Admin:

Projects

Quiz on 4/17

Today:

PKI & Certificates

TLS 1.3
Keys, Names, Parties

\((PK, SK) \leftarrow \text{keyGen}\left(I^\uparrow\right)\) for PK scheme

Distribution & Authentication of PKs

Alice: What is Bob's PK?

\[\text{name}\]

Diffie-Hellman: Directory mapping

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

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”.

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 mgmt.)
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=31411, e=57)."
  (signed, CA) (dated)

- CA = "certificate authority"
  Why is a CA trusted? By whom?
  For what names?

- There can't be one CA...
  - Let's Encrypt is free CA

Alice can learn Bob's PK from his certificate.

(which Bob can give her)

(example: server certificates)

Certificate transparency → google project to log all certificates
  (uses Merkle trees to make append-only log)
X.509 hierarchy

- **trust anchor**
- root certifies US

**root**

**US**

- **IBM**
- **IBM-East**

**John Smith**

DN = "distinguished name"

= "CN=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)
relaying party? \(\text{should be relying party!} \)
\(\text{(they are taking the risk...)}\)

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?)
\(\text{OCSP} (\text{online certificate status protocol})\)
Put's 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 endpoint of a hash chain

\[ x_0, x_1, x_2, x_3, \ldots \]

On day \( d+i \), where \( d = \text{cert issuing date} \),
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 = (TTP key, username)

TTP issues SK to user,

(TTP knows user's SK)

Boneh - Franklin

1. $K_{TTP} = s \cdot P$
   $s = TTP SK$
   $P = \text{generator of } G$

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

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

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

Extend to HiIBE

Hierarchical
BIP-32 (Bitcoin Improvement Proposal)

User-generated tree of PKs. Unlinkable. Can generate child PK from parent PK with learning SK.

Actually, (PK, childcode) pair:

\[
(\text{PK}', \text{cc}') \xrightarrow{i} (\text{PK}, \text{cc}) \xrightarrow{i} (\text{PK}', \text{cc}')
\]

\[ i = \text{child #} \]

\[ (\text{SK}, \text{cc}) \xrightarrow{i} (\text{SK}', \text{cc}') \]

\[ \text{parent} \]

Two kinds of childen: hardened, & not hardened.

Derive (PK', cc') from cc' PK, i:

or cc, SK, i (hardened)

\[
\leftarrow \text{leaves are hardened cannot receive payments cannot make them.}
\]