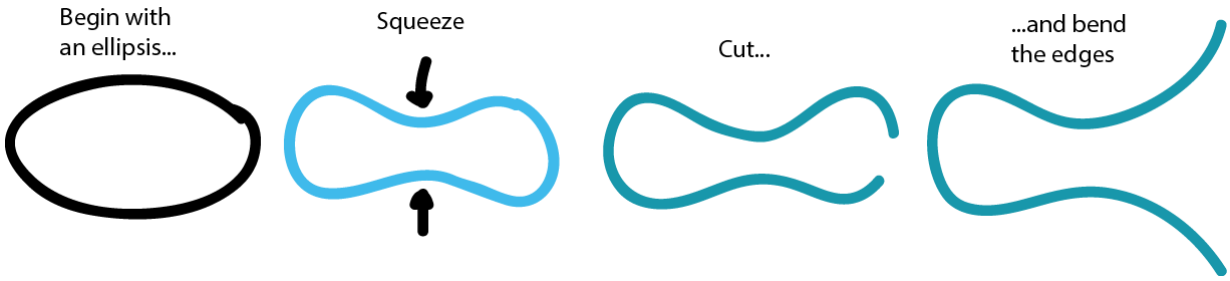
# Elliptic curve cryptography

Elliptic Curve Cryptography (or *ECC* for short) is cryptography based on arithmetics over elliptic curves.

What is an elliptic curve? Imagine that you have an ellipsis. If you squeeze it on the middle, and cut it open on the right end, bending the two ends of the ellipsis outward, you get an elliptic curve. The curve might look a bit different depending on how much you "squeeze and "bend".




> **Definition of an elliptic curve**
>
> An elliptic curve is described by the Weierstrass equation $y^2 = x^3 + ax + b$ for a particular $a, b \in \mathbb{N}$.

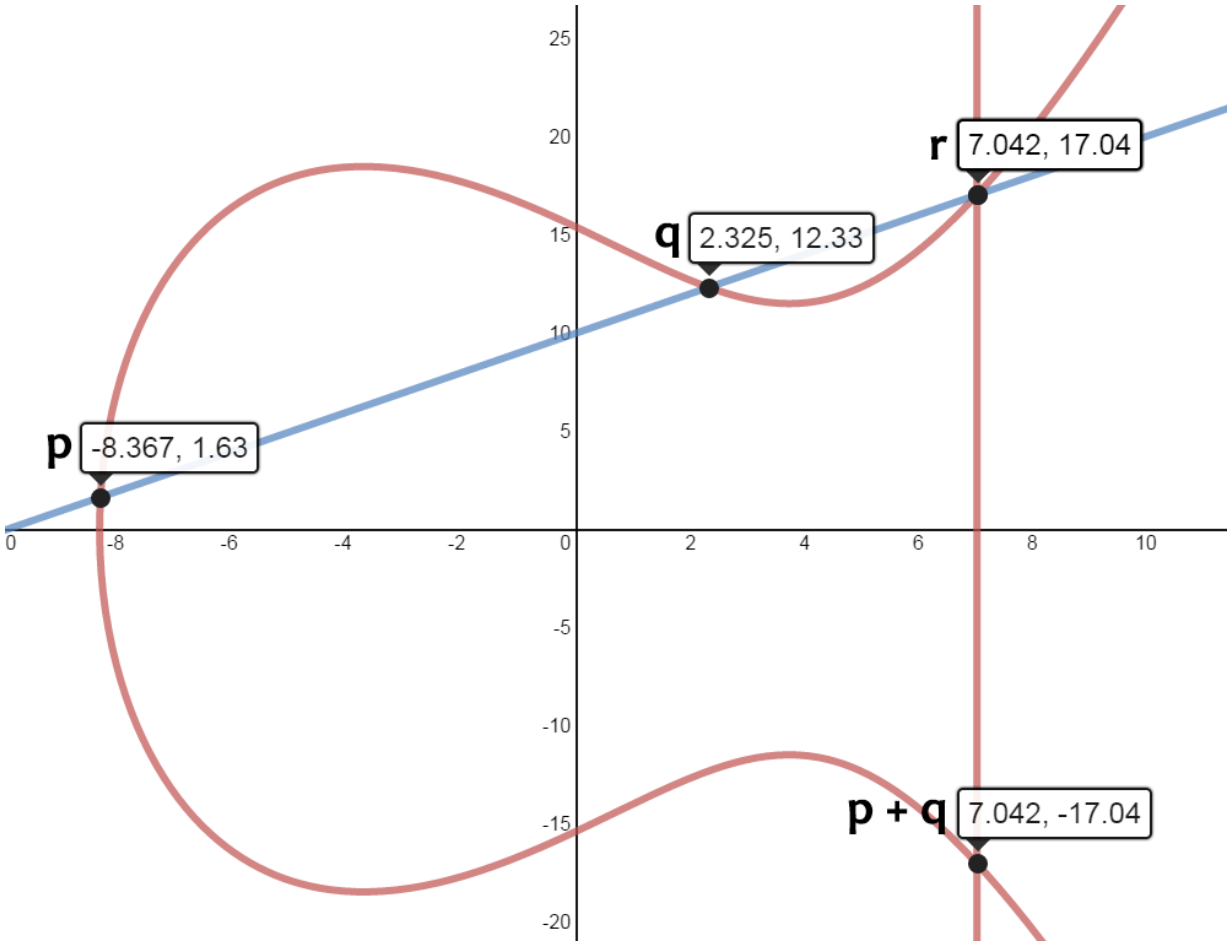
From any (smooth) elliptic curve, we can construct an algebraic group $\mathbb{G}$ consisting of all points $(x, y)$ on the curve. Remember that this group with unary operator $+$ should have the following properties:

1. **Closure** – $\forall a, b \in \mathbb{G} : a + b \in \mathbb{G}$
2. **Associativity** – $\forall a, b, c \in \mathbb{G} : (a + b) + c = a + (b + c)$
3. **Identity element** – $\exists e \in \mathbb{G}; \forall a \in \mathbb{G} : a + e = e + a = a$
4. **Inverse** – $\forall a \in \mathbb{G} : \exists a^{-1}; a + a^{-1} = a^{-1} + a = e$

Before verifying that $\mathbb{G}$ is a group we will try to define the unary operator $+$. This operator has a very nice graphical interpretation over $\mathbb{R}$, so let us go from there.

# Point addition

Say we want to calculate the quantity $p + q$ over the curve $E$, i.e we want to add the points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ on $E$. To do this, we draw a line $L$ crossing $p$ and $q$. Since the curve is a polynomial of third degree, this line will (with one exception) intersect $E$ at a third point $r = (x_r, y_r)$. We then define $p + q$ as $(x_r, -y_r)$.




The coordinates $x_r$ and $-y_r$ can be described using the formula $x_r = k^2 - x_p - x_q$ and $-y_r = k(x_p - x_r) - y_p$ where $k = \frac{y_a - y_b}{x_a - x_b}$ and $a \neq b$.

When $p = q$ we use $E'(x_p)$ to calculate the slope as $k = \frac{3x_p^2 + a}{2y_p}$. This is called point doubling, and it is a special case of point addition. Here, the variable $a$ comes from the Weierstrass equation.

If $-y_p = y_q$ and $x_p = x_q$ we will end up with a vertical line, without a third intersection point. To fix this "problem" we introduce a point $\mathcal{O}$ outside the curve, called *the point at infinity*. We then say that $\mathcal{O}$ is the sum of these two points.

The point at infinity can be represented in any way you like, as long as it is not a point on the curve. It might be more convenient to use homogeneous coordinates. However, if we stick with carthesian coordinates, we can pick $\mathcal{O}$ as the point $(0, 0)$.

# Finite field

In practice, we will not do the calculations over $\mathbb{R}$, but over a finite field. Typically you choose a prime field $\mathbb{Z}_p$ where $p$ is a prime number, or a binary field $\mathbb{Z}_{2^m}$ for some $m \in \mathbb{N}$. We can still use the nice formulae mentioned above, but the geometrical representation of $+$ will no longer make any sense.

Also, instead of division with $x$, we do multiplication with the multiplicative inverse of $x \bmod p$ (if working in $\mathbb{Z}_p$). This routine is denoted with `modInv(x)` in the psuedocode below, and is typically implemented using the Extended Euclidean Algorithm.

Here follows psuedocode for the implementation of $+$ over the finite field $\mathbb{Z}_p$.

```
add(p, q)
    if p = INF
        return q
    if q = INF
        return p
    if p.x = q.x && -p.y = q.y
        return INF
    int k // slope
    if p = q
        k = 3p.x^2 + a * modInv(2p.y)
    else
        k = p.y-q.y * modInv(p.x-q.x)
    x = k^2 - p.x - q.x mod p
    y = k(p.x - x) - p.y mod p
    return (x, y)
```

# Smooth elliptic curves are groups

The elliptic curves from which we can construct groups, are called *smooth*. To determine if

a curve is smooth, we would look for singular points. A *singular point* is a point $(x, y)$ such that $y = 0 \wedge f(x) = 0 \wedge f'(x) = 0$, i.e $f$ contains a double root. A curve is smooth only if it does not contain any singular points. More generally, we can use a discriminant to determine if a curve is smooth.

> **Smooth curves**
> An elliptic curve is called smooth only if $-4a^3 - 27b^2 \neq 0$.

For any such curve, we can verify that the points $(x, y)$ on the curve with the unary operator $+$ forms a group.

1. **Closure** – Follows from the definition of $+$.
2. **Associativity** – Can be proved with Bézout's theorem.
3. **Identity element** – $\mathcal{O}$ is the identity element.
4. **Inverse** – For a point $(x, y)$ we can construct the inverse as $(x, -y)$.

# Trapdoor function

The trapdoor function in ECC is based on the intractability of *the discrete logarithm problem*. The discrete logarithm problem for a finite field $\mathbb{F}$ is to find the secret number $s \in \mathbb{F}$, given $g \times s$. Here $g$ denotes a point on the curve, acting as a generator for the group, and $\times$ denotes repeated addition.

Repeated addition can be thought of as multiplication of a point on the curve with an integer, resulting in another (seemingly random) point. Repeated addition can be made fast in software using the double-and-add approach sketched below.

```
multiply(g, s)
    assert(s > 0)
    q = INF
    // b should contain the unsigned binary representation
    // of s, e.g if s = 9 then b = 1001
    b = s.toBinary
    for i = b.length - 1; i >= 0; --i
        if b[i] = 1
            q = add(q, g)
        g = add(g, g)
    return q
```

Double-and-add should **NOT** be used on a machine which might be monitored by an adversary since the algorithm leaks timing information. Thus, it would be possible for an adversary to mount a side-channel attack and recover information about $s$.

A trapdoor function is commonly used in asymmetric encryption schemes, with a public and a private key. The secret value $s$ is typically the private key, while the product $g \times s$ is chosen as the public key.

# Standard curves

As we have previously seen, an elliptic curve over the finite field $\mathbb{Z}_p$ is described by the tuple $(a, b, n, g, p)$ where $a, b$ describes the shape of the elliptic curve, $n$ is the order of the group (the number of points on the curve), $g$ is a generator (base point) for the curve, and $p$ is a prime defining the finite field $\mathbb{Z}_{\mathrm{p}}$.

There are many, more or less, standardized curves. Particularly the curves recommended by NIST[1], and SECG[2].

---

1. National Security Agency: Mathematical routines for the NIST prime elliptic curves ↩
2. Certicom Research: Standards for Efficient Cryptography - Recommended Elliptic Curve Domain Parameters, Ver. 1 ↩