TOPIC: 6.857

DATE: 4/3/19

FILE UNDER:

PAGE: L15.1

Admin:

Projects

QUIZ on 4/17

Today:

PKI & Certificates

TLS 1.3

TOPIC:

DATE:

FILE UNDER:

PAGE:

Keys, Names, Parties

$(PK, SK) \leftarrow KeyGen(1^\lambda)$ for PK scheme

Distribution & Authentication of PKs

Alice: What is Bob's PK?

(Bob's ← name)

Diffie-Hellman: Directory mapping
names → (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

TOPIC:

FILE UNDER:

DATE:

PAGE:

Names

Identifier, string.

Fixed-length, variable length. Alphabet?

Human-friendly.

Memorable

May have semantics "IBM", "John Smith",

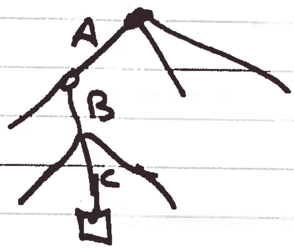
Name of key? of person? of org?

Globally unique? Locally unique? Not nec. unique

Relative to some context or namespace

May be segmented / hierarchical

A.B.C



Nick names.

Pseudonyms.

PK as name? (cf. Bitcoin)

(tries to avoid problem of name mgt.)

TOPIC:

FILE UNDER:

DATE:

PAGE:

Parties

People

Computers, Servers

Organizations

Users.

Trusted Third Parties (TTPs).

TOPIC:

DATE:

FILE UNDER:

PAGE

Certificates

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

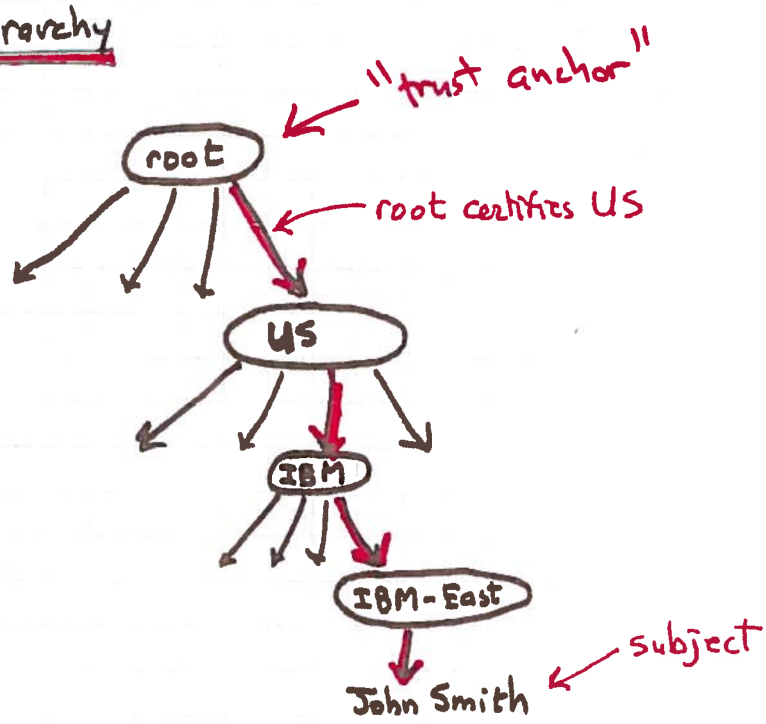
(Loren Kohnfelder, MIT B.S. thesis, 1978)

- 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...
- 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



DN = "distinguished name"

= "CO=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   Issuer PK missing?

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,

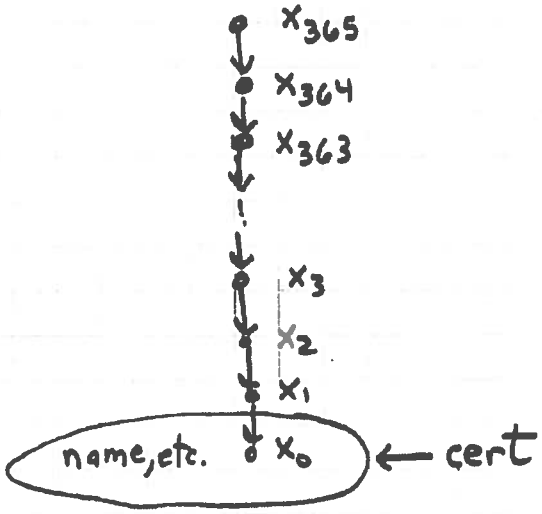
giving list of revoked cert serial #'s

(signed, of course)

CRL can get long!

Method ③: (due to Micali)

cert contains end point of a hash chain



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".

TOPIC

FILE UNDER:

DATE:

PAGE:

IBE - Identity-Based Encryption

User PK = (TTP key, 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
  $$Q_{ID} = H_1(ID) \in G^*$$
  $$d_{ID} = s \cdot Q_{ID} \quad \text{user SK}$$
- 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

TOPIC:

DATE:

FILE UNDER:

PAGE:

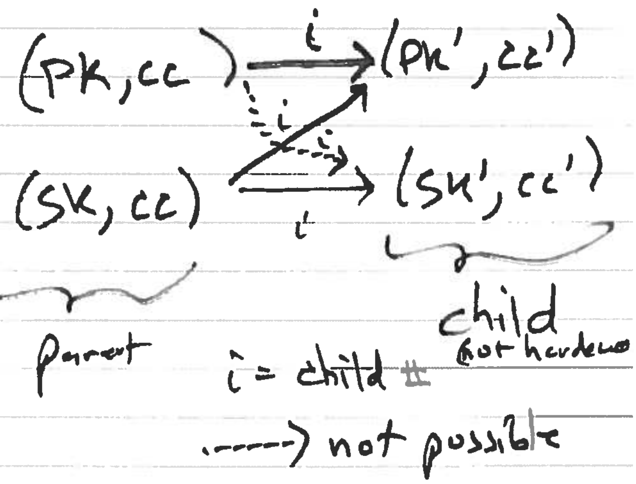
BIP-32 (Bitcoin Improvement Proposal)

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

Actually, (PK, chaincode) pair
↑ random #

(PK', cc')

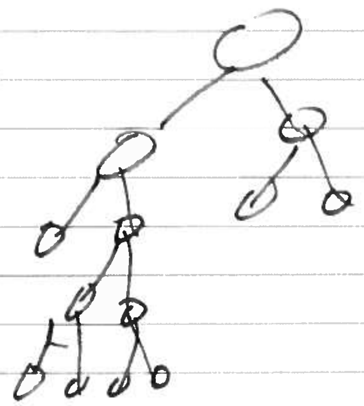
i = child #



Two kinds of children: hardened, & not

derive (PK', cc') from cc, PK, i

or cc, SK, i (hardened)



← leaves are hardened
can receive payments
can't make them.