Finding Small Roots of Bivariate Integer Polynomial Equations Revisited

Jean-Sébastien Coron

Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
jean-sebastien.coron@gemplus.com

Abstract. At Eurocrypt '96, Coppersmith proposed an algorithm for finding small roots of bivariate integer polynomial equations, based on lattice reduction techniques. But the approach is difficult to understand. In this paper, we present a much simpler algorithm for solving the same problem. Our simplification is analogous to the simplification brought by Howgrave-Graham to Coppersmith's algorithm for finding small roots of univariate modular polynomial equations. As an application, we illustrate the new algorithm with the problem of finding the factors of $n = pq$ if we are given the high order $1/4 \log_2 n$ bits of p.

1 Introduction

An important application of lattice reduction found by Coppersmith in 1996 is finding small roots of low-degree polynomial equations [3–5]. This includes modular univariate polynomial equations, and bivariate integer equations.

The problem of solving univariate polynomial equations modulo an integer N of unknown factorization seems to be hard, as for some polynomials it is equivalent to the knowledge of the factorization of N. Moreover, the problem of inverting RSA, *i.e.* extracting e-th root modulo N, is a particular case of this problem. However, at Eurocrypt '96, Coppersmith showed that the problem of finding *small* roots is easy [3, 5], using the LLL lattice reduction algorithm [9]:

Theorem 1 (Coppersmith). *Given a monic polynomial $P(x)$ of degree δ, modulo an integer N of unknown factorization, one can find in time polynomial in $(\log N, 2^\delta)$ all integers x_0 such that $P(x_0) = 0 \mod N$ and $|x_0| \le N^{1/\delta}$.*

The algorithm can be extended to handle multivariate modular polynomial equations, but the extension is heuristic only. Coppersmith's algorithm has many applications in cryptography: cryptanalysis of RSA with small public exponent when some part of the message is known [5], cryptanalysis of RSA with private exponent d smaller than $N^{0.29}$ [1], polynomial-time factorization of $N = p^r q$ for large r [2], and even an improved security proof for OAEP with small public exponent [13] (see [12] for a nice survey).

Coppersmith's algorithm for solving univariate modular polynomial equations was further simplified by Howgrave-Graham in [7]. Apart from being simpler to understand and implement, a significant advantage of Howgrave-Graham's approach is the heuristic extension to multivariate modular polynomial: indeed, depending on the shape of the polynomial, there is much flexibility in selecting the parameters of the algorithm, and Howgrave-Graham's approach enables to easily derive the corresponding bound for the roots. This approach is actually used in all previously cited variants of Coppersmith's technique [1, 2].

Similarly, the problem of solving bivariate integer polynomial equations seems to be hard. Letting $p(x, y)$ be a polynomial in two variables with integer coefficients,

$$p(x, y) = \sum_{i,j} p_{i,j} \cdot x^i y^j$$

it consists in finding all integer pairs (x_0, y_0) such that $p(x_0, y_0) = 0$. We see that integer factorization is a special case as one can take $p(x, y) = N - x \cdot y$. However, at Eurocrypt '96, Coppersmith showed [4, 5] that using LLL, the problem of finding *small* roots of bivariate polynomial equations is easy:

Theorem 2. *Let $p(x, y)$ be an irreducible polynomial in two variables over $\mathbb{Z}$, of maximum degree δ in each variable separately. Let X and Y be upper bounds on the desired integer solution (x_0, y_0), and let $W = \max_{i,j} |p_{ij}| X^i Y^j$. If $XY < W^{2/(3\delta)}$, then in time polynomial in $(\log W, 2^\delta)$, one can find all integer pairs (x_0, y_0) such that $p(x_0, y_0) = 0$, $|x_0| \leq X$, and $|y_0| \leq Y$.*

Moreover, there can be improved bounds depending on the shape of the polynomial $p(x, y)$. For example, for a polynomial $p(x, y)$ of total degree δ in x and y, the bound is $XY < W^{1/\delta}$. As for the univariate modular case, the technique can be heuristically extended to more than two variables. An application of Coppersmith's algorithm for the bivariate integer polynomial case is to factor in polynomial-time an RSA-modulus $n = pq$ such that half of the least significant or most significants bits of p are known [5].

However, as noted in [6], the approach for the bivariate integer case is rather difficult to understand. This means that the algorithm is difficult to implement in practice, and that improved bounds depending on the shape of the polynomial are more difficult to derive. In particular, what makes the analysis harder is that one has to derive the determinant of lattices which are not full rank. The particular case of factoring $n = pq$ when half of the least significant or most significants bits of p are known, was further simplified by Howgrave-Graham in [7], but as noted in [6], this particular simplification does not seem to extend to the general case of bivariate polynomial equations. As suggested in [12], a simplification analogue to what has been obtained by Howgrave-Graham for the univariate modular case would be useful.

In this paper, we present a simple and efficient algorithm for finding small roots of bivariate integer polynomials. Our simplification is analogous to the simplification obtained by Howgrave-Graham for the univariate modular case. We apply lattice reduction to a full rank lattice that admits a natural triangular basis. It is then straightforward to derive the determinant and improved bounds depending on the shape of the polynomial; the heuristic extension to more than two variables is also simpler. However, our algorithm is slightly less efficient than Coppersmith's algorithm, because our algorithm has a polynomial-time complexity only if $XY < W^{2/3\delta-\varepsilon}$ for any fixed $\varepsilon > 0$, whereas Coppersmith's algorithm requires $XY < W^{2/3\delta}$, a slightly weaker condition. In section 7, we illustrate our algorithm with the problem of finding the factors of $n = pq$ if we are given the high order $1/4 \log_2 n$ bits of p, and show that our algorithm is rather efficient in practice.

2 Solving Bivariate Integer Equations: an Illustration

In this section, we first illustrate our technique with a bivariate integer polynomial of the form

$$p(x, y) = a + bx + cy + dxy,$$

with $a \neq 0$ and $d \neq 0$. We assume that $p(x, y)$ is irreducible and has a small root (x_0, y_0). Our goal is to recover (x_0, y_0). As in theorem 2, we let X, Y be some bound on x_0, y_0, that is we have $|x_0| \leq X$ and $|y_0| \leq Y$, and let $W = \max\{|a|, |b|X, |c|Y, |d|XY\}$. Moreover, given a polynomial $h(x, y) = \sum_{i,j} h_{ij} x^i y^j$, we define $\|h(x, y)\|^2 := \sum_{i,j} |h_{ij}|^2$ and $\|h(x, y)\|_\infty := \max_{i,j} |h_{ij}|$. Note that we have:

$$W = \|p(xX, yY)\|_\infty \tag{1}$$

First, we generate an integer n such that :

$$W \leq n < 2 \cdot W \tag{2}$$

and $\gcd(n, a) = 1$. One can take $n = W + ((1 - W) \mod |a|)$. Then we define the polynomial:

$$\begin{aligned} q_{00}(x, y) &= a^{-1} p(x, y) \mod n \\ &= 1 + b'x + c'y + d'xy \end{aligned}$$

We also consider the polynomials $q_{10}(x, y) = nx$, $q_{01}(x, y) = ny$ and $q_{11}(x, y) = nxy$. Note that for all four polynomials $q_{ij}(x, y)$, we have that $q_{ij}(x_0, y_0) = 0 \mod n$.

We consider the four polynomials $\tilde{q}_{ij}(x, y) = q_{ij}(xX, yY)$; we are interested in finding a small linear integer combination of the polynomials $\tilde{q}_{ij}(x, y)$. Therefore, we consider the lattice generated by all linear integer combinations of the coefficient vectors of the $\tilde{q}_{ij}(x, y)$. A basis of the lattice is given by the following matrix L of row vectors:

$$L = \begin{bmatrix} 1 & b'X & c'Y & d'XY \\ & nX & & \\ & & nY & \\ & & & nXY \end{bmatrix}$$

We know that the LLL algorithm [9], given a lattice spanned by $(u_1, \ldots, u_\omega)$, finds in polynomial time a lattice vector b_1 such that $\|b_1\| \leq 2^{(\omega-1)/4} \det(L)^{1/\omega}$.

More background on lattice reduction techniques will be given in the next section. With $\omega = 4$ and $\det L = n^3 (XY)^2$ we obtain in polynomial time a non-zero polynomial $h(x,y)$ such that

$$\|h(xX, yY)\| \leq 2 \cdot n^{3/4} (XY)^{1/2} \tag{3}$$

Note that we have $h(x_0, y_0) = 0 \mod n$. The following lemma, due to Howgrave-Graham, shows that if the coefficients of $h(x,y)$ are sufficiently small, then the equality $h(x_0, y_0) = 0$ holds not only modulo n, but also over $\mathbb{Z}$.

Lemma 1 (Howgrave-Graham). *Let $h(x,y) \in \mathbb{Z}[x,y]$ which is a sum of at most ω monomials. Suppose that $h(x_0, y_0) = 0 \mod n$ where $|x_0| \leq X$ and $|y_0| \leq Y$ and $\|h(xX, yY)\| < n/\sqrt{\omega}$. Then $h(x_0, y_0) = 0$ holds over the integers.*

Proof. We have:

$$\begin{aligned}
|h(x_0, y_0)| &= \left| \sum h_{ij} x_0^i y_0^i \right| = \left| \sum h_{ij} X^i Y^j \left(\frac{x_0}{X}\right)^i \left(\frac{y_0}{Y}\right)^j \right| \\
&\leq \sum \left| h_{ij} X^i Y^j \left(\frac{x_0}{X}\right)^i \left(\frac{y_0}{Y}\right)^j \right| \leq \sum \left| h_{ij} X^i Y^j \right| \\
&\leq \sqrt{\omega} \|h(xX, yY)\| < n
\end{aligned}$$

Since $h(x_0, y_0) = 0 \mod n$, this gives $h(x_0, y_0) = 0$. $\square$

Assume now that:

$$XY < n^{1/2}/16 \tag{4}$$

Then inequality (3) gives:

$$\|h(xX, yY)\| < n/2 \tag{5}$$

which implies that $h(x_0, y_0) = 0$. Moreover, from (1), (2) and (5) we get:

$$\|h(xX, yY)\| < n/2 < W \leq \|p(xX, yY)\|_\infty \leq \|p(xX, yY)\|$$

This shows that $h(x,y)$ cannot be a multiple of $p(x,y)$. Namely, if $h(x,y)$ is a multiple of $p(x,y)$, then it follows from the definition of p and h that we must have $h(x,y) = \lambda \cdot p(x,y)$ with $\lambda \in \mathbb{Z}^*$. This would give $\|h(xX, yY)\| = |\lambda| \cdot \|p(xX, yY)\| \geq \|p(xX, yY)\|$, a contradiction.

Eventually, since $p(x,y)$ is irreducible and $h(x,y)$ is not a multiple of $p(x,y)$,

$$Q(x) = \text{Resultant}_y(h(x,y), p(x,y))$$

gives a non-zero integer polynomial such that $Q(x_0) = 0$. Using any standard root-finding algorithm, we can recover x_0, and finally y_0 by solving $p(x_0, y) = 0$. Using inequality (4) and $n \geq W$, this shows that if :

$$XY < \frac{W^{1/2}}{16}$$

one can find in time polynomial in $\log W$ all integer pairs (x_0, y_0) such that $p(x_0, y_0) = 0$, $|x_0| \leq X$, and $|y_0| \leq Y$.

This bound is weaker than the bound $XY < W^{2/3}$ given by theorem 2 for $\delta = 1$. We will see in section 4 that by adding more multiples of $p(x, y)$ into the lattice, we recover the desired bound.

3 Background on Lattices and Polynomials

3.1 The LLL Algorithm

Let $u_1, \ldots, u_\omega \in \mathbb{Z}^n$ be linearly independent vectors with $\omega \leq n$. The lattice L spanned by $< u_1, \ldots, u_\omega >$ consists of all integral linear combinations of $u_1, \ldots, u_\omega$, that is:

$$L = \{ \sum_{i=1}^{\omega} n_i \cdot u_i | \; n_i \in \mathbb{Z} \}$$

Such a set of vectors u_i's is called a lattice *basis*. All the bases have the same number of elements, called the *dimension* or *rank* of the lattice. We say that the lattice is full rank if $\omega = n$. Any two bases of the same lattice L are related by some integral matrix of determinant ± 1. Therefore, all the bases have the same Gramian determinant $\det_{1 \leq i,j \leq d} < u_i, u_j >$. One defines the *determinant* of the lattice as the square root of the Gramian determinant. If the lattice is full rank, then the determinant of L is equal to the absolute value of the determinant of the $\omega \times \omega$ matrix whose rows are the basis vectors $u_1, \ldots, u_\omega$.

The LLL algorithm [9] computes a short vector in a lattice :

Theorem 3 (LLL). *Let L be a lattice spanned by $(u_1, \ldots, u_\omega)$. The LLL algorithm, given $(u_1, \ldots, u_\omega)$, finds in polynomial time a vector b_1 such that:*

$$||b_1|| \leq 2^{(\omega-1)/4} \det(L)^{1/\omega}$$

3.2 Bound on the Factors of Polynomials

We use the following notation: given a polynomial $h(x) = \sum_i h_i x^i$, we define $||h||^2 := \sum_i |h_i|^2$ and $||h||_\infty := \max_i |h_i|$. We use the same notations for bivariate polynomials, as defined in section 2. The following two lemmata will be useful in the next section:

Lemma 2. *Let $a(x, y)$ and $b(x, y)$ be two non-zero polynomials over $\mathbb{Z}$ of maximum degree d separately in x and y, such that $b(x, y)$ is a multiple of $a(x, y)$ in $\mathbb{Z}[x, y]$. Then:*

$$||b|| \geq 2^{-(d+1)^2} \cdot ||a||_\infty$$

Proof. The proof is based on the following result of Mignotte [11]: let $f(x)$ and $g(x)$ be two non-zero polynomials over the integers, such that $\deg f \leq k$ and f divides g in $\mathbb{Z}[X]$; then :

$$\|g\| \geq 2^{-k} \cdot \|f\|_\infty$$

Let $f(x) = a(x, x^{d+1})$. Then we have $\deg f \leq (d+1)^2$ and the polynomials $a(x,y)$ and $f(x)$ have the same list of non-zero coefficients, which gives $\|f\|_\infty = \|a\|_\infty$. Similarly, letting $g(x) = b(x, x^{d+1})$, we have $\|g\| = \|b\|$. Moreover $f(x)$ divides $g(x)$ in $\mathbb{Z}[x]$. Using the previous result of Mignotte, this proves lemma 2. □

Lemma 3. *Let $a(x,y)$ and $b(x,y)$ be as in lemma 2. Assume that $a(0,0) \neq 0$ and $b(x,y)$ is divisible by a non-zero integer r such that $\gcd(r, a(0,0)) = 1$. Then $b(x,y)$ is divisible by $r \cdot a(x,y)$ and:*

$$\|b\| \geq 2^{-(d+1)^2} \cdot |r| \cdot \|a\|_\infty$$

Proof. Let $\lambda(x,y)$ be the polynomial such that $a(x,y) \cdot \lambda(x,y) = b(x,y)$. We show that r divides $\lambda(x,y)$. Assume that this is not the case, and let λ_{ij} be a coefficient of $x^i y^j$ in $\lambda(x,y)$ not divisible by r. Take the smallest (i,j) for the lexicographic ordering. Then we have that $b_{ij} = \lambda_{ij} \cdot a(0,0) \mod r$, where b_{ij} is the coefficient of $x^i y^j$ in $b(x,y)$. Since $a(0,0)$ is invertible modulo r and $b_{ij} = 0 \mod r$, this gives a contradiction. This shows that $r \cdot a(x,y)$ divides $b(x,y)$. Applying the previous lemma to $r \cdot a(x,y)$ and $b(x,y)$, this terminates the proof. □

4 Finding Small Roots of Bivariate Integer Polynomials

We prove the following theorem:

Theorem 4. *Let $p(x,y)$ be an irreducible polynomial in two variables over $\mathbb{Z}$, of maximum degree δ in each variable separately. Let X and Y be upper bounds on the desired integer solution (x_0, y_0), and let $W = \max_{i,j} |p_{ij}| X^i Y^j$. If for some $\varepsilon > 0$,*

$$XY < W^{2/(3\delta)-\varepsilon} \tag{6}$$

then in time polynomial in $(\log W, 2^\delta)$, one can find all integer pairs (x_0, y_0) such that $p(x_0, y_0) = 0$, $|x_0| \leq X$, and $|y_0| \leq Y$.

Proof. We write:

$$p(x,y) = \sum_{0 \leq i,j \leq \delta} p_{ij} x^i y^j$$

and let (x_0, y_0) be an integer root of $p(x,y)$. As previously we let

$$W = \|p(xX, yY)\|_\infty$$

First we assume that $p_{00} \neq 0$ and $\gcd(p_{00}, XY) = 1$. We will see in appendix A how to handle the general case.

We select an integer $k \geq 0$ and let $\omega = (\delta + k + 1)^2$. We generate an integer u such that $\sqrt{\omega} \cdot 2^{-\omega} \cdot W \leq u < 2W$ and $\gcd(p_{00}, u) = 1$. As in section 2, one can take

$$u = W + ((1 - W) \mod |p_{00}|) .$$

We let $n = u \cdot (XY)^k$. We have that $\gcd(p_{00}, n) = 1$ and:

$$\sqrt{\omega} \cdot 2^{-\omega} \cdot (XY)^k \cdot W \leq n < 2 \cdot (XY)^k \cdot W \tag{7}$$

As in section 2, we must find a polynomial $h(x, y)$ such that $h(x_0, y_0) = 0$ and $h(x, y)$ is not a multiple of $p(x, y)$. We let $q(x, y)$ be the polynomial:

$$\begin{aligned} q(x, y) &= p_{00}^{-1} \cdot p(x, y) \mod n \\ &= 1 + \sum_{(i,j) \neq (0,0)} a_{ij} x^i y^j \end{aligned}$$

For all $0 \leq i, j \leq k$, we form the polynomials:

$$q_{ij}(x, y) = x^i y^j X^{k-i} Y^{k-j} q(x, y)$$

For all $(i, j) \in [0, \delta + k]^2 \setminus [0, k]^2$, we also form the polynomials:

$$q_{ij}(x, y) = x^i y^j n$$

We consider the corresponding polynomials $\tilde{q}_{ij}(x, y) = q_{ij}(xX, yY)$. Note that for all $(i, j) \in [0, \delta + k]^2$, we have that $q_{ij}(x_0, y_0) = 0 \mod n$, and the polynomial $\tilde{q}_{ij}(x, y)$ is divisible by $(XY)^k$.

Let $h(x, y)$ be a linear integer combination of the polynomials $q_{ij}(x, y)$; the polynomial $\tilde{h}(x, y) = h(xX, yX)$ is also a linear combination of the $\tilde{q}_{ij}(x, y)$ with the same integer coefficients. We have that $h(x_0, y_0) = 0 \mod n$ and $(XY)^k$ divides $h(xX, yY)$. Moreover $h(x, y)$ has maximum degree $\delta + k$ independently in x and y, therefore it is the sum of at most ω monomials. As in section 2, we are interested in finding a polynomial $h(x, y)$ such that the coefficients of $h(xX, yY)$ are small enough, for the following two reasons:

1) if the coefficients of $h(xX, yY)$ are sufficiently small, then the equality $h(x_0, y_0) = 0$ holds not only modulo n, but also over $\mathbb{Z}$. From lemma 1, the condition is:

$$\|h(xX, yY)\| < \frac{n}{\sqrt{\omega}} \tag{8}$$

2) if the coefficients of $h(xX, yY)$ are sufficiently small, then $h(x, y)$ cannot be a multiple of $p(x, y)$. Using lemma 3, the condition is:

$$\|h(xX, yY)\| < 2^{-\omega} \cdot (XY)^k \cdot W \tag{9}$$

This condition is obtained by applying lemma 3 with $a(x, y) = p(xX, yY)$, $b(x, y) = h(xX, yY)$ and $r = (XY)^k$. Then we have $a(0, 0) = p_{00} \neq 0$ and

$\gcd(a(0,0),(XY)^k) = 1$. Under condition (9), $h(xX, yY)$ cannot be a multiple of $p(xX, yY)$ and therefore $h(x,y)$ cannot be a multiple of $p(x,y)$.

Using inequality (7), we obtain that the first condition (8) is satisfied whenever the second condition (9) is satisfied.

We form the lattice L spanned by the coefficients of the polynomials $\tilde{q}_{ij}(x,y)$. The polynomials $q_{ij}(x,y)$ have a maximum degree of $\delta + k$ separately in x and y; therefore, there are $(\delta + k + 1)^2$ such coefficients. Moreover, there is a total of $(\delta+k+1)^2$ polynomials. This gives a full rank lattice of dimension $\omega = (\delta+k+1)^2$. In figure 1, we illustrate the lattice for $\delta = 1$ and $k = 1$.

	1	x	y	xy	x^2	x^2y	y^2	xy^2	x^2y^2
XYq	XY	$a_{10}X^2Y$	$a_{01}XY^2$	$a_{11}X^2Y^2$					
Yxq		XY		$a_{01}XY^2$	$a_{10}X^2Y$	$a_{11}X^2Y^2$			
Xyq			XY	$a_{10}X^2Y$			$a_{01}XY^2$	$a_{11}X^2Y^2$	
xyq				XY		$a_{10}X^2Y$		$a_{01}XY^2$	$a_{11}X^2Y^2$
x^2n					X^2n				
x^2yn						X^2Yn			
y^2n							Y^2n		
xy^2n								XY^2n	
x^2y^2n									X^2Y^2n

Fig. 1. The lattice L for $\delta = 1$ and $k = 1$

It is easy to see that the coefficient vectors of the polynomials $\tilde{q}_{ij}(x,y)$ form a triangular basis of L. The determinant is then the product of the diagonal entries. For $0 \le i,j \le k$, the contribution of the polynomials $\tilde{q}_{ij}(x,y)$ to the determinant is given by:

$$\prod_{0 \le i,j \le k} (XY)^k = (XY)^{k(k+1)^2}$$

The contribution of the other polynomials $\tilde{q}_{ij}(x,y)$ is then:

$$\prod_{(i,j)\in[0,\delta+k]^2 \setminus [0,k]^2} X^i Y^j n = (XY)^{\frac{(\delta+k)(\delta+k+1)^2}{2} - \frac{k(k+1)^2}{2}} n^{\delta(\delta+2k+2)}$$

Therefore, the determinant of L is given by:

$$\det(L) = (XY)^{\frac{(\delta+k)(\delta+k+1)^2 + k(k+1)^2}{2}} n^{\delta(\delta+2k+2)} \qquad (10)$$

Using LLL (see theorem 3), we obtain in time polynomial in $(\log W, \omega)$ a non-zero polynomial $h(x,y)$ such that:

$$\|h(xX, yY)\| \le 2^{(\omega-1)/4} \cdot \det(L)^{1/\omega} \qquad (11)$$

Note that any vector in the lattice L has integer coefficients divisible by $(XY)^k$; this means that in practice, it is more efficient to apply LLL to the lattice $(XY)^{-k}L$.

From inequality (11) we obtain that the conditions (8) and (9) are satisfied when:

$$2^{(\omega-1)/4} \cdot \det(L)^{1/\omega} < 2^{-\omega} \cdot (XY)^k \cdot W \tag{12}$$

In this case, we have that $h(x_0, y_0) = 0$ and $h(x, y)$ is not a multiple of $p(x, y)$. Since $p(x, y)$ is irreducible,

$$Q(x) = \text{Resultant}_y(h(x, y), p(x, y))$$

gives a non-zero integer polynomial such that $Q(x_0) = 0$. Using any standard root-finding algorithm, we can recover x_0, and finally y_0 by solving $p(x_0, y) = 0$.

Using inequality (7), we obtain that inequality (12) is satisfied when:

$$XY < 2^{-\beta} W^{\alpha} \tag{13}$$

where

$$\alpha = \frac{2(k+1)^2}{(\delta+k)(\delta+k+1)^2 - k(k+1)^2} \tag{14}$$

$$\beta = \frac{10}{4} \cdot \frac{(\delta+k+1)^4 + (\delta+k+1)^2}{(\delta+k)(\delta+k+1)^2 - k(k+1)^2} \tag{15}$$

We have that for all $\delta \geq 1$ and $k \geq 0$:

$$\alpha \geq \frac{2}{3\delta} - \frac{2}{3 \cdot (k+1)} \tag{16}$$

and:

$$\beta \leq \frac{4k^2}{\delta} + 13 \cdot \delta \tag{17}$$

Then, taking $k = \lfloor 1/\varepsilon \rfloor$, we obtain from (13), (16) and (17) the following condition for XY:

$$XY < W^{2/(3\delta)-\varepsilon} \cdot 2^{-4/(\delta \cdot \varepsilon^2) - 13\delta} \tag{18}$$

For an XY satisfying (18), we obtain a bivariate integer polynomial root-finding algorithm running in time polynomial in $(\log W, \delta, 1/\varepsilon)$.

For an XY satisfying the slightly weaker condition (6), we exhaustively search the high order $4/(\delta \cdot \varepsilon^2) + 13\delta$ bits of x_0, so that condition (18) applies, and for each possible value we use the algorithm described previously. For a fixed $\varepsilon > 0$, the running time is polynomial in $(\log W, 2^{\delta})$. This terminates the proof of theorem 4. □

As in [5], the efficiency of our algorithm depends on the shape of the polynomial $p(x, y)$. The previous theorem applies when $p(x, y)$ has maximum degree δ separately in x and y. If we assume that $p(x, y)$ has a total degree δ in x and y, we obtain the following theorem, analogous to theorem 3 in [5] (the proof is given in appendix B).

Theorem 5. *Under the hypothesis of theorem 4, except that $p(x, y)$ has total degree δ, the appropriate bound is:*

$$XY < W^{1/\delta - \varepsilon}$$

5 Comparison with Coppersmith's Algorithm

We note that under the following condition, stronger than (6) :

$$XY < W^{2/(3\delta)-\varepsilon} \cdot 2^{-13\delta}$$

Coppersmith's algorithm is polynomial-time in $(\log W, \delta, 1/\varepsilon)$ (see [5], theorem 2), whereas our algorithm is polynomial-time in $(\log W, \delta)$ but exponential-time in $1/\varepsilon$. Coppersmith's algorithm is therefore more efficient than ours for small values of ε. This implies that under the following condition, weaker than (6) :

$$XY < W^{2/(3\delta)}$$

Coppersmith's algorithm is still polynomial in $(\log W, 2^{\delta})$ (see [5], corollary 2), which is no longer the case for our algorithm.

6 Extension to More Variables

Our algorithm can be extended to solve integer polynomial equations with more than two variables. As for Coppersmith's algorithm, the extension is heuristic only.

Let $p(x, y, z)$ be a polynomial in three variables over the integers, of degree δ independently in x, y and z. Let (x_0, y_0, z_0) be an integer root of $p(x, y, z)$, with $|x_0| \leq X$, $|y_0| \leq Y$ and $|z_0| \leq Z$. Let ℓ be an integer ≥ 0. As for the bivariate case, we generate an integer n such that $n = 0 \mod (XYZ)^{\ell}$, and a polynomial $q(x, y, z)$ such that $q(x_0, y_0, z_0) = 0 \mod n$ and $q(0, 0, 0) = 1 \mod n$. Then we consider the lattice L generated by all linear integer combinations of the polynomials $x^i y^j z^k X^{\ell-i} Y^{\ell-j} Z^{\ell-k} q(xX, yY, zZ)$ for $0 \leq i, j, k \leq \ell$ and the polynomials $(xX)^i (yY)^j (zZ)^k \cdot n$ for $(i, j, k) \in [0, \delta + \ell]^3 \setminus [0, \ell]^3$. If the ranges X, Y, Z are small enough, then by using LLL we are guaranteed to find a polynomial $h_1(x, y, z)$ such that $h_1(x_0, y_0, z_0) = 0$ over $\mathbb{Z}$ and $h_1(x, y, z)$ is not a multiple of $p(x, y, z)$. Unfortunately, this is not enough. For small enough ranges X, Y, Z, we can also obtain a second polynomial $h_2(x, y, z)$ satisfying the same property. This can be done by bounding the norm of the second vector produced by LLL, as in [1, 8]. Then we could take the resultant between the three polynomials $p(x, y, z)$, $h_1(x, y, z)$ and $h_2(x, y, z)$ in order to obtain a polynomial $f(x)$ such that $f(x_0) = 0$. But we have no guarantee that the polynomials $h_1(x, y, z)$ and $h_2(x, y, z)$ will be algebraically independent, for example we might have $h_2(x, y, z) = x \cdot h_1(x, y, z)$. This makes the method heuristic only.

7 Practical experiments

An application of solving bivariate equations described in [5] is factoring an RSA modulus $n = pq$ when the high-order bits of p are known. Using our algorithm from theorem 4, we obtain the following theorem, whose proof is given in appendix C.

Theorem 6. *For any $\varepsilon > 0$, given $n = pq$ and the high-order $(1/4 + \varepsilon)\log_2 n$ bits of p, we can recover the factorization of n in time polynomial in $\log n$.*

By comparison, Coppersmith's algorithm provides a slightly better result since only the high-order $1/4 \log_2 n$ bits of p are required (see theorem 4, [5]). The result of practical experiments are summarized in table 2, using Shoup's NTL library [14]. It shows that our bivariate polynomial root-finding algorithm works well in practice.

N	bits of p given	lattice dimension	running time
512 bits	144 bits	25	35 sec
512 bits	141 bits	36	3 min
1024 bits	282 bits	36	20 min

Fig. 2. Running times for factoring $N = pq$ given the high-order bits of p, using our bivariate integer polynomial root finding algorithm on a 733 Mhz PC running under Linux.

We have also implemented the factorization of $n = pq$ with high-order bits known using the simplification of Howgrave-Graham [7]. Results are given in table 3. It shows that the simplification of Howgrave-Graham is much more efficient in practice. Namely, the factorization of a 1024-bit RSA modulus knowing the high-order 282 bits of p takes roughly 20 minutes using our bivariate polynomial root finding algorithm, and only one second using Howgrave-Graham's simplification. This is due to the fact that the Howgrave-Graham simplification enables to obtain a lattice with a lower dimension (but it applies only to the particular case of factoring with high-bits known, not to the general case of finding small roots of bivariate integer polynomials).

N	bits of p given	lattice dimension	running time
1024 bits	282 bits	11	1 sec
1024 bits	266 bits	25	1 min
1536 bits	396 bits	33	19 min

Fig. 3. Running times for factoring $N = pq$ given the high-order bits of p, using Howgrave-Graham's algorithm on a 733 Mhz PC running under Linux.

8 Conclusion

We have presented an algorithm for finding small roots of bivariate integer polynomials, simpler than Coppersmith's algorithm. The bivariate integer case is now as simple to analyze and implement as the univariate modular case. Our algorithm is asymptotically less efficient than Coppersmith's algorithm, but experiments show that it works well in practice; however, for the particular case of integer factorization with high-bits known, the Howgrave-Graham simplification appears to be more efficient.

References

1. D. Boneh and G. Durfee, *Crypanalysis of RSA with private key d less than $N^{0.292}$*, proceedings of Eurocrypt '99, vol. 1592, Lecture Notes in Computer Science.
2. D. Boneh, G. Durfee and N.A. Howgrave-Graham, *Factoring $n = p^r q$ for large r*, proceedings of Crypto '99, vol. 1666, Lecture Notes in Computer Science.
3. D. Coppersmith, *Finding a Small Root of a Univariate Modular Equation*, proceedings of Eurocrypt '96, vol. 1070, Lecture Notes in Computer Science.
4. D. Coppersmith, *Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known*, proceedings of Eurocrypt' 96, vol. 1070, Lecture Notes in Computer Science.
5. D. Coppersmith, *Small solutions to polynomial equations, and low exponent vulnerabilities.* J. of Cryptology, 10(4)233-260, 1997. Revised version of two articles of Eurocrypt '96.
6. D. Coppersmith, *Finding small solutions to small degree polynomials.* In Proc. of CALC '01, LNCS, Sptinger-Verlag, 2001.
7. N.A. Howgrave-Graham, *Finding small roots of univariate modular equations revisited.* In Cryptography and Coding, volume 1355 of LNCS, pp. 131-142. Springer Verlag, 1997.
8. C.S. Jutla, *On finding small solutions of modular multivariate polynomial equations.* Proceedings of Eurocrypt '98, Lecture Notes in Computer Science. vol. 1402.
9. A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients.* Mathematische Ann., 261:513-534, 1982.
10. U. Maurer, *Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters*, Journal of Cryptology, vol. 8, no. 3, pp. 123-155, 1995.
11. M. Mignotte, *An inequality about factors of polynomials.* Math Comp. 28, 1153-1157, 1974.
12. P.Q. Nguyen and J. Stern, *The two faces of lattices in cryptology.* Proceedings of CALC '01, LNCS vol. 2146.
13. V. Shoup, *OAEP reconsidered.* Proceedings of Crypto '01, vol. 2139, Lecture Notes in Computer Science.
14. V. Shoup, *Number Theory C++ Library (NTL) version 5.3.1.* Available at `www.shoup.net`.

A Finding Small Roots in the General Case

The algorithm described in section 4 assumes that $p(0,0) \neq 0$ and also that $\gcd(p(0,0), XY) = 1$. Here we show how to handle the general case.

If $p(0,0) = 0$, we use a simple change of variable to derive a polynomial $p^*(x,y)$ such that $p^*(0,0) \neq 0$. This is done as follows: we write p as $p(x,y) = x \cdot a_0(x) + y \cdot c(x,y)$, where a_0 is a polynomial of maximum degree $\delta - 1$. Since $p(x,y)$ is irreducible, we must have $a_0 \neq 0$. Since $\deg a_0 \leq \delta - 1$, there exists $0 < i \leq \delta$ such that $a_0(i) \neq 0$. Then $p(i,0) \neq 0$ and letting $p^*(x,y) = p(x+i,y)$, we obtain that $p^*(0,0) \neq 0$ and use $p^*(x,y)$ instead of $p(x,y)$.

If $\gcd(p(0,0), XY) \neq 1$, we generate two random primes X' and Y' such that $X < X' < 2X$ and $Y < Y' < 2Y$, and X' and Y' do not divide $p(0,0)$. This can be done in polynomial-time using the recursive prime generation algorithm described in [10]. We then use X', Y' instead of X, Y.

B Proof of Theorem 5

We use the same n and the same $q(x,y)$ as in section 4. We use the same polynomials $q_{ij}(x,y) = x^i y^j X^{k-i} Y^{k-j} q(x,y)$, but only for $0 \leq i+j \leq k$ (instead of $0 \leq i,j \leq k$). We also use the polynomials $q_{ij}(x,y) = x^i y^j n$ for $k < i+j \leq k+\delta$.

We obtain a full-rank lattice L of dimension $\omega = (k+\delta+1)(k+\delta+2)/2$, where the coefficient vectors of the polynomials $\tilde{q}_{ij}(x,y)$ form a triangular basis. The contribution of the polynomials $\tilde{q}_{ij}(x,y)$ for $0 \leq i+j \leq k$ to the determinant is given by:

$$\prod_{0 \leq i+j \leq k} (XY)^k = (XY)^{\frac{k(k+1)(k+2)}{2}}$$

and the contribution of the remaining polynomial is:

$$\prod_{k < i+j \leq k+\delta} X^i Y^j n = (XY)^{d \cdot (2+d^2+6k+3k^2+3d(1+k))/6} \cdot n^{d(3+d+2k)/2}$$

which gives:

$$\det L = (XY)^{\frac{3k(1+k)(2+k)+d(2+d^2+6k+3k^2+3d(1+k))}{6}} \cdot n^{d(3+d+2k)/2}$$

As before, the condition is:

$$2^{(\omega-1)/4} \det(L)^{1/\omega} < 2^{-(k+\delta+1)^2} \cdot (XY)^k \cdot W$$

from which we derive the following condition on XY:

$$XY < W^{(1/\delta)-\varepsilon} \cdot 2^{-4/(\delta\varepsilon^2)-13\delta}$$

for $\varepsilon = \mathcal{O}(1/k)$. As previously, we exhaustive search on the high-order $4/(\delta\varepsilon^2) + 13\delta$ bits of x_0, to obtain the bound:

$$XY < W^{(1/\delta)-\varepsilon}$$

while remaining polynomial-time in $(\log W, 2^\delta)$.

C Factoring with High-bits known: Proof of Theorem 6

Let $N = pq$ be an RSA-modulus and assume that we know that high-order $(1/4+\varepsilon)\log_2 N$ bits of p, for $\varepsilon > 0$. By division we also know the high-order $(1/4+\varepsilon)\log_2 N$ bits of q. We write:

$$p = p_0 + x_0 \quad q = q_0 + y_0$$

$$|x_0| < p_0 N^{-1/4-\varepsilon} = X \quad |y_0| < q_0 N^{-1/4-\varepsilon} = Y$$

where p_0 and q_0 are known and x_0 and y_0 are unknown. We define the polynomial:

$$p(x,y) = (p_0 + x)\cdot(q_0 + y) - N = (p_0 q_0 - N) + q_0 x + p_0 y + xy$$

We have that $p(x_0, y_0) = 0$ and:

$$W = \max(|p_0 q_0 - n|, q_0 X, p_0 Y, XY) > q_0 X > \frac{1}{2} N^{3/4-\varepsilon}$$

We have:

$$XY = p_0 q_0 N^{-1/2-2\varepsilon} < N^{1/2-2\varepsilon}$$

which gives:

$$XY < 2W^{2/3-\varepsilon}$$

so that by guessing one additional bit of x_0 we are under the conditions of theorem 4.