Authentication Stuff

Shankar

May 28, 2013
Overview

- Principals (app clients/servers) attach to a network (eg, Internet) establish sessions (over udp/tcp), exchange data, close, detach from channel

- Attackers
  - Network: passive (listen only) or active (intercept/send msgs)
  - Endpoint: OS, apps, memory

- Authentication goals:
  - “Session” authentication
    - Ensure that session peers are who they say they are
  - “Data” authentication
    - Establish session key(s) for data confidentiality/integrity
  - Session authentication w/o data authentication
    - relevant in certain situations
Typical scenario without authentication

client
tcp
server

A

B

<ip addr, port #>

connect(x2)

open
tx(data)

rx()
data

close
closing
closed

open
tcp closing
closing
closed

open to x1

accept()

open
tx(data)

rx()
data
close
closing
closed

[x1, x2, SYN, ...]

[x2, x1, SYN−ACK, ...]

[x1, x2, ACK, ...]

[x1, x2, ..., [data]]

[x2, x1 ..., [data]]
Typical scenario with attacks

- **Client** (A)
  - `connect(x2)`
  - `open`
  - `tx(data)`
  - `rx()`
  - `data`
  - `close`

- **TCP**
  - `<ip addr, port #>`
  - `[x1, x2, SYN, ...]`
  - `[x2, x1, SYN-ACK, ...]`
  - `[x1, x2, ACK, ...]`
  - `[x2, x1, ..., [data]]`
  - `[x1, x2, ..., [data]]`
  - tcp closing

- **Server** (B)
  - `accept()`
  - `open to x1`
  - `tx(data)`
  - `rx()`
  - `data`
  - `close`

**Attack Scenarios**

- **Endpoint attacks**
  - `open`
  - `close`

- **Network attacks**
  - `connect(x2)`
  - `tx(data)`
  - `rx()`
  - `close`
Typical scenario with authentication

Client A

- connecting
- open
- closing
- closed

TCP x1

- connect(x2)
- tcp conn establishment
- tcp data exchange
- tcp closing

TCP x2

- accept()
- open to x1
- authenticated protocol
- authenticated data exchange
- close

Server B

- closing
- closed
Types of Attacks

- Attacks can span multiple types over long durations
- Authentication mechanism should state the attacks it handles

- Network-based attacks (roughly in order of increasing difficulty)
  - Sending arbitrary messages, with incorrect fields
  - Eavesdropping: observing messages in the channel
  - Intercepting messages, changing them, resending them.
  - Easier in WLANs and LANs than wired point-to-point links

- Endpoint attacks (roughly in order of increasing difficulty)
  - Sending arbitrary messages at an endpoint and receiving replies
  - Obtain old/current keys from password files, ...
  - Overrun endpoint app, OS, memory, ...
    - Handled by OS mechanisms, not authentication protocols
  - Not covered here
Types of Attacks II

- “Weak” secret (aka “low-quality” secret)
  - Comes from a space small enough for a brute-force search
  - Eg: Passwords, and keys obtained from them
- “Strong” secret (aka “high-quality” secret): not weak
  - Eg: Key with 128 random bits

- Dictionary attacks (aka password-guessing attacks)
  - Given ciphertext from a weak key and structured plaintext
    apply every possible key on ciphertext until structure appears
  - Not doable if $K$ is strong
  - Online attack: interact with authenticator at every guess
    - Defense: limit number/frequency of attempts
  - Offline attack: interact with authenticator just once
    - Defense: don’t expose relevant ciphertext
“Brute-force” denial-of-service (DOS) attack
- Overload endpoints (usually servers) with excess traffic
- Defenses:
  - increase server resources
  - reject traffic, preferably selectively (statistics, ISP, …)
  - make attacker do more work

“Asymmetric” denial-of-service (DOS) attack
- Exploit flaws in endpoint logic to make endpoints enter erroneous states in which they make no progress

Not covered here in any depth
Conventions: Messages

- Messages are tuples of one or more fields; eg, [23, ['ab7']]
- Fields indexed from 0; eg,
  - $msg[0]$ is 23  $msg[1]$ is ['ab7']  $msg[1][0]$ is 'ab7'
- $rcv \ msg$: get any rcvd msg into $msg$
- $rcv [x, y, z]$: get fields of any rcvd 3-tuple msg into $x$, $y$, $z$
- $rcv [A, z]$, for constant $A$: get field 1 of any rcvd 2-tuple msg of form $[A, \cdot]$
- $[x, y, z] \leftarrow msg$: assigns fields of 3-tuple $msg$ to $x$, $y$, $z$
  - fails if $msg$ not 3-tuple
Conventions: Crypto

- Secret-key encryption and decryption
  - \( \mathcal{E}(msg, key) \): encrypt \( msg \) with \( key \)  
    // includes any IV
  - \( \mathcal{D}(ctx, key) \): decrypt \( ctx \) with \( key \)  
    // \( ctx \) includes any IV

- Hash
  - \( \mathcal{H}(msg) \): hash of \( msg \)  
    //eg, HMAC using SHA-1
  - \( \mathcal{H}(msg, key) \): keyed-hash

- Public-key crypto
  - Let \([pri, pub]\) be a public-key pair
  - \( \mathcal{E}_P(msg, pub) \): encrypt \( msg \) (with public key)
  - \( \mathcal{D}_P(msg, pri) \): decrypt \( msg \) (with private key)
  - \( \text{Sign}(msg, pri) \): signature of \( msg \) (using private key)
  - \( \text{Vrfy}(msg, s, pub) \): verify signature \( s \) of \( msg \) (with public key)
Nonces

- Nonce: new values; generation can be predictable or random
  - Predictable: given one value, attacker can guess the next one
  - Random: not predictable // physical randomness, crypto output
Authentication protocols: Overview

- Special case: One-way session authentication
  - server authenticates client

- General case: Two-way session authentication + session key

- Misc: Authenticating humans, Strong-password protocols

- Scaling to many principals and domains
  - secret-key: key distribution center (KDC)
  - public-key: certification authority (CA)

- Kerberos
- SSL
- IPsec
One-way authentication
One-way authentication: Password; No Crypto

- $pwd_c$ at server holds c’s password, for every client $c$.

<table>
<thead>
<tr>
<th>client $A$ (has $pw$)</th>
<th>server $B$ (has $pwd_A = pw$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send $[A, B, pw]$</td>
<td>rcv $[A, B, z]$</td>
</tr>
<tr>
<td></td>
<td>if ($z \neq pwd_A$) FAIL</td>
</tr>
<tr>
<td></td>
<td>// peer authenticated as $A$</td>
</tr>
</tbody>
</table>

- Channel eavesdropper gets password
  - So use only with secure channel

- Exposure of $pwd$ file reveals all passwords
  - defense: encrypt $pwd$ file with a strong key
One-way authentication: Password; No Crypto at Client

- $hpw_c$ at server holds hash of $c$’s pwd, for every client $c$.

<table>
<thead>
<tr>
<th>client $A$ (has $pw$)</th>
<th>server $B$ (has $hpw_A = \mathcal{H}(pw)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send $[A, B, pw]$</td>
<td>rcv $[A, B, z]$</td>
</tr>
<tr>
<td></td>
<td>if ($\mathcal{H}(z) \neq hpw_A$) FAIL</td>
</tr>
</tbody>
</table>

- Server forgets $z$ after processing msg
- Channel eavesdropper gets password
  - So use only with secure channel
- Exposure of $hpw$ file: need dictionary attacks to reveal passwords
  - Use random “salt”: $hpw_A = [salt, \mathcal{H}(pw|salt)]$
    to ensure each dictionary attack limited to one user
  - Defense: encrypt $hpw$ file with a strong key
One-way authentication: Lamport Hash Scheme – 1

- Server $B$ has
  - $n_A$: # of logins remaining, initially say 1000
  - $hpw_A$: $n_A$-fold hash of $pw$

<table>
<thead>
<tr>
<th>client $A$ (has $pw$)</th>
<th>server $B$ (has $n_A$ and $hpw_A$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send $[A, B, \text{conn}]$</td>
<td>rcv $[A, B, \text{conn}]$</td>
</tr>
<tr>
<td>rcv $[A, B, n]$</td>
<td>send $[B, A, n_A]$</td>
</tr>
<tr>
<td>send $[A, B, \mathcal{H}^{n-1}(pw)]$</td>
<td>rcv $[A, B, y]$</td>
</tr>
<tr>
<td>if ($\mathcal{H}(y) \neq hpw_A$) FAIL</td>
<td>$[hpw_A, n_A] \leftarrow [y, n_A - 1]$</td>
</tr>
</tbody>
</table>

- Need dictionary attack to get $pw$ from either channel eavesdropping or stealing $B$’s client info
One-way authentication: Lamport Hash Scheme – II

- When $n_A$ becomes 1, need to reset with new $hpw_A$ and $n_A$

- Reset option 1:
  - $A$ chooses new $[hpw, n]$ and sends it to $B$ unencrypted.
  - Adequate assuming $B$-to-$A$ authentication is not needed?

- Reset option 2:
  - $A$ sends new $[hpw, n]$ encrypted by a Diffie-Helman key.
  - Is this any better?

- Small $n$ attack:
  - Attacker responds to $[A, B, conn]$ with $[B, A, m]$ where $m < n_A$
  - $A$ responds with $H^{m-1}(pw)$
  - Attacker can authenticate itself as $A$ for $n_A - m$ logins

- SKEY: Internet deployed version of Lamport’s hash scheme
One-way authentication: Secret-key; Open Challenge

- $key_c$ at server holds $c$’s key, for every client $c$

<table>
<thead>
<tr>
<th>Client $A$ (has key $K$)</th>
<th>Server $B$ (has $key_A = K$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send $[A, B, \text{conn}]$</td>
<td>Recev $[A, B, \text{conn}]$</td>
</tr>
<tr>
<td>Recev msg $[B, A, x]$</td>
<td>Generate random $r_B$ // nonce</td>
</tr>
<tr>
<td>Send $[A, B, \mathcal{E}(x, K)]$</td>
<td>Send $[B, A, r_B]$</td>
</tr>
<tr>
<td>Recev msg $[A, B, y]$</td>
<td>If $(y \neq \mathcal{E}(r_B, key_A))$ FAIL</td>
</tr>
</tbody>
</table>

- Here: $r_B$ is the challenge and $\mathcal{E}(r_B, key_A)$ is the response
- $\mathcal{H}(.)$ (keyed-hash) can be used instead of $\mathcal{E}(.)$
- If $r_B$ is predictable, attacker can authenticate itself as $A$ (How?)
- Dictionary attack doable if $K$ is weak and attacker can eavesdrop or attach to $B$’s net address.
One-way authentication: Secret-key; Hidden Challenge

- Configuration as before but now
  - challenge: $\mathcal{E}(r_B, key_A)$ for random $r_B$
  - response: $r_B$

- $\mathcal{H}(.)$ (keyed-hash) cannot be used instead of $\mathcal{E}(.)$

- If $r$ is predictable, attacker can authenticate itself as $A$

- Dictionary attack doable if $K$ is weak and
  - attacker can eavesdrop, or
  - attacker can attach to $B$’s net address, or
  - $R$ has structure (eg, $[B, random]$)

- Hidden Challenge and Hidden Response
  - challenge: $\mathcal{E}(r_B, key_A)$ for random $r_B$
  - response: $\mathcal{E}(r_B + 1, key_A)$
One-way authentication: Secret-key; Timestamp-based

- A and B have clocks that are within D seconds of each other

<table>
<thead>
<tr>
<th>client A (has K, clk_A)</th>
<th>server B (has key_A = K, clk_B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send [A, B, conn, enc(clk_A, K)]</td>
<td>rcv [A, B, conn, x]</td>
</tr>
<tr>
<td>if (</td>
<td>clk_B − D(x, key_A)</td>
</tr>
</tbody>
</table>

- Single transmission suffices
- Attacker can authenticate itself as A
  - within duration of clock skew D
    (defense: B stores every ts from A in last D seconds)
  - if K used with multiple servers
    (defense: include server replica id with timestamp)
  - if B’s clock is set back (or A’s clock is set forward)
- Replacing E by H causes much more work for B
  - Can fix by including unencrypted timestamp in conn msg?
  - Dictionary attack?
One-way authentication: Public-key

- **Configuration**
  - client $A$ has public-key pair $[pri, pub]$  
  - At server $B$, entry $pub_A$ holds $pub$

- **Open challenge, hidden response**
  - challenge: random $r_B$  
  - response: $Sign(r_B, A)$  
  - Can $r_B$ be predictable (instead of random)?  
  - Dictionary attack?

- **Hidden challenge, open response**
  - challenge: $E_P(r_B, A)$ for random $r_B$  
  - response: $r_B$ or $Sign(r_B, A)$
Authentication: two-way + session key
Secret-key $K$; Server Challenges First – I

- response $E(r_B, K) \leftarrow$ server challenge $r_B$
- client challenge $r_A \rightarrow$ response $E(r_A, K)$
- session key: $F(r_A, r_B, K)$  
  \hspace{2cm} // one-way fn in $K$, eg, $H$

<table>
<thead>
<tr>
<th>Client A (has key $K$)</th>
<th>Server B (has key $A = K$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send $[A, B, \text{conn}]$</td>
<td>rcv $[A, B, \text{conn}]$</td>
</tr>
<tr>
<td></td>
<td>generate random $r_B$</td>
</tr>
<tr>
<td>rcv $[B, A, x_B]$</td>
<td>send $[B, A, r_B]$</td>
</tr>
<tr>
<td>generate random $r_A$</td>
<td></td>
</tr>
<tr>
<td>send $[A, B, r_A, E(x_B, K)]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rcv $[A, B, x_A, z_B]$</td>
</tr>
<tr>
<td></td>
<td>if ($z_B \neq E(r_B, key_A)$) FAIL</td>
</tr>
<tr>
<td></td>
<td>send $[B, A, E(x_A, key_A)]$</td>
</tr>
<tr>
<td></td>
<td>session key $S_B \leftarrow F(x_A, r_B, \text{key}_A)$</td>
</tr>
<tr>
<td>rcv $[B, A, z_A]$</td>
<td></td>
</tr>
<tr>
<td>if ($z_A \neq E(r_A, K)$) FAIL</td>
<td>session key $S_A \leftarrow F(r_A, x_B, K)$</td>
</tr>
</tbody>
</table>
Secret-key $K$; Server Challenges First – II

- Usual variations of open/hidden challenges/responses
- Dictionary attacks if $K$ is weak and
  - attacker can eavesdrop, or
  - attacker can attach to $B$’s net address
- What happens if client challenges first?
Secret-Key; Client Challenges First – I

<table>
<thead>
<tr>
<th>client $A$ (has key $K$)</th>
<th>server $B$ (has $key_A = K$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send $[A, B, \text{conn}, r_A]$</td>
<td>rcv $[A, B, \text{conn}, x_A]$</td>
</tr>
<tr>
<td>rcv $[B, A, x_B, z_A]$</td>
<td>send $[B, A, r_B, \mathcal{E}(x_A, key_A)]$</td>
</tr>
<tr>
<td>if ($r_A \neq \mathcal{D}(z_A, K)$) FAIL</td>
<td>rcv $[A, B, z_B]$</td>
</tr>
<tr>
<td>send $[A, B, \mathcal{E}(x_B, K)]$</td>
<td>if ($r_B \neq \mathcal{D}(z_B, key_A)$) FAIL</td>
</tr>
<tr>
<td>session key $F(r_A, x_B, K)$</td>
<td>session key $F(x_A, r_B, key_A)$</td>
</tr>
</tbody>
</table>

- Dictionary attack w/o network attack: send $msg1$, get $msg2$
- Reflection attack if $B$ can serve many clients simultaneously
  - attacker sends $[A, B, \text{conn}, n_A]$; gets $[B, A, n_B, \mathcal{E}(n_A, K)]$
  - attacker sends $[A, B, \text{conn}, n_B]$; gets $[B, A, n'_B, \mathcal{E}(n_B, K)]$
  - attacker can now respond to first request
Secret-Key; Client Challenges First – II

- client challenge $r_A \rightarrow$ response $\mathcal{E}(r_A, K)$
- response $\mathcal{E}(r_B, K) \leftarrow$ server challenge $r_B$

- Defending against reflection attack
  - $B$ remembers $r_B$ and does not accept it
    - difficult with replicated servers
  - response includes structure, eg, indicating sender:
    - $A$’s response: $\mathcal{E}([A, r_B], K)$
  - Use different keys for each direction, say $K$ and $K'$
    - $K'$ can be related to $K$: eg, $K + 1, -K, \bar{K}$

- Thumb-rule: Initiator should be first to authenticate itself
A and B have clocks that are within \( D \) seconds of each other

<table>
<thead>
<tr>
<th>Client A (has key ( K ), ( clk_A ))</th>
<th>Server B (has ( key_A = K ), ( clk_B ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ts \leftarrow clk_A )</td>
<td>( ts \leftarrow clk_A )</td>
</tr>
<tr>
<td>send ([A, B, conn, enc(ts, K)])</td>
<td>rcv ([A, B, conn, x_A])</td>
</tr>
<tr>
<td></td>
<td>( y_A \leftarrow D(x_A, key_A) )</td>
</tr>
<tr>
<td></td>
<td>if (</td>
</tr>
<tr>
<td></td>
<td>send ([B, A, enc(y_A + 1, key_A)])</td>
</tr>
<tr>
<td></td>
<td>session key ( F(y_A, K) )</td>
</tr>
<tr>
<td>rcv ([B, A, z_A])</td>
<td></td>
</tr>
<tr>
<td>if ( D(z_A, K) \neq ts + 1 ) FAIL</td>
<td></td>
</tr>
<tr>
<td>session key ( F(ts, K) )</td>
<td></td>
</tr>
</tbody>
</table>

Defending against replay attack

- \( B \) must remember timestamp values \( ts \) and \( ts + 1 \)
### Public-key

**Client A** (has \([\text{pri}_A, \text{pub}_A], \text{pub}_B\))  
**Server B** (has \([\text{pri}_B, \text{pub}_B], \text{pub}_A\))

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send</td>
<td>([A, B, \text{conn}, \mathcal{E}_P(r_A, B)])</td>
</tr>
</tbody>
</table>
| rcv    | \([A, B, \text{conn}, y_A]\)  
|        | \(x_A \leftarrow \mathcal{D}_P(y_A, B)\)  
| send   | \([B, A, \mathcal{E}([r_B, x_A], A)]\) |
| rcv    | \([A, B, y_B]\)  
|        | if \((r_B \neq \mathcal{D}_P(y_B, B))\) FAIL |
|        | session key \(x_A \oplus r_B\) |

**Example:**

- \(r_A \oplus x_B\)
- \(x_A \leftarrow \mathcal{D}_P(y_A, B)\)
- \(x_A \oplus r_B\)
Authenticated Diffie-Helman

- DH: $A : g^{S_A} \mod p$  
  $B : g^{S_B} \mod p$  
  key $g^{S_A \cdot S_B} \mod p$

- Authenticated DH: incorporates a pre-shared key

- If $A$ and $B$ share a secret-key $K$, here are two ways
  - Encrypt DH public keys with $K$
    - $A$ sends $\mathcal{E}((g^{S_A} \mod p), K)$
    - $B$ sends $\mathcal{E}((g^{S_B} \mod p), K)$
    - shared key: $g^{S_A \cdot S_B} \mod p$

  - Do usual DH, then exchange *keyed*-hashes of DH key.

- If $A$ and $B$ have each other’s public key, here are two ways
  - Encrypt DH quantities with receiver’s public key
  - Sign DH quantities with sender’s private key
Session Key Generation (w/o DH)

- Let $A$ and $B$ exchange challenges $r_A$ and $r_B$ during authentication.
- Let $G(.)$ combine its arguments in some way, eg, concatenation.
- If $A$ and $B$ share long-term secret-key $K$, the session key can be
  - $\mathcal{H}(G(r_A, r_B, K)$
  - $\mathcal{E}(G(r_A, r_B), K')$, where $K'$ is related to $K$
    - $K'$ can be $K$ only if attacker cannot obtain $\mathcal{E}(., K)$
      eg, by attacking authentication handshake.
- If $A$ and $B$ authenticated using public keys, session key can be
  - $G(r_A, r_B)$, if $r_A$ and $r_B$ were never exposed.
Session Keys

- Should differ from long-term key used for authentication
  - To avoid long-term key “wearing out” (offline crypto attack)
- Should be forgotten after session ends
- Should be unique for each session
  - If compromised, only affects data sent in that session.
  - Can be given to relatively untrusted software

“Delegation” (aka “authentication forwarding”):
Suppose A wants B to access C on A’s behalf
- A can give B its password (too risky)
- A can give B a “ticket”:
  \( \mathcal{E}(\text{[allowed operations, expiry time, ...]}, \text{A-B shared key}) \)
Miscellaneous
Countering Denial-of-Service Attacks

- Typically, when a server receives a (potential) connection request, it starts to maintain state for that client (id, challenge, ...).
- Attacker can disable server by flooding with connection requests.
- Defense: Ask client do some “checkable” work before storing state.
- Example: Server
  - has a frequently-changing secret $S$ (not shared with anyone)
  - sends $c = \mathcal{H}(\text{client ip addr}, S)$ to potential client
  - expects $c$ in response before storing state
  - $c$ can involve additional work (eg, reversing a small hash)
Negotiating crypto parameters

- In A–B session initiation: A sends crypto options and B responds with crypto accepted.
- Allows protocol to upgrade to better crypto when it becomes available.
- Because negotiation is done before authentication, need to reconfirm (reiterate negotiation) after authentication.
Authentication of Humans

- Limitations if $A$ is human
  - $A$ can only remember low-quality secret, i.e., password.
  - $A$ cannot do cryptographic operations, so relies on computer.
- Authentication based on password
- Authentication based on physical tokens:
  - physical keys, strip cards, smart cards (with processor), ...
  - can be lost/stolen; so augment with pwd, replaceable
- Biometric features
  - signature, fingerprint, voice recognition, iris/retina scan, ...
  - False negatives, false positives
Attacks on Human I/O

- Key Logger to capture passwords
- Login Trojan Horse to capture passwords
  - Running program on public terminal that imitates login prompt
  - get password, exit with “login failed” message,
    or run virtual OS for duration of user session
- Defenses by OS/hardware:
  - Special prompt symbol at any input field of non-login program
  - Allow only login screen to fill entire display
  - Non-mappable key to interrupt any running program, eg, alt-ctrl-del (but often OS allows remapping)
  - Display number of unsuccessful login attempts since last successful login.
- Any defense fails given a sufficiently naive user
Human A with password gets high-quality key from B

Basic strong password protocols (EKE, SPEKE, PDM)
- Use authenticated Diffie-Hellman
- Strong protection against network attack
- No protection against exposure of B’s pwd file

Augmented strong password protocols (EKE, SPEKE, PDM)
- Strong protection against network attack
- Weak protection against exposure of B’s pwd file

Can be used by A to obtain a strong key (eg, private key) from B
Augmented EKE Strong Password Protocol – I

- Public DH parameters $g$ and $p$
- $A$ has password $pw$
  - $W$ and $W'$: two keys obtained from $pw$
  - $T_A' = g^W \mod p$
- $B$’s entry for $A$ is $[W', T_A']$ (so $W'$ is open but not $W$)
- $A$ and $B$ do authenticated DH using $W'$ to establish session key
  - $A$: random $a$; $B$ random $b$
  - $K_A = K_B = g^{a \cdot b} \mod p$
- $A$ and $B$ also generate DH key $g^{W' \cdot b} \mod p$ for authentication:
  - $A$: $K_A' \leftarrow T_B^W \mod p$
  - $B$: $K_B' \leftarrow (T_A')^b \mod p$
Augmented EKE Strong Password Protocol – II

- **A:** random $a$, $T_A \leftarrow g^a \mod p$, send $\text{enc}(T_A, W')$
- **B:** extract $T_A$, random $b$, $T_B \leftarrow g^b \mod p$, $K_B \leftarrow T_A^b \mod p$, $K'_B \leftarrow (T'_A)^b \mod p$, $H_1 \leftarrow H_1(K_B, K'_B)$, send $\text{enc}(H_1, W')$ to $A$
- **A:** extract $H_1$ and $T_B$, $K_A \leftarrow T_B^a \mod p$, $K'_A \leftarrow (T_B)^W \mod p$, verify $H$ equals $H_1(K_B, K'_B)$ to authenticate $B$, $H_2 \leftarrow H_2(K_B, K'_B)$, send $\text{enc}(H_2)$ to $B$
- **B:** verify $H_2$ equals $H_2(K_B, K'_B)$ to authenticate $A$
- **A and B** mutually authencicated; share strong key $g^{a\cdot b} \mod p$
Scaling to many users and domains
Scaling to $N$ users

- Naive approach: Distinct key for every pair of principals.
  - Not scalable
  - $N^2$ storage at each principal
  - $N$ cost for adding/removing principal

- Secret-key solution: key distribution center (KDC)
- Public-key solution: certification authority (CA)

- Brings up new attacks involving out-of-sync master keys
KDC
Generic
**KDC – I**

- KDC is a special principal in the domain
- Every other principal \( z \) shares a **master key**, say \( K_z \), with KDC
- \( A-B \) session: \( A \) gets [**session key**, **ticket** for \( B \)] from KDC

<table>
<thead>
<tr>
<th>client ( A ) (has ( K_A ))</th>
<th>KDC (has ( K_A, K_B ))</th>
<th>server ( B ) (has ( K_B ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>send ([A, B] ) to KDC</td>
<td>rcv ([A, B] )</td>
<td>generate session key ( S )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( t_A \leftarrow \mathcal{E}([A, B, S], K_A) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( t_B \leftarrow \mathcal{E}([A, B, S], K_B) )</td>
</tr>
<tr>
<td>rcv ([KDC, u_A, u_B] )</td>
<td>send ([KDC, t_A, t_B] ) to ( A )</td>
<td>send ([A, B, t_B] ) to ( A )</td>
</tr>
<tr>
<td>( S_A \leftarrow \mathcal{D}(u_A)[2] )</td>
<td></td>
<td>( S_B \leftarrow \mathcal{D}(v_B)[2] )</td>
</tr>
<tr>
<td>send ([A, B, t_B] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**KDC – II**

- **Advantages of KDC:**
  - Adding new principal: one interaction between principal and KDC
  - Revocation of principal: deactivate principal’s master key at KDC

- **Disadvantages of KDC:**
  - KDC can impersonate anyone to anyone.
  - KDC compromise makes the whole network vulnerable.
  - KDC failure means no new sessions can be started.
  - KDC can be a performance bottleneck.

- Replicating the KDC takes care of the last two disadvantages, but
  - Need to protect all replicas
  - When principal’s master key is changed
    - need to sync replicas
    - need to handle tickets issued with old master key
Session $A.X - B.Y$ across KDCs $X$ and $Y$ that share key

- $A$ sends $[A.X, B.Y]$ to $X$
  
  $X$ generates session key $K_{AY}$
  
  $$t_{XA} \leftarrow \mathcal{E}([A, Y, K_{AY}], K_{AX})$$
  
  $$t_{XY} \leftarrow \mathcal{E}([A, Y, K_{AY}], K_{XY})$$
  
  sends $[t_{XA}, t_{XY}]$ to $A$

- $A$ extracts $K_{AY}$ from $t_{XA}$
  sends $[A.X, B, t_{XY}]$ to $Y$

- $Y$ extracts $K_{AY}$ from $t_{XY}$
  generates session key $K_{AB}$
  
  $$t_{YA} \leftarrow \mathcal{E}([A, Y, K_{AB}], K_{AY})$$
  
  $$t_{YB} \leftarrow \mathcal{E}([A, Y, K_{AB}], K_{BY})$$
  
  sends $[t_{YA}, t_{YB}]$ to $A$

- $A$ extracts $K_{AB}$ from $t_{YA}$
  sends $[A.X, B.Y, t_{YB}]$ to $Y$

- $B$ extracts $K_{AB}$ from $t_{YB}$
Session $A.X_1$–$B.X_N$ across KDCs $X_1, \ldots, X_N$ where $X_j$–$X_{j+1}$ share a key

- $A$ gets [session key $K_{A,X_2}$, ticket $t_{X_1,X_2}$] from $X_1$
- $A$ gets [session key $K_{A,X_3}$, ticket $t_{X_2,X_3}$] from $X_2$
- $\ldots$
- $A$ gets [session key $K_{A,B}$, ticket $t_{X_N,B}$] from $X_N$
- $A$ sends [ticket $t_{X_N,B}$] to $B$

Better: $A$ passes along the sequence of KDCs traversed, so that $B$ sees the entire KDC-chain rather than just $X_N$
KDC
Needham-Schroeder and Otway-Reese
<table>
<thead>
<tr>
<th>client $A$ (has $K_A$)</th>
<th>KDC (has $K_A, K_B$)</th>
<th>server $B$ (has $K_B$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate random $n_1$</td>
<td>rcv $[A, B, n_1]$</td>
<td>generate session key $S$</td>
</tr>
<tr>
<td>send $[A, B, n_1]$ to KDC</td>
<td>generate $t_B \leftarrow \mathcal{E}([A, B, S], K_B)$</td>
<td>$t_A \leftarrow \mathcal{E}([A, B, n_1, S, t_B], K_A)$</td>
</tr>
<tr>
<td></td>
<td>send $[KDC, A, t_A]$ to $A$</td>
<td></td>
</tr>
<tr>
<td>rcv $[KDC, A, t_A]$</td>
<td>$[A, B, n_1, S_A, t_B] \leftarrow \mathcal{D}(t_A, K_A)$</td>
<td></td>
</tr>
<tr>
<td>$[A, B, n_1, S_A, t_B]$</td>
<td>generate random $n_2$</td>
<td></td>
</tr>
<tr>
<td>send $[A, B, t_B, \mathcal{E}(n_2, S_A)]$ to $B$</td>
<td>rcv $[A, B, t_B, x_2]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$[A, B, S_B] \leftarrow \mathcal{D}(t_B, K_B)$</td>
<td>$n_2 \leftarrow \mathcal{D}(x_2, S_B)$</td>
</tr>
<tr>
<td></td>
<td>$n_2 \leftarrow \mathcal{D}(x_2, S_B)$</td>
<td>generate random $n_3$</td>
</tr>
<tr>
<td></td>
<td>send $[B, A, \mathcal{E}([n_2-1, n_3], S_A)]$</td>
<td>send $[B, A, \mathcal{E}([n_2-1, n_3], S_A)]$ to $B$</td>
</tr>
<tr>
<td>rcv $[B, A, x_{23}]$</td>
<td>$[n_2-1, n_3] \leftarrow \mathcal{D}(x_{23}, S_A)$</td>
<td>rcv $[A, B, x_3]$</td>
</tr>
<tr>
<td>$[n_2-1, n_3] \leftarrow \mathcal{D}(x_{23}, S_A)$</td>
<td>send $[A, B, \mathcal{E}(n_3-1, S_A)]$ to $B$</td>
<td>$[n_3-1] \leftarrow \mathcal{D}(x_3, S_A)$</td>
</tr>
</tbody>
</table>
Nonce $n_1$: assures A that msg 2 is response by KDC to msg 1

If $n_1$ not present, attacker with old password of $B$ can impersonate $B$ to $A$
  - $C$ records above exchange (refer to them as old msgs 1, ..., 5)
  - $C$ steals $K_B$; $B$ changes key
  - $C$ decrypts $t_B$ and gets $S$
  - $C$ waits until $A$ initiates session to $B$
  - $C$ intercepts $A$’s new msg 1, responds with old msg 2 $(enc([A, B, t_B], K_A))$
  - After this, attacker has session key used by $A$

Msg 2: id $B$ encrypted by $K_A$ ensures that attacker cannot replay old KDC reply to attacker (requesting to talk to $B$)

Msg 2: doubly encrypting $t_B$: defense against DOS attack
Needham-Schroeder – III

- Vulnerability if $n_1$ is sequential
  - Attacker records $A$–$B$ session with $n_1$ equal to, say $J$
  - Attacker spoofs $A$ to KDC with $n_1 = J + 1$
  - Attacker steals $K_B$; $B$ changes its key
  - Attacker waits for $A$ to initiate session to $B$
  - Attacker impersonates KDC and then $B$
Needham-Schroeder – IV

- Vulnerable to old password exposure
  - Attacker records $A-B$ session
  - Attacker gets $K_A$; $A$ changes it, to say $J_A$
  - Attacker can impersonate $A$ to $B$ by using old msg 3.
    Because $B$ never talks to KDC

- Fix
  - $B$ sends a nonce encrypted by $K_B$ in response to $A$’s request
  - $B$ expects nonce in its ticket
Expanded Needham-Schroeder Protocol

- $A$ sends connect request to $B$
- $B$ generates nonce $n_B$ and responds with $E(n_B, K_B)$
- $A$ sends $[n_1, E(n_B, K_B)]$ to KDC
- KDC generates
  - session key $S$
  - $t_B \leftarrow enc([A, B, S, n_B], K_B)$
  - sends $t_A = enc([n_1, B, S, t_B], K_A)$ to $A$
- $A$ extracts $S$ from $t_A$ and sends $t_B$ to $B$
- ...

// ticket
Otway-Reese Protocol

- A generates nonces $n_A$ and $n_C$
  sends $[A, B, n_C, \mathcal{E}([n_A, n_C, A, B], K_A)]$ to B
- B generates nonce $n_B$
  sends $[B, \mathcal{E}([n_A, n_C, A, B], K_A), \mathcal{E}([n_B, n_C, A, B], K_B)]$ to KDC
- KDC: decrypts both encryptions.
  If the two $n_C$'s are equal
  - generates session key $S$
  - sends $[n_C, \text{enc}([n_A, S], K_A), \text{enc}([n_B, S], K_B)]$ to B
- B: decrypts $\text{enc}([n_B, S], K_B)$
  - if $n_B$ matches, sends $\text{enc}([n_A, S], K_A)$ to A
- A decrypts $\text{enc}([n_A, S], K_A)$
  - if $n_A$ matches, sends $\text{enc}("hello", S)$ to B
- ...
CA
Generic
CA – I

- Every principal $z$ has a PK pair $[pri_z, pub_z]$
- CA is a special principal, say $X$

Every principal $z$
- Has CA’s id, $X$, and public-key, $pub_X$ // trust root
- Has a “certificate”: $[pub_z, ...]$ signed by $X$
- To acquire $y$’s public-key
  - gets $y$’s certificate unsecurely // eg, from $y$, a server
  - verifies the certificate // using $pub_X$

Above is over-simplified
- CA also periodically issues a “certificate revocation list” (CRL)
- Not all principals may have a PK pair
  - Eg, human clients may use pwd
CA – II

- Certificate for $z$ issued by $X$
  - issuer: $X$’s name, address, ...
  - subject: $z$’s name, address, ...
  - subject public-key: $pub_z$
  - expiry time // long-lived: month, year, ...
  - serial number // for CRL
  - $X$’s signature on above

- CRL issued by $X$
  - issuer: $X$’s name, address, ...
  - issue time // frequent: hourly, daily, ...
  - list of serial numbers // revoked unexpired certificates
  - $X$’s signature on above
To verify a certificate, need a sufficiently recent CRL
- validity established as of CRL’s issue date

Verification steps
- verify certificate’s signature  // using \( pub_X \)
- check certificate has not expired
- verify CRL’s signature  // using \( pub_X \)
- check that certificate’s serial number not in CRL

\([X, pub_X]\): only public-key used w/o verification  // “trust root”

A–B session:
- A, B exchange certificates, verify; do public-key authentication
- B has PK-pair, and A shares \( pwd \) with B
  - A gets B’s certificate, verifies; sends \( E_P(pwd, pub_B) \)
Advantages

- CA does not need to be online, so can be more secure
- Note: CA does not participate in $A-B$ authentication
- CA failure does not stop new sessions until certs expire
- Compromised CA cannot decrypt conversations (unlike KDC). (But can impersonate any principal via false certificate)

Disadvantages

- Revocation is complicated: has high-overhead if timely
Certificate chain

- Allows $A$ with trust root $X$ to verify $cert_{B,Y}$

- If $X$ has issued a certificate for $Y$ then $A$ does
  - get $cert_{X,Y}$ and verify (using $pub_X$)
  - $A$ get $cert_{B,Y}$ and verify (using $pub_Y$ from $cert_{X,Y}$)
  - $[cert_{X,Y}, crl_X], [cert_{Y,B}, crl_Y]$ is a “certificate chain”

- Certificate chain from $A$ to $B$
  $[cert_1, crl_1], [cert_2, crl_2], \cdots, [cert_n, crl_n]$
  - $cert_1$’s issuer is a trust root of $A$ \hspace{1cm} // anchor
  - $[cert_j, crl_j]$ verifies public-key of $cert_{j+1}$’s issuer
  - $cert_n$’s subject is $B$ \hspace{1cm} // target
Public-Key Infrastructure (PKI) – oversimplified

- **Top-level CAs**
  - Reputable: Verisign, Thawte, Google, ...
  - Their public-keys pre-configured in OS/browsers/...
  - Hence trust roots by default

- **Upper/Mid-level CAs**
  - Get certificates from top-level CAs or other mid-level CAs
  - Issue certificates
  - Reputable and others // certificates for $10

- **Low-level CAs**
  - Do not get certificates
  - Issue certificates for internal use, accepted on faith

- Large organizations usually pay for top/upper-level certificates
- Individuals and small organizations usually do not
Obtaining strong key via augmented EKE

- Public DH parameters $g$ and $p$
- $A$ has password $pw$
  - $W$ and $W'$: two keys obtained from $pw$
  - strong key $Z$ (not stored with $A$)
- $B$'s entry for $A$ is $[W, \text{enc}(Z, W')]$

$A$: random $a$,

$$T_A \leftarrow g^a \mod p$$
$$W \leftarrow \mathcal{H}(pw)$$
send $\text{enc}(T_A, W)$

$B$: extract $T_A$, random $b$, $T_B \leftarrow g^b \mod p$, $K_B \leftarrow T_A^b \mod p$

send $[T_B, \mathcal{E}(Y, K_B)]$ // $g^{a \cdot b} \mod p$

$A$: extract $T_B$, $K_A \leftarrow T_B^a \mod p$

$Z \leftarrow \mathcal{D}(Y, K_A)$ // $g^{a \cdot b} \mod p$