12th Lecture: 17/2

RSA Public Key Cryptography. If two people want to be able to communicate safely with one another, then they could meet and agree on an encryption scheme which is known to them and them alone. Thus, even if an adversary were to incercept a communication between them, he would not be able to decrypt it.

The problem which Public Key Cryptography (PCC) is meant to solve is to allow pairwise communication between a large number of strangers, who never actually meet to agree on pairwise encyption schemes. This problem became popular in the 1970s and a number of suggestions were proposed, of which the RSA scheme of Rivest, Shamir and Adelman is the best known and the one which has had the greatest longevity, still being in use today.

We now illustrate with an example the core features of how RSA works. We assume that all messages are basic, unpunctuated English text, hence that the only symbols appearing in a message are English letters (A-Z) and spaces. Suppose a user S (sender) wants to send the following message to user R (receiver):

I KNOW WHERE THE LOOT IS

STEP 1: All users of the system use the same formula for converting text to numbers. Firstly, they all use the same *block length* k, which means that the text is broken up into blocks of k symbols, each block is converted to a number and these numbers are sent sequentially. Note that, if the last block contains fewer than k symbols, then the convention is to fill it out with spaces.

In our example, we take $k = 5$ and show what to do with the first block "I KNO". There are 26 English letters (A-Z) and one space symbol, so a total of 27 text symbols. We convert each symbol to a base-27 digit, starting with $A := 0$, $B := 1$ etc up to $Z := 25$ and space $:= 26$. Thus the string "I KNO" is first converted to the base-27 string $(8,\, 26,\, 10,\, 13,\, 14)$. Each possible string uniquely represents an integer in $[0,\, 27^5)$ according to

$$(a_1,\, a_2,\, a_3,\, a_4,\, a_5) \mapsto \sum_{i=1}^{5} a_i 27^{i-1}.$$

Hence in our example, we set

$$M := 8 + 26 \cdot 27 + 10 \cdot 27^2 + 13 \cdot 27^3 + 14 \cdot 27^4 = 7704053.$$

M is called the *unencrypted message*.

STEP 2: Each user of the system chooses a so-called *public encryption key* which is made public, i.e.: is readily available to all users of the system. A public key consists of a pair $(n,\, e)$ of positive integers satisfying the following conditions:

(i) the number n is a product $n = pq$ of two distinct primes p and q, each of which is larger than the largest possible value of an unencrypted message M. Hence, in our example, each of p and q is larger than 27^5. Each user simply finds two such primes, using standard *prime testing algorithms*. There are three important points here:

(a) There are known algorithms (beyond the scope of this course to explore) for testing whether or not a given input is prime, which do not require the input to be factorised and which, in general, run much faster than even state-of-the-art factorising algorithms. Hence, even in realistic settings where p and q might be 1000-digit numbers, testing numbers of such size for primality is feasible.

(b) The density of primes, while it decays to zero as numbers go to infinity, does so slowly. The famous *Prime Number Theorem*, first proven in 1896, makes this precise. It says that $\pi(n) \sim n/\ln n$, where $\pi(n)$ is the number of primes up to n. Hence, even amongst 1000-digit numbers, about $1/\ln(10^{1000}) \approx 1/2200$ of these numbers is prime. So even if one randomly tests numbers for primality, one expects to find a prime after a couple of thousand tests. Thus, finding primes p and q is feasible.

(c) Since users don't communicate, it is theoretically possible for two users to choose exactly the same public key, which would mean (see Step 4 below) that they could also decode each others' messages to third parties. However, in the realistic setting where keys are a couple of thousand digits long, the probablility of two users choosing exactly the same primes p and q is already vanishingly small.

Let's return to our example and choose a key for the user R. We have $27^5/\ln 27^5 \approx 870729$, so there are approximately so many primes up to 27^5. I used the $p(N)$ function in `Wolfram Alpha` which returns the N:th prime and chose

$$p_R = p(900000) = 13834103, \quad q_R = p(1000000) = 15485863$$

and hence

$$n_R = p_R q_R = 214233023785889.$$

This is the first part of R's public key. Note that he will publish n_R, but *not* p_R and q_R. This is crucial !

The second part $e = e_R$ of his key must be a number satisfying

$$\text{GCD}(e,\, \phi(n)) = 1.$$

Now each user knows the factorisation of his own n, hence can compute $\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$. He can then test numbers e at random and run Euclid's algorithm to see if the GCD is 1, until he finds one that works. Euclid's algorithm runs very fast so that is good. Moreover, from (11.7) one can easily see that the fraction of integers which are relatively prime to a given input decays at worst logarithmically in the size of the input. So once again, for a realistic scenario where n has a couple of thousand digits, one expects to have to run Euclid's algorithm at worst a couple of thousand times before finding an e which works.

I used Wolfram Alpha and, for the above n_R, the first number e that I tested worked: $e_R = 102338518678121$. So now we have the full public key of user R:

$$(n_R,\, e_R) = (214233023785889,\, 102338518678121).$$

STEP 3: The sender S uses the public key of the receiver R to encrypt his message, according to the formula

$$M_e := M^{e_R} \pmod{n_R}.$$

This is an $a^b \pmod c$ type computation, which can be performed efficiently using repeated squaring. In our example I used Wolfram Alpha to perform the computation and

got

$$M_e \equiv 7704053^{102338518678121} \pmod{214233023785889} = 145976346420556.$$

M_e is the *encrypted message* and it is this that S sends to R.

STEP 4: The receiver has the tools to be able to *decrypt* M_e, i.e.: to recover M from M_e. He does this in two steps:

(a) He computes

$$d_R \equiv e_R^{-1} \pmod{\phi(n_R)}. \tag{12.1}$$

Since e_R was chosen to be relatively prime to $\phi(n_R)$, we know that d_R exists. And it can be computed efficiently using Euclid's algorithm. In our example, I computed

$$d_R = 113373918417413.$$

(b) I claim that

$$M \equiv M_e^{d_R} \pmod{n_R}. \tag{12.2}$$

Proof: By (12.1) we have $e_R d_R \equiv 1 \pmod{\phi(n_R)}$, hence $\phi(n_R)$ divides $e_R d_R - 1$, hence there is some positive integer t such that $e_R d_R = 1 + t \cdot \phi(n_R)$. Therefore,

$$M_e^{d_R} \equiv (M^{e_R})^{d_R} = M^{1+t\cdot\phi(n_R)} = (M^{\phi(n_R)})^t \cdot M \pmod{n_R}. \tag{12.3}$$

Recall that n_R was chosen to be a product of two primes, each of which was larger than the largest possible value of an unencrypted message M. This guarantees that GCD$(n_R,\, M) = 1$ and hence Theorem 11.7 says that $M^{\phi(n_R)} \equiv 1 \pmod{n_R}$. Substituting this into (12.3) establishes our claim.

Thus the receiver just needs to perform the $(a^b \pmod c)$-type computation in (12.2) in order to recover M.

This brings us to the crucial point about the RSA cryptoystem:

An eavesdropper could recover M if and only if they had knowledge of $\phi(n_R)$, because that is exactly what is needed to perform part (a) of the decryption.

But having knowledge of $\phi(n_R)$ is equivalent to having knowledge of p_R and q_R, see Homework 2, Exercise 7. Hence, the *security* of RSA relies on the difficulty of factorising large numbers.

STEP 5: Finally, the receiver needs to be able to convert an unencrypted message M back to text. This is easy and can be performed by anyone. It merely requires one to write a given decimal number in base 27, for which the standard procedure is repeated division and taking of remainders. In our example,

$$\begin{aligned} 7704053 &= 285335 \cdot 27 + 8, \\ 285335 &= 10567 \cdot 27 + 26, \\ 10567 &= 391 \cdot 27 + 10, \\ 391 &= 14 \cdot 27 + 13, \\ 14 &= 0 \cdot 27 + 14. \end{aligned}$$

The sequence of remainders gives the base-27 representation of M, hence we get back $(8,\ 26,\ 10,\ 13,\ 14) =$ I KNO.