MATHEMATICS OF COMPUTATION
VOLUME 48, NUMBER 177
JANUARY 1987, PAGES 103-121

Implementation of a New Primality Test

By H. Cohen and A. K. Lenstra

Dedicated to Daniel Shanks on the occasion of his 70th birthday

Abstract. An implementation of the Cohen-Lenstra version of the Adleman-Pomerance-Rumely primality test is presented. Primality of prime numbers of up to 213 decimal digits can now routinely be proved within approximately ten minutes.

Introduction. In [2] a theoretically and algorithmically simplified version of the Adleman-Pomerance-Rumely primality testing algorithm [1] was presented. To prove its practical value, we implemented the algorithm from [2]. As a result, numbers of up to 213 decimal digits can be handled within approximately ten minutes of computing time on a CDC Cyber 170/750.

In fact, two programs have been written. The first program, written in Pascal, was devised for numbers of up to 104 decimal digits. In order to increase the portability of the program, we translated it into Fortran and at the same time extended its capacity to 213 decimal digits. This Fortran implementation now runs on the following computers: CDC Cyber 170/750, CDC 205, and Cray 1. For these machines, multiprecision integer arithmetic routines were written in their respective machine languages by D. T. Winter from the Centrum voor Wiskunde en Informatica in Amsterdam.

This paper does not present any new results. We only describe how a slightly improved version of the algorithm from [2] was implemented. No detailed program texts will be given, but we supply enough information for anyone who might be interested in implementing the algorithm from [2], and who was discouraged by the more theoretical approach taken in [2].

The primality testing algorithm, as it has been implemented, is described in Section 1. A further explanation of those parts of the algorithm for which we felt that this might be helpful, can be found in Sections 2 through 6. Some examples and running times are given in Section 7. In the Appendix (which appears in the supplements section at the end of this issue), detailed formulae for multiplication in cyclotomic rings are presented.

By $\mathbf{Z}$ we denote the ring of integers, and by $\mathbf{Q}$ the field of rational numbers. The number of times that a prime number p appears in m is denoted by $v_p(m)$, for $m \in \mathbf{Z}_{\neq 0}$. By $r \mid m$ we mean that r is a positive divisor of m. For a prime power p^k we denote by ζ_{p^k} a primitive p^kth root of unity.

Received September 18, 1985; revised March 4, 1986.
1980 *Mathematics Subject Classification*. Primary 10-04, 10A25.
Key words and phrases. Primality testing.

1. The Primality Test. Combination of the results from [2, Sections 10 and 12] and [5, Section 8] leads to the primality testing algorithm described in this section. For the theoretical background we refer to [2], [5]. The notation that we introduce here will be used throughout this paper.

Let N be some large integer. The primality testing algorithm described here can be used to determine whether an integer n, $1 < n \leqslant N$, is prime. The algorithm consists of two parts. The first part, the preparation of tables, has to be executed only once, because it only depends on N; the second part, the primality test, has to be performed for every number n to be tested.

(1.1) *Preparation of Tables.* (a) Select an even positive integer t with $e(t) > N^{1/2}$ (cf. (1.5)), where

$$e(t) = 2 \cdot \prod_{q \text{ prime},\, q-1 \mid t} q^{v_q(t)+1},$$

and tabulate the primes dividing $e(t)$; these primes will in the sequel be called the q-primes. (In the Fortran program, t is chosen as $55440 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$. Because $e(55440) = 4.920 \cdot 10^{106}$ (rounded off downwards), we can handle numbers of up to 213 decimal digits. For this value of t the number of odd q-primes is 44. By using the improvement $e(t) > N^{1/3}$ mentioned in [2], one could with the same t handle numbers of up to 320 digits, but this has not been implemented. (See also Remark (1.5).) The choice of t can be made, e.g., by table look-up (see for instance Table 1 of [2]).

(b) Perform steps (b1) and (b2) for each odd prime $q \mid e(t)$ (so $q - 1 \mid t$).

(b1) Find by trial and error a primitive root g modulo q, i.e., an integer $g \not\equiv 0 \bmod q$ such that $g^{(q-1)/p} \not\equiv 1 \bmod q$ for any prime $p \mid q - 1$. In our implementation this was done by trying $g = 2, 3, 4, \ldots$ in succession. Make a table of the function $f\colon \{1, 2, \ldots, q-2\} \to \{1, 2, \ldots, q-2\}$ defined by $1 - g^x \equiv g^{f(x)} \bmod q$. (So, first make a table of $\log(g^x \bmod q) = x$, for $x = 1, 2, \ldots, q-2$, and next $f(x) = \log((1 - g^x) \bmod q)$, for $x = 1, 2, \ldots, q-2$.)

(b2) Perform steps (b2a), (b2b), (b2c) for each $p \mid q - 1$ (so $p \mid t$).

(b2a) Put $k = v_p(q-1)$, the number of factors of p in $q - 1$.

(b2b) If $p^k \neq 2$, compute and tabulate

$$j_{p,q} = \sum_{x=1}^{q-2} \zeta_{p^k}^{x+f(x)} \in \mathbf{Z}[\zeta_{p^k}].$$

(b2c) If $p = 2$, $k \geqslant 3$, compute and tabulate

$$j_{2,q}^{*} = \sum_{x=1}^{q-2} \zeta_{2^k}^{2x+f(x)} \in \mathbf{Z}[\zeta_{2^k}],$$

and

$$j_{2,q}^{\#} = \sum_{x=1}^{q-2} \zeta_{2^k}^{2^{k-3}(3x+f(x))} \in \mathbf{Z}[\zeta_{2^k}].$$

Notice that $j_{2,q}^{*} \cdot j_{2,q}$ and $(j_{2,q}^{\#})^2$ correspond, respectively, to $j_{2,q}^{*}$ and $j_{2,q}^{\#}$ from [2, Section 12].

The Jacobi sums in (b2b) and (b2c) can be computed as follows. We represent an element $\sum_{0 \leqslant i < (p-1)p^{k-1}} a_i \zeta_{p^k}^i$ of $\mathbf{Z}[\zeta_{p^k}]$, with $a_i \in \mathbf{Z}$, as a vector $(a_i)_{0 \leqslant i < (p-1)p^{k-1}}$. Initially, we put $a_i = 0$ for $0 \leqslant i < (p-1)p^{k-1}$. Let $a, b \in \mathbf{Z}$; for the computation of $j_{p,q}$ we take $a = b = 1$, for $j_{2,q}^*$ we take $a = 2$, $b = 1$, and for $j_{2,q}^{\#}$ finally $a = 3 \cdot 2^{k-3}$, $b = 2^{k-3}$. For $x = 1, 2, \ldots, q-2$ in succession we do the following:

> Put $l = a \cdot x + b \cdot f(x) \mod p^k$. If $l < (p-1)p^{k-1}$, increase a_l by one. Otherwise, decrease $a_{l-ip^{k-1}}$ by one for $i = 1, 2, \ldots, p-1$. (Notice that, for each x, this is the same as replacing the vector (a_i) by the vector $(a_i) + \zeta_{p^k}^{a \cdot x + b \cdot f(x)}$ modulo the minimal polynomial of ζ_{p^k}, the p^kth cyclotomic polynomial $\sum_{i=0}^{p-1} X^{ip^{k-1}}$.)

At the end of this process we have a representation for the Jacobi sum in the vector (a_i).

This finishes the preparation of the tables.

(1.2) *Remark*. Notice that only the Jacobi sums are tabulated, and not the Jacobi sum powers as in [2, Section 12], because that would require a lot of memory space, even for moderately sized t. This implies that the Jacobi sum powers have to be recomputed for every n. As they are easily calculated, this takes only a relatively small amount of computing time (cf. remark after (6.1)). (In the Pascal program we stored the Jacobi sum powers, as in [2, Section 12]; this resulted in a 1.5% speed-up.)

The reason that the Jacobi sums themselves are tabulated and not recomputed for every n, is that their computation requires too much memory space (namely the space to store the table of the function f).

We now present the primality testing algorithm as it follows from [2, Sections 10 and 12] and [5, Section 8]. A detailed description of the steps of the algorithm can be found in Sections 2 through 6.

(1.3) *The Primality Test*. Let n, $1 < n \leqslant N$, be an odd integer to be tested for primality. Suppose that tables containing t, $e(t)$, the q-primes, and the Jacobi sums are prepared according to (1.1).

Preliminary tests. (a) Test whether $\gcd(t \cdot e(t), n) = 1$. If not, then a prime divisor of n is obtained, because all factors of $t \cdot e(t)$ are known from (1.1). In this case, Algorithm (1.3) is terminated.

(b) Select a trial division bound B and perform the trial division step (2.1) as described in Section 2 for this value of B. If a nontrivial divisor of n is found, then n is composite and Algorithm (1.3) halts. If no nontrivial divisor of n is found and $B > n^{1/2}$, then n is prime and Algorithm (1.3) halts. Otherwise, let l^- be the set of odd prime numbers $\leqslant B$ dividing $n-1$, let r^- be the largest odd factor of $n-1$ without prime factors $\leqslant B$, and let $f^- = (n-1)/r^-$ be the factored part of $n-1$. Similarly, let l^+, r^+, and f^+ be the set of odd prime factors $\leqslant B$, the nonfactored part, and the factored part of $n+1$, respectively.

(c) Select a small positive integer m, and perform the probabilistic compositeness test (3.4) as described in Section 3 at most m times. If, during the execution of (3.4), n is proved to be composite, Algorithm (1.3) halts.

(d) As explained in [2, Section 10], it is useful to distinguish between the prime power factors of t that divide $n-1$ and those that do not divide $n-1$. Declare therefore for all prime powers p^k dividing t a Boolean variable flag_{p^k}, and put flag_{p^k} = "true" if $n \equiv 1 \bmod p^k$, and flag_{p^k} = "false" otherwise.

We could have done something similar for the prime power factors of t that divide $n+1$. We did not incorporate that in our implementations, however (see also Remark (4.6)).

(e) Perform the Lucas-Lehmer test (4.4) as described in Section 4. If n does not pass (4.4), report that (1.3) fails if (4.4) fails, and report that n is composite if that has been proved in (4.4). In either case, Algorithm (1.3) is terminated.

If n passes (4.4) and its primality has been proved in (4.4), report that n is prime and halt. Otherwise let, for p^k such that flag_{p^k} = "true", elements $\beta^i_{p^k}$ of $\mathbf{Z}/n\mathbf{Z}$ be as in (4.2) and (4.4)(c1). Then $\beta^1_{p^k}$ is a zero of the p^kth cyclotomic polynomial, and $\beta^i_{p^k}$ is its ith power.

If n passes the Lucas-Lehmer test, then for each r dividing n there exists an integer $i \geqslant 0$ such that $r \equiv n^i \bmod (f^- \cdot f^+)$ (where $f^- \cdot f^+$ can be replaced by any number built up from primes dividing $f^- \cdot f^+$; cf. (5.2)).

(f) Perform Algorithm (5.5) to select $s = s_1 \cdot s_2 > n^{1/2}$ and a new value for t dividing the old one (cf. (1.5)).

Here s_1 is built up from primes dividing $f^- \cdot f^+$, and s_2 is coprime to s_1 and built up from primes dividing $e(t)$. The factors of s_1 have been dealt with by means of the Lucas-Lehmer test, and the factors of s_2 will be dealt with by means of Jacobi sums.

For the resulting values of t and s we have $n^t \equiv 1 \bmod s$ (cf. [2, Proposition (4.1)], (1.1)(a), and step (a)).

(g) Declare for each prime $p > 2$ dividing t a Boolean variable λ_p. Put λ_p = "true" if $n^{p-1} \not\equiv 1 \bmod p^2$ or $p \mid f^- \cdot f^+$, and λ_p = "false" otherwise.

This λ_p tells us whether or not condition [2, (6.4)], that has to be satisfied for all primes dividing t, is satisfied already for p. For a further explanation of this step we refer to Remark (4.5).

Pseudoprime tests with Jacobi sums. Perform steps (h), (i) for each prime p dividing t.

(h) For each integer $k \geqslant 1$ with $p^k \mid t$, determine integers u_k, v_k such that $n = u_k p^k + v_k$, and $0 \leqslant v_k < p^k$.

(i) Perform steps (i1), (i2), (i3) for each prime $q \mid s_2$ with $p \mid q-1$.

(i1) Put $k = v_p(q-1)$, and $u = u_k$, $v = v_k$ as in (h). Perform steps (i1a), (i1b), (i1c), (i1d). The exponentiations in $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ may be carried out via the multiplication and squaring routines given in the Appendix. For further computational details of this step we refer to Section 6.

(i1a) If $p \neq 2$, put

$$M = \{x \in \mathbf{Z}: 1 \leqslant x \leqslant p^k,\ x \not\equiv 0 \bmod p\},$$

and let σ_x for $x \in M$ be the automorphism of $\mathbf{Q}(\zeta_{p^k})$ for which $\sigma_x(\zeta_{p^k}) = \zeta^x_{p^k}$. Calculate

$$j_{0,p,q} = \prod_{x \in M} \sigma_x^{-1}\big((j_{p,q})^x\big) \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}],$$

and

$$j_{v,p,q} = \prod_{x \in M} \sigma_x^{-1}\left((j_{p,q})^{[vx/p^k]} \right) \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}],$$

where $[y]$ denotes the greatest integer $\leqslant y$ (cf. (6.1)).

(i1b) If $p^k = 2$, put

$$j_{0,2,q} = q, \qquad j_{1,2,q} = 1.$$

(i1c) If $p^k = 4$, calculate

$$j_{0,2,q} = j_{2,q}^2 \cdot q \in \mathbf{Z}[\zeta_4]/n\mathbf{Z}[\zeta_4],$$

and

$$j_{v,2,q} = \begin{cases} 1 & \text{if } v = 1, \\ j_{2,q}^2 & \text{if } v = 3. \end{cases}$$

Notice that $j_{3,2,q} = j_{2,q}^2$, and not $j_{2,q}$ as stated erroneously in [2, step (b2e) of (12.1)].

(i1d) If $p = 2$, $k \geqslant 3$, put

$$L = \{x \in \mathbf{Z}: 1 \leqslant x \leqslant 2^k, \ x \text{ is odd}\}, \qquad M = \{x \in L: x \equiv 1 \text{ or } 3 \bmod 8\},$$

and let σ_x for $x \in M$ be the automorphism of $\mathbf{Q}(\zeta_{2^k})$ for which $\sigma_x(\zeta_{2^k}) = \zeta_{2^k}^x$. Calculate

$$j_{0,2,q} = \prod_{x \in M} \sigma_x^{-1}\left((j_{2,q}^* \cdot j_{2,q})^x \right) \in \mathbf{Z}[\zeta_{2^k}]/n\mathbf{Z}[\zeta_{2^k}],$$

and

$$j_{v,2,q} = \begin{cases} \prod_{x \in M} \sigma_x^{-1}\left((j_{2,q}^* \cdot j_{2,q})^{[vx/2^k]} \right) \in \mathbf{Z}[\zeta_{2^k}]/n\mathbf{Z}[\zeta_{2^k}] & \text{if } v \in M, \\ \left(j_{2,q}^\#\right)^2 \cdot \prod_{x \in M} \sigma_x^{-1}\left((j_{2,q}^* \cdot j_{2,q})^{[vx/2^k]} \right) \in \mathbf{Z}[\zeta_{2^k}]/n\mathbf{Z}[\zeta_{2^k}] & \text{if } v \in L - M, \end{cases}$$

(cf. (6.1)).

(i2) If flag_{p^k} = "true", perform step (i2a), otherwise perform step (i2b).

(i2a) Define a ring homomorphism $\lambda: \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}] \to \mathbf{Z}/n\mathbf{Z}$ by $\lambda(\zeta_{p^k}) = \beta_{p^k}^1$, and verify that there exists an integer $h \in \{0, 1, \ldots, p^k - 1\}$ with

$$\lambda(j_{0,p,q})^u \cdot \lambda(j_{v,p,q}) = \beta_{p^k}^h$$

(cf. Section 6), where $\beta_{p^k}^i$ for $0 \leqslant i < p^k$ are as in (e) (notice that we apply here the results from [2, Section 10] for the case where, in the notation of [2, Section 10], $f = 1$, i.e., $n \equiv 1 \bmod p^k$). If h does not exist, then n is composite and Algorithm (1.3) terminates. Suppose that h exists.

(i2b) Verify that there exists an integer $h \in \{0, 1, \ldots, p^k - 1\}$ with

$$j_{0,p,q}^u \cdot j_{v,p,q} = \zeta_{p^k}^h \bmod n\mathbf{Z}[\zeta_{p^k}]$$

(cf. (6.2)). If h does not exist, then n is composite and Algorithm (1.3) terminates. Suppose that h exists.

(i3) If $h \not\equiv 0 \bmod p$ and p is odd, put λ_p = "true".

Additional tests. Perform steps (j) and (k) for every prime p dividing t for which λ_p = "false".

(j) Select a small prime number q not dividing s such that

$$q \equiv 1 \bmod 2p, \qquad n^{(q-1)/p} \not\equiv 1 \bmod q.$$

(In the Fortran implementation the search for these prime numbers begins at $20p + 1$, and we allow for at most 50 primes of the form $2pm + 1$ to be considered.) If such a prime q cannot be found below a reasonable limit, do the following. Test whether n is a pth power. If so, report that n is composite and halt. Otherwise, halt with the message that the algorithm is unable to prove that n is prime. Suppose now that q has been found. If $n \equiv 0 \bmod q$, then a prime divisor of n is found and the algorithm halts.

(k) Let u, v be integers such that $n = up + v$, with $0 \leqslant v < p$ (cf. (h)), and perform steps (1.1)(b1), (1.1)(b2b), (i1a), (i2). Test whether the resulting $h \in \mathbf{Z}$ satisfies $h \not\equiv 0 \bmod p$. If this is not the case, n is composite, and Algorithm (1.3) halts. Otherwise, put $\lambda_p =$ "true".

Final trial division. We now have proved that for every divisor r of n there exists $i \in \{0, 1, \ldots, t-1\}$ such that $r \equiv n^i \bmod s$. Since $s > n^{1/2}$, the following suffices to determine the divisors of n.

(l) Put $\tilde{n} = n \bmod s$, $r = 1$, and perform steps (l1), (l2), (l3).

(l1) Replace r by $(\tilde{n}r) \bmod s$ in such a way that the new value of r satisfies $0 \leqslant r < s$.

(l2) If $r = 1$, report that n is prime and halt.

(l3) If $r \mid n$ and $r < n$, report that n is composite and halt.

Notice that (l1), (l2), and (l3) are performed at most t times, because $n^t \equiv 1 \bmod s$ (cf. step (f)).

This finishes the description of the primality testing algorithm (1.3).

(1.4) *Remark.* The above formulation of the primality testing algorithm follows from [2, Section 10, (11.5), Section 12] and [5, Section 8]. We do not need λ_2 in (1.3)(g), because λ_2 is already set to "true" by the Lucas-Lehmer test (4.4) (cf. Remark (4.5)). The correctness of (i2a) follows from [2, Section 10].

(1.5) *Remark.* In [6] it is shown that for positive integers d, s, and n such that $\gcd(d, s) = 1$ and $s > n^{1/3}$, there exist at most 11 divisors of n that are congruent to d modulo s. Furthermore, an efficient algorithm is presented to determine all these divisors.

Incorporation of this algorithm in the final trial division (1.3)(l) would change the conditions on $e(t)$ in (1.1)(a) and s in (1.3)(f) into $e(t) > N^{1/3}$ and $s > n^{1/3}$, respectively. We did not implement this.

In the rest of this paper we will have a closer look at the steps of Algorithm (1.3).

2. Trial Division.

2. Trial Division. Step (b) of the primality testing algorithm (1.3), the trial division, has two purposes: to detect composite numbers with a small factor, and to determine the small prime factors of $n^2 - 1$, for numbers n for which we attempt to prove primality. Let B be as in step (b) of (1.3) the trial division bound.

The trial division routine that will be described below needs a table of prime numbers up to B. Our implementations made use of a table of prime numbers up to 10^6. To save memory space, only the differences between consecutive primes were stored in such a way that as many successive differences as possible were packed in one machine word.

For the primes up to 10^6 none of the differences exceeds 1000, so that on the CDC 170/750, which has 48-bit integers, we can accommodate four differences in one single-length integer. (In the Pascal implementation we use the full 60-bit machine words of the CDC 170/750 by packing 6 differences in one machine word; in the Fortran program we do not do so in order to make the program less machine dependent and to increase its portability.)

(2.1) *Trial Division.* First set r^- and r^+ equal to the largest odd factors of $n - 1$ and $n + 1$, respectively, and set l^- and l^+ both equal to the empty set $\varnothing$. Next, for all primes $p \leqslant B$ in succession, do the following:

> If $n + 1 \equiv 1 \bmod p$, then p divides n, so that the execution of Algorithm (2.1) and of Algorithm (1.3) is terminated. Otherwise, if $n + 1 \equiv 0 \bmod p$, remove all factors p from r^+ and replace l^+ by $l^+ \cup \{p\}$, and finally, if $n + 1 \equiv 2 \bmod p$, remove all factors p from r^- and replace l^- by $l^- \cup \{p\}$.

If, after this search for small factors of $n^3 - n$, no factor of n is found, set f^- and f^+ equal to $(n - 1)/r^-$ and $(n + 1)/r^+$, respectively.

This finishes the description of Algorithm (2.1).

(2.2) *Remark.* In the Fortran program, B can be chosen as any integer in $\{11, 12, \ldots, 10^6\}$ (cf. remark before (5.2)). In practice, we always take $B \geqslant 55441$, so that step (a) of (1.3) can be avoided (where 55441 is the initial value of $t + 1$).

(2.3) *Remark.* In the main loop of Algorithm (2.1) we have to perform one division of a 'multiple' $(n + 1)$ by a single-length integer (p) for each prime number $p < 10^6$ (for an explanation of 'multiple' see Section 7). If the product of two consecutive primes p_1 and p_2 can be represented in one single-length integer, as is the case on the CDC 170/750, then we can replace the computation of $(n + 1) \bmod p_1$ and $(n + 1) \bmod p_2$ by the computation of $(n + 1) \bmod (p_1 p_2) = m$, and next $m \bmod p_1$ and $m \bmod p_2$.

Per two primes, this saves one 'multiple'-single division at the cost of two single-single divisions. It depends on the size of n and the actual implementation of the division routines whether this change will result in a speed-up of the trial division routine (on CDC 170/750 it resulted only in a 2% speed-up).

(2.4) *Remark.* In an early version of the Pascal program we attempted to find also some prime factors $> B$ of r^- and r^+ by means of the Pollard rho-method. Because this Pollard step appeared to be quite time-consuming, and because we never found any factor $> B$, we left this step out in later versions.

As a referee pointed out, it might be useful to use Pollard's $p - 1$ (or $p + 1$) method, or even the elliptic curve method, to find extra factors of r^- and r^+.

3. The Probabilistic Compositeness Test.

Probabilistic compositeness tests are well known and can be found at many places in the literature [4], [7], [8], [9]. In step (c) of the primality testing algorithm (1.3) we perform a number of these tests to detect composite numbers that passed the trial division step. Of course, we cannot guarantee that compositeness is always detected here (otherwise the rest of Algorithm (1.3) would have been superfluous), but in practice it never occurred that a composite number passed this step.

For completeness we formulate the probabilistic compositeness test that was applied in Algorithm (1.3); furthermore, we discuss some computational aspects of the test, which will also be useful in the sequel.

Let $n - 1 = u \cdot 2^k$ with u odd and $k \geqslant 1$. An integer a is called a *witness* to the compositeness of n if the following three conditions are satisfied:

(3.1) n does not divide a,

(3.2) $a^u \not\equiv 1 \bmod n$,

(3.3) $a^{u \cdot 2^i} \not\equiv -1 \bmod n$ for $i = 0, 1, \ldots, k-1$.

Obviously, if a is a witness to the compositeness of n, then n is composite. Conversely, if n is an odd composite number, then there are at least $3(n-1)/4$ witnesses to the compositeness of n among $\{1, 2, \ldots, n-1\}$ (cf. [8]). This leads to the following test.

(3.4) *Probabilistic Compositeness Test.* First choose at random an integer a from $\{1, 2, \ldots, n-1\}$. Next verify (3.2) and (3.3) by computing $a^u \bmod n$ (cf. (3.6)), and successively squaring the result modulo n. If (3.2) and (3.3) hold, then n is composite and the execution of Algorithm (1.3) is terminated (notice that (3.1) already holds by virtue of the choice of a). Otherwise, n passes the probabilistic compositeness test.

This finishes the description of the test.

(3.5) *Remark.* In our implementations of Algorithm (1.3) the user can specify how often (3.4) should be performed (m in (1.3)(c)). For composite numbers, a small number of probabilistic compositeness tests ($m = 1$ or $m = 2$) usually suffices to detect compositeness. For numbers that already were declared to be 'probably prime' by others, and that had to be proved prime by (1.3), we skipped the probabilistic compositeness test (3.4) ($m = 0$).

In fact, we only used (3.4) to debug the rest of Algorithm (1.3): If a number passed a small number of probabilistic compositeness tests, and it was declared to be composite by the rest of (1.3), this always led to the discovery of a bug in the implementation of (1.3). Of course, not all bugs are detectable in this way.

(3.6) *Remark.* We now discuss some computational aspects of the exponentiation modulo n in (3.2). As is well known, $a^u \bmod n$ can be computed in $\lfloor \log_2 u \rfloor$ squarings and $\nu(u)$ multiplications of integers modulo n, where $\nu(u)$ is the number of ones in the binary representation of u (cf. [4, Section 4.6.3]). We can improve on the number of multiplications modulo n as follows [4, p. 444].

Instead of the binary representation of u, we use, for some integer m to be specified below, the 2^mary representation $(u_t, u_{t-1}, \ldots, u_1, u_0)$ of u, i.e., $u = u_t 2^{mt} + u_{t-1} 2^{m(t-1)} + \cdots + u_1 2^m + u_0$, where $u_i \in \{0, 1, \ldots, 2^m - 1\}$ and $u_t \neq 0$. Let $u_i = v_i 2^{l_i}$, with v_i odd and $0 \leqslant l_i < m$ for $0 \leqslant i \leqslant t$ (cf. (3.7)).

To compute $a^u \bmod n$, first compute the first 2^{m-1} odd powers of a modulo n by repeated multiplication by $a^2 \bmod n$. This takes 2^{m-1} multiplications of integers modulo n. We get $a_1 = a$, $a_3 = a^3 \bmod n, \ldots, a_{2^m - 1} = a^{2^m - 1} \bmod n$.

Next compute $r = a^{u_t} \bmod n$ by l_t successive squarings modulo n of a_{v_t}. Finally, perform the following three steps for $i = t-1, t-2, \ldots, 1, 0$ in succession:

—raise r to the $(2^{m - l_i})$th power by $m - l_i$ successive squarings modulo n;

—multiply r by a_{v_i} modulo n;

—raise r to the (2^{l_i})th power by l_i successive squarings modulo n.

As a result, we get $r = a^u \bmod n$.

The total number of multiplications modulo n is $2^{m-1} + \nu_m(u)$, where $\nu_m(u)$ is the number of nonzero u_i's; the total number of squarings modulo n is, as in the binary method, $\lfloor \log_2 u \rfloor$. Clearly, m should be chosen in such a way that $2^{m-1} + \nu_m(u)$ is minimal. We estimate $\nu_m(u)$ by $(1 - 2^{-m})\lceil \log_{2^m} u \rceil$ and because u will be of the same order of magnitude as n, we can take m such that $2^{m-1} + (1 - 2^{-m})\lceil \log_{2^m} n \rceil$ is minimized.

(The Fortran implementation was devised for numbers of up to 213 decimal digits, so that we used a fixed value $m = 6$. Notice that for this choice of m the 2^mary method can be expected to perform considerably less multiplications modulo n than the binary method.)

(3.7) *Remark*. Because of their constant use, we precomputed two tables containing v_i and l_i for all possible values of $u_i \in \{0, 1, \ldots, 2^m - 1\}$.

(3.8) *Remark*. In the sequel, we will use the method described in (3.6) for exponentiations in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$ and $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ as well. The only difference then is that we have to apply other squaring and multiplication routines. The same tables as in (3.7) can be used.

4. The Lucas-Lehmer Test. In this section we present the details of the Lucas-Lehmer test that is used in step (e) of (1.3). As we will see in Section 5, the Lucas-Lehmer test enables us to select fewer q-primes in step (f) of (1.3). Because the Lucas-Lehmer test is relatively fast, compared to the tests in step (i) of (1.3), this can save a lot of computing time. Let l^-, l^+, r^-, r^+, f^-, f^+, be as computed in step (b) of (1.3) the odd prime factors $\leqslant B$, nonfactored parts, and factored parts of $n - 1$ and $n + 1$, respectively.

In rare cases we can even omit the rest of (1.3). This happens if the following condition is satisfied, where B denotes the trial division bound:

$$n < \max(f^-, f^+) \cdot f^- \cdot f^+ \cdot B^3. \tag{4.1}$$

This is a slight refinement of what can be found in the literature, namely (4.1) with n replaced by $2n$ [4, p. 378] (see (4.5)).

For an explanation of the Lucas-Lehmer test as it is formulated here, we refer to the extensive literature on this subject [10]. We need the following two auxiliary tests. By p_i we denote the ith prime number.

(4.2) *Test for* $n - 1$. Let p be an odd, not necessarily prime number dividing $n - 1$, and let prod $\in \mathbf{Z}/n\mathbf{Z}$ be an integer modulo n to be specified in (4.4).

Look for a prime number $x \in \{p_1, p_2, \ldots, p_{50}\}$ such that $x^{(n-1)/p} \not\equiv 1 \bmod n$. If no such x is found, Test (4.2) fails. Otherwise, verify that $x^{n-1} \equiv 1 \bmod n$; if this is not the case, Test (4.2) halts, because n is composite. Otherwise, replace prod by prod $\cdot\, (x^{(n-1)/p} - 1) \bmod n$. If prod $= 0$, then the old value of prod has a nontrivial gcd with n. In this case, Test (4.2) halts, because n is composite; otherwise, report that n passes Test (4.2).

If p is prime then, for those $l > 0$ for which p^l divides t and flag$_{p^l}$ = "true", set $\beta_{p^l}^i = x^{i(n-1)/p^l} \bmod n$ for $i = 0, 1, \ldots, p^l - 1$. (In the Fortran implementation, which allows a maximal value $55440 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$ for t, this may be done for $p^l = 3, 9, 5, 7, 11$.)

This finishes the description of Test (4.2).

(4.3) *Test for* $n + 1$. Let p be a not necessarily prime number dividing $n + 1$, and let prod $\in \mathbf{Z}/n\mathbf{Z}$ be as in (4.2). In this test, computations have to be performed in the ring $A = (\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$, for integers u and a to be specified in

(4.4). (We represent elements of A as $x_0 + x_1\alpha$ where x_0, $x_1 \in \mathbf{Z}/n\mathbf{Z}$ and $\alpha = (T \bmod T^2 - uT - a)$.) How these computations should be carried out is explained in Remark (4.9).

Look for an element $x \in A$ of norm one such that $x^{(n+1)/p} \neq 1$ in the ring A (see Remark (4.10)). If no such x is found after 50 trials, Test (4.3) fails. Otherwise, verify that $x^{n+1} = 1$; if this is not the case, Test (4.3) halts, because n is composite. Otherwise, let $x^{(n+1)/p} - 1 = x_0 + x_1\alpha \in A$. Choose $i \in \{0, 1\}$ such that $x_i \neq 0$, and replace prod by prod $\cdot\, x_i \bmod n$. If prod $= 0$, then the old value of prod has a nontrivial gcd with n. In that case, Test (4.3) halts, because n is composite; otherwise, report that n passes Test (4.3).

This finishes the description of Test (4.3).

(4.4) *Lucas-Lehmer Test*. Set prod $\in \mathbf{Z}/n\mathbf{Z}$ equal to one; in prod we accumulate numbers that should be tested for coprimality with n at the end of the test.

We say that this test fails if it fails itself, or if one of the tests (4.2) or (4.3) fails; in either case, the execution of (4.4) can be terminated. It is also possible that n is proved to be composite during execution of this test or one of the tests (4.2) or (4.3). As soon as that happens, the execution of (4.4) halts. If the test does not fail and if n is not proved to be composite in this test, we say that n passes the Lucas-Lehmer test. In the latter case, it is possible that the primality of n is proved, namely if (4.1) holds (cf. step (f)).

(a) For all primes $p \in I^-$ verify that n passes Test (4.2).

(b) If (4.1) holds (i.e., if n is prime, then the Lucas-Lehmer test will be able to prove it) and if $n - 1$ is not completely factored (i.e., $r^- \neq 1$), verify that n passes Test (4.2) with p replaced by r^-.

(c) Define the ring A that has to be used in Test (4.3) by performing (c1) if $n \equiv 1 \bmod 4$ and (c2) if $n \equiv 3 \bmod 4$.

(c1) Case $n \equiv 1 \bmod 4$. Set $u = 0$. Look for a prime number $a \in \{p_1, p_2, \ldots, p_{50}\}$ such that $a^{(n-1)/2} \equiv -1 \bmod n$. If no such a is found, the Lucas-Lehmer test fails. Otherwise, the ring A is defined as $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - a)$.

For those values of $l \geqslant 1$ for which 2^l divides t and for which $\text{flag}_{2^l} =$ "true" we set, in the course of the above computation, $\beta_{2^l}^i = a^{i(n-1)/2^l} \bmod n$ for $i = 0, 1, \ldots, 2^l - 1$.

(c2) Case $n \equiv 3 \bmod 4$. Set $a = 1$. Look for an integer $u \in \{1, 2, \ldots, 50\}$ such that the Jacobi symbol $(\frac{u^2+4}{n})$ equals -1. If no such u is found, the Lucas-Lehmer test fails. Otherwise, the ring A is defined as $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - 1)$. Verify that $\alpha^{n+1} = -1$ in A; if this is not the case, the Lucas-Lehmer test halts, because n is composite.

(d) For all primes $p \in I^+$ verify that n passes Test (4.3).

(e) If (4.1) holds and if $n + 1$ is not completely factored (i.e., $r^+ \neq 1$), verify that n passes Test (4.3) with p replaced by r^+.

(f) Check that gcd(prod, n) $= 1$. If this is not the case, the Lucas-Lehmer test halts, because a nontrivial divisor of n is found. Otherwise, report that n passes the Lucas-Lehmer test, and if (4.1) holds, report that n is prime.

This finishes the description of the Lucas-Lehmer test.

(4.5) *Remark*. Notice that, by (4.4)(c1) and (4.4)(c2) and [2, (7.24), (10.8)], the Lucas-Lehmer test has also proved that the condition [2, (6.4)] that has to be verified for all primes dividing t holds for $p = 2$ (and if this is not proved, it is shown that n is composite unless the test failed). This easily implies the slight improvement mentioned in connection with (4.1).

It follows from (4.2), (4.3), and [2, Proposition (10.7)] that condition [2, (6.4)] also holds for the odd primes dividing $f^- \cdot f^+$. This explains step (1.3)(g).

(4.6) *Remark.* The flag_{p^k} and $\beta^i_{p^k}$ are kept for later use in step (i) of (1.3). As we have seen in Section 1, $\mathrm{flag}_{p^k} =$ "true" implies that we can replace the Jacobi sum test in $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ by a similar but 'cheaper' test in $\mathbf{Z}/n\mathbf{Z}$ (see [2, Section 10]). A similar speed-up is possible for primes p dividing $n + 1$ and t, but we did not implement that.

(4.7) *Remark.* After execution of the Lucas-Lehmer test, the primes $p \in l^- \cup l^+$ can be removed from the list of candidate q-primes in step (f) of (1.3).

(4.8) *Remark.* The method described in (3.6) can be applied for the exponentiations in the Lucas-Lehmer test. The only difference is that in Test (4.3) and in (4.4)(c2) the squarings and multiplications have to be carried out in the ring A instead of in $\mathbf{Z}/n\mathbf{Z}$ (see (4.9), cf. (3.8)).

(4.9) *Remark.* To be able to carry out the exponentiations in the ring $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$, we need multiplication and squaring routines for elements of this ring. Here we explain how these routines can be implemented. We distinguish the following cases: Multiplication for $n \equiv 1 \bmod 4$ (so $u = 0$), multiplication for $n \equiv 3 \bmod 4$ (so $u \neq 0$ and $a = 1$), and a combined squaring routine for elements of norm one in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$. We also mention how α^{n+1} in (4.4)(c2) can be computed.

—Multiplication for $n \equiv 1 \bmod 4$ in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - a)$. Let $x_0 + x_1\alpha$, $y_0 + y_1\alpha \in (\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - a)$; then $(x_0 + x_1\alpha)(y_0 + y_1\alpha) = (x_0 \cdot y_0 + x_1 \cdot y_1 a) + (x_0 \cdot y_1 + x_1 \cdot y_0)\alpha = z_0 + z_1\alpha$. This is computed in three 'multiple'-'multiple' multiplications instead of four as follows (for an explanation of 'multiple' see Section 7): $p_0 = x_0 \cdot y_0$, $p_1 = x_1 \cdot y_1$, $s_0 = x_0 + x_1$, $s_1 = y_0 + y_1$, and $z_0 = (p_0 + ap_1) \bmod n$, $z_1 = (s_0 \cdot s_1 - p_0 - p_1) \bmod n$.

—Multiplication for $n \equiv 3 \bmod 4$ in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - 1)$. Let $x_0 + x_1\alpha$, $y_0 + y_1\alpha \in (\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - 1)$; then $(x_0 + x_1\alpha)(y_0 + y_1\alpha) = (x_0 \cdot y_0 + x_1 \cdot y_1) + (x_0 \cdot y_1 + x_1 \cdot y_0 + x_1 \cdot y_1 \cdot u)\alpha = z_0 + z_1\alpha$, which is computed by $p_0 = x_0 \cdot y_0$, $p_1 = x_1 \cdot y_1$, $s_0 = x_0 + x_1$, $s_1 = y_0 + y_1$, and $z_0 = (p_0 + p_1) \bmod n$, $z_1 = (s_0 \cdot s_1 + (u - 1)p_1 - p_0) \bmod n$.

—Combined squaring in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$. Because we only need this routine for $x_0 + x_1\alpha \in (\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$ of norm one, we have $(x_0 + x_1\alpha)^2 = (x_0 \cdot x_1 \cdot u + 2x_0^2 - 1) + (x_1^2 \cdot u + 2x_0 \cdot x_1)\alpha = z_0 + z_1\alpha$, as is easily verified. This is computed by $s = ux_1 + 2x_0$, and $z_0 = (x_0 \cdot s - 1) \bmod n$, $z_1 = (x_1 \cdot s) \bmod n$.

—Computation of α^{n+1} in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - 1)$. Although α has norm -1, we can apply the above multiplication and squaring (for elements of norm one) by observing that $\alpha^2 = u\alpha + 1$ has norm one, and that $\alpha^{n+1} = (\alpha^2)^{(n+1)/2}$.

(4.10) *Remark.* To get elements of norm one in $A = (\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - uT - a)$ in Test (4.3), we try elements of the form $(\alpha + m)/(\bar{\alpha} + m) \in A$ for $m \in \{1, 2, \ldots, 50\}$, where $\bar{\alpha}$ denotes the conjugate of α (so $\bar{\alpha} = -\alpha$ if $n \equiv 1 \bmod 4$, and

$\bar{\alpha} = u - \alpha$ if $n \equiv 3 \bmod 4$). It is easily verified that this yields

$$\frac{m^2 + a}{m(m+u) - a} + \frac{(2m+u)\alpha}{m(m+u) - a} \quad \text{for both } n \equiv 1 \bmod 4 \text{ and } n \equiv 3 \bmod 4$$

(notice that $(m(m+u) - a)^{-1}$ can be computed in $\mathbf{Z}/n\mathbf{Z}$ unless n is composite).

(4.11) *Remark*. The number 50 in (4.2), (4.3), and (4.4)(c) is arbitrarily chosen, but in practice is sufficient. See [2, remark preceding (10.4), (11.6)] for a discussion of this point.

(4.12) *Remark*. There are inequalities similar to (4.1) under which *only* the tests for $n - 1$ (Test (4.2)) need to be done, or *only* the tests for $n + 1$ (Test (4.3)). For instance, if $f^- \geqslant n^{1/2}$, then execution of (4.4)(a) suffices to prove the primality of n. If $f^- < n^{1/2}$ but $f^- \cdot B \geqslant n^{1/2}$, then n must also pass Test (4.2) with p replaced by r^-. Similar inequalities hold for $n + 1$.

5. Selection of t **and** s. It follows from [5, Section 8] that the Lucas-Lehmer test can be combined with the primality testing algorithm from [2, Section 12]. Here we describe how this can be done.

Let t be as in (1.1)(a). Assume for the moment that every prime $p \mid t$ satisfies condition [2, (6.4)], i.e.,

(5.1) for every prime divisor r of n there exists a p-adic integer $l_p(r) \in \mathbf{Z}_p$ such that $r^{p-1} = (n^{p-1})^{l_p(r)}$ in the group $1 + p\mathbf{Z}_p$

(where $\mathbf{Z}_p$ denotes the ring of p-adic integers). In (4.5) we have seen that this condition already holds for $p = 2$. For the other primes p dividing t for which we need this condition, a Boolean variable λ_p is declared in step (g) of Algorithm (1.3); as soon as the condition is proved to hold for such a p, we put $\lambda_p =$ "true". On successful termination of Algorithm (1.3) all λ_p will be set to "true", which justifies the above assumption.

For every prime power $p^k \geqslant 2$ dividing t, we define a *cost* $c_{p^k} \in \mathbf{Z}$. This cost c_{p^k} is an estimate (in milliseconds for instance) of the running time needed to perform step (i) of Algorithm (1.3) for p^k and one q-prime with $k = v_p(q-1)$. In step (1.3)(i) the most time will be spent in the uth powering in (1.3)(i2); if $\text{flag}_{p^k} =$ "true" this computation can be done in $\mathbf{Z}/n\mathbf{Z}$ (as in (i2a)), otherwise we work in $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ (as in (i2b)).

Defining c_{p^k}("true") and c_{p^k}("false") as the cost of (i2a) and (i2b), respectively, we set $c_{p^k} = c_{p^k}(\text{flag}_{p^k})$. Both c_{p^k}("true") and c_{p^k}("false") depend on the implementation and the number of binary bits of n, and they are best determined empirically as functions of the number of bits of n (this is what we have done in the Fortran implementation).

Having defined c_{p^k}, we define the cost $w(q)$ of a q-prime as

$$w(q) = \sum_{p \mid q-1,\, k = v_p(q-1)} c_{p^k}.$$

Another function of the number of bits of n that we will need, and that is best determined empirically, is an estimate for the running time needed for one iteration of the final trial division step of Algorithm (1.3) (that is, one execution of (l1), (l2), and (l3) of (1.3)). For a fixed value of n we denote this running time by c_{ftd}. Of course, c_{ftd} is measured in the same units as c_{p^k}.

As in (1.3)(b), let $f^- \cdot f^+$ be the factored part of $n^2 - 1$, and assume that $v_p(f^- \cdot f^+) = v_p(n^2 - 1)$ for the primes p dividing t (this implies that in the Fortran implementation the trial division bound should be at least 11).

(5.2) Let t' be an even divisor of t. Defining

$$s_1 = \left(\tfrac{1}{2}\right) \cdot \prod_{\substack{p \text{ prime} \\ p \mid f^- \cdot f^+}} p^{v_p(t') + v_p(f^- \cdot f^+)},$$

then

(5.3) for all r dividing n we have that $r \equiv n^{l(r)} \bmod s_1$,

where $l(r) \equiv l_p(r) \bmod p^{v_p(t')}$ for all $p \mid t'$. As mentioned in (1.3)(e), this follows from the fact that n passed the Lucas-Lehmer test. Observe also that (5.1) is satisfied for the primes dividing s_1, because of the Lucas-Lehmer test (cf. step (1.3)(g) and Remark (4.5)).

If $s_1 > n^{1/2}$, then (5.3) suffices to prove the primality of n by means of the final trial division (1.3)(l) with t and s replaced by t' and s_1, respectively. If, on the other hand, $s_1 \leqslant n^{1/2}$, let $\tilde{s}_2$ be a product of distinct q-primes such that $q - 1 \mid t'$ and $q \nmid s_1$ (so, these q-primes can be found among the factors of $e(t)$ and are tabulated in (1.1)(a)).

The pseudoprime tests with Jacobi sums as in (1.3) (with t replaced by t'), combined with (5.3), yield

for all r dividing n we have that $r \equiv n^{l(r)} \bmod (s_1 \cdot s_2)$,

where

$$s_2 = \tilde{s}_2 \cdot \prod_{\substack{p \text{ prime} \\ p \mid t',\, p \mid \tilde{s}_2}} p^{v_p(n^{p-1} - 1) + v_p(t') - 1}, \tag{5.4}$$

and $l(r)$ as above. Obviously, in order to be able to prove the primality of n by means of (1.3)(l), we should choose $\tilde{s}_2$ in such a way that $s_1 \cdot s_2 > n^{1/2}$.

We now discuss how $\tilde{s}_2$ should be chosen such that $s_2 > n^{1/2}/s_1$ and $\Sigma_{q \mid \tilde{s}_2} w(q)$ is minimal (where we take the minimum over $\tilde{s}_2$ for which $s_2 > n^{1/2}/s_1$). In [2, Section 4] we have seen that this problem can be formulated as a knapsack problem, which makes an efficient way of finding an optimal solution unlikely to exist. As suggested in [2, Section 4], we approximate an optimal solution in the following way.

First we put

$$\tilde{s}_2 = \prod_{\substack{q \text{ prime} \\ q - 1 \mid t',\, q \nmid s_1}} q,$$

and s_2 as in (5.4). If $s_2 \leqslant n^{1/2}/s_1$, then the current value of t' is too small and (5.2) fails. If, on the other hand, $s_2 > n^{1/2}/s_1$, we proceed as follows. As long as $\tilde{s}_2$ has a prime factor q such that $s_2/q^{v_q(s_2)} > n^{1/2}/s_1$, we choose such a q with $w(q)/\log(q^{v_q(s_2)})$ as large as possible, and replace $\tilde{s}_2$ and s_2 by $\tilde{s}_2/q$ and $s_2/q^{v_q(s_2)}$, respectively.

From (5.2) we get the following algorithm for the selection of t and s.

(5.5) *Selection of t and s.* For all even divisors t' of t do the following:

> Apply (5.2) and compute for those values of t' for which (5.2) does not fail the corresponding approximations $\tilde{s}_2$ (and s_2) to the optimal q-primes choice, and the total cost $c(t') = t' \cdot c_{ftd} + \sum_{q|\tilde{s}_2} w(q)$.

Replace t by the value of t' for which $c(t')$ is minimal, and put $s = s_1 \cdot s_2$, where s_1 and s_2 correspond to the chosen value for t. This finishes the description of (5.5).

(5.6) *Remark.* If we add a test "$r \leqslant n^{1/2}$" in step (l3) of Algorithm (1.3) before the test "$r | n$" (and perform the latter only if the former is satisfied), then we can replace the $t' \cdot c_{ftd}$-term in Algorithm (5.5) by $t' \cdot c_{ftd} \cdot n^{1/2} \cdot s^{-1}$ (where s corresponds to t'). Of course, this slightly increases the value of c_{ftd}.

(5.7) *Remark.* It is possible that Algorithm (5.5) chooses t and $s = s_1 \cdot s_2$ such that there is an odd prime number p dividing t for which $p \nmid q - 1$ for all primes q dividing s_2. It can then be proved that p divides s, with $v_p(s) = v_p(t) + v_p(n^{p-1} - 1)$. Removing $v_p(t)$ factors p from s allows us to remove the same number of factors p from t also. This does not change the set of numbers that are congruent to a power of n modulo s. The resulting value of s, however, may be smaller than $n^{1/2}$, and therefore it might be reasonable to take these s's also into account in Algorithm (5.5).

This complicates step (l3) of Algorithm (1.3), where we will have to trial divide all numbers of the form $r + i \cdot s \leqslant n^{1/2}$ for $i \geqslant 0$, and accordingly change the $t' \cdot c_{ftd}$-term in Algorithm (5.5) into $t' \cdot c_{ftd} \cdot n^{1/2} \cdot s^{-1}$. We did not implement this.

(5.8) *Remark.* The choice of $t = 55440$ guarantees that the Fortran implementation can handle numbers of up to 213 decimal digits. From (5.2) it follows that larger numbers can also be handled if we are able to find enough prime divisors of $n^2 - 1$.

(5.9) *Remark.* With respect to Remark (5.7) we mention the following, not implemented improvement, which is due to H. W. Lenstra, Jr. Instead of choosing $s > n^{1/2}$, we could take $s > n^{1/2}t$, where the factor t may be replaced by any sufficiently large number. We then expect that only one of the t possible divisors of n in step (1.3)(l) is $\leqslant n^{1/2}$. At the cost of one test "$r \leqslant n^{1/2}$" per iteration of (1.3)(l), this saves us most trial divisions.

It is not unlikely that this will prove to be an important improvement for larger values of n than we tested.

6. Pseudoprime Tests with Jacobi Sums.

Let q be a prime number dividing s_2 and let p be a prime number dividing $q - 1$. Here we explain how the pseudoprime tests with Jacobi sums in (1.3)(i) and (1.3)(j), (k) for the pair q, p^k can be performed. So we put $k = v_p(q - 1)$ in case of (1.3)(i), and $k = 1$ in case of (1.3)(j), (k). Let $m = (p - 1)p^{k-1}$.

The computations in (1.3)(i) can all be done in the cyclotomic ring $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$. In (1.3)(i2a), in the case flag$_{p^k}$ = "true", we can work in the subring $\mathbf{Z}/n\mathbf{Z}$ after application of the homomorphism λ. This case will be discussed at the end of this section. First we explain how to compute in $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$, how to handle the inverse of σ_x in (1.3)(i1), and how we implemented (1.3)(i2b).

An element $a = \sum_{i=0}^{m-1} a_i\zeta_{p^k}^i \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ is represented as a vector $(a_i)_{i=0}^{m-1}$, where $a_i \in \{0, 1, \ldots, n-1\}$. Addition and subtraction of two elements of $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ is done by componentwise addition or subtraction modulo n of the corresponding vectors. Multiplication of two elements of $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ can be seen as multiplication of two polynomials of degree less than m with coefficients in $\mathbf{Z}/n\mathbf{Z}$ and modulo the p^kth cyclotomic polynomial $\sum_{i=0}^{p-1} X^{ip^{k-1}}$.

A straightforward implementation would need m^2 integer multiplications, whereas, owing to a theorem of Winograd [4, p. 495], $2m - 1$ integer multiplications suffice. We did not implement Winograd's methods, however, because they involve a large overhead of additional operations. Instead we used special formulae for multiplication and squaring for each p^k, which improve considerably on the m^2-method, but which do not achieve Winograd's $(2m - 1)$ bound for the integer multiplications. In the Appendix, these formulae are given for $p^k = 3, 4, 5, 7, 8, 9, 11, 16$.

Better formulae can certainly be given and the authors would be happy to hear of nonnegligible improvements. For example, in auxiliary routine 3, one 'multiple'-'multiple' multiplication can be gained by noting that the second time auxiliary routine 1 is called, the quantity $a_2 \cdot b_2$ is recomputed. This would gain three such multiplications in the multiplication for $p = 11$, and one in the squaring for $p = 11$.

The formulae in the Appendix have all been obtained by using recursively the identity

$$\begin{aligned}(A_1X + A_0)(B_1X + B_0)\\ &= A_1B_1X^2 + ((A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0)X + A_0B_0,\end{aligned}$$

which uses only three multiplications instead of four. This was combined with trial and error methods to eliminate unnecessary multiplications and, if possible, also some additions or subtractions. (The identity above was already used to compute in $(\mathbf{Z}/n\mathbf{Z})[T]/(T^2 - a)$ for $n \equiv 3 \bmod 4$, see Remark (4.9).) It seems plausible that the number of multiplications in squaring for $p = 7$ can be reduced from 14 to 12 (as for $p^k = 9$). Also, the number of multiplications in squaring for $p = 11$ seems really too high.

The inverse of the automorphism σ_x from (1.3)(i1a) can be computed as follows.

(6.1) *Computation of* σ_x^{-1}. For $a = (a_i)_{i=0}^{m-1} \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ this algorithm computes $b = (b_i)_{i=0}^{m-1} \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ such that $\sigma_x^{-1}(a) = b$.

Let $a_i = 0$ for $i \geqslant m$. First we put, for $i = 0, 1, \ldots, m-1$ in succession, $b_i = a_{xi \bmod p^k}$. Next we replace, for $i = m, m+1, \ldots, p^k - 1$ in succession, $b_{i-jp^{k-1}}$ by $(b_{i-jp^{k-1}} - a_{xi \bmod p^k}) \bmod n$ for $1 \leqslant j < p$.

As a result, we have b such that $\sigma_x(b) = a$.

The small powers of elements of $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ that we need in (1.3)(i1) are computed by repeated multiplication in the same iteration that computes the $j_{0,p,q}$ and $j_{v,p,q}$ (in (1.3)(i1a) and (1.3)(i1d)). The uth power in (1.3)(i2b) clearly should not be done by repeated multiplication. Instead, we use the method described in (3.6) with the squaring and multiplication in $\mathbf{Z}/n\mathbf{Z}$ replaced by the squaring and multiplication in $\mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ (cf. (3.8) and Appendix).

The integer $h \in \{0, 1, \ldots, p^k - 1\}$ in (1.3)(i2b) is determined in the following way.

(6.2) *Determination of h.* For $a = (a_i)_{i=0}^{m-1} \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$ this algorithm determines an integer $h \in \{0, 1, \ldots, p^k - 1\}$ such that $a = \zeta_{p^k}^h$, if such an h exists.

If there exists an integer $l \in \{0, 1, \ldots, m-1\}$ such that $a_l = 1$ and $a_i = 0$ for $0 \leqslant i < m$ and $i \neq l$, or if there exists an integer $l \in \{0, 1, \ldots, p^{k-1} - 1\}$ such that $a_{l+jp^{k-1}} \equiv -1 \bmod n$ for $0 \leqslant j < p - 1$ and $a_i = 0$ for the other indices, then put $h = l$ and (6.2) terminates. Otherwise, h does not exist and (6.2) fails, which implies that, in Algorithm (1.3), n is proved to be composite.

Finally, we discuss what should be done in (1.3)(i2a), in the case that flag_{p^k} = "true". For $a = (a_i)_{i=0}^{m-1} \in \mathbf{Z}[\zeta_{p^k}]/n\mathbf{Z}[\zeta_{p^k}]$, we compute $\lambda(a) = \sum_{i=0}^{m-1} a_i \beta_{p^k}^i \in \mathbf{Z}/n\mathbf{Z}$ by means of a Horner scheme, or by means of the powers $\beta_{p^k}^i$ for $0 \leqslant i < m$, which were computed in (1.3)(e). To raise $\lambda(j_{0,p,q}) \in \mathbf{Z}/n\mathbf{Z}$ to the uth power, we apply (3.6), and determination of h is simply done by comparing $\lambda(j_{0,p,q})^u \cdot \lambda(j_{v,p,q})$ with $\beta_{p^k}^i$ for $0 \leqslant i < p^k$, where of course the equality should hold modulo n.

7. Examples and Running Times. In both our implementations we distinguish between two kinds of fixed-length multiprecision integers, the ordinary 'multiples', and the so-called 'doubles'. The number of binary bits of a 'multiple' should be somewhat larger than the number of binary bits of n, and a 'double' contains twice as many bits as a 'multiple'. Addition and subtraction of two 'multiples' ('doubles') again yields a 'multiple' ('double'), multiplication of two 'multiples' yields a 'double', and remaindering modulo a 'multiple' of a 'multiple', or of a 'double', yields a 'multiple'. For all these operations the classical algorithms (cf. [4]) were used.

In the Pascal program, devised for numbers of up to 104 decimal digits, a 'multiple' ('double') is represented by 8 (16) words of 47 binary bits each; in the Fortran program a 'multiple' ('double') contains 16 (32) words of 47 bits. In Table 1 we give the average running times (in milliseconds) of the elementary arithmetic operations on a CDC 170/750. These routines were written in the assembly language Compass. Notice that the numbers in Table 1 are still too small to get the expected ratios of the running times. For example, for the multiplication we would expect that the 16 word entry takes four times as long as the 8 word entry; instead, we get a ratio $0.21/0.07 = 3$. This is due to overhead cost.

TABLE 1
Average running times of elementary arithmetic operations on the CDC 170/750 *in milliseconds*

'multiple' consists of	8 words of 47 bits	16 words of 47 bits
'multiple' + 'multiple'	0.014	0.019
'multiple' · 'multiple'	0.07	0.21
'double' mod 'multiple'	0.20	0.47

The running times of the various steps of the CDC 170/750 version of the Fortran program are given in Table 2. For each number d in the first row we tested 20 prime numbers of d decimal digits. Each prime was selected by drawing a random number of d digits and using the program to determine the least prime exceeding the number drawn.

For each step of Algorithm (1.3) listed in the first column of Table 2, and for each number of digits d in its first row, the table contains the following data: Average

TABLE 2
Running times of the Fortran program on the CDC 170/750 *in seconds* (*see text*)

number of digits	100	120	140	160	180	200
trial division up to 10^6	7.965	7.972	7.963	7.951	7.973	7.950
	0.039	0.025	0.027	0.047	0.016	0.035
	8.019	8.010	8.022	8.010	7.999	8.000
	7.824	7.887	7.904	7.778	7.926	7.859
four probabilistic compositeness tests	0.567	0.759	0.957	1.292	1.558	1.998
	0.015	0.023	0.029	0.054	0.059	0.127
	0.602	0.803	0.999	1.387	1.680	2.191
	0.544	0.723	0.906	1.181	1.472	1.552
Lucas-Lehmer test	2.211	2.419	3.705	5.086	5.354	6.653
	0.936	0.777	1.547	2.722	2.031	2.214
	3.930	4.348	6.371	12.615	9.494	10.469
	0.724	0.864	0.480	2.147	1.365	2.834
selection of t and s	0.017	0.017	0.016	0.015	0.014	0.015
	0.003	0.003	0.003	0.002	0.002	0.002
	0.023	0.024	0.023	0.019	0.020	0.020
	0.011	0.012	0.012	0.011	0.011	0.012
Jacobi sum tests	37.334	78.151	130.251	205.347	308.475	438.143
	15.696	24.042	42.919	45.350	56.701	80.472
	62.705	113.357	186.919	252.452	392.170	560.381
	12.426	34.503	52.947	64.833	206.021	205.896
additional tests	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
final trial division	2.336	8.468	13.525	26.501	36.341	40.978
	1.379	7.062	5.257	8.301	0.658	1.606
	6.216	27.571	28.782	33.927	37.930	43.292
	1.099	2.422	2.546	16.045	35.280	35.761
total running time	50.442	97.797	156.429	246.204	359.728	495.748
	15.203	28.274	43.122	44.144	55.833	80.025
	75.416	147.259	210.756	298.144	439.039	614.254
	26.031	51.077	77.316	111.888	259.021	258.859

running time $\bar{t} = (\Sigma_{i=1}^{20} t_i)/20$, the sample standard deviation $((\Sigma_{i=1}^{20}(t_i - \bar{t})^2)/19)^{1/2}$, the maximal running time, and the minimal running time. All times are in seconds. For running times of the Pascal program we refer to [2, Table 3]. Notice that in Table 2 the average time to do the trial division does not depend on the size of the dividend. This is because, at the time we made the table, our 'multiple'-'single' division routine did not care about leading zeros, and because the Compass routines are written for numbers in fixed multiprecision (16 words of 47 bits in Table 2). This is, of course, quite inefficient, and the running times given in Table 2 could be easily improved by making the precision vary with the number of digits of n.

The Fortran program was used to prove the primality of some of the numbers of the Cunningham tables [3], which were not yet proved to be prime. To illustrate the primality testing algorithm (1.3) we will go through the primality proof for one of these numbers, namely

$$\begin{aligned} n = {} & 387650433531799750146939103531910970866358962518 06\\ & 2302982289092672371151411524515556647925609871 7968\\ & 3104968360539125133039103105418470259112815585 8755\\ & 9700056356937703949226241396723616837470247248 1350\\ & 48208451745439902122005282381436679587515252273, \end{aligned}$$

being one of the factors of $2^{892} + 1$. (To handle this number, which has 247 decimal digits, we used 'multiples' of 24 words of 47 bits; as a consequence, the basic operations became somewhat slower.)

Of course, we cannot guarantee beforehand that the Fortran program, with a maximal value of 55440 for t, will be able to prove the primality of this number, because $n > N$ (cf. (1.1)(a)). In several respects, however, n appears to be a lucky number. The running times below are on a CDC 170/750.

After verification of (1.3)(a), we performed (2.1) with $B = 10^6$. After 8755 milliseconds we found $l^- = \{7, 223, 2017, 4001, 162553\}$ and $l^+ = \{3, 19, 367\}$. Because n was already declared to be 'probably prime' in the Cunningham tables, we did not perform any probabilistic compositeness test (3.4), so $m = 0$ in (1.3)(c) (cf. (3.5)).

In (1.3)(d) we found flag_3 = "false", flag_4 = "true", flag_5 = "false", flag_7 = "true", flag_8 = "true", flag_9 = "false", flag_{11} = "false", flag_{16} = "true". This implies that the Jacobi sum tests are relatively cheap for $p^k = 4, 7, 8, 16$. The Lucas-Lehmer test (4.4) for the primes in $l^- \cup l^+ \cup \{2\}$ took 14679 milliseconds. Because many prime divisors of $n^2 - 1$ were found, all remaining q-primes (that is, the q-primes except 2, 3, 7, and 19) just appeared to be sufficient to get $s_1 \cdot s_2 > n^{1/2}$. The distinct primes dividing s_2 are

$$\begin{aligned} \{ & 5, 11, 13, 17, 23, 29, 31, 37, 41, 43, 61, 67, 71, 73, 89, 113, 127, 181,\\ & 199, 211, 241, 281, 331, 337, 397, 421, 463, 617, 631, 661, 881,\\ & 991, 1009, 1321, 2311, 2521, 3697, 4621, 9241, 18481, 55441\}. \end{aligned}$$

The corresponding t value is 55440. In (1.3)(g) all λ_p for $p \mid t$ were found to be "true" already. The pseudoprime tests with Jacobi sums in (1.3)(h)&(i) were performed in 806940 milliseconds. We list some typical timings (in seconds) in Table 3.

The additional tests in (1.3)(j)&(k) do not have to be performed, because the λ_p were already "true" in (1.3)(g); notice that λ_p = "true" also follows from the h

TABLE 3
Running times of Jacobi sum tests on the CDC 170/750 *in seconds*

p^k	q	running time of (1.3)(i1)	h in (1.3)(i2)
3	13	2.079	1
4	13	0.986	1
2	23	0.975	0
11	23	26.256	8
5	41	5.427	3
8	41	1.019	4
7	1009	1.118	5
9	1009	9.908	0
16	1009	1.164	11

values for $p = 3, 5, 7, 11$ in Table 3 (cf. (1.3)(i3)). The 55440 trial divisions in (1.3)(l) took 56296 milliseconds. It follows that the primality proof for this n was completed within 15 minutes.

We conclude this section by listing in Table 4 the running times (in seconds) of the Fortran program when executed on CDC 170/750, CDC 205, and Cray 1, and applied to

$$\begin{aligned} n = {} & 339549724935349607481986319204055049743924044985 99\\ & 702177572561409137820040418618554524643093152503 80\\ & 597793344033094834542260922844183825913373096203 64\\ & 938100840903721641622176153759 \end{aligned}$$

(this is one of the 180-digit primes that were used for Table 2).

TABLE 4

	running time	'multiple' represented as
CDC 170/750	378.007	16 words of 47 bits
CDC 205	590.623	32 words of 24 bits
Cray 1	196.544	32 words of 24 bits

Obviously, the architecture of the Cray 1 is better suited for computations on integers of this size than the CDC 205. To take full advantage of the vector registers of the CDC 205, much longer vectors should be used, whereas the Cray 1 is designed to handle vectors of length 64 (which are, in our case, the 'doubles').

Acknowledgments are due to H. W. Lenstra, Jr. for his great help in writing this paper, and to D. T. Winter for writing the multiprecision routines.

Université de Bordeaux I
Talence, France

Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

Department of Computer Science
The University of Chicago
Chicago, Illinois 60637

1. L. M. ADLEMAN, C. POMERANCE & R. S. RUMELY, "On distinguishing prime numbers from composite numbers," *Ann. of Math.*, v. 117, 1983, pp. 173–206.

2. H. COHEN & H. W. LENSTRA, JR., "Primality testing and Jacobi sums," *Math. Comp.*, v. 42, 1984, pp. 297–330.

3. J. BRILLHART, D. H. LEHMER, J. L. SELFRIDGE, B. TUCKERMAN & S. S. WAGSTAFF, JR., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ Up to High Powers*, Contemp. Math., vol. 22, Amer. Math. Soc., Providence, R. I., 1983.

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

5. H. W. LENSTRA, JR., *Primality Testing Algorithms* (*after Adleman, Rumely and Williams*), Séminaire Bourbaki, v. 33, 1981, pp. 243–257; in Lecture Notes in Math., vol. 901, Springer-Verlag, Berlin, 1981.

6. H. W. LENSTRA, JR., "Divisors in residue classes," *Math. Comp.*, v. 42, 1984, pp. 331–340.

7. H. W. LENSTRA, JR. & R. TIJDEMAN, *Computational Methods in Number Theory*, Mathematical Centre Tracts 154, 155, Mathematisch Centrum, Amsterdam, 1982.

8. M. O. RABIN, "Probabilistic algorithms for primality testing," *J. Number Theory*, v. 12, 1980, pp. 128–138.

9. R. SOLOVAY & V. STRASSEN, "A fast Monte-Carlo test for primality," *SIGACT News*, v. 6, 1977, pp. 84–85; erratum, *ibid.*, v. 7, 1978, p. 118.

10. H. C. WILLIAMS, "Primality testing on a computer," *Ars Combin.*, v. 5, 1978, pp. 127–185.