Admin: Quiz in-class Wed April 6th

Coverage: through Monday April 4th's lecture.

Closed book but open notes. (Your own notes & posted notes only.)

Outline:

- Secret-sharing with short shares
- Key establishment
  - Direct
  - Server-assisted: symmetric, asymmetric } Needham-Schroeder protocols

Next time: Large scale PKI:

X.509

SPKI/SDSI

cert revocation

Note: We have $0 \le m < p$ so each share is "as big as" m.
Shares are (i, y_i); each y_i "same size as" p or m.
If file is very large: can break into pieces, share each piece. (bytes) GF(2^8)
But still, each person gets total share as big as original m.
How to reduce share sizes?

Example: I have 100 MB file to share among $n=20$ friends;
I want $t=10$ to be able to reconstruct it.
With original scheme, each gets 100 MB share.
Goal: each has 100MB/t = 10 MB share, ?

Idea 1: Give up information-theoretic (unconditional) security for computational security.

Idea 2: Encrypt message, then share ciphertext, & share key.
of ciphertext,

Idea 3: In secret-sharing, secret is set of all t coefficients, not just constant term. (No longer info-theoretic secure; each point on curve eliminates some polynomials. t shares reduces possibilities from p^t down to 1.
Each share reduces # polynomials by factor of p.)

Given (M, n, t)

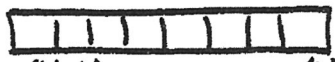
M = message to be shared (100 MB)

AES key $= s$ (128 bits)

Share $s \Rightarrow$ n shares $s_1, s_2, \ldots, s_n$
s.t. any t can reconstruct s
(info-theoretically secure)
Give i^{th} party s_i, $1 \le i \le n$. (and i)

Encrypt: $C = AES_s(M)$ (100 MB)

Break C into chunks $C_1, C_2, \ldots, C_\ell$ (each t byte)
$\ell = 100\text{ MB}/t$

Let $C' = C_k =$  t bytes

over $GF(2^8)$

$f_k(x) = a_0^{(k)} + a_1^{(k)} x + \cdots + a_{t-1}^{(k)} x^{t-1}$ [$a_1^{(k)}$ not random bytes used as coefs]

Give party i share $f_k(i)$ of k^{th} chunk
1 byte long, chunk is size t bytes

Total size of party i's share = 128 bits (for s_i)
$+$ 10 bits (for i)
$+$ $100\text{MB}/t$ for $f_1(i), \ldots, f_\ell(i)$
$\approx 100\text{ MB}/t$

Reconstruction: t parties
reconstruct s
reconstruct $f_k(x)$ for $k = 1, 2, \ldots, \ell$
$\Rightarrow C$
decrypt $\Rightarrow M$

Thm: If encryption function (e.g. AES) is secure, then so is this secret sharing scheme. Even if adversary gets C, he gets no information on M. (aside from length)

Threshold crypto (fully distributed)

- Can we avoid having secret s (in secret sharing of key) ever exist?
  - Dealing – make up shares s_i, s is implicitly generated; never explicitly exists
  - Reconstruction – don't reconstruct s
    instead, use shares of key to produce shares of signature, then reconstruct signature

How about RSA? [Fouque/Stern paper]

- generate $PK = (n, e)$ in distributed manner
- no one party ever knows factorization $n = p \cdot q$
- yet each party has share d_i of decryption key
  yet p & q exist & have suitable properties (size, etc.)
- given message m, each party can compute share $\sigma(d_i, m)$ of signature
- signature shares can be combined in public manner to produce $\sigma(m)$

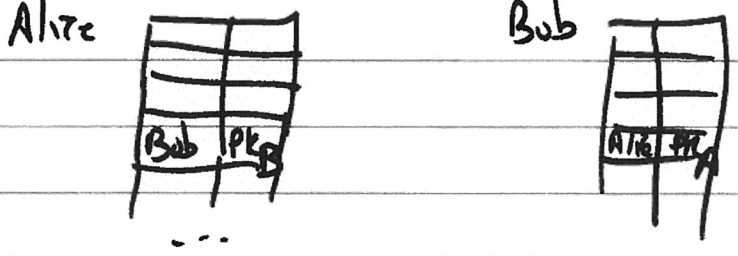
Key management / key distribution

themes: crypto, keys, names, individuals, trust, identity, scaling, usability, certificates, PKI, trusted intermediaries

- keys need to be shared to be useful (at least PK part for PK crypto)
- how is such sharing to be arranged?

- Directly (by physical mtg)

meet in private (no eavesdroppers)
recognize each other (authentication)
share PK's, or symmetric keys
save in database: (when Alice has > 1 contact...)



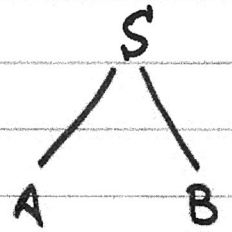
note appearance of names tied to entries...

privacy not needed if PK's are exchanged (as opposed to symm. keys)

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 key mgt, as we'll see...

Indirect / Two-link / TTP (trusted third party) or server



Alice & Bob can't meet in person, but they have each met with trusted third party S & exchanged keys.

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

However, as we'll see, they need to trust S to behave properly & setup protocol (aka "key exchange" needs to be well designed).

Needham & Schroeder proposed 2 such protocols: one for symmetric keys, and one for public keys.

NS Symmetric Protocol

K_{AS} = key shared between A & S
K_{BS} similarly for B & S
} already set up

"nonce" means a "use once" value

N_A nonce generated by Alice
N_B " " " Bob

could be from a counter, or a long random value...
Used to protect against certain forms of "replay attack"...

K_{AB} : key that gets setup here between A, B (and S)

$\{M\}_K$: message M encrypted & authenticated with key K
(e.g. encrypt M using AES in suitable mode, & key K_1,
append $MAC_{K_2}(M)$, where K_1 & K_2 derived from K)
(literature often is vague about properties of $\{\}_K$...)

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 $S \rightarrow A$: $\{N_A, K_{AB}, B, \underbrace{\{K_{AB}, A\}_{K_{BS}}}_{\text{blob}}\}_{K_{AS}}$ — Ok here's key K_{AB} & blob to give B

③ $A \rightarrow B$: $\{K_{AB}, A\}_{K_{BS}}$ — A knocks on B's door; B decrypts & checks blob

④ $B \rightarrow A$: $\{N_B\}_{K_{AB}}$ — B challenges A with K_{AB}

⑤ $A \rightarrow B$: $\{N_B - 1\}_{K_{AB}}$ — A responds

- Who knows K_{AB}? A, B, S
- S must be <u>trusted</u>: can pretend to be A to B or vice versa...
- note roles of names: handles by which to identify parties
  addresses to which msgs can be sent
  text strings that can be included in messages
- if no nonces in ①, ② & cut out ④, ⑤: could replay earlier session to A or B (can do same if K_{AB} later compromised...) → fix with <u>timestamps</u> (Kerberos)

PK protocol

K_{PX}, K_{SX} public & secret keys of X
S has K_{PA}, K_{PB}
A & B have K_{PS}

6.857 Rivest
4/1/09 L15.5

① $A \to S$: A, B — PK req

② $S \to A$: $\{K_{PB}, B\}_{K_{SS}}$ — signed "cert" for B's PK

③ $A \to B$: $\{N_A, A\}_{K_{PB}}$ — knock
encrypted, bound together
"psst! I'm A, and here's N_A... tell me"

③' $B \to S$: B, A — PK req

③'' $S \to B$: $\{K_{PA}, A\}_{K_{SS}}$ — signed "cert" for A's PK

(③' and ③'' are new / inserted)

④ $B \to A$: $\{N_A, N_B\}_{K_{PA}}$ — encrypted, bound together (non malleable...)

⑤ $A \to B$: $\{N_B\}_{K_{PB}}$ — yep, I'm really here...

- ③' & ③'' could be replaced by including blob/cert $\{K_{PA}, A\}_{K_{SS}}$ in ②
- at end only A & B know N_A & N_B; eavesdroppers don't...

Attack! (Gavin Lowe) 17 yrs later!
automated analysis

intruder I gets A to initiate communication with I

- then passes knock $\{N_A, A\}$ on to B (after re-encrypting with K_{PB})
- B responds with $\{N_A, N_B\}_{K_{PA}}$, which I sends to A
- A sends $\{N_B\}_{K_{PI}}$ to I. I decrypts it & gets N_B
- 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. WRONG

Fix: (4) $B \rightarrow A: \{N_A, N_B, B\}$

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...)

Huge literature on such key-establishment protocols...