Pseudoprimes and Primality Proving with Elliptic Curves (ECPP)

Andreas Balser

February 20, 2001

This talk is about finding out whether a given number is prime or composite. First we will take a look at what is called *Pseudoprime Tests*, and later on we will get into elliptic curves to use them to give a proof for the primality of a number with *certificate*.

The topic is up to date, because in cryptography (i.e. in particular in all areas of e-commerce), all the algorithms depend on large primes; therefore, there is a need to find new big prime numbers all the time. Of course, the problem of factoring a given number into its prime factors is as important (in fact, probably even more important; for if someone finds a fast way of factoring a number e-commerce is not safe anymore). Along the way, we will also see why these two problems are not equivalent.

What does the word *certificate* used above mean in this context? A primality test is a procedure that tells us whether a given number n is prime or not. If the answer is "yes" or "no" only, then it is a test without certificate; if (in the "yes"-case) the test also returns data that enables us to verify the correctness of the output at any given time easily (that is with much less effort than re-running the test), it is a test "with certificate". We will see that ECPP is a primality test with certificate, one of its advantages over other comparably fast tests.

1 Pseudoprime Tests

A *pseudoprime test* is a procedure which, given a number n, tells you either "n might be a prime number" or "n is definitely not a prime number". In the latter case, most pseudoprime tests return a certificate against the primality of n. The point is that pseudoprime tests test a certain property of prime numbers (usually involving a random number $a \not\equiv 0 \mod n$). If n fails to satisfy the property with respect to a, we know that n is not prime, and say that a is a *witness* against the primality of n. Otherwise, we are still uncertain, and we call n a *pseudoprime for the base a* (w.r.t. the pseudoprime test used).

For obvious reasons, a pseudoprime test is also called a *non-primality test*.

Pseudoprime Test 1 (trial division): Given n, pick a random number $a \in \{2, 3, \ldots, [\sqrt{n}]\}$. If $a|n$, then n is not a prime number.

This procedure, called trial division, is a very inefficient pseudoprime test, because n only detects the compositeness of n by guessing a divisor.

We will take a look at two more elaborate pseudoprime tests; unfortunately, these tests do not always return a divisor even if they prove that n cannot be a prime number.

Pseudoprime Test 2 (Fermat's test): Given n, pick a random number $a \in \{2, 3, \ldots, n-2\}$. If $a^{n-1} \not\equiv 1 \mod n$, then n is not a prime number.

Proof. This is just rephrasing Fermat's little theorem, which says that for a prime number p, we have $a^{p-1} \equiv_p 1$ for all $a \not\equiv 0 \mod n$.

This fact is clear if you know that $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ is a field. Therefore, $\mathbb{Z}_p^* = \mathbb{Z}_p \backslash \{0\}$ is a group (the multiplicative group of $\mathbb{Z}_p$) of order $p-1$. In every group, the order of any element is a divisor of the group order; hence $a^{p-1} \equiv_p 1$ has to hold. □

This test clearly proves the compositeness of n in case it guesses a divisor a of n.

An example that it catches composite numbers more often than Fermat's test is
$2^8 = 256 = 9 \cdot 28 + 4 \equiv_9 4 \neq 1$.

Remarks:

- To compute the power, one uses the algorithm `fast-exponentiation` ([2], Alg 1.1)
- We need not try $a = n-1$, because $n-1 \equiv_n -1$, and so certainly $(n-1)^{n-1} \equiv_n (-1)^{n-1} = 1$, since we can assume that n is odd.

There is a natural question to be asked: How probable is it that this test detects the compositeness of n?

Unfortunately, there is a whole class of numbers for which Fermat's test only detects compositeness if it guesses a multiple of a divisor.

These numbers are called *Carmichael numbers.* The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$. Carmichael numbers can be classified: *An odd composite number n is a Carmichael number if and only if it is square free and for every prime divisor p of n, we have $p-1|n-1$* (Proof: [3], 6.3.1).

Miller-Rabin Test

The Miller-Rabin test is one of the most sophisticated pseudoprime tests known:

Proposition 3 (basis for Miller's test). *If n is of the form $n = 1 + 2^t n_0$ (for some t and odd n_0), then*

$$a^{n-1} - 1 = \left(a^{n_0} - 1\right)\left(a^{n_0} + 1\right)\left(a^{2n_0} + 1\right)\left(a^{4n_0} + 1\right)\cdots\left(a^{2^{t-1}n_0} + 1\right)$$

Proof. We will prove this by induction: For $t = 0, 1$, the statement is obviously true. For the inductive step, let $n = 1 + 2^{t+1}n_0$. Then we have

$$a^{n-1} - 1 = a^{2^{t+1}n_0} - 1 = \left(a^{2^t n_0} - 1\right)\left(a^{2^t n_0} + 1\right)$$

which together with the inductive hypothesis finishes the proof. □

Pseudoprime Test 4 (Miller-Rabin Test): Given any odd number n, find t, n_0 (with n_0 odd) such that $n = 1 + 2^t n_0$.
Pick $a \in \{2, 3, \ldots, n-1\}$. If n is prime, then either $a^{n_0} \equiv_n 1$ or $a^{2^j n_0} \equiv_n -1$ for some $j \in \{0, 1, \ldots, t-1\}$.

Proof. Assuming that n is prime, we know from Fermat's little theorem $n | a^{n-1} - 1$. With the factorization of the right-hand side given above, because n is prime, it has to divide one of the factors. □

This is a very efficient test: Writing $n = 1 + 2^t n_0$ is easy to achieve. Computing the powers we need is also straightforward, as we only have to compute a^{n_0}; after that, we need only square the previous number at each stage.

The reason why this test is used so often is the following:

Proposition 5 (Error-probability of the Miller-Rabin Test). *For an arbitrary odd composite number n there are at most $\frac{n}{4}$ numbers in $\{2, \ldots, n-1\}$, for which the Miller-Rabin Test fails to detect the compositeness.*

Proof. [6, Theorem 2.33] □

The important consequence of this proposition is: After applying the test k times without rejection, the probability that n is composite is at most $\left(\frac{1}{4}\right)^k$.

2 Pocklington's Theorem

Next, we have to consider Pocklington's theorem, because the algorithm using elliptic curves which we are heading towards is derived from the following theorem.

Proposition 6 (Pocklington's Theorem). *Let n be a natural number, and s a divisor of $n-1$ satisfying $s > \sqrt{n}$. If there is a number m such that for every prime divisor q of s, the following conditions are satisfied, then n is prime.*

$$m^{n-1} \equiv 1 \mod n \tag{1}$$
$$\gcd\left(m^{(n-1)/q} - 1, n\right) = 1 \tag{2}$$

Proof. Assume that n is not a prime. So there is a prime divisor $p \leq \sqrt{n}$ of n.
Let $b = m^{(n-1)/s} \mod n$. Then

$$b^s \equiv \left(m^{(n-1)/s}\right)^s \equiv m^{n-1} \equiv 1 \mod n$$

(by assumption (1)). Therefore, we also have $b^s \equiv 1 \mod p$.
For every prime divisor q of s, we get

$$b^{s/q} \not\equiv 1 \mod p$$

because: If q' were a prime divisor of s with $b^{s/q'} \equiv 1 \mod p$, then p would be a divisor of $b^{s/q'} - 1$, contradicting (2).

This means: s is the order of b in $\mathbb{Z}_p^*$ (proof e.g. in [3], 2.14.1).

The order of this group is $p-1$, implying that s is a divisor of $p-1$. The contradiction $s > \sqrt{n} \geq p$ finishes the proof. □

This proposition gives rise to a primality test. Note that we have to find either a sufficiently large prime divisor of $n-1$ or enough divisors that yield a sufficiently large s by multiplication. Then we keep picking numbers m and test the conditions. If some m satisfies the conditions, n is prime. If some m fails 1 or fails 2 "badly" (i.e. the gcd is not a multiple of n), it is not.

This procedure will eventually terminate, because in case n is a prime, the existence of a generator of $\mathbb{Z}_n^*$ clearly shows the existence of a suitable number m.

In case n is not prime, a divisor of n will eventually be found. This might take very long; however, this does not matter, because primality tests are only applied to pseudoprime numbers (i.e. we are almost certain that n is a prime anyway).

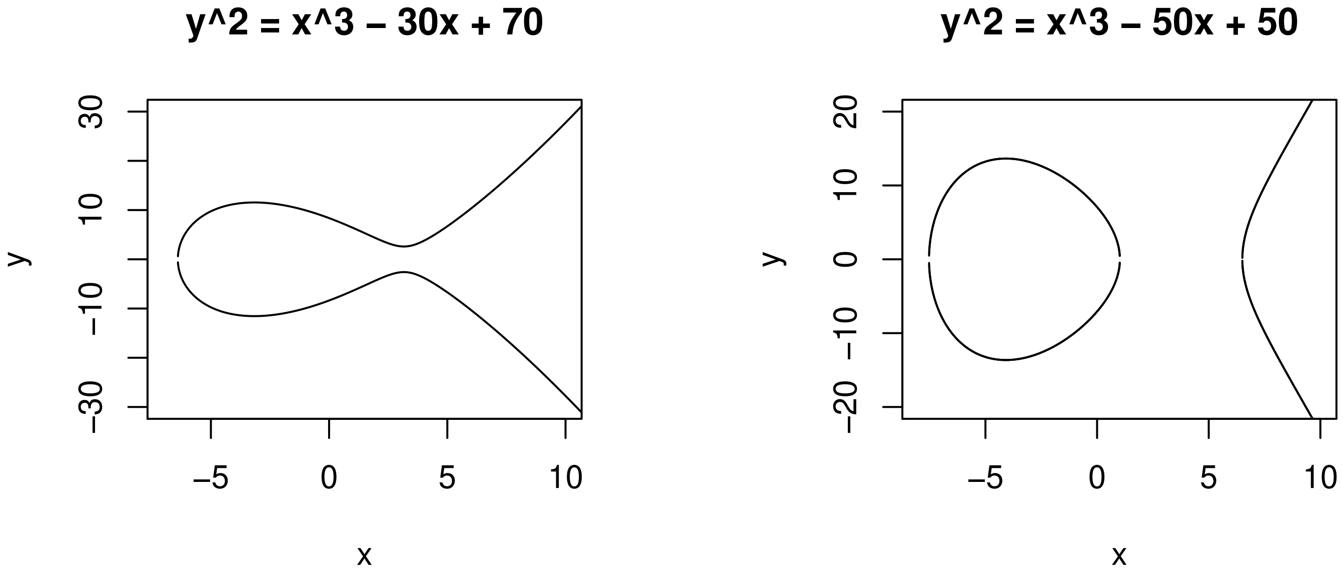

Figure 1: Some regular elliptic curves

3 Short introduction to Elliptic Curves

As stated above, we want to generalize Pocklington's Theorem to be suitable for elliptic curves.

For a fixed prime number p, elliptic curves provide a wide variety of finite abelian groups related to p; in classical number theory, the finite field $\mathbb{Z}_p$ is fixed.

An elliptic curve[1] is the set of points in the plane that satisfy an equation like

$$y^2 = x^3 + ax + b$$

where $4b^3 + 27b^2 \neq 0$; otherwise, the elliptic curve is called singular, and is not suitable for our purposes.

In fact, to define the group operation on these elliptic curves, we need to add a point called $\mathcal{O}$. This point is called "point at infinity", and is imagined to be lying infinitely far up along the y-axis; its existence can be understood regarding the elliptic curve to be lying in the projective plane.

The group operation works as follows: You are given two points P, Q on the elliptic curve. The line joining these two points will always intersect the elliptic curve in a third point: If $P = Q$, we take the tangent line at P. If the line that joins P and Q is vertical, then the third point of intersection is $\mathcal{O}$. If the line intersects E twice only, then it is tangential to one of the points; this point is used as if it were a double intersection point.

[1] Actually, ellptic curves can be described a little more generally; but the more general forms can be brought into the described form by a change of variables. In order to perform this change of variables considering elliptic curves over $\mathbb{Z}_n$, we will have to assume $n \neq 2, 3$

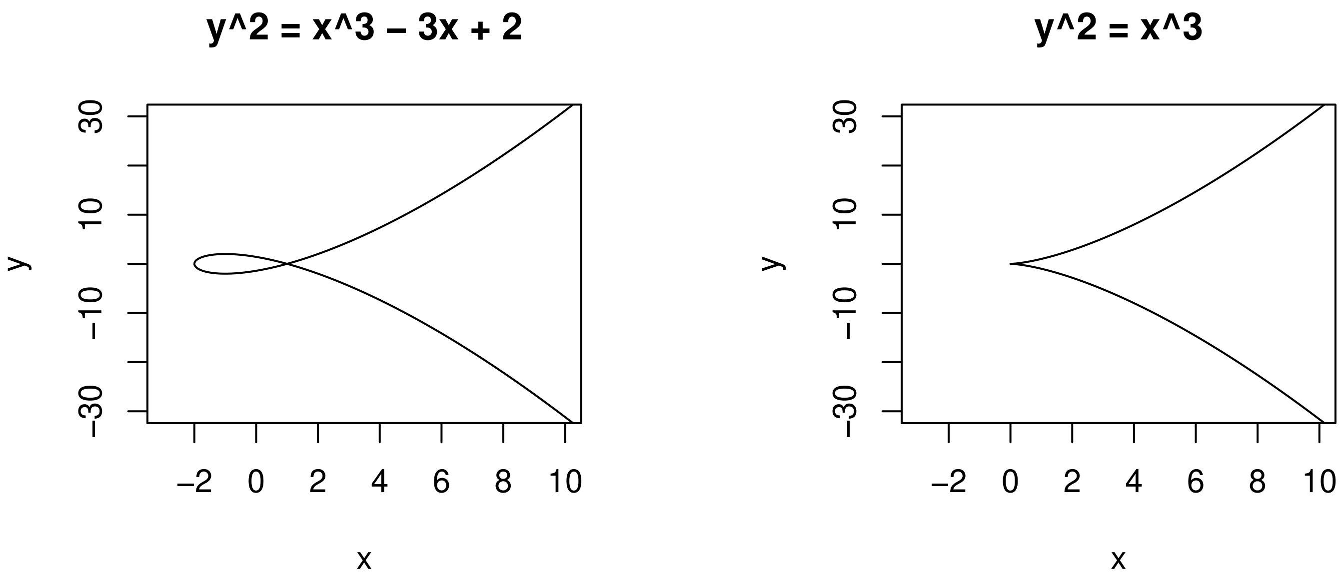

Figure 2: Some singular elliptic curves

Let R be the third point of intersection. If $R = \mathcal{O}$, then let $P + Q = \mathcal{O}$; otherwise, $R = (r_1, r_2)$, and we define $P + Q = (r_1, -r_2)$.

This turns out to be a group: $\mathcal{O}$ is the neutral element[2], and the inverse of any other element $P = (p_1, p_2)$ is $(p_1, -p_2)$.

The group axioms, including commutativity, are not hard to verify, except for associativity. A proof of associativity can be found in [5].

The operation on an elliptic curve is usually denoted additively; hence, we will refer to $\mathcal{O}$ as zero.

It is possible to work out the formulas for addition on an elliptic curve: Let $P_i = (x_i, y_i)$ be two non-zero points on the elliptic curve. The parameter that we need to compute is λ, the slope of the tangent/secant line.

If $x_1 \neq x_2$, then

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = (y_2 - y_1)(x_2 - x_1)^{-1}$$

If $P_1 = P_2, y_1 \neq 0$, then

$$\lambda = \frac{3x_1^2 + a}{2y_1} = (3x_1^2 + a)(2y_1)^{-1}$$

Then $P_1 + P_2 = (x_3, y_3)$ with

[2] Actually, it is possible to make an arbitrary point the neutral element (using a similar geometric way of defining the group operation), but $\mathcal{O}$ is the usual choice.

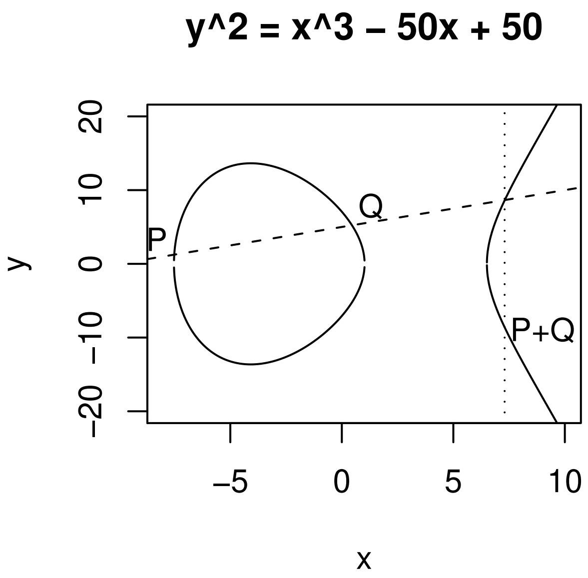

Figure 3: Addition of points on an elliptic curve

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_3 - x_1) - y_1$$

In the other cases, the computation of $P_1 + P_2$ is trivial (if the x-coordinates agree, but not the y-coordinates, $P_1 + P_2 = \mathcal{O}$; in the second case, if the y-coordinate is 0, also $P_1 + P_2 = \mathcal{O}$.

It is also possible to consider elliptic curves over finite fields; in particular, over $\mathbb{Z}_p$, where p is a prime number.

The addition formulas remain valid in the form given above (where division means multiplying by the multiplicative inverse).

Some notation:
When $m \in \mathbb{N}, P \in E$ (where E is some elliptic curve), we write $m \cdot P := \underbrace{P + P + \cdots + P}_{m \text{ times}}$;
For negative numbers m, we write $m \cdot P = \underbrace{-P - P - \cdots - P}_{|m| \text{ times}}$.

Determining the order of the group of points of an elliptic curve is hard; Schoof's algorithm, the best one known, has time complexity $O(\log^8 n)$.

However, we have the following estimate:

Theorem 7 (Hasse's Theorem). *Let E be an elliptic curve over $\mathbb{Z}_p$. Then the order of the group is $|E| = p + 1 + t$, where $|t| \leq 2\sqrt{p}$.*

4 Elliptic Curves over Rings $\mathbb{Z}_n$

Now that we know how to deal with an elliptic curve over $\mathbb{Z}_p$, where p is prime, let us recall the goal of the article: primality proving via elliptic curves.

Therefore we want to work with elliptic curves over $\mathbb{Z}_n$, where we do not know whether n is prime or composite.

For two points satisfying the congruence $y_i^2 \equiv x_i^3 + ax_i + b \mod n$, can we compute their "sum" according to the rules given in the previous section? Yes, we can do that; i.e. unless $x_2 - x_1$, $2y_1$ resp. is not invertible in the ring $\mathbb{Z}_n$. What happens if this occurs? Inverses are (usually) computed with the euclidean algorithm: for n, f we find numbers d, e such that $\gcd(n, f) = 1 = dn + ef \equiv_n ef$; so e is the inverse of f. If finding these numbers fails, the euclidean algorithm computes the (non-trivial) common divisor of f, n; which clearly proves the compositeness of n.

Hence, we can compute in the set of points of an elliptic curve over $\mathbb{Z}_n$. For our purposes, we can assume that the computation works; if not, we have also reached our goal.

In particular, if n is not prime, and p is one of its prime divisors: Every point on E (the set of points on some elliptic curve over $\mathbb{Z}_n$) corresponds to a point on E_p, the "modulo p reduced" elliptic curve, where the coefficients a, b of E are reduced modulo p.

So $P = (x, y)$ corresponds to $P_p := (x \mod p, y \mod p)$ and $\mathcal{O}$ corresponds to $\mathcal{O}_p$.

Because computing modulo induces a ring-homomorphism, we have $(P + Q)_p = P_p + Q_p$ and $(m \cdot P)_p = m \cdot P_p$ (if the left-hand side of each equation is defined).

5 ECPP

Theorem 8 (Elliptic Curves and Primes). [‡] *Let $n > 1$, with $\gcd(n, 6) = 1$[§], and E be an elliptic curve over $\mathbb{Z}_n$. If there is a number m with a prime divisor $q > (\sqrt[4]{n} + 1)^2$ and a point $P \in E$ such that*

$$m \cdot P = \mathcal{O} \tag{3}$$
$$\frac{m}{q} \cdot P \neq \mathcal{O} \tag{4}$$

then n is prime.

Proof. Let $Q = (m/q) \cdot P$. Assume n is not prime. Then there is a prime divisor $p \leq \sqrt{n}$ of n.

Consider the reduced elliptic curve E_p: We have $Q_p \neq \mathcal{O}_p$, and $q \cdot Q_p = (q \cdot Q)_p = (q \cdot (m/q) \cdot P)_p = (m \cdot P)_p = \mathcal{O}_p$.

[‡] There is a more general version of this theorem which resembles Pocklington's Theorem more closely; However, we do not discuss the genereal version here.

[§] We have to assume that in order for our addition-formulas to work

Hence, E_p contains an element of order q. So $q \mid |E_p|$, in particular $q \le |E_p|$.
But this is a contradiction, because

$$|E_p| \le p + 1 + 2\sqrt{p} = (\sqrt{p}+1)^2 \le \left(\sqrt[4]{n}+1\right)^2 < q$$

according to our assumption and Hasse's theorem. □

Can we use this theorem to construct an efficient algorithm? There are three open parameters: m, q, P; it is not even clear that suitable values exist.

First of all, we will fix the parameter m to be $|E|$, the number of points on the curve. This ensures the first condition for all points P if n is a prime. Furthermore, if m has a sufficiently large prime divisor, there is a point with the required properties:

Theorem 9 (Existence of suitable points). *Let n be a prime (except for $2, 3$), and E an elliptic curve over $\mathbb{Z}_n$. If $m = |E|$ has a prime divisor $q > (\sqrt[4]{n}+1)^2$, then there is a point P with the required properties.*

Proof. The first condition is satisfied for all points P. First we claim that $q \nmid (m/q)$, which is equivalent to $q^2 \nmid m$; otherwise, we would have

$$m \ge q^2 > \left(\sqrt[4]{n}+1\right)^4 = \left(\sqrt{n}+2\sqrt[4]{n}+1\right)^2 > n + 2\sqrt{n} + 1$$

in contradiction to Hasse's theorem.

According to the classification theorem of finite abelian groups, E is isomorphic to the direct sum of $\mathbb{Z}_q$ and another finite group, where $\mathbb{Z}_q$ is considered additively.

The group $\mathbb{Z}_q$ has a generator (in fact, every number $1, \ldots, q-1$ will do). Such a generator g satisfies $\frac{m}{q} \cdot g \not\equiv_q 0$, because (m/q) is not a multiple of q. So $(m/q) \cdot P \neq \mathcal{O}$ holds for points P that are mapped onto a generator of $\mathbb{Z}_q$ when projected into $\mathbb{Z}_q$. □

This proposition shows the way to an algorithm:

- Pick a random elliptic curve over $\mathbb{Z}_n$.
- Find its order m.
- Does m have a sufficiently large prime divisor? If not, pick another curve.
- Check random points for the two properties, until you find a suitable point or some computation fails.

Unfortunately, some questions (theoretical and practical ones) remain unresolved:

- How can we determine the order of an elliptic curve?

- How do we factorize this number?
- Is there always a suitable elliptic curve over $\mathbb{Z}_n$?

Here are some answers:

There are indeed algorithms that count the number of points; but these are quite inefficient. We will see a way how to avoid having to count the number of points.

In order to factorize the number m, one can use the $p-1$ method, or ECM, a factoring method using elliptic curves. These algorithms split off small factors. This might leave us with a prime number, because the probability that an arbitrary number is composed from one big and many small primes is much higher than for the number to have several big prime divisors.

Still, we have to ensure that the remaining number is indeed a prime number.

This is done by recursively applying the algorithm to the smaller number which we found; before starting the recursion, we use a pseudoprime test (several times) so that we can expect the number to be prime.

In this way, we continue until the remaining number is so small that we can test it by looking it up in a table of primes or use another method.

The number of numbers that we have to test is not much greater than $\log n$, because in the worst case, the new number is slightly greater than half the previous number.

Theory could not yet give a definitive answer to the question whether there is always a number with suitable factorization in the "Hasse-interval" $[n+1-2\sqrt{n}, n+1+2\sqrt{n}]$. Clearly, this is a huge draw-back for runtime-analysis. However, experience shows that the algorithm never has any problems to find suitable elliptic curves.

One of the hints that show that ECPP is efficient is: It follows from the proof of Theorem 9 that the probability for a random point P to satisfy the conditions of Theorem 8 is $(q-1)/q \approx 1$. So we expect not having to check many points.

In practice, the algorithm has only one competitor which is about as fast: The Jacobi-sum test. But this algorithm does not yield a certificate, and is thus less popular than ECPP.

There is one question remaining: How do we find the order of E? The answer is: We do not compute it in the first place. Instead, one can find the possible orders of curves over $\mathbb{Z}_n$ using a procedure called "complex multiplication". For n and a number called discriminant, we can check whether with this setup, there is an elliptic curve over $\mathbb{Z}_n$. If so, we can determine the possible orders before we actually compute the parameters of the curve (see [2] for details).

Finally, here is the algorithm in some symbolic programming language. It uses recursion; this makes it easier to understand. In practice, one would use a while-loop structure to avoid the inefficient recursion.

5.1 The Algorithm

```
ECPP (n)
// Input: a number n which has no common divisor with 6
// Output:
//       if n is a prime: a list of all the intermediate numbers,
//       elliptic curves, their orders, etc.
//       such that using those, it is easy to check that n really is prime.
// otherwise: []; In this case, a divisor is printed directly on the screen.
{
    if (n < SMALL_PRIME)
        if (testSmallPrime(n))
            return [n];
        else return [];

    continue = true;

    while (continue)
        {
        (D, orders) = getDiscriminantAndOrders(n);

        for (i = 0; i < orders.length; i++) {
            q = removeSmallFactors (orders[i]);
            if (q > (sqrt(sqrt(n))+1)^2 AND pseudoPrime(q)
                                        AND (output=ECPP(q)) != [] )
                {
                whichOrder = i;
                continue = false;
                break;
                }
          }
        }
        // this is the recursion: We leave the loop only if q is
        // a prime and is big enough.
        // in particular, we save the output, because it is the certificate.
    // We need to repeat this part until we found a discriminant with
    // suitable factorisation.
    m = orders[whichOrder];
    E = EllipticCurve (D, m, n);

    try{
        do{
            P = E.getRandomPoint();
```

```
            Q = E.multiply( P, m/q );
            R = E.multiply( Q, q);
        }
        while (Q == E.O OR NOT R == E.O);
    }
    catch (EllipticCurveArithmeticException e){
        print e.divisor " is a divisor of " n;
        return [];
    }
    // if there was a problem during the calculation, we print the divisor
    // directly on the screen and return [], so that the higher level of
    // recursion (which might be around) knows that the tested number was
    // not prime.
    // otherwise, we know that n is prime and we found a curve, a point, etc.
    return [n, E, m, P] ++ output;
}

Subroutines, constants, etc.
[]
    // [] is the symbol for an empty list
[a,b,c]
    // a list containing a, b and c
[a,b] ++ output
    // concatenates the two lists
SMALL_PRIME
    // the point at which no more recursion is needed
testSmallPrime :: int -> boolean
    // tests a small number for primality (usually in the form of a table-lookup)
getDiscriminantAndOrders :: N -> (int, [int])
    // yields a discriminant which is suitable to generate elliptic curves
    // over Z_n and the possible orders.
    // This routine has to make sure that several function calls with the
    // same argument do not yield the same output (either probabilistic,
    // or by storing a list of previous requests).
removeSmallFactors :: N -> N
    // removes small prime factors; hoping that a large prime will remain
    // usually (p-1)-method or ECM
pseudoPrime :: N -> Bool
    // performs a pseudoprime test (several times)
EllipticCurve (D, m, n) :: (N, N, N) -> elliptic curve
    // returns an object which represents the elliptic curve over Z_n which
    // can be constructed from the discriminant D and has order m
```

References

[1] I.F. Blake, G. Seroussi & N.P. Smart: *Elliptic Curves in Cryptography*, Cambridge University Press, 1999

[2] J. Buchmann, V. Müller: *Primality Testing*, `ftp://ftp.informatik.tu-darmstadt.de/pub/TI/lecture_notes/primality.ps.gz`, 1992

[3] J. Buchmann: *Einführung in die Kryptographie*, Springer, 1999

[4] H. Cohen: *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics 138, Springer, 1995

[5] S. Hamdy: *Anwendungen elliptischer Kurven in der Kryptographie*, `http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/hamdy/misc/aekk.ps.gz`, 1998

[6] E. Kranakis: *Primality and Cryptography*, Wiley-Teubner, 1986

[7] J.H. Silverman: *The arithmetic of elliptic curves,* Graduate Texts in Mathematics 106, Springer, 1986