22 June Lab Stuff

1 Miller Rabin

Let us recall how the Miller-Rabin primality test works:

The idea is that if n is prime, then the only numbers x such that $x^2 \equiv_n 1$ are 1 and -1. Therefore, if we can find an x that is not 1 or -1 where $x^2 \equiv_n 1$ then n is not prime.

We also know by Fermat's little theorem that if n is prime then $x^{n-1} \equiv_n 1$ for all x. If we write $n = 2^s \cdot d$ where d is odd, we can choose a random x and consdier the following sequence of values:

$$x^d, x^{2d}, x^{2^2 d}, x^{2^3 d}, ..., x^{2^{s-1} d}, x^{n-1}$$

Notice that each term in this sequence is the square of the previous term; e.g. $x^{2^3 d} = (x^{2^2 d})^2$. If n is prime, then the last number in this sequence is 1. Since the only square roots of 1 are 1 and -1, it must be that the second to last term in this sequence is 1 or -1. If the second to last term is 1, then the third to last term must be 1 or -1, and so on all the way down.

If we pick a random x and $x^d \equiv_n 1$ or $x^d \equiv_n -1$, then n might be prime. If x^d is not one of these numbers, then we start squaring. If n is prime then we have to hit 1 eventually, but since the only numbers that square to 1 are 1 and -1 we will always hit -1 first. Turning this around, we get the following way to identify a number as composite:

> If for some x we have x^d not equivalent to 1 or -1, and upon repeating squarings we see 1 before -1, then n is not prime.

This gives us the following algorithm to determine if a number n is composite:

- Write $n - 1 = 2^s \cdot d$ where d is odd.
- Choose some x.
- Compute x^d mod n. If this is 1 or -1, stop; n might be prime.
- Compute x^{2d} mod n. If this is 1, stop; n is definitely not prime. If this is -1, stop; n might be prime. Else, keep going.
- Compute $x^{2^2 d}$ mod n. If this is 1, stop; n is definitely not prime. If this is -1 stop; n might be prime. Else, keep going.
- Compute $x^{2^3 d}$ mod n. If this is 1, stop; n is definitely not prime. If this is -1 stop; n might be prime. Else, keep going.

  $\vdots$

- Compute $x^{2^{s-1} d}$ mod n. If this is 1, stop; n is definitely not prime. If this is -1 stop; n might be prime. Else, n is definitely not prime.

It is important to keep in mind that this test can tell you that n is definitely not prime but it cannot tell you that n is definitely prime. Consequently, it is important to check several values for x; some of them may say that n might be prime while others may tell you n is definitely not prime. If you check many values of x and all of them say that n might be prime, then n is probably prime.

- Write a function `RMwitness(n,x)` that uses x to test if n is prime.
- Write a function `checkprime(n)` that uses `RMwitness(n,x)` to check if a number is prime. The idea is for `checkprime(n)` to run `RMwitness(n,x)` for many values of x. You can choose random values x using the command `ZZ.random_element(a,b)`; this returns a random integer between a and b. To make this even better you can also incorporate trial divison by small primes (say less than 100) and/or the $p - 1$ factor test you wrote yesterday.

- Write a function `findprime(a,b)` that tries to find a prime number between a and b.

When you are done go to `www.summermathprogram2016.blogspot.com`. There are posts containing tests for your ElGamal and RSA functions.