9 Schoof's algorithm

In the early 1980's, René Schoof [3, 4] 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 simple: compute the the trace of Frobenius t modulo many small primes ℓ and use the Chinese remainder theorem to uniquely determine t, which then determines $\#E(\mathbb{F}_q) = q + 1 - t$. Here is a high-level version of the algorithm.

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

```
1. Initialize $M \leftarrow 1$ and $t \leftarrow 0$.
2. While $M \le 4\sqrt{q}$, for increasing primes $\ell = 2, 3, 5, \ldots$ that do not divide $q$:
    a. Compute $t_\ell = \operatorname{tr}\pi \bmod \ell$.
    b. Set $t \leftarrow \bigl(M(M^{-1} \bmod \ell)t_\ell + \ell(\ell^{-1} \bmod M)t\bigr) \bmod \ell M$ and then $M \leftarrow \ell M$.
3. If $t > M/2$ then set $t \leftarrow t - M$.
4. Output $q + 1 - t$.
```

Step 2b uses an iterative version of the Chinese remainder theorem to ensure that

$$t \equiv \operatorname{tr}\pi_E \bmod M$$

holds throughout.[2] This invariant holds trivially after step 1, modulo $M = 1$, and is maintained in step 2b: note that the integer $M(M^{-1} \bmod \ell)$ is congruent to $1 \bmod \ell$ and $0 \bmod M$, while the integer $\ell(\ell^{-1} \bmod M)$ is congruent to $0 \bmod \ell$ and $1 \bmod M$.

Once M exceeds $4\sqrt{q}$, the value of $t \in \mathbb{Z}/M\mathbb{Z}$ uniquely determines $\operatorname{tr}\pi_E \in \mathbb{Z}$: by Hasse's theorem, $|\operatorname{tr}\pi_E| \le 2\sqrt{q} < M/2$, and this allows us to determine the sign of $\operatorname{tr}\pi_E$ in step 3. The key to the algorithm is the implementation of step 2a, which is described in the next section, but 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[3]

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

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

9.1 Computing the trace of Frobenius modulo 2.

We first consider the case $\ell = 2$. Assuming q is odd (which we do), $t = q + 1 - \#E(\mathbb{F}_q)$ 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 $y^2 = f(x)$, then the points of

[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 relevant here because the complexity is overwhelmingly dominated by step 2a.

[3]In fact we only need Chebyshev's Theorem to get this.

order 2 in $E(\mathbb{F}_q)$ are precisely those of the form $(x_0, 0)$, where $x_0 \in \mathbb{F}_q$ is a root $f(x)$. Recall from Lecture 4 that the distinct roots of f in $\mathbb{F}_q$ are precisely the roots of $\gcd(x^q - x, f(x))$. We can thus compute $t_2 \coloneqq \operatorname{tr} \pi_E \bmod 2$ as

$$t_2 = \begin{cases} 0 & \text{if } \deg\bigl(\gcd(f(x), x^q - x)\bigr) > 0; \\ 1 & \text{otherwise.} \end{cases}$$

Note that is a deterministic computation (we need randomness to efficiently *find* the roots of $g(x)$, but not to *count* them), and it takes $O(n\,\mathsf{M}(n))$ time.

Having addressed the case $\ell = 2$ we henceforth assume that ℓ is odd.

9.2 The characteristic equation of Frobenius modulo ℓ

Recall that for $E/\mathbb{F}_q$, the Frobenius endomorphism $\pi_E \in \operatorname{End}(E)$ is defined by the rational map $(x : y : z) \mapsto (x^q : y^q : z^q)$. By Theorem 7.18, it satisfies the characteristic equation

$$\pi_E^2 - t\pi_E + q = 0,$$

with $t = \operatorname{tr} \pi$ and $q = \deg \pi$. Restricting to the ℓ-torsion subgroup $E\,[\ell]$ yields

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

which we view as an identity in $\operatorname{End}(E[\ell])$. Here $t_\ell \equiv t \bmod \ell$ and $q_\ell \equiv q \bmod \ell$ can 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 $[1]_\ell$, the restriction of $[1] \in \operatorname{End}(E)$ to $E[\ell]$ (equivalently the multiplicative identity in the ring $\operatorname{End}(E[\ell])$). We shall take the latter view, regarding

$$q_\ell = q_\ell \cdot [1]_\ell = [1]_\ell + \cdots + [1]_\ell$$

as the sum of q_ℓ copies of $[1]_\ell$, and similarly for t_ℓ. We can efficiently compute q_ℓ using our usual double-and-add method to perform scalar multiplication by q_ℓ, provided that we know how to explicitly represent and perform ring operations on elements of $\operatorname{End}(E[\ell])$; this is the topic of the next section.

Our strategy for determining t_ℓ is simple: for $c = 0, 1, \ldots, \ell - 1$ compute $\pi_\ell^2 - c\pi_\ell + q_\ell$ and check whether it is equal to 0. The following lemma shows that whenever this occurs (which it must, since (1) guarantees this for $c = t_\ell$) we must have $c = t_\ell \in \mathbb{Z}/\ell\mathbb{Z}$. In fact we will prove something stronger.

Lemma 9.2. *Let $E/\mathbb{F}_q$ be an elliptic curve with Frobenius endomorphism π, let ℓ be a prime not dividing q, and let $P \in E[\ell]$ be nonzero. Suppose that for some integer c the equation*

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

holds. Then $c \equiv t_\ell = \operatorname{tr} \pi \bmod \ell$.

Proof. From equation (1) we have

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

and we are assuming that

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

Subtracting these equations yields $(c - t_\ell)\pi_\ell(P) = 0$. 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 $c - t_\ell$. So $c \equiv t_\ell \bmod \ell$. $\square$

9.3 Arithmetic in End($E[\ell]$)

Let $h = \psi_\ell(x, y)$ be the ℓth division polynomial of E. We have assumed that ℓ is odd, so by Lemma 6.20, we in fact have $h \in \mathbb{F}_q[x]$ (no dependence on y). 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 $h(x_0) = 0$; this follows from Corollary 5.27 and Theorem 6.21. To represent elements of End($E[\ell]$) as rational maps, we can thus treat the polynomials appearing in these maps 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

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

and we also note that

$$[1]_\ell = (x \bmod h(x),\ \ (1 \bmod h(x))\, y).$$

We can thus 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 $R = \mathbb{F}_q[x]/(h(x))$ that we may uniquely represent as polynomials in $\mathbb{F}_q[x]$ of degree less than $\deg h = (\ell^2 - 1)/2$ by taking their remainders modulo h.

9.3.1 Multiplication in End($E[\ell]$).

We multiply endomorphisms by composition. If $\alpha_1 = (a_1(x), b_1(x)y)$ and $\alpha_2 = (a_2(x), b_2(x)y)$ are two elements of End($E[\ell]$), then the product $\alpha_1\alpha_2$ in End($E[\ell]$) is given by

$$\alpha_1 \circ \alpha_2 = (a_1(a_2(x)),\ \ b_1(a_2(x))b_2(x)\,y)\,,$$

where we may reduce $a_3(x) = a_1(a_2(x))$ and $b_3(x) = b_1(a_2(x))b_2(x)$ modulo $h(x)$.

9.3.2 Addition in End($E[\ell]$).

Addition of endomorphisms is defined pointwise in terms of addition on the elliptic curve. Given $\alpha_1 = (a_1(x), b_1(x)y)$ and $\alpha_2 = (a_2(x), b_2(x)y)$, to compute $\alpha_3 = \alpha_1 + \alpha_2$, we simply apply the formulas for point addition to the coordinate functions of α_1 and α_2. Recall that the general formula for addition of non-opposite affine points $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ on the elliptic curve $E\colon y^2 = x^3 + Ax + B$ is given by the formulas

$$x_3 = m^2 - x_1 - x_2, \qquad y_3 = m(x_1 - x_3) - y_1,$$

where

$$m = \frac{y_1 - y_2}{x_1 - x_2} \quad (\text{if } x_1 \neq x_2), \qquad\qquad m = \frac{3x_1^2 + A}{2y_1} \quad (\text{if } x_1 = x_2).$$

Using the coordinate functions $x_1 = a_1(x)$, $x_2 = a_2(x)$, $y_1 = b_1(x)y$, $y_2 = b_2(x)y$, in the case $x_1 \neq x_2$ we have

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

where $r = (b_1 - b_2)/(a_1 - a_2)$, and when $x_1 = x_2$ we have

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

where now $r = (3a_1^2 + A)/(2b_1 f)$. Noting that $m(x,y)^2 = (r(x)y)^2 = r(x)^2 f(x)$, the sum $\alpha_1 + \alpha_2 = \alpha_3 = (a_3(x),\ b_3(x)y)$ is defined by

$$a_3 = r^2 f - a_1 - a_2,$$
$$b_3 = r(a_1 - a_3) - b_1.$$

In both cases, provided that the polynomial v in the denominator of the rational function $r = u/v$ is invertible in the ring $\mathbb{F}_q[x]/(h(x))$, we can express r as a polynomial $uv^{-1} \bmod h$ and write $\alpha_3 = (a_3(x), b_3(x)y)$ in our desired form, with $a_3, b_3 \in \mathbb{F}_q[x]/(h(x))$ uniquely represented by polynomials in $\mathbb{F}_q[x]$ of degree less than the degree of h.

But this may not always possible, because the ℓ-division polynomial $h(x)$ need not be irreducible. Indeed, if ℓ divides $\#E(\mathbb{F}_q)$ it certainly will not be irreducible, since $h(x)$ will then have rational roots corresponding to the x-coordinates of rational points of order ℓ, and even when $\ell \nmid \#E(\mathbb{F}_q)$, if E admits a rational isogeny α of degree ℓ then $h(x)$ will be divisible by the polynomial of degree $(\ell-1)/2$ whose roots are the x-coordinates of the nonzero points in the kernel of α. Thus the the ring $\mathbb{F}_q[x]/(h(x))$ is not necessarily a field; it may contain zero divisors, and these elements are not invertible.

At first glance this might appear to be a problem, but in fact it can only help us. If we encounter a rational function $r = u/v$ whose denominator v is not invertible in $\mathbb{F}_q[x]/(h(x))$ then we can obtain a non-trivial factor of h by computing $\gcd(v,h)$: if $v = a_1 - a_2$ then v is nonzero and has degree less than h, since in this case $a_1 \neq a_2$ and $\deg(a_1 - a_2) < \deg(h)$, and if $v = 2b_1 f$ then $\gcd(v,h)$ must divide b_1, because h and f cannot share a common factor (the roots of $f(x)$ in $\overline{\mathbb{F}}_q$ are x-coordinates of 2-torsion points, the roots of $h(x)$ in $\overline{\mathbb{F}}_q$ are x-coordinates of ℓ-torsion points, and $\ell \neq 2$), and $b_1 \neq 0$ has degree less than h.

Our strategy in this situation is to simply replace h by $g = \gcd(v,h)$ and compute t_ℓ by working in the smaller quotient ring $\mathbb{F}_q[x]/(g(x))$, which will be faster because $\deg g < \deg h$; in fact in this situation we will always have $\deg g \leq (\ell-1)/2$, which is much smaller than $\deg h = (\ell^2-1)/2$. Lemma 9.2 implies that we can restrict our attention to the action of π_ℓ on points $P \in E[\ell]$ whose x-coordinates are roots of $g(x)$, even if $\deg g = 1$.

9.4 Algorithm to compute the trace of Frobenius modulo ℓ

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

Algorithm 9.3. Given $E: y^2 = f(x)$ over $\mathbb{F}_q$ 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 $\pi_\ell = (x^q \bmod h,\ (f^{(q-1)/2} \bmod h)y)$ and $\pi_\ell^2 = \pi_\ell \circ \pi_\ell$.
3. Use scalar multiplication to compute $q_\ell = q_\ell[1]_\ell$, and then compute $\pi_\ell^2 + q_\ell$.
   (If a non-invertible denominator arises, update h and return 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 a non-invertible denominator arises, update h and return to step 2).
5. Output $t_\ell = c$.

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. If a non-invertible denominator v is found in either steps 3 or 4 we replace h with whichever of $\gcd(h, v)$ and $h/\gcd(h, v)$ has lower degree; this guarantees that the degree of h is reduced by at least a factor of 2 (but see the next section for a further discussion).

The correctness of the algorithm follows from equation (1) and Lemma 9.2. The algorithm is guaranteed to find some $c\pi_l = \pi_l^2 + q_l$ in step 4 with $c < \ell$, since we know that $c = t_\ell$ works. 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 $\deg g > 0$), and Lemma 9.2 implies that we must have $c = t_\ell$.

The computation of the division polynomial in step 1 of the algorithm can be efficiently accomplished using the double-and-add approach described in Problem Set 3. You will have the opportunity to do a careful complexity analysis Algorithm 9.3 in the next problem set, but it is easy to see that its running time is polynomial in $n = \log q$ and ℓ: every operation involves polynomials over $\mathbb{F}_q$ of degree less then ℓ^2, in step 4 we can have at most ℓ iterations, and we can return to step 2 at most $2\log \ell$ times (in fact this can happen only once). A simple implementation of the algorithm can be found in this Sage worksheet.

9.5 Factors of the division polynomial

As we saw when running our implementation of Schoof's algorithm in Sage, we do occasionally encounter non-invertible denominators and thereby obtain a proper factor g of the ℓ-division polynomial $h = \psi_\ell$. This is not too surprising, since there is no reason why h should necessarily be irreducible, but it is worth noting that whenever this occurs the degree of g is always exactly $(\ell - 1)/2$. Why is this the case?

Any point $P = (x_0, y_0) \in E(\overline{\mathbb{F}}_q)$ for which $g(x_0) = 0$ lies both in $E[\ell]$ and in the kernel of an 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 which must be a subgroup of $\ker \alpha$. It follows that over $\overline{\mathbb{F}}_q$ the polynomial g has 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). If g has any other roots, then there is point Q that lies in the intersection of $E[\ell] \cap \ker \alpha$ but not in C, in which case we must have $\ker \alpha = E[\ell]$, since $E[\ell]$ has ℓ-rank 2; but this is impossible because g is a proper factor of the ℓ-division polynomial h (whose roots are distinct because $\ell \nmid q$). So g must have exactly $(\ell - 1)/2$ roots in $\overline{\mathbb{F}}_q$. Reducing the polynomials that define our endomorphism modulo g corresponds to working in the subring $\mathrm{End}(C)$ of $\mathrm{End}(E[\ell])$.

If we are lucky enough to find such a proper factor g of h, our algorithm then speeds up by at least a factor of ℓ, since we are working modulo a polynomial of degree $(\ell - 1)/2$ rather than $(\ell^2 - 1)/2$. While we are fairly unlikely to stumble across such a g by chance, it turns out that in fact such a g exists for half of the primes ℓ (asymptotically speaking). Not long after Schoof published his result, Noam Elkies found a way to directly compute these polynomials, whose roots are the x-coordinates of points $P = (x_0, y_0)$ that lie in the kernel of a rational isogeny of degree ℓ. We will learn about Elkies' technique later in the course when we discuss modular polynomials. There is another optimization due to A.O.L. Atkin that applies to primes ℓ for which Elkies' optimization does not; together these yield what is known as the Schoof-Elkies-Atkin (SEA) algorithm.

9.6 Some historical remarks

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, in part because robust implementations of fast integer and polynomial arithmetic were not as widely available then as they are now. Even the simple Sage implementation given in the worksheet is already noticeably faster than the baby-steps giant-steps algorithm for $q \approx 2^{80}$ and can readily handle computations over fields of cryptographic size (it might take a day or two for $q \approx 2^{256}$, but this could be improved by at least an order of magnitude using a lower-level implementation in C or C++).

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 a modulo a prime p in time that grows polynomially in $\log p$ when a is held fixed; 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; as we have seen, 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 randomized algorithms), despite the efforts of many experts 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 SEA algorithm) are now the method of choice for computing $\#E(\mathbb{F}_q)$ in fields of large characteristic. In particular, the PARI library that is used by Sage includes an implementation of the SEA algorithm, and over 256-bit fields it takes only a few seconds to compute $\#E(\mathbb{F}_q)$. Today it is feasible to compute $\#E(\mathbb{F}_q)$ even when q is a prime with 5,000 decimal digits (over 16,000 bits), which represents the current record [5].

9.7 The discrete logarithm problem

We now turn to a problem that is generally believed *not* to have a polynomial-time solution.[4] In its most standard form, the *discrete logarithm problem* in a finite group G can be stated as follows:

Given $\alpha \in G$ and $\beta \in \langle\alpha\rangle$, find the least positive integer x such that $\alpha^x = \beta$.

In additive notation (which we will often use), this means $x\alpha = \beta$. In either case, we call x the discrete logarithm of β with respect to the base α and denote it $\log_\alpha \beta$.[5] Note that in the form stated above, where x is required to be positive, the discrete logarithm problem includes the problem of computing the order of α as a special case: $|\alpha| = \log_\alpha 1_G$.

We can also formulate a slightly stronger version of the problem:

Given $\alpha, \beta \in G$, compute $\log_\alpha \beta$ if $\beta \in \langle\alpha\rangle$ and otherwise report that $\beta \notin \langle\alpha\rangle$.

[4]Here we restrict our attention to classical models of computation (e.g. Turing machines). Under a quantum computing model, polynomial-time algorithms for the discrete logarithm problem are known.

[5]The multiplicative terminology stems from the fast that most of the early work on computing discrete logarithms focused on the case where G is the multiplicative group of a finite field.

This can be a significantly harder problem. For example, if we are using a Las Vegas algorithm, when β lies in $\langle\alpha\rangle$ we are guaranteed to eventually find $\log_\alpha \beta$, but if not, we will never find it and it may be impossible to tell whether we are just very unlucky or $\beta \notin \langle\alpha\rangle$. On the other hand, with a deterministic algorithm such as the baby-steps giant-steps method, we can unequivocally determine whether β lies in $\langle\alpha\rangle$ or not.

There is also a generalization called the *extended discrete logarithm*:

> *Given $\alpha, \beta \in G$, determine the least positive integer y such that $\beta^y \in \langle\alpha\rangle$, and then output the pair (x, y), where $x = \log_\alpha \beta^y$.*

This yields positive integers x and y satisfying $\beta^y = \alpha^x$, where we minimize y first and x second. Note that there is always a solution: in the worst case $x = |\alpha|$ and $y = |\beta|$.

Finally, one can also consider a vector form of the discrete logarithm problem:

> *Given $\alpha_1, \ldots \alpha_r \in G$ and $n_1, \ldots, n_r \in \mathbb{Z}$ such that every $\beta \in G$ can be written uniquely as $\beta = \alpha_1^{e_1} \cdots \alpha_r^{e_r}$ with $e_i \in [1, n_i]$, compute the exponent vector $(e_1, \ldots, e_r)$ associated to a given β.*

Note that the group G need not be abelian in order for the hypothesis to apply, it suffices for G to by *polycyclic* (this means it admits a subnormal series with cyclic quotients).

The extended discrete and vector forms of the discrete logarithm problem play an important role in algorithms to compute the structure of a finite abelian group, but in the lectures we will focus primarily on the standard form of the discrete logarithm problem (which we may abbreviate to DLP).

Example 9.4. Suppose $G = \mathbb{F}_{101}^\times$. Then $\log_3 37 = 24$, since $3^{24} \equiv 37 \bmod 101$.

Example 9.5. Suppose $G = \mathbb{F}_{101}^+$. Then $\log_3 37 = 46$, since $46 \cdot 3 \equiv 37 \bmod 101$.

Both of these examples involve groups where the discrete logarithm is easy to compute (and not just because 101 is a small number), but for very different reasons. In Example 9.4 we are working in a group of order $100 = 2^2 \cdot 5^2$. As we will see in the next lecture, when the group order is a product of small primes (i.e. *smooth*), it is easy to compute discrete logarithms. In Example 9.5 we are working in a group of order 101, which is prime, and in terms of the group structure, this represents the hardest case. But in fact it is very easy to compute discrete logarithms in the additive group of a finite field! All we need to do is compute the multiplicative inverse of 3 modulo 101 (which is 34) and multiply by 37. This is a small example, but even if the field size is very large, we can use the extended Euclidean algorithm to compute multiplicative inverses in quasi-linear time.

So while the DLP is generally considered a "hard problem", its difficulty really depends not on the order of the group (or its structure), but on how the group is explicitly represented. Every group of prime order p is isomorphic to $\mathbb{Z}/p\mathbb{Z}$; computing the discrete logarithm amounts to computing this isomorphism. The reason it is easy to compute discrete logarithms in $\mathbb{Z}/p\mathbb{Z}$ has nothing to do with the structure of $\mathbb{Z}/p\mathbb{Z}$ as an additive group, rather it is the fact that $\mathbb{Z}/p\mathbb{Z}$ also use a ring structure; in particular, it is a Euclidean domain, which allows us to use the extended Euclidean algorithm to compute multiplicative inverses. This involves operations (multiplication) other than the standard group operation (addition), which is in some sense "cheating".

Even when working in the multiplicative group of a finite field, where the DLP is believed to be much harder, we can do substantially better than in a generic group. As we shall see, there are sub-exponential time algorithms for this problem, whereas in the generic setting defined below, only exponential time algorithms exist, as we will prove in the next lecture.

9.8 Generic group algorithms

In order to formalize the notion of “not cheating", we define a *generic group algorithm* (or just a *generic algorithm*) to be one that interacts with an abstract group G solely through a *black box* (sometimes called an *oracle*). All group elements are opaquely encoded as bit-strings via a map $\mathrm{id}\colon G \to \{0,1\}^m$ chosen by the black box. The black box supports the following operations.

1. `identity`: output $\mathrm{id}(1_G)$.
2. `inverse`: given input $\mathrm{id}(\alpha)$, output $\mathrm{id}(\alpha^{-1})$.
3. `composition`: given inputs $\mathrm{id}(\alpha)$ and $\mathrm{id}(\beta)$, output $\mathrm{id}(\alpha\beta)$.
4. `random`: output $\mathrm{id}(\alpha)$ for a uniformly distributed random element $\alpha \in G$.

In the description above we used multiplicative notation; in additive notation we would have outputs $\mathrm{id}(0_G)$, $\mathrm{id}(-\alpha)$, $\mathrm{id}(\alpha-\beta)$ for the operations `identity`, `inverse`, `composition`, respectively.

Some models for generic group algorithms also include a black box operation for testing equality of group elements, but we will instead assume that group elements are *uniquely identified*; this means that the identification map $\mathrm{id}\colon G \to \{0,1\}^m$ used by the black box is injective. With uniquely identified group elements we can test equality by simply comparing identifiers, without needing to consult the black box.[6]

The black box is allowed to use *any* injective identification map (e.g., a random one). A *generic algorithm* cannot depend on a particular choice of the identification map; this prevents it from taking advantage of how group elements are represented. We have already seen several examples of generic group algorithms, including various exponentiation algorithms, fast order algorithms, and the baby-steps giant-steps method.

We measure the time complexity of a generic group algorithm by counting *group operations*, the number of interactions with the black box. This metric has the virtue of being independent of the actual software and hardware implementation, allowing one to make comparisons the remain valid even as technology improves. But if we want to get a complete measure of the complexity of solving a problem in a particular group, we need to multiply the group operation count by the bit-complexity of each group operation, which of course depends on the black box. To measure the space complexity, we count the total number of group identifiers stored at any one time (i.e. the maximum number of group identifiers the algorithm ever has to remember).

These complexity metrics do not account for any other work done by the algorithm. If the algorithm wants to compute a trillion digits of pi, or factor some huge integer, it can effectively do so “for free". But the implicit assumption is that the cost of any auxiliary computation is at worst proportional to the number of group operations — this is true of all the algorithms we will consider.

References

[1] Joachim von zur Gathen and Jürgen Garhard, *Modern computer algebra*, third edition, Cambridge University Press, 2013.

[6] We can also sort bit-strings or index them with a hash table or other data structure; this is essential to an efficient implementation of the baby-steps giant-steps algorithm.

[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.

[4] René Schoof, *Counting points on elliptic curves over finite fields*, Journal de Théorie des Nombres de Bordeaux **7** (1995), 219–254.

[5] Andrew V. Sutherland, *On the evaluation of modular polynomials*, in Proceedings of the Tenth Algorithmic Number Theory Symposium (ANTS X), Open Book Series **1**, Mathematical Science Publishers, 2013, 531–555.