Asymmetric crypto

continued

Simple RSA key exchange

- A sends public key d_A and id_A to B
- B selects a random session key k_S
- B sends $c = E_{dA}(k_S)$ to A
- A decrypts $D_{eA}(c) = k_S$, which is used for session (symmetric)

Vulnerable to man-in-the-middle attack - need both confidentiality and authenticity

Naïve uses of RSA

A. Using too short messages makes it easy to find private key

B. Using small private key (for speed) makes it easy to find it

C. Having e, d and n gives you p and q

A. Short m in RSA

Key: $c = (m_1 \times m_2)^e = m_1^e \times m_2^e = c_1 \times c_2 \pmod{n}$
If $m < 2^k$ then probably $m_1, m_2 < 2^{k/2}$
(k=40..64 => p=18-50%)

1. Build sorted table $\{ 1^e, \ldots, (2^{k/2})^e \} \pmod{n}$

2. Find i, j s.t. $c/i^e = j^e \pmod{n}$ for $i,j=1..2^{k/2}$;
   then $m = i \times j$

Uses space $2^{k/2} \log n$ bits, time < $O(2^k)$
For 56-bit DES keys and 1K-bit n:
space 32Gbyte, 2^{29} mod ops << 2^{56} brute f.
Fix: padding (complex, randomized).

B. Small private keys

- A small key makes exponentiation faster (75% or so)
- If $3e < n^{1/4}$ and $q < p < 2q$, then e can be found efficiently (through $\phi(n)$) [Wiener, book pp207]
- E.g, for 1024-bit n, use 256-bit e!
- Possibly worse (also $e < sqrt(n)$)

C. Getting p and q

- If e, d and n are known, we can find p and q efficiently using randomized algorithm [book §5.7.2]
- I.e., if secret key lost, must change also n!

Different breaks

- Total break: adversary gets private key (for asymmetric) or secret key (for symm.)
- Partial break: adversary decrypts ciphertext (or gets info about plaintext) with non-negligible probability
- Distinguishability of plaintexts: with probability > 1/2 the adversary can distinguish the encryptions of two given plaintexts (or encryption from random junk)

Generators and discrete logarithms

- a is a primitive root (or generator) modulo p if Z_p^* is generated by exponentiation of a mod p
  - ex: 2 is a primitive root mod 11
    $Z_{11}^* = \{ 1..10 \} = \{2^{10},2^{1},2^{8},2^{2},2^{4},2^{9},2^{7},2^{3},2^{6},2^{5}\}$
- For any b, and a a generator mod p, a unique i exists s.t. $b=a^i \bmod p$
- i is the discrete logarithm (index) of b for base a, mod p: $i=\mathrm{ind}_{a,p}(b)$

Diffie-Hellman key agreement algorithm

- Public: prime q, generator a mod q
- User A selects private random $x_A < q$, computes $y_A = a^{x_A} \bmod q$
- User B selects x_B and computes y_B same way
- Each sends her y to the other, computes shared $k = (y_B)^{x_A} \bmod q = (a^{x_B} \bmod q)^{x_A} \bmod q = a^{x_B \cdot x_A} \bmod q = (a^{x_A})^{x_B} \bmod q = (y_A)^{x_B} \bmod q$

Diffie-Hellman cryptanalysis

- Known: q, a, y_A, y_B
- To get k, need x_A or x_B
  $x_A = ind_{a,q}(y_B)$
- For q a large prime (> 300 digits), this is computationally infeasible

ElGamal Public-Key

- Like Diffie-Hellman but after exchanging y values, a message $m < q$ can be encrypted:
  1. select random $k < q$,
     compute $K = y_B^k \bmod q$
  2. send (c_1, c_2) where
     $c_1 = a^k \bmod q$
     $c_2 = K \cdot m \bmod q$
  3. decryption:
     $K = c_1^{xB} \bmod q$
     $m = c_2 \cdot K^{-1} \bmod q$