Section 3: Cryptography

In 1802, at the ripe old age of 24, Carl Gauss published his epic mathematical work *Disquisitiones Arithmeticae.* It was here that he published many of his most celebrated results, including the Law of Quadratic Reciprocity. In this work, he also stated:

> *The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extant…[that] the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.*

What Gauss was saying is that the problem of finding large prime numbers and factoring large numbers into their prime factors is so important that we should never give up trying to find the perfect solution. The difficulty of this task is the topic of our present section. Many modern-day encryption systems use extremely large numbers as one of their tools. Since they still cannot be factored, even using the fastest computers, the systems remain secure. We will discuss these kinds of encryptions, and the mathematics behind them in this section.

Before we get into the mathematics, let's review some history. The Roman Empire was one of the first civilizations to employ cryptography. Julius Caesar used a basic substitution cipher in which each letter of the alphabet is replaced by the letter that occurs three places down the alphabet, with the last three letters cycled back to the first three letters. If we represent each letter by a two-digit number, like

A	B	C	D	E	F	G	H	I	J	K	L	M
01	02	03	04	05	06	07	08	09	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

then this can be described using congruence theory. Let x represent the number of the plain text letter and let y represent the number of the encoded text letter, we see that $y \equiv x + 3 \pmod{26}$. So if Caesar wanted to send the message "Get the Greeks", we would translate that into numbers "07 05 20 20 08 05 07 18 05 05 11 19" and then applying the congruence rule, we get the encoded text "10 08 23 23 11 08 10 21 08 08 14 22." The receiver of this message only had to know the congruence rule. They would then apply the reverse rule $x \equiv y - 3 \pmod{26}$ to recover the original message.

This method of encoding messages is extremely simply and therefore not very secure. Caesar himself eventually abandoned such a scheme, but most likely because he did not trust those he was sending messages to. Similar cryptosystems rely on the fact that only the sender and the receiver have access to the encoding process. Most modern day cryptosystems however

are known as public-key cryptosystems. These make the encoding process public, but because of the number theory involved, only the sender and receiver are able to code and decode messages. The method I will outline shortly was invented in 1977 by Rivest, Shamir, and Adleman, and is called the RSA cryptosystem. The system relies heavily on the fact that even the most powerful computers have difficulty factoring large composite integers.

Essentially, we began this topic in Section 1.3. We looked at many primes, of many different types, and searched for some large primes. What we look for now is a way to determine is a given number is prime or composite. Our first tool in this search comes from the great Pierre Fermat. As he did with many of his results, he first stated this theorem in a letter to a friend, in this case Frenicle de Bessy. In the letter, dated October 18, 1640, Fermat stated what has become known as Fermat's Little Theorem but he did not give a proof. This was also commonplace for Fermat. Instead he indicated that he had a proof of the theorem, but it was too long to include in the letter. Both Leipniz (unpublished in 1683) and Euler (published in 1736) eventually provided proofs.

Theorem 2.3.1 (Fermat's Little Theorem) If p is a prime that does not divide a, then $a^{p-1} \equiv 1 \pmod{p}$.

We will illustrate this theorem with an example using very small numbers. Its true power lies in how it handles extremely large numbers with the same ease.

Example 2.3.2 Show that 20 is composite.

Choose $a = 7$. Notice that 20 does not divide a. However, $7^{19} \equiv 3 \pmod{20}$. By Fermat's Little Theorem, this means that 20 is not prime.

Exercise 2.3.3 Show that 30 is composite.

When Euler proved Fermat's Little Theorem, he actually proved a more general result using his phi function.

Definition 2.3.4 ***Euler's phi function*** is defined as follows:

$\phi(n) =$ the number of natural numbers less than n and relatively prime to n.

Example 2.3.5 Compute $\phi(20)$.

There are only 19 possible numbers, so all we need to do is eliminate those with common factors with 20. So we eliminate 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, and 18. That leaves $\phi(20) = 8$

Of course for large numbers, computing $\phi(n)$ in this fashion is unwieldy. In general, a formula would help.

Exercise 2.3.6 Make a list of $\phi(n)$ for $n \le 30$.

Exercise 2.3.7 Using your results in Exercise 2.3.6 as a guide, find a formula for:

(a) $\phi(p)$ where p is a prime

(b) $\phi(pq)$ where p and q are primes

(c) $\phi(p^2)$ where p is a prime.

(d) $\phi(p^n)$ where p is a prime.

Now back to Euler's generalization of Fermat's Little Theorem.

Theorem 2.3.8 (Euler's Theorem) Let n be a natural number and $\gcd(a,n)=1$. Then

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

Notice that Fermat's Little Theorem is a special case of this theorem.

Exercise 2.3.9 Compute $7^{64} \pmod{85}$. Explain how you got your answer.

So how do encryption systems use prime numbers and what does Fermat's Little Theorem have to do with it? I'm glad you asked. I'm now going to outline how RSA encryption works.

Each user chooses a pair of prime numbers p and q, large enough so that the product $n = pq$ cannot be factored with our current computational abilities. For example, if we pick a pair of primes each with 200 digits, n will have approximately 400 digits. This number n is called the ***enciphering modulus***.

Having chosen n, we also compute $\phi(n) = \phi(pq) = (p-1)(q-1)$ and the user then chooses a positive integer k (which is called the ***enciphering exponent***) for which $(k, \phi(n)) = 1$. The numbers n and k are then placed in a public file as the user's personal encryption key. This is accessible to anyone who wishes to send that user a message. Remember, the user knows the factorization of n, but no one else can find it.

Now to send this user a message, we take our message and translate it into a string of numbers (much as Caesar did) and break the string into small, manageable chunks of a few digits each (we want each piece to be smaller than n). So our message is now a sequence of numbers $a_1, a_2, \ldots, a_m$. Next we compute $a_1^{\,k} \pmod{n}, a_2^{\,k} \pmod{n}, \ldots, a_m^{\,k} \pmod{n}$. These values form a new list of numbers $b_1, b_2, \ldots, b_m$. This is the encoded message that is sent to the user.

To decode this message, the user needs to recover the numbers $a_1, a_2, \ldots, a_m$ from the numbers $b_1, b_2, \ldots, b_m$. Recall that each b_i is congruent to $a_i^{\,k} \pmod{n}$. So to determine a_i we

need to solve the congruence $x^k \equiv b_i \pmod{n}$. This is solvable as long as we know $\phi(n)$, which we do.

This whole process is secure because $\phi(n)$ is only known to the user. When there are hundreds of digits in n, finding $\phi(n)$ is prohibitive, even for the fastest computers. What makes this RSA cryptosystem even more useful is that one day in the future when computers have inevitably gotten fast enough to crack the code in a reasonable amount of time, all you have to do is choose larger prime numbers (say with 1000 digits) and it's uncrackable again.