Chapter XI

Algorithmic Aspects of Elliptic Curves

The burgeoning field of computational number theory asks for practical algorithms to compute solutions to arithmetic problems. For example, the Mordell–Weil theorem (VIII.6.7) says that the group of rational points on an elliptic curve is finitely generated, and although we still lack an effective algorithm that is guaranteed to find a set of generators, there are algorithms that often work well in practice. Similarly, Siegel's theorem (IX.3.2.1) says that an elliptic curve has only finitely many S-integral points, but it took 50 years from Siegel's proof of finiteness to Baker's theorem giving an effective bound for the height of the largest solution (IX §5). And Baker's theorem is only the beginning of the story, since it leads to estimates that, although effective, are not practical without the introduction of significant additional ideas.

A full introduction to the computational theory of elliptic curves would require (at least) a book of its own, so in this single chapter we touch on only a few of the many algorithms in the theory. We decided to concentrate on aspects that are especially useful for applications to cryptography, not because these are intrinsically more interesting than other computational problems, but because they form a satisfying whole and because they tie in with many of the other topics covered in this book.

The theme of this chapter is thus that of computations on elliptic curves over (large) finite fields. We describe fast algorithms for computing multiples of points, for determining the number of points in $E(\mathbb{F}_q)$, and for computing the Weil pairing. We briefly survey some cryptographic constructions based on the difficulty of solving the elliptic curve discrete logarithm (ECDLP), and we describe algorithms to solve the ECDLP. We explain how elliptic curves can be used to factor large numbers. In the final section we define and analyze the Tate–Lichtenbaum pairing, which is frequently used in cryptography because it is easier to compute than the Weil pairing.

Lack of space precludes our covering computational problems over global fields, although these are also extremely interesting. In particular, we do not cover

J.H. Silverman, *The Arithmetic of Elliptic Curves, Second Edition*, Graduate Texts in Mathematics 106, DOI 10.1007/978-0-387-09494-6_XI,

algorithms related to modular aspects of elliptic curves, including in particular the computation of L-series, nor do we discuss more advanced algorithmic methods for computing Mordell–Weil groups or for finding integer points on elliptic curves. We do not discuss methods used to find curves of high rank over $\mathbb{Q}$, nor how to find precise estimates for $|\hat{h} - h|$. For an introduction to these and other algorithmic topics, see for example [50, 54, 55, 58, 67, 76, 86, 171, 188, 219, 265, 315].

There are two other topics that would fit naturally into this chapter, but were omitted due to lack of space. The first is efficient implementation of elliptic curve addition, which includes issues of affine versus projective coordinates and the choice of different sorts of equations to minimize the number of field additions, multiplications, and inversions. See for example [16], [22, IV.1], [51, §§13.2, 13.3], or [71]. The second topic is the use of elliptic curves to prove that a number is prime; see [10], [22, §IX.3], [50, §9.2], [51, §25.2.2], or [97].

Finally, while on the topic of elliptic curve algorithms, we mention the free computer packages Pari [202] and Sage [275], both of which contain extensive libraries of algorithms for doing computations on elliptic curves. In particular, Sage includes Cremona's mwrank package, which (attempts to) compute the Mordell–Weil group of elliptic curves over $\mathbb{Q}$. There are also extensive online tables of elliptic curves of various types, e.g., of small conductor, and of modular forms associated to elliptic curves. See for example [53] and [274].

XI.1 Double-and-Add Algorithms

Let E/K be an elliptic curve and let $P \in E(K)$ be a point on E. Suppose that we need to compute $[n]P$ for some large value of n. An obvious way to do this is to compute successively

$$P, \quad [2]P = P + P, \quad [3]P = [2]P + P, \quad \dots \quad [n]P = [n-1]P + P.$$

This naive algorithm takes $n - 1$ steps, where a "step" consists of adding two points.

If n is large, the naive algorithm is completely useless. All practical applications of elliptic curves over large finite fields rely on the following exponential improvement.

Double-and-Add Algorithm 1.1. *Let E/K be an elliptic curve, let $P \in E(K)$, and let $n \ge 2$ be an integer. The algorithm described in Figure 11.1 computes $[n]P$ using no more than $\log_2(n)$ point doublings and no more than $\log_2(n)$ point additions.*

PROOF. During the i^{th} iteration of the loop, the value of Q is $[2^i]P$. Since R is incremented by Q if and only if $\epsilon_i = 1$, the final value of R is

$$\sum_{i \text{ with } \epsilon_i = 1} [2^i]P = \sum_{i=0}^{t} [\epsilon_i 2^i]P = \left[\sum_{i=0}^{i} \epsilon_i 2^i\right] P = [n]P.$$

Each iteration of the loop requires one point duplication and at most one point addition, and since $t \le \log_2 n$, the running time of the algorithm is as stated. □

(1) Write the binary expansion of n as

$$n = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \epsilon_3 \cdot 2^3 + \cdots + \epsilon_t \cdot 2^t$$
$$\text{with } \epsilon_0, \ldots, \epsilon_t \in \{0, 1\} \text{ and } \epsilon_t = 1.$$

(2) Set $Q = P$ and $R = \begin{cases} O & \text{if } \epsilon_0 = 0, \\ P & \text{if } \epsilon_0 = 1. \end{cases}$
(3) Loop $i = 1, 2, \ldots, t$.
(4) Set $Q = [2]Q$.
(5) If $\epsilon_i = 1$, set $R = R + Q$.
(6) End Loop
(7) Return R, which is equal to $[n]P$.

Figure 11.1: The double-and-add algorithm.

Remark 1.2. The double-and-add algorithm (XI.1.1) is not unique to elliptic curves; it is applicable to any group. Thus if G is a group and $g \in G$, we use the binary expansion $n = \sum \epsilon_i 2^i$ to compute g^n as $g^n = \prod (g^{2^i})^{\epsilon_i}$. This requires at most $\log_2 n$ group squarings and at most $\log_2 n$ group multiplications. When the group law in G is written multiplicatively, for example for $G = \mathbb{F}_q^*$, the double-and-add algorithm is instead called the *square-and-multiply* algorithm.

Remark 1.3. The double-and-add algorithm is most often applied to a finite group such as $E(\mathbb{F}_q)$ or $\mathbb{F}_q^*$, rather than to an infinite group such as $E(\mathbb{Q})$. To see why, note that if $P \in E(\mathbb{Q})$, then the theory of canonical heights (VIII §9) says that it takes $O(n^2)$ bits to write down the coordinates of $[n]P$. Thus it is not feasible to compute $[n]P$ for, say, $n > 2^{80}$. On the other hand, the double-and-add algorithm allows us to easily compute $[n]P$ in $E(\mathbb{F}_q)$ when, say, q and n are as large as 2^{1000}. Of course, when we say that the computation is easy, we mean on a computer, not with paper and pencil!

Remark 1.4. The average running time of the double-and-add algorithm to compute $[n]P$ is $\log_2 n$ doublings and $\frac{1}{2}\log_2 n$ additions, since the binary expansion of a random integer n has an equal number of 1's and 0's. We can reduce the average running time by using a *ternary expansion of* n,

$$n = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \epsilon_3 \cdot 2^3 + \cdots + \epsilon_t \cdot 2^t$$
$$\text{with } \epsilon_1, \ldots, \epsilon_t \in \{-1, 0, 1\} \text{ and } \epsilon_t = \pm 1.$$

The only changes in (XI.1.1) are in step (2), where we set $R = -P$ if $\epsilon_0 = -1$, and in step (5), where we set $R = R \pm Q$ if $\epsilon_i = \pm 1$. It is not hard to show that every integer has a unique ternary expansion in which no two consecutive coefficients are nonzero; see Exercise 11.2.

There are two complementary reasons why ternary expansions are advantageous for computing $[n]P$ in $E(\mathbb{F}_q)$. First, ternary expansions tend to have significantly

fewer nonzero ϵ_i, so the number of point additions is reduced. Second, and equally important, the negation operation in $E(\mathbb{F}_q)$ is computationally trivial, so subtraction is no more difficult than addition. This is in marked contrast to $\mathbb{F}_q^*$, where group negation (inversion) is much slower than group addition (multiplication).

Remark 1.5. Koblitz has suggested using the Frobenius map to further speed the computation of $[n]P$. The idea is to use an elliptic curve $E/\mathbb{F}_p$ with p small and to take a point $P \in E(\mathbb{F}_{p^r})$. Then we replace the doubling map with the easier-to-compute Frobenius map. As a practical matter, Koblitz's idea works especially well for $p = 2$, so for concreteness we illustrate using the curve $E/\mathbb{F}_2$ given by the equation

$$E : y^2 + xy = x^3 + 1.$$

We have $E(\mathbb{F}_2) = \mathbb{Z}/4\mathbb{Z}$, so $E/\mathbb{F}_2$ is ordinary and

$$p + 1 - \#E(\mathbb{F}_p) = 2 + 1 - 4 = -1.$$

We use (V.2.3.1) to deduce that the Frobenius map

$$\tau : E(\mathbb{F}_{2^r}) \longrightarrow E(\mathbb{F}_{2^r}), \qquad (x, y) \longmapsto (x^2, y^2),$$

satisfies

$$\tau^2 + \tau + 2 = 0.$$

Using this relation, it is easy to write any integer n in the form

$$n = \epsilon_0 + \epsilon_1\tau + \epsilon_2\tau^2 + \cdots + \epsilon_t\tau^t \qquad \text{with } \epsilon_0, \ldots, \epsilon_t \in \{-1, 0, 1\},$$

where $t \approx 2\log_2(n)$ and at most one-third of the ϵ_i are nonzero. (With somewhat more work, the length of the expansion can be reduced to $t \approx \log_2(n)$; see Exercise 11.3.) Then $[n]P$ can be computed via

$$[n]P = \epsilon_0 P + \epsilon_1\tau(P) + \epsilon_2\tau^2(P) + \cdots + \epsilon_t\tau^t(P).$$

This is generally faster than using the binary or ternary expansion of n, because the Frobenius map on E is far easier to compute than the duplication map.

Remark 1.6. There are many variants of the basic double-and-add method that are used to make it more efficient in various situations. See for example [22, Chapter IV], [51, §9], and Exercise 11.4.

XI.2 Lenstra's Elliptic Curve Factorization Algorithm

Factorization of large numbers has been studied since antiquity, but the subject acquired added significance with the invention of public key cryptography, and in particular the development of the RSA cryptosystem, whose security depends on the difficulty of the factorization problem. Public key cryptography in general, and RSA

in particular, are described in many books; see for example [116, 169, 277]. In this section we focus on the factorization problem itself.

The modern theory of factorization, by which we mean factorization algorithms that take less than exponential time,[1] dates back only to the 1920s. The fastest factorization algorithm currently known is the *number field sieve*, which factors an integer N in approximately

$$\exp\left(c\sqrt[3]{(\log N)(\log\log N)^2}\right) \text{ steps.}$$

Before the invention of the number field sieve, the fastest factorization method was the *quadratic sieve*, whose running time is approximately

$$\exp\left(c\sqrt{(\log N)(\log\log N)}\right) \text{ steps.}$$

(Notice that the cube root has been replaced by a square root. However, due to the different values of the constants, the quadratic sieve is actually the faster of the two algorithms for factoring numbers up to about 10^{100}.)

In this section we describe a factorization method due to Hendrik Lenstra that uses elliptic curves and has a running time comparable to the quadratic sieve. However, Lenstra's algorithm has one useful characteristic that differentiates it from sieve methods. If p is the smallest prime factor of N, then the running time of Lenstra's algorithm is actually

$$\exp\left(c\sqrt{(\log p)(\log\log p)}\right) \text{ steps.}$$

Thus Lenstra's algorithm is especially good at finding prime factors of N that are significantly smaller than $\sqrt{N}$. However, we note that the moduli used for RSA have the form $N = pq$ with primes $p \approx q$, so sieve algorithms are more efficient than Lenstra's algorithm for factoring such numbers.

The prototype for Lenstra's work is an earlier factorization algorithm, due to Pollard, which we briefly describe. Pollard's algorithm is good at factoring numbers N that have a prime factor p such that $p-1$ is a product of small primes. Numbers that are a product of small primes are called *smooth numbers*.[2]

Pollard's $p-1$ Algorithm 2.1. *Suppose that N is a composite number that has a prime factor p such that $p-1$ factors into primes as*

$$p - 1 = q_1^{e_1} q_2^{e_2} \cdots q_t^{e_t}.$$

[1] The running time of an algorithm is measured as a function of the number of bits of the input and output. Thus an algorithm that factors an integer N in time $O(N^c)$ for some $c > 0$ takes *exponential time*, since the number of bits of the input is $\log_2(N)$. Similarly, a *polynomial-time algorithm* is one that runs in time $O\big((\log_2 N)^c\big)$, and a a *subexponential-time algorithm* is one that runs faster than $O(N^\epsilon)$ for every $\epsilon > 0$.

[2] More precisely, a number m is said to be *B-smooth* if every prime p dividing m satisfies $p \le B$. An important theorem of Canfield, Erdős, and Pomerance [34] gives an estimate for the number of B-smooth numbers less than a given bound.

(1) Choose a base value $2 \le a < N$ and set $A = a$.
(2) Loop $i = 1, 2, \ldots, L$.
(3) Replace A with $A^i \bmod N$.
(4) Compute $F = \gcd(A - 1, N)$.
(5) If $1 < F < N$, then return F, which is a nontrivial factor of N.
(6) If $F = N$, go to step (1) and choose a new value of a.
(7) End Loop

Figure 11.2: Pollard's $p - 1$ algorithm.

Let L be the quantity

$$L = \max_{1 \le j \le t} e_j q_j.$$

Then for most base values, the algorithm described in Figure 11.2 finds a nontrivial factor of N for some value of $i \le L$ in the main loop (steps (2)–(7)).

PROOF. During the i^{th} iteration of the loop, the value of A is $a^{i!} \bmod N$. The definition of L ensures that $q_j^{e_j}$ divides $L!$ for each $1 \le j \le t$, so $p - 1 \mid L!$. (See also Exercise 11.6.) It follows from Fermat's little theorem that $a^{L!} \equiv 1 \pmod{p}$, so the value of F in step (4) is divisible by p. Since it is unlikely that $a^{L!} \equiv 1 \pmod{N}$, we obtain a nontrivial factor of N. □

Example 2.2. We use Pollard's algorithm to factor $N = 71384665949740607$. Using the base $a = 2$, we find on the 33^{rd} iteration of the loop that

$$2^{33!} \equiv 58248995050016779 \pmod{71384665949740607},$$
$$\gcd(58248995050016778, 71384665949740607) = 228266501.$$

Thus

$$N = 71384665949740607 = 228266501 \cdot 312725107,$$

and one can check that both factors are prime.

Pollard's algorithm works well for this N because the prime $p = 228266501$ satisfies

$$p - 1 = 228266500 = 2^2 \cdot 5^3 \cdot 7^3 \cdot 11^3,$$

so $p - 1$ divides $33!$.

Pollard's algorithm is a valuable factorization tool, but it applies only to special sorts of numbers, namely those divisible by a prime p such that $p - 1$ is smooth. The significance of $p - 1$ lies in the fact that the multiplicative group $\mathbb{F}_p^*$ has order $p - 1$, so N and $a^{L!} - 1$ share a common factor of p as soon as $L!$ is divisible by $p - 1$. Lenstra's brilliant innovation was to observe that if one replaces the multiplicative group $\mathbb{F}_p^*$ by the points $E(\mathbb{F}_p)$ of an elliptic curve, then the group order varies as E varies. This allows the elliptic curve algorithm to factor a much larger set of numbers.

We first state the algorithm and then explain the various steps in more detail.

(0) Choose a loop bound L.
(1) Choose an elliptic curve $E \bmod N$ and a point $P \in E(\mathbb{Z}/N\mathbb{Z})$.
(2) Set $Q = P$.
(3) Loop $i = 2, 3, \ldots, L$.
(4) Replace Q with $[i]Q$, working in $E(\mathbb{Z}/N\mathbb{Z})$.
(5) If, during the computation of $[i](Q)$, you need the inverse of an element $a \in \mathbb{Z}/N\mathbb{Z}$ and that inverse does not exist, then $\gcd(a, N)$ is (probably) a nontrivial factor of N.
(6) End i Loop
(7) Go to step (1) and choose a new curve and point.

Figure 11.3: Lenstra's elliptic curve factorization algorithm.

Lenstra's Elliptic Curve Factorization Algorithm 2.3. *Let N be a positive integer to be factored, and consider the algorithm described in Figure 11.3. Suppose that N has a prime divisor p such that the loop bound L chosen in step* (0) *and the elliptic curve E chosen in step* (1) *satisfy*

$$\#E(\mathbb{F}_p) = q_1^{e_1} q_2^{e_2} \cdots q_t^{e_t} \qquad \text{and} \qquad L \ge \max\{e_1 q_1, \ldots, e_t q_t\}.$$

(*Here $q_1, \ldots, q_t$ are distinct primes.*) *Then with high probability, the algorithm described in Figure 11.3 factors the integer N.* (*See* (XI.2.4.5) *for advice on how to choose the loop bound L.*)

We now use a series of remarks to discuss various aspects of Lenstra's algorithm.

Remark 2.4.1. We have not heretofore worked with elliptic curves over rings such as $\mathbb{Z}/N\mathbb{Z}$ when N is composite. The fancy way to do this is via the theory of group schemes [266, Chapter IV], but for our purposes it suffices to take $A, B \in \mathbb{Z}/N\mathbb{Z}$ and use a Weierstrass equation

$$E : y^2 = x^3 + Ax + B \qquad \text{with} \quad \Delta = -16(4A^3 + 27B^2) \in (\mathbb{Z}/N\mathbb{Z})^*.$$

The choice of an elliptic curve modulo N in step (1) is thus simply a choice of $A, B \in \mathbb{Z}/N\mathbb{Z}$. (If we are unlucky and $\Delta \notin (\mathbb{Z}/N\mathbb{Z})^*$, then with high probability, $\gcd(\Delta, N)$ is a nontrivial factor of N.)

Remark 2.4.2. Having chosen an elliptic curve modulo N, it is not clear how to efficiently choose a point on that curve, since taking square roots modulo an unfactored N is a hard problem. The trick is to first choose A and the point $P = (x_0, y_0)$, and then set $B = y_0^2 - x_0^3 - Ax_0$.

Remark 2.4.3. Steps (4) and (5) require some explanation. The double-and-add method (XI.1.1) for computing $[i]Q$ involves additions $R_1 + R_2$ and duplications $[2]R$. We perform these operations using the standard formulas from (III.2.3), always working modulo N. For example, to add $R_1 = (x_1, y_1)$ and $R_2 = (x_2, y_2)$, we must compute

$$x(R_1+R_2) \equiv \left(\frac{y_2-y_1}{x_2-x_1}\right)^2 + a_1\left(\frac{y_2-y_1}{x_2-x_1}\right)^2 - a_2 - x_1 - x_2 \pmod{N}.$$

This works fine if the quantity $x_2 - x_1$ is invertible modulo N. However, if it is not invertible and we are unable to compute $R_1 + R_2 \bmod N$, then (Eureka!)

$$\gcd(x_2 - x_1, N) > 1,$$

and there is a good chance that $\gcd(x_2 - x_1, N)$ is a nontrivial factor of N. Thus in computing $[i]Q$ modulo N in step (4), either the computation works, or else we have (probably) factored N.

Remark 2.4.4. If we successfully complete L iterations of the i loop (steps (3)–(6)), the final value of Q is

$$Q = [L!]P \quad \text{in} \quad E(\mathbb{Z}/N\mathbb{Z}).$$

When does this computation fail?

Let p be a prime divisor of N, and let $n = n_p(E) = \#E(\mathbb{F}_p)$. We can use the reduction modulo p map $E(\mathbb{Z}/N\mathbb{Z}) \to E(\mathbb{Z}/p\mathbb{Z})$ to send the point P to the group $E(\mathbb{F}_p)$. Then, when we use the double-and-add method to compute $[n]P = O$ in $E(\mathbb{F}_p)$, at some stage we get a "zero in the denominator." Hence if $n_p(E) \mid L!$, then step (5) is executed and we (probably) find a nontrivial factor of N.

Remark 2.4.5. Notice the analogy with Pollard's algorithm (XI.2.1), where the relevant condition for success was $\#\mathbb{F}_p^* \mid L!$. The advantage of Lenstra's algorithm is that

$$n_p(E) = \#E(\mathbb{F}_p) = p + 1 - a_p(E)$$

varies as we choose different elliptic curves. (Here $a_p(E)$ is the trace of Frobenius (V.2.6).) Lenstra's algorithm succeeds if we manage to choose an elliptic curve E such that for some prime factor p of N, the order of the group $E(\mathbb{F}_p)$ is a smooth number.

In order to optimally implement Lenstra's algorithm, we need to decide how large to choose L, since this determines when we give up on a particular elliptic curve and choose a new one. We know from (V.1.1) that $a_p(E)$ satisfies $\bigl|a_p(E)\bigr| \le 2\sqrt{p}$, and it is not unreasonable to assume that the $a_p(E)$ values are more or less equidistributed in this range as E varies. (See (C.21.4) for a more precise statement.) So the probability of success for a chosen E depends on the distribution of smooth numbers in an interval around p. Using [34] as a heuristic to estimate the number of smooth numbers in short intervals, one can show that the optimal choice for L is approximately $\exp\bigl(c_1\sqrt{(\log p)(\log\log p)}\bigr)$, and that with this choice of L, the expected number of elliptic curves used before a factor of N is found is L^{c_2}. (Here c_1 and c_2 are small absolute constants.) Thus, as noted above, Lenstra's algorithm has the same qualitative running time as the quadratic sieve for numbers that are a product of two large primes, but it is generally much faster for numbers that have a comparatively small prime factor.

Remark 2.4.6. There are many implementation tricks that are used to make Lenstra's algorithm more efficient. We mention in particular the use of several elliptic curves

in parallel to save on mod N inversions, and the use of so-called Stage 2 computations, which are also used for Pollard's algorithm. For details see for example [50, §8.8].

Example 2.5. We use Lenstra's algorithm to factor $N = 6887$. We randomly select $P = (1512, 3166)$ and $A = 14$, and we set

$$B \equiv 3166^2 - 1512^3 - 14 \cdot 1512 \equiv 19 \pmod{6887},$$

so P is a mod N point on the elliptic curve

$$E : Y^2 = X^3 + 14X + 19.$$

We compute successively (always working modulo 6887)

$$\begin{aligned}
[2]P &\equiv (3466, 2996),\\
[3!]P = [3]\big([2]P\big) &\equiv (3067, 396)\ ,\\
[4!]P = [4]\big([3!]P\big) &\equiv (6507, 2654),\\
[5!]P = [5]\big([4!]P\big) &\equiv (2783, 6278),\\
[6!]P = [6]\big([5!]P\big) &\equiv (6141, 5581).
\end{aligned}$$

These values are not, themselves, of any intrinsic interest. To ease notation, we let $Q = [6!]P = (6141, 5581)$. We use the double-and-add algorithm (XI.1.1) to compute $[7]Q = [7!]P$. Thus

$$Q \equiv (6141, 5881), \quad [2]Q \equiv (5380, 174), \quad [4]Q \equiv [2]\big([2]Q\big) \equiv (203, 2038),$$

and

$$\begin{aligned}
7Q &\equiv \big(Q + [2]Q\big) + [4]Q\\
&\equiv \big((6141, 5581) + (5380, 174)\big) + (203, 2038)\\
&\equiv (984, 589) + (203, 2038)\\
&\equiv ???
\end{aligned}$$

When we attempt to perform the final addition, we need the inverse of $203 - 984$ modulo 6887, but

$$\gcd(203 - 984, 6887) = \gcd(-781, 6887) = 71.$$

Thus $71 \mid 6887$, and we find the factorization $6887 = 71 \cdot 97$.

It turns out that in $E(\mathbb{F}_{71})$, the point P satisfies $[63]P \equiv \mathcal{O} \pmod{71}$, while in $E(\mathbb{F}_{97})$, the point P satisfies $[107]P \equiv \mathcal{O} \pmod{97}$. The reason that we succeed in factoring 6887 using $[7!]P$, but not with a smaller multiple of P, is due to the fact that 7! is the smallest factorial that is divisible by 63 (and because 7! is not divisible by 107).

XI.3 Counting the Number of Points in $E(\mathbb{F}_q)$

Let $E/\mathbb{F}_q$ be an elliptic curve defined over a finite field. Hasse's theorem (V.1.1) says that

$$\#E(\mathbb{F}_q) = q + 1 - a_q \qquad \text{with} \quad |a_q| \le 2\sqrt{q}.$$

For many applications, especially in cryptography, it is important to have an efficient way to compute the number of points in $E(\mathbb{F}_q)$. For simplicity, we assume that q is odd and that E is given by a Weierstrass equation of the form

$$E : y^2 = f(x) = 4x^3 + b_2x^2 + 2b_4x + b_6,$$

but with minor modifications, everything that we do also works in characteristic 2.

A straightforward, but not very efficient, method to find $\#E(\mathbb{F}_q)$ is to compute the sum (cf. (V.1.3))

$$a_q = -\sum_{x\in\mathbb{F}_q} \left(\frac{f(x)}{q}\right).$$

Each Legendre symbol $\left(\frac{f(x)}{q}\right)$ can be computed by quadratic reciprocity in $O(\log q)$ steps, so this explicit formula takes $O(q \log q)$ steps, making it an exponential-time algorithm. (See also Exercise 11.14 for an algorithm that computes $\#E(\mathbb{F}_q)$ in $O(\sqrt{q}\,)$ steps.)

In this section we describe Schoof's algorithm [223], which computes $\#E(\mathbb{F}_q)$ in polynomial time, i.e., it computes $\#E(\mathbb{F}_q)$ in $O\big((\log q)^c\big)$ steps, where c is fixed, independent of q. The idea is to compute the value of a_q modulo ℓ for a lot of small primes ℓ and then use the Chinese remainder theorem to reconstruct a_q.

Let

$$\tau : E(\bar{\mathbb{F}}_q) \longrightarrow E(\bar{\mathbb{F}}_q), \qquad (x, y) \longmapsto (x^q, y^q),$$

be the q-power Frobenius map, so (V.2.3.1b) tells us that

$$\tau^2 - a_q\tau + q = 0 \qquad \text{in } \mathrm{End}(E).$$

In particular, if $P \in E(\bar{\mathbb{F}}_q)[\ell]$, then

$$\tau^2(P) - [a_q]\tau(P) + [q]P = O,$$

so if we write $P = (x, y)$ (we assume that $P \ne O$), then

$$(x^{q^2}, y^{q^2}) - [a_q](x^q, y^q) + [q](x, y) = O.$$

A key observation is that since the point $P = (x, y)$ is assumed to have order ℓ, we have

$$[a_q](x^q, y^q) = [n_\ell](x^q, y^q), \quad \text{where } n_\ell \equiv a_q \pmod{\ell} \text{ with } 0 \le n_\ell < \ell.$$

Similarly, we can compute $[q](x, y)$ by first reducing q modulo ℓ.

Of course, we don't know the value of n_ℓ, so for each integer n between 0 and ℓ we compute $[n](x^q, y^q)$ for a point $(x, y) \in E[\ell] \smallsetminus \{O\}$ and check to see whether it satisfies

$$[n](x^q, y^q) = (x^{q^2}, y^{q^2}) + [q](x, y).$$

However, the individual points in $E[\ell]$ tend to be defined over fairly large extension fields of $\mathbb{F}_q$, so we instead work with all of the ℓ-torsion points simultaneously. To do this, we use the division polynomial (see Exercise 3.7)

$$\psi_\ell(x) \in \mathbb{F}_q[x],$$

whose roots are the x-coordinates of the nonzero ℓ-torsion points of E. (For simplicity, we assume that $\ell \neq 2$.) This division polynomial has degree $\frac{1}{2}(\ell^2 - 1)$ since $\ell \nmid q$, and it is easily computed using the recurrence described in Exercise 3.7. We now perform all computations in the quotient ring

$$R_\ell = \frac{\mathbb{F}_q[x, y]}{\big(\psi_\ell(x), y^2 - f(x)\big)}.$$

Thus anytime we have a nonlinear power of y, we replace y^2 with $f(x)$, and anytime we have a power x^d with $d \geq \frac{1}{2}(\ell^2 - 1)$, we divide by $\psi_\ell(x)$ and take the remainder. In this way we never have to work with polynomials of degree greater than $\frac{1}{2}(\ell^2 - 3)$.

Our goal is to compute the value of $a_q \bmod \ell$ for enough primes ℓ to determine a_q. Hasse's theorem (V.1.1) says that $|a_q| \leq 2\sqrt{q}$, so it suffices to use all primes $\ell \leq \ell_{\max}$ such that

$$\prod_{\ell \leq \ell_{\max}} \ell \geq 4\sqrt{q}.$$

The preceding discussion shows that the following algorithm computes $\#E(\mathbb{F}_q)$. The subsequent proof estimates how long the computation takes.

Schoof's Algorithm 3.1. *Let $E/\mathbb{F}_q$ be an elliptic curve. The algorithm described in Figure 11.4 is a polynomial-time algorithm to compute $\#E(\mathbb{F}_q)$; more precisely, it computes $\#E(\mathbb{F}_q)$ in $O\big((\log q)^8\big)$ steps.*

PROOF. We prove that the running time of Schoof's algorithm is $O\big((\log q)^8\big)$. We begin by verifying three claims.

(a) *The largest prime ℓ used by the algorithm satisfies $\ell \leq O(\log q)$.*

The prime number theorem is equivalent to the statement [4, Theorem 4.4(9)]

$$\lim_{X \to \infty} \frac{1}{X} \sum_{\substack{\ell \leq X \\ \ell \text{ prime}}} \log \ell = 1.$$

Hence $\prod_{\ell \leq X} \ell \approx e^X$, so in order to make the product larger than $4\sqrt{q}$, it suffices to take $X \approx \frac{1}{2}\log(16q)$.

(1) Set $A = 1$ and $\ell = 3$.
(2) Loop while $A < 4\sqrt{q}$.
(3) Loop $n = 0, 1, 2, \ldots, \ell - 1$.
(4) Working in the ring R_ℓ, if
$$(x^{q^2}, y^{q^2}) + [q](x, y) = [n](x^q, y^q),$$
then break out of the n loop.
(5) End n Loop
(6) Set $A = \ell \cdot A$
(7) Set $n_\ell = n$
(8) Replace ℓ by the next largest prime.
(9) End A Loop
(10) Use the Chinese remainder theorem to find an integer a satisfying $a \equiv n_\ell \pmod{\ell}$ for all of the stored values of n_ℓ.
(11) Return the value $\#E(\mathbb{F}_q) = q + 1 - a$.

Figure 11.4: Schoof's algorithm.

(b) *Multiplication in the ring R_ℓ can be done in $O(\ell^4(\log q)^2)$ bit operations.*[3]

Elements of the ring R_ℓ are polynomials of degree $O(\ell^2)$. Multiplication of two such polynomials and reduction modulo $\psi_\ell(x)$ takes $O(\ell^4)$ elementary operations (additions and multiplications) in the field $\mathbb{F}_q$. Similarly, multiplication in $\mathbb{F}_q$ takes $O\big((\log q)^2\big)$ bit operations. So basic operations in R_ℓ take $O\big(\ell^4(\log q)^2\big)$ bit operations.

(c) *It takes $O(\log q)$ ring operations in R_ℓ to reduce $x^q, y^q, x^{q^2}, y^{q^2}$ in the ring R_ℓ.*

In general, the square-and-multiply algorithm (XI.1.2) allows us to compute powers x^n and y^n using $O(\log n)$ multiplications in R_ℓ. We note that this computation is done only once, and then the points

$$(x^{q^2}, y^{q^2}) + [q \bmod \ell](x, y) \qquad \text{and} \qquad (x^q, y^q)$$

are computed and stored for use in step (4) of Schoof's algorithm.

We now use (a), (b), and (c) to estimate the running time of Schoof's algorithm. From (a), we need to use only primes ℓ that are less than $O(\log q)$. There are $O(\log q / \log\log q)$ such primes, so that is how many times the A-loop, steps (2)–(9), is executed. Then, each time we go through the A-loop, the n loop, steps (3)–(5), is executed $\ell = O(\log q)$ times.

[3] A bit operation is a basic computer operation on one or two bits. Examples of bit operations include addition, multiplication, and, or, xor, and complement. Fancier multiplication methods based on fast Fourier transforms or Karatsuba multiplication can be used to reduce multiplication in R_ℓ to $O\big((\ell^2 \log q)^{1+\epsilon}\big)$ bit operations, at the cost of a larger big-O constant.

Further, since $\ell = O(\log q)$, claim (b) says that basic operations in R_ℓ take $O\big((\log q)^6\big)$ bit operations. The value of $[n](x^q, y^q)$ in step (4) can be computed in $O(1)$ operations in R_ℓ from the previous value $[n-1](x^q, y^q)$, or we can be inefficient and compute it in $O(\log n) = O(\log \ell) = O(\log\log q)$ R_ℓ-operations using the double-and-add algorithm (XI.1.1).

Hence the total number of bit operations required by Schoof's algorithm is

$$\overbrace{O(\log q)}^{A\text{ loop}} \cdot \overbrace{O(\log q)}^{n\text{ loop}} \cdot \overbrace{O\big((\log q)^6\big)}^{\substack{\text{bit operations per}\\ R_\ell\text{ operation}}} = O\big((\log q)^8\big) \text{ bit operations.}$$

This completes the proof that Schoof's algorithm computes $\#E(\mathbb{F}_q)$ in polynomial time. □

The most time-consuming part of Schoof's algorithm consists of computations in the ring R_ℓ, which is an extension of $\mathbb{F}_q$ of degree $2\ell^2$. So even though the bound for ℓ is linear in $\log q$, if q is reasonably large, then the bound for ℓ and the $\mathbb{F}_q$-dimension of the ring R_ℓ are large.

Example 3.2. Let $q \approx 2^{256}$, which is a typical size used in cryptographic applications. We have

$$\prod_{\ell \le 103} \ell \approx 2^{133.14} > 4\sqrt{q} = 2^{130},$$

so the largest prime ℓ required by Schoof's algorithm is $\ell = 103$. An element of $\mathbb{F}_q[x]/(\psi_\ell(x))$ is represented by an $\mathbb{F}_q$-vector of dimension $103^2 \approx 2^{13.4}$, and each element of $\mathbb{F}_q$ is a 256-bit number, so elements of $\mathbb{F}_q[x]/(\psi_\ell(x))$ are approximately 2^{22} bits, which is more than 16 KB. Although modern computers are quite capable of working with rings whose elements are 16 KB, extensive calculations in such rings take nontrival amounts of time.

The SEA Algorithm 3.3. Suppose that the ℓ-division polynomial $\psi_\ell(x)$ factors in $\mathbb{F}_q[x]$, say $f_\ell(x) \mid \psi_\ell(x)$ with $\deg f_\ell = \ell$. Then we can obtain significant savings in Schoof's algorithm by working in the smaller ring

$$R'_\ell = \frac{\mathbb{F}_q[x,y]}{\big(f_\ell(x), y^2 - f(x)\big)}.$$

Multiplication in the ring R'_ℓ takes $O(\ell^2(\log q)^2)$ bit operations, as compared to $O(\ell^4(\log q)^2)$ bit operations for multiplication in R_ℓ. For the example described in (XI.3.2), this amounts to a potential speedup on the order of 10^4.

The division polynomial $\psi_\ell(x)$ has an $\mathbb{F}_q[x]$-factor of degree ℓ if and only if the group of ℓ-torsion points $E[\ell]$ has a cyclic subgroup $C \subset E[\ell]$ of order ℓ that is defined over $\mathbb{F}_q$, i.e., such that C is a $G_{\mathbb{F}_{q^n}/\mathbb{F}_q}$-invariant subgroup. Equivalently, from (III.4.12), $\psi_\ell(x)$ factors in this way if and only if there is an isogeny $E \to E'$ of degree ℓ defined over $\mathbb{F}_q$.

The idea of using factors of $\psi_\ell(x)$ was proposed and developed by Schoof, Elkies, and Atkin and is known as the *SEA algorithm*. Efficient computation of a factor $f_\ell(x)$ uses the modular polynomial [266, II.6.3]. For a description of the SEA algorithm, see for example [22, Chapter 7], [79], or [224].

XI.4 Elliptic Curve Cryptography

Public key cryptography was invented by Diffie and Hellman[4] in 1976 [65], although they were not able to find a practical method to implement their idea. The first practical public key cryptosystem was devised the following year by Rivest, Shamir, and Adleman [209]. The famed RSA cryptosystem bases its security on the difficulty of factoring large numbers. However, Diffie and Hellman did describe a key exchange algorithm whose security relies on the discrete logarithm problem in $\mathbb{F}_q^*$, and subsequently ElGamal created a public key cryptosystem based on the same underlying problem. Koblitz [128] and Miller [176] suggested replacing the finite field $\mathbb{F}_q$ with an elliptic curve E, with the hope that the discrete logarithm problem in the elliptic curve group $E(\mathbb{F}_q)$ might be harder to solve than the discrete logarithm problem in the multiplicative group $\mathbb{F}_q^*$. Their intuition led to the creation of elliptic curve cryptography.

The subject of cryptography is vast, and although cryptography is not the focus of this book, we take the opportunity in this section and in (XI §7) to briefly indicate some of the ways in which elliptic curves are applied. This material is meant merely to whet the reader's appetite, so be aware that we ignore many of the subtleties inherent in the subject. Readers desiring more information on the mathematical aspects of cryptography may consult any of the numerous volumes on the subject, including for example [116], [169], or [277]. For books devoted to the use of elliptic (and hyperelliptic) curves in cryptography, see for example [22] or [51].

Public key cryptosystems rely on what are known as *one-way trapdoor functions*. These are easy-to-compute injective functions $f : A \to B$ with the property that f^{-1} is very hard to compute in general, but f^{-1} becomes quite easy to compute if someone possesses an extra piece of information k. Thus, if Alice[5] knows the value of k, then Bob can send her a message $a \in A$ by sending her the quantity $b = f(a)$. Alice easily recovers $a = f^{-1}(b)$, since she knows k, while Eve, who does not know k, is unable to compute $f^{-1}(b)$.

It is not clear that one-way trapdoor functions exist, and indeed, it is still an open problem to prove their existence. However, a number of hard mathematical problems have been proposed as the bases for one-way trapdoor functions, including in particular the discrete logarithm problem.

Definition. Let G be group, and let $x, y \in G$ be elements such that y is in the subgroup generated by x. The *discrete logarithm problem* (DLP) is the problem of determining an integer $m \geq 1$ such that

$$x^m = y.$$

[4] The concept of public key cryptography was actually originally described by James Ellis in 1969, but his discovery was classified by the British government and not declassified until after his death in 1997. Two other British government employees, Williamson and Cocks, are the original inventors of the Diffie–Hellman key exchange algorithm and the RSA public key cryptosystem, respectively, but their discoveries were also classified.

[5] In cryptography it is customary to personalize the participants. Typically, Alice and Bob want to communicate, while Eve, the eavesdropper, intercepts and tries to read their messages.

Example 4.1. Each group G has its own discrete logarithm problem. In the next section we describe a collision algorithm (XI.5.2) that takes $O(\sqrt{\#G}\,)$ steps to solve the DLP in virtually any group G. However, this square root estimate is only an upper bound for the computational complexity of the DLP. It turns out that the DLP is significantly easier in some groups than it is in others. We mention three examples in increasing order of difficulty.

(a) *The Additive Group* $\mathbb{F}_q^+$. The DLP for the additive group of a finite field $\mathbb{F}_q$ asks for a solution m to the linear equation $xm = y$ for given $x, y \in \mathbb{F}_q$. To solve this equation, we need only find the multiplicative inverse of x in $\mathbb{F}_q$, which takes $O(\log q)$ steps using the Euclidean algorithm. Thus the DLP in $\mathbb{F}_q^+$ is a very easy problem.

(b) *The Multiplicative Group* $\mathbb{F}_q^*$. The DLP for the multiplicative group of a finite field $\mathbb{F}_q$ asks for a solution m to the exponential equation $x^m = y$ for given $x, y \in \mathbb{F}_q^*$. As already noted, the DLP in any group of order $O(q)$ can be solved in $O(\sqrt{q}\,)$ steps, but there are algorithms for the DLP in $\mathbb{F}_q^*$ that are much faster, taking fewer than $O(q^\epsilon)$ steps for every $\epsilon > 0$. These go by the general name of *index calculus* methods, and they solve the DLP in $\mathbb{F}_q^*$ in

$$\exp\left(c\sqrt[3]{(\log q)(\log\log q)^2}\right) \text{ steps,}$$

where c is a small absolute constant. Thus the index calculus is a subexponential algorithm. We note that it is not a coincidence that the running time of the number field sieve and the index calculus have the same form, since both rely on the distribution of smooth numbers. For further information about the index calculus, see for example [116, §3.8] or [277, §6.2.4].

(c) *An Elliptic Curve* $E(\mathbb{F}_q)$. The elliptic curve discrete logarithm problem, which is abbreviated ECDLP, asks for a solution m to the equation $[m]P = Q$ for given points $P, Q \in E(\mathbb{F}_q)$. Despite extensive research since the mid-1980s, the fastest known algorithms to solve the ECDLP on general curves are collision algorithms taking $O(\sqrt{q}\,)$ steps. Thus the best known algorithms to solve the ECDLP in $E(\mathbb{F}_q)$ take exponential time, i.e., the running time is exponential in $\log q$. This fact is the primary attraction for using elliptic curves in cryptography.

There is a key exchange system based on the DLP that is due to Diffie and Hellman and a public key cryptosystem based on the DLP that is due to ElGamal. These systems work, mutatis mutandis, for any group and are typically applied to (subgroups) of either $\mathbb{F}_q^*$ or $E(\mathbb{F}_q)$. In keeping with the primary subject of this book, we describe these systems in terms of elliptic curves. As already noted, the primary advantage of using elliptic curves is that at present, it is much harder to solve the ECDLP in $E(\mathbb{F}_q)$ than it is to solve the DLP in $\mathbb{F}_q^*$. This means that *elliptic curve cryptography* has key and message sizes that are 5 to 10 times smaller than those for other systems, including RSA and $\mathbb{F}_q^*$-based DLP systems.

The first algorithm that we describe allows Alice and Bob to securely exchange a piece of information whose value neither one of them knows in advance. We discuss later (XI.4.3.3) why this might be useful.

Diffie–Hellman Key Exchange 4.2. *The following procedure allows Alice and Bob to securely exchange the value of a point on an elliptic curve, although neither of them initially knows the value of the point*:

(1) Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, and a point $P \in E(\mathbb{F}_q)$.
(2) Alice selects a secret integer a and computes the point $A = [a]P \in E(\mathbb{F}_q)$.
(3) Bob selects a secret integer b and computes the point $B = [b]P \in E(\mathbb{F}_q)$.
(4) Alice and Bob exchange the values of A and B over a possibly insecure communication line.
(5) Alice computes $[a]B$ and Bob computes $[b]A$. They have now shared the value of the point $[ab]P$.

We briefly mention a few of the many issues that must be addressed before this rough outline of Diffie–Hellman key exchange becomes a usable system.

Remark 4.3.1. Typically, the finite field $\mathbb{F}_q$, the elliptic curve $E/\mathbb{F}_q$, and the point $P \in E(\mathbb{F}_q)$ are preselected and published by a standards body. See (XI.4.7).

Remark 4.3.2. It is essential that the order of P be divisible by a large prime, because a Chinese remainder algorithm due to Pohlig and Hellman [205] shows that the solution time of the ECDLP depends only on the largest prime dividing the order of P. For this and other reasons, it is generally advisable to use a point P of prime order. For details of the Pohlig–Hellman algorithm, see [116, § 2.9], [277, §5.1.1], or Exercise 11.9.

Remark 4.3.3. Diffie–Hellman key exchange allows Alice and Bob to exchange a piece of data that neither knows in advance. This may not seem very useful. However, it is useful, because they can use this "random" piece of data as the secret key for a private key cryptosystem such as the advanced encryption standard (AES).

Remark 4.3.4. Alice and Bob's adversary Eve knows the values of P, $A = [a]P$, and $B = [b]P$, so if Eve can solve the ECDLP, then she can find a (or b) and recover Alice and Bob's secret value. However, in principle, Eve does not need to find a or b. What Eve needs to do is to solve the following problem:

Elliptic Curve Diffie–Hellman Problem
Given three points P, $[a]P$, and $[b]P$ in $E(\mathbb{F}_q)$, compute the point $[ab]P$.

At present, the only way known to solve the elliptic curve Diffie-Hellman problem is to solve the associated elliptic curve discrete logarithm problem, i.e., no one knows how to compute $[ab]P$ from P, $[a]P$, and $[b]P$ without knowing one of a or b.

Remark 4.3.5. There is clearly no need for Alice and Bob to exchange both the x and y coordinates of a point, since the x-coordinate alone determines y up to ± 1. Thus given Bob's x value, Alice can determine $\pm y$ by computing a square root in $\mathbb{F}_q$. (See Exercise 11.8.) It thus suffices for Bob to send the value of x and one additional bit that specifies which square root to take for y. In cryptographic circles, the idea of sending the x-coordinate plus one extra bit is known as *point compression*.

Diffie–Hellman key exchange allows Alice and Bob to exchange a random bit string, but a true public key cryptosystem such as RSA allows Bob to send a specific message to Alice. A public key cryptosystem based on the discrete logarithm problem in $\mathbb{F}_q^*$ was proposed in 1985 by ElGamal [74]. Here is an elliptic curve version.

ElGamal Public Key Cryptosystem 4.4. *The following procedure allows Bob to securely send a message to Alice without any previous communication.*

(1) Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, and a point $P \in E(\mathbb{F}_q)$.
(2) Alice selects a secret integer a and computes the point $A = [a]P \in E(\mathbb{F}_q)$.
(3) Alice publishes the point A. This is her *public key*. The secret multiplier a is her *private key*.
(4) Bob chooses a *plaintext* (i.e., a message) $M \in E(\mathbb{F}_q)$ and a random integer k. He computes the two points

$$B_1 = [k]P \in E(\mathbb{F}_q) \qquad \text{and} \qquad B_2 = M + [k]A \in E(\mathbb{F}_q).$$

(5) Bob sends the *ciphertext* (B_1, B_2) to Alice over a potentially insecure communication line.
(6) Alice uses her secret key a to compute $B_2 - [a]B_1 \in E(\mathbb{F}_q)$. This value is equal to Bob's plaintext M.

There is much to say about the ElGamal cryptosystem, but we content ourselves with a few brief remarks.

Remark 4.5.1. It is easy to verify that Alice recovers Bob's plaintext. Thus

$$B_2 - [a]B_1 = \big(M + [k]A\big) - [a][k]P = M + [k][a]P - [a][k]P = M.$$

Notice how Bob's random integer k disappears from the calculation.

Remark 4.5.2. Just as with Diffie–Hellman key exchange, the field $\mathbb{F}_q$, curve $E/\mathbb{F}_q$, and point $P \in E(\mathbb{F}_q)$ are typically chosen from a list published by some trusted authority; see (XI.4.7). We also note that Alice may choose her private key (step 2) and publish her public key (step 3) without knowing who is planning to send messages to her, nor when those messages will be sent.

Remark 4.5.3. An ElGamal plaintext is a point $M \in E(\mathbb{F}_q)$, while an ElGamal ciphertext is a pair of points $B_1, B_2 \in E(\mathbb{F}_q)$. Thus even with point compression (XI.4.3.5), Bob has to send two bits of information to Alice for every one bit in his message. We say that ElGamal has 2-*to*-1 *message expansion*. This is less efficient than the RSA cryptosystem, whose plaintexts and ciphertexts are the same size.

Remark 4.5.4. In practice, there is no natural way to assign a message written in, say, English, to a point $M \in E(\mathbb{F}_p)$. A variant of the ElGamal system due to Menezes and Vanstone uses the coordinates of a point in E as a mask for the actual message; see Exercise 11.10. We also note that if ElGamal is used in the raw state described in (XI.4.4), then it is subject to various sorts of attacks. All practical secure implementations of modern public key cryptosystems include some sort of internal message structure that allows Alice to verify that Bob's message was properly encrypted.

An example of such a method is the Integrated Encryption Scheme (IES) due to Abdalla, Bellare, and Rogaway [1]. We briefly describe the elliptic curve variant of IES in Exercise 11.11.

Remark 4.5.5. As with Diffie–Hellman key exchange, the ElGamal cryptosystem can be broken by solving the Diffie–Hellman problem (XI.4.3.4). Thus Eve knows $A = [a]P$ and $B_1 = [k]P$, so if she can solve the Diffie–Hellman problem, then she can compute $[ak]P = [k]A$. Since she also knows B_2, she is then able to compute $B_2 - [k]A = M$.

A public key cryptosystem allows Bob and Alice to exchange information. A *digital signature scheme* has a different purpose. It allows Alice to use a private key to sign a digital document, e.g., a computer file, in such a way that Bob can use Alice's public key to verify the validity of the signature. There are a number of practical digital signature algorithms; see for example [116, 169, 277]. We describe one such algorithm that uses elliptic curves.

Elliptic Curve Digital Signature Algorithm (ECDSA) 4.6. *The following procedure allows Alice to sign a digital document and Bob to verify that the signature is valid*:

(1) Alice and Bob agree on a finite field $\mathbb{F}_p$, an elliptic curve $E/\mathbb{F}_p$, and a point $P \in E(\mathbb{F}_p)$ of (prime) order N.

(2) Alice selects a secret integer a and computes the point $A = [a]P \in E(\mathbb{F}_p)$.

(3) Alice publishes the point A. This is her *public verification key*. The secret multiplier a is her *private signing key*.

(4) Alice chooses a digital document $d \bmod N$ to sign.[6] She also chooses a random integer $k \bmod N$. Alice computes $[k]P$ and sets

$$s_1 \equiv x\big([k]P\big) \pmod N \qquad \text{and} \qquad s_2 \equiv (d + as_1)k^{-1} \pmod N.$$

(Here $x\big([k]P\big)$ is only in $\mathbb{F}_p$, but we choose an integer representative between 0 and $p-1$.) Alice publishes the signature (s_1, s_2) for the document d.

(5) Bob computes

$$v_1 \equiv ds_2^{-1} \pmod N \qquad \text{and} \qquad v_2 \equiv s_1 s_2^{-1} \pmod N.$$

He then computes $[v_1]P + [v_2]A \in E(\mathbb{F}_p)$ and verifies that

$$x\big([v_1]P + [v_2]A\big) \equiv s_1 \pmod N.$$

PROOF. We need to check that if Alice follows the procedure described in step (4), then Bob's verification in step (5) works. The point that Bob computes in step (5) is

[6] In practice, Alice applies a *hash function* to her actual document in order to obtain an integer $d \bmod N$. This allows her to sign long documents and prevents various types of attacks. For information about hash functions and their use in cryptography, see for example [51, §§1.6.3, 24.2.5], [116, §8.1], or [169, Chapter 9].

$$\begin{aligned}[v_1]P + [v_2]A &= [ds_2^{-1}]P + [s_1 s_2^{-1}][a]P \quad \text{using the values of } s_1,\ s_2, \text{ and } A,\\ &= \big[s_2^{-1}(d + as_1)\big]P\\ &= [k]P \quad \text{using the value of } s_2.\end{aligned}$$

Hence

$$x\big([v_1]P + [v_2]A\big) = x\big([k]P\big) \equiv s_1 \pmod{N}$$

by definition of s_1. □

Remark 4.7. Before using elliptic curves to exchange keys or messages or to sign documents, Alice and Bob need to choose a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, and a point $P \in E(\mathbb{F}_q)$ having large prime order. This selection process can be time consuming, but there is no need for every Alice and every Bob to choose their own individual fields, curves, and points. The only secret personal information utilized by Alice and Bob consists of the multipliers they use to form multiples of P. In order to make Alice and Bob's life easier, the United States National Institute of Standards and Technology (NIST) published a list [191] of fifteen fields, curves, and points for Alice and Bob to use. For each of five different security levels, NIST gives one curve $E/\mathbb{F}_p$ with p a large prime, one curve $E/\mathbb{F}_{2^k}$, and one Koblitz curve $E/\mathbb{F}_2$ as in (XI.1.5) with a point $P \in E(\mathbb{F}_{2^k})$.

XI.5 Solving the Elliptic Curve Discrete Logarithm Problem: The General Case

Recall that the discrete logarithm problem (DLP) for elements x and y in a group G asks for an integer m such that $x^m = y$. In this section we discuss the best known algorithms for solving the DLP in arbitrary groups. The following rough criteria will be used to describe the complexity of an algorithm.

Definition. We will say that a discrete logarithm algorithm *takes T steps and requires S storage* if, on average, the algorithm needs to compute T group operations and needs to store the values of S group elements. (We will ignore the time it takes to sort or compare lists of elements, since the time for such operations is generally logarithmically small compared to the number of group operations.)

Example 5.1. Let $x \in G$ be an element of order n. The *naive algorithm* for solving the DLP is to compute $x, x^2, x^3, \ldots$ until y is found. This method takes $O(n)$ steps and requires $O(1)$ storage.

We now describe a general algorithm that "square-roots" the number of steps required to solve the DLP compared to the naive algorithm, albeit at the cost of using a significant amount of storage. Algorithms of this type are called *collision algorithms*, because they depend on the fact that it is easier to find collisions (elements that are common to two subsets) than it is to find specific elements in a set. This phenomenon is also known as the *birthday paradox*; see Exercise 11.5.

Proposition 5.2. (Shanks's Babystep–Giantstep Algorithm) *Let G be a group, let $x, y \in G$, and let n be the order of x. Then the following algorithm solves the DLP in $O(\sqrt{n}\,)$ steps with $O(\sqrt{n}\,)$ storage*:

(1) Let $N = \lceil \sqrt{n}\,\rceil$ be the "ceiling" of n, i.e., N is the smallest integer that is greater than or equal to $\sqrt{n}$.
(2) Make a list of the elements (these are the babysteps)
$$x,\ x^2,\ x^3,\ \ldots,\ x^N.$$
(3) Let $z = (x^N)^{-1}$ and make a list of the elements (these are the giantsteps)
$$yz,\ yz^2,\ yz^3,\ \ldots,\ yz^N.$$
(4) Look for a match between the lists in steps (2) and (3). If there is a match, say $x^i = yz^j$, then $y = x^{i+jN}$; otherwise y is not a power of x.

PROOF. Suppose that y is equal to a power of x, say $y = x^m$ with $0 \le m < n$. We write $m = jN + i$ with $0 \le i < N$, so
$$0 \le j = (m-i)/N \le N, \qquad \text{since } m \le n \text{ and } N \ge \sqrt{n}.$$
It follows that x^i is in the first list and $yz^j = yx^{-jN}$ is in the second list, so there is a match $x^i = yz^j$. Hence $y = x^i z^{-j} = x^i (x^{-N})^{-j} = x^{i+jN}$.

We note that there are many ways to check for matches in step (4). For example, we can sort the elements $x, x^2, \ldots, x^N$ in step (2) in $O(N \log N)$ steps, and then it takes $O(\log N)$ steps to check whether any particular element yz^i from step (3) is in the list. So Shanks's algorithm really takes $O\big(\sqrt{n}(\log n)^2\big)$ steps (or a bit less, using fancier sorting algorithms), but as noted earlier, we will ignore the log factors. □

The babystep–giantstep algorithm (XI.5) requires a considerable amount of storage. An alternative collision algorithm, due to Pollard, takes approximately the same number of steps and reduces the storage to essentially nothing. Pollard's algorithm and its variants, which are the most practical methods currently known for solving the ECDLP, rely on the following collision theorem that describes the likelihood of finding terms satisfying $x_{2i} = x_i$ in an iterative sequence $x_0, x_1, x_2, \ldots$.

Theorem 5.3. *Let S be a finite set containing N elements, and let $f : S \to S$ be a function. Starting with an initial value $x_0 \in S$, define a sequence of points $x_0, x_1, x_2, \ldots$ by*
$$x_i = f(x_{i-1}) = \underbrace{f \circ f \circ \cdots \circ f}_{i \text{ iterations of } f}(x_0).$$
Let T be the tail length and let L be the loop length of the orbit
$$x_0, x_1, x_2, x_3, \ldots$$
of x, as illustrated in Figure 11.5. Formally,

T = *largest integer such that x_{T-1} appears only once in the sequence $(x_i)_{i\ge 0}$,*
L = *smallest integer such that $x_{T+L} = x_T$.*

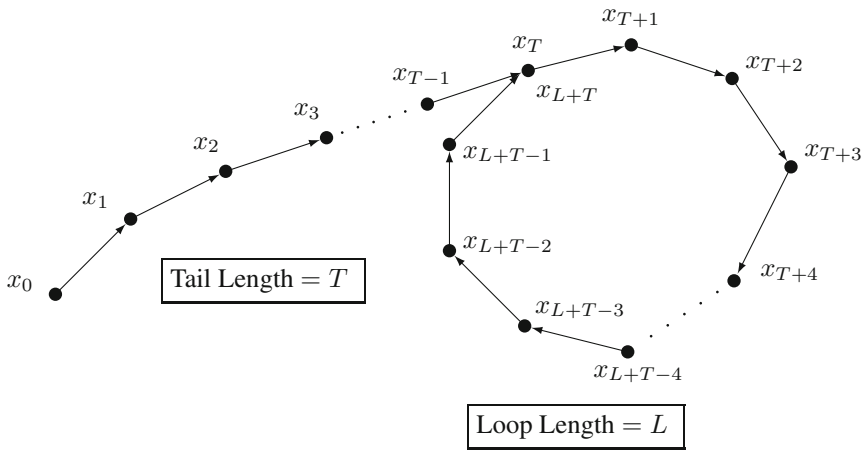

Figure 11.5: The orbit of x_0 in Pollard's ρ algorithm.

(a) *There exists an index* $1 \le i < T + L$ *such that* $x_{2i} = x_i$.
(b) *If* $f : S \to S$ *and its iterates are "sufficiently random" at mixing the elements of* S*, then the expected value of* $T + L$ *is* $\sqrt{\pi N/2}$.

Remark 5.3.1. The shape of the path in Figure 11.5 explains why (XI.5.3) is called the "ρ algorithm."

PROOF. (a) It is clear from Figure 11.5 that for $j > i$ we have

$$x_j = x_i \qquad \text{if and only if} \qquad i \ge T \quad \text{and} \quad j \equiv i \pmod{L}.$$

Hence $x_{2i} = x_i$ if and only if $i \ge T$ and $L \mid i$. The first such i lies between T and $T + L - 1$.
(b) We sketch the proof, which is an exercise in discrete probability theory, and leave the reader to fill in error-estimate details.

If k points $x_0, \dots, x_{k-1}$ are chosen randomly from S, then the probability that they are distinct is

$$\begin{aligned}
\operatorname{Prob}\begin{pmatrix} x_0, x_1, \dots, x_{k-1} \\ \text{are distinct} \end{pmatrix} &= \prod_{i=1}^{k-1} \operatorname{Prob}\left(\begin{matrix} x_i \neq x_j \text{ for} \\ \text{all } 0 \le j < i \end{matrix} \;\middle|\; \begin{matrix} x_0, x_1, \dots, x_{i-1} \\ \text{are distinct} \end{matrix} \right) \\
&= \prod_{i=1}^{k-1} \left(\frac{N-i}{N} \right) \\
&= \prod_{i=1}^{k-1} \left(1 - \frac{i}{N} \right).
\end{aligned}$$

We approximate this last product using the estimate

$$1 - t \approx e^{-t},$$

which is valid for small values of t. (We will apply this estimate with $k = O(\sqrt{N}\,)$, so if N is large, then the quantity $\frac{i}{N}$ with $1 \le i < k$ is small.) This yields

$$\operatorname{Prob}\left(\begin{matrix}x_0, x_1, \ldots, x_{k-1}\\ \text{are distinct}\end{matrix}\right) \approx \prod_{i=1}^{k-1} e^{-i/N} \approx e^{-k^2/2N}.$$

Suppose now that $x_0, \ldots, x_{k-1}$ are distinct. Then the probability that x_k matches one of the earlier values is

$$\operatorname{Prob}\bigl(x_k \text{ is a match} \bigm| x_0, \ldots, x_{k-1} \text{ are distinct}\bigr) = \frac{k}{N}.$$

Combining these two probability estimates, we find that

$$\begin{aligned}
&\operatorname{Prob}\bigl(x_k \text{ is the first match}\bigr)\\
&\qquad = \operatorname{Prob}\bigl(x_k \text{ is a match AND } x_0, \ldots, x_{k-1} \text{ are distinct}\bigr)\\
&\qquad = \operatorname{Prob}\bigl(x_k \text{ is a match} \bigm| x_0, \ldots, x_{k-1} \text{ are distinct}\bigr)\\
&\qquad\qquad\qquad \cdot \operatorname{Prob}\bigl(x_0, \ldots, x_{k-1} \text{ are distinct}\bigr)\\
&\qquad \approx \frac{k}{N} \cdot e^{-k^2/2N}.
\end{aligned}$$

Hence the expected number of steps before finding the first match is

$$\begin{aligned}
&\sum_{k\ge 1} k \cdot \operatorname{Prob}\bigl(x_k \text{ is the first match}\bigr)\\
&\qquad \approx \sum_{k\ge 1} \frac{k^2}{N} \cdot e^{-k^2/2N}\\
&\qquad = \sum_{k\ge 1} \phi\bigl(k/\sqrt{N}\,\bigr) \qquad\qquad \text{letting } \phi(t) = t^2 e^{-t^2/2},\\
&\qquad \approx \sqrt{N} \cdot \int_0^\infty t^2 e^{-t^2/2}\,dt \qquad\qquad \text{using } \sum_{k=1}^{\infty} \phi(k/n) \approx n \int_0^\infty \phi(t)\,dt,\\
&\qquad = \sqrt{\pi N/2}.
\end{aligned}$$

(The square of the integral in the last step may be evaluated via the usual polar coordinates trick.) □

Pollard's algorithm to solve the discrete logarithm problem in a group G uses a self-map $f : G \to G$ that is easy to compute, yet whose iterates mix up the elements of G in a random fashion.

Algorithm 5.4. (**Pollard's ρ algorithm**)
Let G be a group and let $x, y \in G$. Our goal is to compute an integer m satisfying $x^m = y$. We will use (XI.5.3) to find integers i, j, k, ℓ such that

$$x^i y^j = x^k y^\ell.$$

Then $x^{i-k} = y^{j-\ell}$, and assuming that $j - \ell$ is relatively prime to the order n of x,[7] we can solve for y as a power of x.

It is not clear how to define a function $f : G \to G$ that is complicated enough to provide good mixing, yet simple enough to keep track of its orbits. Pollard [206] suggests splitting G into a disjoint union of three sets of approximately equal size,

$$G = A \cup B \cup C,$$

and using the function

$$f(z) = \begin{cases} xz & \text{if } z \in A, \\ z^2 & \text{if } z \in B, \\ yz & \text{if } z \in C. \end{cases}$$

(See Exercise 11.13 for an elliptic curve example.) Such functions work reasonably well in practice, although more complicated functions with better mixing properties are known [293, 294].

Consider the outcome when we repeatedly apply f to the initial point $z_0 = 1$. After i iterations we arrive at a point

$$z_i = \underbrace{f \circ f \circ \cdots \circ f}_{i \text{ iterations of } f}(1) = x^{\alpha_i} y^{\beta_i}$$

for certain integers α_i and β_i. It is difficult to predict, a priori, the values of α_i and β_i, but we can compute them at the same time that we compute $z_1, z_2, \ldots$ by starting with $\alpha_0 = \beta_0 = 0$ and using the iterative formulas

$$\alpha_{i+1} = \begin{cases} \alpha_i + 1 & \text{if } z_i \in A, \\ 2\alpha_i & \text{if } z_i \in B, \\ \alpha_i & \text{if } z_i \in C, \end{cases} \qquad \beta_{i+1} = \begin{cases} \beta_i & \text{if } z_i \in A, \\ 2\beta_i & \text{if } z_i \in B, \\ \beta_i + 1 & \text{if } z_i \in C. \end{cases}$$

Note that we need only keep track of α_i and β_i modulo n, since $x^n = 1$. This keeps the values of α_i and β_i at a manageable size.

In a similar fashion we compute the sequence of points

$$w_0 = 1 \qquad \text{and} \qquad w_{i+1} = f\big(f(w_i)\big).$$

Then

$$w_i = z_{2i} = x^{\gamma_i} y^{\delta_i},$$

where γ_i and δ_i can be computed from γ_{i-1} and δ_{i-1} using two repetitions of the recurrences for α_i and β_i. Of course, the first time we use $w_i = z_{2i}$ to determine which case to apply, and the second time we use $f(w_i) = z_{2i+1}$ to decide. See Exercise 11.12.

[7] In practical applications, the element x usually has prime order (XI.4.3.2), in which case $j - \ell$ is almost certainly prime to n. The general case is discussed later in this section.

We now compute (z_1, w_1), (z_2, w_2), (z_3, w_3),... until we find a pair whose coordinates are the same. Note that each successive (z_i, w_i) may be computed solely in terms of the previous (z_{i-1}, w_{i-1}), so we never need to store more than a few numbers. Assuming that A, B, and C are sufficiently good at mixing the elements of G, our analysis in (XI.5.3) says that we will find a match

$$z_i = w_i = z_{2i}$$

in $O(\sqrt{n}\,)$ steps. The equality $z_i = w_i$ implies that

$$x^{\alpha_i - \gamma_i} = y^{\delta_i - \beta_i}.$$

If $\gcd(\delta_i - \beta_i, n) = 1$, as is typically the case in applications where n is prime, then

$$m \equiv (\alpha_i - \gamma_i)(\delta_i - \beta_i)^{-1} \pmod{n}$$

solves $x^m = y$.

In general, if $d = \gcd(\delta_i - \beta_i, n) > 1$, then we can express y^d as a power of x, say $y^d = x^e$. Then y is equal to one of the elements $x^{(e+nu)/d}$ with $0 \le u < d$. So if d is not too large, this solves the DLP, and if d is large, we can rerun Pollard's algorithm to find another relation between x and y.

Remark 5.4.1. The proof of (XI.5.3) and its subsequent application to Pollard's ρ algorithm (XI.5.4) rely on two heuristic assumptions. First, they assume that iterations of certain self-maps behave as if they were random mixing maps. Second, they assume that the resulting collision is nondegenerate, e.g., in the notation of (XI.5.4), if $n \mid \delta_i - \beta_i$, then the algorithm yields no information. For a proof that the running time of Pollard's algorithm is $O(\sqrt{n}\,)$, see [126, 174], and for a rigorous analysis of the nondegeneracy assumption, see [175].

Remark 5.4.2. Shanks's and Pollard's algorithms (XI.5.2), (XI.5.4) apply to (virtually) any group and show that the DLP in a cyclic group G of order n can be solved in $O(\sqrt{n}\,)$ steps. Now imagine that you are given a black box that performs the group operations in G. This means that you may feed any two group elements x_1 and x_2 into the box and it will compute for you the value of their product x_1x_2, but you have no knowledge of how the computation is performed. In this situation Shoup [253] has shown that any algorithm that solves the DLP in G takes on average at least $O(\sqrt{n}\,)$ steps. Thus despite the fact that the group law on an elliptic curve is far from being a black box, the best known algorithms to solve the ECDLP are qualitatively no better than a black box algorithm.

XI.6 Solving the Elliptic Curve Discrete Logarithm Problem: Special Cases

The fastest known algorithms that solve the ECDLP on all elliptic curves are collision algorithms such as (XI.5.2) and (XI.5.4). However, not all elliptic curves are created

equal. Menezes, Okamoto, and Vanstone [168] suggested using the Weil pairing to reduce the ECDLP to an easier DLP in the multiplicative group of a finite field. (An alternative reduction using the Tate–Lichtenbaum pairing was suggested by Frey and Rück [91].)

Definition. Let $\mathbb{F}_q$ be a finite field, and let $N \ge 1$ be an integer. The *embedding degree of N in $\mathbb{F}_q$* is the smallest integer $d \ge 1$ such that

$$\boldsymbol{\mu}_N \subset \mathbb{F}_{q^d}^*.$$

Since $\mathbb{F}_{q^d}^*$ is a cyclic group of order $q^d - 1$, this is equivalent to d being the smallest integer satisfying

$$q^d \equiv 1 \pmod{N}.$$

Proposition 6.1. (MOV Algorithm [168]) *Let $E/\mathbb{F}_q$ be an elliptic curve, let $P, Q \in E(\mathbb{F}_q)$ be points of prime order N, and let d be the embedding degree of N in $\mathbb{F}_q$. Assume that $\gcd(q-1, N) = 1$. Then there is a polynomial-time algorithm that reduces the ECDLP for P and Q to the DLP in $\mathbb{F}_{q^d}^*$.*

PROOF. We are looking for an integer m such that $Q = [m]P$. We choose a point $T \in E[N](\bar{\mathbb{F}}_q)$ such that P and T generate $E[N]$. Then the value of the Weil pairing $e_N(P,T)$ is a primitive N^{th} root of unity (III.8.1.1), so by definition of embedding degree we have $e_N(P,T) \in \mathbb{F}_{q^d}^*$. Linearity of the Weil pairing (III.8.1) gives

$$e_N(Q,T) = e_N\big([m]P, T\big) = e_N(P,T)^m.$$

We know the values of points P, Q, and T, so if we can solve the discrete logarithm problem

$$e_N(Q,T) = e_N(P,T)^m$$

in $\mathbb{F}_{q^d}^*$, then we recover the value of m, which also solves the ECDLP for P and Q.

The running time of the MOV algorithm is determined by how long it takes to find the point T and how long it takes to compute the Weil pairing values $e_N(Q,T)$ and $e_N(P,T)$. The Weil pairing computations are not a problem, since they may be computed using Miller's (linear-time) algorithm as described in (XI §8). A key fact, which we prove below (XI.6.2), is that $E[N] \subset E(\mathbb{F}_{q^d})$, where d is the embedding degree. Thus all computations may be done in the field $\mathbb{F}_{q^d}$. To construct an appropriate point $T \in E(\mathbb{F}_{q^d})$, we randomly choose points T in $E(\mathbb{F}_{q^d})$ of order N until we find one such that $e_N(P,T)$ is a primitive N^{th} root of unity.[8] □

Lemma 6.2. *Let $E/\mathbb{F}_q$ be an elliptic curve, let $N \ge 1$ be an integer satisfying $\gcd(q-1, N) = 1$, let d be the embedding degree of N in $\mathbb{F}_q$, and suppose that $E(\mathbb{F}_q)$ contains a point of exact order N. Then $E[N] \subset E(\mathbb{F}_{q^d})$.*

[8] We first compute $n = \#E(\mathbb{F}_{q^d})$, which takes polynomial time (XI.3.1). Then, since N is typically a large prime, most points $S \in E(\mathbb{F}_{q^d})$ have the property that $T = [n/N^2]S$ has order N.

PROOF. Let $P \in E(\mathbb{F}_q)$ be the given point of exact order N defined over $\mathbb{F}_q$, and choose a point $T \in E[N]$ such that $\{P, T\}$ is a basis for $E[N]$. Let $\phi \in G_{\mathbb{F}_q/\mathbb{F}_q}$ be the q-power Frobenius map. Since $P \in E(\mathbb{F}_q)$, we have

$$P^\phi = P \qquad \text{and} \qquad T^\phi = [a]P + [b]T \quad \text{for some } a, b \in \mathbb{Z}/N\mathbb{Z}.$$

Using basic properties of the Weil pairing (III.8.1), we find that

$$\begin{aligned} e_N(P,T)^q &= e_N(P,T)^\phi = e_N(P^\phi, T^\phi) \\ &= e_N(P, [a]P + [b]T) = e_N(P,P)^a e_N(P,T)^b = e_N(P,T)^b. \end{aligned}$$

Since $e_N(P,T)$ is a primitive N^{th} root of unity, cf. (III.8.1.1), this implies that

$$b = q \bmod N.$$

Thus $T^\phi = [a]P + [q]T$. Applying ϕ repeatedly to T and using the fact that ϕ fixes P gives

$$T^{\phi^d} = \bigl[a(1 + q + q^2 + \cdots + q^{d-1})\bigr]P + [q^d]T.$$

By definition of embedding degree, we have $q^d \equiv 1 \pmod N$, so $[q^d]T = T$. Further, the assumption that $\gcd(q-1, N) = 1$ implies that

$$1 + q + q^2 + \cdots + q^{d-1} \equiv 0 \pmod N,$$

so $[1+q+q^2+\cdots+q^{d-1}]P = O$. Therefore $T^{\phi^d} = T$, which proves that $T \in E(\mathbb{F}_{q^d})$. □

Remark 6.3. Under plausible assumptions, Balasubramanian and Koblitz [13] show that for most elliptic curves $E/\mathbb{F}_q$, if N is a large prime divisor of $\#E(\mathbb{F}_q)$, then the embedding degree of N in $\mathbb{F}_q$ is proportional to N. Hence for a randomly chosen $E/\mathbb{F}_q$, the MOV algorithm reduces the ECDLP in $E(\mathbb{F}_q)$ to a much harder DLP in $\mathbb{F}_{q^d}$. However, there are special cases for which the embedding degree is small, as in the following example. (See also (XI.9.8).)

Example 6.4. Let $p \geq 5$ be prime, and let $E/\mathbb{F}_p$ be a supersingular elliptic curve. We can compute embedding degrees for E using Exercise 5.15, which implies that

$$\#E(\mathbb{F}_p) = p + 1.$$

Suppose that $P \in E(\mathbb{F}_p)$ is a point of exact order N. Then N divides $\#E(\mathbb{F}_p)$, so $p \equiv -1 \pmod N$. Hence $p^2 \equiv 1 \pmod N$, so N has embedding degree 2 in $\mathbb{F}_p$. Thus the ECDLP on a supersingular curve over $\mathbb{F}_p$ can be reduced to solving the DLP in $\mathbb{F}_{p^2}^*$, for which there are subexponential algorithms. This militates against using supersingular curves in most cryptographic settings. However, we will see later (XI §7) that there are cryptographic applications that make use of the Weil pairing and low embedding degrees. For these applications, supersingular curves may be used; we simply must ensure that it is computationally infeasible to solve the associated DLP in $\mathbb{F}_{p^2}^*$.

The MOV algorithm (XI.6.1) shows that the ECDLP on an elliptic curve with low embedding degree may be reduced to a potentially easier DLP in the multiplicative group of a finite field. We next describe a special situation in which the ECDLP is reduced to an essentially trivial additive DLP

Proposition 6.5. (Semaev [228], Satoh–Araki [218], Smart [269]) *Let $p \ge 3$ and let $E/\mathbb{F}_p$ be an elliptic curve satisfying*

$$\#E(\mathbb{F}_p) = p.$$

(*Such curves are called* anomalous.) *The following algorithm solves the ECDLP in $E(\mathbb{F}_p)$.*

(1) Let $P, Q \in E(\mathbb{F}_p)$ be nonzero points satisfying $Q = [m]P$, where the integer m is not known.
(2) Choose an elliptic curve $E'/\mathbb{Q}_p$ whose reduction modulo p is $E/\mathbb{F}_p$.
(3) Use Hensel's lemma to lift the points P, Q to points $P', Q' \in E'(\mathbb{Q}_p)$.
(4) The points $[p]P'$ and $[p]Q'$ are in the formal group $E'_1(\mathbb{Q}_p)$. Let

$$\log_E : E'_1(\mathbb{Q}_p) \longrightarrow \hat{\mathbb{G}}_a(p\mathbb{Z}_p) \cong p\mathbb{Z}_p^+$$

be the formal logarithm map (IV §5, IV.6.4), and compute

$$pa = \log_E\bigl([p]P'\bigr) \in p\mathbb{Z}_p \qquad \text{and} \qquad pb = \log_E\bigl([p]Q'\bigr) \in p\mathbb{Z}_p.$$

(5) Then $m \equiv a^{-1}b \pmod{p}$.

PROOF. Using the fact that $\#E(\mathbb{F}_p) = p$, we have

$$\widetilde{[p]P'} = [p]P = O \quad \text{and} \quad \widetilde{[p]Q'} = [p]Q = O \qquad \text{in } E(\mathbb{F}_p),$$

so $[p]P'$ and $[p]Q'$ are in the kernel of reduction modulo p. Hence they are in the formal group $E'_1(\mathbb{Q}_p)$. Similarly, if we let $R' = Q' - [m]P'$, then the reduction of R' modulo p is

$$\tilde{R}' = \tilde{Q}' - [m]\tilde{P}' = Q - [m]P = O \qquad \text{in } E(\mathbb{F}_p),$$

so $R' \in E'_1(\mathbb{Q}_p)$. We now compute

$$\begin{aligned}
\log_E\bigl([p]Q'\bigr) &= \log_E\bigl([p]([m]P' + R')\bigr) && \text{since } R' = Q' - [m]P', \\
&= m\log_E\bigl([p]P'\bigr) + p\log_E(R') && \text{valid since } [p]P', R' \in E_1(\mathbb{Q}_p), \\
&\equiv m\log_E\bigl([p]P'\bigr) \pmod{p^2} && \text{since } \log_E(R') \in p\mathbb{Z}_p.
\end{aligned}$$

Substituting $\log_E\bigl([p]P'\bigr) = pa$ and $\log_E\bigl([p]Q'\bigr) = pb$ as in step (4) of the algorithm gives $pb \equiv mpa \pmod{p^2}$, so $m \equiv a^{-1}b \pmod{p}$. □

Remark 6.6. The algorithm described in (XI.6.5) may seem impractical, since it requires lifting points in $E(\mathbb{F}_p)$ to points in $E'(\mathbb{Q}_p)$. However, an examination of the proof shows that we need only lift points modulo p^2 and compute formal logarithms in $\hat{\mathbb{G}}_a(p\mathbb{Z}_p)/\hat{\mathbb{G}}_a(p^2\mathbb{Z}_p) \cong p\mathbb{Z}_p/p^2\mathbb{Z}_p$. Thus we work on an elliptic curve over the ring $\mathbb{Z}/p^2\mathbb{Z}$, or in fancier language, on an elliptic scheme; see (XI.2.4.1).

Example 6.7. We work over the field $\mathbb{F}_{127}$ and consider the anomalous curve and points

$$E : y^2 = x^3+19x+112, \quad P = (106, 72) \in E(\mathbb{F}_{127}), \quad Q = (12, 121) \in E(\mathbb{F}_{127}).$$

We take the same equation to be our lift of E to $\mathbb{Z}/127^2\mathbb{Z}$, and we lift the points P and Q to

$$P' = (106, 13026) \in E(\mathbb{Z}/127^2\mathbb{Z}) \quad \text{and} \quad Q' = (12, 5201) \in E(\mathbb{Z}/127^2\mathbb{Z}).$$

We now want to compute multiples of P' and Q' working modulo 127^2. In order to avoid noninvertible denominators, it is convenient to make the change of variables $z = -x/y$ and $w = -1/y$. This has the effect of moving O to $(z, w) = (0, 0)$; cf. (IV §1). The equation for E now reads

$$E : w = z^3 + 19zw^2 + 112w^3.$$

We use the double-and-add algorithm, working modulo 127^2 with (z, w)-coordinates, to compute

$$[127]P' = (12319, 0) \in E(\mathbb{Z}/127^2\mathbb{Z}) \quad \text{and} \quad [127]Q' = (2159, 0) \in E(\mathbb{Z}/127^2\mathbb{Z}).$$

The elliptic logarithm for $y^2 = x^3 + Ax + B$ starts $\log_E(z) = z + \frac{2}{5}Az^5 + \cdots$, so since we are working modulo 127^2, it suffices to use $\log_E(z) \approx z$. Thus

$$\begin{aligned} \log_E\bigl([127]P'\bigr) &\equiv 12319 \equiv 97 \cdot 127 \pmod{127^2}, \\ \log_E\bigl([127]Q'\bigr) &\equiv 2159 \equiv 17 \cdot 127 \pmod{127^2}. \end{aligned}$$

Finally, we compute $m \equiv 97^{-1} \cdot 17 \equiv 46 \pmod{127}$, which is the desired discrete logarithm. We can check our answer by verifying that $[46]P = Q$ in $E(\mathbb{F}_{127})$.

XI.7 Pairing-Based Cryptography

The Diffie–Hellman key exchange algorithm allows two people to exchange an unspecified piece of data. It was a long-standing problem to find a method that allows three people to perform a similar exchange. Joux found a solution using the Weil pairing

$$e_N : E[N] \times E[N] \longrightarrow \boldsymbol{\mu}_N,$$

which we recall (III §8) is nondegenerate, bilinear, and *alternating*. For cryptographic applications we need a pairing that is nondegenerate on a cyclic subgroup of $E[N]$, but unfortunately the Weil pairing is trivial on such subgroups. One way around this difficulty is to use a curve that admits an isogeny

$$\phi : E \longrightarrow E$$

such that $E[N]$ has a basis of the form $\{T, \phi(T)\}$. In the cryptographic literature, the map ϕ is called a *distortion map*; see [51, §24.2.1b] or [116, §5.9.2]. Using a distortion map, the modified Weil pairing

$$\langle\,\cdot\,,\,\cdot\,\rangle : E[N] \times E[N] \longrightarrow \boldsymbol{\mu}_N, \qquad \langle P, Q\rangle = e_N\bigl(P, \phi(Q)\bigr)$$

has the property that $\langle T, T\rangle$ is a primitive N^{th} root of unity.

Example 7.1. Let E be the elliptic curve $y^2 = x^3 + x$ having complex multiplication by $\mathbb{Z}[i]$, and let ϕ be the isogeny

$$\phi : E \longrightarrow E, \qquad \phi(x, y) = [i](x, y) = (-x, iy).$$

Then ϕ is a distortion map on $E[N]$ for all primes N satisfying $N \equiv 3 \pmod 4$. To see this, let $T \in E[N]$ be a point of exact order N, and suppose that some linear combination of T and $\phi(T)$ is zero. Then

$$\begin{aligned} [a]T + [b]\phi(T) = O \quad &\Longleftrightarrow \quad [a + bi](T) = O \\ &\Longrightarrow \quad [a^2 + b^2](T) = O \\ &\Longrightarrow \quad a^2 + b^2 \equiv 0 \pmod N \\ &\Longrightarrow \quad a \equiv b \equiv 0 \pmod N, \end{aligned}$$

where the last line follows from the assumption that $N \equiv 3 \pmod 4$. (See Exercise 3.26 for another example of a distortion map.)

An alternative to the modified Weil pairing is the Tate–Lichtenbaum pairing

$$\tau : E(K)/NE(K) \times E(K)[N] \longrightarrow K^*/(K^*)^N,$$

which we discuss in (XI §9). If K is a finite (or local) field, then under appropriate conditions the Tate–Lichtenbaum pairing is nondegenerate, in which case we can define a nondegenerate pairing by

$$\langle\,\cdot\,,\,\cdot\,\rangle : E(\mathbb{F}_q)[N] \times E(\mathbb{F}_q)[N] \longrightarrow \boldsymbol{\mu}_N, \qquad \langle P, Q\rangle = \tau(P, Q)^{(q-1)/N}.$$

From a practical perspective, the primary advantage of the Tate–Lichtenbaum pairing over the Weil pairing is that the former can be computed in roughly half the time that it takes to compute the latter (XI.9.3.2). In any case, there is a double-and-add algorithm for both pairings, so they are both easy to compute; see (XI §8) for details.

Tripartite Diffie–Hellman Key Exchange 7.2. (Joux [120]) *The following procedure allows Alice, Bob, and Carl to securely exchange a a piece of information whose value none of them knows in advance*:

(1) Alice, Bob, and Carl agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, a prime N, and a point $T \in E(\mathbb{F}_q)[N]$ such that there is a bilinear pairing

$$\langle\,\cdot\,,\,\cdot\,\rangle : E(\mathbb{F}_q)[N] \times E(\mathbb{F}_q)[N] \longrightarrow \boldsymbol{\mu}_N$$

with the property that $\langle T, T\rangle$ is a primitive N^{th} root of unity.

(2) Alice, Bob, and Carl choose secret integers a, b, and c, respectively, and they compute

$$\overbrace{A = [a]T}^{\text{Alice computes this}}, \qquad \overbrace{B = [b]T}^{\text{Bob computes this}}, \qquad \overbrace{C = [c]T}^{\text{Carl computes this}}.$$

(3) Alice, Bob, and Carl publish the values of A, B, and C.
(4) Alice, Bob, and Carl compute, respectively,

$$\overbrace{\langle B,C\rangle^{a}}^{\text{Alice computes this}},\qquad \overbrace{\langle A,C\rangle^{b}}^{\text{Bob computes this}},\qquad \overbrace{\langle A,B\rangle^{c}}^{\text{Carl computes this}}.$$

(5) Alice, Bob, and Carl have now shared the value $\langle T,T\rangle^{abc}$.

Remark 7.3. If Eve can solve the ECDLP, then she can certainly break tripartite Diffie–Hellman key exchange, since for example she can recover Alice's secret multiplier a from the publicly available points T and $A = [a]T$. However, Eve can also break the system if she can solve the discrete logarithm problem in the multiplicative group $\mathbb{F}_q^*$. Thus Eve knows the values of T, A, and B, so if she can solve the DLP

$$\langle T,T\rangle^{m} = \langle A,B\rangle$$

in $\mathbb{F}_q^*$, she recovers the value $m = ab$. From this she obtains Alice, Bob, and Carl's shared value by computing $\langle T,C\rangle^{ab}$.

The DLP in $\mathbb{F}_q^*$ can be solved in subexponential time (XI.4.1b), so it is currently significantly easier to solve the DLP in $\mathbb{F}_q^*$ than it is to solve the ECDLP in $E(\mathbb{F}_q)$. Thus tripartite Diffie–Hellman key exchange and other pairing-based cryptographic algorithms require that q be sufficiently large to preclude the solution of the DLP in $\mathbb{F}_q^*$.

Another cryptographic application of pairings on elliptic curves is a digital signature scheme that has extremely short signatures, as described in the following result of Boneh, Lynn, and Shacham [24]; see also [51, §24.1.3].

Theorem 7.4. *The following procedure allows Alice to sign a digital document and Bob to verify that the signature is valid.*

(1) Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, a prime N, and a point $T \in E(\mathbb{F}_q)[N]$ such that there is a bilinear pairing

$$\langle\,\cdot\,,\,\cdot\,\rangle : E(\mathbb{F}_q)[N] \times E(\mathbb{F}_q)[N] \longrightarrow \boldsymbol{\mu}_N$$

with the property that $\langle T,T\rangle$ is a primitive N^{th} root of unity.
(2) Alice selects a secret integer a and computes the point $A = [a]T \in E(\mathbb{F}_q)$.
(3) Alice publishes the point A. This is her *public verification key*. The secret multiplier a is her *private signing key*.
(4) Alice chooses a digital document $D \in E(\mathbb{F}_q)$ to sign.[9] She computes and publishes the signature

$$S = [a]D.$$

(5) Bob accepts the signature as valid if the two quantities

$$\langle A,D\rangle \quad\text{and}\quad \langle T,S\rangle$$

are equal.

[9] As with ECDSA (XI.4.6), the point D is really a hash of the actual document. See [51, §24.2.5].

PROOF. Assuming that Alice has constructed A and S as in steps (2) and (4), bilinearity of the pairing yields

$$\langle A, D\rangle = \langle [a]T, D\rangle = \langle T, D\rangle^a \quad \text{and} \quad \langle T, S\rangle = \langle T, [a]D\rangle = \langle T, D\rangle^a,$$

so Bob accepts the signature. □

Remark 7.5.1. The reason that (XI.7.4) is called a short signature scheme is because the signature consists of only a single point in $E(\mathbb{F}_q)$, so with point compression (XI.4.3.5), a single number in $\mathbb{F}_q$. Thus (XI.7.4) gives signatures that are half the size of those produced by ECDSA (XI.4.6). (The use of hyperelliptic curves allows the size of q to be further reduced; see [51].)

Remark 7.5.2. Recall (XI.4.3.4) that the elliptic curve Diffie–Hellman problem asks for the value of $[ab]P$, given the three points P, $[a]P$, and $[b]P$. Potentially easier is the following decision version of this problem:

Decision Diffie–Hellman Problem
Given four points P, $[a]P$, $[b]P$, and Q in $E(\mathbb{F}_q)$, determine whether Q is equal to $[ab]P$.

The short signature scheme (XI.7.4) uses the fact that if E is an elliptic curve with a nondegenerate bilinear pairing, then the decision Diffie–Hellman problem is easy to solve, since $Q = [ab]P$ if and only if $\langle [a]P, [b]P\rangle = \langle Q, P\rangle$.

Remark 7.6. There are a number of other cryptographic constructions that use bilinear pairings on elliptic curves and that depend for their security on the difficulty of solving both the ECDLP in $E(\mathbb{F}_q)$ and the DLP in $\mathbb{F}_q^*$. We mention in particular *ID-based cryptography*, in which Alice may use an arbitrary character string as her public key. For example, her public key could be her email address. To send a message, Bob combines Alice's public key with a universal public key available from some trusted authority. The trusted authority also provides Alice with a personal private key that goes with her ID-based public key. The idea of ID-based cryptosystems was proposed by Shamir [245] in 1985, and a practical system using elliptic curves and pairings was devised by Boneh and Franklin [23] in 2001. For further details, see for example [51, §24.1.2] or [116, §5.10.2].

XI.8 Computing the Weil Pairing

The abstract definition of the Weil pairing requires functions having specified divisors. In this section we describe a double-and-add algorithm due to Victor Miller that computes such functions in linear time. Miller's algorithm makes pairings practical for use in applications such as cryptography.

Theorem 8.1. *Let E be an elliptic curve given by a Weierstrass equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

and let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be nonzero points on E.

(1) Set $T = P$ and $f = 1$
(2) Loop $i = t - 1$ down to 0
(3) Set $f = f^2 \cdot h_{T,T}$
(4) Set $T = 2T$
(5) If $\epsilon_i = 1$
(6) Set $f = f \cdot h_{T,P}$
(7) Set $T = T + P$
(8) End If
(9) End i Loop
(10) Return the value f

Figure 11.6: Miller's algorithm.

(a) *Let λ be the slope of the line connecting P and Q, or the slope of the tangent line to E at P if $P = Q$. (If the line is vertical, set $\lambda = \infty$.) Define a function $h_{P,Q}$ on E as follows*:

$$h_{P,Q} = \begin{cases} \dfrac{y - y_P - \lambda(x - x_P)}{x + x_P + x_Q - \lambda^2 - a_1\lambda + a_2} & \text{if } \lambda \neq \infty, \\ x - x_P & \text{if } \lambda = \infty, \\ 1 & \text{if } P = O \text{ or } Q = O. \end{cases}$$

Then

$$\operatorname{div}(h_{P,Q}) = (P) + (Q) - (P + Q) - (O).$$

(b) **Miller's algorithm.** *Let $N \geq 1$ and write the binary expansion of N as*

$$N = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \cdots + \epsilon_t \cdot 2^t \qquad \text{with } \epsilon_i \in \{0, 1\} \text{ and } \epsilon_t \neq 0.$$

The algorithm described in Figure 11.6 returns a function f_P whose divisor satisfies

$$\operatorname{div}(f_P) = N(P) - ([N]P) - (N - 1)(O),$$

where the functions $h_{T,T}$ and $h_{T,P}$ used by the algorithm are as defined in (a). *In particular, if $P \in E[N]$, then* $\operatorname{div}(f_P) = N(P) - N(O)$.

PROOF. (a) Suppose first that $\lambda \neq \infty$, and let $y = \lambda x + \nu$ be the line through P and Q, or the tangent line at P if $P = Q$. This line intersects E at the three points P, Q, and $-P - Q$, so

$$\operatorname{div}(y - \lambda x - \nu) = (P) + (Q) + (-P - Q) - 3(O).$$

Vertical lines intersect E at points and their negatives, so

$$\operatorname{div}(x - x_{P+Q}) = (P + Q) + (-P - Q) - 2(O).$$

It follows that

$$h_{P,Q} = \frac{y - \lambda x - \nu}{x - x_{P+Q}}$$

has the divisor stated in (a). Finally, the addition formula (III.2.3d) tells us that $x_{P+Q} = \lambda^2 + a_1\lambda - a_2 - x_P - x_Q$, and we can eliminate ν from the numerator of $h_{P,Q}$ using $y_P = \lambda x_P + \nu$.

If $\lambda = \infty$, then $P+Q = O$, so we need $h_{P,Q}$ to have divisor $(P)+(-P)-2(O)$. The function $x - x_P$ has this divisor.

(b) This is a standard double-and-add algorithm, similar to (XI.1.1). The key to analyzing the algorithm comes from (a), which tells us that the functions $h_{T,T}$ and $h_{T,P}$ used in steps (3) and (6) have divisors

$$\begin{aligned}\operatorname{div}(h_{T,T}) &= 2(T) - (2T) - (O),\\ \operatorname{div}(h_{T,P}) &= (T) + (P) - (T+P) - (O).\end{aligned}$$

We consider the effect of executing the i loop, steps (2)–(9), for a given value of i. At the start of the loop the variables T and f have initial values T_i^{start} and f_i^{start}, and at the end of (one execution of) the loop they have final values T_i^{end} and f_i^{end}. We start with T. During the loop, the value of T is doubled and then, if $\epsilon_i = 1$, the value is incremented by P. This gives the relation

$$T_i^{\text{end}} = 2T_i^{\text{start}} + \epsilon_i P.$$

Similarly, the value of f is squared, multiplied by $h_{T,T}$, and then, if $\epsilon_i = 1$, it is multiplied by $h_{2T,P}$. (Note that the value of T has been doubled in step (4) before it is used in step (6).) This yields

$$f_i^{\text{end}} = (f_i^{\text{start}})^2 \cdot h_{T_i^{\text{start}},T_i^{\text{start}}} \cdot h_{2T_i^{\text{start}},P}^{\epsilon_i}.$$

Hence the divisors of f_i^{start} and f_i^{end} are related by

$$\begin{aligned}\operatorname{div}(f_i^{\text{end}}) &= 2\operatorname{div}(f_i^{\text{start}}) + \operatorname{div}(h_{T_i^{\text{start}},T_i^{\text{start}}}) + \epsilon_i \operatorname{div}(h_{2T_i^{\text{start}},P})\\ &= 2\operatorname{div}(f_i^{\text{start}}) + \big(2(T_i^{\text{start}}) - (2T_i^{\text{start}}) - (O)\big)\\ &\qquad + \epsilon_i\big((2T_i^{\text{start}}) + (P) - (2T_i^{\text{start}} + P) - (O)\big)\\ &= 2\operatorname{div}(f_i^{\text{start}}) + 2(T_i^{\text{start}}) - (2T_i^{\text{start}} + \epsilon_i P) + \epsilon_i(P) - (1+\epsilon_i)(O)\\ &\qquad\qquad \text{since } \epsilon_i \in \{0,1\},\\ &= 2\operatorname{div}(f_i^{\text{start}}) + 2(T_i^{\text{start}}) - (T_i^{\text{end}}) + \epsilon_i(P) - (1+\epsilon_i)(O).\end{aligned}$$

Of course, the final values of T and f after a given iteration of the i loop are the initial values for the next iteration, i.e., $T_i^{\text{end}} = T_{i-1}^{\text{start}}$ and $f_i^{\text{end}} = f_{i-1}^{\text{start}}$. (Note that the i loop decrements from $t-1$ to 0.) This allows us to rewrite the recurrences for T and f as

$$\begin{aligned}T_{i-1}^{\text{start}} - 2T_i^{\text{start}} &= \epsilon_i P,\\ \operatorname{div}(f_{i-1}^{\text{start}}) - 2\operatorname{div}(f_i^{\text{start}}) &= 2(T_i^{\text{start}}) - (T_{i-1}^{\text{start}}) + \epsilon_i(P) - (1+\epsilon_i)(O).\end{aligned}$$

These formulas are designed to telescope when they are summed. For example, when the algorithm terminates, the final value of T is

$$\begin{aligned}
T_0^{\text{end}} &= \epsilon_0 P + 2T_0^{\text{start}} \\
&= \epsilon_0 P + \left[\sum_{i=1}^{t-1} 2^i (T_{i-1}^{\text{start}} - 2T_i^{\text{start}})\right] + 2^t T_{t-1}^{\text{start}} \\
&= \epsilon_0 P + \sum_{i=1}^{t-1} 2^i \epsilon_i P + 2^t T_{t-1}^{\text{start}} && \text{using the recurrence for } T_i^{\text{start}}, \\
&= \sum_{i=0}^{t} 2^i \epsilon_i P && \text{since } T_{t-1}^{\text{start}} = P \text{ and } \epsilon_t = 1, \\
&= NP. && \text{since } N = \textstyle\sum \epsilon_i 2^i.
\end{aligned}$$

Finally, we compute the divisor of the function f returned by Miller's algorithm:

$$\begin{aligned}
&\operatorname{div}(f_0^{\text{end}}) \\
&= 2\operatorname{div}(f_0^{\text{start}}) + 2(T_0^{\text{start}}) - (T_0^{\text{end}}) + \epsilon_0(P) - (1+\epsilon_0)(O) \\
&= \left[\sum_{i=1}^{t-1} 2^i \big(\operatorname{div}(f_{i-1}^{\text{start}}) - 2\operatorname{div}(f_i^{\text{start}})\big)\right] + 2(T_0^{\text{start}}) - (NP) + \epsilon_0(P) - (1+\epsilon_0)(O) \\
&\qquad\qquad \text{since } f_{t-1}^{\text{start}} = 1 \text{ and } T_0^{\text{end}} = NP, \\
&= \left[\sum_{i=1}^{t-1} 2^i \big(2(T_i^{\text{start}}) - (T_{i-1}^{\text{start}}) + \epsilon_i(P) - (1+\epsilon_i)(O)\big)\right] + 2(T_0^{\text{start}}) \\
&\qquad\qquad - (NP) + \epsilon_0(P) - (1+\epsilon_0)(O) \\
&= 2^t (T_{t-1}^{\text{start}}) + \sum_{i=0}^{t-1} 2^i \epsilon_i(P) - \sum_{i=0}^{t-1} 2^i (1+\epsilon_i)(O) - (NP) \\
&= N(P) - (N-1)(O) - (NP) \qquad \text{since } T_{t-1}^{\text{start}} = P,\ \epsilon_t = 1, \text{ and } N = \textstyle\sum \epsilon_i 2^i.
\end{aligned}$$

This completes the proof that the function returned by Miller's algorithm has the stated divisor. □

Remark 8.2. Let $P \in E[N](K)$. Miller's algorithm (XI.8.1b) tells us how to compute a function $f_P \in K(E)$ with divisor $N(P) - N(O)$. Further, if $R \in E$ is any point, we can use Miller's algorithm to directly evaluate $f_P(R)$ by evaluating the functions $h_{T,T}(R)$ and $h_{T,P}(R)$ in steps (3) and (6). This allows us to compute the Weil pairing $e_N(P,Q)$ via the alternative definition of $e_N(P,Q)$ from Exercise 3.16. We choose any point $S \in E$ not in the subgroup generated by P and Q, and then

$$e_N(P,Q) = \frac{f_P(Q+S)}{f_P(S)} \Big/ \frac{f_Q(P-S)}{f_Q(-S)}\,.$$

The right-hand side of this formula can be computed using four applications of Miller's algorithm (XI.8.1b). For added efficiency, the two values $f_P(Q+S)$ and $f_P(S)$ of f_P may be computed simultaneously, and similarly for $f_Q(P-S)$ and $f_Q(-S)$.

Example 8.3. Let $E/\mathbb{F}_{631}$ be the elliptic curve

$$y^2 = x^3 + 30x + 34.$$

We have $E(\mathbb{F}_{631}) \cong \mathbb{Z}/5\mathbb{Z} \times \mathbb{Z}/130\mathbb{Z}$, and it is easy to check that the points $P = (36, 60)$ and $Q = (121, 387)$ generate $E(\mathbb{F}_{631})[5]$. In order to compute the Weil pairing with Miller's algorithm, we use the auxiliary point $S = (0, 36) \in E(\mathbb{F}_{631})$. The point S has order 10, so it is not in the subgroup spanned by P and Q. Miller's algorithm gives

$$f_P(Q+S) = 103, \quad f_P(S) = 219, \quad f_Q(P-S) = 284, \quad f_Q(-S) = 204.$$

Hence

$$e_5(P,Q) = \frac{103}{219} \Big/ \frac{284}{204} = 242 \in \mathbb{F}_{631}.$$

We check that $(242)^5 = 1$, so $e_5(P,Q)$ is indeed a fifth root of unity in $\mathbb{F}_{631}$.

Remark 8.4. As an alternative to the Weil pairing, cryptographers often use the Tate–Lichtenbaum pairing described in (XI §9). Miller's algorithm (XI.8.1b) can also be used to compute the Tate–Lichtenbaum pairing (XI.9.3.2).

Remark 8.5. Another linear-time algorithm to compute the Weil and Tate–Lichtenbaum pairings, due to Shipsey and Stange, makes use of elliptic divisibility sequences (Exercises 3.34–3.36) and elliptic nets. See [252, 270, 271] for details.

XI.9 The Tate–Lichtenbaum Pairing

The Weil pairing is often used to define other pairings on elliptic curves. In this section we describe the Tate–Lichtenbaum pairing, which has both theoretical and cryptographic applications. See (C §17) and Exercise 10.24 for other instances of pairings on elliptic curves.

Definition. Let E/K be an elliptic curve, and let $N \ge 1$ be an integer that is prime to $p = \mathrm{char}(K)$ if $p > 0$. The *Tate–Lichtenbaum pairing*

$$\tau : \frac{E(K)}{NE(K)} \times E(K)[N] \longrightarrow \frac{K^*}{(K^*)^N}$$

is defined as follows. Let $P \in E(K)$ and $T \in E(K)[N]$. Choose a point $Q \in E(\bar{K})$ satisfying $[N]Q = P$. The map

$$G_{\bar{K}/K} \longrightarrow \boldsymbol{\mu}_N, \qquad \sigma \longrightarrow e_N(Q^\sigma - Q, T),$$

is a 1-cocycle (see below), so it represents an element of $H^1(G_{\bar{K}/K}, \boldsymbol{\mu}_N)$. Hilbert's Theorem 90 (B.2.5c) says that the connecting homomorphism

$$K^*/(K^*)^N \longrightarrow H^1(G_{\bar{K}/K}, \boldsymbol{\mu}_N)$$

is an isomorphism; hence there exists an element $\alpha \in K^*$, unique up to N^{th} powers, with the property that

$$e_N(Q^\sigma - Q, T) = \sqrt[N]{\alpha}^{\,\sigma} / \sqrt[N]{\alpha} \qquad \text{for all } \sigma \in G_{\bar{K}/K}.$$

The value of the Tate–Lichtenbaum pairing is then

$$\tau(P,T) = \alpha \bmod (K^*)^N.$$

Proposition 9.1. *The Tate–Lichtenbaum pairing is a well-defined bilinear pairing.*

PROOF. Let $\xi(\sigma) = e_N(Q^\sigma - Q, T)$ be the given map $\xi : G_{\bar{K}/K} \to \boldsymbol{\mu}_N$. We use basic properties of the Weil pairing (III.8.1ad) and the assumption that $T \in E(K)[N]$ to verify that ξ is a 1-cocycle:

$$\begin{aligned}
\xi(\sigma\tau) &= e_N(Q^{\sigma\tau} - Q, T) \\
&= e_N(Q^{\sigma\tau} - Q^\tau + Q^\tau - Q, T) \\
&= e_N(Q^\sigma - Q, T)^\tau e_N(Q^\tau - Q, T) \\
&= \xi(\sigma)^\tau \xi(\tau).
\end{aligned}$$

Next we show that $\xi(\sigma)$ depends only on P modulo $NE(K)$. If we replace P by $P + NR$ for some $R \in E(K)$, then Q is replaced by $Q + R$. Since R is defined over K, we have

$$(Q+R)^\sigma - (Q+R) = Q^\sigma - Q \qquad \text{for all } \sigma \in G_{\bar{K}/K},$$

so the value $\xi(\sigma)$ does not change.

Suppose that we replace Q by some other point Q' satisfying $[N]Q' = P$. Then the difference $S = Q' - Q$ is in $E[N]$, so

$$\begin{aligned}
e_N(Q'^\sigma - Q', T) &= e_N\big((Q+S)^\sigma - (Q+S), T\big) \\
&= e_N(Q^\sigma - Q, T) e_N(S^\sigma - S, T) \\
&= e_N(Q^\sigma - Q, T) \frac{e_N(S,T)^\sigma}{e_N(S,T)}.
\end{aligned}$$

(Note that $e_N(S,T)$ is well-defined, since $S \in E[N]$, while $e_N(Q,T)$ is not defined, since in general $Q \notin E[N]$.) Thus the $G_{\bar{K}/K}$-to-$\boldsymbol{\mu}_N$ cocycle coming from Q' differs from the $G_{\bar{K}/K}$-to-$\boldsymbol{\mu}_N$ cocycle coming from Q by the $G_{\bar{K}/K}$-to-$\boldsymbol{\mu}_N$ coboundary $\sigma \mapsto e_N(S,T)^\sigma / e_N(S,T)$, so they represent the same cohomology class.

This completes the proof that the Tate–Lichtenbaum pairing is well-defined. The bilinearity is immediate from the bilinearity of the Weil pairing. □

The Weil pairing is defined by evaluating certain functions at certain points. We can do the same for the Tate–Lichtenbaum pairing.

Proposition 9.2. *Let $T \in E(K)[N]$ and choose a function $f \in K(E)$ satisfying*

$$\operatorname{div}(f) = N(T) - N(O) \qquad \text{and} \qquad f \circ [N] \in (K(E)^*)^N.$$

Then for all $P \in E(K) \smallsetminus \{O, T\}$ we have

$$\tau(P,T) = f(P) \bmod (K^*)^N.$$

Remark 9.3.1. Although (XI.9.2) excludes the case of $\tau(T,T)$, we can use bilinearity to compute $\tau(T,T)$ as

$$\tau(T,T) = \frac{\tau(T+Q,T)}{\tau(Q,T)} = \frac{f(T+Q)}{f(Q)}.$$

More generally,

$$\tau(P,T) = \frac{\tau(P+Q,T)}{\tau(Q,T)} = \frac{f(P+Q)}{f(Q)}$$

for any $Q \in E$ such that f is defined and nonzero at both $P+Q$ and Q, i.e., any point $Q \notin \{O, T, -P, T-P\}$. Notice that with this formulation, there is no need to choose f to satisfy $f \circ [N] \in (K(E)^*)^N$, since the divisor relation $\operatorname{div}(f) = N(T) - N(O)$ determines f up to multiplication by a constant, and taking the ratio eliminates the dependence on the constant.

Remark 9.3.2. Miller's algorithm (XI.8.1b) can be used to efficiently compute the Tate–Lichtenbaum pairing, since it gives a linear-time algorithm to compute the value of f. Comparing the formula for the Tate–Lichtenbaum pairing (XI.9.3.1) to the formula for the Weil pairing (XI.8.2), we see that τ requires two values of f, while e_N requires four values of f. Thus the former is twice as efficient as the latter, which is why the Tate–Lichtenbaum pairing is often preferred for real-world cryptographic applications.

PROOF OF (XI.9.2). As explained in the construction of the Weil e_N-pairing (III §8), there are functions $f, g \in \bar{K}(E)$ satisfying

$$\operatorname{div}(f) = N(T) - N(O) \qquad \text{and} \qquad f \circ [N] = g^N,$$

and (II.5.8) implies that we may choose f and g in $K(E)$, since their divisors are $G_{\bar{K}/K}$-invariant. From the definition of e_N we have

$$e_N(Q^\sigma - Q, T) = \frac{g(X + Q^\sigma - Q)}{g(X)} \qquad \text{for } X \in E.$$

In particular, setting $X = Q$ gives

$$e_N(Q^\sigma - Q, T) = \frac{g(Q^\sigma)}{g(Q)} = \frac{g(Q)^\sigma}{g(Q)}.$$

Comparing this formula with the definition of the Tate–Lichtenbaum pairing yields

$$\tau(P,T) = g(Q)^N = f \circ [N](Q) = f(P) \pmod{(K^*)^N}. \qquad \square$$

The Tate–Lichtenbaum pairing has many applications, both theoretical and practical. In cryptography, it is used on elliptic curves over finite fields; see (XI §7). The following result provides an important nondegeneracy criterion in this situation. For applications of the Tate–Lichtenbaum pairing over local and global fields, see for example [149, 177, 281, 286].

Theorem 9.4. *Let $E/\mathbb{F}_q$ be an elliptic curve defined over a finite field, let $N \ge 1$, let $T \in E(\mathbb{F}_q)[N]$ be a point of exact order N, and make the following assumptions*:

(i) $\boldsymbol{\mu}_N \subset \mathbb{F}_q$, *or equivalently,* $q \equiv 1 \pmod{N}$.

(ii) $E(\mathbb{F}_q)[N^2] = \mathbb{Z}T$, *i.e., the only rational N^2-torsion points are the multiples of T.*

Then the Tate–Lichtenbaum pairing is a perfect pairing, and $\tau(T,T)^{(q-1)/N}$ is a primitive N^{th} root of unity in $\mathbb{F}_q^*$.

PROOF. We begin by proving that the Tate–Lichtenbaum pairing is nondegenerate on the left. Let $\phi \in G_{\bar{\mathbb{F}}_q/\mathbb{F}_q}$ be the q-power Frobenius map. Choose another N-torsion point T' so that T and T' generate $E[N]$. We know that $T^\phi = T$, since $T \in E(\mathbb{F}_q)$, and we write

$$T'^\phi = [a]T + [b]T' \qquad \text{for some } a, b \in \mathbb{Z}/N\mathbb{Z}.$$

We use basic properties of the Weil pairing (II.8.1) to compute

$$\begin{aligned} e_N(T,T')^q &= e_N(T,T')^\phi \\ &= e_N(T^\phi, T'^\phi) \\ &= e_N(T,[a]T+[b]T') \\ &= e_N(T,T)^a e_N(T,T')^b \\ &= e_N(T,T')^b. \end{aligned}$$

Since $e_N(T,T')$ is a primitive N^{th} root of unity (cf. (III.8.1.1)), this implies that

$$b \equiv q \pmod{N}.$$

But our assumption that $\boldsymbol{\mu}_N \subset \mathbb{F}_q$ tells us that $q \equiv 1 \pmod{N}$, so the action of Frobenius on T' is given by

$$T'^\phi = [a]T + T' \qquad \text{for some } a \in \mathbb{Z}/N\mathbb{Z}.$$

We claim that $a \in (\mathbb{Z}/N\mathbb{Z})^*$. To see this, let $d = N/\gcd(a,N)$. Then

$$([d]T')^\phi = [da]T + [d]T' = [d]T',$$

so $[d]T' \in E(\mathbb{F}_q)[N]$. But by assumption, the point T generates $E(\mathbb{F}_q)[N]$, while T and T' generate all of $E[N]$. Hence $[d]T' = O$, which implies that $d = N$ and $\gcd(a, N) = 1$.

Suppose now that $P \in E(\mathbb{F}_q)$ satisfies $\tau(P, T) = 1$. What this really means is that in the definition of the Tate–Lichtenbaum pairing, we have $\alpha \in (\mathbb{F}_q^*)^N$. Thus $\sqrt[N]{\alpha} \in \mathbb{F}_q^*$, which implies that $e_N(Q^\phi - Q, T) = 1$. So if we write

$$Q^\phi - Q = [A]T + [B]T',$$

then again using properties of the Weil pairing we find that

$$1 = e_N(Q^\phi - Q, T) = e_N([A]T + [B]T', T) = e_N(T', T)^B.$$

This implies that $B \equiv 0 \pmod{N}$, so $Q^\phi - Q = [A]T$. Now consider the point

$$Q' = Q - [a^{-1}A]T',$$

where a^{-1} is the inverse of a modulo N. Then

$$Q'^\phi = \Big(Q + [A]T\Big) - [a^{-1}A]\Big([a]T + T'\Big) = Q - [a^{-1}A]T' = Q'.$$

Thus $Q' \in E(\mathbb{F}_q)$, so

$$P = [N]Q = [N]Q' \in NE(\mathbb{F}_q).$$

This proves nondegeneracy of the Tate–Lichtenbaum pairing on the left.

We now consider the value of $\tau(T, T)$. Let k denote the order of $\tau(T, T)$ in $\mathbb{F}_q^*/(\mathbb{F}_q^*)^N$. Bilinearity of the Tate–Lichtenbaum pairing implies that

$$\tau([k]T, T) = \tau(T, T)^k = 1,$$

so the nondegeneracy that we already proved implies that $[k]T \in NE(\mathbb{F}_q)$. So we can write $[k]T = [N]S$ for some $S \in E(\mathbb{F}_q)$. The point T has order N, so $[N^2]S = [kN]T = O$, which shows that $S \in E(\mathbb{F}_q)[N^2]$. But by assumption, the only $\mathbb{F}_q$-rational points in $E[N^2]$ are the multiples of T, all of which have order dividing N. Hence $[N]S = O$, which shows that $[k]T = O$. The point T has exact order N, so $N \mid k$. But k is the order of an element of $\mathbb{F}_q^*/(\mathbb{F}_q^*)^N$, so $k \mid N$. This proves that $k = N$, and hence $\tau(T, T)$ is an element of exact order N in $\mathbb{F}_q^*/(\mathbb{F}_q^*)^N$. It follows immediately that $\tau(T, T)^{(q-1)/N}$ is a primitive N^{th} root of unity.

It also proves nondegeneracy on the right. To see this, let $Q \in E(\mathbb{F}_q)[N]$ satisfy $\tau(P, Q) = 1$ for all $P \in E(\mathbb{F}_q)$. Then $Q = [n]T$ for some integer n, so taking $P = T$ yields

$$1 = \tau(P, Q) = \tau(T, [n]T) = \tau(T, T)^n.$$

We know that $\tau(T, T)$ has exact order N, so $N \mid n$, which shows that $Q = [n]T = O$. □

Remark 9.5. In the situation of (XI.9.4), the natural map

$$E(\mathbb{F}_q)[N] \to E(\mathbb{F}_q)/NE(\mathbb{F}_q)$$

is an isomorphism of cyclic groups of order N, so we can use the Tate–Lichtenbaum pairing to define a nondegenerate symmetric bilinear pairing

$$E(\mathbb{F}_q)[N] \times E(\mathbb{F}_q)[N] \longrightarrow \boldsymbol{\mu}_N, \qquad (P,Q) \longmapsto \tau(P,Q)^{(q-1)/N}.$$

This is the pairing that is typically used for cryptographic applications.

Remark 9.6. Suppose that we randomly choose an elliptic curve $E/\mathbb{F}_q$, compute the order of the group $E(\mathbb{F}_q)$ (e.g., using (XI §3)), and find that there is a large prime N dividing $\#E(\mathbb{F}_q)$. This almost puts us into the situation to apply (XI.9.4), but we may need to extend the field $\mathbb{F}_q$ in order to ensure that it contains $\boldsymbol{\mu}_N$. Let $d \ge 1$ be the embedding degree (XI §6) of N in $\mathbb{F}_q$, i.e., d is the smallest integer such that $\boldsymbol{\mu}_N \subset \mathbb{F}_{q^d}^*$, or equivalently, such that $q^d \equiv 1 \pmod N$. How large should we expect d to be?

If we write

$$\#E(\mathbb{F}_q) = q + 1 - a$$

as usual, then the assumption that $\#E(\mathbb{F}_q)$ is divisible by N implies that

$$q + 1 - a = \#E(\mathbb{F}_q) \equiv 0 \pmod N.$$

Similarly, since we have chosen d to satisfy $\boldsymbol{\mu}_N \subset \mathbb{F}_{q^d}^*$, we have

$$q^d - 1 = \#\mathbb{F}_{q^d}^* \equiv 0 \pmod N.$$

Hence

$$(a-1)^d \equiv 1 \pmod N.$$

We know from (V.1.1) that $|a| \le 2\sqrt{q}$, but within this allowed range, the value of a is more-or-less randomly distributed; see (C.21.4) for a precise statement. The expected order of a randomly chosen element of a randomly chosen cyclic group is a constant multiple of the order of the group (see Exercise 11.17), so if we choose $E/\mathbb{F}_q$ randomly, the embedding degree d is almost certain to be too large for practical applications. (See [13] for a more detailed analysis.)

Remark 9.7. *Supersingular Elliptic Curves*. Let $E/\mathbb{F}_p$ be supersingular elliptic curve with $p \ge 5$ prime, and suppose that $E(\mathbb{F}_p)$ contains a point of prime order N. We have seen (XI.6.4) that the embedding degree of N in $\mathbb{F}_p$ is 2, i.e., $\boldsymbol{\mu}_N \subset \mathbb{F}_{p^2}$, but unfortunately condition (ii) of (XI.9.4) is never true in this situation; see Exercise 11.18. In general, we know from (V.3.1) that every supersingular elliptic curve is isomorphic to a curve defined over $\mathbb{F}_{p^2}$, and supersingular curves always have small embedding degree, so it may be possible to apply (XI.9.4) to a supersingular curve defined over $\mathbb{F}_{p^2}$. As an alternative, one can use a distortion map as in (XI §7); see [51, §24.2.1b] or [116, §5.9.2].

Remark 9.8. *Pairing-Friendly Elliptic Curves*. In general we would like to construct an elliptic curve $E/\mathbb{F}_q$ such that $E(\mathbb{F}_q)$ contains a point of large prime order N and such that the embedding degree d of N in $\mathbb{F}_q$ is not too large. These are called *pairing-friendly elliptic curves*. The exact constraints on the parameters q, N, and d depend on the desired security level, but in any case it is important to balance the difficulty of solving the ECDLP in a subgroup of $E(\mathbb{F}_q)$ of order N against the difficulty of solving the DLP in $\mathbb{F}_{q^d}^*$. For the former, only exponential-time algorithms are known, while there are subexponential algorithms for the latter. See the discussion in (XI.4.1). Further, for computational efficiency we should choose q to be as small as possible.

For example, current algorithms to solve the ECDLP when $N \approx 2^{160}$ take about the same amount of time as current algorithms to solve the DLP when $q^d \approx 2^{1024}$. So for this security level, the embedding degree should be $d \approx 6.4 \log q / \log N$. Typically $\sqrt{q} < N < q$, so d should be roughly between 6 and 12.

Atkin and Morain [10, 183] devised a method using the theory of complex multiplication to find elliptic curves with points of large order and small embedding degree. Their idea is to fix positive integers D and d and to search for integers a, N, and p satisfying the following four conditions:

(1) N and p are primes.

(2) $N \mid p + 1 - a$.

(3) $N \mid p^d - 1$.

(4) The equation $Dy^2 = 4p - a^2$ has a solution $y \in \mathbb{Z}$.

If we can find values for a, N, and p satisfying (1)–(4) and if the class number of the quadratic field $\mathbb{Q}(\sqrt{-D}\,)$ is not too large, say less than 10^5, then Atkin and Morain's CM method yields an elliptic curve $E/\mathbb{F}_p$ with $N \mid E(\mathbb{F}_p)$ and $\boldsymbol{\mu}_N \subset \mathbb{F}_{p^d}$. For a description of the CM method, see for example [22, Chapter VIII] or [51, §18.1], and for various algorithms that have been devised to find (a, N, p) satisfying (1)–(4), see for example [29, 69, 87, 181].

Exercises

11.1. Use the double-and-add algorithm (XI.1.1) to compute $[n]P$ in $E(\mathbb{F}_p)$ for each of the following curves and points.

(a) $E : Y^2 = X^3 + 143X + 367$, $p = 613$, $P = (195, 9)$, $n = 23$.

(b) $E : Y^2 = X^3 + 1541X + 1335$, $p = 3221$, $P = (2898, 439)$, $n = 3211$.

11.2. Let n be a positive integer.

(a) Prove that n has a unique ternary expansion

$$n = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \epsilon_3 \cdot 2^3 + \cdots + \epsilon_t \cdot 2^t, \qquad \epsilon_0, \ldots, \epsilon_t \in \{-1, 0, 1\},$$

with the property that no two consecutive ϵ_i are nonzero. Such an expansion is called *nonadjacent form* (NAF). (*Hint*. For existence, start with the binary expansion of n and replace consecutive nonzero terms $2^i + 2^{i+1} + \cdots + 2^{i+j-1} + 0 \cdot 2^{i+j}$ with $-2^i + 2^{i+j}$.)

(b) If we assume that the expansion in (a) has $\epsilon_t \neq 0$, prove that $t \le \log_2(2n)$.

(c) Prove that most positive integers have a ternary expansion as in (a) with approximately one-third of the ϵ_i being nonzero.

(d) Convert your proof in (a) into an algorithm and find a ternary expansion for each of the following numbers. Compare the number of nonzero terms in the ternary expansion with the number of nonzero terms in the binary expansion.
(i) 349. (ii) 9337. (iii) 38728. (iv) 8379483273489.

11.3. Let τ represent a quantity satisfying $\tau^2 + \tau + 2 = 0$.

(a) Prove that every positive integer n can be written in the form

$$n = \epsilon_0 + \epsilon_1\tau + \epsilon_2\tau^2 + \cdots + \epsilon_t\tau^t, \qquad \epsilon_0, \ldots, \epsilon_t \in \{-1, 0, 1\},$$

with $t \le 2\lceil \log_2 n \rceil + 1$ and at most one-third of the ϵ_i nonzero. (*Hint*. Repeatedly write integers as $2a + b$ and replace the 2 with $-\tau - \tau^2$.)

(b) More generally, prove that (a) is true for any $n \in \mathbb{Z}[\tau]$, where the upper bound for t is approximately $\log_2 \mathrm{N}_{\mathbb{Z}[\tau]/\mathbb{Z}}(n)$.

(c) Let $E/\mathbb{F}_2$ be the curve in (XI.1.5), let $\tau(x, y) = (x^2, y^2)$ be the Frobenius map, let $P \in E(\mathbb{F}_{2^r})$, and let n be a positive integer. Prove that there is an element

$$\nu = \epsilon_0 + \epsilon_1\tau + \epsilon_2\tau^2 + \cdots + \epsilon_t\tau^t, \qquad \epsilon_0, \ldots, \epsilon_t \in \{-1, 0, 1\},$$

with $[\nu]P = [n]P$ and such that t is (approximately) bounded above by $\log_2 n$. (*Hint*. "Divide" n by $\tau^r - 1$ in $\mathbb{Z}[\tau]$ to find a remainder ν whose norm is approximately bounded by 2^r. Then use (b) and the fact that $\tau^r(P) = P$.)

(d) Devise an algorithm implementing your results in this exercise, and use your algorithm to compute a τ-adic expansion for each of the following values of n.
(i) $n = 931$ (ii) $n = 32755$ (iii) $n = 82793729188$

11.4. The double-and-add algorithm described in (XI.1.1) reads the bits of n from right to left, where we view n as a binary number such as 1001101. Prove that the following left-to-right version also computes nP.

(1) Write the binary expansion of n as $\displaystyle\sum_{i=0}^{t} \epsilon_i \cdot 2^i$.
(2) Set $Q = O$.
(3) Loop $i = t, t-1, \ldots, 2, 1, 0$.
(4) Set $Q = [2]Q$.
(5) If $\epsilon_i = 1$, set $Q = Q + P$.
(6) End Loop
(7) Return Q.

11.5. Let S be a set containing n elements.

(a) Suppose that we select an element of S at random, note its value, return the element to the set, and repeat the process m times. Find a formula for the probability that some element has been selected at least twice. (If this happens, we say that there has been a collision.)

(b) If $n = 365$ and $m = 50$, what is the probability of a collision? (This is the probability that among 50 people in a room, at least two have the same birthday. The surprising answer is the origin of the name "birthday paradox.") How many people are required to have a 50% chance that two share a common birthday?

(c) Suppose that n is large. Give a good approximation for the probability of a collision if $m = c\sqrt{n}$, where c is a small constant, say $1 \le c \le 10$. What value of c gives a 50% chance of a collision? What value of c gives a $1 - 10^{-6}$ probability of a collision?

11.6. Pollard's algorithm (XI.2.1) says that if N has a prime factor p with $p - 1 = \prod q_j^{e_j}$, then it suffices to take $L = \max e_j q_j$ in order to (probably) factor N. Show that it is enough that L satisfy $L \ge \sum_{t \ge 1} \lfloor L/q^t \rfloor$. Give a similar statement for the elliptic curve factorization algorithm (XI.2.3).

11.7. Implement Lenstra's elliptic curve factorization algorithm (XI.2.3) and use it to factor N using the given elliptic curve E and point P.

(a) $N = 589$, $E : Y^2 = X^3 + 4X + 9$, $P = (2, 5)$.

(b) $N = 26167$, $E : Y^2 = X^3 + 4X + 128$, $P = (2, 12)$.

(c) $N = 1386493$, $E : Y^2 = X^3 + 3X - 3$, $P = (1, 1)$.

(d) $N = 28102844557$, $E : Y^2 = X^3 + 18X - 453$, $P = (7, 4)$.

11.8. We noted (XI.4.3.5) that it suffices for Bob to send Alice the x-coordinate of his point $P \in E(\mathbb{F}_q)$, together with one extra bit that specifies which of the two possible y-coordinates to use. However, this means that Alice needs to compute a square root in $\mathbb{F}_q$.

(a) Suppose that $q \equiv 3 \pmod 4$, and let $a \in \mathbb{F}_q$ be an element that is a square. Prove that $b = a^{(q+1)/4}$ is a square root of a.

(b) Suppose that q is prime and satisfies $q \equiv 5 \pmod 8$. Let $a \in \mathbb{F}_q$ be an element that is a square, and let

$$b = \begin{cases} a^{(q+3)/8} & \text{if } a^{(q-1)/4} = 1, \\ 2a(4a)^{(q-5)/8} & \text{if } a^{(q-1)/4} = -1. \end{cases}$$

Prove that $b^2 = a$.

11.9. Let G be a group, and suppose that you know an algorithm that takes $T(n)$ steps to solve any discrete logarithm problem $h = g^m$ in G if the element g has order n.

(a) Let $g \in G$ have order n and suppose that n factors as $n = n_1 n_2 \cdots n_t$ with $\gcd(n_i, n_j) = 1$ for all $i \ne j$. Find an algorithm that solves $h = g^m$ in (approximately) $\sum T(n_i)$ steps. (*Hint.* Solve $(g^{n/n_i})^m = h^{n/n_i}$ for each i and combine the solutions using the Chinese remainder theorem.)

(b) Let $g \in G$ have order n with $n = \ell^k$ a prime power. Find an algorithm that solves $h = g^m$ in (approximately) $kT(\ell)$ steps.

11.10. This exercise describes the Menezes–Vanstone variant of the ElGamal cryptosystem.

1. Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, and a point $P \in E(\mathbb{F}_q)$.
2. Alice selects a secret integer a and computes the point $A = [a]P \in E(\mathbb{F}_q)$.
3. Alice publishes the point A. This is her *public key*. The secret multiplier a is her *private key*.
4. Bob chooses a *plaintext* $(m_1, m_2) \in \mathbb{F}_q^2$ and a random integer k. He computes the two points $B_1 = [k]P$ and $B_2 = [k]A$.
5. Bob writes B_2 as $(x, y) \in E(\mathbb{F}_q)$, sets $c_1 = xm_1$ and $c_2 = ym_2$, and sends Alice the ciphertext (B_1, c_1, c_2).

(a) Explain how Alice can use the ciphertext (B_1, c_1, c_2) and her secret multiplier a to recover Bob's plaintext (m_1, m_2).

(b) What is the message expansion (XI.4.5.3) of MV-ElGamal?
(c) Explain how Eve can break MV-ElGamal if she can solve the Diffie–Hellman problem (XI.4.3.4).

11.11. This exercise describes the Elliptic Curve Integrated Encryption Scheme (ECIES). It combines the discrete logarithm problem with several other cryptographic constructions, including a hash function that we denote by $\mathcal{H}$, a message authentication code that we denote by $\mathcal{M}$, and a private key cryptosystem that we denote by $\mathcal{P}$.[10]

1. Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E/\mathbb{F}_q$, and a point $P \in E(\mathbb{F}_q)$.
2. Alice selects a secret integer a and computes the point $A = [a]P \in E(\mathbb{F}_q)$.
3. Alice publishes the point A. This is her *public key*. The multiplier a is her *private key*.
4. Bob chooses a *plaintext* m and a random number k.
5. Bob computes $[k]A$ and uses the hash function to compute $\mathcal{H}\big(x\big([k]A\big)\big)$. He breaks this value into two pieces, say b_1 and b_2, which he uses as keys.
6. Bob uses the private key cryptosystem and the MAC to compute the two values
$$c = \mathcal{P}(b_1; m) \qquad \text{and} \qquad d = \mathcal{M}(b_2; c).$$
7. Bob computes $B = [k]P$.
8. Bob sends Alice the triple (B, c, d).

(a) Explain how Alice can recover the value of $x\big([k]A\big)$ that Bob used in step (5). This allows Alice to use the hash function to compute b_1 and b_2.
(b) Explain how Alice can then recover the message m.
(c) Explain how Alice can check the validity of the ciphertext c by recomputing $\mathcal{M}(b_2; c)$ and verifying that it agrees with the value of d sent by Bob.
(d) Explain why it is difficult for Eve to find a triple (B, c, d) that Alice accepts as valid unless she knows the plaintext m that corresponds to c via Alice's decryption process.

11.12. In the description of Pollard's ρ method in (XI §5), we gave an algorithm for computing the coefficients α_i and β_i in the expression $R_i = [\alpha_i]P + [\beta_i]Q$. Give a similar algorithm, in the form of two tables, for the coefficients γ_i and δ_i of the point
$$S_i = R_{2i} = [\gamma_i]P + [\delta_i]Q.$$
In other words, give the values of γ_{i+1} and δ_{i+1} in terms of γ_i and δ_i depending on whether the values of x_{S_i} and $x_{f(S_i)}$ are in A, B, or C.

11.13. Let $E/\mathbb{F}_p$ be an elliptic curve defined over a field of prime order. As described in (XI.5.4), one way to define a mixing function $f : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$ for use in Pollard's ρ algorithm (XI.5.3) is to write $E(\mathbb{F}_p)$ as a disjoint union of three sets A, B, C. For example, we might take
$$\begin{aligned} A &= \big\{P \in E(\mathbb{F}_p) : 0 \le x(P) < \tfrac{1}{3}p\big\},\\ B &= \big\{P \in E(\mathbb{F}_p) : \tfrac{1}{3}p \le x(P) < \tfrac{2}{3}p\big\},\\ C &= \big\{P \in E(\mathbb{F}_p) : \tfrac{2}{3}p \le x(P) < p\big\}. \end{aligned}$$

[10] Informally, a hash function is an easy-to-compute, hard-to-invert function; a message authentication code (MAC) is a hash function that requires a secret key; and a private key cryptosystem is a one-to-one function that is easy to compute in both directions if one knows the secret key, but hard to compute otherwise. For precise definitions and examples, see [169].

Using this choice of A, B, and C, write a computer program implementing Pollard's ρ algorithm and use it to solve the following discrete logarithm problems, i.e., find a value of m satisfying $Q = [m]P$.

(a) $p = 541$, $E : y^2 = x^3 + 442x + 211$, $P = (238, 345)$, $Q = (180, 148)$.

(b) $p = 7919$, $E : y^2 = x^3 + 1356x + 1654$, $P = (6007, 296)$, $Q = (2821, 6396)$.

(c) $p = 104729$, $E : y^2 = x^3 + 25780x + 74070$, $P = (6588, 76182)$, $Q = (14624, 59879)$.

11.14. Let G be an abelian group whose order is bounded by a known quantity, say $\#G \le n$, and let $x \in G$.

(a) Adapt Shanks's babystep–giantstep algorithm (XI.5.2) to find the order of x in time $O(\sqrt{n}\,)$ and space $O(\sqrt{n}\,)$.

(b) Adapt Pollard's ρ algorithm (XI.5.4) to find the order of x in time $O(\sqrt{n}\,)$ while using only space $O(1)$. (*Hint.* A direct adaptation of (XI.5.4) does not work, since the exponents $\alpha_i, \ldots, \delta_i$ cannot be reduced by the unknown order of the group. Instead write G as a disjoint union $G = A_1 \cup \cdots \cup A_t$, choose several random exponents $e_1, \ldots, e_t$ between 2 and n, and define $f : G \to G$ by $f(z) = g^{e_j} z$ if $z \in A_j$. Show that a match $z_{2i} = z_i$ is likely to be found with exponents $\alpha_i, \cdots, \delta_i$ that are $O(n\sqrt{n}\,)$.)

(c) Explain how to use an algorithm that finds the order of elements in G to determine the order of the group G.

11.15. Working over the field $\mathbb{F}_{137}$, consider the curve and points

$$E : y^2 = x^3 + 86x + 98, \quad P = (56, 85) \in E(\mathbb{F}_{137}), \quad Q = (54, 86) \in E(\mathbb{F}_{137}).$$

(a) Verify that E is anomalous, i.e., $\#E(\mathbb{F}_{137}) = 137$.

(b) Lift P and Q to points $P' = (56, —)$ and $Q' = (54, —)$ in $E(\mathbb{Z}/137^2\mathbb{Z})$.

(c) Compute the elliptic logarithms of $[137]P'$ and $[137]Q'$ modulo 137^2.

(d) As in (XI.6.5) and (XI.6.7), use the results from (c) to solve the discrete logarithm problem, i.e., find an integer m such that $Q = [m]P$ in $E(\mathbb{F}_{137})$.

11.16. Let $E/\mathbb{F}_{631}$ be the elliptic curve $y^2 = x^3 + 30x + 34$ from (XI.8.3).

(a) The points $P' = (617, 5)$ and $Q' = (121, 244)$ are in $E(\mathbb{F}_{631})[5]$. Use Miller's algorithm to compute $e_5(P', Q')$.

(b) Let $P = (36, 60)$ and $Q = (121, 387)$ be the points from (XI.8.3). Express P' and Q' as linear combinations of P and Q, and use linearity of e_N to express $e_5(P', Q')$ as a power of $e_5(P, Q)$.

(c) Verify that the value of $e_5(P', Q')$ from (a) and the value of $e_5(P, Q)$ from (XI.8.3) are consistent with the relation that you found in (b).

11.17. (a) Let C_m be a cyclic group of order m. Prove that the average order of an element of C_m is

$$A(C_m) = \frac{1}{m} \sum_{d \mid m} d\phi(d).$$

(b) Prove that

$$\frac{1}{X}\sum_{m\le X} A(C_m) = \frac{\zeta(3)}{2\zeta(2)}X + O(\log X),$$

where $\zeta(s)$ is the Riemann zeta function.

(c) Deduce that the expected order of a randomly chosen element in a randomly chosen cyclic group is proportional to the order of the group.

11.18. Let $p \ge 5$, let $E/\mathbb{F}_p$ be a supersingular elliptic curve, and let N be a prime such that $E(\mathbb{F}_p)$ contains a point of order N.

(a) Prove that $N^2 \mid \#E(\mathbb{F}_{p^2})$. (*Hint*. Use Exercise 5.15.)

(b) Deduce that one of the following statements is true:

(i) $E[N] \subset E(\mathbb{F}_{p^2})$.

(ii) $E(\mathbb{F}_{p^2})$ contains a point of order N^2.