Key Management/Distribution

Administrivia

- Snafu on books
  - Probably best to buy it elsewhere
- Paper assignment and first homework
  - Next week (9/24)

Cryptography in Use

- Provides foundation for security services
- But can it bootstrap itself?
  - Must establish shared key
  - Straightforward plan
    - One side generates key
    - Transmits key to other side
    - But how?

Two Problems

- Peer-to-peer key sharing
- Prob 1: Known peer, insecure channel
- Prob 2: Secure channel, unknown peer

Security Through Obscurity?

- Caesar ciphers
  - Very simple permutation
  - Only 25 different cases
  - Relies strictly on no one knowing the method

Passwords

- Reduces permutation space to key space
- Caesar cipher: one-letter "key"
- 10-letter key for MSC reduces 26! ($\sim 4 \times 10^{20}$) to ${}_{26}C_{10}$ ($\sim 2 \times 10^{13}$)
- 8-byte key for DES reduces $2^{64}!$ ($\sim 10^{10^{200}}$) to 2^{56} ($\sim 10^{17}$)

The Enigma Machine

- Broken first by Polish, then by English
  - Not as easily as widely regarded
- Weaknesses in key distribution
  - Day keys plus scramblers
  - "Session keys" encrypted in duplicate
  - Enigma did not use OFB/CFB

Peer-to-Peer Distribution

- Technically easy
- But it doesn't scale
  - Hundreds of servers…
  - Times thousands of users…
  - Yields ~ million keys
- Centralized key server
  - Needham-Schroeder

Basic Idea

- User sends request to KDC: {s}
- KDC generates a random key: $K_{c,s}$
  - Encrypted twice: $\{K_{c,s}\}K_c$, $\{K_{c,s}\}K_s$
  - Typically called ticket and credentials, resp
  - Ticket forwarded with application request
- No keys ever traverse net in the clear

Problem #1

- How does user know session key is encrypted for the server? And vice versa?
- Attacker intercepts initial request, and substitutes own name for server
  - Can now read all of user's messages intended for server

Solution #1

- Add names to ticket, credentials
  - Request looks like {c, s}
  - $\{K_{c,s}, s\}K_c$ and $\{K_{c,s}, c\}K_s$, resp
- Both sides can verify intended target for key sharing
- This is basic Needham-Schroeder

Problem #2

- How can user and server know that session key is fresh?
- Attacker intercepts and records old KDC reply, then inserts this in response to future requests
  - Can now read all traffic between user and server

Solution #2

- Add nonces to ticket, credentials
  - Request looks like {c, s, n}
  - $\{K_{c,s}, s, n\}K_c$ and $\{K_{c,s}, c, n\}K_s$
- Client can now check that reply made in response to current request

Problem #3

- User now trusts credentials
- But can server trust user?
- How can server tell this isn't a replay?
- Legitimate user makes electronic payment to attacker; attacker replays message to get paid multiple times
  - Requires no knowledge of session key

Solution #3

- Add challenge-response
  - Server generates second random nonce
  - Sends to client, encrypted in session key
  - Client must decrypt, decrement, encrypt
- Effective, but adds second round of messages

Problem #4

- What happens if attacker does get session key?
  - Answer: Can reuse old session key to answer challenge-response, generate new requests, etc

Solution #4

- Replace (or supplement) nonce in request/reply with timestamp [Denning, Sacco]
  - $\{K_{c,s}, s, n, t\}K_c$ and $\{K_{c,s}, c, n, t\}K_s$, resp
  - Also send $\{t\}K_{c,s}$ as authenticator
    - Prevents replay without employing second round of messages as in challenge-response

Problem #5

- Each client request yields new known-plaintext pairs
- Attacker can sit on the network, harvest client request and KDC replies

Solution #5

- Introduce Ticket Granting Server (TGS)
  - Daily ticket plus session keys
  - (How is this different from Enigma?!)
- TGS+AS = KDC
  - This is modified Needham-Schroeder
  - Basis for Kerberos

Problem #6

- Active attacker can obtain arbitrary numbers of known-plaintext pairs
  - Can then mount dictionary attack at leisure
  - Exacerbated by bad password selection

Solution #6

- Preauthentication
  - Establish weak authentication for user before KDC replies
  - Examples
    - Password-encrypted timestamp
    - Hardware authentication
    - Single-use key

Public Key Distribution

- Public key can be public!
  - How does either side know who and what the key is for? Private agreement? (Not scalable.)
- Must delegate trust
  - Why?
  - How?

Certification Infrastructures

- Public keys represented by certificates
- Certificates signed by other certificates
  - User delegates trust to trusted certificates
  - Certificate chains transfer trust up several links

Examples

- PGP
  - "Web of Trust"
  - Can model as connected digraph of signers
- X.500
  - Hierarchical model: tree (or DAG?)
  - (But X.509 certificates use ASN.1!