CS 70, March 10, 2005

- review from Tuesday
  - RSA algorithm
  - proof
  - observation: there are enough primes to make it relatively easy to find two of them; there are approximately x/(ln x) primes between 1 and x
    - 1 in 20 SIDs are prime
  - observation: encryption function and decryption function can be applied in either order
- authentication
  - situation: message isn't necessarily secret, but you want to make sure you know who authored it
  - Alice creates a digital signature S for her message M: S = M^d mod n, where (d,n) is Alice's private key; she then might encrypt it using Bob's public key
  - Bob, on receiving the message, would unencrypt it using his private key, then apply Alice's public key to it to make sure it came from her and wasn't tampered with
  - actually what usually happens is that Alice signs a *digest* of the message M, formed by applying a hash function h to M, then appends the signed shorter version of the message to the actual M; Bob then applies the same hash function h to M to get the digest, then applies Alice's public key to the signature and compares the result with h(M)
    - note that Alice is essentially signing every message that hashes to the same digest; this is a security risk
    - recently SHA-1, a common hash function for this purpose, was compromised; a simpler function that delivers the same SHA-1 hash value was found
  - a certificate provides assurance that Alice's *public key* is correct; it adds a signature (e.g. of Verisign) to the public key that verifies its authenticity
  - there are also time-stamp authorities to verify the time that a message was composed/ sent
- practical uses of RSA
  - private-key algorithms are much faster (~1000 times)
    - in general, the public exponent is usually much smaller than the private exponent, meaning that decrypting a msg is faster than encrypting and verification of a signature is faster than signing; that's OK because signing and encrypting are only done once, while verifying a given signature may happen many times
  - RSA is usually used to encode a key that can then be used with a private key system

CS 70, March 10, 2005

- ssh
  - the ssh-keygen program generates a public/private key pair by randomly selecting large primes; this takes a couple of minutes on nova.cs
  - Your private key is stored in .ssh2/id_dsa_2048_a; your public key is stored in .ssh2/id_dsa_2048_a.pub
  - Login procedure from account A to account B: ssh on A sends a signed session identifier (known only to client and server, encrypted with A's private key) to B's ssh server. B makes sure A's public key is in a file named authorized_keys and is correct. If so, B's ssh server allows the login to B.
  - ssh also maintains and checks a database containing id info for every host it's ever been used with; if a host changes, the user is warned
  - All communications back and forth are encrypted with a private-key method.
- other related applications
  - playing poker over the Internet ("Mental Poker", by SRA, in *The Mathematical Gardner*, edited by David Klarner, published by Wadsworth in 1981)
    - Alice and Bob agree on encryption/decryption functions E and D for which
      $E_K(X)$ is the encrypted version of a message X under key K.
      $D_K(E_K(X)) = X$ for all messages X and keys K.
      $E_K(E_J(X)) = E_J(E_K(X))$ for all messages X and keys J and K.
      Given X and $E_K(X)$, one can't derive K for all X and K.
      Given any messages X and Y, one can't find keys J and K such that $E_J(X) = E_K(Y)$.
    - Alice and Bob choose secret keys A and B, respectively.
    - Bob encrypts the fifty-two messages "2 of clubs", "3 of clubs", …, "ace of spades" with his secret key B. He then randomly rearranges the encrypted deck and sends it all to Alice.
    - Alice selects five cards (messages) at random and sends them back to Bob, unencrypted (they were already encrypted by Bob). Bob decrypts these messages to find out what his hand is.
    - Now Alice selects five other messages, encrypts them with her secret key A, and sends them to Bob. Each of these messages is now doubly encrypted, once by Bob, once by Alice. Bob decrypts them (using commutativity of the encryption/decryption function), then sends them back to Alice. After decrypting those messages, Alice knows what her cards are.
    - At the end of the game, both players reveal their secret keys. Now either player can check that the other was "actually dealt" the cards he or she claimed to have during play. It's computationally difficult to reveal the wrong key.

CS 70, March 10, 2005

- fingerprinting
  - Suppose we want to transmit a file to the moon. There are occasional bit errors. We'd like to verify that our file got there correctly. We could of course re-send the file a second time, but bandwidth to the moon is very expensive, so it'd be nice to have a better solution.
  - Solution: Fingerprinting. Suppose we know we'll want to send a message x of length n bits. We agree in advance on a random prime p chosen randomly from the primes up to n^3. Write x as a number $0 \le x < 2^n$ in the obvious way. Then our fingerprint is $F_p(x) = x \bmod p$. We send $F_p(x)$ along with x. Note that $F_p(x)$ has length 3 lg n bits, so is much shorter than x.
  - What's the probability a random transmission error goes undetected? Well, suppose the receiver receives (y,c) instead of $(x,F_p(x))$, and suppose y is different from x. Then the probability a bit error goes undetected is the probability the fingerprint c looks valid:

    $\Pr[F_p(y) = c]$

    $= \Pr[p \mid y-c]$

    $\le$ (# prime divisors of y-c) / (# primes $< n^3$)

    $\le n$ / (# primes $< n^3$)

    $\sim n / (n^3 / (\ln n^3))$

    $\le 1/n$.

    So the error probability is pretty low -- and this is true even no matter how many errors in the transmission there might be. Note that this is also very fast: O(n lg n) time to compute the fingerprint, and O(lg n) extra bits sent along with the n bit message.