A Gentle Introduction to Elliptic Curve Cryptography

Tanner Prynn

May 5, 2015

Contents

1 Elliptic Curves

An **elliptic curve** is a set of points satisfying an equation of the form

$$y^2 = x^3 + ax + b$$

for coefficients a, b and variables x, y in some field F (of characteristic not 2 or 3). We place one additional restriction on an elliptic curve, which is that $4a^3 + 27b^2 \neq 0$. This condition ensures that the curve is *non-singular*, which allows us to find a tangent line to any point on the curve [2, §3.1].

Let's start with a few examples of curves, plotted over the real numbers. Figure 1 shows a simple elliptic curve. The three points where it crosses the x-axis correspond to the zeroes of the polynomial $x^3 - 3x + 1$. Figure 2 shows the elliptic curve $y^2 = x^3 - 2x + 2$, which only crosses the x-axis once in $\mathbb{R}$.

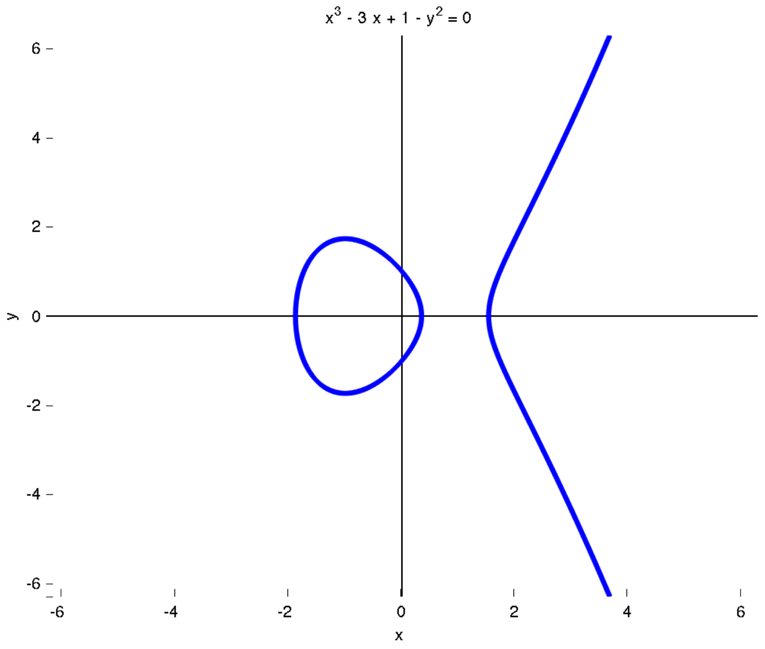


Figure 1: The elliptic curve $y^2 = x^3 - 3x + 1$

Figure 3 shows two singular curves. To define a group operation on the points of the curve, we need to be able to take a tangent line to each point. So we avoid these cases with that additional restriction on the coefficients.

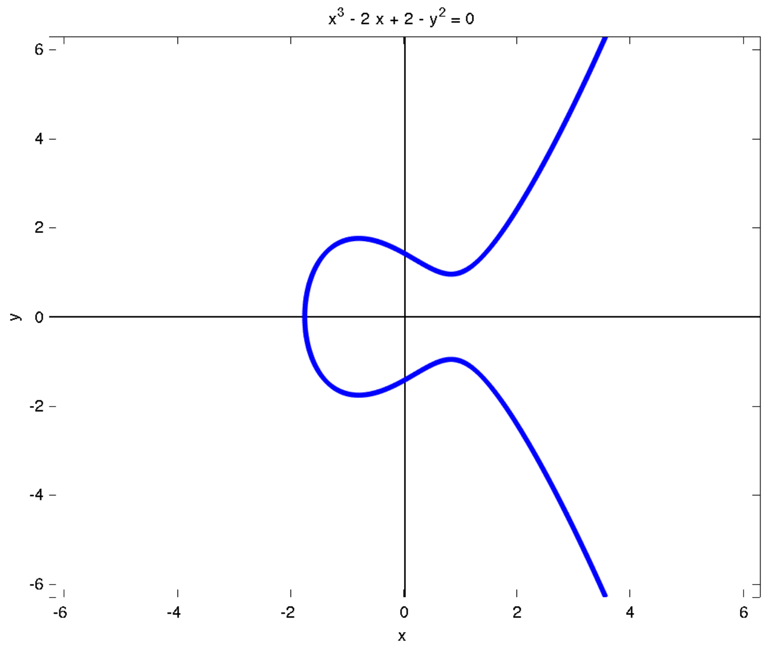
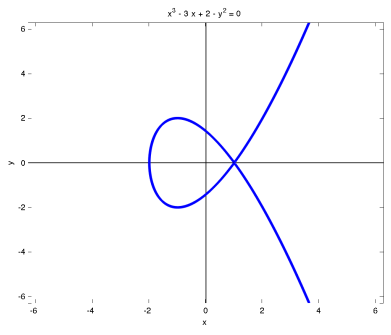
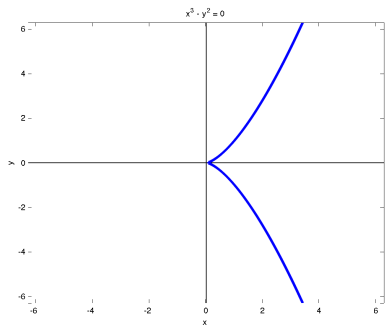


Figure 2: The elliptic curve $y^2 = x^3 - 2x + 2$



(a) The curve $y^2 = x^3 - 3x + 2$



(b) The curve $y^2 = x^3$

Figure 3: These curves have a 'singularity': a point where the tangent is not clearly defined.

1.1 Defining a Group Operation

Now, we have an equation for a curve, and we can define the set of points

$$C = \{(x, y) \mid x, y \in F \text{ and } y^2 = x^3 + ax + b\}$$

which is a subset of of the plane F^2. We want to make the set C into a group, so we need to define an operation on it. Let's call that operation $*$, and we'll define $P_1 * P_2$ as the third intersection of the line through P_1 and P_2 which lies on the curve C. Figure 4 shows this operation for the simple case of two different points.

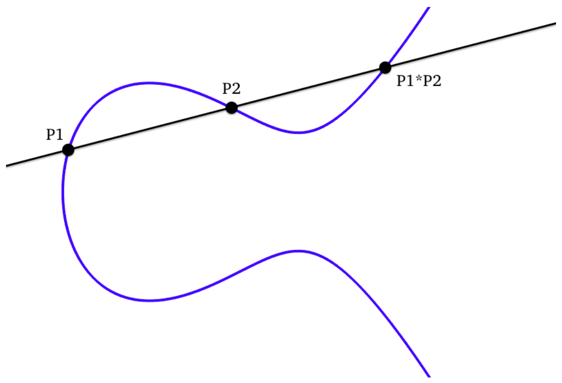


Figure 4: Finding the third point of intersection on the curve $y^2 = x^3 - 2x + 2$

It is not immediately clear that this operation is well-defined. If we consider the curve and line algebraically, we can define their equations in projective space. Then, there are three solutions (counting multiplicity) to the system of equations

$$\begin{aligned} Y^2 Z &= X^3 + aXZ^2 + bZ^3 \\ Y &= \alpha X + \beta Z \end{aligned}$$

So the line and the elliptic curve intersect at three points.

Returning to the affine plane, what other cases do we need to consider? First, we need to define $*$ when $P_1 = P_2$. We want to have the line through P_1 hit the curve at exactly two points, P_1 and $P_1 * P_1$. To achieve this, let the line through P_1 be the tangent to the curve. Then the tangent line intersects the curve at one additional point, as desired (figure 5) [3, §I.2].

Now, we come to the case of a vertical line. A vertical line will intersect our curve at exactly two points, because the curve is symmetric over the x-axis. But those two points will violate our $*$ operation, because there isn't a third-point where the line intersects the curve (figure 6). This leads us to take an idea from projective geometry: that there is an extra point on the curve called the *point at infinity*.

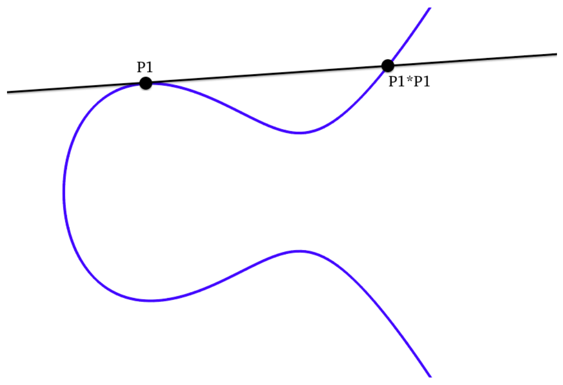


Figure 5: A tangent line intersects the curve at two points.

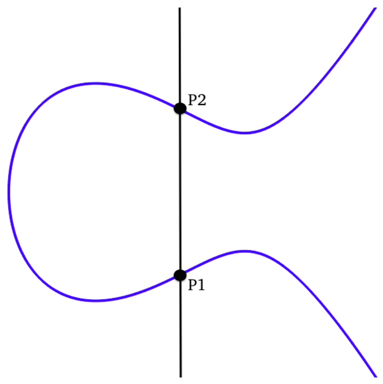


Figure 6: A vertical line only intersects the curve at two points.

The point at infinity (denoted ∞) is a point in the projective plane, so there isn't an intuitive way to draw it in our standard plane. However, we can understand ∞ as 'outside' of the plane, and simply treat it as a special geometric object. As a benefit, we can take the third intersection of a vertical line to be ∞. We then need to redefine the set of points C to include ∞:

$$C = \{\infty\} \cup \{(x, y) \mid x, y \in F \text{ and } y^2 = x^3 + ax + b\}$$

Thus we have disposed of this problematic case [4, §2.2].

Having ∞ on our curve is also useful because it is a unique point which exists on every elliptic curve. This makes it an ideal candidate for the identity element of our group operation. We need to redefine our operation to make this true, however. If we reconsider the case of the vertical line, we have two points P_1 and P_2 such that $P_1 * P_2 = \infty$. Because the curve is symmetric, all we need to do to get P_1 from P_2 is to reflect over the x-axis.

Let's define a new operation $+$ and say that, for any point $P \in C$, $P = P + \infty = \infty + P$. To produce $+$ from our previous operation $*$, we only need to add one additional step: a reflection over the x-axis. We will see later that this operation is commutative, which is why we chose to use the addition symbol [4, §2.2]. Figure 7 revisits the previous cases to show how this new operation works.

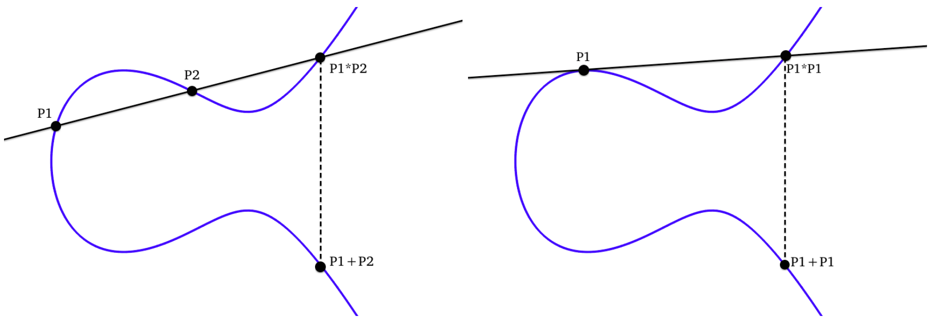


Figure 7: The + operation for two points on an elliptic curve

1.2 Deriving the Group Law

Now that we have defined the group operation geometrically, we can derive a formula for it.

1.2.1 Identity

Define ∞ to be the identity. For any point P on C, $P + \infty = \infty + P = P$.

1.2.2 Addition

Let $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ on C with $P_1 \neq P_2$. If $x_1 = x_2$, then the line is vertical and we define $P_3 = \infty$. Otherwise, the equation of the line through P_1 and P_2 has the form $y = mx + b$, where $m = \frac{y_2 - y_1}{x_2 - x_1}$ and $b = y_1 - mx_1$. Now substitute this into the equation of our curve to obtain

$$(mx + b)^2 = x^3 + ax + b$$

Expand and rewrite to get a cubic equation in x

$$0 = x^3 - m^2x^2 + (a - 2mb)x + b - b^2$$

We know that this equation has three roots (of which we know two), so it must be equal to

$$0 = (x - x_1)(x - x_2)(x - x_3)$$

Multiplying out, we find that the coefficient of the x^2 term has the form $-x_1 - x_2 - x_3$. So $-m^2 = -x_1 - x_2 - x_3$, and we can solve for the third x-coordinate in terms of the first two points:

$$x_3 = m^2 - x_1 - x_2$$

Finally, the third point is reflected across the x-axis. To find the y-coordinate, we substitute our x-coordinate in to the equation of our line and negate it.

$$y_3 = -(mx_3 + b)$$

1.2.3 Doubling

Let P_1 on C. We defined the addition of a point with itself by taking the tangent line to that point on the curve. We can find the slope of the tangent line by implicit differentiation:

$$\begin{aligned} 2y\frac{dy}{dx} &= 3x^2 + a \\ \frac{dy}{dx} &= \frac{3x^2 + a}{2y} \\ m &= \frac{3x_1^2 + a}{2y_1} \end{aligned}$$

Then we simply follow the same process for addition.

$$\begin{aligned} x_2 &= m^2 - 2x_1 \\ y_2 &= -(mx_1 + b) \end{aligned}$$

1.2.4 Inverses

Let $P = (x, y)$ on C. Define $-P = (x, -y)$. The line through P and $-P$ is vertical, and hence intersects the curve at P, $-P$, and ∞. ∞ is a unique point on our curve, and has the special property that "reflecting across the x-axis" (negating the y-coordinate) returns ∞. So $P + (-P) = \infty$.

1.2.5 Closure

Now we have fully defined the group law for an elliptic curve. The closure of the set $\{\infty\} \cup \{(x, y) \mid x, y \in F \text{ and } y^2 = x^3 + ax + b\}$ follows from the group law and the closure of the field F.

1.2.6 Commutativity

The group law is commutative because the line through two points is defined symmetrically. In other words, the line through points A and B is the same as the line through points B and A.

1.2.7 Associativity

We will prove the associativity of the group law geometrically, but it also follows algebraically from the formulas above. To complete the proof, we will use a theorem from projective geometry:

Theorem (Cayley-Bacharach Theorem)**.** *Let C, C_1, and C_2 be three cubic curves. Suppose C goes through eight of the nine points of C_1 and C_2. Then C goes through the ninth intersection point.*

Let P, Q, R on our elliptic curve E. To show that $P+(Q+R) = (P+Q)+R$, it suffices to show that $P*(Q+R) = (P+Q)*R$. Our goal is to define two cubic curves, one of which intersects $P*(Q+R)$ and the other which intersects $(P+Q)*R$. Then, if their other eight points of intersection match, it must be that $P*(Q+R) = (P+Q)*R$.

Let C_1 be the curve through $P*(Q+R)$ and C_2 the curve through $(P+Q)*R$. We will construct C_1 and C_2 directly. Consider a linear equation $f(x, y) = ax + by + c = 0$. Given three such equations f_1, f_2, f_3, we can construct a cubic curve $g(x, y) = (f_1(x, y))(f_2(x, y)(f_3(x, y)) = 0$. Then every zero of g is on one of the lines; so the curve is the union of the points on each of the lines and the point at infinity.

Let's start with C_1 (figure 8). We want to deconstruct the point $P*(Q+R)$ into three linear equations. The first line is the line between Q and R, whose third point of intersection is $Q*R$. The second line is between $Q*R$ and ∞, which intersects the curve at $Q+R$. The third line is between $Q+R$ and P, which intersects the curve at $P*(Q+R)$.

For C_2, we define the lines similarly (figure 9). Line one has intersections P, Q, and $P*Q$. Line two has intersections $P*Q$, ∞, and $P+Q$. Line three has intersections $P+Q$, R, and $P*(Q+R)$. Thus we have two curves C_1 and C_2 which intersect at eight points, so their ninth point of intersection is equal by Cayley-Bacharach. [3, §1.2]

This completes our proof that $(E, +)$ is a group.

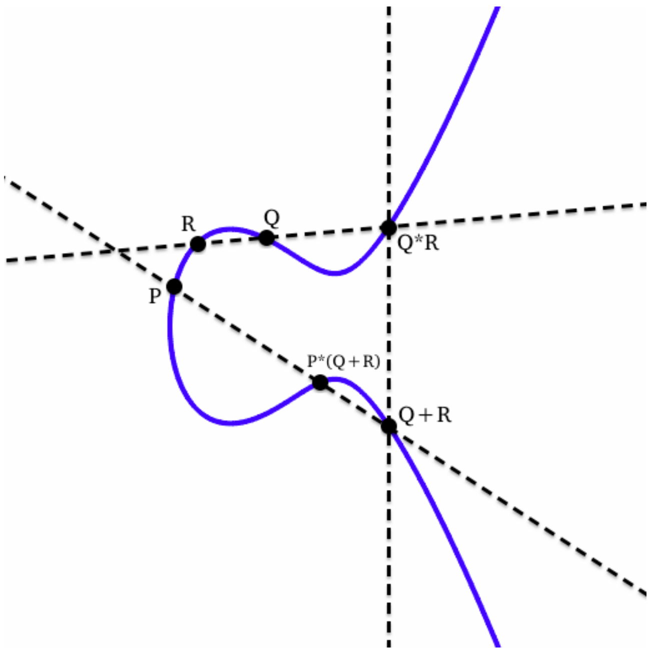


Figure 8: Defining C_1 as the union of three lines

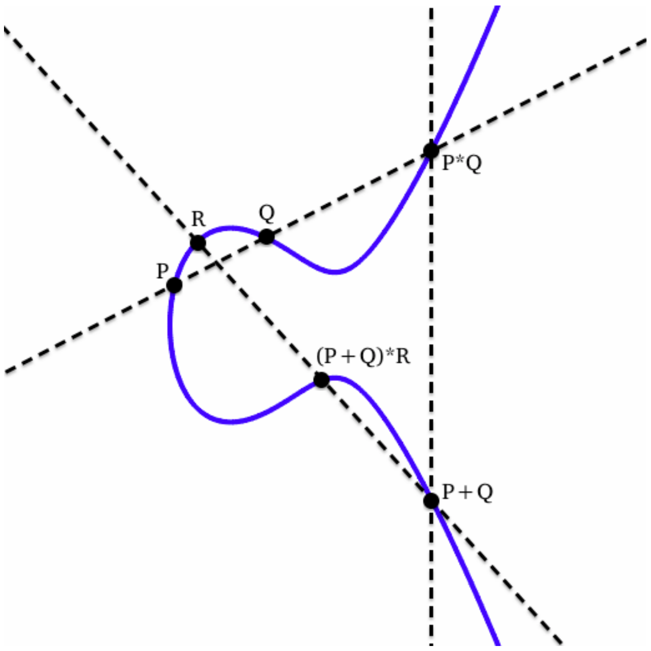


Figure 9: Defining C_2 as the union of three lines

1.3 Weierstrass Form and Admissible Changes of Variables

Before moving on, we will take a moment to revisit the definition of an elliptic curve. Initially, we defined an elliptic curve to have the relatively simple equation $y^2 = x^3 + ax + b$. This type of curve is actually a subset of all elliptic curves, and the equation is in **short Weierstrass form**. To use this equation, we had to restrict our field so that it is not characteristic 2 or 3.

The most general elliptic curve equation takes the form

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

Again, the curve must be non-singular. However, the condition for singularity is significantly more complicated, so it will not be reproduced here. If we *are* working over a field with characteristic different from 2 and 3, we can use an **admissible change of variables** to transform a general curve equation to one in short Weierstrass form.

Performing an admissible change of variables for an elliptic curve amounts to constructing an isomorphism between two general curves. Every admissible change-of-variables takes the form

$$(x, y) \mapsto (u^2x + r, u^3y + u^2sx + t)$$

for some u, s, r, t in our field and $u \neq 0$. The transformation from general equation to short Weierstrass form uses the following map:

$$(x, y) \mapsto \left(\frac{x - 3a_1^2 - 12a_2}{2^2 3^2}, \frac{y - 3a_1x}{2^3 3^3} - \frac{a_1^3 + 4a_1a_2 - 12a^3}{2^3 3} \right)$$

Our restriction to fields with characteristic not 2 or 3 now makes sense: this admissible change of variables requires the inverse of 2 and 3 to be defined [2, §3.1].

2 The Discrete Logarithm Problem

Given a group G with operation $*$ we can define a map $\cdot : \mathbb{Z} \times G \to G$ by

$$n \cdot g \mapsto \underbrace{g * g * \cdots * g}_{n \text{ times}}$$

Let's first consider the case of $\mathbb{Z}_p^*$, the multiplicative group of integers mod p. In this case, our map $\cdot$ is equivalent to exponentiation. Let $a, b \in \mathbb{Z}_p^*$ and assume $\exists n \in \mathbb{Z}$ such that $a^n = b$. Over the real numbers, we call solving for n finding a logarithm. Over $\mathbb{Z}_p^*$, n is the *discrete logarithm* of b with respect to the base a.

For an elliptic curve C, we also have a group structure. In this case, the map $\cdot$ is equal to the repeated addition of a point to itself. We can call this map *point multiplication.* For a point P on C, $nP = P + P + \cdots + P$. Again, there is an analogue to the logarithm. If we have two points P, Q on C where $nP = Q$, then n is the *elliptic-curve discrete logarithm* (ECDL) of Q with respect to P.

2.1 Trap-Door Functions

Why do we bother defining such a simple operation? It turns out that this is the exact operation we will use to construct a cryptosystem. In fact, many modern cryptosystems are based on a class of operations called *trap-door functions.* A trap-door function is a function that is simple to compute in one direction, but very costly to compute in the other direction. In the case of the ECDL, it is simple to compute a point multiplication, but hard to compute the logarithm.

Note that this is not a formal definition. In general, a trap-door function is infeasible to break, given current computing resources. However, most trap-door functions can only be conjectured to be secure. While we currently do not know any algorithms which solve the ECDLP, there is no guarantee that a fast algorithm does not exist. Next, we'll consider a basic approach to solving ECDLP.

2.2 Attacks on Discrete Logarithms

Imagine we're playing a game. I pick a point P on an elliptic curve, and tell you the curve's parameters and the point I picked. Then I pick a secret integer s and tell you sP. Your job is to find s. This is the basic model of a user and attacker in cryptosystem which is based on the ECDL. Given P and sP, can the attacker find s? Let's look at how the attacker might go about doing that.

2.2.1 Naive Multiplication

Our first attempt will be to simply try every integer k. If $kP = sP$ then we know $k = s$. If our curve has only a small number of points, then a computer will chew through this problem in seconds. Let's say our curve has N points, and each point multiplication takes a maximum of M operations. Then naive multiplication has algorithmic complexity $O(NM)$.

From a user's perspective, M should be small, so curve operations are fast. In order to make the attacker's life difficult, N should be large: the chosen curve should have a large number of points. Many curves used for cryptography are over large prime fields, with a massive number of points (more than 2^{200}). Using a naive algorithm against such a curve, even with an extremely fast computer, probably won't finish in our lifetime.

2.2.2 Baby Step Giant Step

There are a number of algorithms which reduce the amount of time it takes to solve the ECDLP. Let's consider an elliptic curve (or a subgroup) which is cyclic, with generator G and order N. The user picks n and computes nG.

The Baby-Step Giant-Step algorithm rewrites the point $P = nG$ as $(im + j)G$, with $m = \lceil\sqrt{N}\rceil$. Then jP is computed for $0 \leq j < m$ and stored. Finally, multiplication is used to find imG for $0 \leq i < m$, which is subtracted from P to solve for jG.

Algorithm 1 Baby-Step Giant-Step for ECDLP

```
$m \leftarrow \sqrt{N}$
for $0 \leq j < m$ do
    Compute and store $(j, jP)$
end for
for $0 \leq i < m$ do
    Compute $Q - imP$
    if $Q - imP = jP$ for some $j$ then
        return $n \equiv j + im$
    end if
end for
```

Baby-Step Giant-Step completes in $O(\sqrt{N}M)$ time and $O(\sqrt{N})$ space complexity. This is a large improvement over naive multiplication, but still very slow on cryptographic curves. While there are some algorithms which solve specific cases of the ECDLP, there is no known algorithm which feasibly solves this problem on a general cryptographic curve. The security of an elliptic curve cryptosystem relies on this fact [4, §5].

3 Elliptic-Curve Diffie-Hellman Exchange

The split between what we recognize as modern and pre-modern cryptography coincides directly with the development of public key cryptography. Public key cryptography is a class of cryptosystems which are defined by the fact that each user has a public key, which they publish for anyone to see, and a private key, which they keep secret. The public key allows anyone who wants to communicate with that user to derive some information which is only shared between those two users, even over an insecure channel.

One of the major problems of pre-modern cryptography was that every method of communication required a shared key. This problem is called the key distribution problem. Essentially, if two parties wanted to communicate they needed to meet and exchange a key - there was no way to communicate over an insecure channel without first communicating over a secure channel. Public key cryptography solves this problem by allowing two parties to communicate without the need of a shared key, or to securely derive one over an insecure channel.

3.1 The Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange is a method of key derivation which allows two users to derive a shared key over an insecure channel. It is typical of cryptography to define communication algorithms as follows: imagine two users, Alice and Bob. Alice and Bob want to communicate securely, but can only talk over an insecure line. Somewhere on the insecure line an attacker, Eve, is listening in and attempting to decrypt their communication.

The **Elliptic Curve Diffie-Hellman Exchange** (ECDHE) proceeds as follows:

Algorithm 2 Elliptic Curve Diffie-Hellman Exchange

1. Alice and Bob agree on a curve C and a generating point P. (In practice, these parameters are standardized and known to everyone.)
2. Alice generates a random, secret integer a and sends Bob aP.
3. Bob generates a random, secret integer b and sends Alica bP.
4. Alice computes $a(bP)$.
5. Bob computes $b(aP)$.
6. Alice and Bob now share the key abP.

At this point, it is clear that Alice and Bob share the point abP. Eve, however, only knows the points aP and bP, which she cannot use to compute abP. Thus for Eve to break the ECDHE, she must solve the ECDL: given aP, find a, or given bP, find b. From now on, Alice and Bob can use a symmetric encryption algorithm (such as AES) to communicate securely. Their shared key is derived from abP. Note that abP is a point on a curve, while most encryption algorithms desire keys to be a string of random bytes. Typically Alice and Bob will use an agreed upon method of key derivation, such as a hash of the x-coordinate, to generate the actual session key [4, §6.2].

3.2 Implementation Details of ECDH

3.2.1 Choice of Curve

We now have all of the necessary information to implement a theoretical cryptosystem. However, one of the most important choices in cryptographic design is the choice of parameters: what curve will we use, over what field, and what will the generating point be? Wrong choices in these areas can result in massive holes in the security of the ECDL. Daniel J. Bernstein and Tanja Lange have produced a database of the numerous choices of curves, as well as a discussion of various security flaws.

For our implementation, we will use Bernstein's *Curve25519*. Curve25519 is an elliptic curve in Montgomery form, over the prime field $\mathbb{Z}_p$ where $p = 2^{255} - 19$. This curve was designed for security: for example, it is not vulnerable to attacks based on subgroups of the curve [1].

3.2.2 Point Multiplication

Performing point multiplications is one consideration that is less important in theory but extremely important in reality. The naive method is to compute nP by summing n copies of P: $P + P + \cdots + P$. This algorithm is exponential, which is to say, extremely slow for a real-world application. We can see an immediate improvement from the decomposition of n into its binary representation. Then, each zero bit of n corresponds to a doubling of the point P, and each one bit of n corresponds to adding P and doubling.

Let $b_m b_{m-1} \cdots b_1 b_0$ be the binary representation of n (where b_m is the most-significant bit). The Double-and-Add algorithm is described below.

Algorithm 3 Elliptic Curve Point Multiplication: Double-and-Add

```
$Q \leftarrow \infty$
**for** $0 \leq i \leq m$ **do**
    **if** $b_i = 1$ **then**
        $Q \leftarrow Q + P$
    **end if**
    $Q \leftarrow 2Q$
**end for**
**return** $Q$
```

While Double-and-Add is much faster than repeated addition, it demonstrates one of the difficulties of real-world cryptography. Specifically, this algorithm leaks information about n by performing a different number of doublings and additions based on the binary representation of n. This information leakage makes an implementation which uses Double-and-Add vulnerable to timing or power analysis. To avoid this vulnerability in practice, an ECDHE implementation will use the Montgomery Ladder, a multiplication algorithm which operates in constant time [2, §3.3].

3.2.3 A Pedagogical Implementation of ECDHE

The result of this document is an implementation of the Elliptic-Curve Diffie-Hellman Exchange, using Curve25519. The code is available in `curve25519.rb`. I have followed Bernstein's specification fairly closely, but there are some deviations. The code is designed for clarity and understanding, so my code deviates in the following ways:

- Use of (x, y) coordinates and formulas, rather than XZ coordinates
- Use of Double-and-Add for point multiplication rather than Montgomery Ladder
- Use of SHA256 for key derivation rather than Salsa20

The use of (x, y) coordinates and the Double-and-Add algorithm are equivalent to Bernstein's choices, but cause some penalties in speed and security. The use of SHA256 as a hashing algorithm is not equivalent to Bernstein's choice, but represents a widely-available hash which is an acceptable choice for key derivation.

References

[1] Daniel J. Bernstein: *Curve 25519: new Diffie-Hellman speed records*. Published online, 2006.

[2] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone: *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[3] Joseph Silverman and John Tate: *Rational Points on Elliptic Curves*. Springer, 1994.

[4] Lawrence Washington: *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall/CRC, 2nd edition, 2008.