Cryptography and RSA

1 RSA

Cryptography is concerned with the following scenario: Two people (we shall call them Alice and Bob) wish to communicate in the presence of an eavesdropper (Eve). Suppose that Alice wants to send to Bob secret message x. Cryptography provides solutions of the following form: Alice computes a function of x, $e(x)$, using a secret key, and sends $e(x)$ along the channel tapped by Eve. Bob receives $e(x)$, and using his key (which in traditional cryptography is the same as Alice's, but in modern cryptography it is not) computes a function $d(e(x)) = x$, recovering the message. Eve is presumably unable to recover x from $e(x)$ because she does not have the key.

A classical cryptographic method is *letter substitution*. Alice and Bob have agreed on a permutation π of the letters A,...,Z, and a message $m = a_1 a_2 \dots a_n$ consiting of n letters becomes $e(m) = \pi(a_1)\pi(a_2)\dots\pi(a_n)$. Despite the fact that there are 26! possible keys, this is a very weak cryptosystem, subject to the obvious *letter frequency attack*.

With the advent of the computer, cryptographers were able to create cryptosystems in which whole *blocks of letters* (or bits) are substituted, thus resisting any frequency attack. The most successful among them is the *Data Encryption Standard* (DES), a U. S. government sponsored cryptographic method proposed in 1976. DES uses a key with 64 bits (although only 56 bits are really used) to compute, through an extremely complicated sequence of bit operations, an encoding (or decoding) function of any input bitstring with 64 bits (to encode longer files, you break them in 8-byte blocks). The proponents of DES claim that it is secure, its detractors suspect that, in subtle ways, it is not. Key size of 56 is widely believed now to be inadequate for serious cryptography, since 2^{56} is no longer a very large number.

Around the same time the DES was invented, a very exciting new idea was proposed: *Public-key cryptography.* Bob would have two keys, his private key k_d, known only to him, and his public key, k_e. (It is perhaps more helpful to think that k_e is a *lock* while k_d is the lock's key.) k_e is known to everybody —it is on Bob's homepage, for example. (Continuing our dubious metaphor, this is as if there were many copies of the lock out there, with which people can seal boxes containing messages and send to Bob.) If anybody, for example Alice, wants to send a message x to Bob, then she encrypts into $e(x)$ it using the public key. Bob decrypts it using his private key. The clever part is this: (1) Decryption is correct, that is, $d(e(x)) = x$; (2) You cannot feasibly compute Bob's private key from his public key, or x from $e(x)$, so the protocol is secure (exactly like there is no way to figure out the key by looking at a good lock.)

RSA (from the initials of Ron Rivest, Adi Shamir, and Len Adleman, the three computer scientists who invented it) is a clever way to realize the public key idea. Here is how it works:

- *Key Generation* Bob finds two large primes p and q. ("Large" nowadays means a couple of hundreds of decimal digits, soon it may mean a thousand). To find such primes, Bob repeatedly generates integers in this range, and submits them to the Fermat test, until two of them pass it. Then Bob computes $n = p \cdot q$. Also, Bob computes a random integer $e < n$, with the only restriction that e must be prime to $(p-1)$ and $(q-1)$. *The pair (n, e) is now Bob's public key*, and Bob announces it. (In practice, to make encoding easier (see below), e is often taken 3. Of course, we have to reject primes that are 1 mod 3.)

Now Bob must generate his secret key. All he has to do is compute (by Euclid's algorithm) $d = e^{-1} \bmod (p-1)\cdot(q-1)$. *The pair* (n, d) *is Bob's private key.*

- Operation Whenever Alice (or anybody else) wants to send a message to Bob, she does the following:
  - She first breaks the message into bitstrings of length $\lfloor \log n \rfloor$ (remember, this is in the many hundreds of bits). She then encodes each bitstring by the following algorithm.
  - Let x be such a bitstring, and consider it as an integer mod$n$. Alice computes $x^e \bmod n$. This is the encoded message $e(x)$ Alice sends to Bob.
  - Bob, upon receipt of $e(x)$, computes $e(x)^d \bmod n$. A little algebra (and recalling Fermat's little Theorem for products of two primes from the previous lecture) gives:

$$e(x)^d = x^{d\cdot e} = x^{1+m\cdot(p-1)\cdot(q-1)} = x \bmod n$$

    Notice that this sequence of equations establishes that Bob decodes correctly. The first equation recalls the definition of $e(n)$. The second follows from the fact that d is the inverse of $e \bmod (p-1)\cdot(q-1)$, and thus if you multiply d with e you get 1 plus a multiple of $(p-1)\cdot(q-1)$. The last inequality recalls Fermat's little Theorem: $x^{(p-1)\cdot(q-1)} = 1 \bmod n$ (unless of course x is a multiple of p or q, a very unlikely event); so, multiples of $(p-1)\cdot(q-1)$ in the exponent can be ignored.

So, Alice can encode using Bob's public key. Bob can decode, using the private key only he knows. How about an eavesdropper? If Eve gets $e(x) = x^e \bmod n$, she can do several things: She could try all possible x's, encode them with Bob's public key, and find the correct one —that takes far too long. Or, she can compute $d = e^{-1} \bmod (p-1)\cdot(q-1)$ and from this $e(x)^d \bmod n$, which is x. But in order to do this she needs to know $(p-1)\cdot(q-1)$; and if she knows both $p \cdot q$ and $(p-1)\cdot(q-1)$, she knows p and q (can you see why?). In other words, she can factor n, an arbitrary product of two large primes —nobody knows how to do this fast. Or, finally, she could use her own ingenious method that decodes $e(x)$ without factoring n —it is widely believed that no such method exists.

So, in all available evidence, RSA is safe with suitably large n.

2 Signatures

Public-key cryptography is not only a clever and radically counterintuitive idea, it is a *fundamental* such idea. Because, once you have it, several tasks that seem impossible can be done. For example, digital signatures.

Suppose that Alice wants to send Bob a *signed* message. That is, she wants to send a message $ms(m)$, where m is an ordinary message, and $s(m)$ is a *signature*, a piece of text which depends on m and uniquely identifies Alice (i.e., Bob is sure the message is coming from Alice and not an impostor; also, Bob can convince a court of law that Alice indeed sent him this message). Sounds impossible, right?

Here's how it's done: $s(m) = d'(m)$, where $d'(\cdot)$ is Alice's *decoding* function. That is, Alice imagines for a second that the message m she is about to sent to Bob was received by her from somebody else, and applies to it her decoding algorithm (raise to her secret key modulo her public key).

Once Bob receives $ms(m)$, he is plausibly sure that Alice sent it, because only Alice has the private key that can transform m to $s(m)$.

And there are more counterintuitive feats that can be accomplished by applying public key cryptography: Playing poker over the Net, and proving to someone that a theorem is true without giving away anything about the truth (such demonstrations are known as *zero-knowledge proofs*).