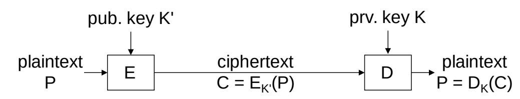
Public Key Cryptography

BİL 448/548
Internet Security Protocols
Ali Aydın Selçuk

Public Key Cryptography

- The single most important idea in modern cryptography.
- Proposed by Diffie & Hellman, 1976.
- Asymmetric key cryptography:



- It shouldn't be possible to obtain K from K'. So, K' can safely be made public.

Public Key Cryptography

PKC solves the classical "key distribution problem":

- If there is no secure channel, how can A & B share the key securely?

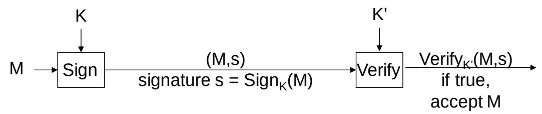
PKC solution:

- Alice makes her encryption key K' public
- Everyone can send her an encrypted message: $C = E_{K'}(P)$
- Only Alice can decrypt it with the private key K: $P = D_K(C)$

Public Key Cryptography

PKC also solves the message source authentication problem:

- Only Alice can "sign" a message, using K.
- Anyone can verify the signature, using K'.



Only if such a function could be found...

Discrete Logarithm Problem

- DLP: Given g and $y = g^x$, what is x?
- Easy over $\mathbb{Z}$.
  E.g., if $2^x = 4096$, $x = 12$.
- Hard over $\mathbb{Z}_p$.
  E.g., if $2^x = 28 \pmod{113}$, x = ?

Diffie-Hellman Key Exchange

- Public: prime p, generator g.
- Alice chooses random a (secret);
  Bob chooses random b (secret).

Alice → Bob: $g^a \bmod p$

Bob → Alice: $g^b \bmod p$

Alice ↔ Bob: $K = g^{ab} \bmod p$

Alice: $K = (g^b)^a \bmod p$

Bob: $K = (g^a)^b \bmod p$

Security of DH

- Discrete Log Problem: Given p, g, $g^a \bmod p$, what is a?
- DH Problem: Given p, g, $g^a \bmod p$, $g^b \bmod p$, what is $g^{ab} \bmod p$?
- Conjecture: DHP is as hard as DLP.
  (note: Neither is proven to be NP-hard.)

Efficiency of DH

Generating a large prime

- Generate a random number & test for primality.
- Primality testing is efficient.
- Density of primes:

Prime Number Theorem: For $\pi(n)$ denoting the number of primes $\leq n$, we have

$$\pi(n) \sim n / \ln n.$$

That is,

$$\lim_{n \to \infty} (\pi(n) \ln n) / n = 1.$$

Efficiency of DH

How to compute (g^a mod p) for large p, g, a?

$$x^n = \begin{cases} (x^k)^2 & \text{if } n = 2k \\ (x^k)^2 x & \text{if } n = 2k + 1 \end{cases}$$

"Repeated squaring": Start with the most significant bit of the exponent.

E.g. Computing 3^{25} mod 20. $25 = (11001)_2$

$y_0 = 3^{(1)} \bmod 20 = 3$

$y_1 = 3^{(11)} \bmod 20 = 3^2\, 3 \bmod 20 = 7$

$y_2 = 3^{(110)} \bmod 20 = 7^2 \bmod 20 = 9$

$y_3 = 3^{(1100)} \bmod 20 = 9^2 \bmod 20 = 1$

$y_4 = 3^{(11001)} \bmod 20 = 1^2\, 3 \bmod 20 = 3$

Further efficiency with preprocessing x^i, $i < 2^k$, for some k.