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 \( \rightarrow \) asymmetric \( \rightarrow \) Needham-Schroeder protocols

Next time: Large scale PKI:
- X.509
- SPKI/SDSI
- cert revocation
Note: We have $0 \leq m < p$, 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. (byte)
Put still, each person gets share as big as original file.

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 $100 \text{MB}/t = 10 \text{MB}$ 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 information-theoretic
secure; each point on curve eliminates some polynomials.
$t$ shares reduces possibilities from $p^t$ down to 0.]
Each share reduces # polynomials by factor of $p$.}
Given \((M, n, t)\)

- \(M\) = message to be shared (100 MB)
- \(t\) = AES key (128 bits)

Share \(s = n\) shares \(s_1, s_2, \ldots, s_n\)

- s.t. any \(t\) can reconstruct \(s\) (info-theoretically secure)

Give i-th party \(s_i, 1 \leq i \leq n\). (and i)

Encrypt:

\[ C = \text{AES}_k (M) \] (100 MB)

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

Let \(C' = C_k = \begin{array}{cccccccc} a_0 & a_1 & a_2 & \cdots & a_{t-1} \end{array} \) t bytes

- over GF(2^8)

- \[ f_k(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \] not random

- [all bytes used as coeffs]

Give party i share \(f_k(i)\) of k-th chunk

- 1 byte long, chunk by size \(t\) bits

Total size of party i's share = 128 bits (for \(s_i\))

- + 10 bits (for i)

- + 100MB/t for \(f_1(i), \ldots, f_l(i)\)

\[ \approx 100 \text{ MB}/t \]

Reconstruction: t parties

- reconstruct \(\hat{s}\)

- reconstruct \(f_k(x)\) for \(k = 1, 2, \ldots, l\)

\[ \Rightarrow C \]

- decrypt \(\Rightarrow M\)

### Claim

If encryption function (e.g. AES) is secure, then so is this secret-sharing scheme. Even if adversary gets \(C\), he asks no information on \(M\) (aside from truth)
Threshold crypto (fully distributed)

- Can we avoid having secret $s$ (in secret-sharing or $k_y$) ever exist?
  - Dealing: make up shares $s_i$, $s$ is implicitly generated; never explicitly exists
  - Reconstruction: don’t reconstruct $s$ instead, use shares $t_{s,a} \cdot t_{y,k}$ to produce shares of signature, then reconstruct signature

How about RSA? [Fouque/Stein paper]

- generate $PK = (n, e)$ is distributed manner
- no one path ever knows factorization $n = p \cdot q$
- yet each path has share $d_i$ of decryption key
- yet $p$ & $q$ exist & have suitable properties (size, etc.)
- given message $m$, each path can compute share $\sigma_i \cdot (d_i \cdot m)$ 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 pub for PK crypt)
  - how is such sharing to be arranged?

- **Directly** (by physical mtg)
  
  Alice    Bob

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

  Alice    Bob

  Bob PK

  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 mgmt, as we'll see.

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

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

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

"Nonce" means a "use once" value

\[ N_A \text{ nonce generated by Alice} \]
\[ N_B \text{ """" Bob} \]

could be from a counter, or a long random value...

Used to protect against certain forms of "reply attack"...

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

\[ \text{\{ } M \text{\} }_k : \text{message } M \text{ encrypted & authenticated with by } k \]
(e.g., encrypt M using AES in suitable mode, & by k,
append \( \text{MAC}_{K_a}(M) \), where \( K_a \) & \( K_b \) derived from K)

(literature often is vague about properties of \( \text{\{ } 3 \text{\} } \))
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. $S \rightarrow A : \{N_A, K_{AB}, B, \{K_{AB}, A^3\}_{K_{BS}}, \text{blob}\}$
   - Oh here's key $K_{AB}$ & blob to give B

3. $A \rightarrow B : \{K_{AB}, A^2\}_{K_{BS}}$
   - A knocks on B's door
   - B decrypts & checks blob

4. $B \rightarrow A : \{N_B^2\}_{K_{AB}}$
   - B chills, A with $K_{AB}$

5. $A \rightarrow B : \{N_B^{12}\}_{K_{AB}}$
   - A responds

- Who knows $K_{AB}$? A, B, S

- $S$ must be trusted: 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, 4, 5: could replay earlier session to A or B
  (can do same if $K_{AB}$ later compromised...) → fix with timestamps
  (Kerberos)
PK protocol

1. A → S: A, B  
   (PK req)

2. S → A: \{K_{PB}, B^2\}_{K_{SS}}  
   (signed "cert" for B's PK)

3. A → B: \{N_A, A^2\}_{K_{PB}}  
   (Knock)
   (encrypted, bound together)
   "post! I'm A, and here's N_A..."
   (non-malleable...)

3' B → S: B, A  
   (PK req)

3'' S → B: \{K_{PA}, A^2\}_{K_{SS}}  
   (signed "cert" for A's PK)

4. B → A: \{N_A, N_B^2\}_{K_{PA}}  
   (encrypted, bound together)

5. A → B: \{N_B^2\}_{K_{PB}}  
   (yep, I'm really here...)

- 3' & 3'' could be replaced by including blob/"cert" \{K_{PA}, A^2\}_{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 comm with I
  o then passes knock $E_{N_A, A^3}$ on to B (after re-encrypt w/ $K_{PB}$)
  o B responds with $E_{N_A, N_B^3}^{K_{PA}}$, which I sends to A
  o A sends $E_{N_B^3}^{K_{PI}}$ to I. I decrypts it & gets $N_B$
  o I sends $E_{N_B^3}^{K_{PB}}$ to B

Now B thinks he is sharing $N_A$ & $N_B$ only with A, but I knows $N_B$, WRONG

Fix: ① $B \rightarrow A$: $E_{N_A, N_B, B^3}$

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...