#28: More Number Theory and RSA

May 9, 2009

This week, we'll explore the math behind RSA, an ingenious system for *public-key cryptography*. Never heard of public-key cryptography? Read on!

1 The key exchange problem and public-key cryptography

Suppose Ronnie wants to send a secret message to Nathan. The problem is that Ronnie is in Paris and Nathan is in São Paulo.[1] If Ronnie wants to use, say, a Vigenère cipher to encrypt his secret message, he needs to use a secret keyword—and Nathan needs to know the keyword, too. If they already agreed on a keyword beforehand (back when they were both living in Virginia), then there's no problem. But otherwise, it will be very difficult for them to agree on a secret keyword: presumably, the reason Ronnie wants to encrypt his message to Nathan in the first place is because people are eavesdropping on their communication! So sending a secret key to Nathan by normal means (in the mail, over the phone) is out of the question. It *might* be possible to securely transmit a secret key in some clever way, so that Ronnie can be reasonably sure that Nathan will get it and no one else will—but it will probably be a lot of work, and it will probably only work once.

Problem 1. Come up with a clever way for Ronnie to get a secret key to Nathan. Assume that they didn't talk about it beforehand, and that there are spies who are trying their best to eavesdrop on their communication. (So, for example, you can be sure that spies will be listening to any phone conversations and reading any mail.)

Okay, but what if Nathan wants to send individual secret messages to each of the 1,000 members of his fan club? It just isn't possible to go through all the difficult work of securely getting 1,000 different secret keys to each of them. So it seems like Nathan is out of luck.

[1] I could tell you what they are doing in those places, but then I'd have to kill you.

This is called the *key exchange problem*, and is a fundamental problem for any *symmetric cipher*. A symmetric cipher is one where you need the same secret key both for encoding and decoding secret messages. The secrecy of messages encoded using a symmetric cipher is only as good as the secrecy of the secret key, but if both people need to know the key, being able to agree on a key without other people finding out what it is can be a real problem.

However, there are other kinds of ciphers known as *asymmetric ciphers*. In an asymmetric cipher, *different* keys are used for encrypting and decrypting. Of course, this isn't necessarily any better—now people wishing to communicate need to agree on *two* keys that work together. But there's another sort of asymmetric cipher which avoids problem.

Here's how it works: with a symmetric cipher, Ronnie and Nathan have to agree on a secret key. With a *public-key* system, they don't need to agree on anything at all! Nathan uses a particular mathematical algorithm (which we'll talk about later) to choose *two* related keys: one, his *public key*, he posts on his website, his blog, in his email signature, for the whole world—even his enemies—to see. The other, his *private key*, he keeps absolutely secret, even from his most trusted friends. When Ronnie wants to send Nathan a secret message, he looks up Nathan's public key (which is easy, since it is very public—perhaps there is even a directory listing people's public keys, like a phone book) and uses it to encrypt the message. But now comes the interesting part about Nathan's keys: *the only way to decrypt the message is by using Nathan's private key*! The mathematical algorithm that Nathan used to choose his two keys guarantees this. So Ronnie can be sure that Nathan is the only person who can read the message (assuming that Nathan has kept his private key a secret). The result is that *anyone at all* can send messages *to* Nathan, but Nathan is the only one who can read them. If Nathan wants to send a secret message back to Ronnie, of course, Ronnie can generate his own pair of public/private keys, and Nathan can use Ronnie's public key to encrypt the message.

This sounds like a wonderful system, but it almost seems too good to be true. You might have some questions, such as:

1. How is this possible?
2. If you use a mathematical algorithm to choose a public and private keys, how come people can't just reverse the algorithm to compute your private key from the public one?

Excellent questions! For now, you can just take my word that it is possible. The first people to come up with a way of doing this, in 1977, were Ronald Rivest, Adi Shamir, and Leonard Adleman—and the system they invented is named after their initials, RSA. RSA (or something closely related to it) is still used today by computers all over the world for transmitting encrypted information (such as when you send your credit card number to Amazon). Before we see how RSA works, we'll have to (surprise!) learn a bit more math!

2 Bézout's Identity and the Extended Euclidean Algorithm

Recall that the gcd (greatest common divisor) of any two numbers is the largest natural number which evenly divides both of them. It turns out the gcd has an interesting property, known as *Bézout's Identity* (named for the French mathematician Étienne Bézout, who proved a more general version than the one shown here):

Bézout's identity

For any two natural numbers a and b, there are integers x and y for which

$$ax + by = \gcd(a, b).$$

For example, suppose $a = 10$ and $b = 6$. $\gcd(10, 6) = 2$, so Bézout's identity says that there should be integers x and y for which $10x + 6y = 2$.

Problem 2. Find integer values of x and y for which $10x + 6y = 2$. Note that x and y should be *integers*—that is, they *can* be negative (obviously, one of them will have to be, otherwise $10x + 6y$ would be too big!), but they *cannot* have any fractional part. For example, $x = -1.6, y = 3$ works, but does not count as a solution since x is not an integer.

Problem 3. For each pair of numbers, find the gcd, and then find integers x and y which satisfy Bézout's identity.

(a) $a = 3$, $b = 2$

(b) $a = 20$, $b = 5$

(c) $a = 19$, $b = 23$

At this point you're probably wondering if there's a faster way to find x and y than just guessing. Of course, I wouldn't tell you about it if there wasn't! It turns out that we can extend Euclid's Algorithm (which, remember, is for finding the gcd) so that it finds the gcd of two numbers and x and y in Bézout's identity at the same time. This is called (drumroll) the Extended Euclid Algorithm.[2]

Here's how it works. Suppose we are trying to find the gcd of 19 and 7. Remember how Euclid's algorithm works: we repeatedly divide the smaller number into the larger, and take the remainder. We could write the steps of Euclid's algorithm in a vertical column, like this:

$$\begin{array}{c} 19 \\ 7 \\ 5 \\ 2 \\ 1 \end{array}$$

That is, 19 divided by 7 gives a remainder of 5; 7 divided by 5 gives a remainder of 2, and 5 divided by 2 gives a remainder of 1; since 2 divided by 1 gives a remainder of 0, the gcd of 19 and 7 is 1. To extend this to also compute x and y from Bézout's identity, we make two more columns of numbers. We start like this:

n	x	y
19	1	0
7	0	1
5		
2		
1		

Note that I have labelled the columns n, x, and y, so it will be easier to talk about them. For each row, it should be the case that $19x + 7y = n$. This is obviously true for the first two rows: $19 \cdot 1 + 7 \cdot 0 = 19$, and $19 \cdot 0 + 7 \cdot 1 = 7$. We know that the n in the very last row is the gcd of 19 and 7, so if we can

[2] Bet you can never guess why.

only find a way to fill in the missing rows, the values of x and y in the very last row will be the numbers from Bézout's identity!

Here's how to compute the missing rows. Since 19 divided by 7 is 2 (with a remainder of 5), to get the third row we subtract 2 times the second row from the first row. That is, in the x column, $1 - 2 \cdot 0 = 1$, and in the y column, $0 - 2 \cdot 1 = -2$. Now the table looks like this:

n	x	y
19	1	0
7	0	1
5	1	-2
2		
1		

Let's check: is $19 \cdot 1 + (-2) \cdot 7 = 5$? Yup!

Now, 7 divided by 5 is 1 (remainder 2), so we subtract 1 times the third row from the second row: $0 - 1 \cdot 1 = -1$, and $1 - 1 \cdot (-2) = 3$. (You have to be really careful with your positives and negatives when doing this algorithm!) Now the table looks like this:

n	x	y
19	1	0
7	0	1
5	1	-2
2	-1	3
1		

And sure enough, $19 \cdot (-1) + 3 \cdot 7 = 21 - 19 = 2$. Finally, we fill in the last row: 2 goes into 5 twice, so we subtract twice the fourth row from the third row: $1 - 2 \cdot (-1) = 1 + 2 = 3$, and $-2 - 2 \cdot (3) = -8$.

n	x	y
19	1	0
7	0	1
5	1	-2
2	-1	3
1	3	-8

And now for the final check: $19 \cdot 3 + 7 \cdot (-8) = 57 - 56 = 1$! Sure enough, we have found Bézout's x and y. Now you try a few.

Problem 4. For each pair of numbers, use the Extended Euclidean Algorithm to find the gcd and Bézout's x and y.

(a) 10 and 19

(b) 28 and 16

(c) 37 and 84

3 Modular inverses

Okay, but why would anyone care about finding Bézout's x and y? It seems more like a curiosity than anything actually important.

Well, consider the following question. Suppose we are doing arithmetic modulo n, and we have some natural number j. Is there another natural number k for which $jk \equiv 1 \pmod{n}$? If such a k exists, it is called the *modular inverse* of $j \pmod{n}$. Modular inverses are important in a number of applications, and, as we will see, play a particularly important role in RSA.

Why are modular inverses interesting? Well, if we were using normal arithmetic instead of modular arithmetic, asking whether there is some *natural* number k for which $jk = 1$ would be a silly question. Is there a natural number k such that, say, $5k = 1$? Of course not! $5k$ will always be too big (unless $k = 0$ which doesn't work either). The only solution to the equation $5k = 1$ is $k = 1/5$, but that isn't a natural number. *But* if we are using modular arithmetic—on a number "circle" instead of a number line—then things wrap around, and this isn't a silly question anymore.

Problem 5. Find a natural number k such that $5k \equiv 1 \pmod{7}$.

Problem 6. Can you find a natural number k for which $5k \equiv 1 \pmod{10}$? If so, state a value of k that works; if not, explain why.

Suppose $\gcd(j, n) = 1$ and we want to find the modular inverse of $j \pmod{n}$. Using the Extended Euclidean Algorithm, we can find integers x and y for which $jx + ny = 1$. But consider this equation modulo n. ny is obviously divisible by n, so $ny \equiv 0 \pmod{n}$. But that means that

$$jx + ny \equiv jx + 0 \equiv jx \equiv 1 \pmod{n}.$$

So x is the modular inverse of j!

Problem 7. For each value of j and n, find the modular inverse of $j \pmod{n}$, or state that no modular inverse exists.

(a) $j = 5, n = 9$

(b) $j = 7, n = 21$

(c) $j = 2, n = 3$

(d) $j = 7, n = 22$

There's only one more thing you need to know about to understand how RSA works.

4 The totient function and Euler's Theorem

Euler's *totient function*, denoted by $\varphi(n)$ and sometimes also called the *phi function*, counts how many natural numbers less than n are *relatively prime* to n, that is, have a gcd of 1 with n. (You can make the symbol φ with `\varphi`.)

For example, $\varphi(8) = 4$, since there are four numbers less than 8 which are relatively prime to 8 (that is, share no common factors with 8): 1, 3, 5, and 7. As another example, $\varphi(14) = 6$, since 1,3,5,9,11,and 13 are relatively prime to 14.

Problem 8. Compute each of the following.

(a) $\varphi(9)$

(b) $\varphi(60)$

(c) $\varphi(17)$

(d) $\varphi(15)$

Problem 9. If p is a prime number, what is $\varphi(p)$?

Problem 10. It turns out that if a and b are any two relatively prime numbers, $\varphi(ab) = \varphi(a)\varphi(b)$. Using your solution to the previous problem, if p and q are prime numbers, what is $\varphi(pq)$?

In 1736, Euler proved a very famous (and important) theorem about φ, called Euler's Theorem: if a and n are relatively prime, then

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

I won't show you the proof (or make you do it!), but you can take my (well, really Euler's) word for it. This turns out to be one of the keys that makes RSA work.

5 RSA

Okay, we're finally ready to see how RSA works! Remember, the goal is to somehow generate two keys—a public key and a private key—and a method of using them to encrypt and decrypt messages, so that messages encrypted with the public key can be decrypted only with the private key. We also want to make sure that it's very difficult to figure out the private key if you only know the public key.

Start by picking two large prime numbers, p and q. We'll use $p = 5$ and $q = 7$. Of course, these are not actually very large, but that's just so that the example will be manageable. In practice you would choose primes that are hundreds or even thousands of digits long. (There are fast algorithms for testing whether a given number is prime, so you can just randomly choose, say, 1000-digit numbers until you find two that are prime.) Let n be the product of p and q; in our example, $n = 5 \cdot 7 = 35$. From now on we will be working modulo n.

Now compute $\varphi(n) = \varphi(pq) = (p-1)(q-1)$. In our example, $\varphi(35) = (5-1)(7-1) = 24$. Now pick any number e which is relatively prime to $\varphi(n)$; this will be the public key. In our example, we might pick, say, $e = 5$. (Note that it is important that e is relatively prime to $\varphi(n)$; in our example, 5 is relatively prime to 24 so this is okay.)

Now, here's why it's important that e is relatively prime to $\varphi(n)$: the next step is to compute the modular inverse of $e \pmod{\varphi(n)}$, using the Extended Euclidean algorithm. Call the modular inverse d. This will be the private key. In our example, the modular inverse of 5 (mod 24) is 5, since $5 \cdot 5 = 25 \equiv 1 \pmod{24}$. Whoops! This is a bad choice for e; we don't want the public and private keys to be the same! So we go back and choose a different value of e: let's try $e = 7$. The modular inverse of 7 (mod 24) is... 7. Hmm. Actually, it turns out that 24 is a very bad value for $\varphi(n)$: every number

relatively prime to 24 is its own modular inverse! So we have to go back to the drawing board and pick different values of p and q.[3]

This time, let's choose $p = 3$ and $q = 11$. Therefore $n = 33$, and $\varphi(n) = (3-1)(11-1) = 20$. We pick a value of e relatively prime to 20: let's try $e = 3$. The modular inverse of 3 (mod 20) is 7, since $3 \cdot 7 = 21 \equiv 1 \pmod{20}$. Oh good! The public key e and private key d are different.

Now, we make public the values of n and e. We must *keep secret* p, q, $\varphi(n)$, and d. Given only n and e, the only way for an evil spy to figure out d would be to factor n into $p \cdot q$; from there the evil spy could follow the exact same steps we did to compute d. But this is the key: *factoring is hard*! That is, factoring thousand-digit numbers takes a *Very Long Time*. You can do it—IF you have lots of supercomputers and are willing to wait a hundred or a thousand or a trillion years for the answer (depending on how big the number is you want to factor).[4] On the other hand, *multiplying* 1000-digit numbers is very easy. Your computer can multiply two 1000-digit numbers in a fraction of a second. We got to choose p and q and then multiply them to get n, leaving the evil spy with the much more difficult task of reversing the process!

Now, suppose someone wants to send us a message. They know the values of e and n. Here's what they do: they somehow convert their message into a number, or a sequence of numbers, each of which is smaller than n. It doesn't really matter how the conversion process works, as long as everyone knows what it is and it can be reversed. For example, suppose someone wanted to transmit the message "HI". Since $n = 33$, they can just use a number for each letter, just like you did last week. So, they could use 7 for H and 8 for I. In practice, n is much larger, and you would convert whole paragraphs at a time into one big number, but the idea is the same.

Now for each number a that they want to encode, they compute

$$b = a^e \pmod{n}.$$

[3] I didn't do this on purpose: I just randomly chose values for p and q and only realized later that it wasn't a good example. But I decided to leave it in, because actually, in some sense it *is* a good example. When using RSA, you often have to just keep trying different things until you find one that works well; some choices can accidentally generate insecure keys, as you have seen from this example. In practice, with a computer doing the work, this isn't a problem. Computers don't mind doing something over and over until they find something that works.

[4] Technically, no one has proved this for certain—there is a small chance that there actually is a fast way to factor large numbers, and it's only that no one has been clever enough to come up with it yet—but there are many very good reasons for believing that this is not true.

In this example, they would compute

$$7^3 \pmod{33}$$

and

$$8^3 \pmod{33}.$$

There are fast ways to perform this "modular exponentiation." For example, you can actually keep reducing mod 33 as you go:

$$7^3 = 7 \cdot 7 \cdot 7 \equiv 49 \cdot 7 \equiv 16 \cdot 7 \equiv 112 \equiv 13 \pmod{33}$$

and

$$8^3 = 8 \cdot 8 \cdot 8 \equiv 64 \cdot 8 \equiv (-2) \cdot 8 \equiv (-16) \equiv 17 \pmod{33}.$$

So the encrypted message they would transmit would be "13 17".

On our end, we perform the same process, but with d instead of e! That is, we compute

$$b^d \pmod{n}.$$

So, in this example we compute

$$13^7 \pmod{33}$$

and

$$17^7 \pmod{33}.$$

Computing $13^7 \pmod{33}$ might seem daunting, but it's actually not too bad. We can use the method of "repeated squaring." Note that $13^7 = 13^1 \cdot 13^2 \cdot 13^4$. So, we compute:

$$13^2 \equiv 13 \cdot 13 \equiv 169 \equiv 4 \pmod{33},$$

and therefore

$$13^4 \equiv 13^2 \cdot 13^2 \equiv 4 \cdot 4 \equiv 16 \pmod{33},$$

and putting it all together,

$$13^7 \equiv 13 \cdot 13^2 \cdot 13^4 \equiv 13 \cdot 4 \cdot 16 \equiv 52 \cdot 16 \equiv 19 \cdot 16 \equiv 304 \equiv 7 \pmod{33}.$$

Voilá! We have recovered the first number of the original message!

Problem 11. Compute $17^7 \pmod{33}$ using the method of repeated squaring, and show that it is equal to 8, the second number in the original message. Show your work!

Why does this work? Well, $(a^e)^d = a^{ed} = a^{1+k\varphi(n)}$ (the last equality is because we know $ed \equiv 1 \pmod{\varphi(n)}$; hence ed must be one more than some multiple of $\varphi(n)$). Then

$$a^{1+k\varphi(n)} = a \cdot (a^k)^{\varphi(n)} \equiv a \pmod{n},$$

because of Euler's theorem.[5]

Problem 12. Make your own public/private key pair! Choose primes p and q and use them to compute d and e. Report the values of p and q that you chose, as well as the values of n, $\varphi(n)$, e, and d. Choose a short message and encrypt it using your public key.

If you want to make your own public/private key pair in real life (using primes hundreds of digits long instead of primes like 11), I recommend using GnuPG, `http://www.gnupg.org/`.

6 Digital signatures

That's all for the problems on this week's assignment, but I thought you might be interested to know one more way that public/private key systems can be used. Suppose Nathan gets an encrypted message. He decrypts it using his private key. The decrypted message reads, "Dear Nathan, I hate you. —Ronnie." How does Nathan know the message is *really* from Ronnie? It could be from an evil spy who just wants Nathan to *think* that Ronnie hates him. After all, Nathan's public key is public, so *anyone* can encrypt a message to him! This is where *digital signatures* come in. It relies on the fact that the public/private key pairs generated by RSA can be used symmetrically: not only can things encrypted with the public key be decrypted with the private key, but the other way around works too: things encrypted with the private key can be decrypted with the public key.

Let's say Ronnie wants to send Nathan the message "HI" but he wants Nathan to be really sure that it is from him. First, Ronnie encrypts the message "HI" using his (Ronnie's) own *private* key. Suppose he gets "FLERG" as the encrypted message. Then he encrypts "This is a signed message from Ronnie: FLERG" using Nathan's public key; suppose he gets "ZORK".

[5] Actually, Euler's theorem would only apply if a and n are relatively prime, but we don't necessarily know that; however, it can be shown that in this special case this holds no matter what a is.

That seems strange, what's the point? Why would Ronnie encrypt something twice, and using his own private key of all things?

Well, when Nathan gets the encrypted message "ZORK", he first decrypts it using his private key. Remember, no one else can do this, so evil spies looking at the encrypted message "ZORK" won't even know it is from Ronnie. Nathan now gets the message "This is a signed message from Ronnie: FLERG". Here's the interesting part. Nathan now uses Ronnie's *public* key to decrypt "FLERG", and gets out the original message, "HI". Ronnie is the only person who knows his own private key, so if this works, then Nathan can be really sure that it was actually Ronnie who sent the message—no one else would be able to encrypt a message which can be decrypted using Ronnie's public key.

Now, back to the message Nathan got from "Ronnie" saying that Ronnie hates him. Nathan notices that it was not signed, so he can ignore it, or he can even send a message back to wherever the message came from, challenging "Ronnie" to prove that he is really Ronnie by digitally signing a message with his private key. Of course, assuming that Ronnie's private key is secret,[6] no one but Ronnie will actually be able to do this.

The actual details of digital signatures are a bit different in practice (for example, in practice no one actually encodes the entire message using their private key to sign it; they just compute a small number called a *hash* which is a function of the entire message, and then sign that and append it to the message before encrypting it with the recipient's public key), but the basic ideas are the same.

Neat, huh? And this is not just theoretical, either: this is really used all the time for transmitting sensitive information in a secure, authenticated way.

[6] Of course, it's always possible that a hacker stole Ronnie's private key, or that Ronnie gave his private key to someone he thought was trustworthy but wasn't, or that this person Nathan knows as "Ronnie" is actually an evil robot spy from the planet Zorkotron. Cryptography can't solve *all* Nathan's problems...

