Finding the Eigenvalue in Elkies' Algorithm

Markus Maurer and Volker Müller

CONTENTS

One essential part of Elkies' algorithm for computing the group order of an elliptic curve defined over a finite field is the determination of the eigenvalue of the Frobenius endomorphism. Here we compare from a practical point of view several strategies for this search: the use of rational functions, the use of division polynomials, the babystep-giantstep method, and a new modification of this method that avoids the need for two fast exponentiations.

1. INTRODUCTION

Elliptic curves over finite fields have gained a lot of attention in public key cryptography in recent years. Nowadays several institutions are standardizing such systems; see [IEEE 2000], for example. Therefore the question arises how one can find elliptic curves for which the discrete logarithm problem is supposedly hard. In [Müller and Paulus 1998], the authors solve this parameter search by computing the group order of randomly chosen elliptic curves. Group order computations for elliptic curves have made astonishing progress in the last few years; see, for example [Blake et al. 1999; Lehmann et al. 1994; Müller 1995; Lercier 1997]. The Atkin–Elkies algorithm [Atkin 1988; 1992], briefly reviewed in the next section, proved to be very efficient in practice; for a detailed description see [Blake et al. 1999; Müller 1995]. One essential part of it is the determination of the eigenvalue of the so called Frobenius endomorphism modulo suitable small primes. In this paper we compare several strategies for finding this eigenvalue in practice.

We restrict our observations to elliptic curves defined over finite prime fields of large characteristic, but very similar algorithms are also usable for small characteristic fields. The paper is structured as follows: Section 2 gives a short introduction to elliptic curves defined over finite fields of characteristic greater three and the Atkin–Elkies algorithm for

Keywords: elliptic curve, Elkies' algorithm, point counting

1058-6458/2001 $0.50 per page

point counting. In Section 3, we give an overview on the four different algorithms of this paper, and we give average case running times for all these algorithms. Sections 4, 5, and 6 describe each of these algorithms in detail. Finally we describe an important subalgorithm of these methods, a fast probabilistic search strategy in a table of rational functions.

2. THE ATKIN–ELKIES ALGORITHM

Let $\mathbb{F}_q$ be the finite field with q elements, and let $p > 3$ be its (prime) characteristic. Let E be an elliptic curve defined over $\mathbb{F}_q$ by its short Weierstrass equation

$$y^2 = x^3 + a_4 x + a_6, \tag{2–1}$$

where $a_4, a_6 \in \mathbb{F}_q$, and the discriminant $4a_4^3+27a_6^2$ is nonzero. Let $\overline{\mathbb{F}}_q$ be the algebraic closure of $\mathbb{F}_q$. For a field K, $\mathbb{F}_q \subset K \subset \overline{\mathbb{F}}_q$, the set $E(K)$ of K-rational points consists of the affine solutions $(x, y) \in K^2$ of (2–1), together with the point $\mathcal{O}$ "at infinity" obtained by considering the projective closure of (2–1). The set $E(K)$ has a group structure (usually additively written) given by the "tangent and chord method" [Connell 1996], with $\mathcal{O}$ acting as the identity element. The sum of two given points can be computed using the following simple formulas and a few trivial cases:

- The negative of a point $P = (x, y) \in E(K)$ is $-P = (x, -y) \in E(K)$.
- Given points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, with $P_1 \neq -P_2$, we compute the sum (x_3, y_3) as

  $$x_3 = -x_1-x_2+\lambda^2 \quad \text{and} \quad y_3 = -y_1+\lambda\cdot(x_1-x_3), \tag{2–2}$$

  where

  $$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } x_1 \neq x_2, \\[2ex] \dfrac{3x_1^2 + a_4}{2y_1} & \text{if } x_1 = x_2. \end{cases}$$

The algorithms of Atkin [1988; 1992] and Elkies [n.d.] solve a fundamental problem: Given a finite field $\mathbb{F}_q$ and an elliptic curve

$$E = (a_4, a_6) \in \mathbb{F}_q^2,$$

find the number of $\mathbb{F}_q$-rational points on E. A detailed description of both algorithms and their implementation can be found in [Müller 1995] (or in short form, [Lehmann et al. 1994]), and we just sketch the basic ideas here. Both algorithms use properties of the so called *Frobenius endomorphism*:

$$\begin{aligned} \Phi_E : E(\overline{\mathbb{F}}_q) &\longrightarrow E(\overline{\mathbb{F}}_q) \\ (x, y) &\longmapsto (x^q, y^q). \end{aligned}$$

The basic idea of these algorithms is the computation of the group order modulo sufficiently many small primes l, and then use the Chinese Remainder Theorem to determine the group order. The primes l can be divided into two classes: for so called *Atkin primes*, there is no subgroup of $E(\overline{\mathbb{F}}_q)$ of order l which is invariant under the Frobenius endomorphism, whereas for *Elkies primes* there exists at least one such subgroup C. Therefore $\Phi_E(Q) = \alpha\cdot Q$ for all points $Q \in C$ and some integer $\alpha \in \{1, \ldots, l-1\}$. We denote α the *eigenvalue* of Φ_E modulo l. To determine α (which directly induces the group order modulo l), the algorithm of Elkies consists of two steps:

1. Determine for some generating point $Q \in C$ the polynomial

   $$f_C(X) = \prod_{i=1}^{(l-1)/2} \big(X - x(iQ)\big) \in \mathbb{F}_q[X],$$

   where $x(H)$ denotes the x-coordinate of a point H.
2. Use $f_C(X)$ to determine α.

In this paper we focus on the second step of Elkies' algorithm. On average, this part takes about thirty to forty percent of the total running time of the point counting algorithm.

3. EIGENVALUE SEARCH ALGORITHMS

In this section we give a short overview of the four algorithms that we will discuss in this paper, and present run time data for them. Detailed descriptions of the different methods can be found in the following sections.

The basic idea of all algorithms for finding the eigenvalue α is that we can transform the equation $\Phi_E(Q) = \alpha\cdot Q$ for points $Q \in C$ into polynomial equations modulo $f_C(X)$. This transformation was first given by Schoof [1985]. Thus we have to check which integer $1 \leq \alpha \leq l-1$ satisfies

$$(X^q, Y^q) \equiv \alpha\cdot(X, Y) \mod \bigl(f_C(X), Y^2 - X^3 - a_4X - a_6\bigr), \quad (3\text{–}1)$$

where $\alpha\cdot(X, Y)$ denotes the α-th multiple of a formal point (X, Y) on E. Once we find an integer α in the given range with equivalence of x- and y-coordinate in (3–1) we have determined the eigenvalue of Φ_E. From a practical point of view however, it is sufficient to check only equivalence of the y-coordinate of (3–1). Although unproven, we never found a counterexample. Since the y-coordinate $y(-P)$ of $-P$ is $-y(P)$, we compute the right side of (3–1) for $1 \leq \alpha \leq (l-1)/2$ and test whether the y-coordinate or its negative equals Y^q.

We will examine four algorithms for finding the eigenvalue α:

1. the rational function method,
2. the division polynomial method,
3. the babystep-giantstep method, and
4. a modified babystep-giantstep method.

The first two are standard algorithms for the given problem. Since they are rather slow for large primes l, the usage of Shanks' babystep-giantstep algorithm was suggested to improve the running time. The last algorithm is based on a new idea and described for the first time in this paper.

Table 1 gives average running times for all these algorithms, for 50 eigenvalue computations of randomly chosen curves defined over a 50-digit and a 100-digit prime field. It shows that the modified babystep-giantstep algorithm introduced here has the least average running time already for relatively small primes l. Table 2 shows similar data for variants of the four algorithms discussed here, applicable to elliptic curves defined over fields of characteristic two.

All experimental results in this article were obtained with the point counting program eco_prime (eco_gf2n in characteristic 2), contained in the LiDIA package [LiDIA 2000]. The eco_prime program was used on an Intel Celeron 300A processor with 64 MB of main memory.

4. RATIONAL FUNCTIONS AND DIVISION POLYNOMIALS

We describe briefly two standard methods for finding the eigenvalue in Elkies algorithm: the rational function method and the division polynomial method.

Using Rational Functions

The rational function method is directly based on the addition formulas given in (2–2). In [Connell 1996] it is shown that the i-th multiple of a formal point (X, Y) can be represented as

$$i\cdot\bigl(X, Y\bigr) = \bigl(h_{1,i}(X), Y\cdot h_{2,i}(X)\bigr),$$

where the $h_{j,i}(X) \in \mathbb{F}_q(X)$, for $j = 1, 2$, are rational functions. Assume we know the rational functions representing the two points $i\cdot(X, Y)$ and $r\cdot(X, Y)$. Then we can determine the rational function representation of $(i+r)\cdot(X, Y)$ by substituting the coordinates into the addition formulas (2–2). By counting basic polynomial operations for rational functions we can perform an addition of different points with 2 squarings and 19 multiplications modulo $f_C(X)$, doubling a point needs 4 squarings and 13 multiplications modulo $f_C(X)$, and the addition of the formal point (X, Y) only requires 2 squarings and 7 multiplications. Here, we do not count linear operations like multiplication with a scalar.

prime l	rational function	division polynom.	babystep-giantstep	modified babystep-giantstep	prime l	rational function	division polynom.	babystep-giantstep	modified babystep-giantstep
101	6.6	4.6	7.3	3.8	101	18.8	14.5	25.5	13.1
211	51.6	30.8	28.8	17.1	211	134.1	70.0	92.5	50.0
307	125.1	72.0	61.1	36.6	307	315.5	158.2	178.9	105.2
401	131.9	83.3	73.1	54.3	401	293.8	193.6	225.4	141.6

TABLE 1. Average running time in seconds for the eigenvalue search of 50 randomly chosen curves, defined over a 50-digit (left) and over a 100-digit (right) prime field.

prime l	rational function	division polynom.	babystep-giantstep	modified babystep-giantstep
101	50.7	11.8	19.3	6.8
211	201.1	72.8	97.2	42.2
307	514.2	171.7	219.4	89.9
401	987.9	323.1	355.4	127.2

TABLE 2. Average running time in seconds for the eigenvalue search of 50 randomly chosen curves, defined over the field $\mathbb{F}_{2^{165}}$ (chosen because 2^{165} has 50 decimal digits).

All the polynomials can be reduced by $f_C(X)$ and $Y^2 - X^3 - a_4X - a_6$, so that their X-degree and Y-degree is at most $(l-1)/2 - 1$ and 1, respectively. Rational functions are especially useful if memory is very restricted. Combined with fast multiplication methods using addition-subtraction chains (see, e.g., [Gordon 1998]) only 3 formal points (or equivalently 12 polynomials in the variable X) have to be stored.

Using Division Polynomials

Schoof [1985] used *division polynomials* to compute $i\cdot(X,Y)$ for $1 \le i \le (l-1)/2$. These polynomials are defined as follows, where ψ_i should be read as $\psi_i(X,Y)$:

$$\begin{aligned}
\psi_0 &= 0,\\
\psi_1 &= 1,\\
\psi_2 &= 2Y,\\
\psi_3 &= 3X^4 + 6a_4X^2 + 12a_6X - a_4^2,\\
\psi_4 &= 4Y\cdot\bigl(X^6 + 5a_4X^4 + 20a_6X^3 - 5a_4^2X^2\\
&\qquad\qquad - 4a_4a_6X - 8a_6^2 - a_4^3\bigr),\\
\psi_{2i} &= \frac{\psi_i}{2Y}\cdot\bigl(\psi_{i+2}\psi_{i-1}^2 - \psi_{i-2}\psi_{i+1}^2\bigr) \qquad \text{for } i \ge 3,\\
\psi_{2i+1} &= \psi_{i+2}\psi_i^3 - \psi_{i+1}^3\psi_{i-1} \qquad \text{for } i \ge 2.
\end{aligned}$$

To overcome recursion in the computation of division polynomials, we compute division polynomials with growing index and store all polynomials that will be needed later (together with their square and cube). The computation of a new division polynomial then takes at most 3 multiplications modulo $f_C(X)$ (plus a squaring and one more multiplication for the square and cube of it, if needed). Then we can compute $i\cdot(X,Y)$ as

$$i\cdot(X,Y) = \left(X - \frac{\psi_{i-1}\psi_{i+1}}{\psi_i^2},\ \frac{\psi_{i+2}\psi_{i-1}^2 - \psi_{i-2}\psi_{i+1}^2}{4Y\,\psi_i^3}\right). \tag{4–1}$$

Thus, to check a list of m candidates for the eigenvalue of Frobenius with maximal element α_m, we need at most $4\alpha_m + 2m$ multiplications and α_m squarings modulo $f_C(X)$, using a table with at most $3\alpha_m/2$ polynomials of degree at most $(l-1)/2 - 1$.

As Table 1 shows, the division polynomial method is always faster than the rational function method. The random access memory of today's computers is large enough that table storage requirements do not prevent the faster method from being usable even for giant group order computations. Table 3 gives a more detailed comparison of the running time of both methods. It lists the minimal, maximal, and average time for finding the eigenvalue in 50 randomly chosen eigenvalue computations for a finite prime field with 50-digit prime characteristic, and 50 for a field with 100-digit prime characteristic.

5. A BABYSTEP-GIANTSTEP METHOD

The fast exponentiation part becomes less important the bigger the prime l becomes. Therefore faster methods for finding the correct value out of a list of candidates can further improve the running

l	rational function			division polynomial			fast exponentiation Y^q	l	rational function			division polynomial			fast exponentiation Y^q
101	2.5	**6.6**	10.9	2.5	**4.6**	7.3	2.6	101	10.8	**18.8**	27.5	10.2	**14.7**	20.1	10.1
211	8.8	**51.6**	102.0	9.0	**30.8**	48.6	8.8	211	36.9	**134.8**	220.4	36.8	**70.0**	117.3	36.7
307	16.7	**125.1**	275.8	16.7	**72.0**	127.1	16.5	307	66.6	**315.5**	591.9	66.4	**158.2**	287.2	66.4
401	18.5	**131.9**	224.3	18.5	**83.3**	149.5	18.5	401	76.2	**293.8**	497.0	75.9	**193.6**	342.8	75.6

TABLE 3. Minimal, **average**, and maximal time in seconds for finding the eigenvalue of 50 randomly chosen eigenvalue computations for curves defined over a 50-digit (left) and a 100-digit (right) prime characteristic, together with the time for computing Y^q.

time. This section describes one such method, based on the babystep-giantstep idea of Shanks.

Shanks' algorithm applies in many situations in which we have a search problem in a large list of candidates connected by algebraic relations. Here the idea works as follows: Let $1 \le \alpha_1 < \cdots < \alpha_m$ be a list of m candidates for the eigenvalue α. Set $K = \lceil \sqrt{\alpha_m/2}\, \rceil$. Then we try to determine integers $0 \le j \le K$ and $-K \le i \le K$ such that

$$(X^q, Y^q) - j\cdot(2K)\cdot(X,Y) \equiv i\cdot(X,Y) \mod \bigl(f_C(X), Y^2 - X^3 - a_4X - a_6\bigr).$$

Given such integers, we directly can deduce $\alpha \equiv j(2K) + i \bmod l$. The following lemma proves the existence of such integers.

Lemma 5.1. *Let α_m be a positive integer and set $K = \lceil \sqrt{\alpha_m/2}\, \rceil$. For each integer α with $1 \le \alpha \le \alpha_m$, there exist integers i, j with $0 \le i, j \le K$, such that either $\alpha = j(2K) + i$ or $\alpha = j(2K) - i$.*

Proof. Division with remainder, where the remainder is chosen as absolute smallest integer, yields $\alpha = j(2K) + i$, with $-K \le i \le K$. We have

$$j = (\alpha - i)/(2K) \le (\alpha + K)/(2K) \le \alpha_m/(2K) + 1/2 \le K + 1/2.$$

Hence, $0 \le j \le K$. □

Elliptic curves offer the advantage that we can check the two points $i\cdot(X,Y)$ and $-i\cdot(X,Y)$ simultaneously for free. To determine integers i and j for the Frobenius eigenvalue α, the algorithm first computes all points $i\cdot(X,Y)$ for $0 \le i \le K$ and stores the y-coordinates together with the corresponding index i in a table (the babysteps). Then all the points on the left hand side of the shown equation are computed (the giantsteps); for each point we check whether its y-coordinate already shows up in the babystep table. If so, we have found i and j. This idea leads to the following algorithm.

Algorithm 5.2 (Babystep-giantstep).
Input: maximum candidate α_m for eigenvalue; Elkies polynomial $f_C(X)$
Output: eigenvalue α

1. set $K = \lceil \sqrt{\alpha_m/2} \rceil$
2. **for** $i = 0$ **to** K
   a. compute y-coordinate of $i\cdot(X,Y) \bmod f_C(X)$
   b. store y-coordinate at index i in a table
3. compute point $(X^q, Y^q) \bmod f_C(X)$
4. **for** $j = 0$ **to** K
   a. compute y-coordinate of
      $(X^q, Y^q) - j\cdot(2K)\cdot(X,Y) \bmod f_C(X)$
   b. **if** y-coordinate is stored in table at index i
      return $\alpha = 2jK + i$
   c. **if** $-(y\text{-coordinate})$ is stored in table at index i
      return $\alpha = 2jK - i$

From this description it directly follows that at most $2K - 1 \le 2\lceil \sqrt{(l-1)/2}\, \rceil - 1$ point additions of formal points are needed. For large l, this number is significantly smaller than the expected number of point additions in the rational function method and of iterations in the division polynomial method, which is $(l-1)/4$. This fact explains the faster average run time of Algorithm 5.2 (compared to the rational function and division polynomial method) shown in Tables 1.

It follows from the analysis of the costs of point operations in Section 4, that the computation of the points $i\cdot(X,Y)$ needed in the babysteps should be done with division polynomials and equation (4–1).

Assume that we find an integer s such that the set of eigenvalue candidates modulo s is small. Then it is advantageous to choose K "near" $\lceil \sqrt{\alpha_m/2}\, \rceil$, but divisible by s. Then we can exclude several possible values for $i \bmod s$, and we only have to consider indices i which are elements of the small set of candidates modulo s (note that $K \equiv 0 \bmod s$).

An important algorithmic question in Algorithm 5.2 is the implementation of the commands in line (4b) and (4c). Note that all the points computed in either the babysteps or the giantsteps are rational functions, where numerator and denominator are reduced by $f_C(X)$ and $Y^2 - X^3 - a_4X - a_6$. Therefore there remains the problem how to check in an efficient way whether two rational functions are equivalent modulo $(f_C(X), Y^2 - X^3 - a_4X - a_6)$. We will describe a new fast probabilistic algorithm for this problem in Section 8. This probabilistic algorithm improves the overall performance of Algorithm 5.2 significantly, as can be seen in Table 6.

A serious drawback of Algorithm 5.2 is the fact that we need both polynomials X^q and Y^q to determine the y-coordinate of the "giantstep points" $(X^q, Y^q) - j(2K)\cdot(X,Y)$ in step (4a). Thus we have to perform two fast polynomial exponentiations, one for Y^q mod $(f_C(X),\ Y^2-X^3-a_4X-a_6)$, and one for X^q mod $f_C(X)$. Thus the speed advantages given by the smaller number of point additions diminishes.

6. A MODIFIED BABYSTEP-GIANTSTEP ALGORITHM

In this section we present a new idea similar to the babystep-giantstep idea in Section 5, which does not have the disadvantage of two fast exponentiations. The modified babystep-giantstep algorithm uses the fact that for most integers l, α we can find small integers i, j such that $\alpha = 2^{-i} \cdot j \bmod l$, and so

$$2^i \cdot (X^q, Y^q) = j \cdot (X, Y) \bmod \big(f_C(X),\ Y^2 - X^3 - a_4X - a_6\big). \tag{6–1}$$

From the addition formulas (2–2), the advantage of this idea becomes clear: computing the x-coordinate of $2^i \cdot (X^q, Y^q)$ can be done without knowledge of Y^q. The formula (2–2) for computing the x-coordinate of twice a formal point P only contains $y(P)^2$, which can be substituted with the right-hand side of the curve equation. With a test for the x-coordinate of (6–1) we can then determine the eigenvalue, up to the sign.

This idea leads to the following algorithm. Assume that we know bounds K_B and K_G, respectively, such that for every eigenvalue candidate α' there exist integers $0 \le i \le K_B$ and $1 \le j \le K_G$ with $2^i \alpha' \equiv j \bmod l$. Then we first compute the x-coordinates of the left-hand side of (6–1) for all $0 \le i \le K_B$ (the babysteps) and store the rational functions (together with corresponding indices i) in a table. Then we test the right-hand side of (6–1) for all odd $1 \le j \le K_G$ (the giantsteps).

Algorithm 6.1 (Modified babystep-giantstep).
Input: bounds K_B, K_G for number of babysteps and giantsteps; Elkies polynomial $f_C(X)$
Output: $\pm\alpha$
1. compute X^q mod $f_C(X)$
2. **for** $i = 0$ **to** K_B
   a. compute x-coordinate of $2^i \cdot (X^q, \cdot)$ mod $f_C(X)$
   b. store x-coordinate at index i in table
3. **for** $j = 1$ **to** K_G
   a. compute x-coordinate of $j \cdot (X, Y)$ mod $f_C(X)$
   b. **if** x-coordinate is stored in table at index i
      return $\alpha = \pm 2^{-i} \cdot j \bmod l$

Our earlier analysis of the costs of point operations (Section 4) shows that the computation of step (3a) should be done with division polynomials. There remains then the question about the optimal choice for the bounds K_B, K_G and the expected/maximal number of operations which are performed by Algorithm 6.1. Since the computational costs (in polynomial multiplications, squarings) for step (2a) and (3a) are different, we use the weighted cost function $\mathrm{cost}(i,j) = i \cdot c_1 + j \cdot c_2$, where c_1 and c_2 are the costs of step (2a) and (3a), respectively. For any eigenvalue candidate, we determine then all possible values for i, j (note that there are several possibilities to choose i, j), and we choose the pair with minimal costs. We do this computations for all eigenvalue candidates and set the bounds K_B, K_G correspondingly. Since l is rather small, this computation can be done "brute force" (trying all possibilities).

Table 4 lists a few values for K_B, K_G. Moreover we present the expected (under the assumption that we have no information about α; that is to say, all integers 1, …, $l-1$ have the same probability of being the eigenvalue) and the maximal number of point additions needed. The expected/maximal number of point additions is computed by counting the number of necessary polynomial multiplications and squarings, and then dividing by the corresponding values for point addition given in the beginning of Section 4.

Table 4 shows that the expected number of point additions in Algorithm 6.1 is approximately equal to the number of babysteps in Algorithm 5.2, the maximal number is only about twice the number of babysteps. This statement is also true for other primes l. Therefore modified babystep-giantstep (Algorithm 6.1) has an advantage over standard babystep-giantstep (Algorithm 5.2) in that although the number

l	K_B	K_G	exp. # of additions	max. # of additions	K in Alg. 5.2
101	12	10	6.7	9.6	7
211	22	16	10.9	16.4	10
307	23	25	13.2	21.6	12
401	42	32	18.6	32.1	14

TABLE 4. Optimal values for number of babysteps and giantsteps in Algorithm 6.1, and expected and maximal number of point additions.

of expected point additions is the same, only one fast exponentiation is needed. A disadvantage of Algorithm 6.1 is its ambiguous output. Section 7 addresses the problem of computing the sign of the eigenvalue.

Running Times

Table 1 showed that the modified babystep-giantstep algorithm 6.1 is the fastest of all search algorithms described in this paper. Table 5 presents more detailed data on Algorithm 5.2 and 6.1. Again 50 random eigenvalue computations were done for a 50-digit and a 100-digit prime field.

7. FINDING THE SIGN OF AN EIGENVALUE

In this section we will present two algorithms to compute the sign of the eigenvalue of Frobenius. We assume that we already know the eigenvalue modulo the sign.

Dewaghe's Algorithm for Finding the Sign

We first present a generalized version of a theorem of Dewaghe [1998] and show how this theorem can be used to compute the sign of the eigenvalue for primes $l \equiv 3 \bmod 4$.

Theorem 7.1. Let l be an odd prime. Assume that α is an eigenvalue of the Frobenius endomorphism, and $g(X) \in \mathbb{F}_q[X]$ is a divisor of degree d of the corresponding Elkies polynomial $f_C(X)$. Then

$$\operatorname{res}\bigl(g(X),\, X^3 + a_4X + a_6\bigr)^{(q-1)/2} = \alpha^d.$$

Proof. Let $P \in C$ be a nontrivial point. It follows from [Vélu 1971] that all the roots of $f_C(X)$ are given as the x-coordinates of points $i \cdot P$ with $1 \le i \le (l-1)/2$. We can therefore find an integer $1 \le r \le (l-1)/2$ such that $x(r \cdot P)$ is a root of $g(X)$. Since $g(X)$ is a divisor of $f_C(X)$ defined over $\mathbb{F}_q$, we have $g(X^q) = g(X)^q$, so the Frobenius map permutes the roots of $g(X)$. Therefore the elements of the sequence $(x(\Phi_E^j(r \cdot P)))_{j \ge 0} = (x((r\alpha^j) \cdot P))_{j\ge 0}$ are roots of $g(X)$, and the sequence is periodic with period length d.

We examine the order e of α modulo l. Because the period length is d, we have $e \ge d$, and $x(r \cdot P) = x(\alpha^e \cdot r \cdot P) = x(\alpha^{e-d} \cdot r \cdot P)$. We conclude for the points, that $r \cdot P = \pm\alpha^{e-d} \cdot r \cdot P$, so $\alpha^{2(e-d)} = 1 \bmod l$. Therefore the order of α divides $2(e-d)$, which means that there is an integer k with $ke = 2(e-d)$. Because $d \ge 1$, we have $0 \le ke < 2e$, which implies $0 \le k \le 1$. As a consequence, the order of α is either d or $2d$.

If we define $\mathcal{S} = \{r\alpha^j, 0 \le j < d\}$, all the roots of the polynomial $g(X)$ are given as $x(s \cdot P)$ for $s \in \mathcal{S}$. A basic result for resultants gives

$$\operatorname{res}\bigl(g(X), X^3 + a_4X + a_6\bigr) = \prod_{s \in \mathcal{S}} \bigl(x(s \cdot P)^3 + a_4 x(s \cdot P) + a_6\bigr).$$

Therefore we deduce (noting that q is odd)

$$\begin{aligned}\operatorname{res}\bigl(g(X),\, X^3 + a_4X + a_6\bigr)^{(q-1)/2} &= \operatorname{res}\bigl(g(X),\, X^3 + a_4X + a_6\bigr)^{(q^2-q)/2} \\ &= \prod_{s \in \mathcal{S}} \bigl(x(s \cdot P)^3 + a_4 x(s \cdot P) + a_6\bigr)^{(q^2-q)/2} \\ &= \prod_{s \in \mathcal{S}} \frac{y^{q^2}(s \cdot P)}{y^q(s \cdot P)} = \prod_{s \in \mathcal{S}} \frac{y(\alpha^2 \cdot s \cdot P)}{y(\alpha \cdot s \cdot P)} \\ &= \frac{y(\alpha^{d+1} \cdot r \cdot P)}{y(\alpha \cdot r \cdot P)} = \alpha^d,\end{aligned}$$

where the last transformation holds because, as already shown, either the order of α is d, so $\alpha^d = 1$, or it is $2d$, so $\alpha^d = -1$ and $y(\alpha^{d+1} \cdot r \cdot P) = y(-\alpha \cdot r \cdot P) = -y(\alpha \cdot r \cdot P)$. □

l	babystep-giantstep (Algorithm 5.2)				mod. babystep-giantstep (Algorithm 6.1)				l	babystep-giantstep (Algorithm 5.2)				mod. babystep-giantstep (Algorithm 6.1)			
101	5.0	**7.3**	9.0	*5.0*	2.5	**3.8**	4.2	*2.4*	101	20.5	**25.5**	28.3	*20.3*	10.2	**13.1**	14.0	*10.2*
211	15.5	**28.8**	35.9	*16.5*	9.0	**17.1**	19.2	*7.8*	211	68.1	**92.5**	106.4	*65.9*	34.4	**50.0**	55.0	*32.1*
307	31.3	**61.1**	80.9	*31.2*	14.9	**36.6**	43.1	*14.8*	307	126.2	**178.9**	228.3	*126.1*	71.2	**105.2**	116.6	*60.1*
401	35.0	**73.1**	102.1	*34.6*	20.4	**54.3**	63.5	*16.4*	401	144.6	**225.4**	281.7	*142.8*	75.8	**141.6**	162.4	*67.7*

TABLE 5. Minimal, **average**, and maximal running time in seconds for Algorithm 5.2 and 6.1 for 50 randomly chosen eigenvalue computations over a prime field with 50-digit (left) and 100-digit (right) prime charatersitic. In italics we give the average time to compute (X^q, Y^q) (Algorithm 5.2) or X^q (Algorithm 6.1).

Dewaghe derived from this idea an easy method to determine the sign of the eigenvalue of the Frobenius endomorphism for primes $l \equiv 3 \bmod 4$. Then we can choose $g(X) = f_C(X)$, and $\alpha^{deg(g(X))}$ directly corresponds to the Jacobi symbol for α. Since for primes $l \equiv 3 \bmod 4$, -1 is a nonsquare, exactly one of the two elements α and $-\alpha$ is a square modulo l. Thus Theorem 7.1 shows that a resultant computation (which needs only $O(l)$ operations in $\mathbb{F}_q$) uniquely determines the sign of the eigenvalue.

For primes $l \equiv 1 \bmod 4$, this fact is no longer true, and the sign of the eigenvalue can no longer be computed in this way. It should however be noted that only half of the elements in $(\mathbb{Z}/l\mathbb{Z})^*$ are squares such that a resultant computation with $g(X) = f_C(X)$ can be used to lower the number of candidates for the eigenvalue search by a factor two. Before we start with the search for the eigenvalue with one of the algorithms described in this paper, we compute the resultant $\mathrm{res}(f_C(X), X^3 + a_4X + a_6)$, which will exclude half of the eigenvalue candidates.

A Factorization Approach for Finding the Sign

In this section we show how a proper divisor $g(X)$ of the Elkies polynomial can be computed and used to determine the sign of the Frobenius eigenvalue, even for primes $l \equiv 1 \bmod 4$. Instead of computing $X^q \bmod f_C(X)$ in Algorithm 6.1, we choose a random element $a \in \mathbb{F}_q^*$ and compute the polynomial

$$h(X) \equiv (X + a)^{(q-1)/2} \bmod f_C(X).$$

Note that the computation time for this fast exponentiation is essentially the same as for computing $X^q \bmod f_C(X)$. Given $h(X)$, we can determine $X^q \bmod f_C(X)$ with one more squaring, one multiplication and an addition (note that $(X + a)^q \equiv X^q + a^q \equiv X^q + a \bmod f_C(X)$). Then we can compute $\pm\alpha$ with Algorithm 6.1.

Let d be the least common multiple of the orders of α and $-\alpha$ modulo l, divided by two if even. Then

$$X^{q^d} \equiv X\big((\pm\alpha)^d\cdot(X, Y)\big) \equiv X \bmod f_C(X),$$

and therefore $f_C(X)$ must have a factor of degree d. If d is reasonably small, we use $h(X)$ to search for a factor of $f_C(X)$. As in the case of polynomial factorization methods (see [Cantor and Zassenhaus 1981], for example), we hope that

$$\begin{aligned}\gcd\big((X + a)^{(q^d-1)/2} - 1, f_C(X)\big)\\ = \gcd\big(h(X)^{1+q+q^2+\cdots+q^{d-1}} - 1,\ f_C(X)\big)\end{aligned}$$

gives a nontrivial factor $g(X)$ of $f_C(X)$. If we find such a nontrivial divisor, the following methods can be used to determine the sign of the eigenvalue of Frobenius. If the orders of α and $-\alpha$ modulo l do not both divide d, then we can use a resultant computation with $g(X)$ as described in Theorem 7.1 to determine the sign. Otherwise we test the y-coordinate of (3–1), but $f_C(X)$ is replaced by its divisor $g(X)$ such that computation is speeded up. We compute $Y^q \bmod (g(X), Y^2 - X^3 - a_4X - a_6)$ and compare with $y(\alpha\cdot(X, Y)) \bmod (g(X), Y^2 - X^3 - a_4X - a_6)$. If both rational functions are equal mod $g(X)$, we know that α is the eigenvalue, otherwise $-\alpha$ is the correct result.

Unfortunately the integer d is quite large in most cases, and the extra time to factor $f_C(X)$ seems to be "wasted" just for determining the sign of the eigenvalue. In this case it seems to be better in practice to "accept" the nonuniqueness of the result.

8. PROBABILISTIC FAST SEARCH IN A TABLE OF RATIONAL FUNCTIONS

In the giantstep part of Algorithms 5.2 and 6.1 we have to check whether for a given rational function $a(X)/b(X)$ and a given table of rational functions $u_j(X)/v_j(X)$, $j = 1, \ldots, k$, there exists an index $1 \le i \le k$ with

$$\frac{a(X)}{b(X)} \equiv \frac{u_i(X)}{v_i(X)} \bmod f_C(X). \tag{8–1}$$

This equation is equivalent to

$$a(X)\cdot v_i(X) - b(X)\cdot u_i(X) \equiv 0 \bmod f_C(X), \tag{8–2}$$

but a test based on (8–2) would require $2k$ multiplications of polynomials for only one search. An alternative to this obvious solution is the computation of inverses for $v_j(X)$ (during the computation of the table) and $b(X)$ using an extended gcd algorithm. To transform a table of size k we need k polynomial inversions and multiplications; each search needs one inversion, one multiplication and k

polynomial comparisons only. We will list running times for this variant in Table 6.

A Probabilistic Algorithm

We now describe a different probabilistic approach which is based on an idea of Victor Shoup. This method will find every equality in (8–1), but it additionally "finds" a few spurious values — it gives no assurance that (8–1) really is fulfilled. But this can be easily checked by two polynomial multiplications with the denominators.

Let $d = \deg(f_C(X))$ and $\underline{w} \in \mathbb{F}_q^d$ be a random vector. In the following, we represent each residue class of the ring $\mathbb{F}_q[X]/(f_C(X))$ by its unique element of degree less than d. If $a(X)$ is such an element, then $\bar{a} \in \mathbb{F}_q^d$ denotes the coefficient vector of the polynomial $a(X)$. We define the map

$$\begin{aligned} L_{\underline{w}} : \mathbb{F}_q[X]^*/\big(f_C(X)\big) &\longrightarrow \mathbb{F}_q \\ g(X) &\longmapsto \bar{g} \odot \underline{w}, \end{aligned}$$

where $\odot$ is the inner product between the coefficient vector $\bar{g}$ and the vector $\underline{w}$.

Lemma 8.1. *Let $\underline{w} \in \mathbb{F}_q^d$ be a random vector. The map $L_{\underline{w}}$ just defined is linear. If*

$$a(X)\cdot v(X) - b(X)\cdot u(X) \not\equiv 0 \bmod f_C(X),$$

then

$$L_{\underline{w}}\big(a(X)\cdot v(X) - b(X)\cdot u(X)\big) \neq 0$$

with probability $1 - q^{-1}$.

Proof. The first observation follows from the linearity of the inner product. For proving the second observation, we count the number of "bad vectors" $\underline{w}$ as q^{d-1}. □

The map $L_{\underline{w}}$ can therefore be used to check (with high probability) whether (8–2) is satisfied or not. If (8–2) is not satisfied, then with high probability $L_{\underline{w}}$ is nonzero. If however the result of $L_{\underline{w}}$ is zero, we check (8–2) with two multiplications modulo $f_C(X)$.

This strategy obviously depends on a fast way to evaluate the map $L_{\underline{w}}$. Using the linearity of $L_{\underline{w}}$, we get

$$\begin{aligned} L_{\underline{w}}\big(a(X)\cdot v_i(X) - b(X)\cdot u_i(X) \bmod f_C(X)\big) \\ = L_{\underline{w}}\big(a(X)\cdot v_i(X) \bmod f_C(X)\big) \\ -L_{\underline{w}}\big(b(X)\cdot u_i(X) \bmod f_C(X)\big). \end{aligned}$$

Therefore, we have reduced the problem of computing $L_{\underline{w}}$ to the problem of computing the slightly different function

$$L_{a(X),\underline{w}}\big(v(X)\big) := L_{\underline{w}}\big(a(X)\cdot v(X) \bmod f_C(X)\big)$$

for a fixed polynomial $a(X) \in \mathbb{F}_q[X]$. Define the $d \times d$-matrix

$$M_{a(X)} := \left(\ \bar{a}(X) \ \middle| \ \overline{a(X)\cdot X} \ \middle| \ \cdots \ \middle| \ \overline{a(X)\cdot X^{d-1}} \ \right),$$

where the coefficient vector of $a(X)\cdot X^i \bmod f_C(X)$ forms the i-th column of $M_{a(X)}$ (indexing columns and rows of the matrix from zero). The following theorem shows how this matrix $M_{a(X)}$ can be used to compute the function $L_{a(X),\underline{w}}$.

Theorem 8.2. Any polynomial $v(X) \in \mathbb{F}_q[X]^*/(f_C(X))$ satisfies

$$L_{a(X),\underline{w}}\big(v(X)\big) = \big(M_{a(X)}^T\cdot\underline{w}\big) \odot \bar{v}.$$

Proof. The coefficient vector of the image of any polynomial $v(X)$ under the map

$$\begin{aligned} \mathbb{F}_q[X]/\big(f_C(X)\big) \quad &\longrightarrow \quad \mathbb{F}_q[X]/\big(f_C(X)\big), \\ v(X) &\longmapsto v(X)\cdot a(X) \end{aligned}$$

can be computed as $M_{a(X)}\cdot\bar{v}$. Then we get (with $M_{i,j}$ being the entry in row i and column j of $M_{a(X)}$)

$$\begin{aligned} L_{a(X),\underline{w}}\big(v(X)\big) &= L_{\underline{w}}\big(a(X)\cdot v(X)\big) = \big(M_{a(X)}\cdot\bar{v}\big) \odot \underline{w} \\ &= \sum_{j=0}^{d-1}\left(\sum_{i=0}^{d-1} M_{j,i}\cdot\bar{v}_i\right)\cdot\underline{w}_j \\ &= \sum_{i=0}^{d-1} \bar{v}_i\cdot\left(\sum_{j=0}^{d-1} M_{j,i}\cdot\underline{w}_j\right) \\ &= \big(M_{a(X)}^T\cdot\underline{w}\big) \odot \bar{v}. \end{aligned}$$
□

In our application, the random vector $\underline{w}$ and the polynomial $a(X)$ are fixed for one search, whereas $v(X)$ changes for any rational function stored in the table. Therefore the evaluation of $L_{a(X),\underline{w}}(v(X))$ for many polynomials $v(X)$ can be split into two parts: first we compute and store the vector $M_{a(X)}^T\cdot\underline{w}$, then we use Theorem 8.2 to determine $L_{a(X),\underline{w}}(v(X))$ for all polynomials $v(X)$. This idea leads to the following algorithm for fast probabilistic search in a table of rational functions.

Algorithm 8.3 (Fast search in rational function table).
Input: a rational function $a(X)/b(X)$; a table of rational functions $u_j(X)/v_j(X)$ for $1 \le j \le k$; polynomial $f_C(X)$
Output: index i such that (8–1) is fulfilled, or "no match found"
1. choose a random vector $\underline{w} \in \mathbb{F}_q^d$
2. compute vectors $\underline{v}_a = M_{a(X)}^T \cdot \underline{w}$ and $\underline{v}_b = M_{b(X)}^T \cdot \underline{w}$
3. **for** $i = 1$ **to** k
   if $\underline{v}_a \odot \bar{v}_i = \underline{v}_b \odot \bar{u}_i$
      if $a(X) \cdot v_i(X) - b(X) \cdot u_i(X) \equiv 0 \bmod f_C(X)$
         return i
4. **return** "no match found"

Theorem 8.4. Algorithm 8.3 needs $O\big((d^2+dk)\log(q)^2\big)$ bit operations to check equation (8–1).

Proof. The computation of the matrix $M_{a(X)}$ for given $a(X)$ can essentially be done as one polynomial multiplication modulo $f_C(X)$ (note that multiplication with a linear polynomial is cheaper than a general polynomial multiplication). Therefore the computation of the vectors $\underline{v}_a$ and $\underline{v}_b$ can be done in $O(d^2)$ operations in $\mathbb{F}_q$. For k evaluations of $L_{a(X),\underline{w}}$ we have to compute k inner products which need $O(dk)$ operations in $\mathbb{F}_q$. Each operation in $\mathbb{F}_q$ can be done with $O(\log(q)^2)$ bit operations. □

In a practical implementation, a speed advantage can be gained by choosing the vector $\underline{w}$ not randomly, but as the first unit vector. Then the computation of the matrix $M_{a(X)}$ can be speeded up by a factor approximately two, since we only have to compute the first row of matrix $M_{a(X)}$. So, for the column i, not all the coefficients of $a(X) \cdot X^i$ have to be determined, but only coefficients that are needed for a column of bigger index. Thus the larger the index of the column to compute, the smaller the number of coefficients which have to be updated.

Practical Comparison of Extended GCD and Probabilistic Method

We've implemented the two algorithms of Section 8. Table 6 shows the average time needed for k searches in a table of size k, obtained from 50 eigenvalue computations of randomly chosen elliptic curves defined over prime fields of 50-digit and 100-digit characteristic. The table suggests that the new probabilistic search algorithm is at least three times faster than the extended gcd approach. Moreover almost all the running time of Algorithm 8.3 goes into the precomputation in step 2 (computation of the matrix $M_{a(X)}$ and the vector $\underline{v}_a = M_{a(X)}^T \cdot \underline{w}$).

9. CONCLUSION

We have compared four different methods for finding the eigenvalue of the Frobenius endomorphism, an important part in point counting algorithms. Our timings indicate that the new babystep-giantstep algorithm 6.1, together with the fast table search method of Section 8, is the best method for finding eigenvalues if the prime l is reasonably large (say, bigger than 100) and $l \equiv 3 \bmod 4$. For primes $l \equiv 1 \bmod 4$ however, its nonunique output complicates predictions about the optimal eigenvalue search method. If the characteristic of the field is huge, it may be favorable to use either the division polynomial or the babystep-giantstep method instead; for medium sized fields, our LiDIA implementation eco_prime prefers still to use Algorithm 6.1. So it remains to answer the open question whether there is an equivalent to Dewaghe's method to determine the sign of an eigenvalue for primes $l \equiv 1 \bmod 4$ without factoring the Elkies polynomial.

ACKNOWLEDGEMENTS

We thank an anonymous referee who pointed out to us several weaknesses in the first version of this paper. His help improved the quality and readability

l	k	ext. gcd method	Alg. 8.3	Step 2 of Alg. 8.3
101	5	1.34/ 2.75	0.34/ 0.56	0.33/ 0.55
211	8	6.83/13.98	2.21/ 3.63	2.19/ 3.57
307	9	15.03/30.84	5.13/ 8.39	5.06/ 8.27
401	10	21.64/44.81	9.14/14.84	9.03/14.63

TABLE 6. Average running times in seconds for k searches in a table of size k, with the extended gcd method and Algorithm 8.3, obtained from fifty eigenvalue computations of randomly chosen curves defined over a 50-digit prime field and fifty over a 100-digit prime field (a slash separates the two values). In each case k was chosen to be $\lceil\sqrt{(l-1)/4}\,\rceil$, the maximal number of babysteps and giantsteps in Algorithm 5.2.

of the paper significantly. We also thank Mike Scott for several helpful suggestions.

REFERENCES

[Atkin 1988] A. O. L. Atkin, "The number of points on an elliptic curve modulo a prime, I", preprint, 1988. Author's email address: aolatkin@math.uic.edu.

[Atkin 1992] A. O. L. Atkin, "The number of points on an elliptic curve modulo a prime, II", preprint, 1992.

[Blake et al. 1999] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic curves in cryptography*, London Math. Soc. Lecture Note Series **265**, Cambridge Univ. Press, Cambridge, 1999.

[Cantor and Zassenhaus 1981] D. G. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields", *Math. Comp.* **36**:154 (1981), 587–592.

[Connell 1996] I. Connell, "Elliptic curve handbook", preprint, 1996. See ftp://www.math.mcgill.ca/pub/ECH1/.

[Dewaghe 1998] L. Dewaghe, "Remarks on the Schoof–Elkies–Atkin algorithm", *Math. Comp.* **67** (1998), 1247–1252.

[Elkies n.d.] N. Elkies, "Explicit isogenies", preprint. Author's email address: elkies@math.harvard.edu.

[Gordon 1998] D. M. Gordon, "A survey of fast exponentiation methods", *J. Algorithms* **27**:1 (1998), 129–146.

[IEEE 2000] The Institute of Electrical and Electronics Engineers, "Std 1363: Standard specifications for public key cryptography", technical report, 2000. See http://standards.ieee.org.

[Lehmann et al. 1994] F. Lehmann, M. Maurer, V. Müller, and V. Shoup, "Counting the number of points on elliptic curves over finite fields of characteristic greater than three", pp. 60–70 in *Algorithmic number theory* (Ithaca, NY, 1994), edited by L. M. Adleman and M.-D. Huang, Lecture Notes in Computer Science **877**, Springer, Berlin, 1994.

[Lercier 1997] R. Lercier, *Algorithmiques des courbes elliptiques dans les corps finis*, Ph.D. thesis, 1997.

[LiDIA 2000] The LiDIA Group, "LiDIA: a library for computational number theory, release 2.0", software, Technische Universität Darmstadt, 2000. See http://www.informatik.tu-darmstadt.de/TI/LiDIA.

[Müller 1995] V. Müller, *Die Berechnung der Punktanzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer* 3, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany, 1995.

[Müller and Paulus 1998] V. Müller and S. Paulus, "On the generation of cryptographically strong elliptic curves", preprint, 1998.

[Schoof 1985] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Math. Comp.* **44**:170 (1985), 483–494.

[Vélu 1971] J. Vélu, "Isogénies entre courbes elliptiques", *C. R. Acad. Sci. Paris Sér. A* **273** (1971), 238–241.

Markus Maurer, Centre for Applied Cryptographic Research, Department of C & O, University of Waterloo, Waterloo, Ontario, Canada, N2l 3G1 (m2maurer@cacr.math.uwaterloo.ca)

Volker Müller, Universitas Kristen Duta Wacana, Jl. Dr. Wahidin 5–19, 55224 Yogyakarta, Indonesia (vmueller@ukdw.ac.id)

Received October 8, 1999; accepted in revised form January 11, 2001