1. Material covered

(sketch)

These notes are not meant as course notes and are not carefully written. They serve mainly as a summary and/or reminder for what we have done in class.

On November 21, we did the following.

- We briefly talked about m-division polynomials and the fact that if E is an elliptic curve over a finite field $\mathbb{F}_q$, then there are positive integers m and n such that $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}/m \times \mathbb{Z}/mn$.
- We stated, without proof, the Hasse-Weil bound, which states that for every elliptic curve E over a finite field $\mathbb{F}_q$, we have
$$|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}.$$
- We then described the Baby-Step Giant-Step algorithm to count the number of points on an elliptic curve E over a finite field $\mathbb{F}_q$. This involves picking a random point $P \in E(\mathbb{F}_q)$. If E is given by $y^2 = f(x)$, then this can be done by choosing random x-coordinates until you find an $x_1 \in \mathbb{F}_q$ for which $f(x_1)$ is a square; then actually finding a square root y_1 of $f(x_1)$ can be done with the algorithm of Tonelli-Shanks. This algorithm requires an element that is not a square, which can be found fast probabilistically. The Baby-Step Giant-Step algorithm is described in paragraph 2 of this paper `www.mat.uniroma2.it/~schoof/ctg.pdf` by René Schoof. A generalization of Tonelli-Shanks is described in Algorithm 3.3 of the thesis `www.opt.math.tugraz.at/~cvdwoest/maths/dissertatie.pdf` of Christiaan van de Woestijne.
- We continued with Pollard's $p-1$ method to factorize integers, which has nothing to do with elliptic curves, except that Lenstra's method is based on this idea. See Silverman-Tate, section IV.4.
- We also described Lenstra's algorithm to factor integers using elliptic curves, which is inspired by Pollard's $p-1$ method in the sense that it replaces the group $\mathbb{F}_p^*$ for some prime divisor p of n by the group $E(\mathbb{F}_p)$ for some elliptic curve. The benefit is that we have the choice of many elliptic curves. See for this method also Silverman-Tate, section IV.4.

  Given an integer n, we choose a bound B and set $K = \mathrm{lcm}(1, 2, \ldots, B)$. We are in good shape if the order of the group G in question ($\mathbb{F}_q^*$ for Pollard's $p-1$ method and $E(\mathbb{F}_q)$ for Lenstra's elliptic curves method) is B-smooth, which means that all prime divisors of $\#G$ are at most B. Then for any given point $P \in E(\mathbb{F}_q)$, we have a good chance that the order of P divides K, so that $KP = 0$. Note, however, that we do not yet know p. So take instead an elliptic curve E over $\mathbb{Z}/n$ with a point P. This reduces to an elliptic curve E' over $\mathbb{F}_p$ and if $\#E'(\mathbb{F}_p)$ is B-smooth, then we're in business; we may then hope that for the reduction $P' \in E'(\mathbb{F}_p)$ we have $KP' = 0$. This would mean that if we try to compute KP over $\mathbb{Z}/n\mathbb{Z}$, then we have to invert an element that is 0 modulo p, which is impossible in $\mathbb{Z}/n\mathbb{Z}$. Again, we do not actually know p, but we can detect the fact that inverting an element $a \in \mathbb{Z}/n\mathbb{Z}$ is impossible. When this happens, it is because $\gcd(a, n)$ is not equal to 1, so by computing $\gcd(a, n)$, we obtain a nontrivial factor (namely p) of n. In Silverman-Tate, section IV.4, you can find an example.

  There is a theorem of Canfield, Erdős, and Pomerance that states that
$$\#\{x \in \{1, \ldots, T\} \ : \ x \text{ is } T^{1/u}\text{-smooth}\} \sim \frac{T}{u^u},$$
  if $T \to \infty$ and $u \to \infty$ subject to $T^{1/u} > (\log T)^{1+\epsilon}$. This means that a random number x around T has "probability" $1/u^u$ to be $T^{1/u}$-smooth.

  We conclude that if n contains a factor p (which we do not yet know), then $\#E'(\mathbb{F}_p)$ will be approximately $p+1$, so it is B-smooth for $B = p^{1/u}$ with probability around $1/u^u$, and we have to try around u^u elliptic curves E over $\mathbb{Z}/n\mathbb{Z}$ before we find one where $E'(\mathbb{F}_p)$ is $p^{1/u}$-smooth. For each of these curves, we compute KP for some point P on E over

$\mathbb{Z}/n$, which takes $\log K \sim B = p^{1/u}$ steps. In total this gives around

$$u^u \cdot p^{1/u}$$

steps for the algorithm. Optimizing this for u gives

$$u \sim \left(\frac{2\log p}{\log\log p}\right)^{1/2}$$

and

$$B \sim e^{\sqrt{\frac{1}{2}\log p \log\log p}}$$

and total number of steps around a constant times

$$e^{\sqrt{2\log p \log\log p}},$$

which is sub-exponential in $\log p$. These are the numbers that optimize finding the factor p of an integer n. Note again that we do not yet know p, so we can not choose our parameter B based on this. Instead, we increment B during the algorithm, as long as we have not yet found a prime factor of n.

In practice, this method is significantly improved with a so-called *second stage*. An interesting website to play with, that uses elliptic curve factorization, is

`http://www.alpertron.com.ar/ECM.HTM`.

2. Homework

Implement one of the two algorithms using elliptic curves mentioned above. I.e., do one of the two following:

- Write a function in Sage that takes as input two elements a and b of a finite field $\mathbb{F}_q$, with $4a^3 + 27b^2 \neq 0$ and $\operatorname{char}\mathbb{F}_q \neq 2$ and returns the order of the group $E(\mathbb{F}_q)$, where E is the elliptic curve given by $y^2 = x^3 + ax + b$.
- Write a function in Sage that takes a composite integer n and returns a prime factor p of n. It is fine if your algorithm returns an error saying that it found an element a that it could not invert modulo n, because in that case taking the greatest common divisor of n and a gives a nontrivial factor of n on which we could apply the algorithm again to find a prime divisor; of course it is nicer to avoid the error messages, though.

Of course, the better and faster your algorithm works, the better your grade.

You may work in pairs again, but beware: anybody could at any time be asked to clarify how and why the algorithm he or she wrote works.

Finally, of course both point counting and factorization already exist in Sage and you can not make use of that. You can use it to verify whether your algorithm works though! Instead of giving an explicit list of functions that you are not allowed to use, let's just say that you can not use any functions that you yourself consider cheating...

Have fun!