Math 189H Joy of Numbers Activity Log

Tuesday, November 29, 2011

Ralph Abraham: "My specific goal is to revolutionize the future of the species. Mathematics is just another way of predicting the future."

Franklin D. Roosevelt: "If you treat people right they will treat you right - ninety percent of the time."

Perrin's function: $P(0 = 3, P(1) = 0, P(2) = 2$, and $P(n) = P(n-2) + P(n-3)$. If n is prime than $n|P(n)$. The opposite statement is false; there are composite n for which $n|P(n)$. But the first instance of this failure is for $n = 271,411$. $P(271,411)$ has 33,150 digits.

Today we started with some questions related to our (last!) exam, and worked some examples similar to some of the questions on the exam; an example of using brute force factoring and the Euclidean algorithm to recover someone's private decryption exponent from their public modulus and encryption exponent, and two examples of computing the order of a number a modulo n, by computing $\phi(n)$ and and a^k mod n for every number dividing $\phi(n)$ (starting from the smallest).

Then we introduced another public key cryptosystem that is based on modular exponentiation. This one works a little differently; by passing information to one another in public, Alice and Bob can 'agree' on a secret key that they can use for further communications. The process is called the *Diffie-Hellman Key Exchange* system [although, again, Wikipedia reports that the basic idea had been discovered a few years before it was publicly known (in 1976) by a British analyst, Malcolm Williamson]. The basic setup is:

Alice and Bob agree on a choice of (large!) prime p and a number a relatively prime to p. [With what we will do with it, it may as well be less than p, too.] Alice and Bob each, privately, choose numbers k_A and k_B to use as exponents. They then each compute a^k mod p for their chosen exponent, creating numbers $n_A \underset{p}{\equiv} a^{k_A}$ and $n_B \underset{p}{\equiv} a^{k_B}$. They then (publicly!) report the numbers n_A, n_B to one another.

At this point Alice now knows n_B and her own k_A, and Bob knows n_A and his own k_B. If effect, Alice knows a^{k_B} (mod p) <u>without</u> knowing k_B, and Bob knows a^{k_A} (mod p), without knowing k_A (!). This allows Alice and Bob to compute the same number (which will be their secret):

$$n_A^{k_B} \underset{p}{\equiv} (a^{k_A})^{k_B} = a^{k_A k_B} \underset{p}{\equiv} z \underset{p}{\equiv} a^{k_B k_A} = (a^{k_B})^{k_A} \underset{p}{\equiv} n_B^{k_A}$$

Bob can compute the number on the left, and Alice can compute the one on the right; and they are both equal to one another, mod p. They can then use the number z (or rather, to hide all of the computations, its remainder on division by p) as a 'multiplier' to encypt messages: to encrypt the message m, what they <u>send</u> is mz mod p. To decrypt the message, the recipient needs to 'divide by z', that is, multiply by the inverse of z (mod p). This they can do, because they know z and p; the Euclidean Algorithm will allow them to find x and y with $zx + py = 1$, giving us the inverse, x.

The key (no pun intended) to the security of this cryptosystem is that in order to compute z, Alice used her private exponent k_A, and Bob used his private exponent k_B. On the other hand, the only information that an eavesdropper is in possession of is p, a, n_A, and n_B. This means that Eve knows both a^{k_A} and a^{k_B} (mod p) without knowing either k_A or k_B. The question is, is this enough information for Eve to construct the shared secret $z \underset{p}{\equiv} a^{k_A k_B}$? This is known as the *Diffie-Hellman Problem*:

Knowing a, a^k, and a^ℓ (all mod p), what is the fastest way to compute $a^{k\ell}$ (mod p) ?

The prevailing conclusion among researchers working on this appears to be that the fastest way is to recover either k or ℓ (so that, basically, you can do exactly what Alice and Bob do to compute $a^{k\ell}$). Focusing on one of them for concreteness, recovering k from knowledge of a and a^k we recognized as something we could do for 'ordinary' numbers; it can be done by taking a logarithm: $k = \log_a(a^k)$. But these aren't ordinary numbers! We are working modulo a prime, p. And fun things happen when you raise a to powers, mod p; it can be difficult (we think!) to determine what exponent got you to the result, because of the 'wrap-around' effect of taking remainders modulo p. For example (to recycle some of the computations we did at the start of class!) we saw that the order of (hm, what were the numbers? These numbers won't be what we used...) 2 modulo 13 is 12, so 12 is the smallest positive k so that $2^k \equiv 1$ mod 13. This actually means (using the pigeonhole principle) that every number from 1 to 12 is a power of 2, mod 13, so for example $7 = 2^k$ mod 13 for some k. Finding that k, other than by brute force (there <u>are</u> only 12 exponents to try...), requires some very deep thought. Or you can try $5^k \equiv 6$ mod 97 for some k; which one? The point really is that for our cryptosystem we are going to be using big numbers for our prime p and hidden exponents k_A, k_B, to make a brute force attack (to recover k_A from a and a^{k_A}; try all possible exponents until you get something congruent to a^{k_A} (!)) impractical.

The problem of recovering k from a and a^k modulo p is known as the *discrete logarithm problem*. As it turns out, the best currently known methods for recovering k run in roughly the same time as the best known factoring algorithm (GNFS); so Diffie-Hellman currently has the same level of security as RSA, using moduli of the same size. Diffie-Hellman is in fact used more often for protecting your communications and transactions than RSA is, partly because it is so easy to modify on the fly. Once two people agree on a prime modulus p and a base a, they can change their shared secret by simply choosing different exponents and exchanging the computations a^{k_A}, a^{k_B} with one another; building their new secret multiplier from this is a matter of fast exponentiation, and then the Euclidean Algorithm (which is also fast...) gives the decryption multiplier. In this way they could protect every communication using a different shared key; anybody capable of recovering one of their k_A from a^{k_A} (after a few thousand years? or by tying up a few thousand computers for a year?) will be able to read <u>one</u> message...