Grade 11/12 Math Circles

Cryptography, Part 2

Introduction

In the previous lesson, we discussed various methods for encrypting and decrypting messages in such a way that no one but the intended recipient can remove the encryption and recover the original message. However, all of these methods had the same drawback: both sender and receiver are required to exchange a shared, secret *key* before they begin communicating. Because of the symmetry between the secret information the sender and receiver have, these methods of encryption are all grouped under the term *symmetric key cryptography* nowadays.

At the end of the previous lesson, we mentioned that in the 1970s, a whole new way of performing cryptography was invented, known as *public key cryptography*. Without the discovery of this form of cryptography, secure communication over the internet would have been impossible to implement! The goal of this lesson is to explain how public key cryptography works in the abstract, before introducing one of the first and most common methods for public key cryptography, known as the *RSA cryptosystem*.

Public Key Cryptography

The idea behind public key cryptography was invented independently in the 1970s by two different people: Whitfield Diffie, an American academic, and James Ellis, who worked for the British government. For a long time, credit for public key cryptography went to Diffie, because Ellis could not share his government work with the general public. That's the nature of this top-secret business!

The big breakthrough here is that encryption and decryption do not have to use the same key. Suppose for a moment that Nick can generate a pair of keys, which we call a *public key* and a *private key*. As the name suggests, the private key is kept secret and never revealed to anybody (even to Shefaza, who Nick wishes to communicate with). On the other hand, the public key is published *in public*, where everyone can see (not just Shefaza).

Anyone wishing to encrypt a message to Nick would use his public key on their plaintext to generate the ciphertext. Now the catch is that Nick's public key can only be used for encryption, and not for decryption! Nick's private key, on the other hand, is good only for decryption, and cannot be used to encrypt messages.

In Simon Singh's excellent book on cryptography, *The Code Book*, he gives a useful metaphor explaining how public key cryptography works. Suppose Nick has an unlimited supply of identical open padlocks (copies of the public key), but has only one key for opening these padlocks (the private key). Anyone wanting to send a secret message to Nick would pick up one of his unlocked padlocks from some public location. Then, they can put the message in a box and lock it with the padlock. Since Nick has the only key, he is the only one who can remove the locked padlock and read the message in the box.

Notice that public key cryptography removes the problem of getting keys from Nick to Shefaza. Nick can just publish his public key online, and Shefaza can download it at her convenience in order to send encrypted messages to Nick. Nick doesn't have to worry that his public key is so widely available, because it can't be used for decrypting messages. In the same way, if you want to send your credit card number to a company over the internet, you just need to download the company's public key, and then you're ready to encrypt.

Both Diffie and Ellis started out by defining public key cryptography in theoretical terms, but there was still work to do. For starters, is it even possible to describe a workable cipher that uses public and private keys? No matter how the cipher works, we have two basic requirements:

- It must be very hard to decrypt a message without the private key, *even if you know the public key*. In other words, the process of encryption must be easy to do, but extremely hard to undo.
- There must be an additional, special piece of information (the private key) that suddenly makes decryption very easy.

The goal of this lesson is to look at the *RSA cryptosystem*, which was the first proposed cipher meeting both of these requirements. In order to understand this cryptosystem, we will first have to take a look at a fascinating branch of math known as *number theory*. This is the branch of math that studies properties of *integers* and *prime numbers*, which we introduce below.

Prime Numbers

Mathematicians have a fancy name for the numbers $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$. We call them *the integers*. The *positive integers* are $1, 2, 3, 4, \ldots$ (the integers that happen to be positive). Not all integers are created equal, though! The coolest ones are the *prime numbers*. A positive integer p is called *prime* if its only positive factors are 1 and p. For instance, both 2 and 11 are prime.

Positive integers that aren't prime are called *composite*. For example, 21 is composite because

$21 = 3 \cdot 7$, so that 3 and 7 are both factors of 21 different from 1 and 21. There is one exception to this rule: even though 1 satisfies the definition of prime number as we've just stated it, we actually say that 1 is neither prime nor composite.

Stop and Think

Why would we not wish to consider 1 a prime number?

Unique Prime Factorization

Prime numbers are special because they form the building blocks for all positive integers. To be specific, every positive integer larger than 1 is either itself prime, or can be written as a product of two or more prime numbers. In fact, this can be done in exactly one way (except for rearranging the order of the factors, like when $3 \cdot 7 = 7 \cdot 3$).

Why might prime numbers be useful for public key cryptography? It turns out that it's a whole lot easier to multiply numbers together than it is to break up a number into its prime factors.

Stop and Think

To see how hard factorization can be, see how long it takes you (by hand) to decide whether 4019881 is prime, or if not, to find its prime factors!

Therefore, if the public key uses a number with large prime factors, and the private key involves the factors themselves, it would be a lot easier to calculate the public key knowing the private key than to find the secret prime factors, knowing only the public key. However, this comes with a word of caution: at the moment, all we know for certain is that the fastest-known tricks for factoring are monumentally slower than the fastest-known tricks for multiplying two numbers together. If someone ever comes along with a technique for fast factoring, the security of RSA will be completely destroyed!

Greatest Common Divisors

Another important ingredient in RSA is the idea of the *greatest common divisor* (or *gcd*) of two integers. First, we need an official definition of what it means for an integer to divide another one. If a and b are integers, we say that a *divides* b, and write $a \mid b$, if we can find another integer c such

that $b = ac$.

Example 1

Note that $2 \mid 6$, because $6 = 2 \cdot 3$. (Here $a = 2, b = 6, c = 3$.)

Likewise, note that $5 \mid 100$ because $100 = 5 \cdot 20$. (Here $a = 5, b = 100, c = 20$.) We have $-5 \mid 100$ too, because $100 = (-5) \cdot (-20)$.

On the other hand, 4 does not divide 6 (and we write $4 \nmid 6$). If it did, then we would have $6 = 4c$ for some integer c, so that $c = \frac{6}{4} = \frac{3}{2}$. But $\frac{3}{2}$ is not an integer.

Given two integers a and b, we can then talk about their greatest common divisor. This will be an integer d such that d is a *common* divisor of a and b, in the sense that $d \mid a$ and $d \mid b$. But d must also be the *greatest* such integer, meaning that no integer larger than d divides both a and b. We will use $\gcd(a, b)$ to refer to the greatest common divisor of a and b.

Example 2

How can we calculate the gcd of two integers? One way to do it is to factor both of those integers as products of primes.

For example, if we wanted to calculate $\gcd(12, 28)$, we can write

$$12 = 2 \cdot 2 \cdot 3$$
$$28 = 2 \cdot 2 \cdot 7.$$

From this, we take all the primes that are common to both factorizations, and multiply them together to get the gcd. Here, we get $\gcd(12, 28) = 2 \cdot 2 = 4$.

Exercise 1

Compute the following greatest common divisors by factoring the integers as products of primes.

(a) $\gcd(11, 31)$

(b) $\gcd(629, 357)$

(c) $\gcd(36652, 68552)$

In solving these exercises, you may have come to appreciate that calculating gcds in this way can become difficult when the numbers become large, equivalent in difficulty to factoring an integer into its product of primes. If there were no better way to calculate a gcd, then RSA would become impossible to use, since a gcd calculation is part of the process of setting up the public and private keys!

So now we discuss the efficient calculation of gcds, using a technique known as the *Euclidean algorithm*.

The Euclidean Algorithm

The basic procedure behind the Euclidean algorithm is division with remainder, which you may recognize from previous mathematical studies. Given two positive integers a and b, we can always find unique integers q and r such that $a = qb + r$, and where $0 \leq r < b$. The integer q is called the *quotient* of the division, and r is called the *remainder*.

The thought behind this process may be illustrated with an example. Suppose we wished to divide 101 by 11 with remainder. We take 101 and keep subtracting 11 until doing it one more time would give something negative:

$$\begin{aligned} 101 - 11 &= 90 \\ 90 - 11 &= 79 \\ 79 - 11 &= 68 \\ 68 - 11 &= 57 \\ 57 - 11 &= 46 \\ 46 - 11 &= 35 \\ 35 - 11 &= 24 \\ 24 - 11 &= 13 \\ 13 - 11 &= 2. \end{aligned}$$

This last number we arrive at is our remainder, so that $r = 2$. The number of times we subtracted 11 is our quotient, so that $q = 9$ in this case. Writing it out in the required format, we get $101 = 9 \cdot 11 + 2$.

If you have a calculator in hand, there is a substantial shortcut: typing in 101 divided by 11 gives $9.1818\ldots$. The value of q is just the integer you get when ignoring what comes after the decimal point

(in this case, as already noted, $q = 9$). To get r, you can then just calculate $a - qb = 101 - 9 \cdot 11 = 2$.

All right, so we've talked about divison with remainder, but how does it relate to calculating the gcd of two numbers? The answer lies in doing division with remainder repeatedly. Earlier, we verified that $\gcd(12, 28) = 4$. What happens if we divide 28 by 12 with remainder? We get

$$28 = 2 \cdot 12 + 4.$$

Here, the remainder is 4, which is the same as the gcd! Then, we can take 12 and divide it by 4 with remainder, to end up with

$$12 = 3 \cdot 4 + 0.$$

We finish with a remainder of 0, because 4 actually divides 12.

Okay, but why are we doing this? To make the point, let's try this with 11 and 31, where you hopefully verified that $\gcd(11, 31) = 1$ in Exercise 1. First, we divide 31 by 11 with remainder:

$$31 = 2 \cdot 11 + 9.$$

Here, the remainder is 9. Now, let's divide 11 by the remainder, 9:

$$11 = 1 \cdot 9 + 2.$$

Next, let's take 9 and divide it by the new remainder, 2:

$$9 = 4 \cdot 2 + 1.$$

Finally, we divide 2 by this newest remainder, 1:

$$2 = 2 \cdot 1 + 0.$$

We've now reached a remainder of 0, and the last remainder before that, 1, is actually the gcd of 11 and 31.

In both cases, the last remainder we reached before 0 was the gcd of the two numbers! Does this always work? The answer is yes, but the proof of this fact is not all that relevant to our purposes, so our time is better spent on other topics.

Now, we started out by saying that this procedure is more efficient at calculating gcds than prime

factorization. However, the computation of $\gcd(11, 31)$ using the Euclidean algorithm used a bunch of steps, and it was easier to compute this particular gcd using prime factorizations. The savings in time comes when the numbers are much larger.

Exercise 2

Use the Euclidean algorithm to calculate the following gcds:

(a) $\gcd(36652, 68552)$ (this was done in Exercise 1 in another way)

(b) $\gcd(4019881, 10394)$

Back-Substitution (Extended Euclidean Algorithm)

Once we have run the Euclidean algorithm on two integers a and b to calculate $\gcd(a, b)$, the work we've done has one extra side benefit: we can use it to find integers x and y such that $ax + by = \gcd(a, b)$. In other words, we can use the calculations to write $\gcd(a, b)$ as a sum of a multiple of a and a multiple of b. This will also be useful when we discuss the RSA cryptosystem.

Let's explain how this works using the $\gcd(31, 11)$ computation we carried out earlier.

Example 3

When we verified that $\gcd(31, 11) = 1$ using the Euclidean algorithm, we performed the following divisions with remainder:

$$31 = 2 \cdot 11 + 9$$
$$11 = 1 \cdot 9 + 2$$
$$9 = 4 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0.$$

We would now like to use these calculations to find integers x and y such that $31x + 11y = 1 = \gcd(31, 11)$. Let's ignore the last line of this computation, and focus on the first three. We re-write each of these equations so that the remainder is by itself on one side of the equation,

and write the three equations in reverse order (the order in which we'll be using them):

$$1 = 9 - 4 \cdot 2$$
$$2 = 11 - 1 \cdot 9$$
$$9 = 31 - 2 \cdot 11.$$

Notice that the first equation gives 1 as the sum of a multiple of 9 and a multiple of 2 (to be specific, $1 \cdot 9 + (-4) \cdot 2$). We'd like to write 1 as a multiple of 31 plus a multiple of 11, in the end. To help with this, we use the second equation to substitute for the number 2 in the first equation, replacing 2 with a multiple of 11 plus a multiple of 9. When this is done, we will have written 1 as a multiple of 11 plus a multiple of 9 as well:

$$\begin{aligned} 1 &= 9 - 4 \cdot 2 \\ &= 9 - 4 \cdot (11 - 1 \cdot 9) \\ &= 9 - 4 \cdot 11 + 4 \cdot 9 \\ &= 5 \cdot 9 - 4 \cdot 11. \end{aligned}$$

Notice at every step that we've collected the terms in such a way that we're always working with multiples of 9 and 11. Finally, we use the third equation from the re-arranged Euclidean algorithm work to substitute for the number 9, replacing it with a multiple of 11 plus a multiple of 31. This will allow us to write 1 as a multiple of 31 plus a multiple of 11, as we wish:

$$\begin{aligned} 1 &= 5 \cdot 9 - 4 \cdot 11 \\ &= 5 \cdot (31 - 2 \cdot 11) - 4 \cdot 11 \\ &= 5 \cdot 31 - 10 \cdot 11 - 4 \cdot 11 \\ &= 5 \cdot 31 - 14 \cdot 11. \end{aligned}$$

Again, note how we always collect the terms in such a way that we're working with multiples of 11 and 31. We now conclude that a solution to $31x + 11y = 1 = \gcd(31, 11)$ is given by $x = 5$ and $y = -14$.

The technique used in the previous example is often called the *Extended Euclidean algorithm*, and it relies on a process of *back substitution* using the work from the ordinary Euclidean algorithm. The best way to get a feel for how this works is to try a couple examples yourself:

Exercise 3

Using the Extended Euclidean algorithm, find integers x and y satisfying each of the following equations:

(a) $28x + 12y = 4$ (**Hint:** We calculated $\gcd(28, 12)$ using the Euclidean algorithm earlier.)

(b) $63x + 91y = 7$

The Euler ϕ Function

There is one more important gcd-related ingredient to RSA that we must discuss. This tool is called the *Euler phi function*. (By the way, "Euler" is pronounced like "oiler", rather than "you-ler".) Given a positive integer n, we define $\phi(n)$ to be the number of integers m between 1 and n such that $\gcd(m, n) = 1$.

Example 4

To see a couple examples of Euler phi function calculations, notice that $\phi(5) = 4$. This is because $1, 2, 3, 4$ do not share any common prime factors with 5, so that $\gcd(m, 5) = 1$ for $m = 1, 2, 3, 4$. But $\gcd(5, 5) \neq 1$, so exactly four numbers m between 1 and 5 satisfy $\gcd(m, 5) = 1$.

On the other hand, note that $\phi(8) = 4$, because $1, 3, 5, 7$ do not share any prime factors in common with 8, while $2, 4, 6, 8$ do.

You might already guess that it's hard to take an exhaustive approach like the one above to calculate $\phi(n)$ when n is large. You might be hoping that there's a formula for calculating $\phi(n)$, and if so, your hopes are about to come true! The only catch is that we need the prime factorization of n in order to make the calculation.

For RSA, it turns out we only need one special case of this formula.

Example 5

To work up to the formula needed for RSA, let's start with calculating $\phi(p)$, where p is a prime number. By definition of a prime, p has no positive factors except for 1 and p. So, if we take any integer m with $1 \leq m \leq p$, then notice that $\gcd(m, p)$ is a factor of p, so it must be equal

to either 1 or p. But, if $\gcd(m, p) = p$, then p must be a factor of m, which cannot happen if m is smaller than p.

Hence, we must have $\gcd(m, p) = 1$ for all integers m between 1 and $p - 1$, and $\gcd(p, p) = 1$, so $\phi(p) = p - 1$ for any prime p.

Example 6

For the setup of RSA, we will be interested in calculating $\phi(n)$ in the case where $n = pq$, where p and q are distinct prime numbers. Suppose we are given an integer m such that $1 \leq m \leq n$. What can the possible values for $\gcd(m, n)$ be? Since the gcd must divide n, there still aren't too many possibilities. Indeed, the only options are $1, p, q$, and pq.

Furthermore, since we've taken $1 \leq m \leq n$, the only way we can have $\gcd(m, n) = pq = n$ is if $m = n$, since $\gcd(m, n)$ must divide m, and no number larger than m can do that.

If $\gcd(m, n) = p$, then m must be a multiple of p. It's easy to count up how many of those there are in the range of possible values for m. Indeed, we can have $m = p, 2 \cdot p, 3 \cdot p, \ldots$, all the way up to $q \cdot p$. There are exactly q numbers listed here, but we already counted $m = qp = n$ elsewhere, so we only get $q - 1$ numbers m for which $\gcd(m, n) = p$.

Similarly, if $\gcd(m, n) = q$, then m must be a multiple of q, which means we could have $m = q, 2 \cdot q, 3 \cdot q, \ldots, p \cdot q$. All the numbers (except for $p \cdot q$) are different from the ones on the list of multiples of p, because none of $q, 2 \cdot q, 3 \cdot q, \ldots, (p-1) \cdot q$ are divisible by p. This is where the assumption that p and q are different is important! This gives us $p - 1$ additional integers m in the desired range such that $\gcd(m, n) = q$.

Now, if we want to count the number of integers m such that $1 \leq m \leq n$ and $\gcd(m, n) = 1$, we can do it by elimination. We start with the full list of n integers between 1 and m, then subtract off the ones for which the gcd is equal to either n, p, or q. This gives us $n-1-(p-1)-(q-1) = n - 1 - p + 1 - q + 1 = n - p - q + 1$ such positive integers.

However, since $n = pq$, we can write the final answer in a more pleasing way by factoring. We find that

$$\phi(n) = pq - p - q + 1 = (p - 1)(q - 1)$$

when $n = pq$ and p and q are distinct primes.

As we will see, a number n of the form described in Example 6 and the value of $\phi(n)$ are both

important parts of the private key in RSA. Part of the security of RSA is based around the fact that $\phi(n)$ cannot be computed quickly *without* knowing the prime factors of n.

Modular Arithmetic, A.K.A. Clock Arithmetic

Our final ingredient in RSA is a way of performing arithmetic, which you use whenever you perform calculations with time. If it is 10 o'clock now and your friend wants to meet you in three hours, you do the mental addition, get an answer of 13 o'clock, but then silently change that answer to 1 o'clock because clocks only have 12 hours on them.

When you get to 12, you "loop" around and start back at 1 again. When you do this, mathematicians would say you are doing arithmetic "modulo 12". If we wanted to write this calculation mathematically, we would say

$$10 + 3 \equiv 1 \pmod{12}.$$

Out loud, we would read it as "10 plus 3 is congruent to 1 modulo 12". We use the $\equiv$ sign rather than the $=$ sign to signal that we're working with modular arithmetic, and the $\pmod{12}$ signals that the looping around happens when we get to 12.

If you think about this a little more deeply, the same thing happens when we work with minutes too. If you have ever listened to CBC Radio, you may hear the host announce the time as "45 minutes past the hour, or 15 minutes past the hour in Newfoundland". The island of Newfoundland has the charming quirk of setting their clocks half an hour ahead of the rest of Atlantic Canada, which means national radio hosts have to do a lot of clock arithmetic.

If you start with 45 minutes past the hour and add 30 minutes, you'll end up with 75 minutes. Since there are only 60 minutes in an hour, we'll have to loop around and start counting back at 0 when we hit 60 minutes. This way, we get the correct answer of 15 minutes past the hour, rather than saying "75 minutes past the hour". Here, we're doing calculations "modulo 60". To write this mathematically,

$$45 + 30 \equiv 15 \pmod{60}.$$

Calculations involving time are done modulo 12 or modulo 60, but there's nothing stopping us from doing arithmetic modulo other integers, like 7. If we add 3 and 5 modulo 7, we first get 8, but we know we can loop around and start again when we get to 7, so we can actually say

$$3 + 5 \equiv 1 \pmod{7}.$$

(Thinking about a hypothetical seven-hour clock helps with visualizing this.)

Since we loop around every time we get to 7, adding (or subtracting) multiples of 7 from an integer does not change the result modulo 7. Therefore, we can say all of the following things:

$$\begin{aligned} 9 &\equiv 2 \pmod 7 \\ -5 &\equiv 2 \pmod 7 \\ 72 &\equiv 2 \pmod 7. \end{aligned}$$

All of 9, -5, and 72 differ from each other by a multiple of 7, which is why all of these are "the same" modulo 7.

Officially, we would define two integers a and b to be congruent modulo 7 if their difference is a multiple of 7, or in other words, if $7 \mid (a-b)$. Similarly, given two integers a and b and any positive integer n, we would define $a \equiv b \pmod n$ to mean that $n \mid (a-b)$, building off of the examples above.

One of the best features of this definition (from the point of view of RSA) is that the operations of addition, subtraction, and multiplication all behave nicely with respect to congruence modulo n. For instance, we noted that $1 \equiv 8 \pmod 7$ and $2 \equiv 72 \pmod 7$, so 1 and 8 can be used interchangeably in calculations, and so can 2 and 72. For example,

$$\begin{aligned} 1 \cdot 2 &\equiv 2 \pmod 7 \\ 8 \cdot 72 &\equiv 576 \pmod 7, \end{aligned}$$

and $576 \equiv 2 \pmod 7$ because $576 - 2 = 574 = 7 \cdot 82$.

What really makes RSA work is that the powers of a number often behave unpredictably in modular arithmetic.

Example 7

If we work modulo $n = 31$, let's see how the powers of 3 behave:

$$\begin{aligned}
3^1 &\equiv 3 \pmod{31} \\
3^2 &\equiv 9 \pmod{31} \\
3^3 &\equiv 27 \pmod{31} \\
3^4 &\equiv 3 \cdot 27 \equiv 3 \cdot (-4) \equiv -12 \equiv 19 \pmod{31} \\
3^5 &\equiv 3 \cdot 3^4 \equiv 3 \cdot 19 \equiv 57 \equiv 26 \pmod{31} \\
3^6 &\equiv 3 \cdot 3^5 \equiv 3 \cdot 26 \equiv 3 \cdot (-5) \equiv -15 \equiv 16 \pmod{31}.
\end{aligned}$$

Each time, notice how we choose to write the final result of the computation as an integer between 0 and 30, because that's the simplest way to represent an integer modulo 31 (think of it like the way we always report minutes in the hour between 0 and 59).

Writing the answers in this way, notice how unpredictable the list of powers of 3 is after the first few steps: $3, 9, 27, 19, 26, 16, \ldots$. Suppose we were to try to find an exponent e such that $3^e \equiv 2 \pmod{31}$, if one exists at all. How would you find it, other than exhaustively computing powers of 3? The fact that no one knows an easy way to solve such problems computationally is another feature RSA uses to guarantee security.

The RSA Cryptosystem

At last, we have everything we need to describe the first working example of public key cryptography. Here, the plaintext and ciphertext will be numbers, not strings of letters. This might seem strange compared to our examples from the previous lesson, but makes perfect sense in our digital age. Everything stored in computers is a sequence of 0s and 1s. In other words, every piece of data is represented by a number (admittedly, one in base 2 rather than our usual base 10). Therefore, every message you send with your computer can be represented as a number.

For Nick and Shefaza to use public key cryptography, remember that Nick needs to set up both a public key (for others to send encrypted messages to Nick), and a private key (so that Nick can decrypt the messages he receives).

To create a public key, Nick will perform the following steps:

1. Choose two different, very large prime numbers p and q.
2. Multiply p and q together to obtain an integer N. At the same time, calculate $\phi(N) = (p-1)(q-1)$ to use later. To encrypt and decrypt messages, we will be working modulo N.
3. Choose an *encryption exponent* e, an integer between 2 and $\phi(N)-1$, such that $\gcd(e, \phi(N)) = 1$. (We would make this gcd calculation using the Euclidean algorithm.)

At this point, Nick can tell everyone the values of N and e: this will be his public key. To calculate his private key, there is only one more step:

4. Find integers d and f such that $de + f\phi(N) = 1$, and choose the solution so that $0 \leq d \leq \phi(N) - 1$. (To obtain these values of d and f, we would run the Extended Euclidean algorithm on e and $\phi(N)$. Some adjustment may be needed to obtain a value of d in the desired range, though.)

Nick's private key is the number d. At this point, he should lock away (or destroy) the values of p, q, and $\phi(N)$ and never tell them to anyone. If he did, the private key would be compromised. Why? If Nick ever told anyone the value of $\phi(N)$, they could compute the value of d in the same way Nick did, using just e and $\phi(N)$, because e is already public information.

Likewise, if Nick ever leaked the value of either of p or q, then someone could easily get the other prime factor through ordinary division. As we know, it's easy to calculate $\phi(N)$ when you know p and q (that's how Nick did it in the first place), and we just explained why leaking the value of $\phi(N)$ is a very bad thing.

The last thing we need to do is describe how encryption and decryption work. Let's say Shefaza wants to send M as a plaintext (M for "message"), where M is between 0 and $N-1$. She can look up the public numbers e and N, and then calculate the unique number C between 0 and $N-1$ such that

$$C \equiv M^e \pmod{N}.$$

(Here, C is for "ciphertext".) Shefaza would then send C to Nick.

To decrypt, Nick takes C and calculates the unique integer M_0 between 0 and $N-1$ such that

$$M_0 \equiv C^d \pmod{N}.$$

Notice Nick is the only one who knows what d is, so he is the only one able to do this. Somewhat magically, it turns out that $M_0 = M$, resulting in a successful decryption!

The proof that RSA works depends on some mathematical results called *Euler's theorem* and the

Chinese Remainder Theorem, and to give that proof would make this already-long lesson even longer! So, we will leave the justification out.

For now, we just illustrate with a single example, and leave it to you to try some more!

Example 8

Let's say Nick chooses the primes $p = 5$ and $q = 7$. (In reality, these primes should have hundreds of digits, but this is easier to follow.) He would then compute

$$N = p \cdot q = 5 \cdot 7 = 35$$
$$\phi(N) = (p-1) \cdot (q-1) = 4 \cdot 6 = 24.$$

Let's also suppose Nick chooses $e = 5$ for his encryption exponent (you can easily check that $\gcd(5, 24) = 1$). Nick would go ahead and publish $N = 35$ and $e = 5$ for his public key. Then, to calculate his private key, he would run the Extended Euclidean algorithm to find that $5d + 24f = 1$ has a solution with $d = 5$ and $f = -1$. So, as it turns out, Nick's decryption exponent d is also 5.

Now, let's say Shefaza comes along and wants to send $M = 3$ as a plaintext. She computes $M^e = 3^5$ modulo 35:

$$3^5 \equiv 243 \equiv 33 \pmod{35}.$$

Shefaza sends $C = 33$ as a ciphertext. When Nick receives this ciphertext, he computes $C^d = 33^5$ modulo 35:

$$33^5 \equiv (-2)^5 \equiv -32 \equiv 3 \pmod{35}.$$

So Nick retrieves $M_0 = 3$, the same as the plaintext Shefaza started with. Cool!

Exercise 4

Try this for yourself! Choose primes p and q, and calculate the public and private keys. If you have a friend to try this with, have them send you a ciphertext and try to decrypt it.