Chapter 15

Elliptic curve cryptography and pairings

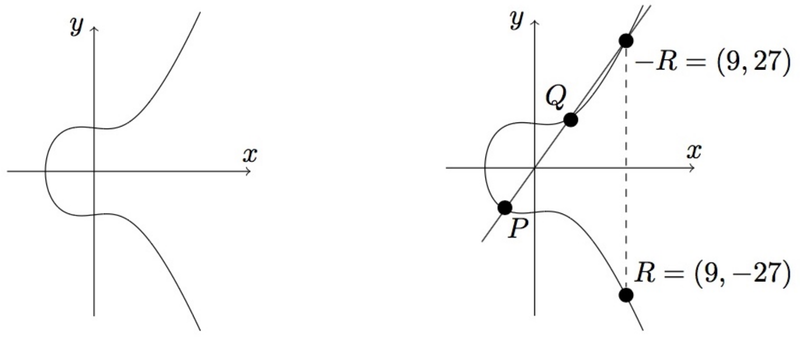
In previous chapters we saw many applications of the discrete log, CDH, and DDH assumptions in a finite cyclic group $\mathbb{G}$. Our primary example for the group $\mathbb{G}$ was the multiplicative group (or subgroup) of integers modulo a sufficiently large prime p. This group is problematic for a number of reasons, most notably because the discrete log problem in this group is not sufficiently difficult. The best known algorithm, called **the general number field sieve** (GNFS), discussed in Chapter 16, runs in time $\exp(\tilde{O}((\log p)^{1/3}))$. It was used in 2019 to solve a discrete log problem modulo a general 795-bit prime. This algorithm is the reason why, in practice, we must use a prime p whose size is at least 2048 bits. High security applications must use even larger primes. Arithmetic modulo such large primes is slow and greatly increases the cost of deploying cryptosystems that use this group.

Several other families of finite cyclic groups with an apparent hard discrete log have been proposed. Of all these proposals, the group of points of an elliptic curve over a prime finite field is the most suitable for practice, and is widely used on the Internet today. The best known discrete log algorithm in an elliptic curve group of size q runs in time $O(\sqrt{q})$. This means that to provide security comparable to AES-128, it suffices to use a group of size $q \approx 2^{256}$ so that the time to compute discrete log is $\sqrt{q} \approx 2^{128}$. The group operation uses a small number of arithmetic operations modulo a 256-bit prime, which is considerably faster than arithmetic modulo a 2048-bit prime.

Additional structure. As we will see, certain elliptic curve groups have an additional structure, called a **pairing**, that is enormously useful in cryptography. We will see many examples of encryption and signature schemes built using pairings. These systems exhibit powerful properties that are beyond what can be built using the multiplicative group of the integers modulo a prime. Some examples include aggregate signatures, broadcast encryption, functional encryption, and many others. The bulk of the chapter is devoted to exploring the world of pairing-based cryptography.

15.1 The group of points of an elliptic curve

Elliptic curves come up naturally in several branches of mathematics. Here we will follow their development as a branch of arithmetic (the study of rational numbers). Our story begins with Diophantus, a greek mathematician who lived in Alexandria in the third century AD. Diophantus



(a) The curve (b) Adding $P = (-1, -3)$ and $Q = (1, 3)$

Figure 15.1: The curve $y^2 = x^3 - x + 9$ over the reals (not drawn to scale)

was interested in the following problem: given a bivariate polynomial equation, $f(x, y) = 0$, find rational points satisfying the equation. A rational point is one where both coordinates are rational, such as $(1/2,\ 1/3)$, but not $(1, \sqrt{2}\,)$. Diophantus wrote a series of influential books on this subject, called the *Arithmetica*, of which six survived. Fourteen centuries later, Fermat scribbled his famous conjectures in the margins of a latin translation of the Arithmetica. An insightful short book by Bashmakova [11] describes Diophantus' ideas in a modern mathematical language.

Much of the Arithmetica studies integer and rational solutions of quadratic equations. However, in a few places Diophantus considers problems of higher degree. Problem 24 of book 4, which is the first occurrence of elliptic curves in mathematics, looks at a cubic equation. The problem is equivalent to the following question: find rational points $(x, y) \in \mathbb{Q}^2$ satisfying the equation

$$y^2 = x^3 - x + 9. \tag{15.1}$$

Fig. 15.1 shows a plot of this curve over the real numbers. We do not know what compelled Diophantus to ask this question, but it is a good guess that he would be shocked to learn that the method he invented to answer it now secures Internet traffic for billions of people worldwide.

One can easily verify that the six integer points $(0, \pm 3)$, $(1, \pm 3)$, $(-1, \pm 3)$ are on the curve (15.1). Diophantus wanted to find more rational points on this curve.

He proceeded to derive new rational points from the six he already had. Here is one way to do it, which is slightly different from what Diophantus did. Let $P := (-1, -3)$ and $Q := (1, 3)$, both satisfying (15.1). Let's look at the line passing through P and Q, as shown in Fig. 15.1b. One can easily verify that this line is simply $y = 3x$. It must intersect the curve $y^2 = x^3 - x + 9$ at exactly three points. To see why, observe that if we substitute $3x$ for y in (15.1) we obtain the univariate cubic equation $(3x)^2 = x^3 - x + 9$. We already know two rational roots of this cubic equation: $x_1 = -1$ from the point P, and $x_2 = 1$ from the point Q. It is not difficult to show that a cubic with rational coefficients that has two rational roots, must also have a third rational root x_3. In our case, this third rational root happens to be $x_3 = 9$. Setting $y_3 = 3x_3$ we obtain a new point on the curve (15.1), namely $(9, 27)$. For reasons that will become clear in a minute, we denote this point by $-R$. We get another point for free, $(9, -27)$, which we call R. More generally, for a point $T = (x, y)$ on the curve, we let $-T$ be the point $-T := (x, -y)$.

This technique for building rational points is called the **chord method**. It is quite general: given two distinct rational points U and V on the curve, where $U \neq -V$, we can pass a line through them, and this line must intersect the curve at a third rational point W. For example, applying this to the points P and R gives two new points $(-\frac{56}{25}, \frac{3}{125})$ and $(-\frac{56}{25}, -\frac{3}{125})$.

The chord method was re-discovered several times over the centuries, but it finally stuck with the work of Poincaré on algebraic curves [130]. Poincaré likened the process of constructing a new rational point from two known rational points to an addition operation in a group. Specifically, for distinct points U and V on the curve, with $U \neq -V$, let W be the point on the curve obtained by passing a line through U and V and finding its third point of intersection with the curve. Then Poincaré defines the sum of U and V, denoted $U \boxplus V$, as

$$U \boxplus V := -W. \tag{15.2}$$

Fig. 15.1b shows this addition rule applied to the points P and Q. Their sum $P \boxplus Q$ is the point $R = (9, -27)$. Defining addition as in (15.2) makes this operation associative, when it is well defined. Recall that associativity means that $(U \boxplus V) \boxplus W = U \boxplus (V \boxplus W)$.

We will show in the next section how to enhance this addition rule so that the set of points on the curve becomes a group. Some of the most beautiful results in number theory, and some of the deepest open problems, come from trying to understand the properties of the group of rational points on elliptic curves [8].

Going back to Diophantus, his approach for finding rational points on (15.1) is a variation of the method we just saw. Instead of passing a line through two distinct points, Diophantus chose to pass a tangent to the curve at one of the known points. Say we pass a tangent at the point $P = (-1, -3)$. As before, it is not difficult to show that on a cubic curve with rational coefficients, if (x_1, y_1) is a rational point with $y_1 \neq 0$, then the tangent at (x_1, y_1) must intersect the curve at exactly one more point T, and this point must also be rational. In our case, the tangent at $P = (-1, -3)$ is the line $y = -\frac{1}{3}x - \frac{10}{3}$. It intersects the curve at the point P and at the point $(\frac{19}{9}, -\frac{109}{27})$ which is indeed rational. This method, called the **tangent method**, is another way to build a new rational point from a given rational point (x_1, y_1), when $y_1 \neq 0$. As we will see, it corresponds to adding the point P to itself, namely computing $P \boxplus P$.

15.2 Elliptic curves over finite fields

The curve (15.1) is an example of an elliptic curve defined over the rationals. For cryptographic applications we are mostly interested in elliptic curves over finite fields. For simplicity, we only consider elliptic curves defined over a finite field $\mathbb{F}_p$ where $p > 3$ is a prime.

Definition 15.1. *Let $p > 3$ be a prime. An **elliptic curve** E defined over $\mathbb{F}_p$ is an equation*

$$y^2 = x^3 + ax + b, \tag{15.3}$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$. We write $E/\mathbb{F}_p$ to denote the fact that E is defined over $\mathbb{F}_p$.

The condition $4a^3 + 27b^2 \neq 0$ ensures that the equation $x^3 + ax + b = 0$ does not have a double root. This is needed to avoid certain degeneracies.

The set of points on the curve. Let $E/\mathbb{F}_p$ be an elliptic curve, and let $e \geq 1$. We say that a point (x_1, y_1), where $x_1, y_1 \in \mathbb{F}_{p^e}$, is a point **on the curve** E if (x_1, y_1) satisfies the curve equation (15.3). When $e = 1$ the point (x_1, y_1) is defined over the base field $\mathbb{F}_p$. When $e > 1$ the point is defined over an extension of $\mathbb{F}_p$.

The curve includes an additional "special" point $\mathcal{O}$ called **the point at infinity**. Its purpose will become clear in a minute. We write $E(\mathbb{F}_{p^e})$ to denote the set of all points on the curve E that are defined over $\mathbb{F}_{p^e}$, including the point $\mathcal{O}$.

For example, consider the curve $E:\ y^2 = x^3 + 1$ defined over $\mathbb{F}_{11}$. Then

$$E(\mathbb{F}_{11}) = \Big\{\mathcal{O},\ \ (-1,0),\ \ (0,\pm1),\ \ (2,\pm3),\ \ (5,\pm4),\ \ (7,\pm5),\ \ (9,\pm2)\Big\} \tag{15.4}$$

This curve has 12 points in $\mathbb{F}_{11}$ and we write $|E(\mathbb{F}_{11})| = 12$.

A classic result of Hasse shows that $|E(\mathbb{F}_{p^e})| = p^e + 1 - t$ for some integer t in the interval $|t| \le 2\sqrt{p^e}$. This shows that the number of points on $E(\mathbb{F}_{p^e})$ is close to $p^e + 1$. The set $E(\mathbb{F}_p)$ in example (15.4) has exactly $p + 1$ points (so that $t = 0$).

Remark 15.1 (Point counting). A beautiful algorithm due to Schoof [141] can be used to compute the number of points in $E(\mathbb{F}_{p^e})$ in time polynomial in $\log(p^e)$. Hence, $|E(\mathbb{F}_{p^e})|$ can be computed efficiently even for a large prime p. Elkies and Atkin showed how to reduce the running time, and the resulting point counting method is called the Schoof-Elkies-Atkin algorithm, or simply, the **SEA algorithm**. □

The addition law. As we discussed in the previous section, there is a natural group law defined on the points of an elliptic curve. The group operation is written additively using the symbol "$\boxplus$" to denote point addition. We define the point at infinity $\mathcal{O}$ to be the identity element: for all $P \in E(\mathbb{F}_{p^e})$ we define $P \boxplus \mathcal{O} = \mathcal{O} \boxplus P = P$.

Now, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points in $E(\mathbb{F}_{p^e})$. The sum $P \boxplus Q = (x_3, y_3)$ is defined using one of the following three rules:

- if $x_1 \ne x_2$ we use the chord method. Let $s_c := \frac{y_1 - y_2}{x_1 - x_2}$ be the slope of the chord through the points P and Q. Define
$$x_3 := s_c^2 - x_1 - x_2 \qquad \text{and} \qquad y_3 := s_c(x_1 - x_3) - y_1.$$
- if $x_1 = x_2$ and $y_1 = y_2$ (i.e., $P = Q$), but $y_1 \ne 0$, we use the tangent method. Let $s_t := \frac{3x_1^2 + a}{2y_1}$ be the slope of the tangent at P. Define
$$x_3 := s_t^2 - 2x_1 \qquad \text{and} \qquad y_3 := s_t(x_1 - x_3) - y_1.$$
- if $x_1 = x_2$ and $y_1 = -y_2$ then define $P \boxplus Q := \mathcal{O}$.

This addition law makes the set $E(\mathbb{F}_{p^e})$ into a group. The identity element is the point at infinity. Every point $\mathcal{O} \ne P = (x_1, y_1) \in E(\mathbb{F}_{p^e})$ has an additive inverse, namely $-P = (x_1, -y_1)$. Finally, it can be shown that this addition law is associative. The group law is clearly commutative, $P \boxplus Q = Q \boxplus P$ for all $P, Q \in E(\mathbb{F}_{p^e})$, making this an abelian group.

As in any group, for a point $P \in E(\mathbb{F}_{p^e})$ we write $2P := P \boxplus P$, $\ 3P := P \boxplus P \boxplus P$, and more generally, $\alpha P := (\alpha - 1)P \boxplus P$ for any positive integer α. Note that αP can be computed using at most $2 \log_2 \alpha$ group operations using the repeated squaring algorithm (Appendix A).

15.2.1 Montgomery and Edwards curves

Equation (15.3) for an elliptic curve is called the **Weierstrass form** of the curve. There are many equivalent ways of describing an elliptic curve and some are better suited for computation than the Weierstrass form. We give two examples.

Montgomery curves. A **Montgomery curve** $E/\mathbb{F}_p$ in the variables u and v is written as

$$Bv^2 = u^3 + Au^2 + u$$

for some $A, B \in \mathbb{F}_p$ where $B(A^2-4) \neq 0$. This curve equation can be easily changed into Weierstrass form via the change of variables $u := Bx - A/3$ and $v := By$. The number of points on a Montgomery curve, $|E(\mathbb{F}_{p^e})|$, is always divisible by four. Exercise 15.4 explores the computational benefit of Montgomery curves. They will also come up in the next section.

Some Weierstrass curves defined over $\mathbb{F}_p$ cannot be put into Montgomery form over $\mathbb{F}_p$ by a change of variables. For example, if the Weierstrass curve has an odd number of points over $\mathbb{F}_p$, then it has no Montgomery form over $\mathbb{F}_p$. Such Weierstrass curves cannot benefit from the speedup associated with Montgomery curves. The curve P256, discussed in the next section, is an example of such a curve.

Edwards curves. Another way to describe an elliptic curve $E/\mathbb{F}_p$ is in Edwards form, which is

$$x^2 + y^2 = 1 + dx^2y^2$$

where $d \in \mathbb{F}_p$ satisfies $d \neq 0, 1$. Again, this curve can be put into Weierstrass form via a simple rational change of variable. The beauty of the Edwards form is that the chord and tangent addition law is extremely easy to describe. For points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_{p^e})$, we define

$$P \boxplus Q := \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2},\ \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

That's it. There is no need for three separate rules. The identity is the point $\mathcal{O} = (0, 1)$ and the inverse of a point (x_1, y_1) is $(-x_1, y_1)$. The points $(\pm 1, 0)$ have order four, which means that the number of points on an Edwards curve, $|E(\mathbb{F}_{p^e})|$, is always divisible by four.

The simplicity of the addition law on an Edwards curve makes it easier to resist timing attacks on the implementation. It also leads to a very fast implementation.

15.3 Elliptic curve cryptography

Let $E/\mathbb{F}_p$ be an elliptic curve and let $E(\mathbb{F}_p)$ be the group of points on this curve that are defined over $\mathbb{F}_p$. We know that $E(\mathbb{F}_p)$ is a finite abelian group. Suppose that it also happens to be cyclic, meaning that $E(\mathbb{F}_p)$ is generated by some point $P \in E(\mathbb{F}_p)$. We can now ask about the complexity of problems like discrete log, computational Diffie-Hellman (CDH), and decision Diffie-Hellman (DDH) in the group $E(\mathbb{F}_p)$. Once we establish that discrete log, CDH, and DDH are hard in this group, we can instantiate all the constructions we covered in the previous several chapters using the group $E(\mathbb{F}_p)$. The resulting systems are called **elliptic curve cryptosystems**.

Let us review the discrete log problem in $E(\mathbb{F}_p)$. Let P be a point in $E(\mathbb{F}_p)$ of order q, meaning that $qP = \mathcal{O}$. The discrete log problem in $E(\mathbb{F}_p)$ is as follows: We are given P and q, along with another point $Q := \alpha P$, for some $\alpha \in \mathbb{Z}_q$. Here αP is the point obtained by adding P to itself α times using the addition law. We want to compute α, the discrete log of Q base P in $E(\mathbb{F}_p)$.

In Chapter 16 we will show that there is a generic discrete log algorithm that computes discrete log in every cyclic group of order q using at most $O(\sqrt{q}\,)$ group operations. We want a curve $E/\mathbb{F}_p$, or a set of curves, where the best known discrete log algorithm in $E(\mathbb{F}_p)$ runs in about the same time as this generic algorithm. That is, there is no known algorithm that can compute discrete log in $E(\mathbb{F}_p)$ much faster than $\sqrt{q}$ time, where $q := \big|E(\mathbb{F}_p)\big|$.

Insecure curves. For many elliptic curves $E/\mathbb{F}_p$ over a prime field $\mathbb{F}_p$, the best known discrete log algorithm runs in time $O(\sqrt{q}\,)$, where $q := \big|E(\mathbb{F}_p)\big|$. However, there are several notable exceptions where discrete log is much easier. Three examples are:

- In Section 16.1.2.2 we will show that if $\big|E(\mathbb{F}_p)\big|$ is composite, and all its prime factors are less than some bound $q_{\max}$, then there is an algorithm to compute discrete log in $E(\mathbb{F}_p)$ in time $\tilde{O}(\sqrt{q_{\max}})$. In particular, if all the prime factors of $\big|E(\mathbb{F}_p)\big|$ are small, say less than 2^{80}, then the discrete log problem in $E(\mathbb{F}_p)$ is easy in practice. For this reason we will only use a curve $E/\mathbb{F}_p$ if $\big|E(\mathbb{F}_p)\big|$ is equal to either q, $4q$, or $8q$ for some prime number q. This will ensure that the algorithm from Section 16.1.2.2 runs in time $O(\sqrt{q}\,)$.
- When $\big|E(\mathbb{F}_p)\big| = p$, the discrete log problem in $E(\mathbb{F}_p)$ is solvable in polynomial time. These curves are called *anomalous curves* and should never be used in cryptographic applications.
- Suppose there is a small integer $\tau > 0$ such that $|E(\mathbb{F}_p)|$ divides $p^\tau - 1$. Then discrete log on $E(\mathbb{F}_p)$ reduces to discrete log in the finite field $\mathbb{F}_{p^\tau}$, as explained in Section 15.4. Discrete log in $\mathbb{F}_{p^\tau}$ can be solved using variants of the general number field sieve (GNFS) discrete log algorithm discussed in Section 16.2. For example, if τ is small, say $\tau = 2$, and p is a 256-bit prime, then discrete log in $E(\mathbb{F}_p)$ can be solved efficiently: first reduce a given discrete log problem to $\mathbb{F}_{p^2}$, and then apply GNFS in $\mathbb{F}_{p^2}$. Since $\mathbb{F}_{p^2}$ is a finite field of size about 2^{512}, the discrete log problem in $\mathbb{F}_{p^2}$ can be solved in a reasonable amount of time (several hours). To prevent this attack we need to ensure that p^τ is sufficiently large so that GNFS in $\mathbb{F}_{p^\tau}$ is infeasible.

To avoid these pitfalls, many implementations use a curve from a fixed collection of curves that have been vetted and shown to not be vulnerable to the attacks listed above. The three most widely used curves are called **secp256r1**, **secp256k1**, and **Curve25519**. We discuss these curves in the next two subsections.

Remark 15.2 (Discrete log in $E(\mathbb{F}_{p^e})$***).*** Let $E/\mathbb{F}_p$ be an elliptic curve defined over the prime finite field $\mathbb{F}_p$. Asymptotically, discrete log in the group $E(\mathbb{F}_{p^e})$, for $e \geq 3$, is not as hard as one would like. For $e \geq 2$ there is a discrete log algorithm in the group $E(\mathbb{F}_{p^e})$ that runs in conjectured time $\tilde{O}(p^{2-2/e})$ [70]. For $e = 3$ this evaluates to $\tilde{O}(p^{1.33})$, for $e = 4$ it evaluates to $\tilde{O}(p^{1.5})$, and so on. This is asymptotically faster than the generic discrete log algorithm that, for $e = 3$ and $e = 4$, runs in time $O(p^{1.5})$ and $O(p^2)$, respectively. For this reason, the group $E(\mathbb{F}_{p^e})$ can only be used if p is sufficiently large so as to make this algorithm impractical. Taking $p \geq 2^{256}$ is sufficient. □

15.3.1 The curves secp256r1 and secp256k1

Two widely used elliptic curves, called **secp256r1** and **secp256k1**, are specified in a standard called SEC2, where SEC is an acronym for "standards for efficient cryptography." Both curves are defined over a 256-bit prime field, hence the "256" in their names. The 'r' in secp256r1 signifies that the curve is a random curve, meaning that it was generated by a certain sampling procedure. The 'k' in secp256k1 signifies that the curve is a *Koblitz curve* as explained below. The curve secp256r1 is widely used in Internet protocols, while secp256k1 is widely used in blockchain systems.

The curve secp256r1. This curve was approved by the U.S. National Institute of Standards (NIST) for federal government use in a standard published in 1999. The NIST standard refers to this curve as **Curve P256**. All implementations of TLS 1.3 are required to support this curve for Diffie-Hellman key exchange. It is the only mandatory curve in the TLS 1.3 standard, as discussed in Section 21.10.

The curve secp256r1 is defined as follows:

- The curve is defined over the prime $p_{\mathrm{r}} := 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. The special structure of this prime is meant to improve the performance of arithmetic modulo p_{r}.
- The curve has the Weierstrass form $y^2 = x^3 - 3x + b$ where $b \in \mathbb{F}_{p_{\mathrm{r}}}$ written as a 255-bit number in hexadecimal is:

  $b :=$ `5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`.

- The number of points on this curve is a prime number. Recall that it must be close to p_{r}.
- The standard also specifies a point G_{r} that generates the entire group $E(\mathbb{F}_{p_{\mathrm{r}}})$, where E is the curve secp256r1.

How was the odd looking parameter b selected? The reality is that we do not really know. The standard lists an unexplained constant called a **seed** S. This seed was provided as input to a public deterministic algorithm, that generated the parameter b. This process was designed to select a random curve that resists the known discrete log attacks. The problem is that we do not know for sure how the seed S was selected. An organization that wants to use secp256r1 might worry that S was chosen adversarially so that discrete log on the resulting curve is easy. Currently we do not know how to select such a seed even if we wanted to, so this concern is just an intriguing speculation. As far as we can tell, secp256r1 is a fine curve to use. It is widely used in Internet protocols.

The curve secp256k1. The second curve in the SEC2 standard is called secp256k1. This curve was selected for the digital signature scheme in the Bitcoin blockchain. Subsequent blockchains inherited the same curve. One reason for this choice is the concern over the choice of seed in the curve secp256r1 (also known as Curve P256) discussed in the previous paragraph.

The curve secp256k1 is defined as follows:

- The curve is defined over the prime $p_{\mathrm{k}} := 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Again, the special structure of this prime is meant to improve the performance of arithmetic modulo p_{k}.
- The curve has the Weierstrass form $y^2 = x^3 + 7$. Note that there are no unexplained large constants in the Weierstrass form.
- The number of points on this curve is a prime number. Recall that it must be close to p_{k}.
- The standard also specifies a point G_{k} that generates the entire group $E(\mathbb{F}_{p_{\mathrm{k}}})$, where E is the curve $y^2 = x^3 + 7$.

Properties of secp256k1. The curve secp256k1 has two interesting properties. First, it is a Koblitz curve, which means that there is a useful map defined on it. To explain what that means, let E be the curve $y^2 = x^3 + 7$, and let $p := p_{\mathrm{k}}$ be the prime over which secp256k1 is defined. Let q be the size of $E(\mathbb{F}_p)$. It so happens that $p \equiv 1 \pmod 3$ and therefore there exists $1 \neq \omega \in \mathbb{F}_p$ such that $\omega^3 = 1$.

Now, consider the map

$$\phi : \mathbb{F}_p^2 \to \mathbb{F}_p^2 \quad \text{defined by} \quad \phi(x, y) := (\omega x, y). \tag{15.5}$$

It is easy to verify that if $(x, y) \in \mathbb{F}_p^2$ is a point in $E(\mathbb{F}_p)$ then $\phi(x, y) = (\omega x, y)$ is also in $E(\mathbb{F}_p)$. Indeed, if (x, y) satisfies $y^2 = x^3 + 7$ in $\mathbb{F}_p$ then so does $(\omega x, y)$ because $\omega^3 = 1$. Let us also define $\phi(\mathcal{O}) := \mathcal{O}$ so that ϕ is a map from $E(\mathbb{F}_p)$ to $E(\mathbb{F}_p)$.

The general theory of elliptic curves tells us that the map ϕ in (15.5) must be a group homomorphism. Therefore, since $E(\mathbb{F}_p)$ is a cyclic group, there must be a constant λ in $\mathbb{Z}_q$ such that for all P in $E(\mathbb{F}_p)$ we have

$$\phi(P) = \lambda \cdot P,$$

where $\lambda \cdot P$ is the point in $E(\mathbb{F}_p)$ obtained by adding P to itself λ times.

We can determine the constant λ as follows. We know that for all non-zero $P \in E(\mathbb{F}_p)$, the three points P, $\phi(P)$, and $\phi^2(P) := \phi(\phi(P))$ all have the same y-coordinate, and therefore they are colinear — they all lie on the horizontal line through P. By definition of the addition law, this means that their sum must be zero. In other words, for all $P \in E(\mathbb{F}_p)$ we have

$$\mathcal{O} = P + \phi(P) + \phi^2(P) = (1 + \lambda + \lambda^2) \cdot P.$$

We can therefore conclude that $1 + \lambda + \lambda^2 = 0$ in $\mathbb{Z}_q$. Hence, λ must be one of the two non-trivial cube roots of unity in $\mathbb{Z}_q$. Indeed, $q \equiv 1 \pmod 3$ so that a non-trivial cube root exists in $\mathbb{Z}_q$. Thus, we know the value of λ.

How is this map useful? The fact that the map ϕ is so easy to calculate can be used to speed up multiplication on the curve secp256k1. Let $\alpha \in \mathbb{Z}_q$ and suppose that we want to calculate $\alpha \cdot P$ for some $P \in E(\mathbb{F}_p)$. For most α in $\mathbb{Z}_q$ it is possible to find integers τ_0, τ_1, τ_2 such that

$$\alpha = \tau_0 + \tau_1 \lambda + \tau_2 \lambda^2 \quad \text{and } \left|\tau_i\right| \leq 2q^{1/3} \text{ for } i = 0, 1, 2.$$

Then to compute αP it suffices to compute

$$\alpha P = \tau_0 \cdot P + \tau_1 \cdot \phi(P) + \tau_2 \cdot \phi^2(P). \tag{15.6}$$

This equality converts a multiplication by α into three multiplications where the multipliers τ_0, τ_1, τ_2 are much smaller than α. The expression on the right hand side of (15.6) can computed about two to three times faster than computing αP directly. This is done using a fast multi-exponentiation algorithm discussed in Appendix A. The end result is that the map ϕ can be used to speed up multiplication on secp256k1.

Finally, we note that the curve secp256k1 has another useful property that is a bit magical. If E is the curve $y^2 = x^3 + 7$ and $p := p_{\mathrm{k}}$ is the prime over which secp256k1 is defined, then

$$\text{the group } E(\mathbb{F}_p) \text{ has size } q \qquad \text{and} \qquad \text{the group } E(\mathbb{F}_q) \text{ has size } p.$$

This property is useful in certain zero knowledge proof systems, but we will not make use of it here.

Security of discrete log on secp256r1 and secp256k1. Because the primes $p_{\rm r}$ and $p_{\rm k}$ are close to 2^{256}, the number of points on both curves is also close to 2^{256}. Therefore, computing discrete log on these curves using a generic discrete log algorithm takes approximately 2^{128} group operations. We assume that no algorithm can compute discrete log much faster than that. The intent is that discrete log on both curves (as well as CDH and DDH on both curves) should be at least as hard as breaking AES-128. Consequently, if one is aiming for the level of security provided by AES-128, then either curve can be used for Diffie-Hellman key exchange, public-key encryption, and digital signatures.

Curves with higher security. Some high-security applications use AES-256 to encrypt plaintext data. In these cases, one should use an elliptic curve with a higher security parameter. One option is another curve from SEC2 called **secp521r1**, whose size is approximately 2^{521}. It is defined over the Mersenne prime $p = 2^{521} - 1$. Discrete log on this curve is believed to require at least 2^{256} group operations. This curve is also approved by the U.S. national institute of standards (NIST) for federal government use.

15.3.2 A security twist

Every elliptic curve $E/\mathbb{F}_p$ has a related curve $\tilde{E}/\mathbb{F}_p$ called the **twist** of E. Let $c \in \mathbb{F}_p$ be some quadratic non-residue in $\mathbb{F}_p$. If E is the curve $y^2 = x^3 + ax + b$ then its twist $\tilde{E}$ is the curve $cy^2 = x^3 + ax + b$. It is a simple exercise to show that $\left|E(\mathbb{F}_p)\right| + \left|\tilde{E}(\mathbb{F}_p)\right| = 2p + 2$. Since the number of points on $E(\mathbb{F}_p)$ is $p+1-t$, it follows that the number of points on $\tilde{E}(\mathbb{F}_p)$ must be $\tilde{n} := p+1+t$.

We say that a curve $E/\mathbb{F}_p$ is **twist secure** if discrete log is intractable on both $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_p)$. For $E/\mathbb{F}_p$ to be twist secure we need, at the very least, that both $n = |E(\mathbb{F}_p)|$ and $\tilde{n} = |\tilde{E}(\mathbb{F}_p)|$ are prime numbers, or are small multiples of large primes.

Why do we need twist security? Consider a system where Bob has a secret key $\alpha \in \mathbb{Z}_q$. Under normal operation, anyone can send Bob a point $P \in E(\mathbb{F}_p)$ and Bob will respond with the point αP. One system that operates this way is the oblivious PRF in Section 11.6.2. Before responding, Bob had better check that the given point P is in $E(\mathbb{F}_p)$; otherwise, the response that Bob sends back could compromise his secret key α, as discussed in Exercise 15.1 (see also Remark 12.1 where a similar issue came up). Checking that a point $P = (x_1, y_1)$ satisfies the curve equation is quite simple and efficient. However some implementations use the optimizations outlined in Exercises 15.2 and 15.4, where Bob is only sent the x-coordinate of P. The y-coordinate is not needed and is never sent. In this case, checking that the given $x_1 \in \mathbb{F}_p$ is valid requires a full exponentiation to confirm that $x_1^3 + ax_1 + b$ is a quadratic residue in $\mathbb{F}_p$ (see Appendix A.2.3). Suppose Bob skips this expensive check. Then an attacker could send Bob an $x_1 \in \mathbb{F}_p$ that is the x-coordinate of a point $\tilde{P}$ on the twist $\tilde{E}(\mathbb{F}_p)$. Bob would then respond with the x-coordinate of $\alpha\tilde{P}$ in $\tilde{E}(\mathbb{F}_p)$. If discrete log in $\tilde{E}(\mathbb{F}_p)$ were easy, this response would expose Bob's secret key α. Hence, if Bob skips the group membership check, we must ensure, at the very least, that discrete log in $\tilde{E}(\mathbb{F}_p)$ is intractable so that $\alpha\tilde{P}$ does not expose α. Twist security is meant to ensure exactly that.

The curves secp256r1 and secp256k1 were not designed to be twist secure. The size of the twist of secp256r1 is divisible by $34905 = 3 \times 5 \times 13 \times 179$. Consequently, discrete log on the twist is $\sqrt{34905} \approx 187$ times easier than on secp256r1 (see Section 16.1.2.2). Similarly, for secp256k1 the size of the twist is divisible by $3^2 \times 13^2 \times 3319 \times 22639$, and consequently, discrete log on the twist is $\approx 2^{18}$ times easier than on `secp256k1`. These are important facts to remember, but not a significant enough concern to disqualify secp256r1 or secp256k1.

15.3.3 Curve25519

Curve25519 is designed to support an optimized group operation and to be twist secure. The curve is defined over the prime $p := 2^{255} - 19$, which is the reason for its name. This p is the largest prime less than 2^{255} and this enables fast arithmetic in $\mathbb{F}_p$.

It is easiest to describe Curve25519 as a Montgomery curve, namely a curve in the form $E : By^2 = x^3 + Ax^2 + x$ for some $A, B \in \mathbb{F}_p$ where $p > 3$. Exercise 15.4 shows that these curves support a fast multiplication algorithm to compute αP from P where $P \in E(\mathbb{F}_p)$ and $\alpha \in \mathbb{Z}$. We noted earlier that the number of points $|E(\mathbb{F}_p)|$ on a Montgomery curve is always a multiple of four.

Curve25519 presented as a Montgomery curve is simply

$$y^2 = x^3 + 486662 \cdot x^2 + x.$$

The number of points on this curve is eight times a prime. We say that the curve has **cofactor** eight. The curve is generated by a point $P = (x_1, y_1)$ where $x_1 = 9$. For completeness, we note that Curve25519 can also be presented as the Edwards curve $x^2 + y^2 = 1 + (121665/121666)x^2y^2$.

Why the constant 486662? When defining a Montgomery curve, the smaller A is, the faster the group operation becomes, as explained in Exercise 15.4. For the best performance we need $(A - 2)/4$ to be small [17]. Dan Bernstein, who designed this curve, chose the smallest possible A so that the curve is secure against the known discrete log attacks. He also made sure that the order of the curve and the order of its twist are either four times a prime or eight times a prime. Dan Bernstein writes [18]:

> The smallest positive choices for A are 358990, 464586, and 486662. I rejected $A = 358990$ because one of its primes is slightly smaller than 2^{252}, raising the question of how standards and implementations should handle the theoretical possibility of a user's secret key matching the prime; discussing this question is more difficult than switching to another A. I rejected 464586 for the same reason. So I ended up with $A = 486662$.

This explanation is a bit more satisfying than the unexplained constants in the random curve secp256r1, and the unexplained prime in secp256k1.

15.4 Pairing based cryptography

Up until now, we used elliptic curves as an efficient group where the discrete log problem and its variants, CDH and DDH, are believed to be difficult. This group makes it possible to instantiate efficiently many of the schemes described in earlier chapters. We now show that certain elliptic curves have an additional structure, called a **pairing**. The pairing enables a world of new schemes that could not be built from discrete log groups without this additional structure. The resulting schemes make up an important area called **pairing based cryptography**.

To present pairing based schemes we focus on the new capabilities enabled by a pairing and abstract away the details of the elliptic curve group. As in earlier chapters, we will write the group operation multiplicatively. This is a little different from the first part of this chapter where we used additive notation for the group operation to be consistent with traditional elliptic curve mathematical notation.