CS 2742 (Logic in Computer Science) – Fall 2008
Lecture 21

Antonina Kolokolova

November 12, 2008

6.1 Application to cryptography

Recall Caesar's cipher: letter +3. Encode with $Code(i) = Letter(i + 3) \mod 26$, decode with $Letter(i) = Code(i - 3) \mod 26$. Problem: seeing enough messages can figure out the code, and knowing how to encode can easily decode.

6.1.1 Private-key cryptosystem

A natural way to make the system more secure is to apply a different rule for encoding each letter. For that, the parties exchanging encoded messages can agree on a key (e.g., text from a book) which they would "add" to the message to obtain the code. For example, the key could be "here" and the message: "come": the resulting code, using the same convention for letters as before, could be h+c=k, e+o=t, r+m=e, e+e=j giving "ktej". This code is much harder to break, it is not possible to crack it without knowing the key. However, knowing the key it is easy to decode: just subtract the key from the codeword $\mod 26$. Beware, though, that for a key to be really secure one should never reuse the same key to encode several messages.And, of course, there is a problem of securely agreeing on the same key in the first place.

It is much easier to think about this encoding when both the message and the key are binary strings. Recall from the Boolean function part of the course that there is a function "exclusive-or" (also called "XOR"), written as $\oplus$, which has the same truth table as $\vee$ except $1 \oplus 1 = 0$. That is, $0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$. Now, to encode a binary string (say $m = 101010$) by a key (say $k = 110011$) it is enough to take a bitwise XOR of them $c = m \oplus k = 101010 \oplus 110011 = 011001$. Notice that $\oplus$ has a very useful feature: for any binary string, an $\oplus$ of this string with itself is a string of all 0s. So doing an $\oplus$ with

the same string twice "cancels out" that string. This gives a decoding algorithm: if c is a codeword, then $c \oplus k = m \oplus k \oplus k = m$.

It can be proven that it is impossible to distinguish a completely random string of 0s and 1s from such a string made by key $\oplus$ message, where key and message are binary strings. Therefore, this encoding is secure (as long as the key is secure and only used to encode one message: if the key is used repeatedly then it compromises the security).

6.2 Public-key cryptosystem.

Although private-key cryptosystem is very secure, it has some drawbacks, the main of which is that the two parties need to exchange the keys beforehead in a secure way. But often in real life two parties want to exchange messages securely without having a chance to communicate privately before. For such a situation, a public-key cryptosystems were developed.

Intuitively, public-key cryptosystem works as follows. Suppose there are two people, call them Alice and Bob, who want to communicate securely. In particular, suppose that Bob wants to send a secure message to Alice. In public-key cryptography, Alice starts by generating a private key and a public key for herself. Private key she keeps secret; her public key she publishes online. Now, when Bob wants to send Alice a message that only she can read, he takes her public key and uses it to encode the message (in a different way than private-key encoding). He sends the message to Alice, who now uses her private key to decode the message.

The best known public-key cryptosystem is RSA (there are others, based on elliptic curves, lattices and other complicated mathematical objects). Security of the RSA cryptosystem is based on the assumption that given a product of two large prime numbers it is computationally infeasible to factor it. Here, by "large" people usually mean 128-bit long (that is, of a value up to 2^{128}), or something similar. This assumption is challenged by the quantum computers: it is indeed possible to factor numbers efficiently on quantum computers. However, the current quantum computer technology only allows operations on very small numbers: quantum computers available now can only operate with the total of about 16 bits.

RSA technology is based on the properties of $\mod n$ relation. Let us start by looking at several ways of looking at $\mod n$.

The following are equivalent ways to describe $a \equiv b(\mod n)$.

- $n|(a-b)$
- $a = b + kn$ for some k
- a and b have the same nonnegative remainder divided by n

- $a \mod n = b \mod n$.

In order to see how RSA works, we need to understand modular arithmetic: that is, how to do operations $\mod n$. Let $a = c(\mod n)$ and $b = d(\mod n)$. Then

- $(a + b) \equiv (c + d)(\mod n)$
- $(a - b) \equiv (c - d)(\mod n)$
- $ab \equiv cd(\mod n)$
- $a^m \equiv c^m(\mod n)$ for all positive m

An important object is an inverse of a number $\mod n$: an inverse s of a modulo n is a positive s such that $as \equiv 1 \mod n$.

Finally, with these definitions we are ready to describe RSA cryptography:

- Take large primes p and q and an integer e relatively prime to $(p - 1)(q - 1)$.
- The public key becomes pq and e
- Encrypt with $C = M^e \mod pq$
- Find d which is a positive inverse to e modulo $(p - 1)(q - 1)$.
- Decrypt with $M = C^d \mod pq$.

Take $pq = 55$, $e = 3$ – this is the public key. Message "HI" becomes 08, 09 and gets encoded as $8^3 \mod 55 = 17$ and $9^3 \mod 55 = 14$, so the message becomes "17 14".

To decrypt this message, Alice needs to compute d such that d is a positive inverse of e modulo $(p - 1)(q - 1)$. To compute the inverse, recall that if gcd(a,n)=1 then there exists s such that $as \equiv 1 \mod n$. Use Euclid's algorithm to find the inverse. Don't worry about this until your discrete math course, if you don't know this.

Why it works? Look at $C = M^e \mod pq$. Then $M = (M^e \mod pq)^d \mod pq$. So $(M^e \mod pq)^d = M^{ed} \mod pq$ Thus suffices to show $M = M^{ed} \mod (pq)$. But $ed = 1 \mod (p - 1)(q - 1)$. You would need to have some working knowledge of modular arithmetic to work through the proof.