Admin:

Project presentations: 5/8, 5/13, 5/15. [Writeups due 5/15. writeups ≈ 20 pages... 30 pgs]
guidance: 13 min/talk

Talk today on Bitcoin: 4 pm today 32G-882

Today:

Key establishment:

Direct

Server-assisted: Needham-Schroeder protocols
symmetric
asymmetric

Names

CertB: X.509

SPKI/SDSI

Key management / Key distribution

- themes: crypto, keys, names, individuals, trust, identity, scaling, usability, certificates, PKI, TTP's
- Crypto keys need to be shared/distributed to be useful.
- How is such sharing/distribution to be arranged?

① Directly – by physical meeting

Meet in private ⇒ no eavesdroppers (privacy)

Recognize each other ⇒ authentication

Share/exchange PK's or symmetric keys

Save in DB:

Alice

name	key
—	—
—	—
Bob	PK_B
—	—

...

Bob

name	key
—	—
Alice	PK_A
—	—
—	—

...

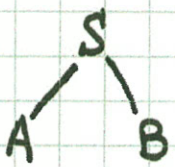
Note use of names to identify entries

Privacy not needed if keys are PK's.

Alice or Bob could be a computer (e.g. Alice installs key in computer Bob, or computer Alice gives user Bob her public key...)

Such direct meetings are necessary foundation of any approach to key mgt, as remote meetings don't provide either authentication or privacy naturally...

② Indirect / Two-link using TTP/server



Alice & Bob can't meet directly, but they have both previously met with S & exchanged keys.

They can use S to broker a "key setup" operation so that A & B end up sharing a key, more-or-less as if they had met directly.

But: they need to trust S to behave properly and setup protocol ("key exchange" protocol) needs to be well designed (tricky!)

Needham & Schroeder proposed two protocols: one for symmetric keys & one for public keys.

Needham-Schroeder Symmetric Key Exchange Protocol

Already shared:

K_{AS} – symmetric key shared between A & S

K_{BS} – " " " " B & S

Goal:

K_{AB} – " " " " A & B (and S)

Notations: (standard)

- N_A nonce generated by A

  N_B nonce generated by B

  "nonce" = a "use once" value (never repeats)

  could be counter, or long random value

  Nonces can protect against "replay attacks"

- $\{M\}_K$ = message M encrypted & authenticated with key K (e.g. derive K_1 & K_2 from K, encrypt M using AES in suitable mode, then append MAC_{K_2} of ciphertext).

  Literature often vague about properties of $\{\}_K$

≡ "authenticated encryption"

Symmetric Protocol:

① $A \rightarrow S: A, B, N_A$ ≡ "Hi, I'm Alice & I want to talk with Bob. N_A is my "request nonce" "

② $S \rightarrow A: \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$ ≡ "OK. Here's K_{AB} for you & a blob to give to B."

"a blob"

③ $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$ ≡ "Hi B. I'm A. Here's a blob" (Bob checks & decrypts blob)

④ $B \rightarrow A: \{N_B\}_{K_{AB}}$ ≡ "Hi A. Here's a challenge to prove you know K_{AB}."

⑤ $A \rightarrow B: \{N_B - 1\}_{K_{AB}}$ ≡ "That's easy. Here's $N_B - 1$ encrypted with K_{AB}."

Notes: A, B, and S know K_{AB}

S must be trusted! (S can pretend to be A to B or vice versa...)

Note roles of names: handles to identify parties, addresses to send messages to, text strings to put in messages

If no nonces (or perhaps timestamps), Adversary could replay earlier protocol to Alice or Bob...

Kerberos

Public-Key Protocol:

K_{PX}, K_{SX} ≡ public, secret key of party X

Assume: S knows K_{PA}, K_{PB}

A & B know K_{PS}

① $A \rightarrow S$: A, B ≡ "request from A for K_{PB}"

② $S \rightarrow A$: $\{K_{PB}, B\}_{K_{SS}}$ ≡ Signed "cert" for B's PK

③ $A \rightarrow B$: $\{N_A, A\}_{K_{PB}}$ ≡ "Hi! I'm A, with nonce N_A"

new

③' $B \rightarrow S$: B, A ≡ "request from B for K_{PA}"

③'' $S \rightarrow B$: $\{K_{PA}, A\}_{K_{SS}}$ ≡ Signed "cert" for A's PK

④ $B \rightarrow A$: $\{N_A, N_B\}_{K_{PA}}$ ≡ "hi" (note these are bound together, nonmalleable.)

⑤ $A \rightarrow B$: $\{N_B\}_{K_{PB}}$ ≡ "yep, I'm here!"

Note: 3' & 3'' could be removed by having cert 3'' inside ②

At end: only A & B know N_A & N_B; eavesdropper doesn't

Is this secure! How do you tell?

Attack found by automated analysis!

(Gavin Lowe found it in 1995 – 17 years later!)

① Intruder I gets A to initiate communication with I

② I passes knock $\{N_A, A\}$ on to B (after reencrypting with K_{pB})

③ B responds with $\{N_A, N_B\}_{K_{PA}}$, which I sends to A

④ A sends $\{N_B\}_{K_{PI}}$ to I. I decrypts & gets N_B

⑤ I sends $\{N_B\}_{K_{PB}}$ to B

Now B thinks he is sharing N_A & N_B only with A,

but I knows N_B. This is wrong.

Fix: ④ $B \to A : \{N_A, N_B, B\}_{K_{PA}}$

Moral: Be explicit in protocols!

(e.g. give session id both ways, & identities...)

Give hash of shared transcript in each message
(i.e. of all previous messages)...

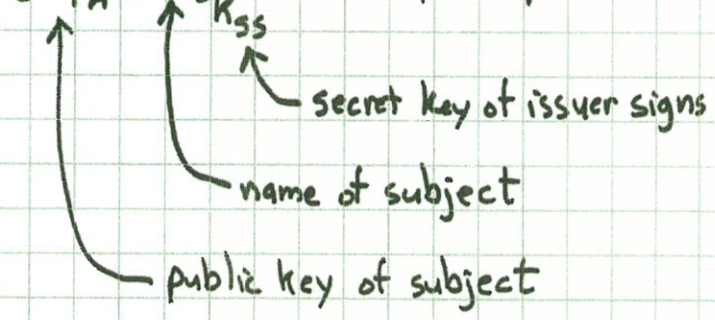
Automated analysis works!

Huge literature on key establishment protocols.

Certificates:

- Typically: a signed statement from an "issuer" about a "subject" & his public key
- In Needham-Schroeder:

$\{K_{PA}, A\}_{K_{SS}}$ is prototype certificate



S certifies that Alice's PK is K_{PA}

- Others can get this cert from S, from A, or from some online database.

(Kohnfelder's MIT bachelor's thesis, 1978)

- Scaling: how to scale from 100 users to 10^{10}?

Everything starts breaking:

- there is no one server everyone trusts
- one server can't handle load
- what are names ?

Names:

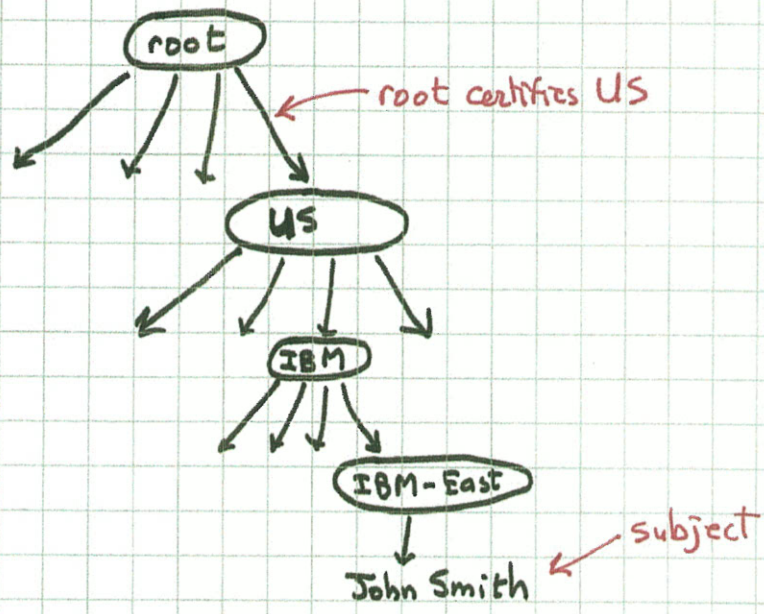
- How does Alice know Bob's name?
- Can Alice trust Bob when he says his name is ...
- Who ensures that names are unique?

  How is this done? Can it be manipulated/compromised?

  [Compare: email addresses...]
- If Alice can get Bob's (name or email address) correctly already, why can't she get his PK the same way?

One approach: Hierarchical naming system

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 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)

SPKI/SDSI

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

is really a global name

- Two types of certificates:

  name cert = associates name with key or another name (!)

  auth cert = gives permission (authorization) to subject(s)

- Name certs:

  Provide elegant & flexible "algebra" of names

  Allows one to describe groups (multivalued names)

cert = (K, A, X)

$K.A \Rightarrow X$

give as "rewrite rule"

K = issuer PK
A = identifier (name)
X = PK (key)
or PK.id (name)
or $PK.id_1.id_2 \ldots .id_k$

extended name

Examples: (let K be my PK)

$K.Bob \Longrightarrow K_0$ "I say Bob's PK is K_0"

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

$K.Bob \Rightarrow K_1.Bob\text{-}Smith$ "My Bob is the same as K_1's Bob Smith"

$K.Bob \Rightarrow K.mit.Bob\text{-}Smith$ "My Bob is same as mit's Bob Smith"

$K.mit \Rightarrow K_2$

Inference

$K.Bob \Rightarrow K_2.BobSmith$ inferred rule

Groups: $K.friends \Rightarrow K.Alice$
$K.friends \Rightarrow K.Bob$
} group members

...

Let K_2 = mit PK

$K_2.faculty \Rightarrow K_2.eecs.faculty$

$K_2.faculty \Rightarrow K_2.math.faculty$

⋮

(group defined as union of other groups)

Authorization certs & ACL's

ACL = access control list

Can put name on ACL for a resource:

"Only individuals in K.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] ⇒ K.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

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)

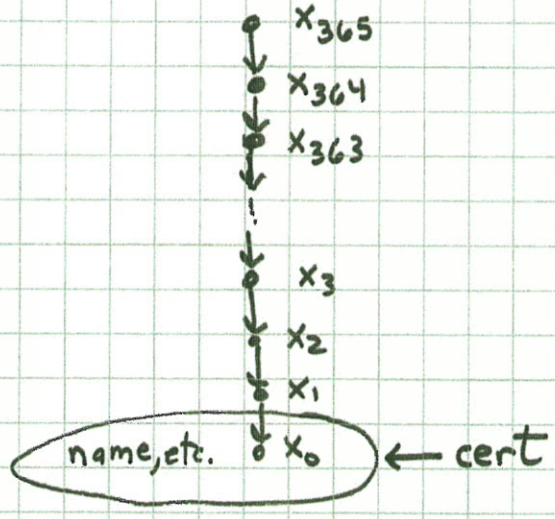
Puts heavy load on server!

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".