Public Key Cryptography and Clock Arithmetic Warm-up

Lecture notes for Access 2009 by Erin Chamberlain and Nick Korevaar.

We've discussed Caesar Shifts and other substitution ciphers, and we've seen how easy it can be to break these ciphers by using frequency analysis. If Mary Queen of Scots had known this, perhaps she would not have been executed.

It was a long time from Mary Queen of Scots and substitution ciphers until the end of the 1900's. Cryptography underwent the evolutionary and revolutionary changes which Simon Singh chronicles in *The Code Book*. If you are so inclined and have appropriate leisure time, you might enjoy reading Chapters 2-5, to learn some of these historical cryptography highlights: People came up with more complicated substitution ciphers, for example the Vigenère square. This Great Cipher of France baffled people for quite a while but a determined cryptographer Etienne Bazeries had a Eureka moment after three years of work, cracked the code, and possibly found the true identity of the Man in the Iron Mask, one of the great mysteries of the seventeeth century. Edgar Allan Poe and Sir Arthur Conan Doyle even dabbled in cryptanalysis. Secrecy was still a problem though because the key needed to be sent, and with the key anyone could encrypt and decrypt the messages. Frequency analysis was used to break all of these codes.

In 1918 Scherbius invented his Enigma machine, but Alan Turing's machine (the first computer?) helped in figuring out that supposedly unbreakable code, and hastened the end of the second World War. During this same war Americans used code talkers, but there were many cases of friendly fire in which the Native American code talker was killed.

As computers became more commonplace, code creators adapted. Since computers only deal with strings of 0's and 1's, each letter in a message is replaced by its ASCII binary number, creating long strings of numbers which can be scrambled and otherwise manipulated, sent, and then descrambled and read. By the mid 1970's there were amazingly complicated encryption algorithms which could be made essentially unbreakable. For example, in Chapter 6 (which you will want to read) Singh mentions the Lucifer cipher, a special version of which is known as the Data Encryption Standard, or DES. Horst Feistel's Lucifer encrypts messages according to a scrambling operation. Lucifer can be set up with large enough keys so that it is secure, and a version which is small enough for the U.S. National Security Agency (NSA) to crack became the widely-used DES.

However, up until this point, no matter how convoluted the the encryption methods were, and how frequently the keys were changed for security reasons, all methods required that both parties to the message possessed *the* key for encryption and decryption.... and it was just assumed, because this had always been the case, that if you possessed the method to encrypt a message, then this was equivalent after perhaps a little work, to also knowing how to decrypt it. By the mid 1970's there were thousands of couriers flying all over the world, whose only job was to transfer cipher keys.

As the precursor to the internet, namely the ARPAnet, was beginning to grow, Whitfield Diffie and Martin Hellman, as well as others, realized the huge potential for electronic transactions, together with the need for assured security. Diffie-Hellman were perhaps the first (or were they?) to realize that there was an entirely new way to think of cryptography; that perhaps there were encryption keys which you could let everyone in the world know, but for which you could never the less keep secret the decryption key. This would solve the problem of key distribution, since if you wanted to receive secure messages you could tell the world how to encrypt anything they wanted to send you, but only you would know the decryption key which could stay safely at home. Diffie-Hellman called such encryption keys, "trapdoor", or "one way" functions, because knowing the encryption function did not automatically allow clever people to work out the decryption function. In 1977, Ronald Rivest, Adi Shamir and Leonard Adleman described one of the easiest one-way functions, and the resulting method of public key cryptography is called RSA, in their honor. As we shall see, this method relies on number theory and modular ("clock") arithmetic, and we're going to learn about it!

Here's a great example to get the discussion going, even though the encryption function is NOT a one-way function!

Example 1. The goal is to give a formula for a 4-letter Caesar shift. First we will assign a number value to each letter in the alphabet according to the table below. Then we want to send the message "REPLY" to someone using the Caesar shift. For each letter in the message, replace it with its number value. Put each of those values in our encrypting function $f(x) = x + 4$ but cycle around the alphabet when you need to. For example, $f(X) = f(23) = 23 + 4 = 27$ which is the same as $1 = B$. Find the corresponding letter for the new numbers. What is your encrypted message? What is a formula for the decryption function?

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Remember that Caesar shifts are easy to decrypt, and they have the key distribution problem. Caesar shifts correspond to some sort of "clock" addition, and the decryption function is a corresponding clock subtraction. If we grouped letters (and their corresponding numbers) together into packets, then we could use clock numbers ("moduli") hugely bigger than 26, but it would be just as easy as before to figure out the decryption function from the encryption function. You can also multiply and take powers in clock arithmetic. It will turn out that power functions are effectively one-way functions for clock numbers which are products of huge primes, and this is the basis for RSA cryptography. To get to this realization we're first going to have to get really good at modular addition and multiplication.

Clock Arithmetic: Addition and Multiplication

Example 2. (Military time.) People in the military (and just about every other country) use a 24-hour clock for telling time. So 3:00 in the morning is 3:00, but 3:00 in the afternoon is 15:00. If you go to a military base and are told to be on the practice field at 18:00, what time (on a 12-hour clock) should you be there?

Example 3. If it is 10 am and Josh is picking you up in 7 hours, assuming he is on time, what time will he be there? (In military time and standard time.)

These are simple problems, ones that we have done without thinking it was math, for a long time. But what is really going on here? We add the numbers together, but we don't care about how many revolutions are made, just about what is left over. For small numbers like these, we see that it is easy enough to add and figure out the correct number, but for large numbers can you figure out a fast way to do this?

Example 4. If it is 5:00 and you have to leave for the airport in 39 hours, what time do you need to leave (military time)?

Example 5. If it is 8:00, and you have an appointment in 1984609 hours, what time is your appointment (military time)? Hint: use your calculator.

The fancy math term for this type of arithmetic is **modular arithmetic**. It's "time" to be precise:

Definition 1. Let n be a positive integer, and let a and b be integers. We write

$$a \equiv b \mod n$$

(and we say a **is congruent to** b **mod** n) when a and b differ by a multiple of n (i.e. when $a - b$ is a multiple of n). If $a \equiv b \mod n$ and $0 \leq b < n$ then b is called the **residue** of $a \mod n$. (If a was positive, then b would just be the remainder of a divided by n. You can think of b as the "clock" number corresponding to a.)

Example 6. What is the residue of 1984609, $\mod 24$? (Hint, use your work from the previous example!)

Here are some important properties of modular arithmetic:

Property 1: If a, b, and n are integers, and if $a \equiv b \mod n$, then $b \equiv a \mod n$.

Property 2: If a, b, and n are integers, and if $a \equiv b \mod n$ and $c \equiv d \mod n$, then $a + c \equiv b + d \mod n$.

Property 3: If a, b, and n are integers, and if $a \equiv b \mod n$ and $c \equiv d \mod n$, then the products $ac \equiv bd \mod n$.

These properties just demonstrate that this type of arithmetic behaves like we want it to. Here's why they're true: If $a - b$ is a multiple of n, then $b - a$ is the opposite multiple of n. If $b = a + kn$ and $d = c + ln$ then $b + d = a + c + (k + l)n$ and $bd = (a + kn)(c + ln) = ac + (kc + al + kln)n$.
Let's look at some examples to see why these properties help us with modular arithmetic:

Example 7. We know that $17 \equiv 2 \mod 5$ and $14 \equiv 4 \mod 5$. Find

$$17 + 14 \mod 5,$$

$$(17)(14) \mod 5,$$

$$17^3 \mod 5,$$

using the properties above to save work.

Here are some exercises to complete before tomorrow's class. (Yes, this is fun homework.)

Exercise 1. Find the residue x which solves the equation $x - 8 \equiv 9 \mod 13$.

Exercise 2. List all of the integers x between -50 and 50 which satisfy $x \equiv 7 \mod 17$.

Exercise 3. Fill in the missing residue value: $-3 \equiv \underline{\quad\quad} \mod 11$.

Exercise 4. Find residue values for $21 + 83 \mod 5$, and for $(21)(83) \mod 5$ efficiently.

Exercise 5. Find all of the integers y between 1 and 100 which satisfy $y \equiv 13 \mod 20$.

Exercise 6. Fill in the missing residue numbers:

1. $19 \equiv \underline{\quad\quad} \mod 6$
2. $20568 \equiv \underline{\quad\quad} \mod 19$
3. $-39 \equiv \underline{\quad\quad} \mod 16$

Exercise 7. Solve for x in the equation $3 - x \equiv 7 \mod 8$.

Exercise 8. Find the residue numbers for $7^5 \mod 9$, and for $2^{10} \mod 7$, efficiently.

Exercise 9. Find the missing residue number x, if it exists:

1. $3x \equiv 5 \mod 8$
2. $2x \equiv 5 \mod 8$