Elliptic curves modulo primes

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 (or, maybe, even 3072bit) 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

Specific elliptic curves in wide use

It is not an easy task to "randomly generate" cryptographically secure elliptic curves plus suitable base point. That is a reason why pre-selected elliptic curves are of practical importance.

The following are a few examples of specific elliptic curves that are widely used in practice.

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

Example 220. For signing transactions, Bitcoin uses the elliptic curve

$$y^2 = x^3 + 7 \pmod{p}, \quad p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1,$$

together with the base point $P = (P_x, P_y)$ such that, in hexadecimal,

$$P_x = 79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798.$$

(The y-coordinate P_y can be "lifted" from this; see the Sage code below.)

Where is this coming from? This is one of several vetted choices of elliptic curves compiled by the Standards for Efficient Cryptography Group (SECG), an industry consortium, in SEC 2: http://www.secg.org/

The particular curve above is secp256k1 in that document. While the equation of the curve and the prime p are clearly chosen to be "nice" (and so that the curve has nice properties; for instance its order is again a prime q; consequently, all regular points have order q themselves and, thus, generate all other points), it is much more mysterious how the point P was chosen:

https://crypto.stackexchange.com/questions/60420/

On the other hand, the choice of point is believed to not make much of a difference; in particular, it is not hard to see that the discrete logarithm problem is equally difficulty for all points.

Here is how to compute with that elliptic curve in Sage:

```
>>> p = 2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1
>>> E = EllipticCurve(GF(p), [0,7])
>>> P = E.lift_x(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798)
>>> P
(55066263022277343669578718895168534326250603453777594175500187360389116729240: 3267051\
00207588169780830851305070431844712733806592432759389043357573374824 24: 1)
>>> E.order()
115792089237316195423570985008687907852837564279074904382605163141518161494337
>>> is_prime(E.order())
1
>>> P.order()
115792089237316195423570985008687907852837564279074904382605163141518161494337
>>> 100*P
(10730358229073309792484219397246502205314821177519437367153951831350019463975 2: 103795\
96610878271744680668402374216846236544927263979079559154460683600744663 8: 1)
```

Example 221. A few years ago, more than 90% of webservers used one specific, NIST specified, elliptic curve referred to as 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 previous NIST standard (2006–2014).

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

Example 222. A popular alternative is the curve Curve25519. In addition to some desirable theoretical advantages, its parameters are small ("nothing-up-my-sleeve numbers") 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