Section 6 Lecture 39 - The RSA algorithm

The RSA algorithm

The RSA algorithm uses *private* and *public* keys. The private keys are known only to the individual person and are never distributed in any form, not even to anyone you communicate with as a trusted friend – so when Alice and Bob communicate they do **not** know each other's private keys. Public keys are freely available and can be known by anyone – they use them to contact you.

I am going to go through a worked example just to explain *how* it all works and what is the point of it. We'll then show the proofs as to *why* it all works!

Setting up keys

Suppose Bob wants to receive a message from Alice. He needs to set up his private and public keys.

- He begins by choosing two, different, very large prime numbers p and q at random. He may get these from a list of primes, or using a computer, but they should be random and not connected or dependent on each other in some obvious way. In this example, I will use the *tiny* prime numbers $p = 61$ and $q = 53$ so you can see the idea – numbers actually used in the real world tend to be hundreds or thousands of digits long. He doesn't tell anyone his choice of numbers, so they are private.

- He then works out the product $n = p \times q$, which is his public key that anyone can know. So, he calculates $n = 61 \times 53 = 3233$. The idea is anyone can know this number, because just given the number (3233 in this case) they can't work out the factors, it just takes too long. OK, it's not too hard here (although remember from earlier that factorising 3233 would have taken you a little while), but the sort of numbers we use in reality, it is beyond all computing power to be able to factorise them.

So at this point everyone can know the Bob's public key n, but no-one can know p and q, the two numbers he chose to make it. So p and q are private.

- Next, he works out the Euler totient function $\varphi(n)$. Remember that this is the number of integers less than n which are coprime (have no factors in common) to n. Generally this is a very hard problem (even harder than finding factors, because you have so much to check), so it is totally infeasible for anyone to work it out. Well, apart from one person. We proved earlier that if n is the product of two primes p and q, then $\varphi(n) = (p - 1)(q - 1)$, and Bob can easily work this out as he (and only he) knows what p and q are. So in our case, $\varphi(n) = 60 \times 52 = 3120$

- Next, he chooses some number (public exponent) e – the only constraints are that it is smaller than $\varphi(n)$ and coprime to it. It doesn't need to actually be that large for decent security, let's just say we choose 17. This can be made public – again people will use it to communicate with him, along with n.

- Finally, Bob works out a number (private exponent) d such that $de \equiv 1 \pmod{\varphi(n)}$. Bob can do this easily even by trial and error, but no-one else can, because they don't know what $\varphi(n)$ is! Bob keeps this private to him. This will be his key for decrypting Alice's message when she sends it to him. In this case, you can show that $d = 2753$ is a suitable solution. Remember, only he knows this number.

Bob has now done all he needs to do to set up his keys. To quickly summarise:

- He created two very large primes p and q and set $n = pq$
- He worked out the Euler totient function $\varphi(n) = (p-1)(q-1)$
- He allowed n, together with a small prime e, to be made public
- He worked out his own key d which satisfies the equation $de \equiv 1 \pmod{\varphi(n)}$., to be kept private

Remember that anyone else only knows n and e. The numbers are so large that they cannot work out anything else, even with all the computing power in the world!

<u>Sending a message to Bob</u>

So, let's suppose Alice wants to send a message to Bob. Bob needs to set private and public keys to allow Alice to make a communication to him.

Note that we are going to assume that all messages sent are numbers. It is easy to convert any text message into a number (e.g. use 00-25 for letters, ASCII codes, lots of ways) so we'll assume messages are sent as numbers – they are probably sent as binary anyway along the communication channel!

Let's suppose Alice's message is m (technically, we need to make the constraint that it is smaller than n, but n should be so huge that this will never be a problem!) For my example I will take $m = 123$

She works out the number $c = m^e\ (mod\ n)$ – she can do this as knows e and n, Bob made them public!

In this case, we have $m = 123$, $e = 17$, $n = 3233$ and so she needs to work out $(123)^{17} \pmod{3233}$. Whilst I wouldn't expect you to do this in your head, it's straightforward for a computer – it works out to be 855.

This is her encrypted message, so she then sends this number $c = 855$ to Bob.

Now, I will just tell you for now (proof later) that in fact $m = c^d \pmod{n}$. So Bob works out $855^{2753} \pmod{3233}$ (in this example) and recovers the message 123.
So he has successfully decrypted this message. But why is this secure?

The point is that the only person who can work out c^d *(mod n)* is someone who knows what *d* is. And the only person who knows what *d* is, is Bob. No-one could have hacked it, because he never tried to distribute it. Even Alice didn't know it. So no-one else can work this value out and hence decrypt the message.

Just to summarise the whole procedure:

- It all starts with Bob choosing two prime numbers p and q which no-one else can know
- Using these, he publishes public keys n and e which anyone can see
- He calculates a private key d which, again, no-one else can know
- Anyone wishing to sends a message m to Bob sends $c = m^e$ (*mod n*)
- He decrypts the message by calculating $m = c^d$ (mod n)

This is a whole new way of looking at cryptography and secret messages. No longer, if someone wants a message sent to be readable only by them, do they have to try and agree a keyword between them.

It's not a perfect system as written here – as anyone can send a message to Bob, there's no way of verifying it actually came from Alice and not somebody pretending to be her. I'll discuss briefly in the lecture notes as to how this idea has developed into the system we use today.

Proof

We should prove that if $c = m^e$(mod n) then $m = c^d$(mod n)

To do this I am going to give two preliminary lemmas – a "lemma" is a smaller result that will be used to help in a main theorem. Both of these are well-known – the first is Fermat's Little Theorem (not to be confused with his famed Last Theorem)

Lemma 1 (Fermat's Little Theorem): If p is prime and r and s are positive integers with $r \equiv s$ (mod $p - 1$), then we have $a^r \equiv a^s$ (mod p) for any integer a.

The second is a special case of what is called the Chinese Remainder Theorem:

Lemma 2: For primes p and q, if we have $a \equiv b$ (mod p) and $a \equiv b$ (mod q), then we also have $a \equiv b$ (mod pq)

We will use these to help us prove our theorem. In the following, all the symbols p, q, n, d, e, m, c etc refer to them as defined in the RSA definition. You may wish to refresh your memory of these by looking back as you read the proof.

Theorem: In the RSA system, if $c = m^e$(mod n) then $m = c^d$ (mod n)

Proof:

Firstly note that since $c = m^e$ *(mod n)*, then $c^d = m^{de}$ *(mod n)* by just putting both sides to the power d.

Now, $de \equiv 1$ (mod $\varphi(n)$) $\equiv 1$ (mod $\varphi((p-1)(q-1)$) by the definitions of d and Theorem 1.

So by the definition of modulus, de is some multiple of $(p - 1)(q - 1)$, plus 1. Let's say that $de = k(p - 1)(q - 1) + 1$. Then obviously $de = (k(p - 1))(q - 1) + 1$ just adding in a couple of brackets, and so the remainder when you divide by $(q - 1)$ is still 1, and similarly for dividing by $(p - 1)$.

So we've shown that $de \equiv 1 \pmod{p - 1}$ and $de \equiv 1 \pmod{q - 1}$.

By Lemma 1, with $r = de$, $s = 1$, and $a = m$, since we have $de \equiv 1 \pmod{p - 1}$ we must have $m^{de} \equiv m \ (mod\ p)$, and similarly $m^{de} \equiv m \ (mod\ q)$

From Lemma 2, this means we have $m^{de} \equiv m \ (mod\ pq)$

So, going right back to the start, since $c^d = m^{de} \ (mod\ n)$ and $n = pq$, we have

$c^d = m^{de} \ (mod\ n)$	
$= m^{de} \ (mod\ pq)$	(since $n = pq$)
$= m \ (mod\ pq)$	(from the note about Lemma 2)
$= m \ (mod\ n)$	(since $n = pq$)

Just writing the two congruent terms the other way round (after all, if x is congruent to y then y is congruent to x, they have the same remainder) we finish with the required result $m = c^d \pmod{n}$

Such proofs aren't necessarily intuitive - when reading proofs, go through once and then back and look again and fill in the "gaps", trying to get the basic idea as to what is trying to be done and where things are coming from.