equations (over the integers) [16]. We note that, in contrast to the previous approach, the method here is only a heuristic.

In order to analyze bivariate polynomial congruences we must introduce a few observations. The first is that there is a simple generalization of Theorem 3.1 to multivariate polynomials.

Theorem 4.1. *Let $h(x, y) \in \mathbb{R}[x, y]$ be a polynomial of which is a sum of at most w monomials, and let $X, Y \in \mathbb{R}$ be given. Suppose that*

(1) *$h(x_0, y_0) \in \mathbb{Z}$ for some $|x_0| < X$ and $|y_0| < Y$;*

(2) $\|h(xX, yY)\| < \frac{1}{\sqrt{w}}$.

Then $h(x_0, y_0) = 0$.

Suppose we are given a polynomial $f(x, y)$ and a real number M, and we wish to find a pair $(x_0, y_0) \in \mathbb{Z} \times \mathbb{Z}$ for which $f(x_0, y_0) \equiv 0 (\operatorname{mod} M)$. The idea is a straightforward generalization of the approach in Section 3. We define

$$g_{i,j,k}(x, y) := x^i y^i \left(\frac{f(x, y)}{M}\right)^k,$$

and observe $g_{i,j,k}(x_0, y_0) \in \mathbb{Z}$ for all $i, j, k \geq 0$. We build a lattice from $g_{i,j,k}(xX, yY)$ by selecting certain indices (i, j, k) so that the determinant of the resulting lattice is "small enough", and compute an *LLL*-reduced basis for this lattice. Lemma 2.2 bounds the norm of the first vector $h_1(xX, yY)$ of this *LLL*-reduced basis, allowing us to use Theorem 4.1 to show that $h_1(x_0, y_0) = 0$.

However, a single bivariate equation may be insufficient to recover the desired root. To obtain another relation, we use the second vector $h_2(xX, yY)$ of the *LLL*-reduced basis. Lemma 2.3 tells us

$$\|h_2(xX, yY)\| \leq 2^{\frac{w}{4}} \left[\frac{\det(L)}{\|h_1(xX, yY)\|}\right]^{\frac{1}{w-1}}. \tag{4.1}$$

So we must also provide a lower bound on the norm of h_1.

Suppose the indices of the $g_{i,j,k}$ are chosen so that $k \leq m$ for some m. Then all coefficients of $g_{i,j,k}(xX, yY)$ are integer multiples of M^{-m}. Thus, since $h_1(xX, yY) \neq 0$, we know it has at least one coefficient greater than or equal to M^{-m} in absolute value. So $\|h_1(xX, yY)\| \geq M^{-m}$. Equation (4.1) becomes

$$\|h_2(xX, yY)\| \leq 2^{\frac{w}{4}} (M^m \det(L))^{\frac{1}{w-1}}.$$

This gives us

$$\det(L) < M^{-m} 2^{\frac{-w(w-1)}{4}} w^{\frac{-(w-1)}{2}} \Rightarrow \|h_2(xX,\ yY)\| < \frac{1}{\sqrt{w}}. \tag{4.2}$$

In particular, this condition is usually simplified as

$$\det(L) \ll M^{-m} \Rightarrow \|h_2(xX,\ yY)\| < \frac{1}{\sqrt{w}}. \tag{4.3}$$

Hence, we obtain another polynomial $h_2(x,\ y) \in \mathbb{R}[x,\ y]$ such that $h_2(x_0,\ y_0) = 0$. It follows that $h_1(x,\ y)$ and $h_2(x,\ y)$ are *linearly* independent. If we make the assumption that $h_1(x,\ y)$ and $h_2(x,\ y)$ are also *algebraically* independent, we can solve for y_0 by computing the resultant $h(y) = \mathrm{Res}_x(h_1,\ h_2)$. Then y_0 must be a root of $h(y)$, and these roots are easily determined. From this, we may find x_0 as a root of $h_1(x,\ y_0)$.

It is not clear why linear independence of h_1 and h_2 should imply algebraic independence, and in fact it is easy to construct (artificial) examples where this is not the case, for instance, the polynomial $x - y \equiv 0 \pmod{M}$ has too many solutions (even in $\mathbb{Z}_2$) thus the method must fail at this step (since all other steps are provable). So at the moment this step of the method is only a heuristic. However, growing experimental evidence [33, 3, 9, 22] shows that it is a very good heuristic for polynomials of interest in cryptology.

Example 4.1. Generic result.

Suppose we are given a polynomial $f(x,\ y)$ of total degree d with at least one monic monomial $x^a y^{d-a}$ of maximum total degree. Also suppose integers $X,\ Y,$ and M are given. We wish to find an integer pair $(x_0,\ y_0)$ such that $|x_0| < X,\ |y_0| < Y,$ and $f(x_0,\ y_0) \equiv 0 \pmod{M}$.

We will follow an approach suggested by Coppersmith [16], worked out in detail by Jutla [33]. Let $m > 1$ be an integer to be determined later. Define

$$g_{i,j,k}(x, y) := x^i y^i \left(\frac{f(x)}{M}\right)^k.$$

We use as a basis for our lattice L the polynomials $g_{i,j,k}(xX, yY)$, where the indices (i, j, k) come from the following set:

$$S := \left\{(i, j, k) \in \mathbb{Z}^3 | i + j + kd \leq md \text{ and } i, j, k \geq 0 \text{ and } (i < a \text{ or } j < d - a)\right\}.$$

Denote by S_m the set of polynomials $g_{i,j,k}(xX, yY)$ such that $(i, j, k) \in S$. Every $g \in S_m$ has total degree less than md. Indeed, the set S_m is in one-to-one correspondence with the set of monomials $\{x^{\alpha} y^{\beta} | \alpha + \beta \leq md\}$ given by $g_{i,j,k}(xX, yY) \leftrightarrow x^{i+ka} y^{j+k(d-a)}$. We may write these polynomials as the rows of a matrix in a way that puts the coefficient of the corresponding monomial on the diagonal. The resulting matrix is lower diagonal, and the contribution of $g_{i,j,k}(xX, yY) \in S_m$ to the diagonal is $M^{-k} X^{i+ka} Y^{j+k(d-a)}$. A straightforward but tedious calculation shows that the resulting lattice has determinant

$$\det(L) = \left(\prod_{\alpha=0}^{md} \prod_{\beta=0}^{md-\alpha} X^{\alpha} Y^{\beta}\right)\left(M^{-m} \prod_{k=0}^{m-1} (M^{-k})^{\frac{d}{2}((2m-2k-1)d+3)}\right).$$

For simplicity we carry out the computation using low-order terms. We find

$$\det(L) = (XY)^{\frac{d^3}{6} m^3 + o(m^3)} M^{-\frac{d^2}{6} m^3 + o(m^3)}.$$

To use condition (4.3), we require $\det(L) \ll M^{-m}$. This leads to

$$XY < M^{\left(\frac{1}{d}\right) - \varepsilon},$$

where $\varepsilon \to 0$ as $m \to \infty$. (We note that to use the more precise condition (4.2) requires $\det(L) < \gamma_m \cdot M^{-m}$, where $\gamma_m = 2^{\frac{-w^2}{2}} w^{\frac{-w}{2}}$ for $w = O(m^2 d^2)$. So $-\log_M \gamma_m = \frac{O(m^4 d^4)}{\log M}$, implying the method will not work if m is too large and M is too small. In most applications, however, we find $\log_M \gamma_m \approx 0$ and this term may be safely-ignored.)

We note that the shape or coefficients of a particular polynomial may allow for a better selection of basis polynomials $g_{i,j,k}$. For instance, when $f(x, y)$ has degree d in each variable separately and is monic in the monomial $x^d y^d$, a different choice of basis leads to the improved bound $XY < N^{\frac{2}{3d}}$.

5. Public Key Cryptography

In this section, we present the notion of a public key cryptosystem, and in particular, the RSA public key cryptosystem. There are many good formal definitions for public key cryptosystems [25, 21, 39, 49], and we do not try to cover all of them here. Instead we try to develop the intuition that will be useful later.

A public key (or asymmetric) cryptosystem is a method for securing communication between parties who have never met before. More precisely, a public key cryptosystem is described by the following:

- a set $\mathcal{M}$ of *plaintexts* (or *messages*), and a set $\mathcal{C}$ of *ciphertexts*;
- a set $\mathcal{K}_p$ of *public keys*, and a set $\mathcal{K}_s$ of *secret keys*;
- a *key generation* algorithm key-gen: $\mathbb{Z} \to \mathcal{K}_p \times \mathcal{K}_s$;
- an *encryption* algorithm $\mathcal{K}_p \times \mathcal{K} \to \mathcal{C}$; and,
- a *decryption* algorithm $D : \mathcal{K}_s \times \mathcal{C} \to \mathcal{M}$.

The key generation, encryption, and decryption algorithms can be randomized and should run in expected time polynomial in the length of their inputs. For all $\langle K_p, K_s \rangle$ output by "key-gen" and all messages $M \in \mathcal{M}$ we must have that $D(K_s, E(K_p, M)) = M$.

The input to the key generation algorithm is called the *security parameter*. The hope is that as the security parameter increases, the resources required to break the cryptosystem using the resulting keys should increase more rapidly than the resources required to use it. Ideally, the running time of a break should be a (sub-) exponential function of n, while the running time of key-gen, E, and D should be some (small) polynomial in n.

Suppose Ali is a user of a public key cryptosystem. To initialize she chooses a security parameter n and computes $\langle K_p, K_s \rangle$:= key-gen(n). When another user Mohd wishes to send a message to Ali securely, he obtains Ali's public key K_p and computes the ciphertext $C := E(K_p, M)$. He sends ciphertext C is sent to Ali, who upon obtaining it computes the original message $M = D(K_s, C)$.

The security requirements of a cryptosystem can be defined in many ways. In general, when defining a security goal it is important to state what resources are available to an attacker and *what success criteria* the attacker must fulfill. A very basic requirement is that it should not be possible to derive the secret key from the public key efficiently; indeed, it is considered the most devastating cryptanalytic break to compute K_s from K_p in (say) time polynomial in the security parameter. We will see examples of this in next sections. We might consider security against partial key exposure, where information about K_s (say perhaps a subset of bits of K_s) allows an attacker to compute all of K_s. There are many issues that arise in determining good notions of security, and we do not try to address them all here. There are many good surveys on the subject [39, 21].

Since the publication of New Directions, there have been countlessly many proposals for public key cryptosystems. Our primary focus in this work, however, will be on the RSA public key cryptosystem and simple variants [46, 51, 50]. Now we present the basic RSA scheme in the next section.

6. The RSA Public Key Cryptosystem

In this section, we outline the basic RSA public key cryptosystem [46].

Let n be a security parameter. The key generation algorithm for RSA computes primes p and q approximately $n/2$ bits in length, so that $N := pq$ is an integer n bits in length. More precisely, p and q are random primes subject to the constraint that $N = pq$ is an n-bit number and

$$\frac{\sqrt{N}}{2} < q < p < 2\sqrt{N}.$$

We denote the set of all such N as $\mathbb{Z}_{(2)}$. Typically $n = 1024$, so that N is 1024 bits in length; p and q are primes typically chosen to be approximately 512 bits each.

The key generation algorithm selects integers e and d such that $ed \equiv 1 \bmod \phi(N)$, where $\phi(N) = N - p - q + 1$ (also called the Euler totient function). We call e the *public exponent* and d the secret exponent. The value N is called the *public modulus*. An RSA public key is the pair of integers $\langle N, e \rangle$. The corresponding secret key is the pair $\langle N, d \rangle$. Thus $\mathcal{K}_p = \mathcal{K}_s = \mathbb{Z}_{(2)} \times \mathbb{Z}$.

How e and d are chosen depends on the application. Typically, e is chosen to satisfy certain constraints (say e is small, like $e = 3$), then d is picked from $\{1, \ldots, \phi(N)\}$ to satisfy $ed \equiv 1 \bmod \phi(N)$. However, this process may be done in reverse, and in many applications d is chosen first (say to make d short, as in Subsection 7.4).

Messages and ciphertexts are represented as elements of $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$. Suppose Mohd wishes to send a message $M \in \mathbb{Z}_N^*$ to Ali. He obtains Ali's public key $\langle N, e \rangle$ and computes $C = M^e \bmod N$ which he sends to Ali. Upon receiving C, Ali may compute

$$C^d \equiv M^{ed} \equiv M (\bmod N),$$

where the last equivalence follows from Euler's theorem.

For digital signing, the roles of these operations are reversed. If Ali intends to sign the message M she computes $S = M^d \bmod N$ and sends $\langle M, S \rangle$ to Mohd. Mohd checks $M \overset{?}{\equiv} S^e \bmod N$.

This presentation simplifies RSA encryption and signing; in practice, randomized padding of the messages [1, 5] is required before exponentiation to prevent several security flaws [2, 20, 19]. We will not go into the details here, since all attacks in subsequent sections succeed regardless of the padding scheme that is being used.

7. Previous Attacks on RSA

In this section, we summarize several previously-known attacks on the RSA public key cryptosystem relevant to this work. We follow the presentation of the recent survey of attacks on RSA [6] and refer to it for a comprehensive listing of attacks on RSA.

7.1. Factoring

The most straightforward attack on RSA is factorization of the modulus $N = pq$. Once a factor p is discovered, the factor $q = N / p$ may be computed, so $\phi(N) = N - p - q + 1$ is revealed. This is enough to compute $d \equiv e^{-1} \bmod \phi(N)$.

The current fastest method for factoring is the “general number field sieve” [26]. It has a running time of $\exp((c + O(1)) \cdot (\log N)^{1/3} (\log \log N)^{2/3})$ for some $1 < c < 2$. The size of N is chosen to foil this attack. The largest integer that has been successfully factored using this method was the 512-bit RSA challenge modulus RSA-155, factored in 1999 using a massive distributed implementation of GNFS on the Internet [14]. Even though the speed of computer hardware continues to accelerate, it seems unlikely that the best factoring algorithms will be able to factor say 1024-bit RSA moduli in the next twenty years.

7.2. Håstad’s attack on broadcasted messages

In order to speed up RSA encryption (and signature verification), it is useful to use small value for the public exponent e, say $e = 3$. However, this opens up RSA to the following attack, discovered by Håstad [28].

Let us start with a simpler version. Suppose Mohd wishes to send the same message M to k recipients, all of whom are using public exponent equal to 3. He obtains the public keys $\langle N_i, e_i \rangle$ for $i = 1, \ldots, k$, where $e_i = 3$ for all i. Naively, Mohd computes the ciphertext $C_i = M^3 \bmod N_i$ for all i and sends C_i to the i-th recipient.

A simple argument shows that as soon as $k \geq 3$, the message M is no longer secure. Suppose Eman intercepts C_1, C_2, and C_3, where $C_i = M^3 \bmod N_i$. We may assume $\gcd(N_i, N_j) = 1$ for all $i \neq j$ (otherwise, it is possible to compute a factor of one of the N'_i s). By the Chinese Remainder Theorem, she may compute $C \in \mathbb{Z}^*_{N_1, N_2, N_3}$ such that $C \equiv C_i \bmod N_i$. Then $C \equiv M^3 \bmod N_1 N_2 N_3$; however, since $M < N_i$ for all i, we have $M^3 < N_1 N_2 N_3$. Thus $C = M^3$ holds over the integers, and Eman can compute the cube root of C to obtain M.

Håstad proves a much stronger result. To understand it, consider the following naive defense against the above attack. Suppose Mohd applies a pad to the message M prior to encrypting it so that the recipients receive slightly different messages. For instance, if M is m bits long, Mohd might encrypt $i \cdot 2^m + M$ and send this to the i-th recipient. Håstad proved that this linear padding scheme is not secure. In fact, he showed that any fixed polynomial applied to the message will result in an insecure scheme.

Theorem 7.1 (Håstad)**.** *Suppose* $N_1, \ldots, N_k$ *are relatively prime integers and set* $N_{\min} = \min_i(N_i)$. *Let* $g_i(x) \in \mathbb{Z}_{N_i}[x]$ *be* k *polynomials of maximum degree* d. *Suppose there exists a unique* $M < N_{\min}$ *satisfying*

$$g_i(M) = 0 \pmod{N_i} \quad \textit{for all } i \in \{0, \ldots, k\}.$$

Furthermore suppose $k > d$. *There is an efficient algorithm which, given* $\langle N_i, g(x)\rangle$ *for all i, computes M.*

Proof. Since the N_i are relatively prime, we may use the Chinese Remainder Theorem to compute coefficients T_i satisfying $T_i \equiv 1 \pmod{N_i}$ and $T_i \equiv 0 (\bmod N_j)$ for all $i \neq j$. Setting $g(x) := \sum_i T_i g_i(x)$ we see $g(M) = 0(\bmod \prod N_i)$. Since the T_i are nonzero we have that $g(x)$ is not identically zero. If the leading coefficient of $g(x)$ is not one, then we may multiply by its inverse to obtain a monic polynomial $g(x)$.

The degree of $g(x)$ is at most d. By Coppersmith's theorem (Theorem 3.2), we may compute all integer roots x_0 satisfying $g(x_0) \equiv 0 \bmod \prod N_i$ and $|x_0| < (\prod N_i)^{1/d}$. But we know $M < N_{\min} < (\prod N_i)^{1/k} < (\prod N_i)^{1/d}$, so M is such a root. □

This can be applied to the problem of broadcast RSA as follows. Suppose the i-th plaintext is padded with a polynomial $f_i(x)$, so that $C_i \equiv (f_i(M))^{e_i} (\text{mod } N_i)$. Then the polynomials $g_i(x) := (f_i(x))^{e_i} - C$ satisfy the above relation. The attack succeeds once $k > \max_i(e_i \cdot \deg f_i)$.

We note that Håstad's original result was significantly weaker, requiring $k = O(d^2)$ messages, where $d = \max_i(e_i \cdot \deg f_i)$. This is because the original result used the Håstad method for solving polynomial congruences (see Example 3.2) instead of the full Coppersmith method.

This attack suggests that randomized padding should be used in RSA encryption.

7.3. Coppersmith attack on short random pads

Like the previous attack, this attack exploits a weakness of RSA with public exponent $e = 3$. Coppersmith showed that if randomized padding is used improperly then RSA encryption is not secure [16]. Coppersmith addressed the following question: if randomized padding is used with RSA, how many bits of randomness are needed ?

To motivate this question, consider the following attack. Suppose Mohd sends a message M to Ali using a small random pad before encrypting. Eman obtains this and disrupts the transmission, prompting Mohd to resend the message with a new random pad. The following attack shows that even though Eman does not know the random pads being used, she can still recover the message M if the random pads are too short.

For simplicity, we will assume the padding is placed in the least significant bits, so that $C_i = (2^m M + r_i)^e \ (\text{mod } N)$ for some small m and random $r < 2^m$. Eman now knows

$$C_1 = (2^m M + r_1)^e \quad (\text{mod } N) \quad \text{and} \quad C_2 = (2^m M + r_2)^e \quad (\text{mod } N),$$

for some unknown M, r_1 and r_2. Define

$$f(x, y) := x^e - C_1 \text{ and } g(x, y) := (x + y)^e - C_2.$$

We see that when $x = 2^m M + r_1$, both of these polynomials have $y = r_2 - r_1$ as a root mod N. We may compute the resultant $h(y) := \text{Res}_x\ (f, g)$ which will be of degree at most e^2. Then $y = r_2 - r_1$ is a root of $h(y) \bmod N$. If $|r_i| < (1/2)N^{1/e^2}$ for $i = 1, 2,$ then we have that $|r_2 - r_1| < N^{1/e^2}$. By Coppersmith's theorem (Theorem 3.2), we may compute all of the roots $h(y)$, which will include $r_2 - r_1$. Once $r_2 - r_1$ is discovered, we may use a result of Franklin and Reiter [18] to extract M (see [6] for details).

7.4. Wiener's attack on short secret exponent

To speed up RSA decryption and signing, it is tempting to use a small secret exponent d rather a random $d \leq \phi(N)$. Since modular exponentiation takes time linear in $\log_2 d$, using a d that is substantially shorter than N can improve performance by a factor of 10 or more. For instance, if N is 1024 bits in length and d is 80 bits long, this results in a factor of 12 improvement while keeping d large enough to resist exhaustive search.

Unfortunately, a classic attack by Wiener [53] shows that a sufficiently short d leads to an efficient attack on the system. His method uses approximations of continued fractions. This attack is stated in the following theorem:

Theorem 7.2 (Wiener)**.** *Suppose* $N = pq$ *and* $\frac{\sqrt{N}}{2} < q < p < \sqrt{N}$. *Furthermore suppose* $d < \frac{1}{3}N^{1/4}$. *There is an algorithm which, given N and e, generates a list of length* log N *of candidates for d, one of which will equal d. This algorithm runs in time linear in* log N.

Proof. Since $ed \equiv 1 (\text{mod}\, \phi(N))$, there is some k such that $ed - k\phi(N) = 1$. We may write this as

$$\left|\frac{e}{\phi(N)} - \frac{k}{d}\right| = \frac{1}{d\phi(N)}.$$

Hence $\frac{e}{\phi(N)}$ is an approximation to $\frac{k}{d}$. The attacker does not know $\phi(N)$, but he does know N. Since $\frac{\sqrt{N}}{2} < q < p < 2\sqrt{N}$ we have $p + q - 1 < 3\sqrt{N}$, and thus $N - \phi(N) < 3\sqrt{N}$. Now if the attacker uses $\frac{e}{N}$ as an approximation we find

$$\left|\frac{e}{N} - \frac{k}{d}\right| = \left|\frac{ed - k\phi(N) + k\phi(N) - kN}{Nd}\right|$$

$$= \left|\frac{1 - k(N - \phi(N))}{Nd}\right| \leq \left|\frac{3k\sqrt{N}}{Nd}\right| = \left|\frac{3k}{d\sqrt{N}}\right|.$$

Since $e < \phi(N)$, we know $k < d < \frac{1}{3}N^{1/4}$. Thus

$$\left|\frac{e}{N} - \frac{k}{d}\right| = \frac{1}{dN^{1/4}} < \frac{1}{2d^2}.$$

This is a classic approximation relation, and there are well-known methods [27, Theorem 177] to solve it. Such methods produce a list of all integers pairs (k_i, d_i) satisfying $\gcd(k_i, d_i) = 1$ and

$$\left|\frac{e}{N} - \frac{k_i}{d_i}\right| < \frac{1}{2d_i^2}.$$

This list is of length at most log N. Since $ed - k\phi(N) = 1$ we know $\gcd(k, d) = 1$. Hence, $d = d_i$ for some $i \in \{1, \ldots, \log N\}$. □

8. Cryptanalysis via the Defining Equation

Since $ed \equiv 1 \bmod \phi(N)$, this implies there exists an integer k such that

$$ed + k(N + 1 - 1(p + q) = 1. \tag{8.1}$$

This equation succinctly summarizes the RSA, and we will refer to it frequently throughout this work.

As discussed earlier, a break of the RSA public key cryptosystem can be defined in several ways. Most obviously, the scheme is broken if an attacker is able to recover the secret exponent d. Since factorization of the modulus $N = pq$ leads to recovery of the private key d, this is also a total break. All of the attacks presented in subsequent sections are of this type, and involve either a direct computation of the private key d or one of the factors p of the public modulus N, given the public key information $\langle N, e\rangle$ alone.

In [4, 7, 8, 10, 11, 12, 13], we can see several examples where the value $s = p + q$ is computed from the public information. We note that this immediately allows the recovery of the factorization of N; indeed, when $s = p + q$, then p and q are the two roots of the equation $x^2 - sx + N = 0$.

We emphasize that our results in this work come from the basic RSA equations; our attacks do not use plaintext/ciphertext pairs or signatures, so they hold regardless of any padding schemes used. It is an interesting open question to determine if the attacks presented in this work can be improved if a particular padding is in use, or if the adversary is given access to known or chosen plaintext/ciphertext pairs or chosen signatures.

9. The Lattice Factoring Method

In recent years, integers of the form $N = p^r q$ have found applications in cryptography. For example, Fujioke et al. [23] use a modulus $N = p^2 q$ in an electronic cash scheme. Okamoto and Uchiyama [44] use $N = p^2 q$ for an elegant public key system. Recently, Takagi [51] observed that RSA decryption can be performed significantly faster by using a modulus of the form $N = p^r q$. In all of these applications, the factors p and q are primes of 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 integers (not necessarily prime) 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 $\sqrt{\log p}$, our algorithm factors N in polynomial time. This is a new class of integers that can be factored efficiently. This is discussed in Subsection 11.2. When $N = p^r q$ with r on the order of $\sqrt{\log p}$, our algorithm factors N faster than the best previously-known method- the elliptic curve method (ECM) [38]. 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 integers of the form $N = p^r q$ with large r are inappropriate for cryptographic purposes. In particular, Takagi's proposal [47] should not be used with a large r. Here is a rough idea of how the algorithm's efficiency depends on the parameter r. Suppose p and q are k-bit integers and $N = p^r q$. When $r = k^{\varepsilon}$, the our method runs (asymptotically) in time $T(k) = 2^{(1-\varepsilon)+O(\log k)}$. Hence, when $\varepsilon = 1$, the modulus N is roughly k^2 bits long and the algorithm will factor N in polynomial time in k. When $\varepsilon = \frac{1}{2}$, the algorithm asymptotically outperforms ECM. The algorithm's efficiency and its comparison with previously-known factoring methods is discussed in Section 13.

We ran experiments to compare our method to ECM factoring. It is most interesting to compare the algorithms when $\varepsilon \approx \frac{1}{2}$, namely, $r \approx \sqrt{\log p}$. Unfortunately, since $N = p^r q$ rapidly becomes too large to handle, we could only experiment with small values of p. 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 12.

An additional feature of our algorithm is that it is able to make use of partial information about a factor. This is sometimes called factoring with a hint. In particular, our method gives an algorithm for factoring $N = pq$ when half of the bits of the factor p are known. This gives an elegant restatement of a theorem originally due to Coppersmith [16] for factoring with a hint using bivariate polynomial equations. In the case that $r = 1$, our presentation also coincides with an extension of Coppersmith's theorem developed by Howgrave-Graham [31]. Our version has several practical advantages, and will be an important tool used in partial key exposure attacks discussed in the next section. This is discussed in Subsection 11.1.

10. Algorithm to Factor Integers of the Form $N = p^r q$

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. Recall that $\exp(n) = 2^n$ and logarithms should be interpreted as logarithms to the base 2.

Theorem 10.1. *Let $N = p^r q$ where $q < p^e$ for some e. 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 $\gamma = T_{LLL}(r^2, (r+c)\log N)$. The algorithm is deterministic, and runs in polynomial space.

Note that 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 $c \approx 1$ we have that $\frac{c+1}{r+c} = O\left(\frac{1}{r}\right)$. Hence, the larger r is, the easier the factoring problem becomes. When $r = \varepsilon \log p$ 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) [38].

• For small r, the algorithm runs in 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.

We prove Theorem 10.1 by extending the approach for finding solutions to univariate congruences developed in Section 4. The main tool we will need is the following slight variant of Theorem 3.1.

Lemma 10.1. *Let $h(x) \in \mathbb{R}[x]$ be a polynomial of degree w, and let $m \in \mathbb{Z}$ and $X \in \mathbb{R}$ be given. Suppose there is some $|x_0| < X$ such that*

(1) *$h(x_0) \in q^{-m}\mathbb{Z}$, and*

(2) $\|h(xX)\| < q^{-m} / \sqrt{w}$.

Then $h(x_0) = 0$.

Proof. Apply Theorem 3.1 to the polynomial $q^m h(x)$. □

Note that for simplicity we assume r and c are given to the algorithm of Theorem 10.1. Clearly, this is not essential since one can try all possible values for r and c until the correct values are found.

10.1. 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 the other words, $|P - p| < X$ for some large X. For now, our objective is to find p given N, r, and P. This is clearly equivalent finding the point $x_0 := P - p$.

Define the polynomial $f(x) := (P + x)^r / N$ and observe $f(x_0) = 1 / q$. Let $m > 0$ be an integer to be determined later. For $k = 0, \ldots, m$ and any $i \geq 0$ define

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

Observe that $g_{i,k}(x_0) = x_0^i q^{-k} \in q^{-m} \mathbb{Z}$ for all $i \geq 0$ and all $k = 0, \ldots, m$.

Theorem 3.1 suggests that we should look for a low-norm integer linear combination of the $g_{i,k}$ of weighted norm less than $q^{-m} / \sqrt{w}$. Let L be the lattice spanned by

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

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

The values of m and w will be determined later. To use Lemma 2.2, we must bound the determinant of the resulting lattice. Let M be a matrix whose rows are the coefficients vectors for the basis of L (see Figure 10.1). 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

$$\det(M) = \left(\prod_{k=0}^{m-1}\prod_{i=0}^{r-1} X^{rk+i}N^{-k}\right)\left(\prod_{j=mr}^{w-1} X^{j}N^{-m}\right)g \leq X^{w^2/2}N^{mr(m+1)/2-wm}.$$

Lemma 2.2 guarantees that the LLL algorithm will find a short vector $h(xX)$ in L satisfying

$$\|h(xX)\|^{w} \leq 2^{w^2/4}\det(L) \leq 2^{w^2/4}X^{w^2/2}N^{mr(m+1)/2-wm}. \tag{10.1}$$

Furthermore, since $h(xX)$ is an integer linear combination of the $g_{i,k}(xX)$, the corresponding $h(x)$ as an integer linear combination of the $g_{i,k}(x)$. Therefore $h(x_0) \in q^{-m}\,\mathbb{Z}$.

$$\begin{array}{c|ccccccccc|}
 & 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 \\
g_{0,0}(xX) & 1 & & & & & & & & \\
g_{1,0}(xX) & & x & & & & & & & \\
g_{0,1}(xX) & - & - & x^2N^{-1} & & & & & & \\
g_{1,1}(xX) & & - & - & x^3N^{-1} & & & & & \\
g_{0,2}(xX) & - & - & - & - & x^4N^{-2} & & & & \\
g_{1,2}(xX) & & - & - & - & - & x^5N^{-2} & & & \\
g_{0,3}(xX) & - & - & - & - & - & - & x^6N^{-3} & & \\
g_{1,3}(xX) & & - & - & - & - & - & - & x^7N^{-3} & \\
g_{2,3}(xX) & & & - & - & - & - & - & - & x^8N^{-3}
\end{array}.$$

Example LFM lattice for $N = p^2q$ when $m = 3$ and $d = 9$. The entries marked with "–" represent non-zero off-diagonal entries we may ignore.

Figure 10.1. Example LFM lattice.

To apply Theorem 3.1, we also require that

$$\|h(xX)\| < q^{-m} / \sqrt{w}.$$

Plugging in the bound on $\|h(xX)\|$ from Equation (10.1) and reordering terms, we see this condition is satisfied when

$$(\sqrt{2}X)^{w^2/2} < q^{w^2/2} N^{-wm} N^{wm} N^{-mr(m+1)/2} / (\sqrt{w})^w. \tag{10.2}$$

We may substitute $q^{-1}N = p^r$. Because $q < p^c$ for some c, we know $N < p^{c+r}$. So inequality (10.2) is satisfied when the following holds:

$$(\sqrt{2}X)^{w^2/2} < q^{wm-mr(c+r)(m+1)/2} / (\sqrt{w})^w.$$

We note that $(\sqrt{w})^{(2/w)} \le \sqrt{2}$ for all $w \ge 4$, so this leads to

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

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 $m_0 = \left\lfloor \frac{w}{r+c} - \frac{1}{2} \right\rfloor$, and we may choose w_0 so that w_0 so that $\frac{w_0}{r+c} - \frac{1}{2}$ is within $\frac{1}{2r+c}$ of an integer. Plugging in $m = m_0$ and $w = \max\{w_0, 4\}$ and working through tedious arithmetic results in the bound

$$X < (1/2)p^{1 - \frac{c}{r+c} \frac{r}{w}(1+\delta)}, \quad \text{where} \quad \delta = \frac{1}{r+c} - \frac{r+c}{4w}.$$

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

$$X < (1/2)p^{1 - \frac{c}{r+c} - 2\frac{r}{w}}. \tag{10.3}$$

So when X satisfies inequality (10.3), the LLL algorithm will find a vector $h(xX)$ in L satisfying $\|h(xX)\| < q^{-m} / \sqrt{w}$. The polynomial $h(x)$ is an integer linear combination of the $g_{i,k}(x)$ and thus satisfies

$h(x_0) \in q^{-m}\mathbb{Z}$. But since $w \geq 4, \|h(xX)\|$ is bounded, we have by Theorem 3.1 that $h(x_0) = 0$. Traditional root-finding methods [45] such as Newton-Raphson can extract the roots of $h(x)$. Given a candidate for x_0, it is easy to check if $P + x_0$ divides N. Since h is of degree $w - 1$, there are at most w roots to check before x_0 is found and the factorization of N is exposed.

We summarize this result in the following theorem:

Theorem 10.2. *Let $N = p^r q$ be given, and assume $q < p^c$ for some c. Furthermore assume that P is an integer satisfying*

$$|P - p| < (1/2)p^{1-\frac{c}{r+c}-2\frac{r}{w}},$$

for some w. Then the factor p may be computed from N, r, c, and P by an algorithm whose running time is dominated by $T_{LLL}(w, \lfloor 2w/r \rfloor \cdot \log N)$.

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

Proof of Theorem 10.1. Suppose $N = p^r q$ with $q < p^c$ for some c. Let $w = 2r(r + c)$. Then, by Theorem 10.2 we know that given an integer P satisfying

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

the factorization of N can be found in time $T_{LLL}(w, \lfloor 2w/r \rfloor \cdot \log N)$. Let $\varepsilon := \left\lceil \frac{c+1}{r+c} \right\rceil$. We proceed as follows:

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

(b) For all $j = 0, \ldots, 2^{\varepsilon k+1}$ do: n

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

(d) Run the algorithm of Theorem 10.2 using the approximation P.

The outermost loop to determine the length k of p is not necessary if the size of p is known. If p is k bits long then one of the candidate values P generated in step (c) will satisfy $|P - p| < 2^{(1-\varepsilon)k-1}$ and hence $|P - p| < (1/2)p^{1-\varepsilon}$ as required. Hence, the algorithm will factor N in the required time.

11. Applications

11.1. Factoring $N = pq$ with a hint

One immediate application of Theorem 10.2 is the factorization of integers of the form $N = pq$ when partial information about one of the factors is known. Suppose $(n/4) + \delta$ bits of one of factors of an n-bit value $N = pq$ is known for some small $\delta > 0$, Coppersmith showed [16] how to apply a bivariate version of his technique to solve the integer factorization problem using this partial information. The lattice factoring method described here is the first application of the univariate Coppersmith method to the problem of integer factorization with a hint. This method generalizes to integers of the form $N = p^r q$, while the bivariate method does not. Therefore, it appears this technique appears to be superior to the original bivariate Coppersmith approach.

Our method has significant performance advantages over the original bivariate method of Coppersmith. Given $(n/4) + \delta$ bits of one of the factors of $N = pq$, the bivariate method of Coppersmith builds a lattice $n^2/(9\delta^2)$ with entries of size at most $n^2/(2\delta)$. Our method creates lattices of dimension n/δ with entries of size at most $2n^2/\delta$. This results in a substantial performance improvement.

Factoring $N = pq$ knowing MSBs of p. We first describe the result for $N = pq$ when most significant bits of p are known.

Corollary 11.1 (MSBFact)**.** *Let $N = pq$ of binary length n be given and assumeq $q < p$. Furthermore assume $P > 0$ and $t > n/4$ are integers satisfying*

$$|P - p| < 2^{(n/2)-t}.$$

Then there is an algorithm that given N, P, and t computes the factor p in time $T_{LLL}(w, 2nw)$, where $w = \left\lceil \frac{n}{t-(n/4)} \right\rceil$. We denote the running time of this algorithm by $T_{\text{MSBFact}}(n, t)$.

Proof. In order to use Theorem 10.2 we must choose w such that

$$|P - p| < (1/2)p^{1-\frac{1}{2}-\frac{2}{w}}.$$

This is achieved once $|P - p| < 2^{\frac{n}{2}\left(\frac{1}{2}-\frac{2}{w}\right)}$, i.e., $w \geq \frac{n}{t-(n/4)}$. □

Some comments:

• When P and p are $n/2$-bit integers such that P matches p on the t most significant bits, we have $|P - p| < 2^{(n/2)-t}$. Informally, we say that MSBFact is given the t most significant bits of p.

• Note that as t increases, $T_{\text{MSBFact}}(n, t)$ decreases. While this follows from the fact that w is inversely proportional to t, it is also makes intuitive sense since the factoring problem naturally becomes easier as more bits of p are given to MSBFact.

• This algorithm can be extended to $t \leq n/4$ by running MSBFact sequentially with approximations

$$P_j := P - 2^{(n/2)-t} + (2j-1)2^{(n/4)-1},$$

for $j = \{1, \dots, 2^{(n/4)-t+1}\}$. One of these P_j will satisfy $|P_j - p| < 2^{(n/4)-1}$. Hence the total running time is $2^{(n/4)-t+1}.T_{\text{MSBFact}}(n, (n/4) + 1).2_{(n/4)-t+1}$.

• For $t = (n/4) + 1$ we have $w = n$. In practice, this may result in a lattice too large to handle (e.g., factoring $n = 1024$ bit RSA moduli). The previous trick can be applied to get a running time of $2^{(n/4)-t+c}$. $T_{\text{MSBFact}}(n, (n/4) + C)$ for any $C > 0$ to get $w = \lceil n/C \rceil$.

• In most cases w can be taken to be much smaller, but the method is no longer provable.

Factoring $N = pq$ knowing LSBs of p. We now describe a slight variant of the lattice factoring method that can be used to factor $N = pq$ when least significant bits of a factor p are known.

Corollary 11.2 (LSBFact)**.** *Let $N = pq$ of binary length n be given and assume $q < p$. Furthermore assume $P > 0$, $R > 0$, and $n/2 \geq t > n/4$ are integers satisfying*

$$P \equiv p(\text{mod } R), \quad R \geq 2^t.$$

Then there is an algorithm that given N, P, R, and t computes the factor p in time $T_{LLL}(w, 6nw)$, where $w = \left\lceil \frac{n}{t-(n/4)} \right\rceil$. We denote the running time of this algorithm by $T_{\text{LSBFact}}(n, t)$.

Proof. In this problem, we are seeking to discover the value $x_0 := (P-p)/R$, where $|x_0| < 2^{n/2-t}$. We cannot apply Theorem 10.2 directly, so we derive the following variant. We have $\gcd(R, N) = 1$ (otherwise we know the factorization of N immediately), so we can compute a and b satisfying $aR + bN = 1$. Define the polynomial

$$f(x) := (aP + x)^r / N,$$

and observe

$$f(aRx_0) = (aP + aRx_0)^r / N = (ap)^r / N = a^r / q \in q^{-1}\mathbb{Z}.$$

But

$$f(aRx_0) = f(x_0 - bNx_0) = f(x_0) + C,$$

for some $C \in \mathbb{Z}$, so $f(x_0) \in q^{-1}\mathbb{Z}$.

This encodes the factorization problem as a univariate root-finding problem, and we use exactly the same techniques used in Subsection 10.1 to solve it. Namely, let $m > 0$ be an integer to be determined later. For $k = 0, \dots, m$ and any $i \geq 0$ define

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

Then $g_{i,k}(x_0) \in q^{-m}\mathbb{Z}$ for all $i \geq 0$ and all $k \leq m$. We build a lattice from these polynomials and use *LLL* to find a short vector. The proof follows as in Lemma 10.2, and we derive the same bound $|x_0| < (1/2)$ $p^{1-\frac{c}{r+c}-2\frac{c}{w}}$. In this case $r = c = 1$, so the bound is achieved once $|x_0| \leq 2^{\frac{n}{2}(\frac{1}{2}-\frac{2}{w})}$, i.e., $w \geq \frac{n}{t-(n/4)}$. □

Some comments:

• When P and p are $n/2$-bit integers such that P matches p on the t least significant bits, we have $P \equiv p(\bmod 2^t)$. Informally, we say that LSBFact is given the t least significant bits of p.

• Note that as t increases, $T_{\text{LSBFact}}(n, t)$ decreases. While this follows from the fact that w is inversely proportional to t, it is also makes intuitive sense since the factoring problem naturally becomes easier as more bits of p are given to LSBFact.

• This algorithm can be extended to $t \leq n/4$ by running LSBFact with $R' = 2^{(n/4)-t}.R$, and approximations $P_j := P + (j-1)R$ for $j = \{1, \ldots, 2^{(n/4)-t}\}$. One of these P_j will satisfy $P_i \equiv p \bmod R'$, where $R' \geq 2^{n/4}$. Hence the total running time is $2^{(n/4)-t+1}.T_{\text{LSBFact}}(n, (n/4)+1)$.

• For $t = (n/4)+1$ we have $w = n$. In practice, this may result in a lattice too large to handle (e.g., factoring $n = 1024$ bit RSA moduli). The previous trick can be applied to get a running time of $2^{(n/4)-t+C}. T_{\text{LSBFact}}(n, (n/4)+C)$ for any $C > 0$ to get $w = \lceil n/C \rceil$.

• In most cases w can be taken to be much smaller, but the method is no longer provable, see [30, 37, 41, 42, 52].

11.2. Polynomial-time factoring for $N = p^r q$, $r = \Omega(\log p)$

When $r \approx \log p$ the lattice factoring method runs in time polynomial in $\log N$. This is a new class of integers that can be efficiently factored. We state this formally in the following:

Corollary 11.3. *Let $N = p^r q$ where $q < p^c$ for some c. Suppose $r = M \log p$. The factor p can be recovered from N, r, and c by an algorithm with a running time of $2^{(c+1)/M}.T_{LLL}((1/M)\log N, (r+c)\log N)$. The algorithm is deterministic, and runs in polynomial space.*

Proof. This follows from Theorem 10.1 with the observation that

$$\left(\frac{c+1}{r+c}\right)\log p = \frac{c+1}{M} - \frac{(c+1)c}{M\log p + c} \leq \frac{c+1}{M},$$

and $r^2 = (1/M)\log N$.

12. Experiments

We implemented the lattice factoring method using Maple version 5.0 and Victor Shoup's Number Theory Library package [47]. 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 9. Using *NTL*'s implementation of *LLL*, it reduces the lattice from Section 9, 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.

We tested MSBFact, LSBFact, and the algorithm of Theorem 10.2. The algorithm of Theorem 10.1 uses Theorem 10.2 and simple exhaustive search. Examples of running times for LSBFact are given in Section 12. Running times for MSBFact are similar. Here we restrict attention to the core algorithm given in Theorem 10.2. Example running times of this algorithm are listed in Figure 12.1. To extend this to the full version (Theorem 10.1) would require exhaustive search for the "bits given".

P	***N***	***r***	**Bits**	**Lattice**	**Running**
(bits)	**(bits)**		**given**	**dim.**	**time**
64	576	8	16	49	14 minutes
80	1280	15	20	72	5.2 hours
96	768	7	22	60	1.6 hours
96	960	9	22	65	3.2 hours
100	600	5	23	69	1.7 hours

Experiments performed on a 1GHz Intel Pentium III running Linux.

Figure 12.1. Running times for LFM.

This introduces a large multiplicative factor in the running times listed above. The resulting running times 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.

Optimizations. The execution times of the algorithms presented in this sections are dominated by the running time of the *LLL* algorithm. In this section, we address several practical concerns that greatly improve the performance of this step.

The first observation is that in our experiments, the short vector returned by the *LLL* algorithm almost always corresponds to a polynomial of degree $w-1$. This means that a linear combination which yields a short vector will include those basis vectors corresponding to the $g_{i,k}$ and $g_{j,k}$ of greatest degree. We focus attention of the *LLL* algorithm on these basis vectors by using the following ordering on the basis vectors:

- $g_{j,m}(xX)$ for $j = w - mr - 1, \ldots, 0$ followed by
- $g_{i,k}(xX)$ for $k = m, m-1, \ldots, 0$ and $i = 0, \ldots, r-1$.

This resulted in a speedup of over factor of two compared to the natural ordering, in which *LLL* spent a large amount of time reducing a portion of the basis that would ultimately be irrelevant.

The second observation is that in an *LLL*-reduced lattice, the worst-case result for the shortest vector will be

$$T_\alpha(p) = 3xp((\log p)^\alpha).$$

Implementations of *LLL* often try to improve this by reducing the "fudge factor" of $2^{(w-1)/4}$. However, as the analysis from Subsection 4.1 shows, the final contribution of this term is negligible. Thus a high-quality basis reduction is unnecessary, and running times can be greatly improved by deactivating features such as Block Korkin-Zolotareff reduction.

13. Comparison to Other Factoring Methods

We restate the Theorem 10.1 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

$$T_{\alpha}(p) = \exp((\log p)^{\alpha}).$$

This function is analogous to the $L_{\alpha,\beta}(p)$ function commonly used to describe the running time of factoring algorithms [5]. Recall that

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

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 10.1.

Corollary 13.1. *Let $N = p^r q$ be given where p and q are both k bit integers. Suppose $r = (\log p)^{\varepsilon}$ for some ε. Then given N and r, a non-trivial integer factor of N can be found in time*

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

where 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 behaviours as *r* goes to infinity. We write $r = (\log p)^{\varepsilon}$. The standard running times [15, 37] of several algorithms are summarized in the following table, ignoring polynomial factors.

Method	**Asymptotic running time**
Lattice factoring method	$\exp((\log p)^{1-\varepsilon})$
Elliptic curve method	$\exp(1.414 \cdot (\log p)^{1/2}(\log\log p)^{1/2})$
Number field sieve	$\exp(1.902 \cdot (\log N)^{1/2}(\log\log N)^{2/3})$

Since $N = p^r q$ and $r = k^{\varepsilon}$, we know that

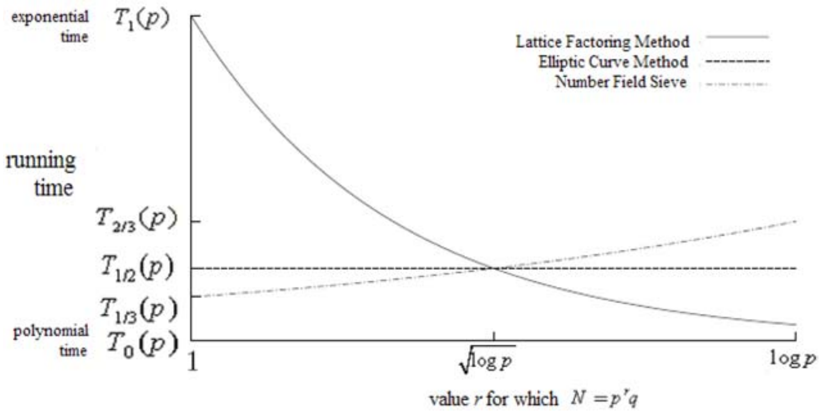
$$\log N = r\log p + \log q \geq rk = k^{1+\varepsilon}.$$

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

Method	**Asymptotic running time**
Lattice factoring method	$\exp(k^{1-\varepsilon}) = T_{1-\varepsilon}(p)$
Elliptic curve method	$\exp(1.414 \cdot k^{1/2}(\log p)^{1/2}) > (T_{1/2}(p))^{1.414}$
Number field sieve	$\exp(1.902 \cdot k^{(1+\varepsilon)/3}((1+\varepsilon)^{2/3}) > (T_{(1+\varepsilon).3}(p))^{1.902}$

We are particularly interested in the exponential component of the running times, which is tracked in Figure 13.1. Notice that when $\varepsilon = \frac{1}{2}$, then all three algorithms run in time close to $T_{1/2}(p)$.

Practical comparison to ECM. Of particular interest in Figure 13.1 is the point at $r = \sqrt{\log p}$ (i.e., $\varepsilon = \frac{1}{2}$), where ECM, LFM, and NFS have similar asymptotic running times. We refer the reader to



Comparison of subexponential running times of current factoring methods as a function of *r*. Both axes are logarithmic, and polynomial time factors are suppressed.

Figure 13.1. Asymptotic comparison of LFM with ECM and NFS.

Figure 12.1 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. Below, we reproduce a table of some example running times [52, 48] for factorizations performed by ECM.

size of 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 seconds
128 bits	231 hours	$r = 10$: 7000 r 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 $r \approx 20.$

14. 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 $r = \varepsilon \log p$ for a fixed constant $\varepsilon > 0$ the integer N can be factored in polynomial time. Hence, if p and q are k bit primes, the integer $N = p^k q$ can be factored by a polynomial time algorithm. Even when $r \approx \sqrt{\log p}$ such integers can be factored in time that is asymptotically faster than the best current methods. For much smaller r, our algorithm provides an efficient method for factoring N provided a “hint” of appropriate quality is available.

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 the univariate root-finding technique. 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.

Acknowledgements

The authors are grateful to the referees and the editors for the careful reading of the manuscript. The remarks and the corrections motivated the authors to make some valuable improvements.

References

[1] M. Bellare and P. Rogaway, Optimal Asymmetric Encryption, In Proceedings Eurocrypt 94, Lecture Notes in Computer Science, Vol. 950, Springer-Verlag, pp. 92-111, 1994.

[2] D. Bleichenbacher, Chosen Ciphertext Attacks Against Protocols based on the RSA Encryption Standard PKCS #1, In Proceedings Crypto 98, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 1-12, 1998.

[3] D. Bleichenbacher, On the Security of the KMOV Public Key Cryptosystem, In Proceedings Crypto 97, Lecture Notes in Computer Science, Vol. 1294, Springer-Verlag, pp. 235-248, 1997.

[4] D. Boneh, Finding Smooth Integers using CRT Decoding, In Proceedings STOC 2000, pp. 265-272, Portland, Oregon, 2000.

[5] D. Boneh, Simplified OAEP for the RSA and Rabin Functions, In Proceedings Crypto 2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 275-291, 2001.

[6] D. Boneh, Twenty years of attacks on the RSA cryptosystem, Notices of the AMS 46(2) (1999), 203-213.

[7] D. Boneh, R. DeMillo and R. Lipton, On the Importance of Checking Cryptographic Protocols for Faults, In Proceedings Eurocrypt 97, Lecture Notes in Computer Science, Vol. 1233, Springer-Verlag, pp. 37-51, 1997. 90 BIBLIOGRAPHY91.

[8] D. Boneh and G. Durfee, Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$, In Proceedings Eurocrypt 99, Lecture Notes in Computer Science, Vol. 1592, Springer-Verlag, pp. 1-11, 1999.

[9] D. Boneh and G. Durfee, Cryptanalysis of RSA with private key d less than $N^{0.292}$, IEEE Transactions on Information Theory 46(4) (2000), 1339-1349.

[10] D. Boneh, G. Durfee and Y. Frankel, An Attack on RSA Given a Small Fraction of the Private Key Bits, In Proceedings Asiacrypt 98, Lecture Notes in Computer Science, Vol. 1514, Springer-Verlag, pp. 25-34, 1998.

[11] D. Boneh, G. Durfee and N. Howgrave-Graham, Factoring $N = p^r q$ for Larger, In Proceedings Crypto 99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 326-337, 1999.

[12] D. Boneh and R. Venkatesan, Breaking RSA may not be Equivalent to Factoring, In Proceedings Eurocrypt 98, Lecture Notes in Computer Science, Vol. 1403, Springer-Verlag, pp. 59-71, 1998.

[13] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz and A. Sahai, Exposure-Resilient Functions and All-or-Nothing Transforms, In Proceedings Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, pp. 453-469, 2000.

[14] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam and P. Zimmermann, Factorization of 512-bit RSA Key using the Number Field Sieve, In Proceedings Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, 2000, Factorization announced in August, 1999.

[15] D. Coppersmith, Modifications to the number field sieve, Journal of Cryptology 6 (1993), 169-180.

[16] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerabilities, Journal of Cryptology 10 (1997), 233-260.

[17] D. Coppersmith, Finding Small Solutions to Small Degree Polynomials, In Proceedings Cryptography and Lattice Conference, Lecture Notes in Computer Science, Vol. 2146, Springer-Verlag, 2001.

[18] D. Coppersmith, M. Franklin, J. Patarin and M. Reiter, Low Exponent RSA with Related Messages, In Proceedings Eurocrypt 96, Lecture Notes in Computer Science, Vol. 1070, Springer-Verlag, pp. 1-9, 1996.

[19] D. Coppersmith, S. Halevi and C. S. Jutla, ISO 9796 and the New Forgery Strategy, Presented at Rump Session of Crypto 99, 1999.

[20] S. Coron, D. Naccache and J. P. Stern, On the Security of RSA Padding, In Proceedings Crypto 99, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, pp. 1-18, 1999.

[21] R. Cramer and V. Shoup, A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack, Advances in Cryptology-Crypto 98, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 13-25, 1998.

[22] G. Durfee and P. Nguyen, Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt 99, In Proceedings Asiacrypt 2000, Lecture Notes in Computer Science, Vol. 1976, Springer-Verlag, pp. 14-29, 2000.

[23] A. Fujioke, T. Okamoto and S. Miyaguchi, ESIGN: An Efficient Digital Signature Implementation for Smartcards, In Proceedings Eurocrypt 91, Lecture Notes in Computer Science, Vol. 547, Springer-Verlag, pp. 446-457, 1991.

[24] C. F. Gauss, Disquisitiones Arithmeticae, Leipzig, 1801.

[25] O. Goldreich, Foundations of Cryptography – Fragments of a Book.

[26] D. Gordon, Discrete Logarithms in GF(p) using the number field sieve, SIAM J. Discrete Math. 6 (1993), 124-138.

[27] G. Hardy and E. Wright, An Introduction to the Theory of Numbers, Fourth Edition, Oxford Clarendon Press, 1975.

[28] J. Håstad, Solving simultaneous modular equations of low degree, SIAM Journal on Computing 17(2) (1998), 336-341.

[29] C. Hermite, Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres, deuxième letter, J. Reine Agnew., Math. 40 (1850), 279-290.

[30] N. Howgrave-Graham, Computational Mathematics Inspired by RSA, Ph.D. Thesis, University of Bath, 1999.

[31] N. Howgrave-Graham, Extending *LLL* to Gaussian integers, Unpublished Manuscript, March 1998.

http://www.bath.ac.uk/~mapnahg/pub/gauss.ps

[32] N. Howgrave-Graham, Finding Small Roots of Univariate Modular Equations Revisited, In Proceedings Cryptography and Coding, Lecture Notes in Computer Science, Vol. 1355, Springer-Verlag, pp. 131-142, 1997.

[33] C. Jutla, On Finding Small Solutions of Modular Multivariate Polynomial Equations, In Proceedings Eurocrypt 98, Lecture Notes in Computer Science, Vol. 1403, Springer-Verlag, pp. 158-170, 1998.

[34] P. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems, In Proceedings Crypto' 96, Lecture Notes in Computer Science, Vol. 1109, Springer-Verlag, pp. 104-113.

[35] A. Korkine and G. Zolotareff, Sur les formes quadratiques, Math. Ann. 6 (1873), 336-389.

[36] L. Lagrange, Recherches d'arithmétique, Mouv. Mém. Acad., 1773.

[37] A. Lenstra and H. W. Lenstra Jr., Algorithms in Number Theory, In Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity), Chapter 12, pp. 673-715, 1990.

[38] H. W. Lenstra Jr., Factoring integers with elliptic curves, Annuals of Mathematics 126 (1987), 649-673.

[39] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[40] H. Minkowski, Geometrie der Zahlen, Teubner-Verlag, Leipzig, 1896.

[41] P. Nguyen and J. Stern, Lattice Reduction in Cryptology: An Update, In Algorithmic Number Theory – Proceedings of ANTS IV, Lecture Notes in Computer Science, Vol. 1838, Springer-Verlag, 2000.

[42] P. Nguyen and J. Stern, The Two Faces of Lattices in Cryptology, In Proceedings Cryptography and Lattices Conference, Lecture Notes in Computer Science, Vol. 2146, Springer-Verlag, 2001.

[43] I. Niven, H. Zuckerman and H. Montgomery, An Introduction to the Theory of Numbers, Jon Wiley & Sons, Fifth Edition, pp. 87-88, 1991.

[44] T. Okamoto and S. Uchiyama, A New Public Key Cryptosystem as Secure as Factoring, In Proceedings Eurocrypt 98, Lecture Notes in Computer Science, Vol. 1403, Springer-Verlag, pp. 310-318, 1998.

[45] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Second Edition, Cambridge University Press, pp. 347-393, 1997.

[46] R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21(2) (1978), 120-126.

[47] V. Shoup, Number Theory Library (NTL).

http://www.shoup.net/ntl/

[48] R. Silverman and S. Wagstaff, A practical analysis of the elliptic curve factoring algorithm, Mathematics of Computation 61 (1993).

[49] D. Stinson, Cryptography: Theory and Practice, CRC Press, 1994.

[50] H. Sun, W. Yang and C. Laih, On the Design of RSA with Short Secret Exponent, In Proceedings Asiacrypt 99, Lecture Notes in Computer Science, Vol. 1716, Springer-Verlag, pp. 150-164, 1999.

[51] T. Takagi, Fast RSA-Type Cryptosystem Modulo $p^k q$, In Proceedings Crypto 98, Lecture Notes in Computer Science, Vol. 1462, Springer-Verlag, pp. 318-326, 1998.

[52] E. Verheul and H. van Tilborg, Cryptanalysis of less short RSA secret exponents, Applicable Algebra in Engineering, Communication, and Computing 8 (1997), 425-435.

[53] M. Wiener, Cryptanalysis of short RSA secret exponents, IEEE Transactions on Information Theory 36(3) (1990), 553-558.

■