Admin: Quiz on Wed (4/6/11)
Open notes only. (No books, laptops, smartphones,...)

Outline:
- certificates
- scaling
- X.509
- SPKI/SDSI
- revocation
- IBE
Certificates:
- Last time we did Needham-Schroeder
- $\{ K_{PA}, A^2 \}_{K_{SS}}$ is a prototypical certificate:
  - server signing key
  - "Alice"
  - Alice's PK
  - $S$ certifies that Alice's key is $K_{PA}$
- Others can get this from $S$, or from $A$.

Scaling
- How do we go from 100 users to $10^8$ users?
  - Everything starts breaking:
    - there is no one server everyone trusts (?)
    - one server can't handle load
    - what are names?? $\Leftarrow$ subtle, but hard & important

Names:
How does Alice know Bob's name?
Who guarantees that names are unique? How is this done?
  (compare: email addresses...)
If Alice can get Bob's name (e-mail address) correctly, why can't she get his PK the same way?
X.509 hierarchy

```
root CA
   ↓ cert
   ↓
US
   ↓
IBM
   ↓
IBM-East
   ↓
John-Smith
```

DN= "distinguished name":

CD=US/ORG=IBM/DIV=IBM-EAST/CN=John-Smith

Names become unwieldy for people to use.

Certs have: version #
         cert serial #
         sig. alg.
         Issuer DN
         Subject DN
         Validity period
         Subject PK alg & key
         Issuer unique #
         
         extensions: type; crit/non-crit/value
               key usage (enc/sig, cert/sig)
               
               
               
               
               
               
               
               SHA-256, alt subject alt name, etc. constraints
SPKI/SDSI

- no global names
- each PK has its own name space (each key is a CA...)
- Certs have validity period
- two types of certs: name & auth
  - name cert: (K issues PK)
    - \( K \cdot Alice \Rightarrow \text{value} \ (\text{signed by } K) \)
    - \( \overset{\rightarrow}{K} \) (another PK)
    - or \( \overset{\rightarrow}{\text{another name}} \)

  e.g. \( K \cdot Bob \Rightarrow K_0 \)

  \( K \cdot Bob \Rightarrow K_1 \cdot Bob-Smith \)

  \( K \cdot Alice \Rightarrow K_2 \cdot \text{eecs}.Alice-Smith \)

  \( \overset{\rightarrow}{K_2 \cdot \text{eecs}} \Rightarrow K_3 \)

  \( K_3, Alice-Smith \Rightarrow K_A \)

  naturally have groups: \( K \cdot \text{friends} \Rightarrow K_1, Alice \)
  \( K \cdot \text{friends} \Rightarrow K_1, Bob \)
  \( K \cdot \text{friends} \Rightarrow K_1 \cdot \text{m.it.eecs.student} \)
Can put group name on ACL
   \[ k \to \text{friends my red this directory} \]

Given a set of ACS, it is possible to tell if some local name
(from ACL) can evaluate to your key.

\[ \text{Authent: key + right} \Rightarrow \text{key (delegatable, or not)} \]
\[ \Rightarrow \text{name} \]

\[ \text{Example: } k_i \overset{[\text{read d:}]}{\Rightarrow} k_j, \text{ Alice (no delegable)} \]

working group
seen before

A good alg for determining authorization
Certificate revocation

Why?  
- key compromise
  - change of affiliation or authorization
  - change of name (e.g., merger)

fairly high "churn rate"...

Certificate says "good until 2015-12-01" —
who decides if that is good enough

Issuer?

Relying party? \( \Leftarrow \) should be relying party...

Issuer has authoritative DB, cert is month snapshots... [Note that DB itself may not yet fully reflect key compromise, etc.]

Method 1: on-line check
  - ask issuer if cert still good; signed response
  - OCSP (online certificate status protocol)
    - heavy load on server!

Method 2: CRL's (Certificate revocation list)
  - Server periodically issues CRL, giving list of cancelled (revoked) cert serial #s, signed
    - can get long!
method 3: (Micali)

- \( x_3 \)
- \( x_2 \)
- \( x_1 \)

-cut contains end point \( x_0 \) of chain of hash values

-cut sized

one day \( d+i \), where \( d = \text{cut-date} \)

need \( x_i \) (or \( x_j \) for \( j > i \)) to validate cut. Server can give \( x_i \) to principal who can do about, or issue \( x_i \) in response to inquiry...

relies on hash of times & checks

no \( x_i \) given out, cut "expires"...
Identity-based encryption (IBE)

everyone has "identity," (e.g. email address)

Single TTP $T$

anyone can encrypt to other, knowing only PK of $T$ & ID of recipient

i.e. Alice's PK = $(PK_T, \text{"alice@abc.com"})$

Alice's SK - she got this from $T$ (note trust issue)

uses bilinear maps:

$G_1, G_2$ groups of prime order $q$, $g$ generator $G_1$ (public)

$e: G_1 \times G_1 \rightarrow G_2$ s.t. $e(g^a, g^b) = e(g, g)^{ab}$

$s = T's$ SK

g^s = T's$ PK

$H: \text{ID's} \rightarrow \text{elts of } G_1$  public hash fn

user gives $ID_A$ to $T$, gets $H(ID_A)^s$ back

Alice gets SK

To encrypt Alice: use key $e(H(ID_A)^s, g^s)$

Alice decrypts using key $e(H(ID_A)^s, g)$

= by magic of bilinear maps

* mention LES (Adleman/Hohenberger/Rivest)