Primality Proving with Elliptic Curves

Laurent Théry, Guillaume Hanrot

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Primality Proving with Elliptic Curves

Laurent Théry — Guillaume Hanrot

N° 6155

Mars 2007

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6155--FR+ENG

Primality Proving with Elliptic Curves

Laurent Théry , Guillaume Hanrot

Thème SYM — Systèmes symboliques
Projets Marelle et Cacao

Rapport de recherche n° 6155 — Mars 2007 — 18 pages

Abstract: Elliptic curves are fascinating mathematical objects. In this paper, we present the way they have been represented inside the Coq system, and how we have proved that the classical composition law on the points is internal and gives them a group structure. We then describe how having elliptic curves inside a prover makes it possible to derive a checker for proving the primality of natural numbers.

Key-words: elliptic curve, formalising mathematics, primality proving

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Prouver la primalité avec des courbes elliptiques

Résumé : Les courbes elliptiques sont des objets mathématiques fascinants. Dans ce travail, nous présentons la façon dont elles ont été représentées dans le système COQ, et comment nous avons prouvé le fait que la loi de composition classique est bien interne et munit l'ensemble des points de la courbe d'une structure de groupe. Nous décrivons alors comment cela permet d'obtenir un vérificateur permettant de prouver la primalité de nombres naturels.

Mots-clés : courbe elliptique, formalisation des mathématiques, preuves de primalité

1 Introduction

Elliptic curves are a very classical topic of interest in number theory and algebraic geometry, and the basis for a very large body of theory, with lots of far-fetched generalisations. Their most famous application is, without any doubt, the proof of Fermat's last theorem. They are also commonly used in algorithmic number theory since the mid 80's for factoring integers [15] or proving primality [8], but have also made their way more recently in cryptography, see e.g. [5].

It is then natural to try formalising them inside a proof system. In this work, the main property we are interested in is their group structure. The difficult part of the formalisation is in proving associativity. In our formal proof, we follow an elementary algebraic approach that is particularly suited to proof systems. Our main source of inspiration has been the proof proposed by Stefan Friedl [7]. The proof is very technical and consists in massaging equalities over an abstract field under various conditions. The only sour point is that some of these equalities (exactly 3) are so huge that they cannot be reasonably handled by a human being. In his paper, Stefan Friedl advocates the use of a computer algebra like CoCoA [4] to check these equalities. Reflecting these computations inside the proof system is the main difficulty of the formal proof. In this paper, we explain how this has been done inside the proof system Coq [22].

Once the group structure has been established, we can use elliptic curves in a very effective way to prove the primality of some natural numbers. For this, we use *prime certificates*: various primality proving algorithms provide the user with a compact output, a *certificate,* with the property that it can be easily checked, whereas generating it is usually cpu-intensive.

We have already played with the idea of prime certificates and in particular of Pocklington certificates in [9, 10]. This allows us to prove the primality of some large numbers like the millennium prime (a prime number with exactly 2000 digits discovered by John Cosgrave) or the 27^{th} Mersenne prime $2^{44497} - 1$. The only drawback of the certificates we have been using so far is that they only apply to a restricted class of prime numbers. For example, a Pocklington certificate can be generated for N if $N - 1$ can be factored; in particular, we are not able to generate a Pocklington certificate in the case where $(N - 1)/2$ turns out not to be prime (this can be shown by compositeness tests) but it is impossible to factor it, which is a typical case as soon as N grows large. In contrast, under some deep (typically Crámer's conjecture) conjectures of analytic number theory, elliptic curve certificates exist for any prime and can be found in polynomial time. At the moment, the largest prime for which an elliptic certificate has been generated has 20562 decimal digits. Ten days are needed to verify its certificate [18].

The paper is organised as follows. In Section 2, we recall what elliptic curves are and explain how their group structure has been formally established. In Section 3, we focus on prime certificate and show how it is possible to formally derive a checker for elliptic curve certificates.

2 Elliptic curves and group structure

In algebraic geometry, an elliptic curve is defined as follows.

Definition. Let K be a commutative field. An elliptic curve $E(K)$ is a pair (C, P) where C is a (complete) algebraic curve of genus 1 defined over $\mathbb{C}$ and P a point of $C(K)$.

This definition is not very tractable in practice, and the following characterization, which follows easily from the Riemann-Roch Theorem [21], is rather used as a definition.

Proposition. Every elliptic curve over K admits an affine plane model of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{1}$$

where the distinguished point of the definition is the projective point $(0 : 1 : 0)$, i.e. the direction $y \to \infty$. If the characteristic of K is greater than 5, the equation further simplifies to

$$y^2 = x^3 + Ax + B. \tag{2}$$

Conversely, a nonsingular curve of the form (1) – or (2) if the characteristic is greater than 5 – defines an elliptic curve. For (2) the non-singularity assumption is equivalent to $4A^3+27B^2 \neq 0$.

Most of the interest for elliptic curves stems from the fact that they come with a natural group structure. A composition law $(P, Q) \mapsto P + Q$ over $E(K)$ can be constructed as follows:

- The neuter element is the point at infinity;
- Let (PQ) be the line through P and Q (the tangent to the curve at P if $P = Q$).
- Let R be the third point of intersection of (PQ) and K.
- The point $P + Q$ is the symmetrical of R with respect to the line $y = 0$.

An alternative way of seeing the composition law is that three aligned points add up to zero, which is the point at infinity.

Theorem 1 *$(E(K), +)$ is a group.*

The only difficult part is associativity. Classically, there are two roads for proving this. The "low road" uses either computer algebra (the one that we shall carefully take later on) or elementary but subtle geometric considerations. The "high road" instead defines a group by taking the quotient of the free abelian group over $E(K)$ by a subgroup of relations (generated by the relations "three aligned points add up to zero"), and proceeds to prove that each class contains one and only one representative consisting of a single point (so that the underlying set is in bijection with $E(K)$) and that the operation is the same as $+$. All these are consequences of the (non-trivial) Riemann-Roch Theorem.

In our formalisation, we start from (2). We take an arbitrary field K of characteristic other than 2 (this is the only assumption that is needed to prove the group structure) and two elements A and B such that $4A^3 + 27B^2 \neq 0$. The type `elt` that represents the elements of the curve is defined as follows

```
Inductive elt: Set :=
  inf_elt: elt
| curve_elt (x: K) (y: K) (H: y^2 = x^3 + A * x + B):  elt.
```

An element of type `elt` is either 0 (`inf_elt`) or an element of the curve (`curve_elt`) that contains an `x`, a `y` and a proof `H` that the point $(\mathtt{x}, \mathtt{y})$ is on the curve.

For the operations on the curve, we need to define the opposite and the addition. Given a point p of the curve, we define $-p$ as

- If $p = 0$ then $-p = 0$
- If $p = (x, y)$ then $-p = (x, -y)$

Given two points p_1 and p_2 on the curve, we define $p_1 + p_2$ as

- If $p_1 = 0$ then $p_1 + p_2 = p_2$
- If $p_2 = 0$ then $p_1 + p_2 = p_1$
- If $p_1 = -p_2$ then $p_1 + p_2 = 0$
- If $p_1 = p_2 = (x, y)$ and $y \neq 0$ then

  let $l = (3x^2 + A)/2y$ and $x_1 = l^2 - 2x$ in
  $p_1 + p_2 = (x_1, -y - l(x_1 - x))$.
- If $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ and $x_1 \neq x_2$ then

  let $l = (y_2 - y_1)/(x_2 - x_1)$ and $x_3 = l^2 - x_1 - x_2$ in
  $p_1 + p_2 = (x_3, -y_1 - l(x_3 - x_1))$.

We also use $p_1 - p_2$ as a shortcut for $p_1 + (-p_2)$. Note that, as points of the curve contain a proof that they belong to the curve, the definition of these operations must also incorporate a proof that these operations are internal. For example, the opposite is represented by a function `opp` whose type is `elt` → `elt` and its definition is

```
Definition opp p :=
   match p with
     inf_elt => inf_elt
   | curve_elt x y H => curve_elt x (-y) opp_lem
   end.
```

where `opp_lem` represents the proof that if we know that $y^2 = x^3 + Ax + B$ (this is `H`) then we have $(-y)^2 = x^3 + Ax + B$.

We get that 0 is the neuter element, $-p$ is the opposite of p, and the addition is commutative directly from the definitions. Proving associativity is rather technical. A detailed description of the formal proof is given in [23]. Here, we only sketch the proof and discuss the most interesting issues.

As the definition of addition contains five cases, the proof heavily relies on case analysis. The addition $p_1 + p_2$ is trivial if $p_1 = 0$ or $p_2 = 0$ or $p_1 = -p_2$. When the addition is non-trivial we

$$\begin{array}{rll}
 & x_1 - x_2 \neq 0 & \wedge \\
 & x_4 - x_3 \neq 0 & \wedge \\
 & x_2 - x_3 \neq 0 & \wedge \\
 & x_5 - x_1 \neq 0 & \wedge \\
 & y_1^2 = x_1^3 + Ax_1 + B & \wedge \\
 & y_2^2 = x_2^3 + Ax_2 + B & \wedge \\
 & y_3^2 = x_3^3 + Ax_3 + B & \wedge \\
 & x_4 = (y_1 - y_2)^2/(x_1 - x_2)^2 - x_1 - x_2 & \wedge \\
 & y_4 = -(y_1 - y_2)/(x_1 - x_2)(x_4 - x_1) - y_1 & \wedge \\
 & x_6 = (y_4 - y_3)^2/(x_4 - x_3)^2 - x_4 - x_3 & \wedge \\
 & y_6 = -(y_4 - y_3)/(x_4 - x_3)(x_6 - x_3) - y_3 & \wedge \\
 & x_5 = (y_2 - y_3)^2/(x_2 - x_3)^2 - x_2 - x_3 & \wedge \\
 & y_5 = -(y_2 - y_3)/(x_2 - x_3)(x_5 - x_2) - y_2 & \wedge \\
 & x_7 = (y_5 - y_1)^2/(x_5 - x_1)^2 - x_5 - x_1 & \wedge \\
 & y_7 = -(y_5 - y_1)/(x_5 - x_1)(x_7 - x_1) - y_1 & \\
\Rightarrow & x_6 - x_7 = 0 &
\end{array}$$

where

$$(x_4, y_4) = p_1 \oplus_g p_2,\ (x_5, y_5) = p_2 \oplus_g p_3,$$

$$(x_6, y_6) = (p_1 \oplus_g p_2) \oplus_g p_3 \text{ and } (x_7, y_7) = p_1 \oplus_g (p_2 \oplus_g p_3)$$

Figure 1: Condition for the x component of the first case.

use the notation $p_1 \oplus p_2$. In turn, this addition can correspond to the tangent case, $p_1 \oplus_t p_2$, where $p_1 = p_2$ or the general case, $p_1 \oplus_g p_2$, where $p_1 \neq p_2$. For associativity, it is easy to discharge the trivial cases and what remains to be proved is then that $p_1 \oplus (p_2 \oplus p_3) = (p_1 \oplus p_2) \oplus p_3$. Each of these additions can be tangent or general, this can be further reduced to four main cases:

1. $p_1 \oplus_g (p_2 \oplus_g p_3) = (p_1 \oplus_g p_2) \oplus_g p_3$.
2. $p_1 \oplus_g (p_2 \oplus_t p_2) = (p_1 \oplus_g p_2) \oplus_g p_2$.
3. $p_1 \oplus_g (p_1 \oplus_g (p_1 \oplus_t p_1)) = (p_1 \oplus_t p_1) \oplus_t (p_1 \oplus_t p_1)$
4. $p_1 \oplus_g (p_2 \oplus_g (p_1 \oplus_g p_2)) = (p_1 \oplus_g p_2) \oplus_t (p_1 \oplus_g p_2)$

The first three cases are discharged using explicit computations while the last case is derived using auxiliary properties such as the cancellation law, i.e. if $p + p_1 = p + p_2$ then $p_1 = p_2$. To understand what it means to perform cases $1 \div 3$ by explicit computations, let us consider the first case. If we take the x component of equation 1 and unfold the definition of addition, we get the conditional equation to be satisfied given in Figure 1. We can now transform this equation into a polynomial one and then use Buchberger algorithm [3] to check that the equation is satisfied. This is where the computer algebra system comes into play. Integrating Buchberger algorithm in a safe way inside a prover is possible and has already been done [6, 13], but cases $1 \div 3$ can be decided by a much simpler strategy. To illustrate this strategy, let us consider a more elementary example, i.e. the proof that the addition is internal in the tangent case. It amounts to proving that

$$\begin{array}{lll} & y^2 = x^3 + Ax + B & \wedge \\ & l = (3x^2 + A)/2y & \wedge \\ & x_1 = l^2 - 2x & \wedge \\ & y_1 = -y - l(x_1 - x) & \wedge \\ \Rightarrow & y_1^2 = x_1^3 + Ax_1 + B & \end{array}$$

The first step is to get rid of division by normalising the equation into an expression of the form $N/D = 0$ where N and D are two polynomial expressions. We are then left with proving that $N = 0$. In our example, a naive normalisation gives

$$2^{10}y^8 - 2^{10}y^6x^3 - 2^{10}Ay^6x - 2^{10}By^6 = 0$$

The second step is to replace all the occurrences of y^2 by $x^3 + Ax + B$.

$$2^{10}(x^3+Ax+B)^4 - 2^{10}(x^3+Ax+B)^3x^3 - 2^{10}A(x^3+Ax+B)^3x - 2^{10}B(x^3+Ax+B)^3 = 0$$

The final step is to use distributivity, associativity, commutativity and collect equal monomials so that everything cancels out.

Note that even a dedicated computer algebra system like Maple may fail, by using the general Gröbner machinery, to prove such a large identity. The remedy, like here, is to perform "by hand" the reductions modulo the ideal $< y_1^2 - x_1^3 - Ax_1 - B, y_2^2 - x_2^3 - Ax_2 - B >$, by using the fact that the basis given for this ideal is already a Gröbner basis for the lex ordering $y_1 > y_2 > x_1 > x_2$.

To sum up, three ingredients are needed to automate the proof of the three lemmas corresponding to cases $1 \div 3$: a procedure to normalise polynomial expressions (last step), a procedure to rewrite polynomial expressions (second step) and a procedure to normalise rational expressions (first step).

The procedure to normalise polynomial expressions was already present in COQ and is described in [11]. Its main characteristic is to use an internal representation of polynomials in Horner form. This representation is unique up to variable ordering. Given a polynomial P and a variable x, we write P as $P_1 + x^iQ_1$ where x does not occur in P_1 and is not a common factor in Q_1 and we proceed on P_1 and Q_1 recursively. The normal form is further simplified writing 0 for $m0$ and P for $0 + P$ and $P + 0$. For example

$$2^{10}y^8 - 2^{10}y^6x^3 - 2^{10}Ay^6x - 2^{10}By^6$$

is represented as

$$y^6((B(-2^{10}) + x(A(-2^{10}) + x^2(-2^{10}))) + y^2 2^{10})$$

for the ordering $y > x > A > B$. Once the procedures to add and multiply polynomials in Horner form have been defined, the normal form of a polynomial P is obtained by a simple structural traversal of P. As described in [11], this leads to a very effective way of proving ring equalities by just checking that the normal form of the left side of the equality is structurally equal to the normal form of the right side of the equality.

Rewriting has been implemented in a naive way on Horner representation. It uses a simple procedure that, given a monomial m, splits a polynomial P in a pair (P_1, Q_1) in such a way that $P = P_1 + mQ_1$. Now rewriting once with the equation $m = R$ is performed by building the polynomial $P_1 + RQ_1$. For example, if we want to rewrite the previous polynomial with the equation $y^2x^2 = z + t$, we first split the polynomial

$$y^6((B(-2^{10}) + x(A(-2^{10}) + x^2(-2^{10}))) + y^2 2^{10})$$

into

$$\textbf{(}\ y^6((B(-2^{10}) + x(A(-2^{10}))) + y^2 2^{10})\ \textbf{,}\ y^4x(-2^{10})\ \textbf{)}$$

We then build

$$(y^6((B(-2^{10}) + x(A(-2^{10}))) + y^2 2^{10}))\ + (z + t)\,(y^4x(-2^{10}))$$

and normalise it. For rewriting with respect to a list of equations, we just iterate the single rewriting operation for all the equations in a fair way. Note that in our specific case we are always rewriting with equations of the type $y_i^2 = x_i^3 + Ax_i + B$, so we take a special care in choosing a variable ordering for the Horner representation such that the y_i are privileged. By doing so, the splitting procedure has only to visit a small part of the polynomial.

Normalising rational equalities is not as simple as for polynomial expressions. When reducing the initial expression to a common denominator, the expressions for the numerator and the denominator can grow very quickly. Our experiments with the proofs of cases $1 \div 3$ have shown that getting a normalised numerator of a reasonable size was mandatory in order to be able to complete the following rewriting phase. So, when normalising, some reductions are needed. For example, when normalising $P_1/Q_1 + P_2/Q_2$ we can do much better than building the naive fraction $(P_1Q_2 + P_2Q_1)/Q_1Q_2$. Unfortunately, gcd algorithms for multivariate polynomials are far more complex to implement than the simple Euclidean algorithm for univariate polynomials. So, we have just implemented a syntactic heuristic. Our heuristic factorises the products that are in common between Q_1 and Q_2 by a simple structural traversal of the lists of products that compose Q_1 and Q_2. For example, normalising $1/x + 1/xy + 1/y = 0$ gives $y + 1 + x = 0$.

The three procedures (polynomial normaliser, rewriter and rational normaliser) are put together in a single tactic `field`. This tactic works on arbitrary field structures. Given F a rational expression, calling the tactic `field[H₁ H₂]` attempts to solve the goal $F = 0$ using the rewriting rules `H₁` and `H₂`. The three procedures have been defined inside the COQ logic using the two-level approach [1]. Their correctness can then be stated inside the proof system and formally proved. This ensures once and for all applications that the `field` tactic only performs valid simplifications. A key aspect of this tactic is its efficiency. Proving the group law requires non-trivial computation. Having a relatively efficient procedure is mandatory to be able to complete the proof. On an Intel Xeon (2.66 GHz) with 2Gb of RAM, compiling the definition of elliptic curves up to associativity takes one minute and twenty seconds in COQ.

3 Prime certificate

The group structure on $E(\mathbb{Z}/p\mathbb{Z})$ is used in the same way as the group structure on $(\mathbb{Z}/p\mathbb{Z})^*$ is used in Pocklington's $p-1$ test, which we recall:

Proposition 1 (Pocklington) *Let N be an integer. Assume that there exists a coprime to N and s such that*

- $a^s = 1 \bmod N$ *;*
- *for all prime divisor p of s, one has $gcd(a^{s/p} - 1, N) = 1$.*

Then, for any prime $q|N$, one has $q = 1 \bmod s$, so that if $s \geq \sqrt{N}$, N is prime.

The main drawback of Pocklington's test is the fact that if one is not able to factor $N-1$ to some extent (a little thought shows that the first condition implies, if N is actually prime, $s|N-1$, since the conditions stated imply that s is the order of $a \bmod N$), then there is no way one can apply the theorem.

The following theorem is thus much more versatile, the main point being that, under reasonable conjectures, we are always able to find, in polynomial time, a curve over $\mathbb{Z}/N\mathbb{Z}$ with cardinality which can be factored.

We make use of points in projective coordinates, which is required to give a meaningful statement. This allows to give formulas without division for the group law, which thus remain meaningful in any ring – since $\mathbb{Z}/N\mathbb{Z}$ can be assumed to be a field only after N has been proved prime...

Proposition 2 (Goldwasser-Kilian) *Let N be an integer. Assume that there exist an elliptic curve $y^2 = x^3 + Ax + B$ with $A, B \in \mathbb{Z}$ and $gcd(4A^3 + 27B^2, N) = 1$, a point $P = (x_P : y_P : 1)$ such that $y_P^2 = x_P^3 + Ax_P + B \bmod N$, and an integer s such that*

- $s.P = (0 : 1 : 0) \bmod N$ *;*
- *for all prime $p|s$, $(s/p).P = (x_p : y_p : z_p) \bmod N$ with $gcd(z_p, N) = 1$.*

Then, for all prime $q|N$, we have the number of points of $E(\mathbb{Z}/q\mathbb{Z})$ is such that $\#E(\mathbb{Z}/q\mathbb{Z}) = 0 \bmod s$.

In view of the Hasse-Weil bound [21], which states that for all prime q and elliptic curve E over $\mathbb{Z}/q\mathbb{Z}$, one has $\#E(\mathbb{Z}/q\mathbb{Z}) \leq (\sqrt{q}+1)^2$, we see that N is prime as soon as $s > \sqrt{q} + 1$. In the sequel, we shall use the slightly weaker but much easier bound $\#E(\mathbb{Z}/q\mathbb{Z}) \leq 2q + 1$, which shows that N is prime as soon as $s > \sqrt{2q+1}$.

An elliptic curve certificate is thus defined recursively by:

$$C_N := (N, A, B, x_P, y_P, s, (q_1, C_{q_1}), \ldots, (q_k, C_{q_k})),$$

where the q_i are the prime factors of s, and checking the certificate amounts to checking the hypotheses of the theorem, hence modular arithmetic.

Computing a certificate is a much harder problem. The most efficient way is to use the theory of complex multiplication, after Atkin and Morain. For a more extensive bibliography, the interested reader can refer to the nice survey of primality proving by Morain [19].

The idea behind the formal proof of Proposition 2 is to relate computations with projective coordinates done in $\mathbb{Z}/N\mathbb{Z}$ with the same computations with standard coordinates done in $\mathbb{Z}/q\mathbb{Z}$ when q is a prime divisor of N. We use a non-standard definition of projective coordinates where we still single out the zero. A point is either 0 or a triplet $(x : y : z)$. We also define the projection t_q as $t_q(0) = 0$ and $t_q((x : y : z)) = (x \, mod \, q/z \, mod \, q, y \, mod \, q/z \, mod \, q)$. Note that the projection in the second case is possible only if $z \, mod \, q$ has effectively an inverse in $\mathbb{Z}/q\mathbb{Z}$ for prime divisor q of N, so only if $gcd(n, N) = 1$. In the following, we write the property of n having an inverse as $inv_N(n)$.

We have to be careful when computing in $\mathbb{Z}/N\mathbb{Z}$ since we want the computation in $\mathbb{Z}/N\mathbb{Z}$ to project faithfully to the same computation in $\mathbb{Z}/q\mathbb{Z}$. Suppose that we test to zero a variable n. Obviously if we have $n = 0$, we also have $n \, mod \, q = 0$, but the converse may not be true: $n \neq 0$ does not imply that $n \, mod \, q \neq 0$. So every time we have a test to zero on a variable n, we have to check the extra condition that $inv_N(n)$ to ensure that the computation on $\mathbb{Z}/q\mathbb{Z}$ would take the same branch. For this reason, all the functions that we have defined for $\mathbb{Z}/N\mathbb{Z}$ take an extra argument that represents the side conditions so far and return a pair composed of the result plus the new side conditions. We encode all the side conditions in a single number using the property that $inv_N(mn)$ implies $inv_N(m)$ and $inv_N(n)$.

A result (p_1, sc_1) of a computation can be correctly interpreted only if we have $inv_N(sc_1)$ when $p_1 = 0$ or $inv_N(z_1 sc_1)$ when $p_1 = (x_1 : y_1 : z_1)$. In the following, we use the notation $\lceil p_1, sc_1 \rceil$ to indicate that this property holds. To give a concrete example, consider the function *double* that doubles a point p_1 under the initial side condition sc_1. Its definition is

```
$double(p_1, sc_1) =$
  if $p_1 = 0$ then $(0,\ sc_1)$ else
  let $(x_1 : y_1 : z_1) = p_1$ in
    if $y_1 = 0$ then $(0, z_1 sc_1)$ else
    let $m = 3x_1^2 + Az_1^2$ and $l = 2y_1 z_1$ in
    let $l_2 = l^2$ and $x_2 = m^2 z_1 - 2x_1 l_2$ in
    $((x_2 l : l_2(x_1 m - y_1 l) - x_2 m : z_1 l_2 l),\ sc_1)$
```

and its correctness is stated as

$$\forall p_1 \, sc_1 \, p_2 \, sc_2, \ \texttt{let} \ (p_2, sc_2) = double(p_1, sc_1) \ \texttt{in} \ \lceil p_2, sc_2 \rceil \Rightarrow t_q(p_2) = 2.t_q(p_1)$$

Another key property is that the final side condition includes the initial one

$$\forall p_1 \, sc_1 \ p_2 \, sc_2, \ \texttt{let}(p_2, sc_2) = double(p_1, sc_1) \ \texttt{in} \ \lceil p_2, sc_2 \rceil \Rightarrow \lceil p_1, sc_1 \rceil$$

Knowing that these two properties must hold, we can now explain how the side conditions have been generated in the body of the function *double*. When p_1 is zero, no condition has to be added. When p_1 is $(x_1 : 0 : z_1)$, the result is zero so we include the final side condition that z_1 must be invertible. Finally, when p_1 is $(x_1 : y_1 : z_1)$ with $y_1 \neq 0$, a defensive final side conditions would be $y_1 z_1 sc_1$ to insure that y_1 and z_1 are invertible but this is not necessary since if we look at the z component, $z_1 l_2 l$, of the resulting point, that contains already the product $y_1 z_1$.

The function *add* that adds two points is defined in a similar way. With the two functions *double* and *add*, we can define the function *scal* that adds n times the point p_1 under the side condition sc_1:

```
$scal(n, p_1, sc_1) =$
    if $n = 1$ then $(p_1, sc_1)$ else
    if $n$ is even then let $(p_2, sc_2) = double(p_1, sc_1)$ in $scal(n/2, p_2, sc_2)$ else
    let $(p_2, sc_2) = double(p_1, sc_1)$ in
    let $(p_3, sc_3) = scal((n-1)/2, p_2, sc_2)$ in
    $add(p_1, p_3, sc_3)$
```

and its correctness is stated as

$$\forall n\, p_1\, sc_1,\ p_2\, sc_2,\ \texttt{let}\ (p_2, sc_2) = scal(n, p_1 sc_1)\ \texttt{in}\ \lceil p_2, sc_2 \rceil \Rightarrow t_p(p_2) = n.t_p(p_1)$$

Given a list $l = [(q_1, \alpha_1), \ldots, (q_n, \alpha_n)]$ such that $s = q_1^{\alpha_1} \ldots q_n^{\alpha_n}$, checking that $s.a = 0$ and $(s/q_i).a \neq 0$ for $1 \leq i \leq n$ is done by factorising computations as much as possible. We first compute $p_1 = (s/(q_1 \ldots q_n)).a$. The function *scalL* that, given a point p_1 and a list l such that $l = [q_1, \ldots, q_n]$ and a side condition sc_1, verifies that $(q_1 \ldots q_n).p_1 = 0$ and $(q_1 \ldots q_n/q_i).p_1 \neq 0$ for every q_i is defined as follows

```
$scalL(l, p_1, sc_1) =$
    if $length(l) = 0$ then $(p_1, sc_1)$ else
    let $(p_2, sc_2) = scal(hd(l), p_1, sc_1)$ in
    let $(p_3, sc_3) = scal\_list(tl(l), p_1, sc_2)$ in
        if $p_3 = 0$ then $(0, 0)$ else
        let $(x_3, y_3, z_3) = p_3$ in $scalL(tl(l), p_2, z_3 sc_3)$
```

where the functions *hd* and *tl* return the head element and the tail of a list respectively and the function *scal_list* applied to l and p_1, where $l = [q_i, \ldots, q_n]$, computes $(q_i \ldots q_n).p_1$. In every recursive call of the function *scalL*, p_3 represents one of the $(q_1 \ldots q_n/q_i).p_1$. If p_3 is 0, we put

0 in the side condition in order to invalidate the final result. If p_3 is not 0, $t_p(p_3)$ must also be non-zero so we add its z component into the side condition.

The actual representation of COQ certificates is a list of elementary certificates rather than a tree. This list must be well-founded in the sense that the primality of a number that is needed by an elementary certificate at position i is justified by another certificate at position $j > i$. An elementary certificate is either an immediate certificate, a Pocklington certificate or an elliptic certificate. An immediate certificate is for numbers whose primality is given by a predefined theorem. In our library there are predefined theorems for the first 5000 prime numbers. Our elliptic certificates slightly differ from the standard ones to accommodate the certificates produced by the tool PRIMO [17]. An elliptic certificate is composed of seven elements: $\{N, A, B, x_1, y_1, n_0, [(q_1, \alpha_1), \ldots, (q_n, \alpha_n)]\}$:

- N is the number to be proved prime;
- A and B are the parameters of the elliptic curve $y^2 = x^3 + Ax + B$;
- x_1 and y_1 are the coordinates of the initial point, in projective coordinates the initial point is $p_0 = (x_1 : y_1 : 1)$;
- n_0 is an initial scalar, so the point whose order is checked is $n_0.p_0$.
- $[(q_1, \alpha_1), \ldots, (q_n, \alpha_n)]$ is a complete factorisation of the order s of $n_0.p_0$, i.e. $s = q_1^{\alpha_1} \ldots q_n^{\alpha_n}$ where all the q_i are prime.

An example of such a certificate is

$$
\begin{array}{l}
\{\\
\quad 329719147332060395689499,\\
\quad -94080,\\
\quad 9834496,\\
\quad 0,\\
\quad 3136,\\
\quad 8209062,\\
\quad [(40165264598163841, 1)]\\
\}
\end{array}
$$

It allows to assess the primality of 329719147332060395689499 knowing that 40165264598163841 is prime with the curve $y^2 = x^3 - 94080x + 9834496$ and the point $8209062.(0, 3136)$ whose order is 40165264598163841.

The function *testCertif* that checks an elliptic certificate c is defined as follows:

$testCertif(c) =$

let $(N, A, B, x_1, y_1, n, l) = c$ in

let $p_0 = (x_1, y_1, 1)$ in

let $sc_0 = 4A^3 + 27B^2$ in

```
let $(p_1, sc_1) = scal(n, p_0, sc_0)$ in
let $s = mull(l)$ in
let $(n', l') = split(l)$ in
let $(p_2, sc_2) = scal(n', p_1, sc_1)$ in
let $(p_3, sc_3) = scalL(l', p_2, sc_2)$ in
if $N > 1$ and $odd(N)$ and $4N < (s-1)^2$ and
   $y_1^2 \, mod \, N = (x_1^3 + Ax_1 + B) \, mod \, N$ and
   $p_3 = 0$ and $gcd(sc3, N) = 1$
then *true* else *false*
```

where, if $l = [(q_1, \alpha_1), \ldots, (q_n, \alpha_n)]$, the function *mull* applied to l returns $q_1^{\alpha_1} \ldots q_n^{\alpha_n}$ and the function *split* applied to l returns $(q_1^{\alpha_1 - 1} \ldots q_n^{\alpha_n - 1}, \, [q_1, \ldots, q_n])$. The correctness of the function *testCertif* is stated as

$$\forall c, (testCertif(c) = true \;\wedge\; (\forall q \, \alpha, (q, \alpha) \in getL(c) \Rightarrow prime(q))) \Rightarrow prime(getN(c))$$

where *getN* and *getL* are the accessor functions for the first component and the last component of the certificate respectively.

Proving the correctness of the function *testCertif* is straightforward. It is standard program verification. Getting the side conditions right was the only tricky part. Note that it would be very difficult to detect a missing side condition just by testing. The program only returns a boolean value and it is mainly applied to certificates for which a positive answer is expected. Missing a side condition never invalidates valid certificates. Compared to standard checkers, we had to compromise with the bound on the order s of an element over $\mathbb{Z}/q\mathbb{Z}$. We use the naive bound $4N < (s-1)^2$ that is easy to establish. Standard checkers use the sharper Hasse-Weil bound. Getting this bound formally is a challenge on its own.

4 Some benchmarks

Figure 2 shows some benchmarks on prime numbers from 25 to 250 decimal digits. These benchmarks have been done on a processor Intel Xeon (2.66 GHz) with 2Gb of RAM. For each number length, there are nine examples. We generate the i^{th} example of length n by starting with the digit i and repetitively incrementing the previous digit by 1 modulo 10 to get the next digit. Once we have n digits, we just take the first prime number above this number. For example, the nine prime numbers with 25 digits are:

		25	50	75	100	125	150	175	200	225	250
	$n-1$	1	0	4	2	3	5	5	2	6	7
1	*ell*	1	4	8	13	14	17	20	26	28	28
	time	0.2	8	36	105	297	523	865	1570	2353	3018
	$n-1$	0	5	3	3	2	3	4	4	3	6
2	*ell*	2	1	7	10	19	21	24	24	27	35
	time	0.9	4	26	105	333	569	1007	1704	2462	3573
	$n-1$	2	0	1	4	4	1*	4	3	5	7
3	*ell*	2	1	7	10	19	20*	24	25	27	35
	time	0.1	11	42	132	288	513*	1277	1774	2477	2960
	$n-1$	2	3	4	0	4	6	3	3	7	5
4	*ell*	2	4	7	12	12	9	22	27	30	40
	time	1.1	11	49	130	257	339	948	1754	2275	4688
	$n-1$	1	3	3	5	3	5	8	3	4	3
5	*ell*	1	2	6	9	11	20	20	29	33	37
	time	0.7	3	36	143	264	514	1092	1859	2882	4090
	$n-1$	1	2	4	2*	4	2	7	4	2	1
6	*ell*	0	3	8	12*	14	19	21	22	34	33
	time	0.1	6	39	187*	223	479	977	1317	2811	3727
	$n-1$	0	1	5	2	1	2	4	5	7	3
7	*ell*	2	4	10	14	16	19	27	27	28	34
	time	0.8	11	54	148	329	453	1259	1514	2498	3327
	$n-1$	2	2	3	7	4	5	2	7	2	1
8	*ell*	0	4	7	6	17	20	28	31	34	40
	time	0.1	6	44	91	310	489	1442	2252	2817	3721
	$n-1$	2	2	3	5	7	4	5	7	5	5
9	*ell*	1	4	10	13	9	20	27	26	29	33
	time	0.4	7	46	139	194	549	1229	2061	2509	3645

Figure 2: Some benchmarks for numbers from 25 to 250 digits.

1234567890123456789012353, 2345678901234567890123567,
3456789012345678901234573, 4567890123456789012345689,
5678901234567890123456797, 6789012345678901234567903,
7890123456789012345678973, 8901234567890123456789017,
9012345678901234567890149

To generate certificates, we automatically convert the ones produced by the program PRIMO. For each example, we give the number of Pocklington certificates and elliptic certificates that compose the overall certificate and the time in seconds that COQ needs to verify the certificate. Out of the 1538 elliptic certificates only two could not be proved inside COQ. They are indicated by a star in Figure 2. This comes from the fact that our condition $4N < (s-1)^2$ is stronger than the usual bound given by Hasse Theorem. The two certificates that fail were for $N = 896191029759386718510467381341$, $s = 1501356318335977$ and $N = 1352449364766564198472$3, $s = 95147824089$ respectively. Since in each case the value of N is relatively small, we could replace the elliptic certificate with an alternative Pocklington certificate. Note that this should happen only for quite small values of N, so that the corresponding parts of the certificate can always be replaced by Pocklington certificates. Another strategy, if one has access to the code looking for the certificates, is to modify it very slightly to make it generate only certificates that can be checked using the modified Hasse bound (which, somehow, corresponds to making the search for certificates "less aggressive" and content itself with certificates of "lesser quality").

All the examples and the code are available at

`http://coqprime.gforge.inria.fr/`

They compile with the current version 8.1 of the COQ system. The time to check a certificate grows quickly with the number of digits, for example we need one hour to check a number with 250 decimal digits. Note that the theory predicts that the worst-case complexity for checking a certificate for N with an underlying quadratic multiprecision arithmetic is $O((\log N)^4)$, which more or less corresponds to what we observe. Note also that it would be easy to gain a small constant factor in the current implementation by optimizing our function *scal*, eg. by using sliding window methods and/or addition-substraction chains.

In order to be able to check larger numbers, we would need two main improvements in the COQ system. First, all our computations are done inside COQ with user-defined data-structures, i.e. we use the modular arithmetic defined in [10] based on a datatype composed of 256 constructors that simulates an 8-bit arithmetic. We would need a direct access to the machine 32-bit arithmetic instead. In [10], some tests for Pocklington certificates with a native 32-bits arithmetic exhibit a speed-up of 80. Such speed-up would make it possible, in our case, to verify a 250-digit number in less than a minute. We have also experienced with extracting our code to OCAML [16] and linking it with the arbitrary-precision integer library BIGNUM [20]. We got in this case an even bigger speed-up since checking one of our 250-digit numbers only takes 0.2 second and we were capable to check in 36 seconds the millennium prime, a prime with 2000 decimal digits whose certificate generated by PRIMO contains 245 elliptic certificates and 8 Pocklington ones.

Second, we are in a purely functional setting and a large amount of time is thus spent in allocating memory for new numbers. Computations should use constant memory to be really efficient. For this, we would need COQ to accommodate destructive datastructures as proposed for ACL2 in [2] with a monadic approach.

5 Acknowledgments

Many people made this work possible. Joe Hurd was the first to draw our attention to formalising elliptic curves inside a prover [14]. After unsuccessful attempts, John Harrison revitalised our interest showing us that he could automatically solve the equation of the *x* component of the generic case in less than 3 minutes inside HOLLIGHT [12] with his integrated version of

Buchberger algorithm. Bruno Barras and Benjamin Grégoire helped us with their expertise in integrating the `field` tactic into COQ. Finally, Marcel Martin modified the PRIMO system adding an option to disable $N+1$ certificates that we are not yet capable to handle.

6 Conclusion

We believe that what is presented in this paper is a very nice illustration of a key aspect of formal verification. Defining prime numbers from scratch is very simple in a proof assistant. It is done with five definitions in COQ: one for the natural numbers, one for the addition, one for the multiplication, one for the divisibility and finally one for the primality. All these definitions are straightforward, so it is easy to get convinced that the notion of primality has been correctly captured in the proof system. Then, the fact that there exists a formal proof for each of the numbers of Figure 2 ensures that the primality of these numbers follows logically from these five definitions. For the actual proofs, we are free to use any elaborate devices like in our case elliptic certificates. Of course, this is true only if the proof system is sound. It is then crucial to keep the trusted base of the prover as small as possible. This is what we have done here, reflecting the symbolic manipulations and internally checking the prime certificates generated by PRIMO.

Another aspect that this example illustrates is how important is for a proof system not only to reason right but also to compute right. Our initial attempts in proving cases $1 \div 3$ have clearly indicated that we were on the fringe of what was actually possible to prove inside COQ: the parser had problems to handle the huge terms we were generating and using the standard rewriting tactic of COQ was generating titanic proof terms. Reflecting symbolic manipulation was the key point that made everything possible. All we have developed was generic enough to be integrated into the COQ system. Nevertheless, a way to look at what we have done is to consider that we have just been reflecting a specific rewriting strategy. It would be interesting to investigate how we could reflect a generic rewriting strategy instead and get back our symbolic manipulation by simple instantiation.

Finally, the initial motivation that has lead to this work was to see how far we could go in proving primality inside a proof system. With elliptic certificates, we have reached the current state of the art. We can now prove the primality of any number with less than 300 decimal digits. We believe that, with the future improvement of numerical evaluation inside COQ, this limit will quickly increase. Anyway, the infrastructure is there. In particular, this is, to our knowledge, the first time that associativity has been formally proved inside a proof system. We hope that this initial effort will motivate deeper mathematical formalisations with elliptic curves inside proof systems.

References

[1] Samuel Boutin. Using Reflection to Build Efficient and Certified Decision Procedures. In *TACS'97*, volume 1281 of *LNCS*, pages 515–529, Sendai, Japan, 1997.

[2] Robert S. Boyer and J Strother Moore. Single-threaded objects in ACL2. In *PADL'2001*, volume 2257, pages 9–27, 2001.

[3] Bruno Buchberger. Introduction to Gröbner Bases. In Bruno Buchberger and Franz Winkler, editors, *Gröbner Bases and Applications*, pages 3–31. Cambridge University Press, 1998.

[4] Antonio Capani, Gianfranco Niesi, and Lorenzo Robbiano. Cocoa: Computations in commutative algebra. Available at `http://cocoa.dima.unige.it/`.

[5] Henri Cohen and Gerhard Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall / CRC, 2005.

[6] Jérôme Créci and Loïc Pottier. Gb: une procédure de décision pour le système Coq. In *JFLA'2004*, pages 83–90, 2004.

[7] Stefan Friedl. An elementary proof of the group law for elliptic curves. Excerpt from undergraduate thesis, Regensburg (1998), available at `http://www.labmath.uqam.ca/~friedl/papers/AAELLIPTIC.PDF`.

[8] Shafi Goldwasser and Joe Kilian. Almost all primes can be quickly certified. In *Proceedings of the 18th STOC*, pages 316–329, 1986.

[9] Benjamin Grégoire and Laurent Théry. A Purely Functional Library for Modular Arithmetic and its Application to Certifying Large Prime Numbers. In *IJCAR*, volume 4130 of *LNCS*, pages 423–437, 2006.

[10] Benjamin Grégoire, Laurent Théry, and Benjamin Werner. A Computational Approach to Pocklington Certificates in Type Theory. In *FLOPS*, volume 3945 of *LNCS*, pages 97–113, 2006.

[11] Benjamin Grégoire and Assia Mahboubi. Proving ring equalities done right in Coq. In *TPHOLS'05*, volume 3603 of *LNCS*, pages 98–113.

[12] John Harrison. HOL light: A tutorial introduction. In *FMCAD'96*, volume 1166, pages 265–269, 1996.

[13] John Harrison. Complex quantifier elimination in HOL. In *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. University of Edinburgh, 2001.

[14] Joe Hurd. Formalized elliptic curve cryptography. Available at `http://www.cl.cam.ac.uk/~jeh1004/research/talks/elliptic-talk.pdf`.

[15] Hendrik W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. Math.*, 126:649–673, 1987.

[16] Xavier Leroy. Objective Caml. Available at `http://pauillac.inria.fr/ocaml/`, 1997.

[17] Marcel Martin. PRIMO - primality proving. Available at `http://www.ellipsa.net/`.

[18] François Morain. Certificate for $(((((((((2^3 + 3)^3 + 30)^3 + 6)^3 + 80)^3 + 12)^3 + 450)^3 + 894)^3 + 3636)^3 + 70756)^3 + 97220$. Available at `http://www.lix.polytechnique.fr/Labo/Francois.Morain`.

[19] François Morain. La primalité en temps polynomial, d'après Adleman, Huang ; Agrawal, Kayal, Saxena. *Séminaire Bourbaki*, 3(55), 2002.

[20] Valérie Ménissier-Morain. The CAML Numbers Reference Manual. Technical Report 141, INRIA, 1992.

[21] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986.

[22] The COQ development team. The Coq Proof Assistant Reference Manual v7.2. Technical Report 255, INRIA, 2002. Available at `http://coq.inria.fr/doc`.

[23] Laurent Théry. Proving the group law for elliptic curves formally. Technical Report 330, INRIA, 2007.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399