Explain it like I'm 2: RSA

Sam Nolan

May 2, 2018

Coming up with your own encryption scheme is a difficult task. You have to transform data in such a way that it cannot be transformed back without a key, and also does not give away any information about the message.

RSA is one such algorithm. It's special because it is an asymmetric cryptosystem. Meaning that there is one key that encrypts the information, and the other key decrypt it.

It does this through modular arithmetic and powers. To explain RSA, we will have to look into the mathematical concepts that it is based off. If you don't feel great about maths, don't worry! We'll go through all the concepts.

Power laws

RSA makes use of this simple power law:

$$(a^b)^c = a^{bc}$$

Try it out for yourself! Punch in a few numbers.

If you're still not convinced, let's think about it for a second, $(a^b)^c$ means that there are c groups of (a^b) multiplied together, and a^b means that there is b copies of a that are multiplied together. The amount of a in this expression is bc, which shows that the expression above is true.

That's all for exponents that we'll need, now for the fun part!

Modular Arithmetic

Sounds horrifying! It isn't that bad, we use modular arithmetic every day.

The clock is a form of modular arithmetic. This is because if it is 11 o'clock and you wait for 2 hours, it will be 1 o'clock. This tick over is what makes time modular.

This is expressed as the following:

$$(11 + 2) \bmod 12 = 1$$

This $\bmod$ is an operation in the same way that addition and multiplication is. $\bmod 12$ gets rid of any spare 12's in the 13. An easy way to get rid of these

spare 12s is to divide the number by 12 and find the remainder. This operation is represented by a % in Java.

There is another notation that this can be expressed as.

$$11 + 2 \equiv 1 \mod 12$$

The $\equiv$ sign is a telltale sign that something is different. The $\equiv$ sign is read as "is congruent to", and the entire expression reads as "11 + 2 is congruent to 1 under mod 12". This is saying that if we are using modulo 12, then 11 + 2 and 1 are equivalent.

The $\mod 12$ is in a sense a marker telling you to treat the following relationship as modular. It's like saying, "If we are talking about clocks, then 11 + 2 is equivalent to 1" The $\mod 12$ part of this equation is stating something about the entire equation, and not just the 1.

If you don't like thinking about different systems, the statement is basically saying:

$$11 + 2 \mod 12 = 1 \mod 12$$

I like modular arithmetic! Imagine if there was only 12 numbers in existence and they just kept looping around! No more massive sums!

To top it off, when using modular arithmetic, there is no such thing as decimals, negatives or division! Only whole numbers.

Working with modular arithmetic algebreically works very similar to working with normal numbers. For instance, multiplying both sides of the equation by the same number will still create the same number. The only rule of thumb is that if it's an operation that can make a fraction or decimal out of a whole number, then it's banned from this point onwards.

What I mean is because dividing by 2 can create half a number when dividing by an odd number, it is therefore banned. That means:

- No surds
- No fractions or decimals
- No logarithms
- No division
- No negatives

Coprimes and our least favourite times table

Which was your least favourite times table? In my opinion, 7 wins the prize all the way. With seemingly no pattern between the numbers.

There are some times tables that have patterns with the last digit. For instance, our beloved 5 times tables, which can only end in 0 or 5 and makes our life much easier.

Let's look at our 7 times tables last digits, but with a new perspective. To use modular arithmetic, $\mod 10$ will just get the last digit of our number. This

is because $\mod 10$ gets rid of any excess 10s, which also includes any 100s (because 100s are made of 10s)

So, we have the following:

$$7 \times 1 \equiv 7 \mod 10 \tag{1}$$
$$7 \times 2 \equiv 4 \mod 10 \tag{2}$$
$$7 \times 3 \equiv 1 \mod 10 \tag{3}$$
$$7 \times 4 \equiv 8 \mod 10 \tag{4}$$
$$7 \times 5 \equiv 5 \mod 10 \tag{5}$$
$$7 \times 6 \equiv 2 \mod 10 \tag{6}$$
$$7 \times 7 \equiv 9 \mod 10 \tag{7}$$
$$7 \times 8 \equiv 6 \mod 10 \tag{8}$$
$$7 \times 9 \equiv 3 \mod 10 \tag{9}$$
$$7 \times 10 \equiv 0 \mod 10 \tag{10}$$

Yuck. But there's something cool about this, if you multiply 7 by a number under mod 10, **any** number as a last digit is possible. Count them above!

This feature is often used to shuffle lists, if we have a list of 10 elements, as element 1 can be put in place 7, 2 in 4 and so on. As a bit of a side note this technique is used by the online library of babel to shuffle their library.

This occurs when the mod we are using is **coprime** to the number we are multiplying. A number is coprime when there is no common factors between the numbers other than 1. Try it out! The numbers that are coprime to 10 are 1, 3, 7 and 9, all of which can have any digit as a last one.

This property comes in useful for our next part.

Modular Multiplicative Inverses

My god that sound scary. It's not really, it's similar to all the other inverses we have learnt in primary school.

What is an inverse? If I apply an operation to a number, the inverse of that operation reverses the original. For instance, the inverse of $+1$ is -1 because if you have any number, then add 1, and take 1, you get back the same number.

A multiplicative inverse is an operation that reverses the multiplication operation. In normal real numbers, if you multiply a number by x, to get the original number, you divide by x. This can also be represented by multiplying by $\frac{1}{x}$ or x^{-1}. With real numbers, $\frac{1}{x}$ and x^{-1} are exactly the same, just different ways of writing it.

However, this is where the "Modular" part comes in. Because as we noted before, in modular arithmetic, there is no such thing as division. So what are we to do?

Well, unlike real numbers, when you multiply under modular arithmetic, numbers can actually get smaller. For instance:

$$7 \times 2 \equiv 4 \mod 10$$

Here, by multiplying by 2, we go from 7 to 4. That's special to modular arithmetic. It means that we can get back to where we started by multiplying rather than dividing.

So say we are trying to find a modular multiplicative inverse of 7 for mod 10. This means that under mod 10, if I multiply a number by 7, and then multiply it by the modular multiplicative inverse of 7, we should get the original number.

This is stated mathematically as the following:

$$x \times 7 \times 7^{-1} \equiv x \mod 10$$

Note here my notation for the inverse of 7 as 7^{-1}. For those who are in to maths, this no longer means $\frac{1}{7}$, and instead means the modular multiplicative inverse of 7 for mod 10. This is what is called an abuse of notation, meaning that what is stated is not formally correct and can be misleading, but is a very common form of notation and is generally accepted.

Another thing to take into account is that 7^{-1} is dependent on mod 10. When using your online calculator and need to input the mod value, always choose the modulo value that appears in the same equation.

What's special about multiplicative inverses in real numbers is that if I multiply a number x by it's multiplicative inverse x^{-1}. Then $x \times x^{-1} = 1$.

This is also true in modular arithmetic, so $7 \times 7^{-1} \equiv 1 \mod 10$. So all we need to find the number that multiplied by 7 gives 1 under mod 10. This number happens to be 3. So we therefore say that 3 is the multiplicative inverse of 7, or with maths:

$$7^{-1} \equiv 3 \mod 10$$

This means that under mod 10, if we are to multiply a number by 7, and then by 3, we'll get the same number again. Try it out!

One interesting thing to note is that the inverse of 7 mod 10 is 3, and also the opposite is true, that is, the inverse of 3 mod 10 is 7. It is the same as with real numbers.

$$3^{-1} = 7 \mod 10$$
$$7^{-1} = 3 \mod 10$$

Finally, it is only possible to have a modular multiplicative inverse if the number that you are multiplying by is coprime to the modular. As we saw before, multiplying 7 under mod 10 can get any numbers 0-9, which means that it's guaranteed to be able to find one combination that gives a 1. This only works if the number is coprime to the mod. For instance, you won't find a number where:

$$8 \times x \equiv 1 \mod 10$$

Because 8 is not coprime to 10.

RSA, key generation

Now for the moment of truth, now that we've looked at all the previous topics, we'll be able to tackle RSA head on.

First of all, we'll look at key generation. Using 2 prime numbers p and q we are able to generate prime number n by multiplying them, and then we get $\phi(n)$, which we use to generate e and d.

For those who are in to maths, the $\phi(n)$ is actually *Euler's Totient Function.* Which, for when n is made up of 2 primes, $\phi(n)$ happens to be $(p-1)(q-1)$. You can look up the actual definition of $\phi(n)$ if you're interested.

So now that we have generated n and $\phi(n)$, we generate e and d. First we choose e to be coprime to $\phi(n)$. Why? Because it makes the modular multiplicative inverse of e possible. So we then choose d to be the modular multiplicative inverse of e.

The only property that we actually care about is that e and d are multiplicative modular inverses of each other over $\phi(n)$. That is:

$$d \times e \equiv 1 \mod \phi(n)$$

If you choose any two numbers for d and e that follow this rule, RSA will work. It might not be secure, but it will work. For instance, choosing 1 for both d and e would create a system where no encryption or decryption will ever happen. As raising the message to the power of 1 will do nothing to it. The encryption and decryption steps will work, but it will not be secure.

Because all we care about is that d and e are coprime, and d ande e are of equal status, we can generate these the other way around. Find a number d that is comprime to $\phi(n)$ and then find e. d and e are in a sense equivalent, of equal status. Why e is usually chosen before d in practice is outlined later, but in theory, they are interchangeable.

d is the modular multiplicative inverse of e and vice versa.

$$d^{-1} \equiv e \mod \phi(n)$$
$$e^{-1} \equiv d \mod \phi(n)$$

The Encryption and Decryption

Now to the heart of it! Here we show you why RSA seems to magically encrypt and decrypt numbers.

$$M^e \equiv C \mod n$$

Where M is the message and C is the cipher text.

The message is then decrypted by:

$$C^d \equiv M \mod n$$

To show this is true, I'll substitute the definition of C in the encryption step into the decryption step. So that:

$$(M^e)^d \equiv M \mod n$$

Now if we remember from our first section, this is the same as

$$M^{ed} \equiv M \mod n$$

Now, as e is the modular multiplicative inverse of d, as $d \times e = 1$, doing this operation cancels out the d and the e so

$$M^1 \equiv M \mod n$$

And there we have it! Taking a message to the power of e encrypts the message, and d simply cancels out e because it's the multiplicative inverse!

One way to compare d and e is to think of our ceaser shift. d is like adding 3 to the letter and e is subtracting 3. The metaphor is useful because it shows that it can also be done the other way and work. i.e. subtracting 3 to encrypt and then adding 3 do encrypt. As d and e are simply inverses of each other.

This is useful in digital signing and is simply the reverse of encryption, we'll cover it later.

$\phi(n)$ and the real story

OK, I lied to you, there's a flaw in my logic above. But the way that it's illustrated allows you to understand what you can do with RSA, helps you remember the encryption and decryption steps and gives you an idea of where digital signing comes from. The truth is a bit more ugly.

If you're not a maths person, skip this section, because it doesn't help to explain anything and doesn't give much more understanding of RSA. If you're a maths person, read on!

So, you may have noticed that $d \times e = 1$ is not true, and the real story is:

$$d \times e \equiv 1 \mod \phi(n)$$

And we are not doing a mod under $\phi(n)$, saying that $d \times e = 1$ is not true. So we come back here:

$$M^{ed} \equiv M \mod n$$

Given the definition of $d \times e$ above, it tells us that $d \times e$ is some multiple of $\phi(n)$ plus 1. That is, $d \times e$ can be written as:

$$1 + a\phi(n)$$

Where a is some arbitrary natural (whole) number. Let's substitute this in for ed

$$M^{1+a\phi(n)} \equiv M \mod n$$

If you know your exponent laws well, you can split up exponents that have a plus in it, so:

$$M^1 \times M^{a\phi(n)} \equiv M \mod n$$

Now I can get rid of the M on either side and we get:

$$M^{a\phi(n)} \equiv 1 \mod n$$

Now, If $M^{\phi(n)}$ is 1 under mod n then that would mean that any power of it would also be one. So if the following is true, the above is true

$$M^{\phi(n)} \equiv 1 \mod n$$

I'm going to stop here, as the above is a theorem by the name of *Euler's Theorem*, which is a bit more complicated. If you really want to, you can look at proofs for this on the internet. This is however true.

If you did make it all the way down here, we can say the the extra $a\phi(n)$ get cancelled out because of Euler's Theorem. That way we can still view it in the cancelling out manner as I outlined above.

Digital Signing

You can imagine that taking a number to the power of e times the exponent by e and then taking it the power of d divides the exponent by e and gets back to 1.

e is like the number 2, and d is like $\frac{1}{2}$. If you multiply by 2, you can reverse it by multiplying by $\frac{1}{2}$. But the reverse is also true, if you multiply by $\frac{1}{2}$, it can be reversed by multiplying by 2.

Encryption normally uses e to make a transformation that d undoes. However, the reverse is true, you can use d to change the information that e can undo.

This means that you can in a sense "decrypt" plaintext using the private key into the cipher text and then "encrypt" it back into the plaintext using the public key.

For this section I'll use the word encrypt to mean encrypt with private key and decrypt to mean decrypt with public key.

This means that data can be transformed in a way that everyone can unlock with the public key. What's the point of that?

It actually comes in very useful when we are signing documents. By being able to encrypt a small document (such as a hash) that is decryptable by everyone, you are able to prove that you were the one that encrypted the hash.

How this works is that you hash the document you are sending, encrypt the hash with your private key, and then others can read the decrypt the hash and verify that what you sent was what you wanted to send, and that it actually was sent by you.

Digital signing is simply encryption in reverse, using the private key to encrypt and the public key to decrypt

RSA in practice, e and d

In this section I'll talk about what is done to choose e and d in practice, and why e is chosen first.

It's a very simple reason really, it's because we cheat. We usually just choose e to be 65537 ($2^{16} + 1$), which is prime, so will therefore by coprime to almost any number.

Keep in mind that this doesn't mean the everyone's public key is the same, because the public key is also made of n, which will be different for every key.

What's also cool about $2^{16} + 1$ is that it's got a lot of binary 0s in it (10000000000000001_2), which actually makes exponentiation faster in practice.

That's the only reason why we generate e before d. Because we don't even really "generate" it, we just keep it constant.

I hope that made sense. If you have any feedback, or found any errors, these papers are now on Github! at `github.com/Hazelfire/EILI2`. You can submit an issue or just simply send me feedback at `s3723315@student.rmit.edu.au`.

If you have any other topics that you think need a decent explanation, send me an email and I'll try and write one up! These papers really help me with my understanding of the course material too. So I'd be more than happy to write a few more.