Admin: PS #4 out. Due Monday

Quizzes being graded. (It was a long quiz!)

Outline:

□ certs
□ scaling
□ X.509
□ SPKI/SDSI
□ cert revocation
□ IBE
Certificates:

- Last time we did Needham-Schroeder
- \( SK_{PA}, A^3_{PA} \) is a prototypical certificate:
  - \( S \) certifies that Alice's key is \( K_{PA} \)
  - Alice's PK
  - Server signing key

- 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 (email address) correctly, why can't she get his PK the same way?
X.509 hierarchy

DN = "distinguished name" =
    CO=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, ..)
    cert-policies, subject alt name, path constraints
SPKI/SDSI

- no global names
- each PK has its own name space (each key is a CA...
- certs have valid time period
- two types of certs: name & auth
  name cert: (K -> issues PK)
  \[ K, Alice \rightarrow value \quad \text{(signed by K)} \]
  \[ \rightarrow K' \quad \text{(another PK)} \]
  or \[ K \rightarrow \text{another name} \]

  e.g. \[ K, Bob \rightarrow K_0 \]
  \[ K, Bob \rightarrow K_1, Bob-Smith \]
  \[ K, Alice \rightarrow K_2, eecs, Alice-Smith \]
  \[ K_2, eecs \rightarrow K_3 \]
  \[ K_3, Alice-Smith \rightarrow K_A \]

  naturally have groups: \[ K, friends \rightarrow K_1, Alice \]
  \[ K, friends \rightarrow K_1, Bob \]
  \[ K, friends \rightarrow K_1, m.t.eecs, student \]
Can put group name on ACL
  K, friends may read this directory

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

Auth crit: key + right \implies\ key \iff \text{name} (delegatable, or not)

example: \[ K \underbrace{[\text{read \$d:/\$}]} \implies K_1, \text{Alice} \ (\text{no delegable}) \]

working group
secure browser

\text{1 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? \( \checkmark \) should be relying party...

issuer has authoritative DB, certs are merely snapshots... [Note that DB itself may not yet fully reflect key compromises, etc...]

Method 1: on-line check
ask issuer if cert still good; signal 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)

\[ \downarrow x_3 \]
\[ \downarrow x_2 \]
\[ \downarrow x_1 \]

\[ \text{can't contains end point } x_0 \text{ of chain of hash values} \]

\[ \text{cut } \frac{30}{=10} \text{ sized} \]

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

need \( x_i \) (or \( x_j \) for \( j > i \)) to validate cert. Server can give \( x_i \) to principal who can't be about, or issue \( x_i \) in response to inquiry...

relying party knows \( i \) times & checks

no \( x_i \) given out, cert “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, "alice@abc.com")

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

uses bilinear maps:

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

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

s = T's SK

g^s = T's PK

H : ID's \to \text{elts of } G_1, \Public key fn

user gives ID_A to T, gets H(ID_A)^s back \underbrace{\text{Alice's SK}}

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

Alice decrypt using key e(H(ID_A)^s, g) = by magic of bilinear maps

* mention LES (Adida/Hohenberger/Rivest)