Review. Addition $P \boxplus Q$ of points P, Q on an elliptic curve.

For cryptographic purposes, elliptic curves are usually considered modulo a (large) prime p.

Example 219. Let us consider $y^2 = x^3 - x + 9$ (the elliptic curve from the previous examples) modulo 7. List all points on that curve.

Solution. Note that, because we are working modulo 7, there are only 7 possible values for each of x and y. Hence, we can just go through all $7^2 = 49$ possible points (x, y) to find all points on the curve.

Or, better, we go through all possibilities for x (such as $x = 2$) and determine the corresponding possible values for y (if $x = 2$, then $y^2 = 2^3 - 2 + 9 = 15 \equiv 1 \pmod 7$ which has solutions $y \equiv \pm 1 \pmod 7$).

Doing so, we find 9 points: $O, (0, \pm 3), (\pm 1, \pm 3), (2, \pm 1)$.

[Recall that O is the special point "at ∞" which serves as the neutral element with respect to $\boxplus$.]

Comment. A theorem of Hasse–Weil says that the number of points on an elliptic curve modulo p is always close to p (this is indeed what we expect because, for each of the p choices for x, we get an equation of the form $y^2 \equiv a \pmod p$ which has 2 solutions if a is a nonzero quadratic residue [and for a random a the odds are about 50% that it is quadratic]). Moreover, we can compute the exact number of points very efficiently.

By taking everything modulo 7, we still have the previously introduced addition rule $\boxplus$.

For instance. $(0, 3) \boxplus (1, -3) = (35, 207) \equiv (0, -3) \pmod 7$

Here is how we can use Sage to list all points, as well as add any two of them:

```
>>> E7 = EllipticCurve(GF(7), [-1,9])
>>> E7.points()
```

$[(0:1:0), (0:3:1), (0:4:1), (1:3:1), (1:4:1), (2:1:1), (2:6:1), (6:3:1), (6:4:1)]$

```
>>> E7(0,3) + E7(1,-3)
```

$(0:4:1)$

```
>>> E7(1,-3) + E7(0,-3)
```

$(6:3:1)$

Multiples of a point are simply denoted with nP. For instance, $3P = P \boxplus P \boxplus P$.

We then have a version of the **discrete logarithm** problem for elliptic curves:

(discrete logarithm) Given P, xP on an elliptic curve, determine x.
(computational Diffie–Hellman) Given P, xP, yP on an elliptic curve, determine $(xy)P$.

Comment. Interestingly, it appears that the computational Diffie–Hellman problem (CDH) is more difficult for elliptic curves modulo p than for regular multiplication modulo p. Indeed, suppose that p is an n-digit prime. Then the best known algorithms for regular CDH modulo p has runtime $2^{O(\sqrt[3]{n})}$, whereas the best algorithm for the elliptic curve CDH modulo p has runtime $\sqrt{p} \approx 2^{n/2} = 2^{O(n)}$.

As a consequence, it is believed that a smaller prime p can be used to achieve the same level of security when using elliptic curve Diffie–Hellman (ECDH). In practice 256bit primes are used, which is believed to provide security comparable to 2048bit regular Diffie–Hellman (DH); this makes ECDH about ten times faster in practice than DH.

Comment. On the other hand, due to that reduced bit size, quantum computing attacks on elliptic curve cryptography, if they become available, would be more feasible compared to attacks on ElGamal/RSA.

Armin Straub
straub@southalabama.edu

Comment. Apparently, more than 90% of webservers use one specific, NIST specified, elliptic curve: P-256:

$$y^2 = x^3 - 3x + 41058363725152142129326129780047268409114441015993725554835256314039467401291,$$

taken modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ (the fact that $p \approx 2^{256}$ makes the computations on the elliptic curve much faster in practice). The initial point $P = (x, y)$ on the curve has huge coordinates as well.

Using this single curve is sometimes considered to be problematic, especially following the concerns that the NSA may have implemented a backdoor into Dual_EC_DRBG, which was a NIST standard 2006–2014.

https://en.wikipedia.org/wiki/Dual_EC_DRBG

A popular alternative is the curve Curve25519. In addition to some desirable theoretical advantages, its parameters are small and therefore not of similarly mysterious origin as the ones for P-256:

$$y^2 = x^3 + 486662x^2 + x, \quad p = 2^{255} - 19, \quad x = 9.$$

[Instead of points with (x, y) coordinates, one can actually work with just the x-coordinates for an additional speed-up.]

https://en.wikipedia.org/wiki/Curve25519

```
>>> E = EllipticCurve(GF(2^255-19), [0,486662,0,1,0])
```

```
>>> E
```

$y^2 = x^3 + 486662\,x^2 + x$

```
>>> E.order()
```

57896044618658097711785492504343953926856930875039260848015607506283634007912

```
>>> log(E.order(),2).n()
```

255.000000000000

```
>>> P = E.lift_x(9)
```

```
>>> P
```

(9: 43114425171068552920764898935933967039370386198203806730763910166200978582548: 1)

```
>>> 100*P
```

(44032819295671302737126221960004779200206561247519912509082330344845040669336: 8626006\
39244757237163427806001665957575078127166632317389150490196167274334 4: 1)

```
>>> P.order()
```

7237005577332262213973186563042994240857116359379907606001950938285454250989

```
>>> log(P.order(),2).n()
```

252.000000000000

```
>>> E.order() / P.order()
```

8

```
>>> 5*(20*P) == 20*(5*P)
```

1

On the other hand, it should be pointed out that it is not an easy task to "randomly generate" cryptographically secure elliptic curves plus suitable base point. That's the reason pre-selected elliptic curves are of importance.

http://blog.bjrn.se/2015/07/lets-construct-elliptic-curve.html