MATHEMATICS OF COMPUTATION
VOLUME 48, NUMBER 177
JANUARY 1987, PAGES 17-28

Solving Bivariate Quadratic Congruences in Random Polynomial Time

By Leonard M. Adleman*, Dennis R. Estes, and Kevin S. McCurley

Dedicated to Daniel Shanks on the occasion of his seventieth birthday

Abstract. It has been known for some time that solving $x^2 \equiv a \pmod{n}$ is as difficult as factoring n, at least in the sense that the two problems are random polynomial time equivalent. By contrast, solving a bivariate quadratic congruence $x^2 - ky^2 \equiv m \pmod{n}$ can usually be done in random polynomial time even if the factorization of n is unknown. This was first proved by Pollard and Schnorr in 1985 under the assumption of the Piltz conjecture for Dirichlet L-functions. We now prove the result without assuming any unproved hypothesis.

1. Introduction.

Let n be an odd positive integer, and let

$$f(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F \tag{1}$$

be a quadratic polynomial with integer coefficients. In this paper we consider the computational problem of constructing a solution to the congruence $f(x, y) \equiv 0 \pmod{n}$, when a solution exists. The classical approach to this problem is to first find the complete prime factorization of n, solve the congruence modulo the primes dividing n, and then use Hensel's lemma and the Chinese Remainder Theorem to construct a solution modulo n. The major drawback to this approach is that the problem of factoring n appears to be computationally infeasible if n is large. The question then arises whether there exists a method for solving $f(x, y) \equiv 0 \pmod{n}$ that does not rely on the ability to factor n.

In some special cases it is known that solving $f(x, y) \equiv 0 \pmod{n}$ is random polynomial time equivalent to factoring n. Consider for example the problem of solving $x^2 \equiv m \pmod{n}$. Rabin [9] has observed that any algorithm that will compute solutions to this congruence will provide a random polynomial time algorithm for factoring n. The algorithm can be easily described as follows:

1. Pick a random number y with $1 \leqslant y \leqslant n - 1$, and compute $m \equiv y^2 \pmod{n}$.
2. Compute a solution x to $x^2 \equiv m \pmod{n}$.
3. Compute $d = \gcd(x + y, n)$. If $1 < d < n$, then you have a proper factorization of n.

Received February 25, 1986.
1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 11Y16; Secondary 11Y05.
*Research sponsored by NSF Grant DCR 8022533.

One can easily prove that if n is not a prime power, then step 3 will give a proper factor of n with probability at least $1/2$. In fact, two of the fastest known factoring algorithms, namely the quadratic sieve algorithm and the continued fraction algorithm, are based on the ability to construct solutions to the congruence $x^2 - y^2 \equiv 0 \pmod{n}$ with $x \not\equiv \pm y \pmod{n}$ [7].

Even though solving quadratic congruences in one variable is equivalent to factoring the modulus, this is not usually the case for quadratic congruences in two variables. It was recently proved by Pollard and Schnorr [8] that if $\gcd(km, n) = 1$, then there exists an algorithm to solve

$$x^2 - ky^2 \equiv m \pmod{n} \tag{2}$$

that runs in random polynomial time if a generalized Riemann hypothesis is true. The main result of this paper is to show that a modified version of their algorithm runs in random polynomial time without the assumption of any unproved hypothesis. In addition, we will show how to reduce a more general problem of solving $f(x, y) \equiv 0 \pmod{n}$ with $f(x, y)$ given by (1) to the problem of solving (2). It should be noted that the special cases in which solving $f(x, y) \equiv 0 \pmod{n}$ is known to be equivalent to factoring n are not covered by our theorem, and we therefore do not shed any light on the computational complexity of factoring. Our modified version of the algorithm for solving (2) is not at all practical to implement, and has a slower running time than the versions discussed in [8] and [11], but it has the advantage that one can fully prove the running time estimates without assuming any unproved hypothesis. While the version of the algorithm that is presented here is not practical, it is probably the case that other variations of the algorithm will work quite well in practice.

Our main result is the following:

THEOREM 1. *Let n be an odd positive integer, and let $f(x, y)$ be given by* (1), *and define $\Delta(f)$, the determinant of f, as follows*:

$$\Delta(f) = \det \begin{bmatrix} 2A & B & D \\ B & 2C & E \\ D & E & 2F \end{bmatrix}. \tag{3}$$

If $\gcd(\Delta(f), n) = 1$, then there exists an algorithm requiring $O(\log(\varepsilon^{-1} \log n) \log^4 n)$ arithmetic operations on integers of size $O(\log n)$ bits that will give a solution to $f(x, y) \equiv 0 \pmod{n}$ with probability $1 - \varepsilon$.

In this paper the term "arithmetic operation" refers to an addition, subtraction, multiplication, or division of ordinary integers. Note that a probabilistic algorithm for solving $f(x, y) \equiv 0 \pmod{n}$ can be classified as a Las Vegas algorithm, since one can easily verify the correctness of a solution in polynomial time.

It is possible to slightly weaken the condition that $\gcd(\Delta(f), n) = 1$. We can, for example, still prove the result if we assume that the complete prime factorization of $\gcd(\Delta(f), n)$ is known (if a solution exists). Of course, if $1 < \gcd(\Delta(f), n) < n$, then we can compute a proper factor of n using the Euclidean algorithm, and one might at first think that one can always use the Chinese Remainder Theorem and/or Hensel's lemma to construct a solution to the original congruence. However, this is not the case with the example

$$x^2 - ty^2 \equiv k \pmod{t^2},$$

where t is odd and composite, and where $\Delta(f) = 8kt$. In the algorithm that we discuss, we would immediately detect the factorization $t^2 = t \cdot t$ and attempt to solve the congruence modulo t and lift the solution modulo t to a solution modulo t^2 using Hensel's lemma. Modulo t, however, the congruence reduces to solving $x^2 \equiv k \pmod{t}$, which has been shown to be essentially as hard as factoring t. The condition on $\Delta(f)$ is intended to rule out cases such as this.

2. Reduction to Solving $x^2 - ky^2 \equiv m \pmod{n}$. The following result shows that in order to solve $f(x, y) \equiv 0 \pmod{n}$, it suffices to solve (2).

THEOREM 2. *Let n be an odd integer, and assume that* $\gcd(\Delta(f), n) = 1$, *where* $\Delta(f)$ *is defined in* (3). *Then there exists a deterministic algorithm requiring* $O(\log^2 n)$ *multiplications modulo n that will, upon input A, B, C, D, E, F, and n, output one of the following*:

(i) *relatively prime integers n_1 and n_2 with $1 < n_i < n$ and $n = n_1 n_2$*;

(ii) *an invertible linear change of variables transforming the congruence $f(x, y) \equiv 0 \pmod{n}$ into a congruence of the form $x^2 - ky^2 \equiv m \pmod{n}$ with* $\gcd(km, n) = 1$;

(iii) *a solution x, y to the congruence $f(x, y) \equiv 0 \pmod{n}$.*

Before we prove Theorem 2, note that if we know a factorization $n = n_1 n_2$ with $1 < n_i < n$ and $\gcd(n_1, n_2) = 1$, then it suffices to solve the two congruences $f(x, y) \equiv 0 \pmod{n_i}$, since we can then combine the results using the Chinese Remainder Theorem to get a solution modulo n. The application of the Chinese Remainder Theorem requires at most $O(\log n)$ multiplications modulo n. Since we can also invert a two-variable linear transformation modulo n in at most $O(1)$ operations modulo n, then to prove Theorem 1 it will suffice to show that we can solve (2) when $\gcd(km, n) = 1$.

To begin the proof of Theorem 2, suppose that we are able to produce a factorization $n = n_1 n_2$ with $1 < n_i < n$. If $\gcd(n_1, n_2) = 1$, then we output n_1, n_2 and stop. If $g = (n_1, n_2) > 1$, then we will argue that either we can find a relatively prime factorization $n = hk$ with $1 < h$, $k < n$, or else both n_1 and n_2 are divisible by n_0, where

$$n_0 = \prod_{i=1}^{m} p_i, \quad \text{if } n = \prod_{i=1}^{m} p_i^{e_i} \text{ is the prime factorization of } n.$$

The algorithm to produce such a factorization, if $g > 1$, is as follows:

1. Set $g = \gcd(n_1, n_2)$, $k = g^2$, and $h = n/g^2$.
2. If $h = 1$, then stop, since $n_0 \mid n_1$ and $n_0 \mid n_2$.
3. Compute $m = \gcd(g, h)$. If $m = 1$, then $n = hk$ gives a factorization with $1 < h$, $k < n$ and $\gcd(h, k) = 1$. If $m > 1$, then replace h by h/m, k by km, and return to step 2.

This algorithm will terminate after at most $O(\log n)$ iterations, since the h's are decreasing. Therefore, for the remainder of the proof we may assume that every integer that arises is either relatively prime to n or else is divisible by n_0.

Note that each prime dividing n fails to divide one of A, B, or D, since $\gcd(\Delta(f), n) = 1$. By the remark in the previous paragraph, we may assume that one of these values is relatively prime to n, and those that are not are divisible by n_0. If

A is relatively prime to n, then we compute $A^{-1} \bmod n$, multiply both sides by $4A^{-1}$, and complete squares to transform $f(x, y) \equiv 0 \pmod n$ into

$$\begin{aligned}(2x + A^{-1}By + A^{-1}D)^2 + (4A^{-1}C - A^{-2}B^2)y^2 + (4A^{-1}E - 2A^{-2}BD)y \\ + 4A^{-1}F - A^{-2}D^2 \equiv 0 \pmod n.\end{aligned}$$

We now make the change of variables

$$u = 2x + A^{-1}By + A^{-1}D, \qquad v = y,$$

so that our new congruence has the form

$$u^2 + Gv^2 + Hv + I \equiv 0 \pmod n. \tag{4}$$

Note that the matrix of our new polynomial is related to the matrix of the old polynomial by

$$\begin{aligned} 4A^{-1}&\begin{bmatrix} 2A & B & D \\ B & 2C & E \\ D & E & 2F \end{bmatrix} \\ &\equiv \begin{bmatrix} 2 & 0 & 0 \\ A^{-1}B & 1 & 0 \\ A^{-1}D & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2G & H \\ 0 & H & 2I \end{bmatrix} \begin{bmatrix} 2 & A^{-1}B & A^{-1}D \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \pmod n, \end{aligned}$$

so that the determinant of the new polynomial is also relatively prime to n.

In a similar manner, we can complete the square if $\gcd(C, n) = 1$. If both A and C are not relatively prime to n, then we can assume that n_0 divides both A and C. In this case, if $\gcd(B, n) = 1$, then we make the change of variables $x = u + v$, $y = v$. This gives us a new polynomial with the same determinant as the original polynomial, but where A is replaced by $A + B + C$. Note that $\gcd(A + B + C, n) = 1$, so that we can again complete squares and reduce to solving a congruence in the form (4). The only case that still has not been reduced to solving (4) is when A, B, and C all have a factor in common with n. In this case we may assume that $\gcd(D, n) = 1$ and n_0 divides A, B, and C. We then compute $d = \gcd(A, B, C)$, take $y = 0$, and find x_i with $Ax_i^2 + Dx_i + F \equiv 0 \pmod{d^i}$, where i is large enough so that d^i is divisible by n. When $i = 1$, the congruence reduces to solving $Dx_1 + F \equiv 0 \pmod d$, which is solvable since $\gcd(D, d) = 1$. We now use what is essentially Hensel's lemma to "lift" a solution $(x_i, 0)$ modulo d^i to a solution $(x_{i+1}, 0)$ modulo d^{i+1}. We let $x_{i+1} = x_i + d^iz$, so that solving $f(x, y) \equiv 0 \pmod{d^{i+1}}$ amounts to solving $(2Ax_i + D)z \equiv -R \pmod d$, where $Ax_i^2 + Dx_i + F = d^iR$. This last congruence is solvable since $n_0 \mid A$, n is odd, and $\gcd(D, n) = 1$. Moreover, the lifting process involves no more than $O(i \cdot \log d) = O(\log^2 n)$ multiplications.

We now turn to the problem of solving (4). If $\gcd(G, n) = 1$, then we can complete the square in the variable v and reduce to solving a congruence of the form (2), with $\gcd(km, n) = 1$. If on the other hand we have $\gcd(G, n) > 1$, then we may assume that n_0 divides G. In this case, since $\gcd(\Delta(f), n) = 1$, it follows that $\gcd(H, n) = 1$. In order to solve (4), it suffices to find v_i satisfying

$$Gv_i^2 + Hv_i + I \equiv 0 \pmod{G^i}, \tag{5}$$

where i is large enough so that G' is divisible by n. The procedure for solving this is exactly the same as was described in the previous paragraph, and this completes the proof of Theorem 2.

3. The Algorithm for Solving (2). We will prove that there exists an algorithm that, upon input k, m, and n with n odd and $\gcd(km, n) = 1$, will output a solution to the congruence (2) with probability $1 - \varepsilon$ in $O(\log(\varepsilon^{-1}\log|k|)\log^3 n \log|k|)$ steps. As was mentioned in Section 1, the algorithm that we analyze in this paper for solving (2) is a modified version of an algorithm first suggested by Pollard, and analyzed by Pollard and Schnorr [8] and Shallit [11]. A rough outline of the original algorithm is as follows.

1. Find a prime p satisfying $p \equiv m \pmod n$ and $(k/p) = 1$.
2. Find a solution x_0 to the congruence $x^2 \equiv k \pmod p$.
3. Use x_0 to find u, v, and k_1 satisfying $u^2 - kv^2 = k_1 p$, with $k_1 = O(|k|^{1/2})$.
4. Since $u^2 - kv^2 \equiv mk_1 \pmod n$, use the composition of binary quadratic forms to reduce to solving $z^2 - kw^2 \equiv k_1 \pmod n$, which is usually equivalent to solving $x^2 - k_1 y^2 \equiv k \pmod n$. If $|k_1| > 1$, then repeat steps 1–4 and reduce to solving a congruence with k_1 replaced by a still smaller value. If $|k_1| = 1$, then use a specialized algorithm to solve $x^2 \pm y^2 \equiv k \pmod n$.

In practice, each of these steps can be carried out using known algorithms, see, e.g., Shanks [12] and Knuth [4]. The only difficulty in analyzing Pollard's algorithm lies in step 1, since we have available only very weak results concerning the distribution of primes in arithmetic progressions, unless we assume the Piltz conjecture for Dirichlet L-functions (the generalized Riemann hypothesis). Without some unproved assumption concerning the distribution of zeros of Dirichlet L-functions, we are unable to prove that there exists an algorithm for constructing a prime in an arithmetic progression a modulo q in random polynomial time. Roughly speaking, our modification of Pollard's algorithm is based on the fact that we are able to prove that for at least half of the values of a modulo q we can find a prime in the arithmetic progression a modulo q in random polynomial time. In order to exploit this fact, we will sometimes reduce the congruence (2) to the problem of solving both $x^2 - ky^2 \equiv Lm \pmod n$ and $x^2 - ky^2 \equiv L \pmod n$, where L is a randomly chosen residue modulo n. We then carry out a reduction procedure on both of these problems. For technical reasons, the primes that we will produce in step 1 for carrying out the reduction step will satisfy the condition $p \not\equiv 1 \pmod 8$ in place of $(k/p) = 1$. An alternative proof might work with the condition $(k/p) = 1$ and an explicit formula version of the Chebotarev density theorem.

It is well known that the distribution of primes in arithmetic progressions modulo q depends on the location of zeros of Dirichlet L-functions formed with characters modulo q (see, e.g., [3, Chapters 19–22]). Let χ be a character modulo q, and let $L(s, \chi)$ be the associated Dirichlet L-function. We will refer to χ as an exceptional character if there exists a real zero β of $L(s, \chi)$ satisfying $\beta > 1 - 0.1/\log q$, and we will further call β an exceptional zero and q an exceptional modulus. It is known (see [5]) that for each integer q there can be at most one exceptional zero, and that if it exists, then the character χ is a real nonprincipal character. It is widely believed that no such zeros exist.

If a and q are positive integers, then we define

$$\vartheta(x;\,q,a)=\sum_{\substack{p\leqslant x\\ p\equiv a\ (\mathrm{mod}\ q)}}\log p,\qquad \pi(x;\,q,a)=\sum_{\substack{p\leqslant x\\ p\equiv a\ (\mathrm{mod}\ q)}}1.$$

The proof of Theorem 1 uses some lemmas concerning $\pi(x;\,q,a)$ and $\vartheta(x;\,q,a)$. The first of these is sometimes referred to as the Brun-Titchmarsh Theorem, although the explicit form that we give here is actually due to Montgomery and Vaughn [6].

LEMMA 1. *If* $x>n>0$, *then* $\pi(x;\,n,a)<2x/(\varphi(n)\log(x/n))$.

The following lemma is due to Bombieri and Gallagher, and a proof appears in [1].

LEMMA 2. *For any* $\varepsilon>0$, *there exists an effectively calculable absolute constant* $D=D(\varepsilon)$ *such that if* $x>n^D$ *and* $\gcd(a,n)=1$, *then*

(i) $\vartheta(x;\,n,a)>(1-\varepsilon)x/\varphi(n)-\chi(a)x^{\beta}/(\beta\varphi(n))$, *if n has an exceptional zero* β, *or*

(ii) $\vartheta(x;\,n,a)>(1-\varepsilon)x/\varphi(n)$, *if n is not exceptional.*

LEMMA 3. *Let n be odd*, $\gcd(a,n)=1$, *and assume that either n is not exceptional, or else if an exceptional character* χ *modulo n exists, then* $\chi(a)=-1$. *Then*

$$\sum_{\substack{p\leqslant x\\ p\equiv a\ (\mathrm{mod}\ n)\\ p\not\equiv 1\ (\mathrm{mod}\ 8)}}1>\frac{x}{3\varphi(n)\log x},$$

provided $x\geqslant n^D$, *where D is an effectively calculable absolute constant.*

Proof. Let b satisfy $b\equiv 1\ (\mathrm{mod}\,8)$, $b\equiv a\ (\mathrm{mod}\,n)$. Then by Lemmas 1 and 2 we have

$$\sum_{\substack{p\leqslant x\\ p\equiv a\ (\mathrm{mod}\ n)\\ p\not\equiv 1\ (\mathrm{mod}\ 8)}}1=\pi(x;\,n,a)-\pi(x;\,8n,b)$$

$$>\frac{x(1-\varepsilon)}{\varphi(n)\log x}-\frac{2x}{\varphi(8n)\log(x/8n)}>\frac{x}{3\varphi(n)\log x},$$

provided ε is small and D is large. □

In the reduction step from $x^2-ky^2\equiv m\ (\mathrm{mod}\,n)$ to $x^2-k_1y^2\equiv k\ (\mathrm{mod}\,n)$, we will make use of the following two-part lemma.

LEMMA 4. *Let n be odd*, $\gcd(m,n)=1$, *D be as in Lemma* 3, *and* $\varepsilon>0$.

(a) *There exists a Monte Carlo algorithm requiring* $O(\log(1/\varepsilon)\log^3 n)$ *arithmetic operations on integers having* $O(\log n)$ *bits that will construct either* (i), (ii), *or* (iii) *below with probability at least* $1-\varepsilon$.

(i) *a prime* p_1 *satisfying* $p_1\not\equiv 1\ (\mathrm{mod}\,8)$, $p_1\equiv m\ (\mathrm{mod}\,n)$, *and* $p_1<n^D$;

(ii) *an integer L with* $\gcd(L,n)=1$, *and primes* p_2, p_3 *satisfying*

$$p_2\equiv L\ (\mathrm{mod}\,n),\quad p_2\not\equiv 1\ (\mathrm{mod}\,8),\quad p_2<n^D,$$

$$p_3\equiv Lm\ (\mathrm{mod}\,n),\quad p_3\not\equiv 1\ (\mathrm{mod}\,8),\quad p_3<n^D;$$

(iii) *an integer d satisfying* $1 < d < n$ *and* $d \mid n$.

(b) *There exists an algorithm having the same running time that will construct either* (i), (ii), *or* (iii) *where the conditions* $p_i \not\equiv 1 \pmod 8$ *are replaced by the conditions* $p_i \equiv 1 \pmod 8$.

Proof. We use the primality test of Solovay and Strassen [13]. In order to test an odd integer $p \geqslant 3$ for primality, we choose a random integer r with $1 < r < p$ and check if

$$r^{(p-1)/2} \equiv (r/p) \pmod p, \tag{6}$$

where the symbol on the right side of the congruence is the Jacobi symbol. If p is prime, then (6) holds for all integers r with $1 < r < p$. If p is not prime, then (6) holds for at most one half of the possible choices of r. If an integer p is found to satisfy (6) for $C_1 \log(1/\varepsilon)$ randomly chosen values of r, where C_1 is a constant, then we will declare it to be "prime". Note that the probability that such a number is actually composite can be made less than $\varepsilon/4$ if C_1 is sufficiently large.

The algorithm for part (a) is to repeat the following sequence of steps up to a maximum of $C_2 \log(1/\varepsilon)\log^2 n$ times for some constant C_2.

1. Choose x randomly with $1 < x < n^D$, $x \equiv m \pmod n$, $x \not\equiv 1 \pmod 8$, and x odd.
2. Choose a random number r with $1 < r < x$. If $\gcd(r, x) > 1$, then go to step 4.
3. Use (6) with $p = x$ to test x for primality. If (6) does not hold, then proceed to step 4. If (6) holds, then go back and repeat steps 2 and 3 up to a maximum of $C_1 \log(1/\varepsilon)$ times for a given x. If x passes all primality tests, then output $p_1 = x$ and stop.
4. Choose L randomly with $1 < L < n$. If $\gcd(L, n) > 1$, then output $d = \gcd(L, n)$ and stop.
5. Choose y, z randomly with $1 < y$, $z < n^D$, $y \equiv L \pmod n$, $z \equiv Lm \pmod n$, $y \not\equiv 1 \pmod 8$, $z \not\equiv 1 \pmod 8$, y, z odd.
6. As in step 3, choose random numbers and use (6) to test both y and z for primality. As soon as either y or z fail a primality test, return to step 1. If both y and z pass $C_1 \log(1/\varepsilon)$ primality tests, then output $p_2 = y$ and $p_3 = z$ and stop.

There are two ways that the algorithm can fail, namely if it declares a composite number to be prime, and if it fails to come across the desired primes. We have already chosen C_1 so that the probability of the first event is less than $\varepsilon/2$. In order to prove that the algorithm will encounter the desired primes with probability at least $1 - \varepsilon/2$, we consider two cases, depending on whether or not an exceptional character χ modulo n exists.

If n is not exceptional, then by Lemma 3, the probability that the x chosen in step 1 is prime is at least

$$\{n^D/(3\varphi(n)D\log n)\}/\{3n^{D-1}/4\} > 4/(9D\log n),$$

so that we have a probability of at least $1 - \varepsilon/2$ of finding a prime x if we examine $O(\log(1/\varepsilon)\log n)$ random values of x.

If on the other hand n is exceptional, and $\chi(m) = -1$, then the preceding argument works without any modification. If $\chi(m) = 1$, then at least one half of the choices for L in step 3 will have either $\gcd(L, n) = 1$ or $\chi(L) = -1$. If $\chi(L) = -1$, then $\chi(mL) = -1$, and by Lemma 3, y and z will both be prime with probability at least $C/\log^2 n$. Hence we have a probability at least $1 - \varepsilon/2$ of finding primes y and z after examining $C_2 \log(1/\varepsilon)\log^2 n$ random values of y and z.

We now estimate the running time of the algorithm. Both sides of (6) can be evaluated in $O(\log n)$ arithmetic operations on integers of size $O(\log n)$ bits, since $\log p = O(\log n)$. Hence an upper bound for the number of operations performed by the algorithm is the maximum number of primality tests, times the maximum number of integers tested, times the number of operations required for each test, or

$$C_1 \log(1/\varepsilon) \cdot C_2 \log(1/\varepsilon)\log^2 n \cdot \log n = O\left(\log^2(1/\varepsilon)\log^3 n\right).$$

This analysis is rather crude, however, since it is extremely unlikely that all of the numbers tested for primality will actually require $C_1 \log(1/\varepsilon)$ primality tests. Even if the first $C_2 \log(1/\varepsilon)\log^2 n$ numbers tested were in fact composite, the probability that it would require more than $3\ C_2 \log(1/\varepsilon)\log^2 n$ primality tests to discover this fact is less than $\varepsilon/2$. Hence the probability that the algorithm produces a correct output after $O(\log(1/\varepsilon)\log^3 n)$ operations is at least $1 - \varepsilon$, saving a factor of $\log(1/\varepsilon)$ in the running time.

The algorithm for part (b) is similar. We need only show that the appropriate arithmetic progressions contain a high density of primes. Let A_1, A_2, and A_3 satisfy $A_1 \equiv m \pmod n$, $A_2 \equiv L \pmod n$, $A_3 \equiv Lm \pmod n$, and $A_i \equiv 1 \pmod 8$. If there is no exceptional zero modulo $8n$, then with probability $1 - \varepsilon$ we will succeed in constructing p_1. If an exceptional character modulo $8n$ exists and $\chi(A_1) = -1$, then we will still succeed in constructing p_1. The only case remaining is if $\chi(A_1) = 1$. In this case we write $\chi = \chi_n\chi_8$, a pointwise product of characters modulo 8 and modulo n. Since it is known that χ_8 is not exceptional (see [10]), it follows that χ_n is nonprincipal. For at least $1/2$ of the choices of L, we will have $\chi_n(L) = 0$ or $\chi_n(L) = -1$. In the first case we get a proper factor of n, and in the second case we have

$$\chi(A_2) = \chi_8(1)\chi_n(mL) = -1, \qquad \chi(A_3) = \chi_8(1)\chi_n(L) = -1,$$

so that we will be able to succeed in constructing p_2 and p_3 with probability $1 - \varepsilon$. □

The ideas behind the following lemmas are contained in [8] and [11], but we restate them here for completeness.

LEMMA 5. *Let* $\gcd(kLm, n) = 1$. *If solutions to any two of the congruences*

$$x^2 - ky^2 \equiv m \pmod n, \quad x^2 - ky^2 \equiv L \pmod n, \quad x^2 - ky^2 \equiv Lm \pmod n$$

are known, then a solution to the third can be found in $O(\log n)$ *multiplications modulo* n.

Proof. This follows from the identity

$$(x^2 - ky^2)(z^2 - kw^2) = u^2 - kv^2, \tag{7}$$

where $u = xz + kyw$, and $v = xw + yz$. Note that if x, y, u, and v are known, then z and w can be recovered by solving a linear system of congruences with determinant $x^2 - ky^2$. The time required to carry this out is dominated by the time needed to invert the determinant modulo n. □

LEMMA 6. *If n is odd and $\gcd(mk_1k_2, n) = 1$, then given solutions to $x_1^2 - k_1y_1^2 \equiv m \pmod n$ and $x_2^2 - k_2y_2^2 \equiv m \pmod n$, one can construct in $O(\log n)$ multiplications modulo n either a solution to $x^2 - k_1k_2y^2 \equiv m \pmod n$, or else a proper divisor of n.*

Proof. If y_1 and y_2 are both invertible modulo n, then

$$\left(x_1y_1^{-1}\right)^2 - m\left(y_1^{-1}\right)^2 \equiv k_1 \pmod n, \qquad \left(x_2y_2^{-1}\right)^2 - m\left(y_2^{-1}\right)^2 \equiv k_2 \pmod n,$$

and by Lemma 5 we can produce integers u, v with $u^2 - mv^2 \equiv k_1k_2 \pmod n$. If v is invertible modulo n, then $(uv^{-1})^2 - k_1k_2(v^{-1})^2 \equiv m \pmod n$. It remains only to deal with the cases of noninvertible y_1, y_2, or v. If $n \mid y_i$, then $x_i^2 \equiv m \pmod n$, giving a solution of the desired congruence. If $n \mid v$, then $u^2 \equiv k_1k_2 \pmod n$, and $x \equiv (m+1)/2 \pmod n$, $y \equiv (m-1)/(2u) \pmod n$ provide a solution. In all other cases, the Euclidean algorithm will produce a nontrivial factor of n. □

In the course of the algorithm, we may occasionally produce a proper factorization of n, and in this case we may argue as in Section 2 that we can either produce a relatively prime factorization $n = n_1n_2$ or else a proper divisor n_3 of n that is divisible by all of the primes dividing n. If the divisor n_3 is produced in the course of the algorithm, then we continue the algorithm with n replaced by n_3, and later use Hensel's lemma to construct a solution modulo a sufficiently large power of n_3 that is divisible by n. In the case of the relatively prime factorization, we can continue the algorithm with n_i in place of n, and later combine the results using the Chinese Remainder Theorem. It was observed in [8] that two arithmetic operations modulo two factors of n take essentially the same time as a single operation modulo n.

Because we shall reduce the problem of solving (2) to the cases $k = \pm 1$, we begin by noting that the case $x^2 - y^2 \equiv m \pmod n$ is trivial, because we may take $x \equiv (m+1)/2 \pmod n$, $y \equiv (m-1)/2 \pmod n$. In [8], the case $k = -1$ was reduced to the case $k = 1$, but we will now give an independent argument for the case $k = -1$ that is very similar to that given in [11]. First use the algorithm of Lemma 4(b) to construct either the prime p_1 or else the primes p_2 and p_3, where $p_i \equiv 1 \pmod 8$. Given the prime p_1, we then use a probabilistic square-root algorithm such as that described in [4, p. 437] to compute a solution x_0 to $x^2 \equiv -1 \pmod{p_1}$. This can be done with probability $1 - \varepsilon$ using $O(\log(1/\varepsilon)\log n)$ arithmetic operations modulo p_1, and since $\log p_1 = O(\log n)$, operations modulo p_1 are essentially as fast as operations modulo n. We next use this in the algorithm of Hermite, as modified in [2], to find integers x and y with $x^2 + y^2 = p_1$, from which it follows that $x^2 + y^2 \equiv m \pmod n$. The algorithm of [2] is equivalent to applying the Euclidean algorithm to p_1 and x_0, so it requires at most $O(\log n)$ operations. Given the primes p_2 and p_3, we use Hermite's algorithm to find integers x, y, z, and w such that $x^2 + y^2 = p_2$ and $z^2 + w^2 = p_3$ and use these in Lemma 5 to find a solution of $u^2 + v^2 \equiv m \pmod n$.

Next we modify an argument in [8] for the cases $k = \pm 2$. As in the case $k = -1$, we first use the algorithm of Lemma 4(b) to construct either the prime p_1 or else the primes p_2 and p_3, where $p_i \equiv 1 \pmod 8$. Let us first assume that we are given p_1. Since $(2/p_1) = (-2/p_1) = 1$, we find an integer x_0 with $x_0^2 \equiv k \pmod{p_1}$. We then define a sequence $q_1, x_1, q_2, x_2, \dots, q_{j+1}$ by taking

$$(8) \qquad \begin{aligned} q_1 &= \left(x_0^2 - k\right)/p_1, \\ x_i &\equiv x_{i-1} \pmod{q_i}, & |x_i| \text{ minimal}, \\ q_{i+1} &= \left(x_i^2 - k\right)/q_i, & i = 1, 2, \dots. \end{aligned}$$

If $|q_i| \geqslant 2$, then $|q_{i+1}| \leqslant |q_i|/4 + |2/q_i| < 5|q_i|/6$, so that the $|q_i|$'s are decreasing, and it follows that $q_{j+1} = \pm 1$ for some j with $j = O(\log n)$. We now have

$$\left(x_0^2 - k\right)\left(x_1^2 - k\right) \cdots \left(x_j^2 - k\right) = (p_1 q_1)(q_1 q_2) \cdots (q_j q_{j+1}),$$

which by the identity (7) can be rewritten as $u^2 - kv^2 = \pm p_1 w^2$ for some integers u, v, and w, using at most $O(\log n)$ operations. If $\gcd(w, n) > 1$, then we have $\gcd(q_i, n) > 1$ for some i. If $n \mid q_i$, then $(x_{i-1})^2 \equiv k \pmod n$, and a solution to $x^2 - ky^2 \equiv m \pmod n$ is easy to construct. In the other cases with $\gcd(w, n) > 1$, $\gcd(q_i, n)$ is a proper factor of n, and we can restart the algorithm as described previously. If $\gcd(w, n) = 1$ then we have $(uw^{-1})^2 - k(vw^{-1})^2 \equiv \pm m \pmod n$. Note that by Lemma 5 we can produce a solution to (2) from a solution to $x^2 - ky^2 \equiv -m \pmod n$, provided we can also produce a solution of $x^2 - ky^2 \equiv -1 \pmod n$. Recall that in the previous case $k = -1$, we showed how to solve $u^2 + v^2 \equiv k \pmod n$, and from this we get either a solution of $x^2 - ky^2 \equiv -1 \pmod n$ or a proper factor of n.

If the algorithm of Lemma 4(b) produces the primes p_2 and p_3, then we use the procedure described above to solve the congruences $x^2 - ky^2 \equiv L \pmod n$ and $x^2 - ky^2 \equiv Lm \pmod n$, later using Lemma 5 to construct a solution of (2).

We now describe the general reduction step if $|k| > 2$. Our goal is to reduce the problem of solving (2) to the problem of solving $x^2 - k_1 y^2 \equiv j \pmod n$, where $|k_1| = O(|k|^{1/2})$. In fact, our reduction step is more complicated, since we may reduce to several problems with different k_1's. The reduction begins by using the algorithm of Lemma 4(a) to produce either the prime p_1 or else the primes p_2 and p_3, all satisfying $p_i \not\equiv 1 \pmod 8$. We consider first the case where p_1 is given. Since $p_1 \not\equiv 1 \pmod 8$, it follows that at least one of the Legendre symbols (k/p_1), $(2k/p_1)$, or $(-k/p_1)$ is equal to 1. We consider these cases separately.

If $(k/p_1) = 1$, then the reduction step is much the same as described in [8]. We define a sequence $q_1, x_1, q_2, x_2, \dots, q_{j+1}$ as in (8), and again the q_i's satisfy $|q_{i+1}| \leqslant |q_i|/4 + |k/q_i|$. If $|q_i| > (13|k|/9)^{1/2}$, then it follows that $|q_{i+1}| \leqslant 49|q_i|/52$, so that $|q_{j+1}| \leqslant (13|k|/9)^{1/2}$ for $j = O(\log|k|)$. We take $k_1 = q_{j+1}$, and note that

$$\left(x_0^2 - k\right)\left(x_1^2 - k\right) \cdots \left(x_j^2 - k\right) = (p_1 q_1)(q_1 q_2) \cdots (q_j q_{j+1}),$$

which, as before, is equivalent to $u^2 - kv^2 = p_1 k_1 w^2$ for some integers u, v, and w. The case $\gcd(w, n) > 1$ will again produce either a square root of k modulo n or else a proper factor of n. If $\gcd(w, n) = 1$, then $(uw^{-1})^2 - k(vw^{-1})^2 \equiv mk_1 \pmod n$, and by Lemma 5, in order to solve (2), it now suffices to solve $x^2 - ky^2 \equiv k_1 \pmod{n}$, which then reduces to solving $x^2 - k_1y^2 \equiv k \pmod{n}$.

Next consider the case that $(-k/p_1) = 1$. By Lemma 6, in order to solve (2), it suffices to solve both $x^2 + y^2 \equiv m \pmod{n}$ and $x^2 + ky^2 \equiv m \pmod{n}$. The former was discussed previously, and the latter can be reduced to solving $x^2 - k_1y^2 \equiv k \pmod{n}$ with $|k_1| < (13|k|/9)^{1/2}$ as in the previous paragraph.

If finally we have $(2k/p_1) = 1$, then by Lemma 6, in order to solve (2), it suffices to solve $x^2 - 2y^2 \equiv m \pmod{n}$ and $x^2 - 2ky^2 \equiv m \pmod{n}$. The former has already been dealt with, and the latter can be reduced to solving $x^2 - k_1y^2 \equiv 2k \pmod{n}$, where now $|k_1| < (26|k|/9)^{1/2}$.

If the algorithm of Lemma 4(a) produces L, p_2, and p_3 instead of p_1, then we simply apply the reduction step to the two congruences $x^2 - ky^2 \equiv L \pmod{n}$ and $x^2 - ky^2 \equiv Lm \pmod{n}$ simultaneously in the same way that was described for the case of p_1.

4. The Running Time Analysis. The algorithm that we have described consists of a sequence of reduction steps, each of which will reduce a congruence $x^2 - ky^2 \equiv m \pmod{n}$ to the problem of solving one or two congruences of the form $x^2 - k_1y^2 \equiv k \pmod{n}$, where $|k_1| < (26|k|/9)^{1/2}$. After the ith reduction step, there are at most 2^i congruences of the form $x^2 - k_iy^2 \equiv m \pmod{n}$ to be solved, and each of these has

$$|k_i| < (26/9)^{1-2^{-i}}|k|^{2^{-i}},$$

so that $|k_i| \leqslant 2$ as soon as $2^i > (\log|k|)/\log(27/26)$. Hence we are required to solve at most $O(\log|k|)$ problems.

We now estimate the probability of success in the algorithm. If there is a probability δ of failure in a reduction step on any individual problem, then since there are $O(\log|k|)$ problems to be solved, we have a probability of failure for the entire algorithm of $O(\delta \log|k|)$. In order to make this probability less than ε, we take $\delta = \varepsilon/\log|k|$, so that a reduction step on any one problem requires at most $O(\log(\varepsilon^{-1}\log|k|)\log^3 n)$ arithmetic operations modulo n. The work done in each reduction step is dominated by the time for the algorithm of Lemma 4, since the square roots can be calculated with probability $1 - \varepsilon$ in $O(\log(1/\varepsilon)\log n)$ operations. Since there are $O(\log|k|)$ individual reductions to be performed, the number of operations in the entire algorithm is $O(\log(\varepsilon^{-1}\log|k|)\log^3 n \log|k|)$ in order to produce a correct output with probability at least $1 - \varepsilon$.

Note also that the storage requirements of the algorithm are modest, requiring no more than $O(\log|k|)$ storage locations containing integers of $O(\log n)$ bits.

Department of Computer Science
University of Southern California
Los Angeles, California 90089-0782

Department of Mathematics
University of Southern California
Los Angeles, California 90089-1113

Department of Mathematics
University of Southern California
Los Angeles, California 90089-1113

1. E. Bombieri, "Le grand crible dans la théorie analytique des nombres" (Avec une sommaire en anglais), *Astérisque*, No. 18, Société Mathématique de France, Paris, 1974.

2. J. Brillhart, "Note on representing a prime as a sum of two squares," *Math. Comp.*, v. 29, 1972, pp. 1011–1013.

3. H. Davenport, *Multiplicative Number Theory*, 2nd ed., revised by Hugh L. Montgomery, Springer-Verlag, Berlin and New York, 1980.

4. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

5. K. McCurley, "Explicit zero-free regions for Dirichlet L-functions," *J. Number Theory*, v. 19, 1984, pp. 7–32.

6. H. L. Montgomery & R. Vaughn, "The large sieve," *Mathematika*, v. 20, 1973, pp. 119–134.

7. C. Pomerance, "Analysis and comparison of some integer factoring methods," in *Computational Methods in Number Theory: Part* 1 (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Math. Centre Tract 154, Math. Centre, Amsterdam, 1982, pp. 89–139.

8. J. M. Pollard & C. P. Schnorr, "Solution of $x^2 + ky^2 \equiv m \pmod{n}$, with application to digital signatures," preprint, 1985.

9. M. Rabin, *Digitalized Signatures and Public-Key Functions as Intractible as Factorization*, MIT Laboratory for Computer Science Report TR-212, 1979.

10. J. B. Rosser, "Real roots of Dirichlet L-series," *Bull. Amer. Math. Soc.*, v. 55, 1949, pp. 906–913.

11. J. O. Shallit, *An Exposition of Pollard's Algorithm for Quadratic Congruences*, Technical Report 84-006, Dept. of Computer Science, University of Chicago, 1984.

12. D. Shanks, *Five Number-Theoretic Algorithms*, Proc. Second Manitoba Conference on Numerical Mathematics, 1972, pp. 51–70.

→ R. Solovay & V. Strassen, "A fast Monte-Carlo test for primality," *SIAM J. Comput.*, v. 6, 1977, pp. 84–85.