6.857 Rivest
4/8/09 L17.1

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
- $\{K_{PA}, A\}_{K_{SS}}$ is a prototypical certificate: S certifies that Alice's key is K_{PA}

  K_{SS} — server signing key
  A — "Alice"
  K_{PA} — Alice's PK

- Others can get this from S, or from A.

Scaling

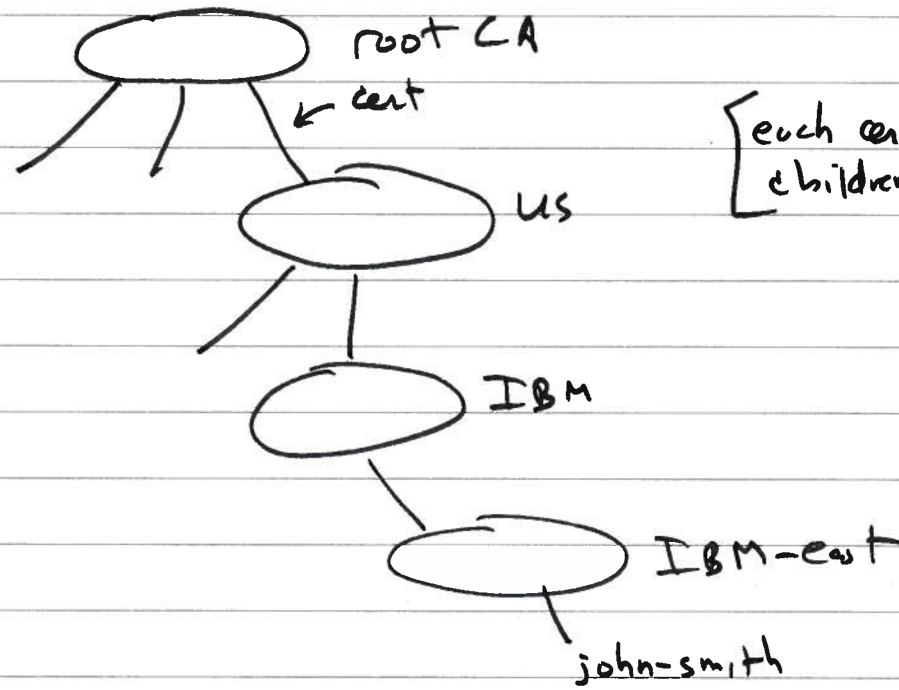
- How do we go from 100 users to 10^8 users?
- Everything starts breaking:
  - there is no one server everyone trusts (?)
  - one server can't handle load
  - what are names?? $\Leftarrow$ subtle, but hard & important

- 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 or email address) correctly, why can't she get his PK the same way?

6.857 Rivest
4/1/09 L17.3

X.509 hierarchy



each certifies its children

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, certsig)...
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 validity period
- two types of certs: name & auth

name cert: (K is issuer PK)

$K.\text{Alice} \Rightarrow \text{value}$ (signed by K)

→ K' (another PK)
or → another name

e.g. $K.\text{Bob} \Rightarrow$ ~~[illegible]~~ K_0

$K.\text{Bob} \Rightarrow K_1.\text{Bob-Smith}$

$K.\text{Alice} \Rightarrow \underbrace{K_2.\text{eecs}}.\text{Alice-Smith}$

$\underbrace{K_2.\text{eecs}} \Rightarrow K_3$

$K_3.\text{Alice-Smith} \Rightarrow K_A$

naturally have groups: $K.\text{friends} \Rightarrow K_1.\text{Alice}$
$K.\text{friends} \Rightarrow K_1.\text{Bob}$
$K.\text{friends} \Rightarrow K_1.\text{mit.eecs.student}$

Can put group name on ACL
K. friends may read this directory

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

<u>**Auth cert:**</u> key + right ⇒ key (delegatable, or not)
or ⇒ name

example: $K \underbrace{[\text{read d:/}]}_{\text{right}} \Rightarrow K_1.\text{Alice}$ (no delegation)

working group
secure browser
∃ 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? ⇐ ✓ 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; 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!

6.857 Rivest
4/8/09 L17.7

<u>method 3</u>: (Micali)

$$\begin{array}{l} \vdots \\ \downarrow x_3 \\ \downarrow x_2 \\ \downarrow x_1 \\ \downarrow \\ \text{cert } \boxed{x_0} \text{ signed} \end{array}$$

cert contains end point x_0 of chain of hash values

one day $d+i$, where d=cert date
need x_i (or x_j for $j>i$) to validate cert. Server can give x_i to principal who cert is about, or issue x_i in response to inquiry...

relying party hashes i times & checks

no x_i given out, cert "expires"...

6.857 Rivest
4/8/09 L17.8

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
~~parameters~~
i.e. Alice's PK $\approx (PK_T,$ "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)
$e: G_1 \times G_1 \rightarrow G_2$ s.t. $e(g^a, g^b) = e(g,g)^{ab}$

s = T's SK
g^s = T's PK
H: ID's $\rightarrow$ elts of G_1 public hash fn

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

To encrypt to Alice: use key $\underline{\underline{e(H(ID_A), g^s)}}$

Alice decrypts using key $\underline{\underline{e(H(ID_A)^s, g)}}$

= by magic of bilinear maps

- mention LES (Adida/Hohenberger/Rivest)