9.1 Schoof's Algorithm

In the early 1980's, René Schoof [3] introduced the first polynomial-time algorithm to compute $\#E(\mathbb{F}_q)$. Extensions of Schoof's algorithm remain the point-counting method of choice when the characteristic of $\mathbb{F}_q$ is large (e.g., when q is a cryptographic size prime).[1]

Schoof's basic strategy is very simple: compute the the trace of Frobenius t modulo many small primes ℓ and use the Chinese remainder theorem to obtain t; this then determines $\#E(\mathbb{F}_q) = q + 1 - t$. The high-level structure of the algorithm appears below.

Algorithm 9.1. Given an elliptic curve E over a finite field $\mathbb{F}_q$ compute $\#E(\mathbb{F}_q)$ as follows:

1. Set $M = 1$ and $t_M = 0$.
2. While $M \le 4\sqrt{q}$, for increasing primes $\ell = 2, 3, 5, \ldots$ that do not divide q:
   a. Compute $t_\ell = t \bmod \ell$.
   b. Set $t_M = \big((M^{-1} \bmod \ell)Mt_\ell + (\ell^{-1} \bmod M)\ell t_M\big) \bmod \ell M$, and then set $M = \ell M$.
3. If $t_M > M/2$ then set $t_M = t_M - M$.
4. Return $q + 1 - t_M$.

Step 2b uses an iterative version of the Chinese remainder theorem to ensure $t_M = t \bmod M$ holds throughout.[2] Once M exceeds $4\sqrt{q}$ (the width of the Hasse interval), t_M uniquely determines the integer t. The key to the algorithm is the implementation of step 2a, which is described in the next section. Let us first consider the primes ℓ that the algorithm uses.

Let ℓ_{max} be the largest prime ℓ for which the algorithm computes t_ℓ. The Prime Number Theorem implies

$$\sum_{\text{primes } \ell \le x} \log \ell \sim x,$$

so $\ell_{max} \approx \log 4\sqrt{q} \approx \frac{1}{2}n = O(n)$, and we need $O\big(\frac{n}{\log n}\big)$ primes ℓ (as usual, $n = \log q$). The cost of updating t_M and M is bounded by $O(\mathsf{M}(n)\log n)$, thus if we can compute t_ℓ in polynomial time, then the whole algorithm will run in polynomial time.

9.2 Computing the trace of Frobenius modulo a prime ℓ.

We first consider the case $\ell = 2$. Then q must be odd, and $t = q + 1 - \#E(\mathbb{F}_q)$ is divisible by 2 if and only if $\#E(\mathbb{F}_q)$ is divisible by 2, equivalently, if and only if $E(\mathbb{F}_q)$ contains a point of order 2. If E has Weierstrass equation $y^2 = f(x)$, the points of order 2 in $E(\mathbb{F}_q)$ are those of the form $(x_0, 0)$, where $x_0 \in \mathbb{F}_q$ is a root $f(x)$. So $t \equiv 0 \bmod 2$ if $f(x)$ has a root in $\mathbb{F}_q$, and $t \equiv 1 \bmod 2$ otherwise.

[1] There are deterministic p-adic algorithms for computing $\#E(\mathbb{F}_q)$ that are faster than Schoof's algorithm when the characteristic p of $\mathbb{F}_q$ is very small; see [2]. But their running times are exponential in $\log p$.

[2] There are faster ways to apply the Chinese remainder theorem; see [1, §10.3]. They are not needed here because the complexity is overwhelmingly dominated by step 2a.

The algorithm that we saw in Lecture 4 for finding roots of polynomials over finite fields is probabilistic, and we want to compute $t \bmod 2$ deterministically. But we don't actually need to *find* the roots of $f(x)$ in $\mathbb{F}_q$, we just need to determine whether any such roots exist. For this we only need the first step of the root-finding algorithm, which computes $\gcd(x^q - x, f(x))$. The degree of this gcd is equal to the number of distinct roots of f, so $t \equiv 0 \bmod 2$ if and only if the degree is positive. This is a deterministic computation, and since the degree of f is fixed at 3, it takes just $O(n\mathsf{M}(n))$ time.

This addresses the case $\ell = 2$; henceforth we assume that the prime ℓ is odd.

9.3 The characteristic equation of Frobenius modulo ℓ

Recall that the Frobenius endomorphism $\pi\,(x, y) = (x^q, y^q)$ satisfies the equation

$$\pi^2 - t\pi + q = 0$$

in the endomorphism ring $\mathrm{End}(E)$. If we restrict π to the ℓ-torsion subgroup $E\,[\ell]$, then

$$\pi_\ell^2 - t_\ell \pi_\ell + q_\ell = 0 \tag{1}$$

holds in $\mathrm{End}(E[\ell])$, where $t_\ell \equiv t \bmod \ell$ and $q_\ell \equiv q \bmod \ell$ may be viewed either as restrictions of the scalar multiplication maps $[t]$ and $[q]$, or simply as scalars in $\mathbb{Z}/\ell\mathbb{Z}$ multiplied by the restriction of the identity endomorphism $[1]_\ell$. We shall take the latter view, regarding q_ℓ as $q_\ell[1]_\ell$, the sum of q_ℓ copies of $[1]_\ell$ (which we can compute using double-and-add).

The key lemma we need is that equation (1) not only uniquely determines t_ℓ, it suffices to verify (1) at any nonzero point in $E[\ell]$.

Lemma 9.2. *Let $E/\mathbb{F}_q$ be an elliptic curve with Frobenius endomorphism π, let $t = \operatorname{tr}\pi$, let ℓ be a prime, and let P be a nonzero point in $E[\ell]$. Suppose that for some integer c the equation*

$$\pi_\ell^2(P) - c\pi_\ell(P) + q_\ell P = 0$$

holds. Then $c \equiv t \bmod \ell$.

Proof. From equation (1) we have

$$\pi_\ell^2(P) - t\pi_\ell(P) + q_\ell P = 0,$$

and we are given that

$$\pi_\ell^2(P) - c\pi_\ell(P) + q_\ell P = 0.$$

Subtracting these equations, we obtain $(c-t)\pi_\ell(P) = 0$. Since $\pi_\ell P$ is a nonzero element of $E[\ell]$ and ℓ is prime, the point $\pi_\ell P$ has order ℓ, which must divide $c - t$. So $c \equiv t \bmod \ell$. □

Let $h = \psi_\ell$ be the ℓ-th division polynomial of E; since ℓ is odd, we know that ψ_ℓ does not depend on the y-coordinate, and we have $h \in$
$Fq[x]$. As we proved in Lecture 6, a nonzero point $P = (x_0, y_0) \in E(\overline{\mathbb{F}}_q)$ lies in $E[\ell]$ if and only if $h(x_0) = 0$. Thus we may represent endomorphisms in $\mathrm{End}(E[\ell])$ as elements of the ring

$$\mathbb{F}_q\,[x, y]\,/(h(x), y^2 - f(x)),$$

where $y^2 = f(x) = x^3 + Ax + B$ is the Weierstrass equation for E.

In the case of the Frobenius endomorphism, we have

$$\begin{aligned}\pi_\ell(x,y) &= \big(x^q \bmod h(x),\ y^q \bmod (h(x), y^2 - f(x))\big) \\ &= \Big(x^q \bmod h(x), \big(f^{(q-1)/2} \bmod h(x)\big)y\Big), \end{aligned} \tag{2}$$

and similarly

$$\pi_\ell^2 = \Big(x^{q^2} \bmod h(x), \big(f^{(q^2-1)/2} \bmod h(x)\big)y\Big). \tag{3}$$

We also note that $[1]_\ell = (x \bmod h(x), \big(1 \bmod h(x)\big)y)$. Thus we can represent all of the nonzero endomorphisms that appear in equation (1) in the form $(a(x), b(x)y)$, where a and b are elements of the polynomial ring $R = \mathbb{F}_q[x]/(h(x))$.

Let us consider how to add and multiply elements of $\mathrm{End}(E[\ell])$ that are represented in this form. Multiplication is easy: the composition of $\big(a_1(x), b_1(x)y\big)$ with $\big(a_2(x), b_2(x)y\big)$ has the form $\big(a_3(x), b_3(x)y\big)$, where the polynomials

$$\begin{aligned} a_3(x) &= a_1(a_2(x)), \\ b_3(x) &= b_1(a_2(x))b_2(x) \end{aligned}$$

are again elements of R (so we reduce them modulo $h(x)$). We now address addition.

9.4 Adding endomorphisms in End($E[\ell]$).

Recall that addition of endomorphisms is defined in terms of the group law on the elliptic curve: $(\alpha_1 + \alpha_2)(P) = \alpha_1(P) + \alpha_2(P)$. Let us assume that α_1 and α_2 are represented in the form $\alpha_1 = \big(a_1(x), b_1(x)y\big)$ and $\alpha_2 = \big(a_2(x), b_2(x)y\big)$, where a_1, a_2, b_1, b_2 are reduced elements in the ring $R = \mathbb{F}_q[x]/(h(x))$. To compute $\alpha_3 = \alpha_1 + \alpha_2$, we simply apply the formulas for point addition. Let us first assume $\alpha_1 \neq \pm\alpha_2$; if $\alpha_1 = -\alpha_2$ then $\alpha_1 + \alpha_2 = 0$, and we will consider the case $\alpha_1 = \alpha_2$ below.

The general formula for computing $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ on the elliptic curve $E\colon y^2 = x^3 + Ax + B$ is

$$m = \frac{y_1 - y_2}{x_1 - x_2}, \qquad x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1.$$

Applying these to the coordinates $x_1 = a_1(x), x_2 = a_2(x), y_1 = b_1(x)y$, and $y_2 = b_2(x)y$, we compute

$$m(x,y) = \frac{b_1(x) - b_2(x)}{a_1(x) - a_2(x)}y = r(x)y,$$

where $r = (b_1 - b_2)/(a_1 - a_2)$. Using the curve equation $y^2 = f(x) = x^3 + Ax + B$ we have

$$\begin{aligned} a_3(x) &= m(x,y)^2 - a_1(x) - a_2(x) \\ &= r(x)^2 y^2 - a_1(x) - a_2(x) \\ &= r(x)^2 f(x) - a_1(x) - a_2(x), \\ b_3(x)y &= m(x,y)\big(a_1(x) - a_3(x)\big) - b_1(x)y \\ &= \big(r(x)(a_1(x) - a_3(x)) - b_1(x)\big)y, \end{aligned}$$

Thus $\alpha_3 = (a_3(x), b_3(x)y)$, where $a_3 = r^2 f - a_1 - a_2$ and $b_3 = r(a_1 - a_3) - b_1$.

The case $\alpha_1 = \alpha_2$ is similar, except that now we use

$$m(x,y) = \frac{3a_1(x)^2 + A}{2b_1(x)y} = \frac{3a_1(x)^2 + A}{2b_1(x)f(x)}y = r(x)y,$$

where $r = (3a_1^2 + A)/(2b_1 f)$.

In both cases, provided that the polynomial in the denominator of r is invertible in the ring $R = \mathbb{F}_q[x]/(h(x))$, we can reduce r to a polynomial modulo h and obtain the sum $\alpha_1 + \alpha_2 = \alpha_3 = \big(a_3(x), b_3(x)y\big)$ in our desired form, with $a_3, b_3 \in R$.

The case where the denominator of r is not invertible might appear to pose a problem, but it actually turns out to be good news, since if this occurs than we can obtain a proper factor g of h by computing g as the gcd of h with the denominator of r. The polynomial g must have positive degree (otherwise the denominator would be invertible), and we argue that its degree is also strictly less than the degree of h. This is clear when the denominator of r is $a_1 - a_2$, since both a_1 and a_2 are reduced modulo h and therefore have lower degree. If the denominator is $2b_1 f$, we note that f and h must be relatively prime, since roots of f correspond to points of order 2, while roots of h correspond to points of odd order ℓ; therefore g must be a factor of b_1, which is reduced modulo h.

Having found a proper factor g of h, our strategy is to replace h by g and restart the computation. Before explaining why this works, let us first give the algorithm.

9.5 Algorithm to compute t modulo ℓ

We now give an algorithm to compute t_ℓ, the trace of Frobenius modulo ℓ.

Algorithm 9.3. Given an elliptic curve $E/\mathbb{F}_q$ and an odd prime ℓ, compute t_ℓ as follows:

1. Compute the ℓth division polynomial $h = \psi_\ell \in \mathbb{F}_q[x]$ for E.
2. Compute π_ℓ and π_ℓ^2 via (2) and (3).
3. Use scalar multiplication to compute $q_\ell = q_\ell[1]_\ell$, and then compute $\pi_\ell^2 + q_\ell$.
   (If an uninvertible denominator arises, replace h by a proper factor and go to step 2).
4. Compute $0, \pi_l, 2\pi_l, 3\pi_l, \ldots, c\pi_\ell$, until $c\pi_l = \pi_l^2 + q_l$.
   (If an uninvertible denominator arises replace h by a proper factor and go to step 2).
5. Return $t_\ell = c$.

Throughout the algorithm, elements of $\mathrm{End}(E[\ell])$ are represented in the form $(a(x), b(x)y)$, with $a, b \in R = \mathbb{F}_q[x]/(h(x))$, and all polynomial operations take place in the ring R.

The correctness of the algorithm follows from equation (1) and Lemma 9.2. The algorithm is guaranteed to find $c\pi_l = \pi_l^2 + q_l$ in step 4 with $c < \ell$, since $c = t_\ell$ works, by (1). Although we may be working modulo a proper factor g of h, every root x_0 of g is a root of h and therefore corresponds to a pair of nonzero points $P = (x_0, \pm y_0) \in E[\ell]$ for which $\pi_\ell^2(P) - c\pi_\ell(P) + q_\ell P = 0$ holds (there is at least one such root, since g has positive degree), and by Lemma 9.2, we must have $c = t_\ell$.

An implementation of Schoof's algorithm can be found in the Sage worksheet

https://hensel.mit.edu:8002/home/pub/5/

9.6 Finding factors of h

As we saw when running our implementation of Schoof's algorithm in Sage, we do occasionally find a proper factor of the ℓth division polynomial $h(x)$. This is not too surprising, since there is no reason why h should be irreducible; indeed, if it happens that $E[\ell] = E[\ell](\mathbb{F}_q)$ then h must split into linear factors. But it is worth noting that when the algorithm finds a factor g of h, the polynomial g always has degree $(\ell-1)/2$. Why is this the case?

Any point $P = (x_0, y_0)$ for which $g(x_0) = 0$ lies both in $E[\ell]$ and in the kernel of some endomorphism α (since x_0 is a root of the denominator of a rational function defining α). The point P is nonzero, so it generates a cyclic group C of order ℓ, and C must lie in the kernel of α (since $\alpha(mP) = m\alpha(P) = 0$ for all m). It follows that g must have at least $(\ell-1)/2$ roots, one for each pair of nonzero points $(x_i, \pm y_i)$ in C (note that ℓ is odd). On the other hand, if g has any other roots, then there is another point Q that lies in the intersection of $E[\ell]$ and $\ker\alpha$, and then we must have $\ker\alpha = E[\ell]$, since $E[\ell]$ has ℓ-rank 2. But this would imply that every root of h is a root of g, which is not the case, since g is a proper divisor of h and all of h's roots are distinct. So g has exactly $(\ell-1)/2$ roots. Reducing the polynomials that define our endomorphism modulo g corresponds to working in the subring $\mathrm{End}(C)$ of $\mathrm{End}(E[\ell])$.

Once we have found such a g, note that the algorithm speeds up by a factor of ℓ, since we are working with polynomials of lower degree. While we are unlikely to stumble across such a g by chance once ℓ is large, it turns out that in fact such a g does exist for approximately half of the primes ℓ. Not long after Schoof published his result, Noam Elkies found a way to systematically find such factors g of h as polynomials representing the kernels of isogenies, which allows one to speed up Schoof's algorithm quite dramatically. We will learn about Elkies' technique later in the course when we cover modular polynomials.

9.7 Some historical remarks

As a historical footnote, when Schoof originally developed this algorithm, it was not clear to him that it had any practical use. This is in part because he (and others) were unduly pessimistic about its practical efficiency (efficient computer algebra implementations were not widely available). Even the simple Sage implementation given in the worksheet is already noticeably faster than baby-steps giant-steps for $q \approx 2^{80}$ and can readibly handle computations over fields of cryptographic size (it would likely take about half a day for $q \approx 2^{256}$, but this could be substantially improved with a lower-level implementation).

To better motivate his algorithm, Schoof gave an application that is of purely theoretical interest: he showed that it could be used to deterministically compute the square root of an integer x modulo a prime p in time that is polynomial in $\log p$, for a fixed value of x (we will see exactly how this works when we cover the theory of complex multiplication). Previously, no deterministic polynomial time algorithm was known for this problem, unless one assumes the extended Riemann hypothesis. But Schoof's square-root application is really of no practical use; there are fast probabilistic algorithms to compute square roots modulo a prime, and unless the extended Riemann hypothesis is false, there are even deterministic algorithms that are much faster than Schoof's approach.

By contrast, in showing how to compute $\#E(\mathbb{F}_q)$ in polynomial-time, Schoof solved a practically important problem for which the best previously known algorithms were fully exponential (including probabilistic approaches), despite the efforts of many working in the field. While perhaps not fully appreciated at the time, this has to be regarded as a major

breakthrough, both from a theoretical and practical perspective. Improved versions of Schoof's algorithm (the Schoof-Elkies-Atkin or SEA algorithm) are now the method of choice for computing $\#E(\mathbb{F}_q)$ in fields of large characteristic and are widely used. In particular, the PARI library that is used by Sage for point-counting includes an implementation of the SEA algorithm.

References

[1] Joachim von zur Gathen and Jürgen Gerhard, *Modern Computer Algebra*, second edition, Cambridge University Press, 2003.

[2] Takakazu Satoh, *On p-adic point counting algorithms for elliptic curves over finite fields*, ANTS V, LNCS **2369** (2002), 43–66.

[3] René Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*. Mathematics of Computation **44** (1985), 483–495.