CS 198 Codebreaking
Fall 2022 Ryan Cottone, Imran Khaliq-Baporia

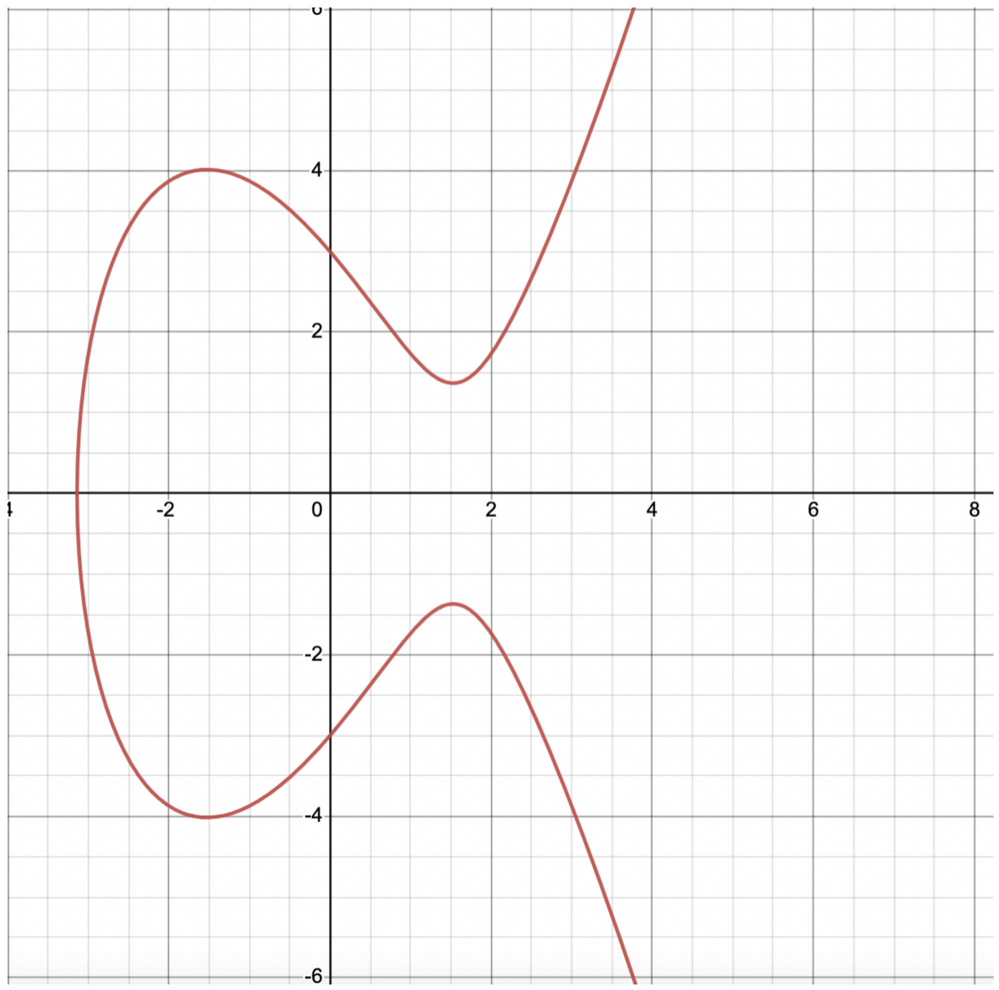
Note 10

Elliptic Curves

Elliptic curve cryptography does operations on an **elliptic curve**, which are equations of the form

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

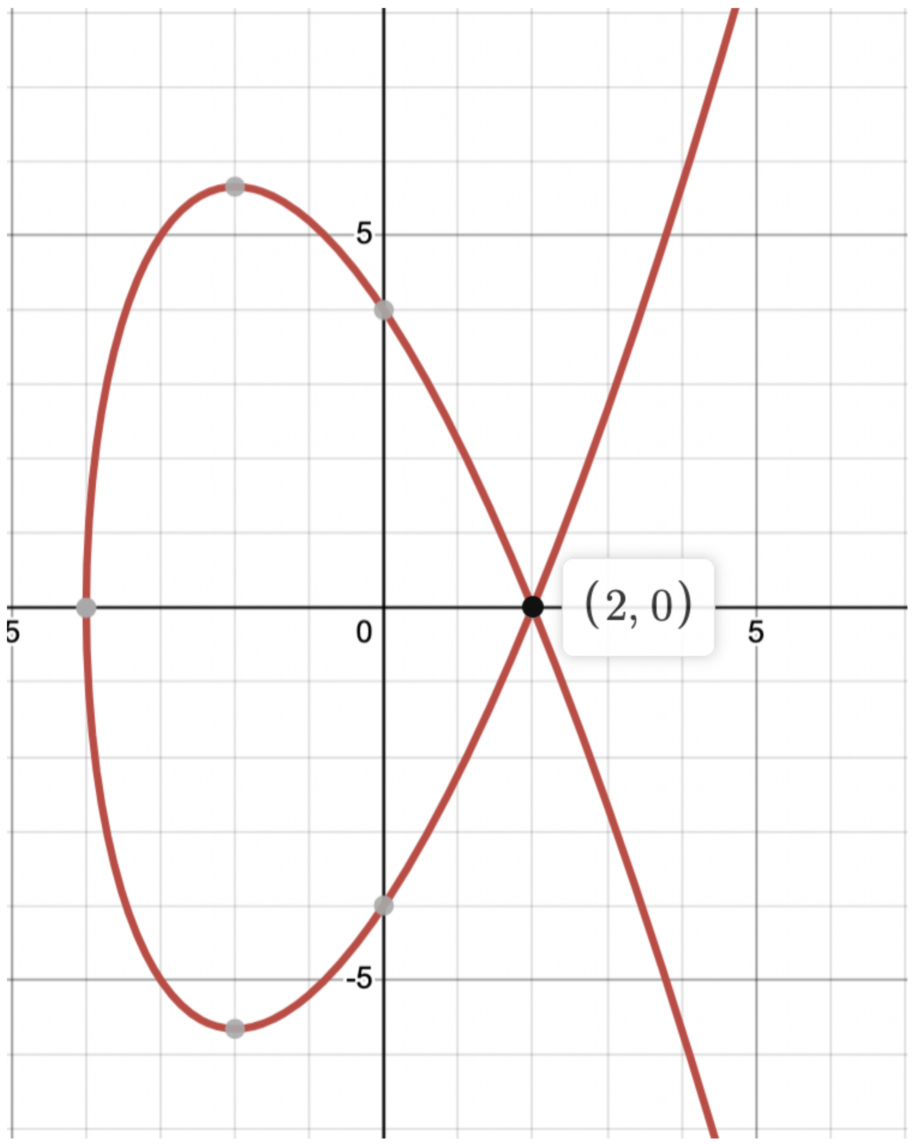
where we choose the coefficients a, b. They look something like this:



In this case, we plotted the curve $y^2 = x^3 - 7x + 9$. There is another key property of elliptic curves we need to test for: **a nonzero discriminant**. The discriminant of an elliptic curve is defined as

$$4a^3 + 27b^2$$

If this is equal to zero, we will end up with a curve like this ($y^2 = x^3 - 12x + 16$):



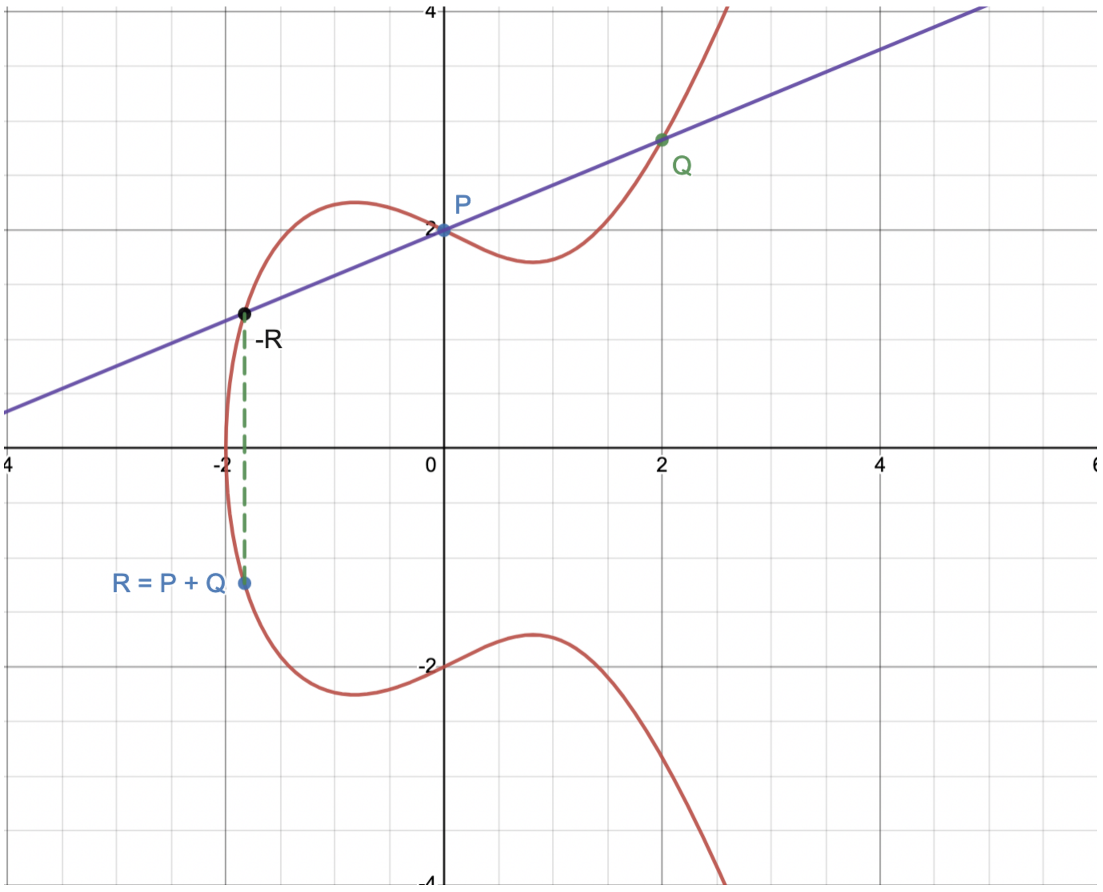
The point at $(2,0)$ is actually a *double root*, which throws a wrench into many nice properties of elliptic curves. Curves with a zero discriminant will always have at least one double root (more generally, it will not be singular).

We also need to define a **point at infinity** as O, which we can think of as some magical point at $(\infty, \cdot)$ and $(-\infty, \cdot)$. We will see how this comes into play shortly.

Elliptic Curve Operations

First we need a way to evaluate a point on the curve, $f(x) = y$ for (x, y). We do this by plugging x into into above polynomial, and taking the square root to find $(x, \sqrt{y})$ and $(x, -\sqrt{y})$ (remember the equation is of the form $y^2 = ...$). Both of these points will be on the curve.

We also define a single operation: *point addition*, between points P, Q as drawing a line between P and Q on the curve, finding where that line intersects the curve, and reflecting it across the xaxis. Denote this as $P \oplus Q$. Here's an example addition between $P = (0, 2)$ and $Q = (2, \sqrt{8})$ over $y^2 = x^3 - 2x + 4$:



Our final result is $R = P \oplus Q$ at $\approx (-1.828, -1.234)$. We can find closed-form equations for this type of operation, where $P = (x_1, y_1), Q = (x_2, y_2), P \oplus Q = (x_3, y_2)$:

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

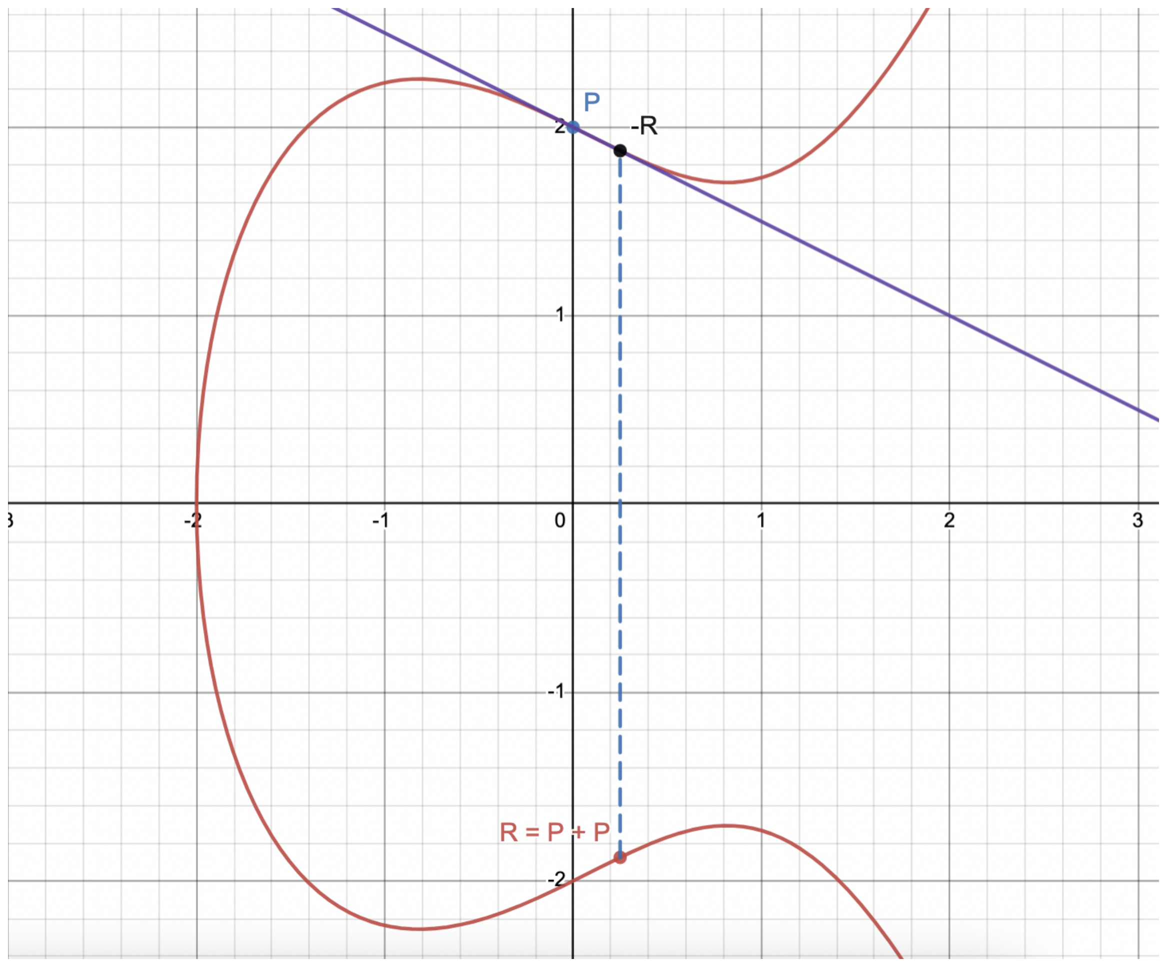
However, if we are working $\mod p$, we instead need to set

$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1} \mod p$$

since we have to use the modular inverse instead of dividing.

There's a few more edge cases, however: $P \oplus P$ and $P \oplus O$.

Adding P to itself involves taking the tangent line and doing the same operation as before:



Since $x_2 = x_1$ here, we can't use the same addition equation. Instead, we have:

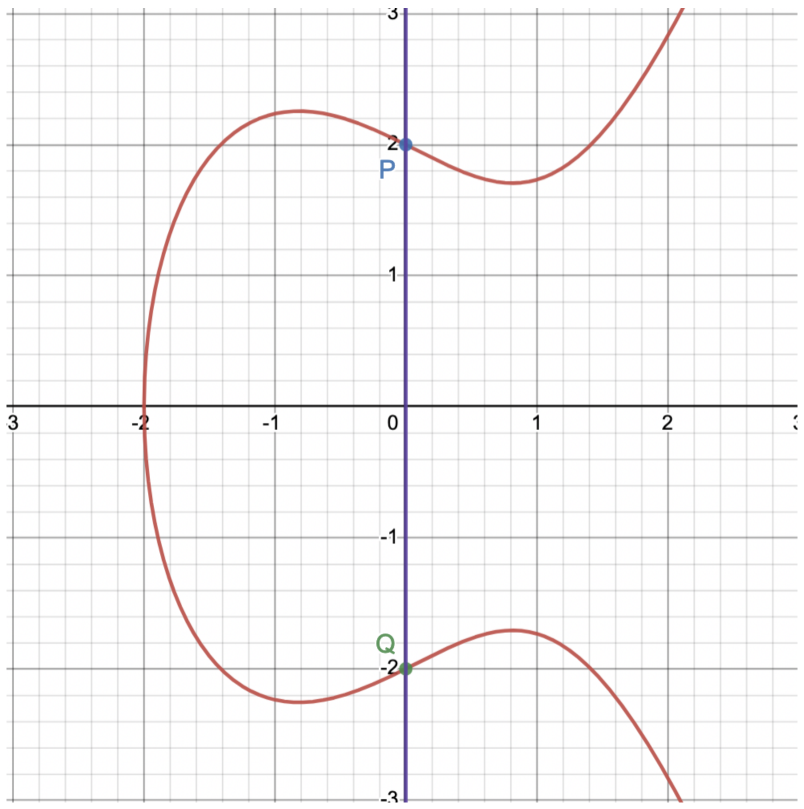
$$\lambda = \frac{3x_1^2 + A}{2y_1}$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

As before, use

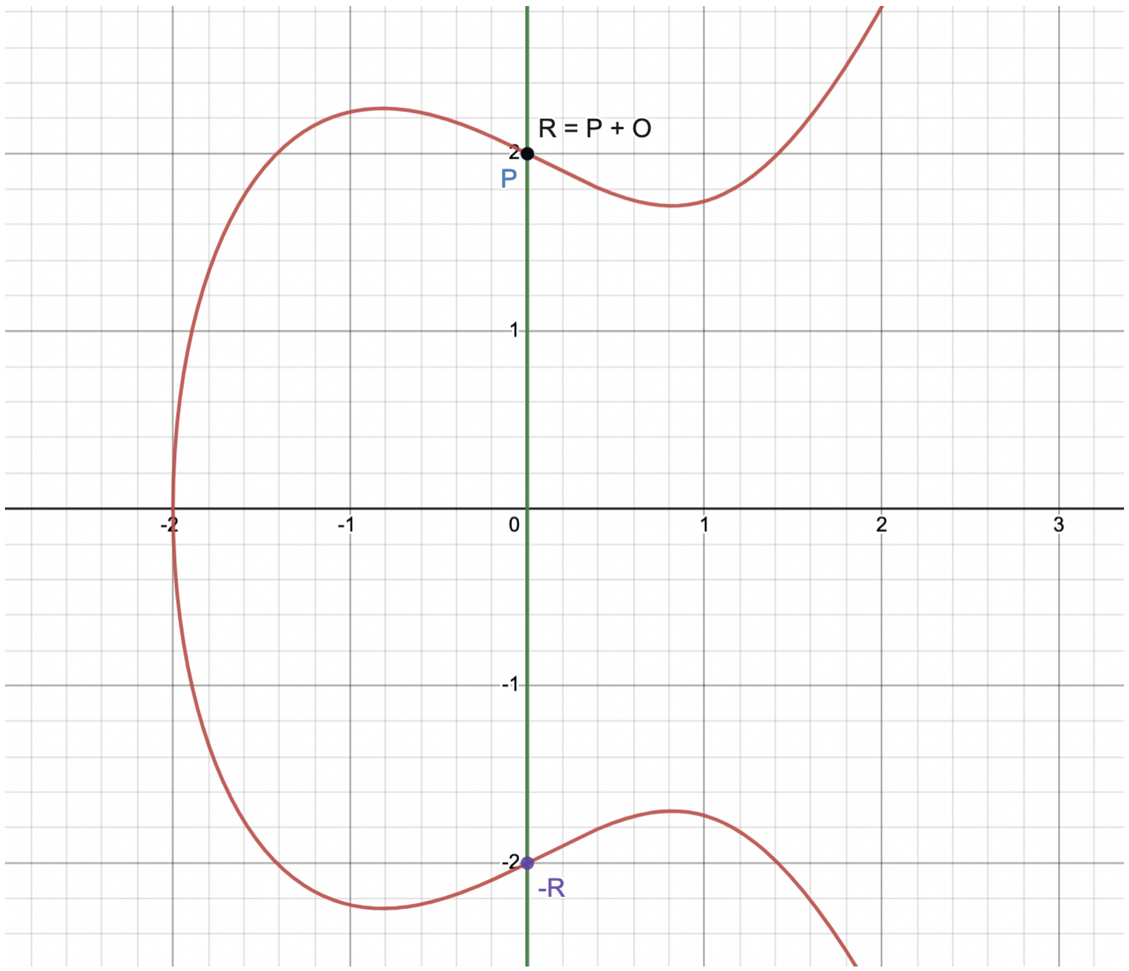
$$\lambda = (3x_1^2 + A)(2y_1)^{-1} \mod p$$

if you are working $\mod p$.

Where does O come in, however? Consider adding two points that share the same x coordinate but have different y coordinates, like $P = (0,2)$ and $Q = (0,-2)$. When we try to draw a line between them, it will be perfectly vertical and never intersect the curve at a third spot! We say the result of this addition is O, at "point at infinity".



Crucially, $P \oplus O = P$ for any point P. You can visualize this as:



Since the line between P and O is a vertical line, the third intersection is always $-P$, which is then reflected back to P!

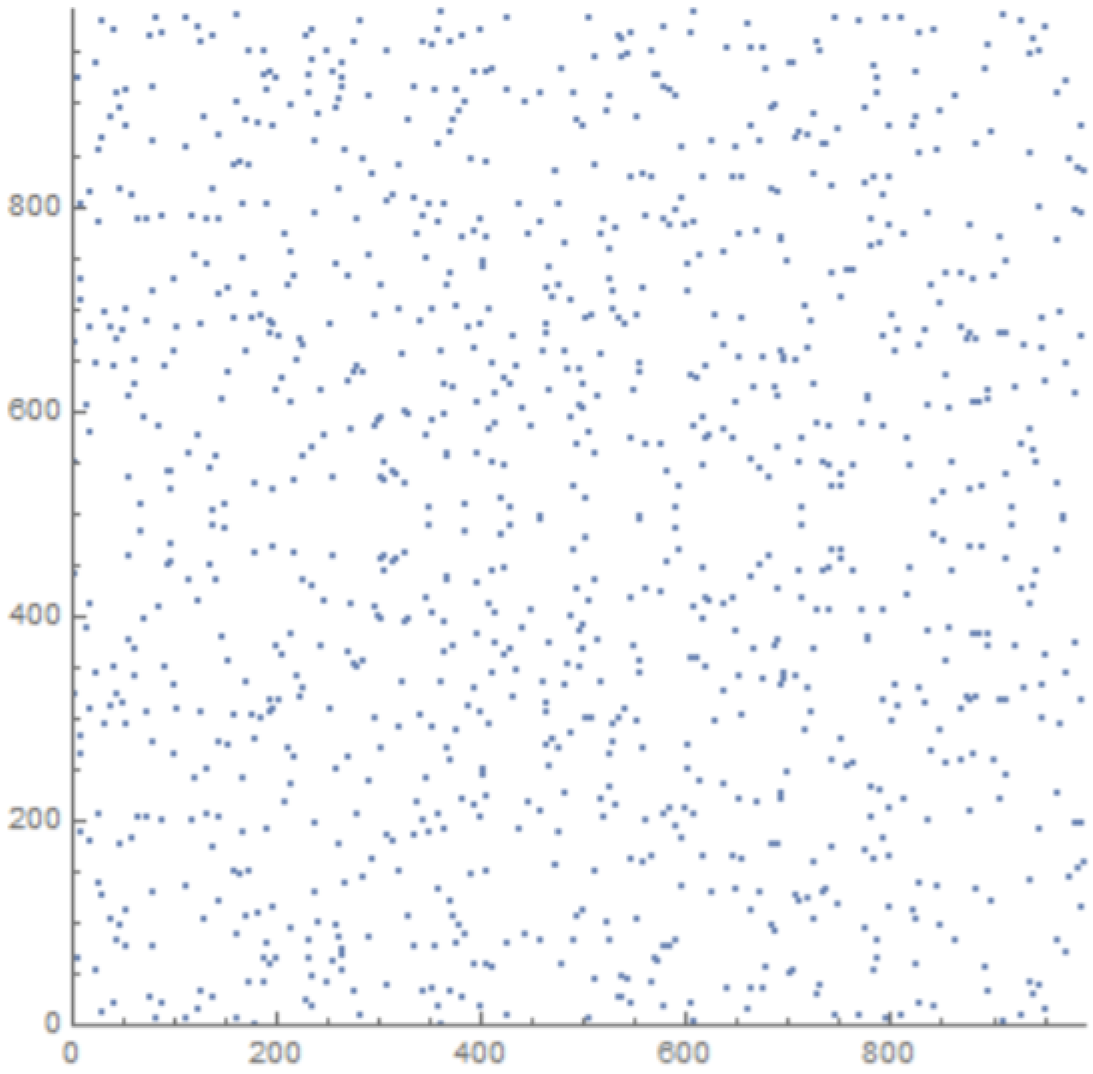
We can define scalar multiplication over elliptic curves as **repeated self-addition**. Take for example $3P$. This represents $P \oplus P \oplus P$.

Finite Field Elliptic Curves

Much like normal arithmetic, normal elliptic curves are not very useful in cryptography, since their operations are trivially reversible. Instead, we can use our old friend modular arithmetic to reduce elliptic curve operations $mod p$!

This does not involve much more work on our end – just reduce the intermediate results of the points mod p, such that $P = (x, y) = (x \mod p, y \mod p)$. This also have some implications for the aforementioned addition equations – instead of dividing, we have to multiply by the modular inverse.

Since we reduce modulo some prime, our set of possible points might look like this:



It would be much harder to reverse an operation from this mess of points! As a result, elliptic curve addition is roughly equal to multiplication in modular arithmetic. Consequently, multiplication in elliptic curves is roughly equal to exponentiation in modular arithmetic!

This means that nP for integer n is roughly as hard to calculate as $x^n \mod p$ (that is, easy if you know n) and very hard to recover n given the final result (discrete logarithm problem). Recovering n given nP is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). It is actually considered **harder** to solve than the DLP over modular fields!

Elliptic curves turn out to be far more secure than normal RSA and equivalents. In fact, a 2048-bit RSA key is roughly equivalent to a **224 bit** ECC key! For this reason, it is very popular to use in modern-day encryption.

Double-and-Add Algorithm

We stated previously that computing nP was efficient for even very large n. How is this possible? It turns out that we can use a very similar algorithm to **modular exponentiation** to accomplish this. We first note that elliptic curve arithmetic is **distributive**:

$$(a+b)P = aP \oplus bP$$

Knowing that, we can take any n and separate it into powers of 2, much like binary. For example $19 = 16+2+1 = 2^4+2^1+2^0$. Therefore,

$$19P = 16P+2P+1P$$

Knowing this, we recognize that adding a point to itself (2P) doubles its coefficient: $2(8P) = 16P$ for example. Adding a point to itself is a very fast operation as we saw before. We can start from P and calculate $2P$, then $4P$, $8P$, etc in $\log_2 n$ time.

Once we have calculate all powers of 2 times P less than n, we use the binary expansion of n to figure out which ones to add. In our 19, example the expansion is 10011, so we add the first, second, and fourth results $(P, 2P, 16P)$ to get $19P$. This is accomplished in a very fast manner, even for extremely large n.

Elliptic Curve Diffie-Hellman

Now to showcase perhaps one of the two most useful applications of elliptic curves: Diffie-Hellman! We covered the Diffie-Hellman key exchange before using modular arithmetic, but using the elliptic-curve version provides more security with less key bits. This is why ECDH is used over modular-arithmetic based DH in most modern systems.

As a quick refresher, Diffie-Hellman involves two parties (Alice and Bob) trying to derive a shared secret without an eavesdropper (Eve) figuring out the secret. Alice and Bob share the public parameters of (g,N) respectively, where g is a generator over $\mod N$.

Alice and Bob then generate the secrets a and b respectively. They find $g^a \mod N$ and $g^b \mod N$ using them, and share both aforementioned values. Once received, Alice finds $(g^b)^a \equiv g^{ab} \mod N$ and Bob finds $(g^a)^b \equiv g^{ab} \mod N$, which is their new shared secret.

Recall our parallels from earlier – multiplication in elliptic curves is roughly equal to exponentiation in modular arithmetic. So, aP for a generator point P over $\mod N$ can be thought of as similar to finding $g^a \mod N$. (A generator point is a point that generates many unique points before wrapping around to itself) Crucially, scalar multiplication of elliptic curves is associative, so $b(aP) = a(bP) = (ab)P$. With that, we can identify the ECDH cryptosystem:

1) Alice and Bob share a generator point P and the modulus N.
2) Alice generates $a \in [2, n-1]$ where $n = \text{ord}(P)$ (This part is not too important to understand, just think of order as the number of unique points we can generate starting at P. We don't want a scalar value larger than that, or it will wrap around mod the order of P)

3) Bob does the same as Alice in 2) for a different b.
4) Alice finds aP, and Bob finds bP using the double-and-add algorithm (or similarly efficient algorithm)
5) Alice and Bob both send aP and bP over the insecure channel.
6) Alice receives bP and computes $a(bP) = (ab)P$.
7) Bob receives aP and computes $b(aP) = (ab)P$.

Since Eve only has aP and bP, she can only find $aP + bP = (a+b)P$, not $(ab)P$. ECDH is extremely similar to its modular arithmetic counterpart! The computation problem of finding $(ab)P$ from aP and bP is known as the **Elliptic Curve Diffie-Hellman Problem (ECDHP)**, and is believed to be computationally hard.

Contributors:

- Ryan Cottone