This note is partly based on Section 1.4 of "Algorithms," by S. Dasgupta, C. Papadimitriou and U. Vazirani, McGraw-Hill, 2007.
An online draft of the book is available at http://www.cs.berkeley.edu/ vazirani/algorithms.html

Public Key Cryptography

In this note, we discuss a very nice and important application of modular arithmetic: the *RSA public-key cryptosystem*, named after its inventors Ronald Rivest, Adi Shamir and Leonard Adleman.

The basic setting for cryptography is typically described via a cast of three characters: Alice and Bob, who with to communicate confidentially over some (insecure) link, and Eve, an eavesdropper who is listening in and trying to discover what they are saying. Let's assume that Alice wants to transmit a message x (written in binary) to Bob. She will apply her *encryption function* E to x and send the encrypted message $E(x)$ (also called the *cyphertext*) over the link; Bob, upon receipt of $E(x)$, will then apply his *decryption function* D to it and thus recover the original message (also called the *plaintext*): i.e., $D(E(x)) = x$.

Since the link is insecure, Alice and Bob have to assume that Eve may get hold of $E(x)$. (Think of Eve as being a "sniffer" on the network.) Thus ideally we would like to know that the encryption function E is chosen so that just knowing $E(x)$ (without knowing the decryption function D) doesn't allow one to discover anything about the original message x.

For centuries cryptography was based on what are now called *private-key* protocols. In such a scheme, Alice and Bob meet beforehand and together choose a secret codebook, with which they encrypt all future correspondence between them. (This codebook plays the role of the functions E and D above.) Eve's only hope then is to collect some encrypted messages and use them to at least partially figure out the codebook.

Public-key schemes, such as RSA, are significantly more subtle and tricky: they allow Alice to send Bob a message without ever having met him before! This almost sounds impossible, because in this scenario there is a symmetry between Bob and Eve: why should Bob have any advantage over Eve in terms of being able to understand Alice's message? The central idea between the RSA cryptosystem is that Bob is able to implement a *digital lock*, to which only he has the key. Now by making this digital lock public, he gives Alice (or, indeed, anybody else) a way to send him a secure message which only he can open.

Here is how the digital lock is implemented in the RSA scheme. Each person has a *public key* known to the whole world, and a *private key* known only to him- or herself. When Alice wants to send a message x to Bob, she encodes it using Bob's public key. Bob then decrypts it using his private key, thus retrieving x. Eve is welcome to see as many encrypted messages for Bob as she likes, but she will not be able to decode them (under certain simple assumptions explained below).

The RSA scheme is based heavily on modular arithmetic. Let p and q be two large primes (typically having, say, 512 bits each), and let $N = pq$. We will think of the plaintext, the message x that Alice wishes to send to Bob, as a number modulo N. (Larger messages can always be broken into smaller pieces and sent separately.) The cyphertext, the encrypted message $y = E(x)$, will also be a number modulo N.

Also, let e be any number that is relatively prime to $(p-1)(q-1)$. (Typically e is a small value such as 3.)

Then Bob's *public key* is the pair of numbers (N, e). This pair is published to the whole world. (Note, however, that the numbers p and q are *not* public; this point is crucial and we will return to it below.)

What is Bob's private key? This will be the number d, which is the *inverse* of e mod $(p-1)(q-1)$. (This inverse is guaranteed to exist because e and $(p-1)(q-1)$ are coprime.)

We are now in a position to describe the encryption and decryption functions:

- **[Encryption]:** When Alice wants to send a message x (assumed to be an integer mod N) to Bob, she computes the value $E(x) = x^e \bmod N$ and sends this to Bob.
- **[Decryption]:** Upon receiving the value $y = E(x)$, Bob computes $D(y) = y^d \bmod N$; this will be equal to the original message x.

Example: Let $p = 5$, $q = 11$, and $N = pq = 55$. (In practice, p and q would be much larger.) Then we can choose $e = 3$, which is relatively prime to $(p-1)(q-1) = 40$. Thus Bob's public key is $(55, 3)$. His private key is $d = 3^{-1} \bmod 40 = 27$. For any message x that Alice (or anybody else) wishes to send to Bob, the encryption of x is $y = x^3 \bmod 55$, and the decryption of y is $x = y^{27} \bmod 55$. So, for example, if the message is $x = 13$, then the encryption is $y = 13^3 = 52 \bmod 55$, and this is decrypted as $13 = 52^{27} \bmod 55$.

To better understand the properties that the encryption function $E(x)$ and the decryption function $D(y)$ must satisfy, we make a brief digression below:

Bijections

Consider a function (or mapping) f that maps elements of a set A (called the *domain* of f) to elements of set B (called the *range* of f). For each element $x \in A$ ("input"), f must specify one element $f(x) \in B$ ("output"). Recall that we write this as $f : A \to B$. We say that f is a *bijection* if every element $a \in A$ has a unique *image* $b = f(a) \in B$, and every element $b \in B$ has a unique *pre-image* $a \in A$: $f(a) = b$.

f is a *one-to-one function* (or an *injection*) if f maps distinct inputs to distinct outputs. More rigorously, f is one-to-one if the following holds: $x \neq y \Rightarrow f(x) \neq f(y)$.

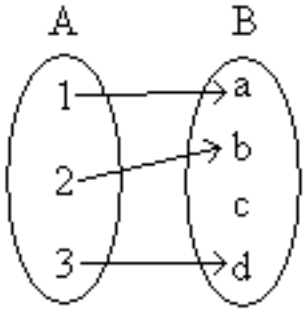

Figure 1: One-to-one

The next property we are interested in is functions that are *onto* (or *surjective*). A function that is onto essentially "hits" every element in the range (i.e., each element in the range has at least one pre-image). More precisely, a function f is onto if the following holds: $\forall y \; \exists x : f(x) = y$. Here are some examples to help visualize one-to-one and onto functions:

Note that according to our definition a function is a bijection iff it is both one-to-one and onto.

Lemma: If there is a function $g : B \to A$, and such that $\forall x \in A g(f(x)) = x$, then f must be one-to-one.

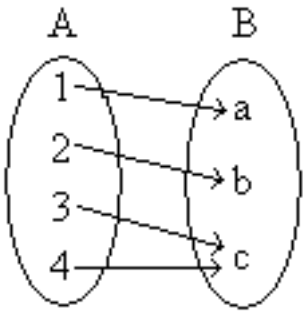

Figure 2: Onto

Proof: Suppose $f(x) = f(x')$. Then $x = g(f(x)) = g(f(x')) = x'$.

Moreover, in the case that f maps a finite set to itself, if f to be one-to-one then it is necessarily onto:

Lemma: If $f : A \to A$ is one-to-one and A is a finite set, then f is a bijection.

Proof: Let $|A| = n$. Since f is one-to-one, there must be n elements in the range of f. But that means that every element in A must be hit.

Let us return to our encryption-decryption pair $E(x)$ and $D(y)$. Since the domain and range are finite (numbers mod N), and since $D(E(x)) = x$, it follows that both $E(x)$ and $D(y)$ are bijections. Moreover, they are bijections with the special property that $E(x)$ can be efficiently computed with knowledge of only x and N. On the other hand, $D(y)$ is hard to compute unless you have further knowledge of a secret key $d = e^{-1} mod\ (p-1)(q-1)$. Let us now see why it is the case that $D(E(x)) = x$.

Theorem 6.1: Under the above definitions of the encryption and decryption functions E and D, we have $D(E(x)) = x \bmod N$ for every possible message $x \in \{0, 1, \ldots, N-1\}$.

The proof of this theorem makes use of a beautiful theorem from number theory known as *Fermat's Little Theorem*, which is the following:

Theorem 6.2: **[Fermat's Little Theorem]** For any prime p and any $a \in \{1, 2, \ldots, p-1\}$, we have $a^{p-1} = 1 \bmod p$.

Proof: Let S be the nonzero integers modulo p; that is, $S = \{1, 2, \ldots, p-1\}$. Here's the crucial observation: the effect of multiplying these numbers by a (modulo p) is simply to permute them. For instance, here's a picture of the case $a = 3, p = 7$:

Let's carry this example a bit further. From the picture, we can conclude

$$\{1, 2, \ldots, 6\} \ = \ \{3 \cdot 1 \bmod 7,\ 3 \cdot 2 \bmod 7, \ldots,\ 3 \cdot 6 \bmod 7\}.$$

Multiplying all the numbers in each representation then gives $6! \equiv 3^6 \cdot 6! \pmod 7$, and dividing by 6! we get $3^6 \equiv 1 \pmod 7$, exactly the result we wanted in the case $a = 3, p = 7$.

Now let's generalize this argument to other values of a and p, with $S = \{1, 2, \ldots, p-1\}$. We'll prove that when the elements of S are multiplied by a modulo p, the resulting numbers are all distinct and nonzero. And since they lie in the range $[1, p-1]$, they must simply be a permutation of S.

The numbers $a \cdot i \bmod p$ are distinct because if $a \cdot i \equiv a \cdot j \pmod p$, then dividing both sides by a gives $i \equiv j \pmod p$. They are nonzero because $a \cdot i \equiv 0$ similarly implies $i \equiv 0$. (And we *can* divide by a, because by assumption it is nonzero and therefore relatively prime to p.)

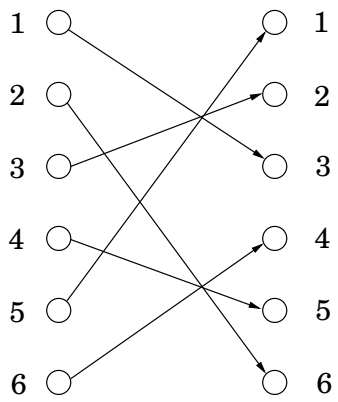

Figure 3: Multiplication by (3 mod 7)

We now have two ways to write set S:

$$S = \{1, 2, \ldots, p-1\} = \{a \cdot 1 \bmod p,\ a \cdot 2 \bmod p, \ldots,\ a \cdot (p-1) \bmod p\}.$$

We can multiply together its elements in each of these representations to get

$$(p-1)! \equiv a^{p-1} \cdot (p-1)! \pmod{p}.$$

Dividing by $(p-1)!$ (which we can do because it is relatively prime to p, since p is assumed prime) then gives the theorem. □

Let us return to proving that $D(E(x)) = x$:

Proof of Theorem 6.1: To prove the statement, we have to show that

$$(x^e)^d = x \bmod N \qquad \text{for every } x \in \{0, 1, \ldots, N-1\}. \tag{1}$$

Let's consider the exponent, which is ed. By definition of d, we know that $ed = 1 \bmod (p-1)(q-1)$; hence we can write $ed = 1 + k(p-1)(q-1)$ for some integer k, and therefore

$$x^{ed} - x = x^{1+k(p-1)(q-1)} - x = x(x^{k(p-1)(q-1)} - 1). \tag{2}$$

Looking back at equation (1), our goal is to show that this last expression in equation (2) is equal to 0 mod N for every x.

Now we claim that the expression $x(x^{k(p-1)(q-1)} - 1)$ in (2) is divisible by p. To see this, we consider two cases:

Case 1: *x is not a multiple of p.* In this case, since $x \neq 0 \bmod p$, we can use Fermat's Little Theorem to deduce that $x^{p-1} = 1 \bmod p$, and hence $x^{k(p-1)(q-1)} - 1 = 0 \bmod p$, as required.

Case 2: *x is a multiple of p.* In this case the expression in (2), which has x as a factor, is clearly divisible by p.

By an entirely symmetrical argument, $x(x^{k(p-1)(q-1)} - 1)$ is also divisible by q. Therefore, it is divisible by both p and q, and since p and q are primes it must be divisible by their product, $pq = N$. But this implies that the expression is equal to 0 mod N, which is exactly what we wanted to prove. □

So we have seen that the RSA protocol is *correct*, in the sense that Alice can encrypt messages in such a way that Bob can reliably decrypt them again. But how do we know that it is *secure*, i.e., that Eve cannot get any

useful information by observing the encrypted messages? The security of RSA hinges upon the following simple assumption:

Given N, e and $y = x^e \bmod N$, there is no efficient algorithm for determining x.

This assumption is quite plausible. How might Eve try to guess x? She could experiment with all possible values of x, each time checking whether $x^e = y \bmod N$; but she would have to try on the order of N values of x, which is completely unrealistic if N is a number with (say) 512 bits. This is because $N \approx 2^{512}$ is larger than estimates for the age of the Universe in femtoseconds! Alternatively, she could try to factor N to retrieve p and q, and then figure out d by computing the inverse of $e \bmod (p-1)(q-1)$; but this approach requires Eve to be able to *factor* N into its prime factors, a problem which is believed to be impossible to solve efficiently for large values of N. She could try to compute the quantity $(p-1)(q-1)$ without factoring N; but it is possible to show that computing $(p-1)(q-1)$ is equivalent to factoring N. We should point out that the security of RSA has not been formally proved: it rests on the assumptions that breaking RSA is essentially tantamount to factoring N, and that factoring is hard.

We close this note with a brief discussion of implementation issues for RSA. Since we have argued that breaking RSA is impossible because *factoring* would take a very long time, we should check that the computations that Alice and Bob themselves have to perform are much simpler, and can be done efficiently.

There are really only two non-trivial things that Alice and Bob have to do:

1. Bob has to find prime numbers p and q, each having many (say, 512) bits.
2. Both Alice and Bob have to compute exponentials mod N. (Alice has to compute $x^e \bmod N$, and Bob has to compute $y^d \bmod N$.)

Both of these tasks can be carried out efficiently. The first requires the implementation of an efficient test for primality as well as a rich source of primes. You will learn how to tackle each of these tasks in the algorithms course *CS*170. The second requires an efficient algorithm for modular exponentiation, which is not very difficult, but will also be discussed in detail in *CS*170.

To summarize, then, in the RSA protocol Bob need only perform simple calculations such as multiplication, exponentiation and primality testing to implement his digital lock. Similarly, Alice and Bob need only perform simple calculations to lock and unlock the the message respectively—operations that any pocket computing device could handle. By contrast, to unlock the message without the key, Eve would have to perform operations like factoring large numbers, which (at least according to widely accepted belief) requires more computational power than all the world's most sophisticated computers combined! This compelling guarantee of security without the need for private keys explains why the RSA cryptosystem is such a revolutionary development in cryptography.