

## 15. RSA Cryptosystems.

These are named after the three Americans, Rivest, Shamir and Adleman who first published details in 1977. In fact, they had been discovered in 1973 by Clifford Cocks. He worked for GCHQ – part of the UK security services – so his discovery was not made public until 1997.

We begin by recalling some ideas from Chapter 1. A *cryptosystem* consists of an *encryption* algorithm and a *decryption* algorithm. Applying the encryption algorithm to a string of objects – the *plaintext* – produces a new string – the *ciphertext*. Applying the decryption algorithm to the ciphertext returns the original message. *Breaking* a cryptosystem means recovering plaintext from ciphertext *without prior knowledge of the decryption algorithm*.

The simplest example is a *substitution cryptosystem*. In this, each occurrence of a given character in plaintext is replaced by another character chosen from the same alphabet. Such cryptosystems can be broken by *frequency analysis* once enough ciphertext is available. This depends on the fact that, in any language, some characters occur more frequently than others. For example, in English text, the letter “e” occurs 12.702% of the time. The next most frequent is “t”, which occurs 9.056% of the time. Given a few pages of ciphertext, we can be virtually certain to identify the characters representing “e” and “t”. For some groups of letters, such as a,o,n,r,l,s, the expected frequencies are so close that some degree of trial and error is needed to identify their representations. The knowledge that we have a plaintext *in English* helps.

Our new systems are also known as *public key* cryptosystems. This is because the encryption algorithm (the *key* which “locks” the message) can be made freely available, i.e. *public*. Unlike most other cryptosystems, a knowledge of the encryption algorithm does *not* lead directly to the decryption algorithm.

In Chapter 1, we also looked at plaintext as a string of pairs or triples of characters. More generally, we can look at the message as a string of *blocks*, each of  $k$  characters. As we increase  $k$ , the *number* of possible blocks increases quite rapidly. The expected frequencies still vary, but the differences become less and less significant. Thus, we need more and more text to compute and compare the relative frequencies. Some calculation shows that the amount of text required to find the relative frequencies for blocks of more than six characters is much larger than the amount available in the entire world. In our cryptosystems, we typically use blocks with length over 100. Thus our systems will be immune from attack by frequency analysis.

### A typical RSA cryptosystem

Suppose we have an English text. We regard it as made up of *blocks* of 125 characters. If the original message does not consist of a multiple of 125 characters, we add random characters at the end to achieve a multiple of 125. Using the ASCII code, each character is represented by an 8-bit binary string. Thus a block is represented by a 1000-bit string. This in turn corresponds to an integer of around 300 decimal digits.

We now apply encryption *to the blocks*. Since there are about  $10^{300}$  possible blocks, it is quite impossible to give an encryption *table*. Here the encryption algorithm is described as a function mapping integers to integers. The decryption algorithm will have a similar form.

We choose primes  $p, q$  with  $m = pq$  having at least 300 decimal digits.  
 We then choose an integer  $r$  such that  $\gcd(r, t) = 1$ , where  $t = (p-1)(q-1)$ .

### The RSA encryption algorithm

The plaintext consists of a sequence of blocks. Each of which encodes as an integer in the range  $[0, m-1]$ . Given such an integer  $a$ , the Division Theorem proves the existence of a unique integer  $b$  such that

- (1)  $b \equiv a^r \pmod{m}$ , and
- (2)  $0 \leq b < m$ .

To send the block in the plaintext given by the integer  $a$ , we transmit the integer  $b$ .

With this algorithm, the ciphertext consists of a string of 1000-bit binary numbers. Each corresponds to an integer in the range  $[0, m-1]$ . We decrypt each of these in turn.

Recall that we chose  $r$  with  $\gcd(r, t) = 1$ . It follows at once that we can find integers  $s$  such that  $rs \equiv 1 \pmod{t}$ . We now choose one such value as  $s$ .

### The RSA decryption algorithm

The ciphertext consists of a sequence of blocks. Each of which encodes as an integer in the range  $[0, m-1]$ . Given such an integer  $b$ , the Division Theorem proves the existence of a unique integer  $a$  such that

- (3)  $a \equiv b^s \pmod{m}$ , and
- (4)  $0 \leq a < m$ .

The block in the ciphertext given by the integer  $b$  decrypts as the integer  $a$ .

Of course, it is far from obvious that these processes have the necessary property to be the encryption and decryption algorithms of a cryptosystem. For this, we require that, if the integer  $a$  encrypts as the integer  $b$ , then  $b$  decrypts as  $a$ . This must hold for any  $a$  in the range  $[0, m-1]$ .

The mathematics behind this can be summed up as follows.

### Theorem

Suppose that  $p$  and  $q$  are distinct primes, and that  $m = pq$ ,  $t = (p-1)(q-1)$ .

Suppose also that  $r$  is an integer with  $\gcd(r, t) = 1$ . Then

- (1) there is an integer  $s$  with  $rs \equiv 1 \pmod{m}$ , and
- (2) for an integer  $a$ ,  $b^r \equiv a \pmod{m} \Leftrightarrow a \equiv b^s \pmod{m}$ .

**Proof** (1) The fact that  $\gcd(r, t) = 1$  guarantees the existence of a suitable integer  $s$ . Suppose that  $rs = 1 + kt$  for some integer  $k$ .

(2) As  $m = pq$ , we work separately modulo  $p$  and modulo  $q$ .

If  $p \nmid a$ , then Fermat's Theorem gives  $a^{p-1} \equiv 1 \pmod{p}$ . Also  $t = (p-1)(q-1)$ .

$$b^s \equiv (a^r)^s \equiv a^{rs} \equiv a^{1+kt} \equiv a \cdot (a^t)^k \equiv a \cdot (a^{(p-1)(q-1)})^k \equiv a \cdot (a^{p-1})^{k(q-1)} \equiv a \cdot 1^{k(q-1)} \equiv a \pmod{p}.$$

If  $p \mid a$ , then  $b^s \equiv a^{rs} \equiv 0 \equiv a \pmod{p}$ .

Thus, for any integer  $a$ ,  $b^s \equiv a \pmod{p}$ . In other words  $p \mid (b^s - a)$ .

Similarly, replacing  $p$  by  $q$  in this argument, we get  $q \mid (b^s - a)$ .

As  $p$  and  $q$  are distinct,  $m = pq \mid (b^s - a)$ , i.e.  $a \equiv b^s \pmod{m}$ .

### Corollary

The RSA encryption and decryption algorithms together give a cryptosystem.

### Proof

Suppose that the integer  $a$  is in the range  $[0, m-1]$ . The encryption algorithm encrypts  $a$  as the integer  $b$ , with  $b \equiv a^r \pmod{m}$ . The decryption algorithm then decrypts  $b$  as the integer  $c$  with  $c \equiv b^s \pmod{m}$  and  $0 \leq c < m$ . Now the theorem shows that we must have  $c \equiv a \pmod{m}$  as each is congruent to  $b^s$  modulo  $m$ . But  $a$  and  $c$  are integers in the range  $[0, m-1]$ , so we have  $c = a$ , as required.

Constructing RSA cryptosystems depends on finding *primes*  $p$  and  $q$  of given size, and an integer  $r$  with  $\gcd(r, (p-1)(q-1)) = 1$ . We must now consider the problem of finding suitable integers.

### Identifying prime numbers.

*With present knowledge, there is no way of proving that a very large integer is prime. The best we can do is to show that the chance that the integer is composite is extremely small. In Chapter 13, we met Fermat's Theorem, which states that*

$$\text{If } p \text{ is prime and } p \nmid a, \text{ then } a^{p-1} \equiv 1 \pmod{p}.$$

The congruence *may* hold with *composite*  $p$ , but such examples are very rare. For fixed  $a$ , we define

**Test A**      If  $0 < a < p$ , then  $a^{p-1} \equiv 1 \pmod{p}$ .

Obviously, any *prime* passed the test. To improve our chances of correctly identifying primes, we add another test. In Chapter 14, we showed that

$$\text{If } p \text{ is prime, and } x^2 \equiv 1 \pmod{p}, \text{ then } x \equiv \pm 1 \pmod{p}.$$

We also indicated that, if  $p$  is composite, then there are always several other possible values of  $x$  modulo  $p$ . If we replace  $x$  by  $a^k$ , we obtain

**Test B**      If  $a^{2k} \equiv 1 \pmod{p}$ , then  $a^k \equiv \pm 1 \pmod{p}$ .

Note that, if we are looking for primes  $p$ , we need only examine odd  $p$ , so  $p = 2k+1$ . If  $p$  passes Test A, then  $a^{p-1} \equiv 1 \pmod{p}$ . This gives us the opportunity to try Test B. Although the proof is complicated, we have

**Theorem** For fixed  $a$ , if  $p > a$  passes tests A and B, then the chance that  $p$  is *composite* is less than 1 in 4.

Experimental evidence suggests that the probability is *very much* less than 1 in 4.

**Corollary** If an integer  $p$  passes tests A and B for 24 different values of  $a$ , then the chance that  $p$  is composite is less than 1 in  $2.5 \times 10^{14}$ .

**Outline of proof** for a single  $a$ , the chance is less than 1 in 4. When we combine tests, we multiply odds. Thus the chance that  $p$  passes the tests for 24 different values of  $a$  is less than 1 in  $4^{24}$ . The latter is approximately  $2.5 \times 10^{14}$ .

**Note** This is as likely as winning the lottery *two weeks in a row*.

### Implementing the tests

Each of the above tests involves evaluating an integer power  $a^n$ , where  $n$  is typically an integer of over 100 digits. Direct evaluation needs  $n-1$  multiplications. When  $n$  is very large, this is impractical. However, in Chapter 3, we saw that  $a^n$  could be evaluated using just  $2N-1$  multiplications, where  $N$  is the number of digits in  $n$ . Even for  $N = 300$ , this is quite feasible, unless the numbers involved become too large to be stored. As an example, suppose that  $a = 10$ ,  $n = 10^N$ ,  $N = 300$ . Then  $a^n$  has  $10^{300}$  digits. This could *not* be stored on any *real* computer.

Fortunately, we actually want to find the *remainder* when  $a^n$  is divided by the integer  $m$ . Typically  $m$  will have up to 300 digits. Now the Division Theorem shows that the remainder  $r$  has  $0 \leq r < m$ . If we reduce modulo  $m$  *after each multiplication* in the course of the calculation, then we never have to deal with an integer of more than 600 digits - the product of two integers of at most 300 digits. Since we have to test for only 24 values of  $a$ , the entire process takes only a few seconds.

To construct a RSA cryptosystem, we need two primes of chosen sizes - see below. We now need to know how many integers we may have to examine in order to find a prime. Once again, we cannot give an absolute guarantee, but we can ensure that the chance of failure is acceptably small. The estimate depends on a classical result in Number Theory. This is quite hard to prove.

### The Prime Number Theorem.

The probability that a random  $N$ -digit integer is prime is approximately 1 in  $N \log_e 10$ .

Using the ideas of combining probabilities, we derive the

### Corollary.

The chances that  $42N$  randomly chosen odd  $N$ -digits are all composite is less than 1 in  $2 \times 10^{14}$ .

**Note.** We chose  $42N$  so the odds were much the same as in the previous section. This is "virtual certainty" in anyone's language.

From the Corollary, finding a 200-digit prime may involve testing  $42 \times 200 = 8400$  integers. As noted above, testing each integer takes only a few seconds. Thus, finding a prime takes a few hours at most. Finding a second, 100-digit, prime is rather quicker. Thus, finding a suitable pair  $\{p, q\}$  of primes is relatively quick.

To continue the construction of an RSA cryptosystem, we need an integer  $r$  which is prime to  $t = (p-1)(q-1)$ . One way is to choose as  $r$  a *prime* greater than  $p$  and  $q$ . It is not immediately clear that this is suitable. We outline the

### **Justification.**

Any common factor of  $r$  and  $t$  which exceeds 1 must be  $r$ , since  $r$  is prime. But each prime factor of  $t$  divides either  $p-1$  or  $q-1$ . Once we have chosen  $r$  greater than  $p, q$ , it cannot divide  $t$ . Thus  $\gcd(r, t) = 1$ .

Finding a suitable prime  $r$  is similar to finding the primes  $p$  and  $q$ . We just start the search at  $m = pq$ . Again, this can be done quickly.

To complete the cryptosystem, we need to find an integer  $s$  with  $rs \equiv 1 \pmod{t}$ . This can be done using the Euclidean Algorithm as the congruence is equivalent to the equation

$$rs + tu = 1, u, s \in \mathbf{Z}.$$

Of course, the Euclidean Algorithm gives a result *very* quickly.

Summing up, we can find integers  $p, q, r, s$  which give an RSA cryptosystem. Provided we choose  $p, q$  with  $m = pq$  of at least 300 digits, we can use 1000-bit blocks. One obvious way to achieve this is by choosing  $p$  of 100 digits, and  $q$  of 200 digits. The reason for choosing primes of different length is explained below.

### **Implementing an RSA cryptosystem.**

We encrypt the message block by block. Suppose that a block corresponds to the (1000-bit) integer  $a$ . This is encrypted as the integer  $b$ , where

$$\begin{aligned} (1) \quad & b \equiv a^r \pmod{m}, \text{ and} \\ (2) \quad & 0 \leq b < m. \end{aligned}$$

As in the section “identifying prime numbers”, evaluating  $b$  takes seconds at worst. Thus the message can be transmitted as rapidly as it can be typed in! Of course, decryption is a similar process, so is equally quick.

Overall, RSA cryptosystems can be constructed quickly – in hours at most. This need be done only *once*. After this, encryption and decryption can be done in real time.

### **The security of RSA cryptosystems.**

As indicated earlier, the values of  $m$  and  $r$  are made *public*. Then *anyone* can encrypt messages.

The value of  $s$ , and those of  $p$  and  $q$  are kept *secret* – known only to the owner of the cryptosystem. They are *not* needed for encryption.

We noted earlier that our 1000-bit cryptosystems are completely immune to attack by frequency analysis.

As far as is (publicly) known, the only way to break these cryptosystems is by finding the factors  $p$  and  $q$  of the published  $m$ . Factorizing integers is notoriously difficult. Even the information that  $m$  has precisely two prime factors does not appear to help. The only quick methods in the public domain work only when  $p, q$  or  $q-p$  are relatively “small”. In this context, integers of up to 10 digits are certainly small. To be very sure that these methods

fail, we chose  $p$  of 100 digits, and  $q$  of 200 digits. Then  $q-p$  is also of 200 digits. Thus, none of  $p$ ,  $q$ ,  $q-p$  are in any sense small.

### **Conclusion.**

We have seen that Number Theory, and, in particular, the concept of congruence and Fermat's Theorem, can be used to construct secure RSA cryptosystems. These are particularly useful since they are of the "public key" variety. Additionally, they can be implemented in real time on devices as simple as mobile phones.