Math 189H Joy of Numbers Activity Log

Thursday, November 17, 2011

Douglas Adams: "I love deadlines. I like the whooshing sound they make as they fly by."

Henri Lebesgue: "In my opinion, a mathematician, in so far as he is a mathematician, need not preoccupy himself with philosophy – an opinion, moreover, which has been expressed by many philosophers."

1093 is prime, so $1093|2^{1092}-1$. In fact, $1093^2|2^{1092}-1$. The only other number under $6,000,000,000$ for which this is known to be true is 3511.

Last time we introduced the rudiments of what is known as the RSA cryptosystem: if we know that $k\phi(n)+1=ed$ and $\gcd(a,n)=1$, then if we write $b=a^e \bmod n$, then $a\equiv b^d \bmod n$. In other words (if $0\le a<n$) if we 'encrypt' a as $b=$ the remainder of a^e on division by n, then the encrypted number b can be 'decrypted' by taking the remainder of b^d on division by n, recovering a. Together with some 'standard' method to turn the sort of message we want to send into a number a (the same system can be used to recover the message from a), gives us a complete cryptosystem.

In practice the way this works is that we have a table for turning symbols into numbers, 'A' $\leftrightarrow$ 11, 'B' $\leftrightarrow$ 12, etc., using numbers of a fixed length, and we string the numbers together to create a single large number a. [If a ends up too long, we cut it into smaller pieces and separately encrypt and send each piece.] The design considerations here are that we need to recover the message from the strung-together number, so we need to use a uniform length (11 to 99 for 89 or fewer symbols, 111 to 999 if you want to encode the entire ASCII character table?) so that we know how to cut the number back up, into single-character length numbers, in order to use the table (backwards) to reconstruct the message.

The encryption machine consists, essentially, of the numbers e and n, and the decryption machine consists of d and n. The way we originally described it d and e were found by factoring $k\phi(n)+1$ for some k. But if we look at $de=k\phi(n)+1$ sideways, it reads $de-k\phi(n)=1$, which reminded us of calculations we've done before, for gcd's. In fact, being able to write this amounts to saying that e and $\phi(n)$ are relatively prime; the Euclidean algorithm (which we decided from now on to abbreviate 'EA' [even if that does also stand for 'Electronic Arts']) would then allow us to quickly compute the corresponding d (and k, which, really, we don't care about).

What is really need, then, to create the cryptosystem is an n, for which we know how to compute $\phi(n)$, and a more or less randomly chosen e relatively prime to $\phi(n)$. (We can then fairly quickly compute the corresponding d.) An attacker, in posession of e and n, wants to be able to take an encrypted message $b=a^e \bmod n$, and recover a, meaning, we decided, they want to be able to take e-th roots. Which in principle didn't sound too bad, until we realized these were e-th roots $\underline{\text{modulo}}$ $\underline{n}$, which complicates things a bit? As an example, telling you that $a^7\equiv 2 \bmod 237$, and challenging you to find a, is asking you for the 7-th root of 2 mod 237. [It happens to be $a=92$.] But we just saw that raising number to the d-th power effectively <u>is</u> taking e-th roots; $(a^e)^d \equiv a \bmod n$. [This was a very roundabout way of discovering, essentially, that the attacker's way of recovering a from $a^e \bmod n$ is really the same as the receiver's way of decrypting: figure out what d is!] So the security of the RSA cryptosystem rests on the answer to: "How hard is it to determine d (the secret exponent) from the provided information, n (the modulus) and e (the 'public' exponent)?".

To which the answer is, it depends on how n was chosen! The person building the cryptosystem with values n, e, d faces their own problem: they need to know how to compute $\phi(n)$, in order to pick an e relatively prime to it and use EA to compute the 'mutiplicative inverse' d of e modulo $\phi(n)$. We have seen (or at least have been told) that determining $\phi(n)$ is quite quick, <u>if</u> we have a complete factorization of n.

But! If our attacker can figure out what $\phi(n)$ is, knowing n, we are in big trouble, because they can then just as quickly compute d from e as we could. So our first instinct in picking numbers n whose value for $\phi(n)$ we knew, namely <u>prime</u> numbers n [where we know that $\phi(n) = n - 1$], will not be very secure! If our attacker for a moment suspects that we are using a prime modulus (or of they simply compute $a^{n-1} \bmod n$ for some small values of a and keep getting 1, which is a good indication of primality), they would then just <u>try</u> the value of d they get by using EA to compute the inverse of $a \bmod n - 1$, and see if that d successfully decrypts (i.e., returns something other than garbage) some of your messages. Which, of course, we know it will (since we were using a prime modulus!), and we are sunk.

So we need to try something better, and being good lazy mathematicians, we next try a modulus n which is the product of two primes, $n = pq$. Then $\phi(n) = (p - 1)(q - 1)$ [unless $p = q$; then is it $(p - 1)p^1$. But a good attacker will test our modulus to see if it is a perfect square, too...], and we can then proceed to find values of e relatively prime to $\phi(n)$ and compute our secret decryption exponent d. The question is, though, can our attacker (who, remember, is in possession of n and e, because we have told the entire world this information, so that anybody can send us a message), if they guess that we are now using a product of two primes for our modulus, recover d from what we have revealed? It would seem (and everybody researching this subject seems to agree) that the attacker must determine $\phi(n)$ (so that they can then quickly use EA to compute d from e). But knowing that $n = pq$ for two primes p, q, how hard is it to find $\phi(n) = (p - 1)(q - 1) = pq - p - q + 1 = n - p - q + 1$? One way to think about this is to ask: if we know both $n = pq$ and $\phi(n)$, what to we also automatically know? Since $\phi(n) = n - p - q + 1$ and we know n, we know $-p - q + 1 = -(p + q - 1)$, so we know $p + q - 1$, and so we know $p + q = m$.

So, if we suspect that you are using an n that is a product of two primes, we then (if we have succeeded in determining $\phi(n)$) know both the product and the sum of those two primes. But, as we were able to see, knowing what both the product and the sum of two numbers are allows us to determine what the two numbers are! We could see this graphically; the graphs of the two functions $xy = n$ and $x + y = m$ [that is, $y = n/x$ and $y = m - x$] will meet at two points. Those points are (p, q) and (q, p) ! Of course, for the purposes of a cryptosystem, p and q are going to be large numbers! For 512-bit RSA (meaning it uses numbers p and q comparable to $2^{512} \approx (2^{10})^{50} \approx (10^3)^{50} = 10^{150}$), p and

q have about 150 digits each, so m is a 150-digit number and n is a 300-digit number! Drawing those two graphs, then, will be 'fun'...

But thinking instead of $xy = m$ and $x + y = m$ as two equations in two unknowns, our prior experience with solving such things suggested solving one equation for one variable in terms of the other, and plugging that into the other equation. Solving $x + y = m$ as $y = m - x$ and plugging in, gives $x(m - x) = n$, or $x^2 - mx + n = 0$. This is just a quadratic equation; we can solve those! This gives $x = (1/2)(m \pm \sqrt{m^2 - 4n})$, which, sure, involves a square root of a 300-digit number, but it <u>is</u> a perfect square! [it secretly is $(p + q)^2 - 4pq = (p - q)^2$], so that won't be too tough. The two solutions we get with our $\pm$ are, in fact, p and q; what we are really computing is $x = (1/2)((p + q) \pm |p - q|)$.

The upshot of this, really, is that knowing $n = pq$ and $\phi(n) = (p - 1)(q - 1)$ allows us to quickly recover both p and q, i.e., to factor n. But if we know how to factor $n = pq$, we can quickly compute $\phi(n)$ (!). So knowing $\phi(n)$ and knowing how to factor n really amount to the same thing (at least in the case of a product of two primes). So the security of RSA rests on the difficulty of factoring (large) numbers. Which, we have repeatedly suggested to ourselves, is really difficult! We've found that <u>knowing</u> that a number is not prime isn't tough; but actually finding factors has proved elusive. [You can think of this as taunting your attacker? 'Yes, you know that my modulus isn't prime, I'll even tell you it has two prime factors, but, Thbbt!, it will do you no good..."]

But we should see all of this in practice! Next time. Then we will look at how to turn this around; instead of people sending you messages that only you can read, RSA can be used to let you send messages to other people, that they can know <u>only</u> <u>you</u> could have sent! [In fact, we can do both: someone can send you a message that only you can read, and only they could have sent! The implications for online commerce are immediate; the message iteself verifies that you are authorized to carry out the transaction (while nobody else knows what that transaction <u>was</u>)!]