Math Circles – Cryptography

Nickolas Rollick – nrollick@uwaterloo.ca

March 7, 2018

Introduction

Last time, we talked about *public key cryptography*, an idea from the 1970s that opened the door for encryption in the internet era. However, the idea is not enough - we still have to describe a real cryptosystem showing public key cryptography is possible in practice. Today, we accomplish this by describing the RSA cryptosystem. The math behind this cryptosystem is really interesting by itself, and will occupy us for most of today's meeting.

RSA Math I: 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 $1, 2, 3, 4, \ldots$ (notice these are just the integers that happen to be positive). Not all positive integers are created equal, though. The coolest ones are the *prime numbers*. A positive integer p is *prime* if its only factors are 1 and p. For example, 2 and 11 are both prime. Positive integers that aren't prime are called *composite*. For instance, 21 is composite because $21 = 3 \cdot 7$. In other words, 3 and 7 are both (prime) 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.

Unique Prime Factorization

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

Just for fun, let's give an argument (or *proof*) of why the first fact is true. Math is full of proofs, and math professors spend much of their time writing proofs of new facts. What is a proof, you might ask? Basically, it's an argument that you can't possibly disagree with. Every part of it seems so naturally true, and they build on each other to form an interesting conclusion.

So how do we prove that every positive integer (except 1) is prime or a product of primes? We start by noticing this is definitely true for the number 2, since 2 is prime. Now, suppose we know the numbers $2, 3, \ldots, n$ are all prime or products of primes. We consider the number $n+1$, and show that it too is prime or a product of primes. Certainly, $n+1$ is either prime or composite. If it's prime, we have what we want. Otherwise, $n+1$ is composite, so it has a factor m different from 1 and $n+1$. In particular, m is one of the numbers $2, 3, \ldots, n$. Also, we can write $n+1 = mk$, where k is another positive integer different from 1 and $n+1$, thus belonging to the list $2, 3, \ldots, n$.

But we assumed that $2, 3, \ldots, n$ were all either prime or products of primes, so m and k both have this property. Now $n+1$ is a product of the numbers m and k, both either prime or products of primes. In other words, $n+1$ is a product of primes!

This strategy actually proves that *every* positive integer is prime or a product of primes. For those who are interested, we used a technique called *mathematical induction*, which is a standard tool in a mathematician's toolbox. The proof that a positive integer is a product of primes in exactly one way (except for order) is best to keep until later, when we have more tools to work with.

Now, here's a hint about how prime numbers might be useful for public key cryptography. It turns out it's a whole lot easier to multiply numbers together than it is to break up a number into its prime factors. If you don't believe me, see how long it takes you 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 somehow involves knowing those prime factors, it would be a lot easier to calculate the public key knowing the prime factors than it would be to find the prime factors knowing only the public key.

I should say 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 numbers together. If someone ever comes along with a technique for fast factoring, the security of RSA will be completely destroyed!

Greatest Common Divisors (GCDs)

Another important ingredient in RSA is the idea of the *greatest common divisor* (or *gcd*) of two integers. First, we should say officially 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 $b = ac$. For example, $2 \mid 6$ because $6 = 2 \cdot 3$ (here, $a = 2, b = 6, c = 3$). Likewise, $5 \mid 100$ because $100 = 5 \cdot 20$. On the other hand, 4 does not divide 6. If it did, we would have $6 = 4 \cdot c$ for some integer c, but then we must have $c = \frac{6}{4} = \frac{3}{2}$, which is not actually an integer.

Given two integers a and b, we can then talk about their greatest common divisor. This will be an integer d that 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 greastest common divisor of a and b.

How do we actually calculate the gcd of two integers? One way to do it is by factoring both of them into primes. For instance, if we want to know $\gcd(12, 28)$, we can write $12 = 2 \cdot 2 \cdot 3$ and $28 = 2 \cdot 2 \cdot 7$. From this, we notice both 12 and 28 are divisible by $2 \cdot 2 = 4$, and they have no other prime factors in common. Therefore, $\gcd(12, 28) = 4$. Now, over to you: what is $\gcd(11, 31)$?

This method of calculating gcds requires factoring numbers into primes, which we just said was not easy to do. What is truly amazing is that there is another way to calculate gcds, one that is much faster for a computer!

The Euclidean Algorithm

The basic idea boils down to division with remainder, something you might remember from past math classes. Given two positive integers a and b, we can always find *unique* integers q and r such that $a = bq + r$, and $0 \leq r < b$. The number r is the *remainder* of the division.

This is best illustrated with an example. Let's say we had to divide 101 by 11 with remainder. We take 101 and keep subtracting multiples of 11 until doing it again would give us something negative:

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

The last number we get is our remainder, i.e. $r = 2$. We had to take away 11 nine times, so we take $q = 9$ and conclude that $101 = 11 \cdot 9 + 2$. If you have a calculator in hand, there is a substantial shortcut: typing in $101/11$ gives $9.1818\ldots$. The value of q is just the integer you get by ignoring what comes after the decimal

point (in this case, 9). To get r, we just calculate $a - bq = 101 - 11 \cdot 9 = 2$. On the question sheet, you will argue why this always works.

All right, so far, so good. We have division with remainder, but how does it relate to the gcd of two numbers? The answer lies in doing division with remainder repeatedly. Earlier, we checked 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, the same as the gcd! Then if we take 12 and divide it by 4 with remainder, we end up with

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

We finish with a remainder of 0, because 4 actually divides 12. Okay, why am I doing this? Notice the last remainder we get before 0 is the gcd!

Maybe that was a fluke. Let's try this with our other example, $\gcd(11, 31)$. First divide 31 by 11 with remainder:

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

The remainder is 9. Now let's divide 11 by 9 with remainder:

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

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

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

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

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

Once again, we got to a remainder of 0, though it took a few more steps. Again, the last remainder we had before 0 is the gcd we wanted! Does this always work?

The answer is yes, but the proof is not super-enlightening, so our time is better spent elsewhere. Now, I started out by saying that this was much faster for computers than finding factors of a and b. But the computation of $\gcd(11, 31)$ needed a bunch of steps, and it was easy to find this gcd by factorization. The savings in time comes when the numbers a and b are much larger. It is faster to find, say, $\gcd(4019881, 10394)$ by a bunch of divisions with remainder than it is to factor these numbers.

The Euclidean algorithm lets us do one other cool thing, if we run it backwards. Once we have calculated $\gcd(a, b)$, it lets us find integers x and y such that $ax + by = \gcd(a, b)$. This is one of the steps in RSA, and it has other uses, too (one of which you'll see somewhere on the question sheet).

Let's explain how this works with our two examples. When finding $\gcd(12, 28)$, one of the equations we found was $28 = 2 \cdot 12 + 4$. Writing this differently, we get $1 \cdot 28 - 2 \cdot 12 = 4$. So, if we wanted to find integers x and y that solve $12x + 28y = \gcd(12, 28)$, we could take $x = -2$ and $y = 1$.

This is doable with our second example too, but there's more steps. To calculate $\gcd(11, 31)$, we did the following divisions with remainder:

$$\begin{aligned} 31 &= 2 \cdot 11 + 9 \\ 11 &= 1 \cdot 9 + 2 \\ 9 &= 4 \cdot 2 + 1. \end{aligned}$$

We want to find integers x and y such that $11x + 31y = \gcd(11, 31)$. To do this, we work through the steps backwards. Start with the last equation, writing it as $1 = 9 - 4 \cdot 2$. From the second equation, we know $2 = 11 - 1 \cdot 9$, so

$$1 = 9 - 4 \cdot (11 - 1 \cdot 9) = 5 \cdot 9 - 4 \cdot 11.$$

Now, the first equation tells us $9 = 31 - 2 \cdot 11$, so

$$1 = 5 \cdot 9 - 4 \cdot 11 = 5 \cdot (31 - 2 \cdot 11) - 4 \cdot 11 = 5 \cdot 31 - 14 \cdot 11.$$

Summing up, the equation $11x + 31y = \gcd(11, 31) = 1$ has the integer solutions $x = -14$ and $y = 5$. This trick works all the time, but writing down the proof is not as enlightening as trying a few examples yourself.

The Euler ϕ Function

Before leaving gcds behind, there is an important function we need in order to describe RSA, called the *Euler phi function*. (By the way, "Euler" is pronounced "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 for which $\gcd(m, n) = 1$. To see a couple quick examples, we have $\phi(5) = 4$, since 1, 2, 3, and 4 do not share any common prime factors with 5, but $\gcd(5, 5) = 5$. On the other hand, $\phi(8) = 4$, because 1, 3, 5, and 7 share no common prime factors with 8, while 2, 4, 6, and 8 all do (each one is divisible by 2).

Because of the way we computed those examples, you might suspect there is a formula for $\phi(n)$ in terms of the prime factors of n. If so, you would be right! For RSA, we'll only need to worry about one special case of this formula. As a warm-up, let's find $\phi(p)$, where p is a prime number. By definition of a prime, p has no factors except for 1 and p. If we take any integer m with $1 \leq m \leq p$, notice that $\gcd(m, p)$ is a factor of p, so it must either be 1 or p. Now, if $\gcd(m, p) = p$, then p divides m, meaning it shows up as one of the prime factors of m. This can't possibly be true if m is smaller than p (if you don't see this right away, it's worth thinking about).

In conclusion, we are forced to have $\gcd(m, p) = 1$ for all integers m between 1 and $p - 1$. Notice there are exactly $p - 1$ of those. Since $\gcd(p, p) = p$, it follows that $\phi(p) = p - 1$ when p is prime. Notice this agrees with our earlier calculation that $\phi(5) = 4$, taking $p = 5$.

Now, for RSA, we really want to calculate $\phi(n)$ in the case where $n = pq$, with p and q different prime numbers. Suppose we have an integer m such that $1 \leq m \leq n$. What are the possible values for $\gcd(m, n)$? Well, this gcd must divide n, so there aren't too many possibilities. The only options are 1, p, q, or pq. Since we've taken $1 \leq m \leq n$, the only way we can get $\gcd(m, n) = pq = n$ is if $m = n$, because the gcd has to divide m, and no integer larger than m can do that.

If $\gcd(m, n) = p$, then m has to be a multiple of p. It's easy to count up how many of those there are in the possible range of values for m. We could have $m = p, p \cdot 2, p \cdot 3, \ldots$, all the way up to $p \cdot q$. There are exactly q numbers on that list, but we counted $p \cdot q$ already in the previous case, so we only get $q - 1$ new possibilities.

Similarly, if $\gcd(m, n) = q$, then m has to be a multiple of q, which means we could have $m = q, 2 \cdot q, 3 \cdot q, \ldots, p \cdot q$. All of the numbers on this list (except for $p \cdot q$) are different from the ones on the last list, because none of $q, 2 \cdot q, \ldots, (p - 1) \cdot q$ are divisible by p. This is where the assumption that p and q are different is important! So, we find $p - 1$ integers m such that $1 \leq m \leq n$ and $\gcd(m, n) = q$.

Finally, if we want to count up the integers m such that $1 \leq m \leq n$ and $\gcd(m, n) = 1$, we can do it by elimination. Start with all n integers between 1 and n, and subtract off the ones where the gcd is n, p, or q. This gives us $n - 1 - (q - 1) - (p - 1) = n - 1 - q + 1 - p + 1 = n - p - q + 1$ such positive integers. But $n = pq$, so we can write the final count in a much more pleasing way with a little factoring. We find that

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

when $n = pq$, with p and q different primes. Again, notice how easy it is to calculate $\phi(n)$ if you know the prime factors of n. It turns out that $\phi(n)$ is part of the information needed to compute the private key in RSA. Part of the security of RSA lies in the fact that $\phi(n)$ cannot be computed quickly *without* knowing the prime factors of n.

All right, it's time to take a little break to try some questions of your own. You can now attempt Questions 1–6 on Question Sheet 2.

RSA Math II: Modular Arithmetic, A.K.A Clock Arithmetic

The other main ingredient in RSA is something you work with whenever you do calculations with time. If it's 10 o'clock now and your friend wants to meet in three hours, you do the mental addition, get the answer 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, you are really doing arithmetic "modulo 12". If we wanted to write this calculation in the way mathematicians do it, we'd say

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

Out loud, this reads "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 $\pmod{12}$ tells us that the looping around begins when we get to 12.

If we think about this a little more deeply, the same thing crops up when talking about minutes, too. If, like me, you listen to a lot of CBC radio, you might 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 this charming quirk of setting their clocks half an hour ahead of the rest of Atlantic Canada, which means the 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 have to loop back around and start counting 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,

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

We work modulo 12 and 60 all the time because of the applications to telling time, 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 this). Since we loop around every time we get to 7, adding (or subtracting) multiples of 7 to an integer does not change the answer 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 2 by a multiple of 7, which is why all of these are "the same" modulo 7.

This idea leads to the "mathematical" definition of being congruent modulo 7. You can check that for integers a and b, we have $a \equiv b \pmod{7}$ if $7 \mid (a - b)$, which is just to say that a and b differ by a multiple of 7. Of course, there's nothing special about 7, 12, or 60 in this respect. Given any positive integer n, we say that $a \equiv b \pmod{n}$ if $n \mid (a - b)$, which is just a fancy way to say that a and b differ by a multiple of n. This captures the idea that we "loop" around every time we get to n.

What is really cool is that addition, subtraction, and multiplication all behave well with this definition of congruence modulo n. In other words, if numbers are congruent ("the same") modulo n, we can choose whichever one we like when adding, subtracting, or multiplying, and the answer won't change. For example, we noticed that $1 \equiv 8 \pmod{7}$ and $2 \equiv 72 \pmod{7}$. Working modulo 7, 1 and 8 can be used interchangeably, as can 2 and 72. Observe:

$$\begin{aligned} 1 + 2 &\equiv 3 \pmod{7} \\ 8 + 72 = 80 &\equiv 3 \pmod{7}, \end{aligned}$$

with the second one being true because $80 - 3 = 77$, a multiple of 7. The same goes for multiplication:

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

where the second one is true because $576 - 2 = 574 = 7 \cdot 82$.

What really makes RSA work is the fact that powers of a number often behave very unpredictably mod

n. Let's take $n = 31$ and see how the powers of 3 behave:

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

Each time, I chose to represent the final answer as a number between 0 and 30, because that's the simplest way to write down a given integer modulo 31. When the answers are written this way, it becomes very unpredictable to track the powers of 3. Already, they're bouncing all over the place, from 27 to 19 to 26 to 16... Suppose I asked you to find an exponent e such that $3^e \equiv 2 \pmod{31}$, if one exists. How would you find it, other than computing powers of 3 to the point of exhaustion? The fact that no one knows an easy way to do such problems computationally is another feature RSA uses to guarantee security.

Nonetheless, there is order to the apparent chaos. There's a crucial fact used in RSA that combines modular arithmetic with our old friend, the ϕ function. This little tidbit often goes by the name of Euler's theorem. If we have integers a and n such that $\gcd(a, n) = 1$, then it turns out that

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

You are welcome to try and prove this, but all the proofs I'm aware of use a little more than what we've talked about today. It's neat to check this in small cases, though. If we take $n = 5$, a prime, we saw earlier that $\phi(5) = 5 - 1 = 4$. Now notice that

$$\begin{aligned}
1^4 &\equiv 1 \pmod{5} \\
2^4 &\equiv 16 \equiv 1 \pmod{5} \\
3^4 &\equiv 81 \equiv 1 \pmod{5} \\
4^4 &\equiv 256 \equiv 1 \pmod{5}.
\end{aligned}$$

Given how random the powers of a number seem to behave modulo n, this is very unexpected...

The RSA Cryptosystem

At last, we have everything we need to talk about the first working example of public key cryptography. Here, the plaintext and ciphertext are numbers, not strings of letters. This might seem strange compared to our examples from last time, but makes total sense in our digital age. Everything stored in a computer is a sequence of 0s and 1s; in other words, every piece of data is represented by a number (admittedly, one written in base 2 rather than the usual base 10). Therefore, every message you send with your computer actually takes the form of a number.

In public key cryptography, remember that Alice needs to set up both a public key (so people can encrypt messages to Alice), and a private key (so Alice can decrypt those messages). To create a public key, Alice performs the following steps:

1. Choose two different, very large primes p and q.
2. Multiply p and q together to get an integer N. At the same time, compute $\phi(N) = (p-1)(q-1)$ to use later. To encrypt and decrypt messages, we will be working modulo N.
3. Choose an *encryption exponent*, an integer e between 2 and $\phi(N) - 1$, such that $\gcd(e, \phi(N)) = 1$.

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

4. Find integers d and f such that $de + f\phi(N) = 1$, where we specify that $0 \leq d \leq \phi(N) - 1$.

To actually find d and f, Alice can take the steps used to compute $\gcd(e, \phi(N))$ and run the Euclidean algorithm backwards, just like we did earlier.

Alice's private key is d. At this point, Alice should lock away the values of p, q, and $\phi(N)$ and never tell anybody what they are. If she does, her private key will be compromised. Why? If Alice ever told anybody the value of $\phi(N)$, they could calculate her private key in the same way Alice did, because e is public information. After that, they can decrypt every message intended for Alice. Likewise, if Alice ever leaked the value of even one of p or q, then someone could easily find the other one through ordinary division. It's easy to calculate $\phi(N)$ when you know p and q (that's how Alice did it), and we just mentioned why leaking the value of $\phi(N)$ is a bad thing.

Okay, but there's one thing missing: how do people encrypt and decrypt messages? Let's say Bob wants to send the number M as a plaintext, where M is between 0 and $N - 1$. He can look up e and N, and calculate the unique integer C such that

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

and $0 \leq C \leq N - 1$. This value of C is the ciphertext that Bob sends to Alice.

To decrypt, Alice takes C, and calculates the unique integer M_0 such that

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

and $0 \leq M_0 \leq N - 1$. Notice Alice is the only one who knows what d is, so she is the only one who can do this. The decryption works as long as $M_0 = M$. We'll check why this is true in the special case that $\gcd(M, N) = 1$ (which is true most of the time anyway). Alice receives an integer C, and we know that $C \equiv M^e \pmod{N}$. Therefore, when she goes to calculate M_0, the result is

$$M_0 \equiv C^d \equiv (M^e)^d \equiv M^{ed} \pmod{N}.$$

Now here's where the clever part comes in. We chose d in such a way that $ed + f\phi(N) = 1$ for some integer f. Therefore, $ed = 1 - f\phi(N)$. We conclude that $M^{ed} = M^{1-f\phi(N)} = M \cdot (M^{\phi(N)})^{-f}$, using the rules of exponents. All in all, we see that

$$M_0 \equiv M^{ed} \equiv M \cdot (M^{\phi(N)})^{-f} \pmod{N}.$$

The final step is sheer magic: remember how we talked about Euler's theorem earlier? Since $\gcd(M, N) = 1$, it tells us that $M^{\phi(N)} \equiv 1 \pmod{N}$. Then $(M^{\phi(N)})^{-f} \equiv 1^{-f} \equiv 1 \pmod{N}$. So,

$$M_0 \equiv M \cdot (M^{\phi(N)})^{-f} \equiv M \cdot 1 \equiv M \pmod{N}.$$

Therefore, N divides $M_0 - M$. But M_0 and M are both integers between 0 and $N - 1$, so this is only possible if $M_0 - M = 0$. In other words, we have $M_0 = M$, and Alice gets the original message back!

Okay, that was a lot to take in at once. Maybe a small example would help. Let's say Alice chooses the primes 5 and 7 for p and q. They'd be way larger than this in practice, but let's keep the computations simple. Then $N = 5 \cdot 7 = 35$ and $\phi(N) = (5 - 1)(7 - 1) = 4 \cdot 6 = 24$. Now let's say Alice takes $e = 5$ as the encryption exponent (it's a quick check that $\gcd(e, \phi(N)) = \gcd(5, 24) = 1$).

Alice would go ahead and publish $N = 35$ and $e = 5$ as her public key. Then, to calculate her private key, she would do some back-substitution calculations with $e = 5$ and $\phi(N) = 24$ and find that $5 \cdot 5 + (-1) \cdot 24 = 1$. This means $d = 5$ and $f = -1$. So, as it turns out, Alice's private key d is also 5.

Now let's say Bob comes along and wants to send $M = 3$ as a plaintext. He goes and computes $M^e \pmod{N}$:

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

Bob sends $C = 33$ to Alice as a ciphertext. Alice now calculates $C^d \pmod{N}$:

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

We see that Alice retrieves $M_0 = 3$, the original plaintext. Magic!

At last, we've fulfilled our promise: we've demonstrated that public key cryptography can actually be put into practice. In fact, many online applications requiring cryptography still use RSA in some form or another. There are now many other public key cryptosystems to choose from, but it all started here.

At the same time, keep in mind that RSA may not be secure forever. If anyone figures out how to factor N quickly, they will be able to calculate the private key at will. But there's another way to attack RSA from a different angle. Suppose you've intercepted Bob's message to Alice. You know there is some number M representing the plaintext, and you have intercepted $M^e \pmod{N}$. The values of e and N are known, but you have to figure out M somehow. As mentioned earlier, no one knows how to do this quickly without knowing d, but that doesn't mean it's impossible. This particular problem is called the *discrete logarithm problem*, and an efficient solution to it will also break RSA.

All right, enough said! It's time for you to have fun with the RSA-related exercises in front of you... You can now attempt Questions 7–10.