IMPLEMENTING THE ASYMPTOTICALLY FAST VERSION OF THE ELLIPTIC CURVE PRIMALITY PROVING ALGORITHM

F. MORAIN

Abstract. The elliptic curve primality proving (ECPP) algorithm is one of the current fastest practical algorithms for proving the primality of large numbers. Its running time currently cannot be proven rigorously, but heuristic arguments show that it should run in time $\tilde{O}((\log N)^5)$ to prove the primality of N. An asymptotically fast version of it, attributed to J. O. Shallit, is expected to run in time $\tilde{O}((\log N)^4)$. We describe this version in more details, leading to actual implementations able to handle numbers with several thousands of decimal digits.

1. Introduction

From the work of Agrawal, Kayal and Saxena [2], determining the primality of an integer N can be done in proven deterministic polynomial time $\tilde{O}((\log N)^{10.5})$. More recently, H.-W. Lenstra, Jr. and C. Pomerance have announced a version in $\tilde{O}((\log N)^6)$ (see [31]). Building on the work of P. Berrizbeitia [7], and reusing classical cyclotomic ideas that originated in the Jacobi sums test [1, 11], D. Bernstein [6] and P. Mihăilescu & R. Mocenigo [34], independently, have given improved probabilistic versions with a claim of proven complexity of $\tilde{O}((\log N)^4)$. For more on primality before AKS, we refer the reader to [14] (see also [36]). For the recent developments, see [5].

All the known versions of the AKS algorithm are for the time being too slow to prove the primality of large explicit numbers. On the other hand, the elliptic curve primality proving algorithm [3] has been used for years to prove the primality of ever larger numbers*. It is sketched in [30] that

Conjecture 1.1. *The cost of ECPP is* $\tilde{O}((\log N)^5)$.

The same article describes an asymptotically fast version of ECPP, attributed to J. O. Shallit and that we will call fastECPP:

Conjecture 1.2. *The cost of fastECPP is* $\tilde{O}((\log N)^4)$.

The aim of the present paper is to describe fastECPP, give a heuristic analysis of it and describe its implementation.

Section 2 collects some well-known facts on imaginary quadratic fields, that can be found for instance in [13]. Section 3 presents the basic ECPP algorithm and analyzes it. In Section 4, the fast version is described and its complexity estimated. Section 5 explains the implementation and Section 6 gives some actual timings on large numbers. All estimates make use of the $\tilde{O}$ notation. Remember that $\tilde{O}(f)$ means $O(f(\log f)^{O(1)})$.

Date: October 21, 2005.

Key words and phrases. Primality proving, elliptic curves, ECPP algorithm.

Projet TANC, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École polytechnique, INRIA, Université Paris-Sud. The author is on leave from the French Department of Defense, Délégation Générale pour l'Armement.

*See the web page of M. Martin, `http://www.ellipsa.net/`, or that of the author, `http://www.lix.polytechnique.fr/Labo/Francois.Morain` that also contains binaries of his program.

2. Quadratic fields

A discriminant $-D < 0$ is said to be fundamental if and only if D is free of odd square prime factors, and moreover $D \equiv 3 \bmod 4$ or $D \bmod 16 \in \{4, 8\}$. The quantity

$$\mathcal{D}(X) = \#\{D \le X, -D \text{ is fundamental}\}$$

is easily seen to be $O(X)$.

A fundamental discriminant may be written uniquely (modulo permutations) as:

$$-D = \prod_{i=1}^{t} q_i^*$$

where q_i^* is either -4 or ± 8, or $q_i^* = (-1/q_i)q_i$ for any odd prime q_i. (We denote by $\left(\frac{a}{p}\right)$ the Legendre symbol.) The number of genera is $g(-D) = 2^{t-1}$ and Gauss proved that this number divides the class number $h(-D)$ of $\mathbf{K} = \mathbb{Q}(\sqrt{-D})$. Moreover, Landau (see e.g., [37, Thm 4.4, p. 143]) proved that $h = O(D^{1/2} \log D)$.

The rational prime p is the norm of an integer in $\mathbf{K}$, or equivalently, $4p = U^2 + DV^2$ in rational integers U and V if and only if the ideal (p) splits completely in the Hilbert class field of $\mathbf{K}$, denoted $\mathbf{K}_H$, an extension of degree $h(-D)$ of $\mathbf{K}$. The probability that a prime p splits in $\mathbf{K}_H$ is $1/(2h(-D))$.

Using Gauss's theory of genera of forms, it is known that if $\left(\frac{q_i^*}{p}\right) = 1$ for all i (equivalently, (p) splits in the genus field of $\mathbf{K}$), then the probability of (p) splitting in $\mathbf{K}_H$ is $g(-D)/h(-D)$.

3. The basic ECPP algorithm

We present a rough sketch of the ECPP algorithm, enough for us to estimate its complexity. We do not insist on what happens if one of the steps fails, revealing the compositeness of N. More details can be found in [3]. Without loss of generality, N is supposed to be greater than 3.

3.1. **Elliptic curves over $\mathbb{Z}/N\mathbb{Z}$.** For us, an elliptic curve E modulo N is an equation $Y^2 = X^3 + aX + b$, a, $b \in \mathbb{Z}/N\mathbb{Z}$ with $\gcd(4a^3 + 27b^2, N) = 1$ and we will let

$$E(\mathbb{Z}/N\mathbb{Z}) = \{(x : y : z) \in \mathbb{P}^2(\mathbb{Z}/N\mathbb{Z}), y^2 z \equiv x^3 + axz^2 + bz^3\} \cup \{O_E = (0 : 1 : 0)\}$$

which ressembles the definition of an actual elliptic curve if N is prime, $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$ being the projective plane over $\mathbb{Z}/N\mathbb{Z}$. The important point here is that if p is a divisor of N, we can consider E as a curve E_p over $\mathbb{Z}/p\mathbb{Z}$ and reduce a point $P \in E(\mathbb{Z}/N\mathbb{Z})$ to $P_p \in E_p(\mathbb{Z}/p\mathbb{Z})$. Moreover, we can define an operation on $E(\mathbb{Z}/N\mathbb{Z})$, called *pseudo-addition*, that adds two "points" P and Q with the usual chord-and-tangent law. This operation either yields a point R or a divisor of N if any is encountered when dividing. If R exists, then it has the property that R_p is the sum of P_p and Q_p on E_p for all prime factors p of N. Note also that O_E reduces to the ordinary point at infinity on E_p.

We will need to exponentiate points in E. This is best defined using the division polynomials (see for instance [4] for a lot of properties on these). Remember that over a field K there exists polynomials $\phi_m(X,Y)$, $\psi_m(X,Y)$, $\omega_m(X,Y)$ such that, if $P = (X : Y : 1)$,

$$[m]P = \underbrace{P + \cdots + P}_{m \text{ times}} = \left(\phi_m(X,Y)\psi_m(X,Y) : \omega_m(X,Y) : \psi_m^3(X,Y)\right). \tag{1}$$

All these polynomials can be computed via recurrence formulas and there is a $O(\log m)$ algorithm for this task (a variant of the usual binary method for exponentiating). A fundamental property is the following:

Theorem 3.1. *Let $\overline{K}$ stands for an algebraic closure of K. Let m be an integer. If $P = (x : y : 1) \in E(\overline{K})$ is such that $[2]P \neq O_E$, then P is of m torsion if and only if $\psi_m(x, y) = 0$.*

For the sake of presenting the algorithm in a simplified setting, we prove (compare [27]):

Proposition 3.1. *Let N' a prime satisfying $(\sqrt{N} - 1)^2 \leq 2N' \leq (\sqrt{N} + 1)^2$. Suppose that E is a curve modulo N, that $P = (x : y : 1) \in E(\mathbb{Z}/N\mathbb{Z})$ is such that* $\gcd(y, N) = 1$, $\psi_{2N'}(x, y) \equiv 0 \bmod N$ *but* $\gcd(\psi_{N'}(x, y), N) = 1$. *Then N is prime.*

Proof: suppose that N is composite and that $p \leq \sqrt{N}$ is one of its prime factors. Let us look at what happens modulo p. Since $p \nmid y$, P_p is not a 2-torsion point on E_p. Since $\psi_{N'}(x, y)$ is invertible modulo p, then $[N'](P_p) \neq O_{E_p}$ and therefore P_p has order $2N'$ on E_p. This is impossible, since $2N' \geq (\sqrt{N} - 1)^2 \geq (p - 1)^2 > (\sqrt{p} + 1)^2 \geq \#E_p$ by Hasse's theorem. $\square$

3.2. **Presentation of the algorithm.** We want to prove that N is prime. The algorithm runs as follows:

[Step 1.] Repeat the following: Find an imaginary quadratic field $\mathbf{K} = \mathbb{Q}(\sqrt{-D})$ of discriminant $-D$, $D > 0$, such that

$$4N = U^2 + DV^2 \tag{2}$$

in rational integers U and V. For all solutions U of (2), compute $m = N + 1 - U$; if one of these numbers is twice a probable prime N', go to Step 2.

[Step 2.] Build an elliptic curve $\overline{E}$ over $\overline{\mathbb{Q}}$ having complex multiplication by the ring of integers $\mathcal{O}_K$ of $\mathbf{K}$.

[Step 3.] Assuming N is prime, it splits completely in $\mathbf{K}_H$. Reduce $\overline{E}$ modulo a prime divisor of (N) in $\mathbf{K}_H$, to get a curve modulo N.
[Step 4.] Find $P = (x : y : 1)$, $\gcd(y, N) = 1$ on E such that $\psi_{2N'}(x, y) = 0$, but $\gcd(\psi_{N'}(x, y), N) = 1$. If this cannot be done, then N is composite, otherwise, it is prime by Proposition 3.1.

[Step 5.] Set $N = N'$ and go back to Step 1.

3.3. **Analyzing ECPP.** We will now analyze all steps of the above algorithm and give complexity estimates using the parameter $L = \log N$. One basic unit of time will be the time needed to multiply two integers of size bounded by L, namely $O(L^{1+\mu})$, where $0 \leq \mu \leq 1$ ($\mu = 1$ for ordinary multiplication, or any $\epsilon > 0$ for fast multiplication).

Clearly, we go through Step 5 $O(L)$ times for proving the primality of N. We consider all steps, one at a time, easier steps first.

3.4. **Analysis of Step 4.** Finding a point P can be done by a simple algorithm that looks for the smallest x such that $x^3 + ax + b$ is a non-zero square modulo N and then extracting a squareroot modulo N, for a cost of $O(L^{2+\mu})$, using for instance the Tonelli-Shanks algorithm (or Lehmer's algorithm when $N - 1$ is divisible by a large power of 2). Using the trick described in [3, §8.6.3], we do not need the modular square root, though...

Computing $\psi_{N'}(x, y)$ costs $O(L^{2+\mu})$, and we need $O(1)$ points on average, so this steps amounts for $O(L^{2+\mu})$.

3.5. **Analyzing Step 2.** The original version is to realize $\mathbf{K}_H/\mathbf{K}$ via the computation of the minimal polynomial $H_D(X)$ of the special values of the classical j-invariant at quadratic integers. More precisely, we can view the class group $Cl(-D)$ of $\mathbf{K}$ as a set of h primitive reduced quadratic forms of discriminant $-D$. If (A, B, C) is such a form, with $B^2 - 4AC = -D$, then

$$H_D(X) = \prod_{(A,B,C)\in Cl(-D)} \left(X - j((-B + \sqrt{-D})/(2A))\right).$$

It is known that $H_D(X) \in \mathbb{Z}[X]$. In [16, 15], it is argued that the logarithmic height of this polynomial (logarithm of the sup norm) is well approximated by the quantity $\pi\sqrt{D}S(D)$ where:

$$S(D) = \sum_{[A,B,C]\in Cl(-D)} \frac{1}{A}.$$

It can be shown that $S(D) = O((\log h)^2)$.

Evaluating the complex roots of $H_D(X)$ and building this polynomial can be done in $\tilde{O}(D)$ operations (see [15]). Note that this step does not require computations modulo N.

Alternatively, we could use the method of [12, 8] for computing the class polynomial and get a proven running time of $\tilde{O}(h^2)$, but assuming a suitable Riemann hypothesis.

3.6. **Analyzing Step 3.** To get E modulo N, we need a root of $H_D(X)$ modulo N. This can be done with the Cantor-Zassenhaus algorithm (see [23, §14.3] for instance). Briefly, we split recursively $H_D(X)$ by computing $\gcd((X + a)^{(N-1)/2} - 1, H_D(X)) \bmod N$ for random a's.

Computing $(X+a)^{(N-1)/2} \bmod (N, H_D(X))$ costs $O(L\mathsf{M}(N, h))$ where $\mathsf{M}(N, d)$ is the time needed to multiply two degree d polynomials modulo N. A gcd of two degree d polynomials costs $\mathsf{M}(N, d) \log d$ (see [23, Ch. 11]).

We expect a splitting after $O(1)$ trials; after each splitting, we carry on with the factor of lowest degree. Since the degree is divided by at least 2 after each splitting, we get a root after $O(\log h)$ steps, but the overall cost is dominated by the first step, for a cost of

$$O(\mathsf{M}(N, h)(L + \log h)).$$

We can assume that $\mathsf{M}(N, d) = O(d^{1+\nu} L^{1+\mu})$ where again $0 \leq \nu \leq 1$.

3.7. **Analysis of Step 1.** This is the crucial step. Given D, testing whether (2) is satisfied involves the reduction of the ideal $(N, \frac{r-\sqrt{-D}}{2})$ that lies above (N) in $\mathbf{K}$, where $r^2 \equiv -D \bmod (4N)$ (if N is prime...). This requires the computation of $\sqrt{-D} \bmod N$ as in Step 4, i.e., a $O(L^{2+\mu})$ time. Then it proceeds with Cornacchia's algorithm, akin to a half gcd (see 5.2), for a cost of $O(L^{1+\mu})$.

In the event that equation (2) is solvable, then we need to check that $m = 2N'$ and test N' for probable primality, which costs again some $O(L^{2+\mu})$.

The heuristic probability of m being of the given form is $O(1/L)$. Though quite realistic, this estimate is out of reach of current techniques in analytical number theory. Using this heuristic, we expect to need $O(L)$ discriminants D. Let us take all discriminants less than $D_{\max}$. Since N splits completely in $\mathbf{K}_H$ with probability $1/(2h(-D))$, we expect to get a useful m provided

$$\sum_{D\leq D_{\max}} \frac{1}{2h(-D)} \gg L.$$

The left hand side dominates

$$\sum_{D\leq D_{\max}} \frac{1}{\sqrt{D}\log D} \gg \frac{\sqrt{D_{\max}}}{\log D_{\max}},$$

so $D_{\max} = O((L \log L)^2) = \tilde{O}(L^2)$ should suffice.

Turning to complexity, Step 1 then solves $\tilde{O}(L^2)$ equations of type (2), followed by $\tilde{O}(L)$ probable primality tests:

$$\tilde{O}(L^2(\underbrace{L^{2+\mu}}_{\sqrt{-D} \bmod N} + \underbrace{L^{1+\mu}}_{\text{reduction}})) + \tilde{O}(L \cdot \underbrace{L^{2+\mu}}_{\text{probable primality}}).$$

which is dominated by the first cost, namely $\tilde{O}(L^{4+\mu})$.

3.8. **Adding everything together.** Taking $D = \tilde{O}(L^2)$ readily implies $h = \tilde{O}(L)$. The cost of Step 2 is $\tilde{O}(L^2)$, and that of Step 3 is $\tilde{O}(L^{3+\mu+\nu})$, which dominates Step 4. All in all, we get that ECPP has heuristic complexity $\tilde{O}(L^{4+\mu})$ for one step, and therefore $\tilde{O}(L^{5+\mu})$ in totality.

3.9. **Remark.** In practice, the dominant term of the complexity of Step 1 is $O(n_D L^{2+\mu})$ where n_D is the number of D's for which we solve equation (2). Depending on implementation parameters and real size of N, this number n_D can be quite small. This gives a very small *apparent complexity* to ECPP, somewhere in between L^3 and L^4 and explains why ECPP seems so fast in practice (see for instance [22]).

4. The fast version of ECPP

4.1. **Presentation.** When dealing with large numbers, all the time is spent in finding the discriminants D, which means that a lot of squareroots modulo N must be computed. A first way to reduce the computations, alluded to in [3, §8.4.3], is to accumulate squareroots, and reuse them, at the cost of some multiplications. For instance, if one has $\sqrt{-3}$ and $\sqrt{5} = \sqrt{-20}/\sqrt{-4}$, then we can build $\sqrt{-15}$, etc.

A better way that leads to the fast version consists in computing a basis of small squareroots and build discriminants from this basis. From the analysis carried out above, we need $\tilde{O}(L^2)$ discriminants to find a good one. The basic version finds them by using all discriminants that are of size $\tilde{O}(L^2)$. As opposed to this, one can build those discriminants as $-D = (-p)(q)$, where p and q are taken from a pool of size $\tilde{O}(L)$ primes.

More formally, we replace Step 1. by Step 1'. as follows:

[Step 1'.]

1.1. Find the $r = \tilde{O}(L)$ smallest primes q such that $\left(\frac{q^*}{N}\right) = 1$, yielding $\mathcal{Q} = \{q_1^*, q_2^*, \ldots, q_r^*\}$.
1.2. Compute all $\sqrt{q^*} \bmod N$ for $q^* \in \mathcal{Q}$.
1.3. For all pairs $(q_{i_1}^*, q_{i_2}^*)$ of $\mathcal{Q}$ for which $q_{i_1}^* q_{i_2}^* = -D < 0$, solve equation (2).

The cost of this new Step 1 is that of computing $r = \tilde{O}(L)$ squareroots modulo N, for a cost of $\tilde{O}(L \cdot L^{2+\mu})$. Then, we still have $\tilde{O}(L^2)$ reductions. The new overall cost of this phase decreases now to:

$$\tilde{O}(L \cdot \underbrace{L^{2+\mu}}_{\text{squareroots}}) + \tilde{O}(L^2 \cdot \underbrace{L^{1+\mu}}_{\text{reduction}}) + \tilde{O}(L \cdot \underbrace{L^{2+\mu}}_{\text{probable primality}}) = \tilde{O}(L^{3+\mu}).$$

Note here how the complexity decomposes as $3 = 1 + 2$ or $2 + 1$ depending on the sub-algorithms.

Putting everything together, we see that this time the dominant step in terms of complexity is that of Step 3 and therefore we end up with a total cost of $\tilde{O}(L^{4+\mu+\nu})$ for this variant of ECPP and $\tilde{O}(L^4)$ asymptotically.

4.2. **Remarks.**

4.2.1. *Complexity issues.* We can reduce further the number of square root computations by using all subsets of $\mathcal{Q}$ and not only pairs of elements. This would call for $r = \tilde{O}(\log\log N)$, since then $2^r = L^2$ could be reached. Though useful in practice, this phase no longer dominates the cost of the algorithm.

Moreover, we can see that several phases of fastECPP have cost $\tilde{O}(L^3)$, which means that we would have to fight hard to decrease the overall complexity below $\tilde{O}(L^4)$.

4.2.2. *A note on discriminants.* Note that we use fundamental discriminants only, as non fundamental discriminants lead to curves that do not bring anything new compared to fundamental ones. Indeed, if $\mathcal{D} = f^2 D$, with D fundamental, then there is a curve having CM by the order of discriminant $\mathcal{D}$. Writing $4N = U^2 + Df^2V^2$, its cardinality is $N + 1 - U$, the same as the corresponding curve associated to D.

4.2.3. *A note on class numbers.* As soon as we use composite discriminants $-D$ of the form $q_{i_1}^* q_{i_2}^*$, Gauss's theorem tells us that the class number $h(-D)$ is even. This could introduce a bias in our estimation, but we conjecture that the effect is not important.

5. Implementation

5.1. **Computing class numbers.** In order to make the search for $D \in \mathcal{D}$ efficient, it is better to control the class number beforehand. Tables can be made, but for larger computations, we need a fast way to compute $h(-D)$. Subexponential methods exist, assuming the Generalized Riemann Hypothesis. From a practical point of view, our D's are of medium size. Enumerating all forms costs $O(h^2)$ with a small constant, and Shanks's baby-steps/giant-steps algorithm costs $O(\sqrt{h})$ but with a large constant. It is better here to use the explicit formula of Louboutin [32] that yields a practical method in $O(h)$ with a very small constant.

5.2. **An improved Cornacchia algorithm.** Step 1 needs squareroots to be computed, then performs Cornacchia's algorithm. Briefly, the algorithm runs as follows (see [38]):

procedure Cornacchia(d, p, t)
  {t is such that $t^2 \equiv -d \bmod p$, $p/2 < t < p$ }
  a) $r_{-2} = p$, $r_{-1} = t$; $w_{-2} = 0$, $w_{-1} = 1$;
  b) **for** $i \geq 0$ **while** $r_{i-1} > \sqrt{p}$ **do**
    compute (a_i, r_i) such that $r_{i-2} = a_i r_{i-1} + r_i$, $0 \leq r_i < r_{i-1}$;
    let $w_i = w_{i-2} + a_i w_{i-1} \quad (*)$;
  c) **if** $r_{i-1}^2 + dw_{i-1}^2 = p$ **then** return (r_{i-1}, w_{i-1}) **else** return $\emptyset$.

We end the **for** loop once $r_{i-1} \leq \sqrt{p} < r_{i-2}$. As is well known, the a_i's are quite small and we can guess their size by monitoring the number of bits of the r_i's, thus limiting the number of long divisions. One can use a fast variant for this half gcd if needed, in a way reminiscent of Knuth.

Moreover, we know that this algorithm almost always returns the empty set in step 2c), since the probability of success if $1/(2h(-d))$. Therefore, when h is large, we can dispense of the multiprecision computations in equation $(*)$. We replace it by single precision computations in base 2^b ($b = 32$ or $b = 64$):

$$w_i = w_{i-2} + a_i w_{i-1} \bmod 2^b$$

and at the end, we test whether $r_{i-1}^2 + dw_{i-1}^2 = p \bmod 2^b$. If this is the case, then we recompute the w_i's and check again.

5.3. **Factoring m.** In practice, trying to factor $m = 2N'$, for a probable prime N', is too restrictive; instead, we check whether $m = cN'$ for some B-smooth number c. The parameter B is critical.

As shown in [20], the number of probable prime tests we will have to perform is heuristically $t = O((\log N)/(\log B))$ and we will end up with N' such that $N/N' \approx \log B$.

For small N, we can factor lots of m doing the following. In a first step, we compute

$$r_i = (N+1) \bmod p_i$$

for all $p_i \leq B$, which costs $\pi(B) L \log B$, where $\pi(B) = O(B/\log B)$ is the number of primes below B and the other term being the time needed to divide a multi-digit number by a single digit number.

Then, sieving both $m = N + 1 - U$ and $m' = N + 1 + U$ is done by computing $u_i = U \bmod p_i$ and comparing it to $\pm r_i$ for primes p_i such that $(-D/p_i) \neq -1$. See [3, 35] for more details and tricks.

The cost of this algorithm for t values of m is

$$O(BL) + O(t \cdot BL)$$

where the second term comes from the computation of $U \bmod p_i$, which is roughly twice as fast as that of $(N+1) \bmod p_i$, since U is $O(\sqrt{N})$. Since we need to perform also t probable prime tests (say, a plain Fermat one), then the cost is

$$O(tBL) + O(tL^{2+\mu}) = O(BL^2) + O(L^{3+\mu})$$

and therefore the optimal value for B is $B = O(L)$.

When N is large, it is better to use the stripping factor algorithm in [20], for a cost of $O(B(\log B)^2)$, the optimal value of B being $B = O(L^3)$.

5.3.1. *Remark.* Suppose we have found N' and that $m = N + 1 - U = cN'$. Then we will have to compute

$$r'_i = (N'+1) \bmod p_i = (r_i - u_i)/c + 1 \bmod p_i.$$

Computing the right hand side is faster, since c is ordinarily small compared to N'.

5.3.2. *Using an early abort strategy.* This idea is presented in [20]. We would like to go down as fast as possible. So why not impose N/N' greater than some given bound? Candidates N' need be tested for probable primality only if this bound is met. From what has been written above, we can insist on $N/N' \approx \log B$. Our implementation introduces a parameter δ and imposes $N/N' \geq 2^\delta$.

5.3.3. *Using new invariants.* Tackling larger and larger N's forces us to use larger and larger D's, leading to larger and larger polynomials H_D. For this to be doable, new invariants had to be used, minimizing the height of their minimal polynomials. This task was done using Schertz's formulation of Shimura's reciprocity law [39], with the invariants of [18, 16] (alternatively see [25, 24]). Note that replacing j by other functions does not the change the complexity of the algorithm, though it is crucial in practice.

5.3.4. *Step 3 in practice.* We already noted that this step is the theoretically dominating one in fastECPP, with a cost of $\tilde{O}(L^{3+\mu+\nu})$. In practice, even for small values of h, we can assume $\nu \approx 0$ (using for instance the algorithm of [33] for polynomial multiplication).

One way to reduce the cost of this phase in practice is to use smooth values of h, and use Galois theory to factor $H_D(X)$ over a tower of extensions. Then, we replace the factorization of a degree h polynomial by a list of smaller ones, the largest prime factor of h being $\log h$. We already used that in ECPP, using [29, 17]. Typical values of h are now routinely in the 10000 zone.

It could be argued that keeping only smooth class numbers is too restrictive. Note however, that class numbers tend to be smoother than ordinary numbers [10].

5.4. **fastECPP.** We give here the expanded algorithm corresponding to step 1'. Using a smoothness bound B, we need approximately $t = \exp(-\gamma)\log N/\log B$ values of m and therefore roughly $t/2$ discriminants. The probability that D is a splitting discriminant is $g(-D)/h(-D)$. Therefore we build discriminants until

$$\sum_D g(-D)/h(-D) \approx t/2.$$

One way of building these discriminants is the following: we let r increase and build all or some of the subsets of $\{q_1^*, \ldots, q_r^*\}$ until the expected number of D's is reached. After this, we sort the discriminants with respect to $(h(-D)/g(-D), h(-D), D)$ and treat them in this order.

6. Benchmarks

First of all, it should be noted that ECPP is not a well defined algorithm, as long as one does not give the list of discriminants that are used, or the principles that generate them.

Since the first phase of ECPP requires a tree search, testing on a single number does not reveal too much. Averaging on more than 20 numbers is a good idea.

Our current implementation uses GMP [26] for the basic arithmetic, which enables one to use `mpfr` [28] and `mpc` [19], thus leading to a complete program that can compute polynomial H_D's on the fly, contrary to the author's implementation of ECPP, prior to version 11.0.5. This turned out to be the key for the new-born program to compete with the old one.

We give below some timings obtained with our new implementation, after a lot of trials. We used as prime candidates the first twenty primes with 1000, 1500, and 2000 decimal digits. Critical parameters are as follows: we used $D \leq 10^7$, $h \leq 1000$, $\delta = 12$ (see section 5.3.2). For 1000 and 1500 decimal digits (resp. 2000 digits), we limited the largest prime factor of h to be ≤ 30 (resp. 100). For the extraction of small prime factors (used in the algorithm described in [20] and denoted EXTRACT in the sequel), we used $B = 8 \cdot 10^6$, 10^7, $3 \cdot 10^7$ for the three respective sizes.

SQRT refers to the computation of the $\sqrt{q_i^*}$, CORN to Cornacchia, PRP to probable primality tests; HD is the time for computing polynomials H_D using the techniques described in [16], jmod the time to solve it modulo p; then 1st refers to the building phase (step 1), 2nd to the other ones; total is the total time, check the time to verify the certificate. Follow some data concering D, h and the size of the certificates (in kbytes). All timings are cumulated CPU time on an AMD Athlon 64 3400+ running at 2.4GHz. In the tables, avg stands for average and std for standard deviation.

Looking at the average total time, we see that it follows very closely the $\tilde{O}((\log N)^4)$ prediction. According to our analysis, the main costs are the square roots, the pseudo primality test and finding a root of H_D, with the same $\tilde{O}(L^{4+\mu})$ heuristic cost. Square roots seem to become comparatively less important as L increases.

7. Conclusions

We have described in greater details the fast version of ECPP. We have demonstrated its efficiency. As for ECPP, it is obvious that the computations can be distributed over a network of computers. We refer the reader to [20] for more details. Note that the current record of 15041 decimal digits (with the number $4405^{2638} + 2638^{4405}$ see [21]), was settled using this approach. Many more numbers were proven prime using either the monoprocessor version or the distributed one, most of them from the tables of numbers of the form $x^y + y^x$ made by P. Leyland[†].

[†] `http://www.leyland.vispa.com/numth/primes/xyyx.htm`

	min	max	avg	std
SQRT	19	34	25	3
CORN	10	24	17	4
EXTRACT	60	84	74	5
PRP	74	124	102	14
HD	0	7	2	2
jmod	42	99	61	11
1st	178	276	234	27
2nd	79	136	99	12
total	260	387	334	34
check	18	22	20	0
nsteps	124	156	143	7
certif	396	456	435	13
D		8740947	120639	608050
h		1000	31	87

TABLE 1. 1000 decimal digits

	min	max	avg	std
SQRT	114	427	171	65
CORN	59	140	95	21
EXTRACT	195	282	230	20
PRP	472	903	664	99
HD	5	13	9	2
jmod	219	471	334	60
1st	868	1590	1192	185
2nd	368	649	508	70
total	1322	2240	1701	230
check	71	94	85	5
nsteps	183	209	198	7
certif	796	968	897	40
D		9644776	201015	848112
h		972	46	111

TABLE 2. 1500 decimal digits

Cheng [9] has suggested to use ECPP to help his improvement of the AKS algorithm, forcing $m = cN'$ to have $N' - 1$ divisible by a given prime large prime of size $O((\log N)^2)$. The same idea can be used to speed up the Jacobi sums algorithm, and this will be detailed elsewhere.

Acknowledgments. The author wants to thank N. Bourbaki for making him dive once again in the field of primality proving and D. Bernstein for stimulating emails on the existence and analysis of fastECPP. My co-authors of [20] were a source of stimulation through their records. Thanks also to P. Gaudry for never ending discussions on how close we are to infinity, as far as fast algorithms are concerned. D. Stehlé and P. Zimmermann for useful discussions around Cornacchia and fast sieving. Thanks to A. Enge for his help in improving the exposition, and to D. Bernardi for his remarks that helped clarify the exposition. Last but not least, thanks to the referee for his very detailed report that eliminated some mistakes and simplified the presentation.

	min	max	avg	std
SQRT	384	820	516	120
CORN	181	390	260	55
EXTRACT	600	853	713	67
PRP	1761	2879	2227	306
HD	6	27	16	5
jmod	969	1539	1255	188
1st	2974	4888	3778	528
2nd	1398	2120	1777	221
total	4494	6795	5557	711
check	213	261	238	13
nsteps	236	262	248	7
certif	1420	1644	1539	64
D		9760387	285217	1026529
h		1000	63	130

TABLE 3. 2000 decimal digits

REFERENCES

[1] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. of Math. (2)*, 117(1):173–206, 1983.

[2] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[3] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Math. Comp.*, 61(203):29–68, July 1993.

[4] M. Ayad. Points S-entiers des courbes elliptiques. *Manuscripta Math.*, 76(3-4):305–324, 1992.

[5] D. Bernstein. Proving primality after Agrawal-Kayal-Saxena. `http://cr.yp.to/papers/aks.ps`, January 2003.

[6] D. Bernstein. Proving primality in essentially quartic expected time. `http://cr.yp.to/papers/quartic.ps`, January 2003.

[7] P. Berrizbeitia. Sharpening "Primes is in P" for a large family of numbers. *Math. Comp.*, 74(252):2043–2059, 2005.

[8] R. Bröker and P. Stevenhagen. Elliptic curves with a given number of points. In D. Buell, editor, *Algorithmic Number Theory*, volume 3076 of *Lecture Notes in Comput. Sci.*, pages 117–131. Springer-Verlag, 2004. 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 2004, Proceedings.

[9] Q. Cheng. Primality proving via one round in ECPP and one iteration in AKS. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Comput. Sci.*, pages 338–348. Springer Verlag, 2003.

[10] H. Cohen and H. W. Lenstra, Jr. Heuristics on class groups of number fields. In H. Jager, editor, *Number Theory, Noordwijkerhout 1983*, volume 1068 of *Lecture Notes in Math.*, pages 33–62. Springer-Verlag, 1984. Proc. of the Journées Arithmétiques 1983, July 11–15.

[11] H. Cohen and H. W. Lenstra, Jr. Primality testing and Jacobi sums. *Math. Comp.*, 42(165):297–330, 1984.

[12] J.-M. Couveignes and T. Henocq. Action of modular correspondences around CM points. In C. Fieker and D. R. Kohel, editors, *Algorihmic Number Theory*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 234–243. Springer-Verlag, 2002. 5th International Symposium, ANTS-V, Sydney, Australia, July 2002, Proceedings.

[13] D. A. Cox. *Primes of the form* $x^2 + ny^2$. John Wiley & Sons, 1989.

[14] R. Crandall and C. Pomerance. *Prime numbers – A Computational Perspective.* Springer Verlag, 2000.

[15] A. Enge. The complexity of class polynomial computations via floating point approximations. Preprint, February 2004.

[16] A. Enge and F. Morain. Comparing invariants for class fields of imaginary quadratic fields. In C. Fieker and D. R. Kohel, editors, *Algorithmic Number Theory*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 252–266. Springer-Verlag, 2002. 5th International Symposium, ANTS-V, Sydney, Australia, July 2002, Proceedings.

[17] A. Enge and F. Morain. Fast decomposition of polynomials with known Galois group. In M. Fossorier, T. Høholdt, and A. Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 2643 of *Lecture Notes in Comput. Sci.*, pages 254–264. Springer-Verlag, 2003. 15th International Symposium, AAECC-15, Toulouse, France, May 2003, Proceedings.

[18] A. Enge and R. Schertz. Modular curves of composite level. *Acta Arith.*, 181(2):129–141, 2005.

[19] A. Enge and P. Zimmermann. mpc — a library for multiprecision complex arithmetic with exact rounding, 2002. Version 0.4.1, available from http://www.lix.polytechnique.fr/Labo/Andreas.Enge.

[20] J. Franke, T. Kleinjung, F. Morain, and T. Wirth. Proving the primality of very large numbers with fastecpp. In D. Buell, editor, *Algorithmic Number Theory*, volume 3076 of *Lecture Notes in Comput. Sci.*, pages 194–207. Springer-Verlag, 2004. 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 2004, Proceedings.

[21] J. Franke, T. Kleinjung, F. Morain, and T. Wirth. A new large primality proof using fastecpp. http://listserv.nodak.edu/archives/nmbrthry.html, July 2004.

[22] W. F. Galway. *Analytic computation of the prime-counting function*. PhD thesis, University of Urbana-Champaign, 2004. http://www.math.uiuc.edu/~galway/PhD_Thesis/.

[23] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[24] A. Gee. Class invariants by Shimura's reciprocity law. *J. Théor. Nombres Bordeaux*, 11:45–72, 1999.

[25] A. Gee and P. Stevenhagen. Generating class fields using Shimura reciprocity. In J. P. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 441–453. Springer-Verlag, 1998. Third International Symposium, ANTS-III, Portland, Oregon, june 1998, Proceedings.

[26] GNU. *The GNU Multiple Precision arithmetic library*. Available from http://www.swox.com/gmp/.

[27] S. Goldwasser and J. Kilian. Primality testing using elliptic curves. *Journal of the ACM*, 46(4):450–472, July 1999.

[28] G. Hanrot, V. Lefèvre, and P. Zimmermann et. al. mpfr — a library for multiple-precision floating-point computations with exact rounding, 2002. Version contained in GMP. Available from http://www.mpfr.org.

[29] G. Hanrot and F. Morain. Solvability by radicals from an algorithmic point of view. In B. Mourrain, editor, *Symbolic and algebraic computation*, pages 175–182. ACM, 2001. Proceedings ISSAC'2001, London, Ontario.

[30] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 12, pages 674–715. North Holland, 1990.

[31] H. W. Lenstra, Jr. and C. Pomerance. Primality testing with Gaussian periods. Preprint. Available as http://www.math.dartmouth.edu/~carlp/PDF/complexity072805.pdf, July 2005.

[32] Stéphane Louboutin. Computation of class numbers of quadratic number fields. *Math. Comp.*, 71(240):1735–1743 (electronic), 2002.

[33] P. Mihăilescu. Fast convolutions meet Montgomery. Preprint, March 2004.

[34] P. Mihăilescu and R. Avanzi. Efficient quasi-deterministic primality test improving AKS. Available from http://www-math.uni-paderborn.de/~preda/papers/myaks1.ps, April 2003.

[35] F. Morain. Primality proving using elliptic curves: an update. In J. P. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 111–127. Springer-Verlag, 1998. Third International Symposium, ANTS-III, Portland, Oregon, june 1998, Proceedings.

[36] F. Morain. La primalité en temps polynomial [d'après Adleman, Huang ; Agrawal, Kayal, Saxena]. *Astérisque*, pages Exp. No. 917, ix, 205–230, 2004. Séminaire Bourbaki. Vol. 2002/2003.

[37] W. Narkiewicz. *Elementary and analytic theory of algebraic numbers*. PWN-Polish Scientific Publishers, 1974.

[38] A. Nitaj. L'algorithme de Cornacchia. *Exposition. Math.*, 13:358–365, 1995.

[39] R. Schertz. Weber's class invariants revisited. *J. Théor. Nombres Bordeaux*, 14:325–343, 2002.

(F. Morain) LIX ÉCOLE POLYTECHNIQUE, CNRS/UMR 7161, INRIA/FUTURS, F-91128 PALAISEAU CEDEX, FRANCE

E-mail address, F. Morain: morain@lix.polytechnique.fr