Codes, Ciphers & Secret Messages

September 2010

We first need to talk a little about *modular arithmetic* (also known as *clock arithmetic*): one fixes a positive integer m (the *modulus*, for real clocks $m = 12$) and considers two integers a and b equal modulo m if they differ by a multiple of m; in other words, a and b have the same remainder when divided by m. In this case, we write $a \equiv b \bmod m$. Here are three examples:

- $m = 12$: We have $2 \equiv 14 \equiv 26 \equiv -10 \bmod 12$. (Think about hours of the day and how they would appear on a clock.)
- $m = 2$: Two integers are equal modulo 2 precisely if they have the same parity.
- $m = 10$: Two integers are equal modulo 10 precisely if they have the same last digit.

We'll concentrate on the last example, so from now on we'll do arithmetic modulo 10. Our goal is to come up with a *code* modulo 10; that is, we want to send a message consisting of digits, and we'd like to encode it in such a way that only our friends can decode it. In the following exercises we'll develop different codes, improving them as we move along.

(1) Become familiar with addition, subtraction, and multiplication modulo 10. Compute some examples, make up a multiplication table, etc.

(2) The first code we'll discuss goes back to Julius Cesar's times (or so the story goes). You agree with your friend on a modulus (say, 10) and a shift parameter (say, 3). The encoding takes a digit d and moves it to $d + 3 \bmod 10$. So let's call the encoded digit e, then

$$e = d + 3 \bmod 10 \qquad \text{which is equivalent to} \qquad d = e - 3 \bmod 10\,.$$

So your friend will take the encoded digit e, subtract 3 from it (modulo 10), and thus retrieve your original digit d. Think about why this way of encryption is not particularly safe.

(3) We don't have division modulo 10 in the usual sense; however, some numbers do have *multiplicative inverses* modulo 10. What we mean is the following: if $a \cdot b \equiv 1 \bmod 10$ then we say that a and b are multiplicative inverses (modulo 10) of each other. Try out some numbers and see which ones have multiplicative inverses modulo 10. Based on this data, come up with a conjecture which numbers have multiplicative inverses modulo m and which don't (where m is arbitrary). Prove your conjecture (one way to do that uses the *Euclidean Algorithm*).

(4) Here is our second code. Once more you agree with your friend on a modulus (say, 10) and a parameter (say, 3). The encoding takes a digit d and multiplies it by 3 modulo 10:

$$e = 3 \cdot d \bmod 10\,.$$

Come up with a decoding scheme. Why does it work? What happens if we replace the parameter 3 by 2? Or 6? Which parameters other than 3 could we have used? Come up with a good argument why this encryption scheme is safer than our first one. Also come up with an argument why this second scheme is still not particularly safe.

(5) Take the numbers $0, 1, 2, \ldots, 9$ and multiply each of them by 3. What happens to this list of numbers? What happens if instead we multiply the list by 2? By 6? Think about how this relates to the previous exercise. Repeat this process in the general case, where we look at the numbers $0, 1, 2, \ldots, m-1$ modulo m.

(6) Show that, if a has no common factor with 10 other than ± 1 (we say that a and 10 are *relatively prime*), then $a^4 \equiv 1 \bmod 10$. Can you see where the exponent 4 comes from? Come up with a similar equation for a general modulus m and think about how you could prove that equation.

(7) The next code we'll describe is due to Diffie and Hellman. It is a *public-key code* because part of the code is known to everyone. Here's how it works: you and your friend choose a prime number p and an integer g between 2 and $p-1$. Both of these numbers are public (so, e.g., you two can safely discuss these numbers on the phone or over email—if someone wiretaps you, no problem). Now you *secretely* choose an integer m, and your friend *secretely* chooses an integer n. You compute $g^m \bmod p$ and tell your friend the result. Your friend computs $g^n \bmod p$ and tells you the result. The secret key that you both can use is

$$s \equiv g^{mn} \equiv (g^m)^n \equiv (g^n)^m \bmod p\,.$$

The last two equalities explain why both you and your friend can easily compute s. You can now use s to encode messages, e.g., using multiplication mod p, and s^{-1} to decode. Can you see why it's hard to compute s if you know p, g, g^m, and g^n? How could you make this cryptosystem safer? Do you see a way to "break" it?

(8) Now we'll define *Euler's ϕ-function*: $\phi(n)$ counts the numbers between 1 and n that are relatively prime to n. Compute the first couple of values of this function: $\phi(1), \phi(2), \phi(3), \ldots$ Find a formula for $\phi(n)$ when n is prime. Find a formula for $\phi(mn)$ in terms of $\phi(m)$ and $\phi(n)$ in the case that m and n are relatively prime. One of Euler's theorem says that, if a is relatively prime to n, then $a^{\phi(n)} \equiv 1 \bmod n$. Conclude that, if $b \cdot c \equiv 1 \bmod \phi(n)$, then $\left(a^b\right)^c \equiv a \bmod n$.

(9) The previous exercise allows us to introduce the *RSA cryptosystem*.[1] Here's how it works: You need two prime numbers p and q, compute their product $m = pq$, find a number b that is relatively prime to $\phi(m) = (p-1)(q-1)$, and compute an inverse c of b modulo $\phi(m)$, i.e., $bc \equiv 1 \bmod \phi(m)$. You keep all of this private except for the numbers m and b which you make public (in particular, your friends know m and b). To send you a message d, your friend encodes it as

$$e = d^b \bmod m\,.$$

You can decode your friend's message by computing

$$d = e^c \bmod m\,.$$

Explain why this decoding works. What makes this cryptosystem safe? How could you make it safer? What would one need to break it?

Matthias Beck math.sfsu.edu/beck

[1] The RSA cryptosystem is named after its discoverers Ron Rivest, Adi Shamir, and Leonard Adleman.