IMPLEMENTING THE ASYMPTOTICALLY FAST VERSION OF THE ELLIPTIC CURVE PRIMALITY PROVING ALGORITHM LESS PRELIMIMARY VERSION 040825

F. MORAIN

Abstract. The elliptic curve primality proving algorithm is one of the fastest practical algorithms for proving the primality of large numbers. Its running time 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, runs in time $\tilde{O}((\log N)^4)$. The aim of this article is to 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], we know that 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 and C. Pomerance have announced a version in $\tilde{O}((\log N)^6)$. These algorithms are for the time being too slow to prove the primality of large numbers. Building on the work of P. Berrizbeitia [6], D. Bernstein [5] and P. Mihăilescu & R. Mocenigo [27], independently, have given improved probabilistic versions with a claim of proven complexity of $\tilde{O}((\log N)^4)$, reusing classical cyclotomic ideas that originated in the Jacobi sums test [1, 10]. For more on primality before AKS, we refer the reader to [12]. For the recent developments, see [4].

On the other hand, the elliptic curve primality proving algorithm [3] has been used for years to prove the primality of always larger numbers*. The algorithm has a heuristic running time of $\tilde{O}((\log N)^5)$. In the course of writing [30], the author rediscovered the article [24], in which an asymptotically fast version of ECPP is described. This version, attributed to J. O. Shallit, has a heuristic running time of $\tilde{O}((\log N)^4)$.

To the best of the author's knowledge, this fast version has never been implemented. Due to the large numbers now being dealt with and the stimulating effect of both the AKS thread and the fair-play concurrence of PRIMO, the author realized that implementing this fast version was now at hand, helped by recent work on the computation of defining polynomials of Hilbert class fields. The aim of this 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. 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. Section 7 describes a compact fingerprint that can be used to rebuild a certificate from it.

Date:

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

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 when $4 \mid D$, $(D/4) \bmod 4 \in \{1, 2\}$. The quantity

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

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

A fundamental discriminant may be written as:

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

where all q_i^*'s are distinct and q_i^* is either -4 or ± 8, or $q_i^* = (-1/q_i)q_i$ for any prime q_i. 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, Siegel proved that $h = O(D^{1/2+\varepsilon})$ asymtotically.

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 $\mathbf{K}$ 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

3.1. **Presentation of the 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 acan be found in [3].

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

$$4N = U^2 + DV^2 \tag{1}$$

in rational integers U and V. For all solutions U of (1), 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.] Reduce $\overline{E}$ modulo N to get a curve E.

[Step 4.] Find $P \neq O_E$ on E such that $[N']P \neq O_E$. If this cannot be done, then N is composite, otherwise, it is prime.

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

Clearly, we need $\log N$ steps for proving the primality of N.

3.2. **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 L, namely $O(L^{1+\mu})$, where $0 \leq \mu \leq 1$ ($\mu = 1$ for ordinary multiplication, or $\epsilon > 0$ for any fast multiplication method). We consider all steps, one at a time, easier steps first.

3.3. **Analysis of Step 4.** This one is easy to analyze, since an exponentiation on a curve costs $O((\log N)^{2+\mu})$, and we need $O(1)$ points on average. Note that this step is a randomized one.

3.4. **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 primitive reduced quadratic forms of discriminant $-D$. If (A, B, C) is such a form, with $B^2 - 4AC = -D$, then

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

In [14], it is argued that the height of this polynomial is well approximated by the quantity:

$$\pi\sqrt{D} \sum_{[A,B,C]\in Cl(-D)} \frac{1}{A},$$

which is $\tilde{O}(D^{1/2}) = \tilde{O}(h)$ by Siegel's theorem.

Evaluating the roots of $H_D(X)$ and building this polynomial can be done in $\tilde{O}(h^2)$ operations (see [13]). This could be seen as a precomputation, since it does not depend on N.

Alternatively, we could use the method of [11, 7] for computing the class polynomial and get a proven running time of $\tilde{O}(h^2)$, but assuming GRH.

3.5. **Analyzing Step 3.** Reducing E modulo N is done by finding a root of $H_D(X)$ modulo N. This can be done with the Cantor-Zassenhaus algorithm (see [19] for instance). Briefly, we split recursively $H_D(X)$ by computing $\gcd((X + a)^{(N-1)/2} - 1, H_D(X)) \bmod N$.

Computing $(X + a)^{(N-1)/2} \bmod (N, H_D(X))$ costs $O((\log N)\mathsf{M}(N, h)) = O(L\mathsf{M}(N, h))$ 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 [19, Ch. 11]). The total splitting requires $\log h$ steps, but the overall cost is dominated by the first one, hence yields a time:

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

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

3.6. **Analysis of Step 1.** Given D, testing whether (1) is done using Cornacchia's algorithm (see [31]). This requires the computation of $\sqrt{-D} \bmod N$, using for instance the Tonelli-Shanks algorithm, for the cost of one modular exponentiation, i.e., a $O(L^{2+\mu})$ time. Then it proceeds with a half gcd like computation, for a cost of $O(L^{1+\mu})$ (see also section 5.3 below).

In the event that equation (1) is solvable, then we need check that $m = 2N'$ and test N' for primality, which costs again some $O(L^{2+\mu})$.

The heuristic probability of m being of the given form is $O(1/\log N)$. Though quite realistic, it is impossible to prove, given the current state of the art in analytical number theory. Using this heuristic, we expect to need $O(L)$ splitting D's. Using all D of size $O(L^a)$, we can dispose of $\mathcal{D}(L^a) = O(L^a)$ discriminants. The probability that D is such that N splits in $\mathbf{K}$ is $O(1/h) = \tilde{O}(D^{-1/2})$ by Siegel's theorem. Therefore, we need $O(L^{a/2+1})$ discriminants to meet our number of m. Thie readily implies that $a \ge a/2 + 1$ or $a \ge 2$.

Turning to complexity, the cost of Step 1 is then:

$$O(L^{a/2+1}(\underbrace{L^{2+\mu}}_{\text{squareroot}} + \underbrace{L^{1+\mu}}_{\text{half gcd}})) + O(L \cdot \underbrace{L^{2+\mu}}_{\text{probable primality}}).$$

Setting the minimal possible value of a to the value 2, we get that this phase takes:

$$O(L^2 \underbrace{(L^{2+\mu} + L^{1+\mu})}_{\text{Cornacchia}}) + O(L \cdot \underbrace{L^{2+\mu}}_{\text{probable primality}})$$

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

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

4. The fast version of ECPP

4.1. **Presentation.** When dealing with large numbers, all the time is spent in the finding of 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. Looking at the analysis carried out above, we see that we need $O(L^2)$ discriminants to find a good one. The basic version finds them by using all discriminants that are of size $O(L^2)$. As opposed to this, one can build those discriminants as $-D = (-p)(q)$, where p and q are taken from a pool of size $O(L)$ primes.

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

[Step 1'.]

1.1. Find the $r = O(L)$ 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 $q_{i_1}^* q_{i_2}^* = -D < 0$, try to solve equation (1).

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

$$O(L \cdot \underbrace{L^{2+\mu}}_{\text{squareroots}}) + O(L^2 \cdot \underbrace{L^{1+\mu}}_{\text{half gcd}}) + O(L \cdot \underbrace{L^{2+\mu}}_{\text{probable primality}})$$

which yields namely $O(L^{3+\mu})$. Note here how the complexity decomposes as $3 = 1 + 2$ or $2 + 1$ depending on the sub-algorithms.

Putting everything together, we end up with a total cost of $\tilde{O}(L^4)$ for this variant of ECPP.

4.2. **Remarks.**

4.2.1. *Complexity issues.* We can slightly optimize the preceding argument, by using all subsets of $\mathcal{Q}$ and not only pairs of elements. This would call for $r = O(\log \log N)$, 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.

5. Implementation

5.1. **Some general comments on the implementation of ECPP.** 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.

5.2. **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 [25] that yields a practical method in $O(h)$ with a very small constant.

5.3. **Improving Cornacchia's algorithm.** Step 1 needs squareroots to be computed, and some half gcd to be performed. Briefly, the algorithm runs as follows (see [31]):

procedure Cornacchia(d, p, t)
{t is such that $t^2 \equiv -d \bmod p$, $p/2 < t < p$ }
2. [Partial Euclid:]
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**
$r_{i-2} = a_i r_{i-1} + r_i$, $0 \leq r_i < r_{i-1}$;
$w_i = w_{i-2} + a_i w_{i-1}$ $\quad (*)$;
c) **if** $r_{i-1}^2 + d w_{i-1}^2 = p$ **then** return (r_{i-1}, w_{i-1}) **else** return $\emptyset$.

We end the **for** loop once we get $r_{i-1} \leq \sqrt{p} < r_{i-2}$. As is well known, the numbers 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, from the theory, we know that this algorithms almost always returns that 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:

$$w_i = w_{i-2} + a_i w_{i-1} \bmod 2^{32}$$

and at the end, we test whether $r_{i-1}^2 + d w_{i-1}^2 = p \bmod 2^{32}$. If this is the case, then we redo the computation of the w_i's and check again.

5.4. **Factoring m.** Critical parameters are that related to the factorization of m, since in practice we try to factor m to get it of the form cN' for some B-smooth number c.

As shown in [18], the number of probable prime tests we will have to perform is $t = O((\log N)/(\log B))$ and we will end up with N' such that $N/N' \approx \log B$.

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

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

for all $p_i \leq B$'s, which costs $\pi(B)(\log N)^{1+\varepsilon}$, 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, 29] for more details and tricks.

The cost of this algorithm for t values of m is

$$O(\pi(B)(\log N)^{1+\varepsilon}) + O(t \cdot \pi(B)(\log N)^{1+\varepsilon})$$

where the second term is that for computing $U \bmod p_i$, which is slightly half 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 $B = O(L)$.

For larger numbers, it is better to use the stripping factor algorithm in [18], for a cost of $O(B(\log B)^2)$, the optimal value of B being $B = O((\log N)^3)$.

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

$$r'_i = (N' + 1) \bmod p_i$$

which may be computed as:

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

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

5.4.2. *Using an early abort strategy.* This idea is presented in [18]. 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 $N/N' \approx \log B$.

It should be remarked that throwing away m's and thus on D's tends to make D and h larger. This can have an impact on the time for computing H_D, and also on the proving part. EAS indeed decreases the number of steps, which tends to decrease the total time for the 1st phase, the 2nd being constant or increasing a little.

5.4.3. *Using new invariants.* Proving larger and larger numbers 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, so as to minimize the size of the minimal polynomials. This task was done using Schertz's formulation of Shimura's reciprocity law [32], with the invariants of [16] as demonstrated in [14] (alternatively see [21, 20]). Note that replacing j by other functions does not the change the complexity of the algorithm, though it is crucial in practice.

5.4.4. *Step 3 in practice.* We already noted that this step is the theoretically dominating one in fastECPP, with a cost of $O((\log L)L^{3+\mu+\nu})$. In practice, for even for small values of h, we can assume $\nu \approx 0$ (using for instance the algorithm of [26] for polynomial multiplication).

Galois theory comes in handy for reducing the $\log L$ term to a $\log \log L$ one, if we insist on h being smooth. Then, we replace the time needed to factor 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 [23, 15]. 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 [9].

5.4.5. *Improving the program.* The new implementation uses GMP[†] for the basic arithmetic, which enables one to use `mpfr` [22] and `mpc` [17], 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.

Here are some timings on a 629 word number on a Xeon:
$\sqrt{-D}$: 34 sec;
Cornacchia: 0.04 sec;
SieveDWithTabCompactMax: 18 sec;
EcppProbablePrimeTest: 34 sec.

5.5. 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 $\{q_1^*, \dots, q_r^*\}$ 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.

5.6. **Distributing the computations.** Many tasks in (fast)ECPP can be distributed (see [28] for the first description): Step 1, 3 and 5 are best choices, and the idea is to use a master/slave approach. The present version uses MPICH for this task. Distribution gains a factor proportional to the number of processors, in good cases.

The process used is of type master/slave again. The first idea is to let all slaves compute the $\sqrt{q_i^*}$, and then take a batch of D's.

To balance load on the slaves, we used the following trick. Each D has splitting probability g/h. We send to each slave discriminants $D_{i_1}, D_{i_2}, \dots, D_{i_k}$ in such a way that

$$\sum_j g(-D_{i_j})/h(-D_{i_j}) \geq 5$$

(the value of 5 is somewhat arbitrary) which corresponds to the fact that on average, 5 values of D will be splitting values.

See [18] for more strategies[‡].

6. Benchmarks

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.

Not really surprisingly, strategies that one could dream of are not necessarily worth the trouble for small numbers (say 1000dd or 2000dd) and get interesting for larger ones.

7. Some numbers proven

All computations were done on a cluster of 6 biprocessors Xeon at 2.66 MHz. We took the following numbers from the tables of P. Leyland[§]; WCT stands for *wall clock time*.

[†]http://www.swox.com/gmp/
[‡]More to be written
[§]`http://www.leyland.vispa.com/numth/primes/xyyx.htm`

x	2177	2589	2551	2438
y	580	218	622	1995
#dd	6016	6055	7127	8046
when	030513-030604	030606-030617	030618-030714	
# steps	801	736	965	1128
1st phase (CPU)	164 days	103 days	235 days	355 days
1st phase (WCT)	164×1.2	103×1.1	235×1.3	355×1.13
$\sqrt{-D}$	81 days	30 days	74 days	138 days
2nd phase	28 days	21 days	55 days	77 days
H_D	2951 sec	1686 sec	18451 sec	22552 sec
CZ	26 days	20 days	50 days	69 days
GHz days			625	1000
Checking	25 hours	22 hours	45 hours	70 hours
$\max h$	1980	2080	3312	3640
$\max D$	7,749,263	19,076,479	52,396,648	87,949,348

TABLE 1. Some large numbers proven with FASTECPP.

Putting $\theta = 7127/6055 \approx 1.177$, we find that $\theta^4 \approx 1.91$ is close to 2, which is coherent with the claimed complexity.

The 7127dd number required 625 GHz-days, and should be compared to the 738 GHz-days for the 6959 decimal digit number of Hans Rosenthal.

8. A compact fingerprint

One advantage of ECPP is the certificate, which consists in $\log N$ curves and points, for a total size of $O((\log N)^2)$, and that can be checked in time $\tilde{O}((\log N)^3)$.

We describe here a short compact version of it that could be used to reproduce the sequence of intermediate primes. The idea is just to give the list of D's used, together with the sign of U. Knowing this and the fact that we sieve up to certain bound, then we can imagine reconstructing the whole proof, though with some effort. It can serve as a convincing element that the author of the proof really went through.

Let $D > 4$. Then $D-$ means that $m = N+1+U$ was used, as $D+$ means $m = N+1-U$. There are three special cases: When the $N-1$ test (resp. $N+1$ test) was used, we write $1-$ (resp. $1+$).

When $D = 3$, there are 6 possible cardinalities. In that case, we can write $4N = L^2 + 27M^2$, $L \not\equiv 0 \bmod 3$, $L > 0$, $M > 0$. We decide to code the cardinalities as in:

cardinality	code
$N+1-L$	a
$N+1+L$	b
$N+1-(L+9M)/2$	c
$N+1+(L+9M)/2$	d
$N+1-(L-9M)/2$	e
$N+1+(L-9M)/2$	f

When $D = 4$, we know that there are 4 possible cardinalities. Write $N = x^2 + 4y^2$, with x odd, $x > 0$, $y > 0$. We decide to code the cardinalities as in:

cardinality	code
$N + 1 - 2x$	a
$N + 1 + 2x$	b
$N + 1 - 4y$	c
$N + 1 + 4y$	d

9. Conclusions

We have described in greater details the fast version of ECPP. We have demonstrated its efficiency. We have thus settled some benchmarks for the fast AKS versions to compete and stimulate further work on the topic.

On 030819 (private communication of J. Franke), the mythical 10000 decimal digits frontier was reached with another implementation of fastECPP (I presume) by J. Franke, T. Kleinjung and T. Wirth[¶]. It is now set to 15041 decimal digits (see transaction in NMBRTHRY). See the article [18] for more details.

Cheng [8] 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 [18] 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.

References

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

[2] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Preprint; available at `http://www.cse.iitk.ac.in/primality.pdf`, August 2002.

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

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

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

[6] P. Berrizbeitia. Sharpening "Primes is in P" for a large family of numbers. `http://arxiv.org/abs/math.NT/0211334`, November 2002.

[7] 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.

[8] 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.

[9] 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.

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

[¶]The certificate for the smallest prime of this size, namely $10^{9999} + 33603$, is available as `ftp://ftp.math.uni-bonn.de:pub/people/franke/p10000.cert`.

[11] 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.
[12] R. Crandall and C. Pomerance. *Prime numbers – A Computational Perspective*. Springer Verlag, 2000.
[13] A. Enge. The complexity of class polynomial computations via floating point approximations. Preprint, February 2004.
[14] 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.
[15] 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.
[16] A. Enge and R. Schertz. Modular curves of composite level. Soumis, 2003.
[17] 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.
[18] 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.
[19] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
[20] A. Gee. Class invariants by Shimura's reciprocity law. *J. Théor. Nombres Bordeaux*, 11:45–72, 1999.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] Stéphane Louboutin. Computation of class numbers of quadratic number fields. *Math. Comp.*, 71(240):1735–1743 (electronic), 2002.
[26] P. Mihăilescu. Fast convolutions meet Montgomery. Preprint, March 2004.
[27] 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.
[28] F. Morain. Distributed primality proving and the primality of $(2^{3539}+1)/3$. In I. B. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Comput. Sci.*, pages 110–123. Springer–Verlag, 1991. Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, Aarhus, Denmark, May 21–24, 1990.
[29] 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.
[30] F. Morain. La primalité en temps polynomial [d'après Adleman, Huang ; Agrawal, Kayal, Saxena]. Séminaire Bourbaki, exposé 917, March 2003.
[31] A. Nitaj. L'algorithme de Cornacchia. *Exposition. Math.*, 13:358–365, 1995.
[32] R. Schertz. Weber's class invariants revisited. *J. Théor. Nombres Bordeaux*, 14:325–343, 2002.

Appendix A. Pseudocertificates of some numbers

Here we give the pseudocertificates of the numbers listed in Table 1.

$x = 2177, y = 580$:

```
5254099+1514287-395179+37016-898747+6131+1460299-14251+4295+9892+2209348+
2819483-147367+54759-15443+248+874067+17491+5695567+700351-78856-51479+
145747-939-691963-103947+21847-437963-378647-381971+643967+130888-7048187-
22867-31715+6491-402019+8971+14179-6702491+841784-6227251-6920932+276631-
3408995+5958712-902203-96856+8839+35572+3128827-19115+424+4121368+34184+
```

```
10943+14251+7963-2346499-28459+4339+4864312+116251+353791-577891+22927-
174179-1898851+84763-25768+7327807+1365263-15907+981347-45316-772-182419+
529339-17124-76408+395187+6668491-9467+469027+1831+15307+1227+2008-64324+
554408-294607-4045519+11-91+10515+40307+4214467-185451+9208-15432-7239811-
3647311-3368-22867-364807+70931-2733139+34663+777067-500419+154776+3079019-
4488223-78103-6603643-910999+273491+49483-37495+7308107+518923+30959-3558616-
355+84031-19156-13432+3917443-410107+11819-292312-3093067+24643+2811243-
6387-107731+11812-472267+6827-1147+3647443-2984264-239647+9739-173951-
6188543+3163-3084824+46303+2979112+115+192868+3a11971+2198611-1917067-
5627-491467-3898399-728227-378031-183892-180643-1576-166699+529351+5996947-
2696-233951+44628-77251+259139+1816+3755+4174447+4b125359+244-17131+69339+
297668+1003+4447-3028+2415316-4954856+233731-912776-2207+202651+25699-
114488-7907+7403-11383+1103-1636-563-2627+3226667+3886703+50212-404+65411-
3067-1912-1859839+14267-426776-107188+4483-2152-46039-273839-109003-1076+
1336+10763+48676-1432+371-307156+7+25663+505415-92819-6067+43+458923-690499-
3243571-163451+842459-58648+523067-12751+5328299+4099-62831+493019+19336-
2328-646068-731-40+77827+1630099+11+32971+46259+10307-448036-3635-14567-
7+127891+318459+104+11-79787-5135656+452+3652328-3508+642883+926291-200747+
11091+3236-3448-2093723+13971-719+59323-1907+5107-3404248+691-339-3078811-
959107+291-1592+1588+1519043+23699-68323-187-1422451+56104+25199+356+85195+
8552+413059+9556+38131-5683-5647+1003+16251+451+35704+4747+164167-515-
1485863-1421192+4324763+4555-388-151819+4b983207+43-742523+32632-407+1166824-
3656963+53027-837547+187723+447091-44743-42488-7353847-427171+1636+703723-
40571-2815907+1255+84379+4852+6499+19+52243-355-499-709867-184-1+28207+
521099+2879-318947+26359-306343+291-66719+432739-3c284043+16552+15415+
2707-289128+392603+211+2047-300907-3028-9771+1600523+20371-1274467-283-
212116+2267-355+52-53571+2667-295739+91012-1068359-13268+111379-4279+103+
999847-388+498455-127427+424-7-667+1208-7483+427003-7003-9304+5924+46324-
608183-55-700852+35467+331-107512+233368+31483+45172-46087+19-1-46363-
1367+19171+1+97483-776-120-2335+691-14068-25624-99899-463-20783-561779+
87371-36227+2318227-19147+4843-2395-632+632+2923+835+7409683+647+1763+
5907-68-3547-19-13603+7528-631-1513063+31043+491287+3079-1050527+2271+
78943-8159-84839-8891-8-6187+25828+3f5331+328+668152-6847-36292-1651+268276-
139-1411-93659+103+8507-24331+1010455+107+413972+4712-113599-3b2779+12439+
9496+13956-19+2215-91+307387-163+33239+429272+2792983-4a65956-88051-482932-
131107+643+366987+156931-57403-3563048+347+248107-220331+7749263-9304+
167+20819+1192-34063+4072-445127-13435+3867-7608-6359272+24+69227+30792+
797959+5163-25507-643-117823-764851+238219+13684-4843-1843-4659-2872+1027+
2324323+544283+15256-1011+88267-21967+153415+1864+427+2787-5332+235-6103+
607-2404-1797923+40760+17587+707347-322467+1+4372+19524-1348+1333492+2683+
116887+12632+96632+5188-1409179+4195-2916923-1526331+144251+88-67-635704-
262099-8-811+11-356-1-907-12388+10603+899+87259-53755+408904+18232+155803-
6340+7-3147-14383+485908+66331+1283-30760-1959+13987+7783-8584-143-5315-
135992+8-307-11-20+244-16808-18868+4072+20584-4c1038403-452+21995-2419+
1592728+4d3811+2373155+3535+51643-75587-24559+2827+295+3412+223-3d188059+
61924+4d427-47683+443+52+1255+8952+7588-1+4744-328-14491-70283+344-7051+
52-62452-855827+9563+115+1207768-9299+787-43-3f1019+643-388-1051-25027+
1115-110783+50072+319304+2152-69583+116+167+51+12179+1051+3d163-6712+1355-
17368+211+24523+291931-183+904+463-1-788-232-190499+163+787+5608+3336-
3992-1203-5707-8+40-19-779527-259+347+163-43+7231+4771-1232047-8072-1528+
504067-23-1123+219-422843-3c4c163-1099+93507+26179-8+3736+7363-8-1192-
7+11203+31783-126187+2047+7240+179+331+8+5371+10264+707+4b67+183-1087-
328583+67+451+163+571-261871-879787+155+31+439-19+1-323+19-19-3b1047-163+
67-4372+3b4b1+955-4b15780-4c372+20+1547+259-4888+4a532+235+3a3d52+523-
4b11+4a8+4b3b123+3a3d1+1+1-1-91+19+1+3b3a3d3c1-1-1-1-1-1-1-1-1-
```

$x = 2589, y = 218$:
$x = 2551, y = 622$:

(F. Morain) LIX École Polytechnique, CNRS/UMR 7650, INRIA/Futurs, F-91128 Palaiseau CEDEX, FRANCE

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