# FAST ALGORITHMS FOR COMPUTING ISOGENIES BETWEEN ELLIPTIC CURVES

A. BOSTAN AND F. MORAIN AND B. SALVY AND É. SCHOST

Abstract. We survey algorithms for computing isogenies between elliptic curves defined over a field of characteristic either 0 or a large prime. We introduce a new algorithm that computes an isogeny of odd degree $\ell$ ($\ell$ different from the characteristic) in time quasi-linear with respect to $\ell$. This is based in particular on fast algorithms for power series expansion of the Weierstrass $\wp$-function and related functions.

## 1. Introduction

In the Schoof-Elkies-Atkin algorithm (SEA) that computes the cardinality of an elliptic curve over a finite field, isogenies between elliptic curves are used in a crucial way (see for instance [4] and the references we give later on). Isogenies have also been used to compute the ring of endomorphisms of a curve [27] and isogenies of small degrees play a role in [21, 15]. More generally, in various contexts, their computation becomes a basic primitive in cryptology (see [24, 44, 34]).

Thus, in this paper, we discuss the complexity of computing isogenies of odd degree $\ell$ between elliptic curves (for the special case $\ell = 2$, formulas exist [42]; see also [14]). Remark that we can restrict to the case $\ell$ prime, since isogenies can be written as compositions of isogenies of prime degree, the case of prime powers using isogeny cycles [16, 14, 20].

We demand that the characteristic $p$ of the base field $\mathbf{K}$ be 0 or $p \gg \ell$. This restriction is satisfied in the case of interest in the application to the SEA algorithm, since otherwise $p$-adic methods are much faster and easier to use [35, 26]. Several approaches to isogeny computation are available in small characteristic: we refer to [11, 31] for an approach via formal groups, [29] for the special case $p = 2$, and [12, 13, 30] for the general case of $p$ small. The case of $p = \ell$ deserves a special treatment, see [12, 31].

Our assumption on $p$ implies that the equations of our curves can be written in the Weierstrass form

$$y^2 = x^3 + Ax + B. \tag{1}$$

In characteristic zero, the curve (1) can be parameterized by $(x, y) = (\wp(z), \wp'(z)/2)$ in view of the classical differential equation

$$\wp'(z)^2 = 4(\wp^3(z) + A\wp(z) + B) \tag{2}$$

satisfied by the Weierstrass $\wp$-function. This is the basis for our computation of isogenies. We thus prove two results, first on the computation of the Weierstrass $\wp$-function, and then on the computation of the isogeny itself.

Our main contribution is to exploit classical fast algorithms for power series computations and show how they apply to the computation of isogenies. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a

*Date*: Preliminary version 4.22 – May 5, 2006.

function such that polynomials of degree less than $n$ can be multiplied in $\mathsf{M}(n)$ base field operations. Using the algorithms of [37, 8], one can take $\mathsf{M}(n) \in O(n \log n \log\log n)$; over fields supporting Fast Fourier Transform, one can take $\mathsf{M}(n) \in O(n \log n)$. We make the standard super-linearity assumptions on the function $\mathsf{M}$, see the following section.

**Theorem 1.** *Let $\mathbf{K}$ be a field of characteristic zero. Given $A$ and $B$ in $\mathbf{K}$, the first $n$ coefficients of the Laurent expansion at the origin of the function $\wp$ defined by (2) can be computed in $O(\mathsf{M}(n))$ operations in $\mathbf{K}$.*

Section 3 gives a more precise version of this statement, that handles the case of fields of positive, but large enough, characteristic.

**Theorem 2.** *Let $\mathbf{K}$ be a field of characteristic $p$, and let $E$ and $\tilde{E}$ be two elliptic curves defined over $\mathbf{K}$ and given in the form (1).*

*Let $\ell$ be an odd integer, and suppose that an isogeny $I$ of degree $\ell$ exists between $E$ and $\tilde{E}$. Let finally $p_1$ be such that the sum of the abscissas of the nonzero points in the kernel of $I$ equals $2p_1$. Then the isogeny $I$ can be computed*

1. *in $O(\mathsf{M}(\ell))$ operations in $\mathbf{K}$, if $p = 0$ or $p > \ell$, if $p_1$ is known;*
2. *in $O(\mathsf{M}(\ell)\log\ell)$ operations in $\mathbf{K}$, if $p = 0$ or $p > 4\ell$, without prior knowledge of $p_1$.*

Taking $\mathsf{M}(n) \in O(n \log n \log\log n)$ shows that the complexity results in Theorems 1 and 2 are nearly optimal, up to polylogarithmic factors, as claimed in the abstract. Notice that the algorithms using modular equations to detect isogenies yield the value of $p_1$ as a by-product. However, in a cryptographic context, this may not be the case anymore; this is why we distinguish the two cases in Theorem 2.

This article is organized as follows. In §2, we recall known results on the fast computation of truncated power series, using notably Newton's iteration. In §3, we show how these algorithms apply to the computation of $\wp$. Then in §4, we recall the definition of isogenies and the properties we need and give our quasi-linear algorithm. In the next section, we survey previous algorithms for the computation of isogenies. Their complexity has not been discussed before; we analyze them when combined with fast power series expansions so that a comparison can be made. Finally, in §6, we report on our implementation.

## 2. A review of fast algorithms for power series

The algorithms presented in this section are well-known; they reduce several problems for power series (reciprocal, exponentiation, ...) to polynomial multiplication.

Our main tool to devise fast algorithms is Newton's iteration; it underlies the $O(\mathsf{M}(\ell))$ result reported in Theorem 1, and in (the practically important) point (1) of Theorem 2. Hence, this question receives most of our attention below, with detailed pseudo-code. We will be more sketchy on some other algorithms, such as rational function reconstruction, referring to the relevant literature.

We suppose that the *multiplication time* function $\mathsf{M}$ satisfies the following classical super-linearity inequalities (see e.g., [22]) for all $n$ and $n'$:

$$\frac{\mathsf{M}(n)}{n} \leq \frac{\mathsf{M}(n')}{n'} \quad \text{if } n \leq n' \qquad \text{and} \qquad \mathsf{M}(nn') \leq n^2\mathsf{M}(n'). \tag{3}$$

In particular, Equations (3) imply the inequality

$$\mathsf{M}(1) + \mathsf{M}(2) + \mathsf{M}(4) + \cdots + \mathsf{M}(2^i) \leq 2\mathsf{M}(2^i).$$

This is the key to show that all algorithms based on Newton's iteration below have complexity in $O(\mathsf{M}(n))$. Cantor and Kaltofen [8] have shown that one can take $\mathsf{M}(n)$ in $O(n \log n \log\log n)$; as a result, most questions addressed below admit similar quasi-linear estimates.

2.1. **Reciprocal.** Let $f = \sum_{i\geq 0} f_i z^i$ be in $\mathbf{K}[[z]]$, with $f_0 \neq 0$, and let $g = 1/f = \sum_{i\geq 0} g_i z^i$ in $\mathbf{K}[[z]]$. The coefficients $g_i$ can be computed iteratively by the formula

$$g_0 = \frac{1}{f_0} \quad \text{and} \quad g_i = -\frac{1}{f_0}\sum_{j=1}^{i} f_j g_{i-j} \quad \text{for} \quad i \geq 1.$$

For a general $f$, the cost of computing $g \bmod z^n$ with this method is in $O(n^2)$; observe nevertheless that if $f$ is a polynomial of degree $d$, the cost reduces to $O(nd)$.

To speed up the computation in the general case, we use Newton's iteration. For reciprocal computation, it amounts to computing a sequence of truncated power series $h_i$ as follows:

$$h_0 = \frac{1}{f_0} \quad \text{and} \quad h_{i+1} = h_i(2 - f h_i) \bmod z^{2^{i+1}} \text{ for } i \geq 0.$$

Then, $h_i = 1/f \bmod z^{2^i}$. As a consequence, $1/f \bmod z^n$ can be computed in $O(\mathsf{M}(n))$ operations. This result is due to Cook for an analogous problem of integer inversion [10], and to Sieveking [41] and Kung [28] in the power series case.

2.2. **Exponentiation.** Let $f$ be in $\mathbf{K}[[z]]$, with $f(0) = 0$. Given $n$ in $\mathbb{N}$, such that $2, \ldots, n-1$ are units in $\mathbf{K}$, the truncated exponential $\exp_n(f)$ is defined as

$$\exp_n(f) = \sum_{i=0}^{n-1} \frac{1}{i!} f^i \bmod z^n.$$

Conversely, if $g$ is in $1 + z\mathbf{K}[[z]]$, its truncated logarithm is defined as

$$\log_n(g) = -\sum_{i=1}^{n-1} \frac{1}{i}(1-g)^i \bmod z^n.$$

The truncated logarithm is obtained by computing the Taylor expansion of $g'/g$ modulo $z^{n-1}$ using the algorithm of the previous subsection, and taking its antiderivative; hence, it can be computed in $O(\mathsf{M}(n))$ operations.

Building on this, Brent [5] introduced the Newton iteration

$$g_0 = 1, \quad g_{i+1} = g_i(1 + f - \log_{2^{i+1}}(g_i)) \bmod z^{2^{i+1}}$$

to compute the sequence $g_i = \exp_{2^i}(f)$. As a consequence, $\exp_n(f)$ can be computed in $O(\mathsf{M}(n))$ operations as well, whereas the naive algorithm has cost $O(n^2)$.

As an application, Schönhage [36] gave a fast algorithm to recover a polynomial $f$ of degree $n$ from its first $n$ power sums $p_1, \ldots, p_n$. Schönhage's algorithm is based on the fact that the logarithmic derivative of $f$ at infinity is the generating series of its power sums, that is,

$$z^n f\left(\frac{1}{z}\right) = \exp_{n+1}\left(-\sum_{i=1}^{n} \frac{p_i}{i} z^i\right).$$

Hence, given $p_1, \ldots, p_n$, the coefficients of $f$ can be recovered in time $O(\mathsf{M}(n))$. This algorithm requires that $2, \ldots, n$ be units in $\mathbf{K}$.

2.3. **First-order linear differential equations.** Let $a, b, c$ be in $\mathbf{K}[[z]]$, with $a(0) \neq 0$, and let $\alpha$ be in $\mathbf{K}$. We want to compute the first $n$ terms of $f \in \mathbf{K}[[z]]$ such that

$$af' + bf = c \quad \text{and} \quad f(0) = \alpha$$

Let $B = b/a \bmod z^{n-1}$ and $C = c/a \bmod z^{n-1}$. Then, defining $J = \exp_n(\int B)$, $f$ satisfies the relation

$$f = \frac{1}{J}\left(\alpha + \int CJ\right) \bmod z^n.$$

Using the previous reciprocal and exponentiation algorithms, $f \bmod z^n$ can thus be computed in time $O(\mathsf{M}(n))$. This algorithm is due to Brent and Kung [7]; it requires that $2, \ldots, n-1$ be units in $\mathbf{K}$.

2.4. **First-order nonlinear differential equations.** We only treat this question in a special case, following again Brent and Kung's article [7, Theorem 5.1]. Let $G$ be in $\mathbf{K}[[z]][t]$, let $\alpha, \beta$ be in $\mathbf{K}$, and let $f \in \mathbf{K}[[z]]$ be a solution of the equation

$$f'^2 = G(z, f), \quad f(0) = \alpha, \quad f'(0) = \beta,$$

with furthermore $\beta^2 = G(0, \alpha) \neq 0$. Supposing that, for $s \geq 2$, the initial segment $f_1 = f \bmod z^s$ is known, we show how to deduce $f \bmod z^{2s-1}$. Write $f = f_1 + f_2 \bmod z^{2s-1}$, where $z^s$ divides $f_2$. One checks that $f_2$ is a solution of the linearized equation

$$2f_1'f_2' - G_t(z, f_1)f_2 = G(z, f_1) - f_1'^2 \mod z^{2s-2}, \tag{4}$$

with the initial condition $f_2(0) = 0$, where $G_t$ denotes the derivative of $G$ with respect to $t$. The condition $f'(0) \neq 0$ implies that $f_1'$ is a unit in $\mathbf{K}[[z]]$; then, the cost of computing $f_2 \bmod z^{2s-1}$ is in $O(\mathsf{M}(s))$ (remark that we do not take the degree of $G$ into account). Finally, the computation of $f$ at precision $n$ is as follows:

(1) Let $f = \alpha + \beta z \bmod z^2$ and $s = 2$;
(2) **while** $s < n$ **do**
  (a) Compute $f \bmod z^{2s-1}$ from $f \bmod z^s$;
  (b) Let $s = 2s - 1$.

Due to the super-linearity of $\mathsf{M}$, $f \bmod z^n$ can thus be computed using $O(\mathsf{M}(n))$ operations. Again, we have to assume that $2, \ldots, n-1$ are units in $\mathbf{K}$.

2.5. **Other algorithms.** We conclude this section by pointing out other algorithms that are used below:

Power series composition. Over a general field $\mathbf{K}$, there is no known algorithm of quasi-linear complexity for computing $f(g) \bmod z^n$, for $f, g$ in $\mathbf{K}[[z]]$. The best results known today are due to Brent and Kung [7]. Two algorithms are proposed in that article, of respective complexities $O(\mathsf{M}(n)\sqrt{n} + n^{\frac{\omega+1}{2}})$ and $O(\mathsf{M}(n)\sqrt{n \log n})$, where $2 \leq \omega < 3$ is the exponent of matrix multiplication (see, e.g., [22, Chapter 12]). Over fields of positive characteristic $p$, Bernstein's algorithm for composition [3] has complexity $O(\mathsf{M}(n))$, but the $O(\,)$ estimate hides a linear dependence in $p$, making it uninteresting in our setting ($p \gg n$).

Rational function reconstruction. Our last subroutine consists in reconstructing a rational function from its Taylor expansion at the origin. Suppose that $f(z)$ is a rational function in $\mathbf{K}(z)$ with numerator and denominator of degree bounded respectively by $n$ and $n'$, and with denominator non-vanishing at the origin; then, knowing the first $n+n'+1$ terms of the expansion of $f(z)$ at the origin, the rational function $f(z)$ can be reconstructed in $O(\mathsf{M}(n+n')\log(n+n'))$ operations, see [6].

## 3. Computing the Weierstrass $\wp$-function

3.1. **The Weierstrass $\wp$-function.** We now study the complexity of computing the Laurent series expansion of the Weierstrass $\wp$-function at the origin, thus proving Theorem 1. We suppose for a start that the base field $\mathbf{K}$ equals $\mathbb{C}$; the positive characteristic case is discussed below. Let thus $A, B$ be in $\mathbf{K} = \mathbb{C}$. The Weierstrass function $\wp$ associated to $A$ and $B$ is a solution of the non-linear differential equation (2); its Laurent expansion at the origin has the form

$$\wp(z) = \frac{1}{z^2} + \sum_{i\geq 1} c_i z^{2i}. \tag{5}$$

The goal of this section is to study the complexity of computing the first terms $c_1, \ldots, c_n$. We first present a "classical" algorithm, and then show how to apply the fast algorithms for power series of the previous section.

3.2. **Quadratic algorithm.** First, we recall the direct algorithm. Substituting the expansion (5) into Equation (2) and identifying coefficients of $z^{-2}$ and $z^0$ gives

$$c_1 = -\frac{A}{5} \quad \text{and} \quad c_2 = -\frac{B}{7}.$$

Next, differentiating Equation (2) yields the second order equation

$$\wp'' = 6\wp^2 + 2A. \tag{6}$$

This equation implies that for $k \geq 3$, $c_k$ is given by

$$c_k = \frac{3}{(k-2)(2k+3)} \sum_{i=1}^{k-2} c_i c_{k-1-i}. \tag{7}$$

Hence, the coefficients $c_1, \ldots, c_n$ can be computed using $O(n^2)$ operations in $\mathbf{K}$.

If the characteristic $p$ of $\mathbf{K}$ is positive, the definition of $\wp$ as a Laurent series fails, due to divisions by zero. However, assuming $p > 2n+3$, it is still possible to define the coefficients $c_1, \ldots, c_n$ through the previous recurrence relation. Then, again, $c_1, \ldots, c_n$ can be computed using $O(n^2)$ operations in $\mathbf{K}$.

3.3. **Fast algorithm.** One possibility to devise a fast algorithm is to consider the function $z^2\wp(z) \in \mathbf{K}[[z]]$, find a differential equation it satisfies, and deduce the expansion of $\wp(z)$. This leads to tedious computations, so we follow a slightly different path. We first introduce new quantities, that are used again in the next section. Define

$$Q(z) = \frac{1}{\wp(z)} \in z^2 + z^6\mathbf{K}[[z^2]] \quad \text{and} \quad R(z) = \sqrt{Q(z)} \in z + z^5\mathbf{K}[[z^2]].$$

The differential equation satisfied by $R$ is

$$R'^2 = BR^6 + AR^4 + 1, \tag{8}$$

from which we can deduce the first terms of $R$:

$$R(z) = z + \frac{A}{10}z^5 + \frac{B}{14}z^7 + O(z^8) = z\left(1 + \frac{A}{10}z^4 + \frac{B}{14}z^6 + O(z^7)\right).$$

Squaring $R$ yields

$$Q(z) = z^2 + \frac{A}{5}z^6 + \frac{B}{7}z^8 + O(z^9) = z^2\left(1 + \frac{A}{5}z^4 + \frac{B}{7}z^6 + O(z^7)\right).$$

Taking the reciprocal of the right-hand series finally yields

$$\wp(z) = \frac{1}{z^2}\left(1 - \frac{A}{5}z^4 - \frac{B}{7}z^6 + O(z^7)\right) = \frac{1}{z^2} - \frac{A}{5}z^2 - \frac{B}{7}z^4 + O(z^5),$$

as requested. Thus, our fast algorithm to compute the coefficients $c_1, \ldots, c_n$ is as follows:

(1) Compute $R(z) \bmod z^{2n+4}$ using the algorithm of §2.4 with $G = Bt^6 + At^4 + 1$;
(2) Compute $Q(z) = R^2(z) \bmod z^{2n+5}$;
(3) Compute $\wp(z) = 1/Q(z) \bmod z^{2n+1}$.

In the first step, we remark that our assumption $R'(0) \neq 0$ is indeed satisfied, hence $R(z) \bmod z^{2n+4}$ can be computed in $O(\mathsf{M}(n))$ operations, assuming $2, \ldots, 2n+3$ are units in $\mathbf{K}$. Using the algorithm of §2.1, the squaring and reciprocal necessary to recover $\wp(z) \bmod z^{2n+1}$ admit the same complexity bound. This proves Theorem 1.

## 4. Fast computation of isogenies

In this section, we recall the basic properties of isogenies and an algorithm due to Elkies [19] that computes an isogeny of degree $\ell$ in quadratic complexity $O(\ell^2)$. Then, we design two fast variants of Elkies' algorithm, by exploiting the differential equations satisfied by some functions related to the Weierstrass function, proving Theorem 2.

4.1. **Isogenies.** The following properties are classical and can be found for instance in [42]. Let $E$ and $\tilde{E}$ be two elliptic curves defined over $\mathbf{K}$. An isogeny between $E$ and $\tilde{E}$ is a rational map that is also a group morphism; here, our isogenies are all non-zero. Pairs of curves related via a non-zero isogeny are built using modular equations; we refer to [4] for details.

The most elementary example of an isogeny is the "multiplication by $m$" map which sends $P \in E$ to $[m]P \in E$, where, as usual, the group law on $E$ is written additively. If $E$ is given through a Weierstrass model, the group law yields formulas for $[m]P$ in terms of the so-called *division polynomials*:

$$[m](x, y) = \left(\frac{\phi_m(x, y)}{y^2 f_m(x)^2}, \frac{\omega_m(x, y)}{y^4 f_m(x)^3}\right)$$

where the polynomial $f_m(x)$ has degree $\Theta(m^2)$.

Given an isogeny $I : E \to \tilde{E}$, there exist a unique isogeny (the *dual isogeny*) $\hat{I} : \tilde{E} \to E$ and a unique integer $\ell$ such that $\hat{I} \circ I = [\ell]$; $\ell$ is called the *degree* of $I$. For instance, the degree of the isogeny $[m]$ is $m^2$ and this is reflected by the degree of the division polynomials.

For simplicity, we suppose in the rest of the article that $\ell$ is an odd integer. The kernel $F$ of an isogeny $I$ of degree $\ell$ is a subgroup of order $\ell$ of $E(\overline{\mathbf{K}})$. It consists of the point at infinity on $E$, which is the zero for the group law, and of $d = \frac{\ell-1}{2}$ pairs of non-zero points with same abscissas $x_Q$, $Q \in F^*$. We then let

$$g(x) = x^d - p_1 x^{d-1} + \cdots$$

be the unique monic polynomial of degree $d$ vanishing at those abscissas $x_Q$. Remark for further use that $p_1$ is the first power sum of $g$, so that $2p_1$ is the sum of the abscissas of the nonzero points in the kernel of $I$, matching the notation used in Theorem 2.

A point $P$ of coordinates $(x_P, y_P)$ is sent by the isogeny $I$ to a point of coordinates $x_{I(P)} = x_P + \sum_{Q\in F^*}(x_{P+Q} - x_Q)$ and $y_{I(P)} = y_P + \sum_{Q\in F^*}(y_{P+Q} - y_Q)$. Using the rational form of the group law of $E$, this results in the following explicit form for $I$, due to Vélu [45]

$$I(x,y) = \left(x + \textstyle\sum_{Q\in F^*}\left(\frac{3x_Q^2+A}{x-x_Q} + 2\frac{x_Q^3+Ax_Q+B}{(x-x_Q)^2}\right), y - y\ \sum_{Q\in F^*}\left(\frac{3x_Q^2+A}{(x-x_Q)^2} + 4\frac{x_Q^3+Ax_Q+B}{(x-x_Q)^3}\right)\right).$$

As a corollary of these formulas, Dewaghe [17] has shown that $I$ can be written as

$$I(x,y) = \left(\frac{N(x)}{D(x)}, y\left(\frac{N(x)}{D(x)}\right)'\right), \tag{9}$$

where $D(x) = g(x)^2$, and $N(x)$ is related to $g(x)$ through the formula

$$\frac{N(x)}{D(x)} = \ell x - 2p_1 - 2(3x^2+A)\frac{g'(x)}{g(x)} - 4(x^3+Ax+B)\left(\frac{g'(x)}{g(x)}\right)'. \tag{10}$$

From now on, we are given two isogeneous curves $E$ and $\tilde{E}$ through their Weierstrass equations. From this input, and possibly that of $p_1$, we want to determine the isogeny $I$. We first describe an algorithm due to Elkies [19] (called Elkies1998 in the sequel), whose complexity is quadratic in the degree $\ell$. Then, we give two fast variants of algorithm Elkies1998, called fastElkies and fastElkies′, of respective complexities $O(\mathsf{M}(\ell))$ and $O(\mathsf{M}(\ell)\log\ell)$.

4.2. **Elkies' quadratic algorithm.** The idea of algorithm Elkies1998 in [19] is to compute the expansion of $N/D$ at infinity, and recover the power sums of the roots of $g$ from it. Our starting remark to present these ideas is that the Weierstrass functions $\wp$ and $\tilde{\wp}$ are related through

$$\frac{N(x)}{D(x)} = \tilde{\wp} \circ \wp^{-1}(x), \tag{11}$$

where $\wp^{-1}$ is the functional inverse of $\wp$. Applying the chain rule, we see that $N/D$ satisfies the non-linear differential equation

$$(x^3+Ax+B)\left(\frac{N(x)}{D(x)}\right)'^2 = \left(\frac{N(x)}{D(x)}\right)^3 + \tilde{A}\left(\frac{N(x)}{D(x)}\right) + \tilde{B}. \tag{12}$$

A second differentiation leads to the following second-order equation:

$$(3x^2+A)\left(\frac{N(x)}{D(x)}\right)' + 2(x^3+Ax+B)\left(\frac{N(x)}{D(x)}\right)'' = 3\left(\frac{N(x)}{D(x)}\right)^2 + \tilde{A}. \tag{13}$$

Writing

$$\frac{N(x)}{D(x)} = x + \sum_{i\geq 1} \frac{h_i}{x^i}$$

and identifying coefficients of $x^{-i}$, $i \geq 1$, from both sides of Equation (13), one deduces the recurrence relation

$$h_k = \frac{3}{(k-2)(2k+3)} \sum_{i=1}^{k-2} h_i h_{k-1-i} - \frac{2k-3}{2k+3} A h_{k-2} - \frac{2(k-3)}{2k+3} B h_{k-3}, \quad \text{for all } k \geq 3, \tag{14}$$

with initial conditions

$$h_1 = \frac{A - \tilde{A}}{5} \quad \text{and} \quad h_2 = \frac{B - \tilde{B}}{7}.$$

The recurrence (14) is the basis of algorithm Elkies1998; using it, one can compute $h_3, \ldots, h_{d-1}$ using $O(\ell^2)$ operations in $\mathbf{K}$.

Next, expanding at infinity the right-hand side of Vélu's formulas, Elkies obtains the following recurrence relation connecting the power sums $p_1, p_2, \ldots$ of $g$ to the coefficients $h_i$ (this relation can also be deduced directly from Dewaghe's Equation (10)):

$$h_i = (4i+2)p_{i+1} + (4i-2)Ap_{i-1} + (4i-4)Bp_{i-2}, \quad \text{for all } 1 \leq i \leq d-1. \tag{15}$$

Elkies' algorithm Elkies1998 assumes that $p_1$ is given. Since $h_1, \ldots, h_{d-1}$ are known, $p_2, \ldots, p_d$ can be deduced from the previous recurrence using $O(\ell)$ operations. The polynomial $g$ is then recovered, either by a quadratic algorithm or the faster algorithm of §2.2, and $N$ and $D$ are deduced using formula (10), in $O(\mathsf{M}(\ell))$ operations.

The complexity of this algorithm is thus in $O(\ell^2)$, the bottleneck being the computation of the coefficients $h_1, \ldots, h_{d-1}$; this algorithm requires that $2, \ldots, 2d+1 = \ell$ be units in $\mathbf{K}$. Observe also the parallel with the computations presented in the previous section, where differentiating Weierstrass' equation yields the recurrence (7), which appears as a particular case of the recurrence (14) (the former is obtained by taking $A = B = 0$ in the latter).

4.3. **Fast algorithms.** We improve on the computation of the coefficients $h_i$ in algorithm Elkies1998, the remaining part being unchanged.

Unfortunately, we cannot directly apply the algorithm of §2.4 to compute the expansion of $N/D$ at infinity using the differential equation (12), since the equation obtained by the change of variables $x \mapsto 1/x$ is singular at the origin. To avoid this technical complication, we rather consider the power series

$$S(x) = x + \frac{\tilde{A} - A}{10} x^5 + \frac{\tilde{B} - B}{14} x^7 + O(x^9) \in x + x^3 \mathbf{K}[[x^2]]$$

such that $\tilde{R} = S \circ R$, with the notation $R = 1/\sqrt{\wp}$ and $\tilde{R} = 1/\sqrt{\tilde{\wp}}$ introduced in §3.3. Knowing the expansion of $S$ at the origin, the expansion of $N/D$ at infinity is easily recovered using

$$\frac{N(x)}{D(x)} = \frac{1}{S\left(\frac{1}{\sqrt{x}}\right)^2}.$$

Applying the chain rule gives the following first order differential equation satisfied by $S(x)$:

$$(Bx^6 + Ax^4 + 1)S'(x)^2 = 1 + \tilde{A}S(x)^4 + \tilde{B}S(x)^6.$$

Using this differential equation, we propose two algorithms to compute $N/D$, depending on whether the coefficient $p_1$ is known or not. In the algorithms, we write

$$S = xT(x^2) \quad \text{and} \quad U(x) = \frac{1}{T(x)^2} \in 1 + x^2\mathbf{K}[[x]] \quad \text{so that} \quad \frac{N(x)}{D(x)} = x\,U\left(\frac{1}{x}\right).$$

The first algorithm, called fastElkies, assumes that $p_1$ is known and goes as follows.

(1) Compute $C(x) = (Bx^6 + Ax^4 + 1)^{-1} \bmod x^{2d+2} \in \mathbf{K}[[x]]$;
(2) Compute $S(x) \bmod x^{2d+3}$ using the algorithm of §2.4 with $G(x,t) = C(x)(1 + \tilde{A}t^4 + \tilde{B}t^6)$, and deduce $T(x) \bmod x^{d+1}$;
(3) Compute $U(x) = 1/T^2(x) \bmod x^{d+1}$ using the algorithm in §2.1;
(4) Compute the coefficients $h_1, \ldots, h_{d-1}$ of $N/D$, using $N(x)/D(x) = xU(1/x)$;
(5) Compute the coefficients $p_2, \ldots, p_d$ of $g'/g$, using the linear recurrence (15);
(6) Recover $g$ from its first $d$ power sums $p_1, \ldots, p_d$, as described in §2.2;
(7) Compute $D = g^2$ and deduce $N$ using Equation (10).

Steps (1) and (5) have cost $O(\ell)$. Steps (2), (3), (6) and (7) can be performed in $O(\mathsf{M}(\ell))$ operations, and Step (4) requires no operation. This proves the first part of Theorem 2.

For our second algorithm (that we call fastElkies′), we do not assume prior knowledge of $p_1$. Its steps (1')–(3') are just a slight variation of Steps (1)–(3), of the same complexity $O(\mathsf{M}(\ell))$ (up to constant factors).

(1') Compute $C(x) = (Bx^6 + Ax^4 + 1)^{-1} \bmod x^{8d+4} \in \mathbf{K}[[x]]$;
(2') Compute $S(x) \bmod x^{8d+5}$ using the algorithm of §2.4 with $G(x,t) = C(x)(1 + \tilde{A}t^4 + \tilde{B}t^6)$, and deduce $T(x) \bmod x^{4d+2}$;
(3') Compute $U(x) = 1/T^2(x) \bmod x^{4d+2}$, using the algorithm in §2.1;
(4') Reconstruct the rational function $U(x)$;
(5') Return $N(x)/D(x) = xU(1/x)$.

Using fast rational reconstruction, Step (4') can be performed in $O(\mathsf{M}(\ell)\log\ell)$ operations in $\mathbf{K}$. Finally, it is easy to check that our algorithm fastElkies requires that $2, \ldots, 2d+1 = \ell$ be units in $\mathbf{K}$, while algorithm fastElkies′ requires that $2, \ldots, 8d+4 = 4\ell$ be units in $\mathbf{K}$. This completes the proof of Theorem 2.

4.4. **Worked example.** Let

$$E : y^2 = x^3 + x + 1 \quad \text{and} \quad \tilde{E} : \tilde{y}^2 = \tilde{x}^3 + 75\tilde{x} + 16$$

be defined over $\mathbb{F}_{101}$, with $\ell = 11$ and $p_1 = 25$. First, from the differential equation

$$(x^6 + x^4 + 1)S'(x)^2 = 1 + 75S(x)^4 + 16S(x)^6, \quad S(0) = 0, \quad S'(0) = 1$$

we infer the equalities

$$\begin{aligned}
& C &=& \ 1 + 100\,x^4 + 100\,x^6 + x^8 + 2\,x^{10} + O(x^{12}), \\
& S &=& \ x + 68\,x^5 + 66\,x^7 + 60\,x^9 + 84\,x^{11} + O(x^{13}), \\
\text{so that} \quad & T &=& \ 1 + 68\,x^2 + 66\,x^3 + 60\,x^4 + 84\,x^5 + O(x^6), \\
\text{and} \quad & T^2 &=& \ 1 + 35x^{\,2} + 31x^{\,3} + 98x^{\,4} + 54x^{\,5} + O(x^{\,6}), \\
\text{whence} \quad & U &=& \ 1 + 66x^{\,2} + 70x^{\,3} + 16x^{\,4} + 96x^{\,5} + O(x^{\,6}).
\end{aligned}$$

We deduce

$$\frac{N(x)}{D(x)} \;=\; x + \frac{66}{x} + \frac{70}{x^2} + \frac{16}{x^3} + \frac{96}{x^4} + O\left(\frac{1}{x^5}\right).$$

At this stage, we know $h_1 = 66, h_2 = 70, h_3 = 16, h_4 = 96$. Equation (15) then writes

$$p_{i+1} = \frac{h_i - (4i-2)p_{i-1} - (4i-4)p_{i-2}}{4i+2}, \quad \text{for all } 1 \le i \le 4$$

and gives $p_2 = 43, p_3 = 91, p_4 = 86, p_5 = 63$. The main equation in §2.2 writes

$$\begin{aligned} x^5 g_{11}\left(\frac{1}{x}\right) &= \exp_6\!\left(-\left(25\,x + \frac{43}{2}x^2 + \frac{91}{3}\,x^3 + \frac{86}{4}\,x^4 + \frac{63}{5}\,x^5\right)\right) \\ &= \exp_6\!\left(76\,x + 29\,x^2 + 37\,x^2 + 29\,x^4 + 48\,x^5\right), \end{aligned}$$

yielding $g_{11}(x) = x^5 + 76x^4 + 89x^3 + 24x^2 + 97x + 5$. For the sake of completeness, we have:

$$N(x) = x^{11} + 51x^{10} + 61x^9 + 44x^8 + 71x^7 + 39x^6 + 81x^5 + 43x^4 + 15x^3 + 5x^2 + 24x + 15.$$

Had we computed the solution $S(x)$ at precision $O(x^{45})$, the expansion at infinity of $N/D$ would have been known at precision $O(1/x^{21})$, and this would have sufficed to recover both $N(x)$ and $D(x)$ by rational function reconstruction, without the prior knowledge of $p_1$.

## 5. A survey of previous algorithms for isogenies

In this section, we recall and give complexity results for other known algorithms for computing isogenies. In what follows, we write the Weierstrass functions $\wp$ and $\tilde{\wp}$ of our two curves $E$ and $\tilde{E}$ as

$$\wp(z) = \frac{1}{z^2} + \sum_{i \ge 1} c_i z^{2i} \quad \text{and} \quad \tilde{\wp}(z) = \frac{1}{z^2} + \sum_{i \ge 1} \tilde{c}_i z^{2i}.$$

All algorithms below require the knowledge of the expansion of these functions at least to precision $\ell$, so they only work under a hypothesis of the type $p \gg \ell$ or $p = 0$.

We can freely assume that these expansions are known. Indeed, by Theorem 1, given $A, B$ and $\tilde{A}, \tilde{B}$, we can precompute the coefficients $c_i$ and $\tilde{c}_i$ up to (typically) $i = \ell - 1$ using $O(\mathsf{M}(\ell))$ operations in $\mathbf{K}$, provided that the characteristic $p$ of $\mathbf{K}$ is either 0 or $> \ell$. This turns out to be negligible compared to the other costs involved in the following algorithms.

5.1. **First algorithms.** A brute force approach to compute $N/D$ is to use Equation (11) and the method of undetermined coefficients. This reduces to computing of $\wp(z)^i \bmod z^{4\ell-2}$ for $1 \le i \le \ell$ and solving a linear system with $2\ell - 1$ unknowns. This direct method requires that $2, \ldots, 4\ell$ be units in $\mathbf{K}$ and its complexity is $O(\ell^\omega)$ operations in $\mathbf{K}$, where $2 \le \omega < 3$ is the exponent of matrix multiplication.

Another possible idea would be to consider the rational functions $N/D$ and $\hat{N}/\hat{D}$ respectively associated to $I$ and its dual $\hat{I}$, noticing that by definition,

$$\frac{N}{D} \circ \frac{\hat{N}}{\hat{D}} = \frac{\phi_\ell}{f_\ell^2}.$$

However, algorithms for directly decomposing $\phi_\ell / f_\ell^2$ [46, 23, 1] lead to too expensive a solution in our case, since they require factoring the degree $\Theta(\ell^2)$ polynomial $f_\ell$. Indeed, over finite fields, even using the best (sub-quadratic) algorithms for polynomial factorization [25],

of exponent 1.815, this would yield an algorithm for computing isogenies of degree $\ell$ in complexity more than cubic with respect to $\ell$, which is unacceptable.

5.2. **Stark's method.** To the best of our knowledge, the first subcubic method for finding $N$ and $D$ is due to Stark [43] and amounts to expanding $\tilde{\wp}$ as a continued fraction in $\wp$, using Equation (11). The fraction $N/D$ is approximated by $p_n/q_n$ and the algorithm stops when the degree of $q_n$ is $\ell - 1$, yielding $D = g^2$. Since $\wp$ and $\tilde{\wp}$ are in $1/z^2 + \mathbf{K}[[z^2]]$, it is sufficient to work with series in $Z = z^2$.

(1) $T := \tilde{\wp}(Z) + O(Z^\ell)$;
(2) $n := 1$;
(3) $q_0 := 1$;
(4) $q_1 := 0$;
(5) **while** $\deg(q_n) < \ell - 1$ **do**
{at this point, $T(Z) = t_{-r}Z^{-r} + \cdots + t_0 + t_1 Z + \cdots + O(Z^{(\ell - \deg q_n - r) - 1})$}
(a) $n := n + 1$;
(b) $a_n := 0$;
(c) **while** $r \geq 1$ **do**
$a_n := a_n + t_{-r} z^r$;
$T := T - t_{-r}\wp^r = t_{-s}Z^{-s} + \cdots$ ;
$r := s$
(d) $q_n := a_n q_{n-1} + q_{n-2}$;
(e) $T := 1/T$;
(6) Return $D := q_n$.

This algorithm (that we call Stark1972) requires $O(\ell)$ passes through Step (5); this bound is reached in general, with $r = 1$ at each step. The step that dominates the complexity is the computation of reciprocals in Step (5.e), with precision $2\ell - 1 - 2\deg q_n - 2r$. The sum of these operations thus costs $O(\ell \mathsf{M}(\ell))$. The multiplications in Step (5.d) can be done in time $O(\ell \mathsf{M}(\ell))$ as well (these multiplications could be done faster if needed). Since the largest degree of the polynomials $a_n$ is bounded by $\ell - 1$, computing all powers of $\wp$ at Step (5.c) also fits within the $O(\ell \mathsf{M}(\ell))$ bound. Finally, knowing $D = g^2$, one can recover the power sum $p_1$ of $g$, and the logarithmic derivative $g'/g = D'/2D$. Then the numerator $N$ can be recovered in cost $O(\mathsf{M}(\ell))$ using Equation (10).

To summarize, the total cost of algorithm Stark1972 is in $O(\ell \mathsf{M}(\ell))$. Remark that compared to the methods presented below, algorithm Stark1972 does not require the knowledge of the first power sum $p_1$ of $g$. Remark also that, even though $r$ will be 1 in general, the computation of the powers $\wp^r$ in Step (5.c) could be amortized in the context of the SEA algorithm. Besides, if we need $g$, as in the course of SEA, we can compute it in $O(\mathsf{M}(\ell))$ operations by computing $\exp((\log D)/2)$.

5.3. **Elkies' 1992 method.** We reproduce the method given in [18], that we call Elkies1992 (see also e.g., [9, 32]). Differentiating twice Equation (6) yields

$$\frac{d^4\wp(z)}{dz^4} = 120\wp^3 + 72A\wp + 48B.$$

More generally, we obtain equalities of the form

$$\frac{d^{2k}\wp(z)}{dz^{2k}} = \mu_{k,k+1}\wp^{k+1} + \cdots + \mu_{k,0},$$

for some constants $\mu_{k,j}$ that satisfy the recurrence relation

$$\mu_{k+1,j} = (2j-2)(2j-1)\mu_{k,j-1} + (2j+1)(2j+2)A\mu_{k,j+1} + (2j+2)(2j+4)B\mu_{k,j+2},$$

with $\mu_{k,k+1} = (2k+1)!$. Using this recurrence relation, the coefficients $\mu_{k,j}$, for $k \leq d-1$ and $j \leq k+1$, can be computed in $O(\ell^2)$ operations in $\mathbf{K}$.

Elkies then showed how to use these coefficients to recover the power sums $p_2, \ldots, p_d$ of $g$, through the following equalities, holding for $k \geq 1$:

$$(2k)!(\tilde{c}_k - c_k) = 2(\mu_{k,0}p_0 + \cdots + \mu_{k,k+1}p_{k+1}).$$

Using these equalities, assuming that $p_1$ and the coefficients $c_k$, $\tilde{c}_k$ and $\mu_{k,j}$ are known, we can recover $p_2, \ldots, p_d$ by solving a triangular system, in complexity $O(\ell^2)$. We can then recover $g$ using either a quadratic algorithm, or the faster algorithm of §2.2.

There remains here the question whether the triangular system giving $p_2, \ldots, p_d$ can be solved in quasi-linear time. To do so, one should exploit the structure of the triangular system, and avoid computing the $\Theta(\ell^2)$ constants $\mu_{k,j}$ explicitly.

5.4. **Atkin's method.** In [2], Atkin gave the following formula enabling the computation of $g$ (see also [33, Formula 6.13] and [38]):

$$g(\wp(z)) = z^{1-\ell}\exp(F(z)), \tag{16}$$

where

$$F(z) = -p_1 z^2 + \left(\sum_{k=1}^{\infty}(\ell c_k - \tilde{c}_k)\frac{z^{2k+2}}{(2k+1)(2k+2)}\right).$$

Since $\ell$ and $c_k, \tilde{c}_k$, for $k \leq d-1$, are all assumed to be known, one can deduce the series $F \bmod z^{2d+2}$ provided that $p_1$ is given. A direct method to determine $g$ is then to compute the exponential of $F$, and to recover the coefficients of $g$ one at a time, as shown in the following algorithm, called Atkin1992. As before, we use series in $Z = z^2$.

(1) Compute the series $P_i(Z) = \wp(Z)^i$ at order $d$, for $1 \leq i \leq d$;
(2) Compute $G(Z) = \exp_{d+1}(F(Z))$;
(3) $T := G$;
(4) $g := 0$;
(5) **for** $i := d$ **downto** $0$ **do**
  {at this point, $T = tZ^{-i} + \cdots$ }
  (a) $g := g + tz^i$;
  (b) $T := T - tP_i$.

Step (1) uses $O(\ell\mathsf{M}(\ell))$ operations; the cost of Step (2) is negligible, using either classical or fast exponentiation. Then, each pass through Step (5) costs $O(\ell)$ more operations, for a total of $O(\ell^2)$. Thus, the total cost of this algorithm is in $O(\ell\mathsf{M}(\ell))$. If this algorithm is used in the context of SEA, Step (1) can be amortized, since it depends on $E$ only. Therefore, all the powers of $\wp$ should be computed for the maximal value of $\ell$ to be used, and stored. Hence, the cost of this algorithm would be dominated by that of Step (5), yielding a method of complexity $O(\ell^2)$.

A better algorithm for computing $g$, avoiding the computation of all the $d$ powers of $\wp$, is based on the remark that Equation (16) rewrites

$$g\left(\frac{1}{x}\right) = \mathcal{I}^{1-\ell}\left((\exp \circ F) \circ \mathcal{I}\right), \tag{17}$$

with $\mathcal{I}(x) = \wp^{-1}(1/x)$, where $\wp^{-1}$ is the functional inverse of $\wp$. The expansion of $\mathcal{I}$ at order $\Theta(\ell)$ can be computed in $O(\ell)$ operations using the differential equation

$$\mathcal{I}'^2 = \frac{1}{4x(1 + Ax^2 + Bx^3)} \quad \text{or} \quad \mathcal{I}' = \frac{1}{2\sqrt{x}} \frac{1}{\sqrt{1 + Ax^2 + Bx^3}}. \tag{18}$$

Now, $\mathcal{J}(x) = \frac{1}{\sqrt{1+Ax^2+Bx^3}}$ satisfies the linear differential equation

$$\frac{\mathcal{J}'}{\mathcal{J}} = -\frac{1}{2} \frac{2Ax + 3Bx^2}{1 + Ax^2 + Bx^3}; \tag{19}$$

extracting coefficients in this equation shows that $\mathcal{I}(x) = x^{\frac{1}{2}} \sum_{i\geq 0} \frac{a_i}{2i+1} x^i$, with

$$a_0 = 1, \quad a_1 = 0, \quad a_2 = -\frac{A}{2}, \quad a_{i+1} = \frac{Ba_{i-2} - 2Bia_{i-2} - 2Aia_{i-1}}{2i+2} \quad \text{for} \quad i \geq 2. \tag{20}$$

This yields the following algorithm, called AtkinModComp:

(1) Compute $G(Z) = \exp_{d+1}(F(Z))$;
(2) Compute $\mathcal{I}(x)$ using Equation (20);
(3) Compute $G(\mathcal{I})$ by modular composition (which is possible since $G$ is in $\mathbf{K}[[Z]] = \mathbf{K}[[x^2]]$);
(4) Deduce $g$ using Equation (17).

The cost of the algorithm is dominated by the composition of the series $G = \exp \circ F$ and $\mathcal{I}$. From §2.5, this can be done in $O(\mathsf{M}(\ell)\sqrt{\ell} + \ell^{\frac{\omega+1}{2}})$ or $O(\mathsf{M}(\ell)\sqrt{\ell \log \ell})$ operations in $\mathbf{K}$.

To do even better, it is fruitful to reconsider the series $G(\mathcal{I}) = (\exp \circ F) \circ \mathcal{I}$ used above, but rewriting it as $\exp \circ (F \circ \mathcal{I})$ instead; this change of point of view reveals close connections with our fastElkies algorithm. More precisely, Atkin's Equation (16) can be rewritten as

$$g(\wp(x)) = \exp\left(-p_1 x^2 + \iint \ell\wp(x) - \tilde{\wp}(x)\right).$$

We can then obtain $g(1/x)$ as the following exponential:

$$g\left(\frac{1}{x}\right) = \exp\left(-p_1\mathcal{I}^2 + \int \mathcal{I}' \int \mathcal{I}' \left(\frac{\ell}{x} - (\tilde{\wp} \circ \mathcal{I})(x)\right)\right) \tag{21}$$

$$= \exp\left(-p_1\mathcal{I}^2 + \int \mathcal{I}' \int \mathcal{I}' \left(\frac{\ell}{x} - \frac{N(1/x)}{D(1/x)}\right)\right) \tag{22}$$

$$= \exp\left(-p_1\mathcal{I}^2 + \int \mathcal{I}' \int \mathcal{I}' \left(\frac{\ell}{x} - \frac{1}{S(\sqrt{x})^2}\right)\right). \tag{23}$$

Then, working out the details, the sequence of operations necessary to evaluate this exponential turns out to be the same as the one used in our algorithm fastElkies of §4.3. This

does not come as a surprise: the relation (15) used in our algorithm follows from Dewaghe's formula (10), which can be rewritten as

$$\frac{N(x)}{D(x)} = \ell x - 2p_1 - 4\sqrt{x^3+Ax+B}\left(\sqrt{x^3+Ax+B}\,\frac{g'(x)}{g(x)}\right)'.$$

Then, Equation (22) is nothing but an integral reformulation of this last equation, taking into account the fact that $\mathcal{I}$ satisfies the differential equation (18).

5.5. **Summary.** In Table 1 we gather the various algorithms discussed in this article, and compare these algorithms from three points of view: their complexity (expressed in number of operations in the base field $\mathbf{K}$), their need for $p_1$ as input and the hypotheses on the characteristic of $\mathbf{K}$ ensuring their validity.

| algorithm | complexity | need of $p_1$ | char restriction |
|---|---|---|---|
| linear algebra | $O(\ell^\omega)$ | no | $p > 4\ell$ |
| Stark1972 | $O(\ell\mathsf{M}(\ell))$ | no | $p > \ell$ |
| Atkin1992 | $O(\ell\mathsf{M}(\ell))$ | yes | $p > \ell$ |
| AtkinModComp | $O(\mathsf{M}(\ell)\sqrt{\ell} + \ell^{\frac{\omega+1}{2}})$ or $O(\mathsf{M}(\ell)\sqrt{\ell\log\ell})$ | yes | $p > \ell$ |
| Elkies1992 | $O(\ell^2)$ | yes | $p > \ell$ |
| Elkies1998 | $O(\ell^2)$ | yes | $p > \ell$ |
| fastElkies | $O(\mathsf{M}(\ell))$ | yes | $p > \ell$ |
| fastElkies′ | $O(\mathsf{M}(\ell)\log\ell)$ | no | $p > 4\ell$ |

TABLE 1. Comparison of the algorithms

## 6. Implementation and benchmarks

We implemented our algorithms using the NTL C++ library [39, 40] and ran the program on an AMD 64 Processor 3400+ (2.4GHz).

We begin with timings for computing the expansion of $\wp$, obtained over the finite field $\mathbb{F}_{10^{2004}+4683}$; they are given in Figure 1.

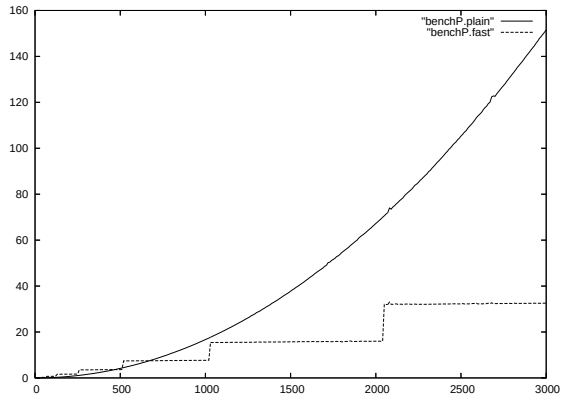


FIGURE 1. Timings for computing $\wp$ on $E : y^2 = x^3 + 4589x + 91128$ over $\mathbb{F}_{10^{2004}+4683}$

The shape of both curves in Figure 1 indicates that the theoretical complexities – quadratic vs. nearly linear – are well respected in our implementation (note that the abrupt jumps at powers of 2 reflect the performance of NTL's FFT implementation of polynomial arithmetic). Moreover, the threshold beyond which our algorithm becomes useful over the quadratic one is reasonably small, making it interesting in practice very early.

We now turn our attention to the pure isogeny part. The first series of timings concerns the computation of isogenies over a small field, $\mathbf{K} = \mathbb{F}_{10^{19}+51}$, for the curve $E : y^2 = x^3 + 4589x + 91128$.

We compare in Figure 2 the performances of the algorithms Elkies1992 from §5.3 and Elkies1998 from §4.2 for isogenies of moderate degree $\ell \leq 400$. Figure 3 compares the timings obtained with the algorithm Elkies1998 and our fast version fastElkies from §4.3, for isogenies of degree up to 6000.

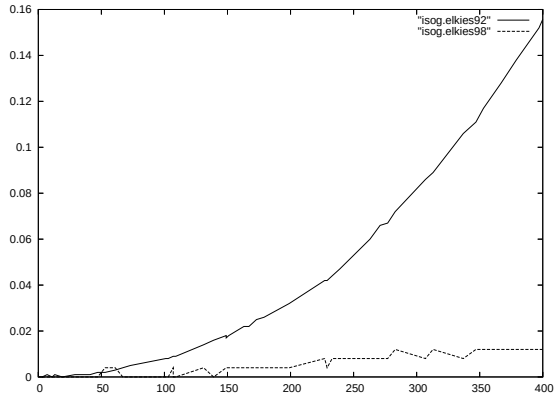


FIGURE 2. Elkies1992 vs. Elkies1998.

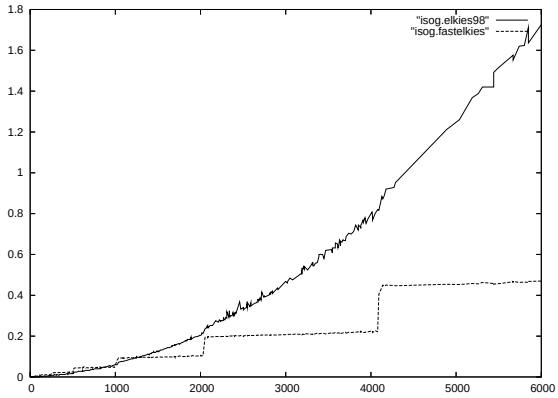


FIGURE 3. Elkies1998 vs. fastElkies.

Next, we compare in Figure 4 the timings obtained by the $O(\mathsf{M}(\ell))$ algorithm fastElkies, that requires the knowledge of $p_1$, to those obtained by its $O(\mathsf{M}(\ell)\log\ell)$ counterpart fastElkies', that does not require this information.

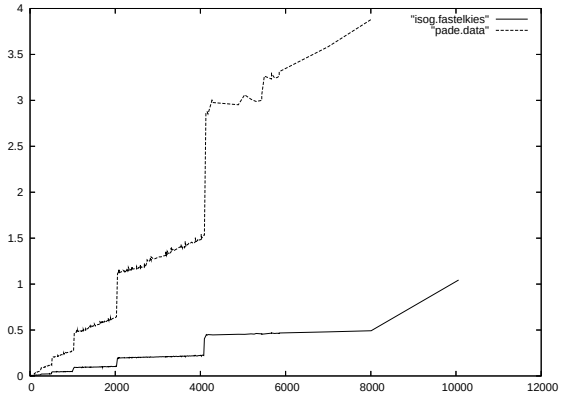


FIGURE 4. FastElkies vs. FastElkies'

In all figures, the degrees $\ell$ of the isogenies are represented on the horizontal axis and the timings are given (in seconds) on the vertical axis. Again, the shape of both curves in Figure 3 shows that the theoretical complexities are well respected in our implementation. The curves in Figure 4 show that the theoretical ratio of $\log \ell$ between algorithms fastElkies and fastElkies′ has a consequent practical impact.

Next, in Tables 2 to 8, we give detailed timings (in seconds) on computing $\ell$-isogenies for the curve

$$E : y^2 = x^3 + Ax + B$$

where

$$A = \lfloor 10^{1990}\pi \rfloor = 31415926 \ldots 58133904,$$
$$B = \lfloor 10^{1990}e \rfloor = 27182818 \ldots 94787610,$$

for a few values of $\ell$, over the larger finite field $\mathbb{F}_{10^{2004}+4683}$, and using various methods: algorithms Elkies1992, Elkies1998 and our fast variant fastElkies, Stark's algorithm Stark1972 and the two versions Atkin1992 and AtkinModComp of Atkin's algorithm.

Tables 2 and 3 give timings for basic subroutines shared by some or all of the algorithms discussed. Table 2 gives the timings necessary to compute the expansions of $\wp$ and $\tilde{\wp}$, using either the classical algorithm or our faster variant: this is used in all algorithms, except our fastElkies algorithm. Table 3 gives timings for recovering $g$ from its power sums, first using the classical quadratic algorithm, and then using fast exponentiation as described in §2.2. This is used in algorithms Elkies1992, and Elkies1998 and its variants.

| | Computing $\wp$ and $\tilde{\wp}$ | | |
|---|---|---|---|
| $\ell$ | order | quadratic | fast |
| 1013 | 511 | 8.6 | 7.0 |
| 2039 | 1024 | 34.6 | 29.9 |
| 3019 | 1514 | 75.7 | 30.3 |
| 4001 | 2005 | 132.7 | 31 |
| 5021 | 2515 | 209.3 | 64.4 |

TABLE 2. Computing $\wp$ and $\tilde{\wp}$

| | Recovering $g$ | |
|---|---|---|
| $\ell$ | quadratic | fast |
| 1013 | 4.2 | 1.1 |
| 2039 | 17.4 | 2.5 |
| 3019 | 38.2 | 5.1 |
| 4001 | 66.9 | 5.5 |
| 5021 | 106.2 | 11.2 |

TABLE 3. Recovering $g$ from its power sums

Tables 4 and 5 give the timings for algorithms Elkies1992 on the one hand and Elkies1998 and our variation fastElkies on the other hand. In Table 4, the columns $\mu$ and $p_i$ give the time used to compute the coefficients $\mu_{i,j}$ and the power sums $p_i$. In Table 5, the column $h_i$ indicates the time used to compute the coefficients $h_i$ of the rational function $N/D$, first using the original quadratic algorithm Elkies1998, then using our faster variant fastElkies. The next column gives the time used to compute the power sums $p_i$ from the $h_i$ using the recurrence (15).

Tables 6 and 7 give timings for our implementation of Atkin's original algorithm Atkin1992, as well as the faster version AtkinModComp using modular composition mentioned in §5.4. In Table 6, the column "exponential" compares the computation of $\exp(F)$ using the naive exponentiation algorithm to the computation using the faster algorithm presented in §2.2; the column $\wp^k$ gives the time for computing all the series $\wp(z)^k$ and the column $g$ that for recovering the coefficients of $g$. Table 7 gives timings obtained using the two modular composition algorithms mentioned in §2.5, called here ModComp1 and ModComp2; the previous

| $\ell$ | Elkies1992 | | | |
|---|---|---|---|---|
| | $\wp, \tilde{\wp}$ | $\mu$ | $p_i$ | $g$ |
| 1013 | | 10.4 | 4.4 | |
| 2039 | See | 49.1 | 17.9 | See |
| 3019 | Table 2 | 130.6 | 38.9 | Table 3 |
| 4001 | | 263 | 68.4 | |
| 5021 | | 496.5 | 106.6 | |

TABLE 4. Algorithm Elkies1992

| $\ell$ | Elkies1998 and fastElkies | | | |
|---|---|---|---|---|
| | $h_i$ | | $p_i$ | $g$ |
| | quadratic | fast | | |
| 1013 | 4.4 | 4.5 | 0.05 | |
| 2039 | 17.3 | 9.6 | 0.1 | See |
| 3019 | 38.0 | 19.5 | 0.16 | Table 3 |
| 4001 | 67.2 | 20.0 | 0.21 | |
| 5021 | 105.0 | 40.7 | 0.27 | |

TABLE 5. Algorithms Elkies1998 and fastElkies

columns give the time for computing $\exp(F)$ and that for computing the requested power of $\mathcal{I}$; the last column gives the time to perform the final multiplication.

Asymptotically, algorithm ModComp2 is faster than algorithm ModComp1, so that the timings in Table 7 might come as a surprise. The explanation is that, for the problem sizes we are interested in, the predominant step of algorithm ModComp1 is the one based on polynomial operations, while the step based on linear algebra operations takes only about 10% of the whole computing time. Thus, the practical complexity of this algorithm in the considered range ($1000 < \ell < 6000$) is proportional to $\mathsf{M}(\ell)\sqrt{\ell}$, while that of algorithm ModComp2 is proportional to $\mathsf{M}(\ell)\sqrt{\ell \log \ell}$. Moreover, the proportionality constant is smaller in the built-in NTL function performing ModComp1 than in our implementation of ModComp2.

| $\ell$ | Algorithm Aktin1992 | | | | |
|---|---|---|---|---|---|
| | $\wp, \tilde{\wp}$ | exponential | | $\wp^k$ | $g$ |
| | | naive | fast | | |
| 1013 | | 88.4 | 1.2 | 72.3 | 4.4 |
| 2039 | See | 370.1 | 4.9 | 304.9 | 17.7 |
| 3019 | Table 2 | 955.9 | 5.1 | 755.8 | 38.9 |
| 4001 | | 1503 | 5.2 | 1218.9 | 67.6 |
| 5021 | | 3180 | 10.8 | 2506.4 | 108.7 |

TABLE 6. Atkin's original algorithm, variations for $\exp(F)$

| $\ell$ | Algorithm AtkinModComp | | | | | |
|---|---|---|---|---|---|---|
| | $\wp, \tilde{\wp}$ | $\exp(F)$ | $\mathcal{I}^{1-\ell}$ | modular composition | | $g$ |
| | | | | ModComp1 | ModComp2 | |
| 1013 | | 1.2 | 2.7 | 14.3 | 35.6 | 0.2 |
| 2039 | See | 2.5 | 6.6 | 45.8 | 111.9 | 0.4 |
| 3019 | Table 2 | 5.1 | 10.4 | 95.3 | 241 | 0.7 |
| 4001 | | 5.2 | 11.6 | 143.2 | 338 | 0.9 |
| 5021 | | 10.9 | 20.9 | 240 | 642 | 1.4 |

TABLE 7. Atkin's algorithm with modular composition

Notice that in all the columns labelled "fast" in Tables 2–7, the timings reflect the already mentioned (piecewisely almost constant) behaviour of the FFT: polynomial multiplication in the degree range 1024–2047 is roughly twice as fast as in the range 2047–4095 and roughly four times as fast as in the range 4096–8191.

Finally, Table 8 gives timings for Stark's algorithm Stark1972; apart from the common computation of $\wp$ and $\tilde{\wp}$, we distinguish the time necessary to compute all inverses (the quadratic algorithm when available, followed by that using fast inversion) and that for deducing the polynomials $q_n$.

| $\ell$ | $\wp, \tilde{\wp}$ | Inverses | | $q_n$ |
|---|---|---|---|---|
| | | quadratic | fast | |
| 1013 | | 23542 | 1222.7 | 28.0 |
| 2039 | See | ?? | 5113.4 | 116.9 |
| 3019 | Table 2 | ?? | 12182 | 258 |
| 4001 | | ?? | 20388 | 418.6 |
| 5021 | | ?? | 38910 | 663.1 |

TABLE 8. Stark's algorithm Stark1972

**Acknowledgments.** We thank Pierrick Gaudry for his remarks during the elaboration of the ideas contained in this work.

## REFERENCES

[1] C. Alonso, J. Gutierrez, and T. Recio. A rational function decomposition algorithm by near-separated polynomials. *Journal of Symbolic Computation*, 19(6):527–544, 1995.

[2] A. O. L. Atkin. The number of points on an elliptic curve modulo a prime (II). Draft. Available at `http://listserv.nodak.edu/archives/nmbrthry.html`, July 1992.

[3] D. J. Bernstein. Composing power series over a finite ring in essentially linear time. *Journal of Symbolic Computation*, 26(3):339–341, 1998.

[4] I. Blake, G. Seroussi, and N. Smart. *Elliptic curves in cryptography*, volume 265 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1999.

[5] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity*, pages 151–176. Academic Press, New York, 1976. Proceedings of a Symposium held at Carnegie-Mellon University, Pittsburgh, Pa., 1975.

[6] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *Journal of Algorithms*, 1(3):259–295, 1980.

[7] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25(4):581–595, 1978.

[8] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[9] L. S. Charlap, R. Coley, and D. P. Robbins. Enumeration of rational points on elliptic curves over finite fields. Draft, 1991.

[10] S. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966.

[11] J.-M. Couveignes. *Quelques calculs en théorie des nombres*. Thèse, Université de Bordeaux I, July 1994.

[12] J.-M. Couveignes. Computing $l$-isogenies using the $p$-torsion. In H. Cohen, editor, *Algorithmic Number Theory*, volume 1122 of *Lecture Notes in Computer Science*, pages 59–65. Springer-Verlag, 1996. Proceedings of the Second International Symposium, ANTS-II, Talence, France, May 1996.

[13] J.-M. Couveignes. Isomorphisms between Artin-Schreier towers. *Mathematics of Computation*, 69(232):1625–1631, 2000.

[14] J.-M. Couveignes, L. Dewaghe, and F. Morain. Isogeny cycles and the Schoof-Elkies-Atkin algorithm. Research Report LIX/RR/96/03, LIX, April 1996. Available at `http://www.lix.polytechnique.fr/Labo/Francois.Morain/`.

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

[16] J.-M. Couveignes and F. Morain. Schoof's algorithm and isogeny cycles. In L. Adleman and M.-D. Huang, editors, *Algorithmic Number Theory*, volume 877 of *Lecture Notes in Computer Science*, pages 43–58. Springer-Verlag, 1994. 1st Algorithmic Number Theory Symposium - Cornell University, May 6-9, 1994.

[17] L. Dewaghe. Isogénie entre courbes elliptiques. *Utilitas Mathematica*, 55:123–127, 1999.

[18] N. D. Elkies. Explicit isogenies. Draft, 1992.

[19] N. D. Elkies. Elliptic and modular curves over finite fields and related computational issues. In D. A. Buell and J. T. Teitelbaum, editors, *Computational Perspectives on Number Theory: Proceedings of a Conference in Honor of A. O. L. Atkin*, volume 7 of *AMS/IP Studies in Advanced Mathematics*, pages 21–76. American Mathematical Society, International Press, 1998.

[20] M. Fouquet and F. Morain. Isogeny volcanoes and the SEA algorithm. In C. Fieker and D. R. Kohel, editors, *Algorithmic Number Theory*, volume 2369 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 2002. Proceedings of the 5th International Symposium, ANTS-V, Sydney, Australia, July 2002.

[21] S. Galbraith. Constructing isogenies between elliptic curves over finite fields. *Journal of Computational Mathematics*, 2:118–138, 1999.

[22] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.

[23] J. Gutierrez and T. Recio. A practical implementation of two rational function decomposition algorithms. In *Proceedings ISSAC'92*, pages 152–157. ACM, 1992.

[24] D. Jao, S. D. Miller, and R. Venkatesan. Do all elliptic curves of the same order have the same difficulty of discrete log? In Bimal Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 21–40, 2005. 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005.

[25] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of Computation*, 67(223):1179–1197, 1998.

[26] K. S. Kedlaya. Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology. *Journal of the Ramanujan Mathematical Society*, 16(4):323–338, 2001.

[27] D. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkeley, 1996.

[28] H. T. Kung. On computing reciprocals of power series. *Numerische Mathematik*, 22:341–348, 1974.

[29] R. Lercier. Computing isogenies in $F_{2^n}$. In H. Cohen, editor, *Algorithmic Number Theory*, volume 1122 of *Lecture Notes in Computer Science*, pages 197–212. Springer Verlag, 1996. Proceedings of the Second International Symposium, ANTS-II, Talence, France, May 1996.

[30] R. Lercier. *Algorithmique des courbes elliptiques dans les corps finis*. Thèse, École polytechnique, June 1997.

[31] R. Lercier and F. Morain. Computing isogenies between elliptic curves over $F_{p^n}$ using Couveignes's algorithm. *Mathematics of Computation*, 69(229):351–370, January 2000.

[32] F. Morain. Calcul du nombre de points sur une courbe elliptique dans un corps fini : aspects algorithmiques. *Journal de Théorie des Nombres de Bordeaux*, 7(1):255–282, 1995.

[33] V. Müller. *Ein Algorithmus zur Bestimmung der Punktanzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer drei*. PhD thesis, Technischen Fakultät der Universität des Saarlandes, 1995.

[34] Alexander R. and Anton S. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `http://eprint.iacr.org/`.

[35] T. Satoh. The canonical lift of an ordinary elliptic curve over a finite field and its point counting. *Journal of the Ramanujan Mathematical Society*, 15:247–270, 2000.

[36] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Mathematisches Institut der Universität Tübingen, 1982. Preliminary report.
[37] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
[38] R. Schoof. Counting points on elliptic curves over finite fields. *Journal de Théorie des Nombres de Bordeaux*, 7(1):219–254, 1995.
[39] V. Shoup. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation*, 20(4):363–397, 1995.
[40] V. Shoup. *The Number Theory Library*. 1996–2005. `http://www.shoup.net/ntl`.
[41] M. Sieveking. An algorithm for division of powerseries. *Computing*, 10:153–156, 1972.
[42] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 1986.
[43] H. M. Stark. Class-numbers of complex quadratic fields. In W. Kuyk, editor, *Modular functions of one variable I*, volume 320 of *Lecture Notes in Mathematics*, pages 155–174. Springer Verlag, 1973. Proceedings International Summer School University of Antwerp, RUCA, July 17-Agust 3, 1972.
[44] E. Teske. An elliptic trapdoor system. *Journal of Cryptology*, 19(1):115–133, 2006.
[45] J. Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences, Série I*, 273:238–241, juillet 1971.
[46] R. Zippel. Rational function decomposition. In Stephen M. Watt, editor, *Symbolic and algebraic computation*, pages 1–6, New York, 1991. ACM Press. Proceedings of ISSAC'91, Bonn, Germany.