Math 189H Joy of Numbers Activity Log

Thursday, December 1, 2011

Voltaire: "There is an astonishing imagination, even in the science of mathematics... We repeat, there was far more imagination in the head of Archimedes than in that of Homer."

David Hilbert: "Physics is much too hard for physicists."

[Addendum to the previous class: during our discussion of the order of a number a mod n, we often discovered numbers a whose order was $\phi(n)$ (rather than a proper factor of $\phi(n)$). Such numbers are usually called *primitive roots of unity* modulo n (because they are solutions to the equation $x^{\phi(n)} \underset{n}{\equiv} 1$ that are 'primitive' they aren't solutions to $x^k \underset{n}{\equiv} 1$ for any smaller exponents k). Not every integer n has a primitive root; for example, $\phi(8) = 4$, but as we have seen from our tables, every number relatively prime to 8 has order 2. If n has a primitive root, then it usually has many of them; in fact, it has precisely $\phi(\phi(n))$ of them!]

Class started with Hannah, Relyn, and Jacob telling us about the mathematics of juggling. If one starts with the idea that exactly one ball should be thrown at each tick of the clock, then a juggling pattern can be described by a sequence of numbers, called a *siteswap*, $a_1, \dots, a_n$, describing how many clock ticks each ball will be in the air (the sequence then repeats itself (in principle) forever). The sequence is actually jugglable (i.e., realized with actual balls) precisely when the sequence $a_1 + 1, a_2 + 2, \dots a_n + n$ gives all n remainders on division by n, <u>and</u> the average of the a_i's (which represents the number of balls being juggled) is a whole number. The first condition really represents the condition that only one ball is returning to the hand at each clock tick; each number <u>is</u> the clock tick that it (first) returns to the hand.

After the presentation, we started with a odd litle number fact: if you start with a number $n = a_0 + 10a_1 + \cdots + 10^k a_k$ [i.e., its digits from right to left are $a_0, \dots, a_k$] and you replace n with the sum of the cubes of its digits $a_0^3 + \cdots a_k^3$, then I stated that you eventually always end up at $153 = 1^3 + 5^3 + 3^3$. This is, of course, false, however: starting with 100000, you go to 1, where you always stay. The correct statement is a bit more involved, although the basic idea is still that if you start with a number having 5 or more digits, the result will always have fewer digits than you started with (because the cube of each digit can be no more than $729 = 9^3$, and then you add them). So to see what eventually happens to a number, you just need to check what happens to the numbers from 1 to 9999, which Maple 15 will happily do for you in a minute or so. The results are that every number, when we repeat this cubing process, eventually falls into one of nine patterns:

$1 \to 1 \to 1 \to \dots$
$55 \to 250 \to 133 \to 55 \to \dots$
$136 \to 244 \to 136 \to \dots$
$153 \to 153 \to \dots$
$160 \to 217 \to 352 \to 160 \to \dots$
$370 \to 370 \to \dots$

$371 \to 371 \to \dots$
$407 \to 407 \to \dots$
$919 \to 1459 \to 919 \to \dots$

We then talked some more about the Diffie-Hellman key exchange system, and in particular about the some of the design considerations in choosing the numbers involved. In Diffie-Hellman (DH), there is a public modulus p (a prime) and a base a (relatively prime to p). Alice and Bob pick exponents k_A and k_B (kept secret) and exchange a^{k_A} mod p and a^{k_B} mod p. Then both can compute the shared secret $z = a^{k_a k_B}$ mod p.

p is chosen to be prime because, well, then any a between 2 and $p-1$ will be relatively prime to p. There are further considerations to be made, though, which stem from what we want from a. In the end, we noted, Eve the eavesdropper doesn't really need to discover k_A in order to learn the secret (by computing z the way Bob does), what Eve really needs is an exponent k, that she knows, so that $a^k \underset{p}{\equiv} a^{k_A}$, since then

$$z \underset{p}{\equiv} a^{k_A k_B} = (a^{k_A})^{k_B} \underset{p}{\equiv} (a^k)^{k_B} = a^{k k_B} \underset{p}{\equiv} (a^{k_B})^k,$$

which Eve knows how to compute because she has intercepted a^{k_B} mod p from Bob, and has figured out k.

And the point is that Eve can find a k like this <u>if</u> the order of a modulo p is 'too small'. Because, as we have seen before, if $a^r \underset{p}{\equiv} 1$, then if $k_A = rq + s$, then $a^{k_A} \underset{p}{\equiv} a^s$. So a^{k_A} is the same, mod p, as a raised to a power between 0 and $r-1$. So if r is too small (on the order of billions? trillions?) we could by brute force check each exponent k from 1 to r, comparing a^k mod p to a^{k_A}, to find an exponent that works.

Since we know that the order of a mod p divides $\phi(p) = p-1$, the problem becomes one of choosing an a so that this order is a <u>large</u> factor of $p-1$. Which can be kind of tricky to guarantee!, since our approach to finding the order of a, as you have seen on your last exam, is to factor $p-1$, to find all of its divisors, and test each one! But if p is large (for security), so is $p-1$, and so factoring $p-1$ is not really an option! But, and here is where the primality of p is really useful, the equation $x^k \underset{p}{\equiv} 1$ has at most (and, in fact, it turns out, exactly) k solutions modulo p, for every divisor of $p-1$. So if we instead ensure that $p-1$ does not have many <u>small</u> divisors (we cannot, for example, avoid the divisor 2; p is a big prime, so it is odd, so $p-1$ is even!), then it follows that there aren't many a whose order will be small, and so we can confidently pick a 'random' a between 2 and $p-1$ and expect that nobody can by brute force find an exponent that will work for them as a stand-in for our secret exponent k_A or k_B.

So not all primes p are created equal for Diffie-Hellman; it pays to find one so that $p-1$ has very few factors below, say, a trillion. Which can be tested with fair certainty by (trial division and) the current best factoring procedures (remember GNFS?); their running time is really determined by the size of the smallest factor. So as you find a factor you start again with the quotient by the factor you found, and when the process eventually starts taking a long time, you can be pretty certain that you have found all of the small factors...