mini_ecdsa

Basics of elliptic curves and ECDSA in Python.

Disclaimer: There is a lot of brute forcing going on here. This is a *low performance* cryptography package not intended for actual use.

Start by defining a Weierstrass curve over a field of characteristic p, or over the rationals. For example, let's say you want to play with the curve y^2 = x^3 + 2x^2 + 1 over F_7.

```
C = Curve(2,0,1,7)
```

To see a list of all the points on the curve, use `C.show_points()`. This will produce a pretty printed set of points. To return a list of point objects, use `C.get_points()`.

If you want to work over the rational numbers and see the finite order rational points on the curve (torsion points), use a 0 in the last argument when you define the curve. After defining the curve, call `C.torsion_group()`. *Warning:* Attempting to find the torsion points on a curve over Q will fill your machine's memory up and cause it to hang unless the curve has a very small discriminant, so be careful. Let's say we want to see the group of torsion points on the curve y^2 = x^3 + x + 2.

```
C = Curve(0,1,2,0)
C.torsion_group()
```

You'll see that it's a cyclic group of order four, as well as the coordinates of all the torsion points. You can also add points, multiply them by scalars, and so on. Try defining some points (make sure they're actually points on the curve) and messing around.

```
C = Curve(0,1,2,0)
P = Point(-1,0)
Q = Point(1,2)
print C.add(P,Q)
print C.mult(Q,4)
```

To use ECDSA to create digital signatures, we first need to publicly agree on a curve over a prime characteristic field with a distinguished point that generates a prime order subgroup. Let's use P = (1341,854) on the curve y^2 = x^3 + x + 1 over F_2833. This point generates a subgroup of order 131. You can check that this point is on the curve using `C.contains(P)` and that its order is 131 using `C.order(P)`.

```
C = Curve(0,1,1,2833)
P = Point(-1,0)
C.contains(P)
C.order(P)
```

A digital signature generated by ECDSA will consist of a point Q on the curve, which is a multiple of P, as well as two values r and s. This signature is returned as a tuple, and printed when the message is signed.

```
m = 'this is a message'
S = sign(m,C,P,order(P))
```

We can then verify that the message is authentic.

```
verify(m,C,P,order(P),S)
```

If you modify the message m in some way before verifying it, the verification procedure will (with very high probability) return False.