Admin:
- Project presentations start Monday May 7th
- Final report is due Wednesday May 16th
- Boat Bank on ZKP Wed this week

Today:
- Public-Key Infrastructures (PKI)
  - Keys, names, parties
  - Certs (X.509)
  - Identity-Based Encryption (IBE)
  - BIP-32
  - SPKI/SDSI

Reading:
- Ferguson, Part IV
- Smart, Chapter 18
Keys, Names, Parties

\[(PK, SK) \leftarrow \text{keyGen}(1^k)\] for PK scheme

Distribution & Authentication of PKs

Alice: What is Bob's PK?

Diffie-Hellman: Directory mapping

\[\text{name} \rightarrow \text{(public) keys}\]

How to implement such a directory?

Do we need a TTP? (Trusted 3rd party)

How to revoke a key? (e.g., after compromise of SK)

Types of PKs: signing

encryption

Certificate authority
Names

Identifier, string.

Fixed-length, variable length. Alphabet?

Human-friendly.

Memorable

May have semantics “IBM”, “John Smith”, “LN”

Name of key? of person? of org?

Globally unique? Locally unique? Not nec. unique?

Relative to some context or namespace

May be segmented/hierarchized

A, B, C

Nick names.

Pseudonyms.

PK as name? (cf. Bitcoin)

(tries to avoid problem of name may?)
Parties

People

Computers, Servers,

Organizations

Users,

Trusted Third Parties (TTPs).
Certificates

Signed stmt from an "issuer" about a "subject" (name)'s public key.


- One entry extracted from master directory, signed.
- "Alice Jones' PK is (RSA, n=314..., e=57).

(signed, CA) (dated)

- CA = "certificate authority"

Why is a CA trusted? By whom?
For what names?

- There can't be one CA...

Alice can learn Bob's PK from his certificate.

(Which Bob can give her)

(example: Server certificates)
X.509 hierarchy

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

DN = "distinguished name"

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

subject's DN

These DN's become unwieldy for people to use.

Certificate chain
Certificate fields:

Version #
Cert serial #
Signature algorithm
Issuer DN
Subject DN
Validity period (not before, not after)
Subject PK (alg & key)
Issuer unique #

Extensions: type & critical/non-critical flag
  key usage (encs, sigs, certs)
  cert policies
  subject alternate name
  path constraints

Used by TLS (SSL)
Certificate revocation

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

Fairly high "churn rate"

If certificate says "good until 2020-12-01"
who decides if that is good enough?
issuer? (current practice)
relying party? ← Should be relying party!
(they are taking the risk...)

Helpful to think about it this way:
- Issuer maintains authoritative DB
- Certificates are merely "snapshots" of items
- Note that DB may not fully reflect key compromises, etc...

Method 1: On-line check:

Relying party asks issuer if cert still good
Issuer gives signed response (new cert?)
≡ OCSP (online certificate status protocol)
     Puts heavy load on server!
Method 2: CRL's (certificate revocation list)
Server periodically issues CRL,
giving list of revoked cert serial #s
(signed, of course)
CRL can get long!

Method 3: (due to Micali)

cert contains end point of a hash chain

\[ \cdots \x_365 \x_364 \x_363 \x_3 \x_2 \x_1 \x_0 \]

name, etc.

On day d+i, where d = cert issuing date,
you need \( x_i \) to validate cert. Server can
give \( x_i \) to keyholder, or to anyone else.
Relying party hashes \( i \) times & checks that result = \( x_0 \).
If no \( x_i \) given out, cert "expires".
IBE - Identity-Based Encryption

User PK = (TTPkey, username)

TTP issues SK to user,

(TTP knows user's SK)

Boneh - Franklin

- $K_{TTP} = s \cdot P$
  
  $s = TTP$ SK
  
  $P =$ generator of $G$

- User with ID wants SK
  
  $Q_{ID} = H_1(ID) \in G^*$
  
  $d_{ID} = s \cdot Q_{ID}$

- Enc m to user using $(K_{TTP}, ID)$
  
  $Q_{ID} = H_1(ID) \in G^*$
  
  $g_{ID} = e(Q_{ID}, K_{TTP})$
  
  $c = (r \cdot P, m \oplus H_2(g_{ID}^r)) = (u, v)$

- Decryption given $(u, v)$
  
  $m = v \oplus H_2(e(d_{ID}, u))$

Extend to HIBE

Hierarchical
SPKI/SDSI

- alternative framework by Ellison, Lampson, Rivest, others
- No global names (sort of...)
- Each PK is a CA & has its own name space

\[ K \cdot A \]

PK of issuer

\( K \cdot A \Rightarrow X \)
given as "rewrite rule"

- identifier or name

- Two types of certificates:
  
  name cert = associates name with key
  
  auth cert = gives permission (authorization) to subject(s)

- Name certs:
  Provide elegant & flexible "algebra" of names
  Allows one to describe groups (multivalued names)

\[ \text{cert} = (K, A, X) \]

\( K = \text{issuer PK} \)

\( A = \text{identifier (name)} \)

\( X = \text{PK} \)

or \( PK_{id}, id_2...id_n \) extended name
Authorization certs & ACL's

ACL = access control list

Can put name on ACL for a resource:

"Only individuals in K's friends may read files in this directory"

Authorization cert:

Issuer key K

Rights being granted [read directory foo]

Delegatable or not

Subject key or name

K [read "/foo"; delegatable] \(\Rightarrow\) K's friends

There are polynomial-time algorithms for determining whether a given collection of name & auth certs implies that a given key is authorized to perform a given action
Examples:

\[
K \cdot \text{Bob} \implies K_0 \\
\text{"I say Bob's PK is } K_0 \text{"}
\]

(Note: the name "Bob" is my choice & arbitrary. It doesn't need to coincide with anything else...)

\[
K \cdot \text{Bob} \implies K_1 \cdot \text{Bob-Smith} \\
\text{"My Bob is the same as } K_1 \text{'s Bob Smith"}
\]

\[
K \cdot \text{Bob} \implies K_\text{mit.} \cdot \text{Bob-Smith} \\
\text{"My Bob is same as mit's Bob Smith"}
\]

\[
K \cdot \text{mit} \implies K_a \\
\]

\[
K \cdot \text{Bob} \implies K_a \cdot \text{BobSmith} \\
\text{inferred rule}
\]

Groups:

\[
K \cdot \text{friends} \implies K \cdot \text{Alice} \\
\{ \text{group members} \}
\]

\[
K \cdot \text{friends} \implies K \cdot \text{Bob} \\
\ldots
\]

Let \( K_a = \text{mit PK} \)

\[
K_a \cdot \text{faculty} \implies K_a \cdot \text{eecs.factulty} \\
K_a \cdot \text{faculty} \implies K_a \cdot \text{math.factulty} \\
\]

(group defined as union of other groups)
BIP-32 (Bitcoin Improvement Proposal)

User-generated tree of PKs. Unlinkable. Can

generate child PK from parent PK with knowing SK.

Actually, \((PK, \text{chaincode}) \text{ pair}\)

\[
\begin{align*}
(PK', \text{cc}') & \quad \xrightarrow{i} \quad (PK, \text{cc}) \\
i & = \text{child #} \\
(PK, \text{cc}) & \quad \xrightarrow{i} \quad (SK', \text{cc}') \\
\text{parent} & \quad i = \text{child #} \\
& \quad \Rightarrow \text{not possible}
\end{align*}
\]

Two kinds of children: hardened, & not
derive \((PK', \text{cc}')\) from \(\text{cc}, PK, i\)
of \(\text{cc}, SK, i\) (hardened)

\[
\begin{align*}
\text{leaves are hardened} \\
\text{can receive payments} \\
\text{can't make them.}
\end{align*}
\]
Open Problems

1. Post-Quantum Crypto
2. Resilience
3. Assurance
4. Crypto Policy