Authentication Stuff

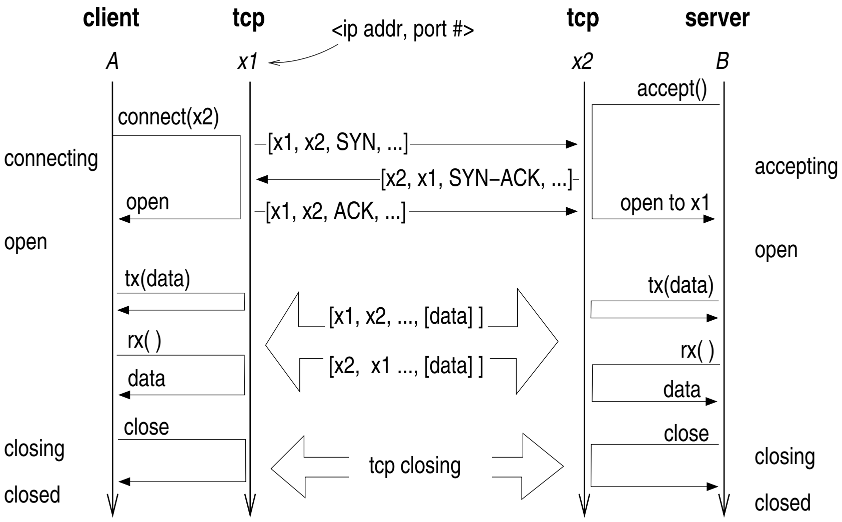
Shankar

May 28, 2013

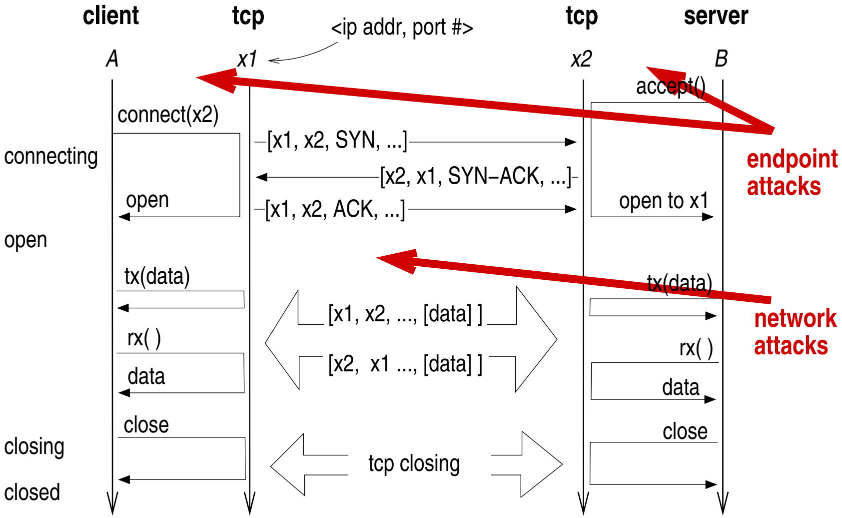
Overview

- 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

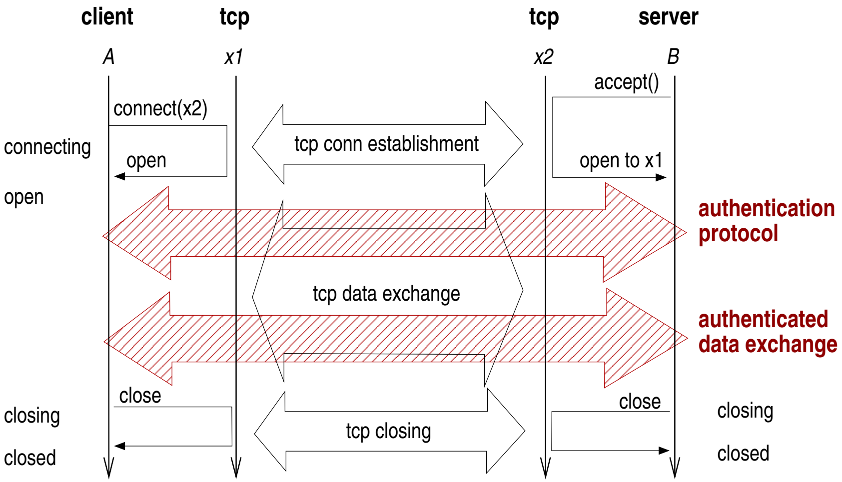
Typical scenario without authentication



Typical scenario with attacks



Typical scenario with authentication



Types of Attacks I

- 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

Types of Attacks III

- "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* // *ctxt* 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
  - $\mathcal{E}_{\mathcal{P}}(msg, pub)$: encrypt *msg* (with public key)
  - $\mathcal{D}_{\mathcal{P}}(msg, pri)$: decrypt *msg* (with private key)
  - $Sign(msg, pri)$: signature of *msg* (using private key)
  - $Vrfy(msg, s, pub)$: verify signature *s* of *msg* (with public key)

Conventions: 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.
-

client A (has pw)	server B (has $pwd_A = pw$)
send $[A, B, pw]$	
	rcv $[A, B, z]$ if $(z \neq pwd_A)$ FAIL // peer authenticated as A

- Channel eavesdropper gets password
  - So use only with secure channel
- Exposure of *pwd* file reveals all passwords
  - defense: encrypt *pwd* fle 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.
-

client A (has pw)	server B (has $hpw_A = \mathcal{H}(pw)$)
send $[A, B, pw]$	
	rcv $[A, B, z]$
	if $(\mathcal{H}(z) \neq hpw_A)$ FAIL

- 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 fle with a strong key

One-way authentication: Lamport Hash Scheme – I

- Server B has
  - n_A : # of logins remaining, initially say 1000
  - hpw_A : n_A-fold hash of pw // $= \mathcal{H}^{n_A}(pw)$
-

client A (has pw)	server B (has n_A and hpw_A)
send $[A, B, \text{conn}]$	
	rcv $[A, B, \text{conn}]$
	send $[B, A, n_A]$
rcv $[A, B, n]$	
send $[A, B, \mathcal{H}^{n-1}(pw)]$	
	rcv $[A, B, y]$
	if $(\mathcal{H}(y) \neq hpw_A)$ FAIL
	$[hpw_A, n_A] \leftarrow [y,\ n_A - 1]$

- 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 $[A, B, \text{conn}]$ with $[B, A, m]$ where $m < n_A$
  - A responds with $\mathcal{H}^{m-1}(pw)$
  - 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

client A (has key K)	server B (has $key_A = K$)
send $[A, B, \text{conn}]$	
	rcv $[A, B, \text{conn}]$
	generate random r_B // nonce
	send $[B, A, r_B]$
rcv msg $[B, A, x]$	
send $[A, B, \mathcal{E}(x, K)]$	
	rcv msg $[A, B, y]$
	if $(y \neq \mathcal{E}(r_B, key_A))$ FAIL

- 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

client A (has K, clk_A)	server B (has $key_A = K$, clk_B)
send $[A, B, \text{conn}, enc(clk_A, K)]$	rcv $[A, B, \text{conn}, x]$ if $(\lvert clk_B - \mathcal{D}(x, key_A)\rvert > D)$ FAIL

- 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 $\mathcal{E}$ by $\mathcal{H}$ 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: $Sign(r_B, A)$ // B can verify this
  - Can r_B be predictable (instead of random)?
  - Dictionary attack?
- Hidden challenge, open response
  - challenge: $\mathcal{E}_{\mathcal{P}}(r_B, A)$ 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 $\mathcal{E}(r_B, K)$ $\longleftarrow$ server challenge r_B
- client challenge r_A $\longrightarrow$ response $\mathcal{E}(r_A, K)$
- session key: $F(r_A, r_B, K)$ // one-way fn in K, eg, $\mathcal{H}$

client A (has key K)	server B (has $key_A = K$)
send $[A, B, \text{conn}]$	
	rcv $[A, B, \text{conn}]$
	generate random r_B
	send $[B, A, r_B]$
rcv $[B, A, x_B]$	
generate random r_A	
send $[A, B, r_A, \mathcal{E}(x_B, K)]$	rcv $[A, B, x_A, z_B]$
	if $(z_B \neq \mathcal{E}(r_B, key_A))$ FAIL
	send $[B, A, \mathcal{E}(x_A, key_A)]$
rcv $[B, A, z_A]$	session key $S_B \leftarrow F(x_A, r_B, key_A)$
if $(z_A \neq \mathcal{E}(r_A, K))$ FAIL	
session key $S_A \leftarrow F(r_A, x_B, K)$	

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

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

- 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 $\longrightarrow$ response $\mathcal{E}(r_A, K)$
  response $\mathcal{E}(r_B, K)$ $\longleftarrow$ 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

Secret-key; Timestamp-based

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

client A (has key K, clk_A)	server B (has $key_A = K$, clk_B)
$ts \leftarrow clk_A$	
send $[A, B, \text{conn}, enc(ts, K)]$	
	rcv $[A, B, \text{conn}, x_A]$
	$y_A \leftarrow \mathcal{D}(x_A, key_A)$
	if $(\lvert y_A - clk_B \rvert > D)$ FAIL
	send $[B, A, enc(y_A + 1, key_A)]$
	session key $F(y_A, K)$
rcv $[B, A, z_A]$	
if $(\mathcal{D}(z_A, K) \neq ts + 1)$ FAIL	
session key $F(ts, K)$	

- Defending against replay attack
  - B must remember timestamp values ts and $ts + 1$

Public-key

client A (has $[pri_A, pub_A]$, pub_B)	server B (has $[pri_B, pub_B]$, pub_A)
send $[A, B, \text{conn}, \mathcal{E}_{\mathcal{P}}(r_A, B)]$	
	rcv $[A, B, \text{conn}, y_A]$
	$x_A \leftarrow \mathcal{D}_{\mathcal{P}}(y_A, B)$
	send $[B, A, \mathcal{E}([r_B, x_A], A)]$
rcv $[B, A, y_{BA}]$	
$[x_B, z_A] \leftarrow \mathcal{D}_{\mathcal{P}}(y_{BA}, A)$	
if $(z_A \neq r_A)$ FAIL	
send $[A, B, \mathcal{E}_{\mathcal{P}}(x_B, B)]$	
session key $r_A \oplus x_B$ // example	
	rcv $[A, B, y_B]$
	if $(r_B \neq \mathcal{D}_{\mathcal{P}}(y_B, B))$ FAIL
	session key $x_A \oplus r_B$

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":
    $\mathcal{E}$([allowed operations, expiry time, ...], 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, ie, 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

Strong Password Protocols

- 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 $enc(T_A, W')$
- 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
  $H_1 \leftarrow \mathcal{H}_1(K_B, K'_B)$
  send $enc(H_1, W')$ 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 $\mathcal{H}_1(K_B, K'_B)$ to authenticate B
  $H_2 \leftarrow \mathcal{H}_2(K_B, K'_B)$
  send $enc(H_2)$ to B
- B: verify H_2 equals $\mathcal{H}_2(K_B, K'_B)$ 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

client A (has K_A)	KDC (has K_A, K_B)	server B (has K_B)
send $[A, B]$ to KDC		
	rcv $[A, B]$	
	generate session key S	
	$t_A \leftarrow \mathcal{E}([A, B, S], K_A)$	
	$t_B \leftarrow \mathcal{E}([A, B, S], K_B)$	
	send $[KDC, t_A, t_B]$ to A	
rcv $[KDC, u_A, u_B]$		
$S_A \leftarrow \mathcal{D}(u_A)[2]$		
send $[A, B, t_B]$		rcv $[A, B, v_B]$
		$S_B \leftarrow \mathcal{D}(v_B)[2]$

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} // for A-Y session
  $t_{XA} \leftarrow \mathcal{E}([A, Y, K_{AY}],\ K_{AX})$ // K_{AX}: A-X key
  $t_{XY} \leftarrow \mathcal{E}([A, Y, K_{AY}],\ K_{XY})$ // K_{XY}: X-Y key
  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} // for A-B session
  $t_{YA} \leftarrow \mathcal{E}([A, Y, K_{AB}],\ K_{AY})$
  $t_{YB} \leftarrow \mathcal{E}([A, Y, K_{AY}],\ K_{BY})$ // K_{BY}: B-Y key
  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 $X_1, \cdots, X_N$ 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
- $\cdots$
- 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

Needham-Schroeder Protocol – I

client A (has K_A)	KDC (has K_A,K_B)	server B (has K_B)
generate random n_1 send $[A, B, n_1]$ to KDC	rcv $[A, B, n_1]$ generate session key S $t_B \leftarrow \mathcal{E}([A, B, S], K_B)$ $t_A \leftarrow \mathcal{E}([A, B, n_1, S, t_B],\ K_A)$ send $[KDC, A, t_A]$ to A	
rcv $[KDC, A, t_A]$ $[\underline{A, B, n_1}, S_A, t_B] \leftarrow \mathcal{D}(t_A, K_A)$ generate random n_2 send$[A, B, t_B, \mathcal{E}(n_2, S_A)]$ to B		rcv $[A, B, t_B, x_2]$ $[\underline{A, B}, S_B] \leftarrow \mathcal{D}(t_B, K_B)$ $n_2 \leftarrow \mathcal{D}(x_2, S_B)$ generate random n_3 send $[B,A,\mathcal{E}([n_2\text{-}1,n_3],S_A)]$
rcv $[B, A, x_{23}]$ $[\underline{n_2\text{-}1}, n_3] \leftarrow \mathcal{D}(x_{23}, S_A)$ send $[A, B, \mathcal{E}(n_3\text{-}1, S_A)]$ to B		rcv $[A, B, x_3]$ $[\underline{n_3\text{-}1}] \leftarrow \mathcal{D}(x_3, S_A)$

Needham-Schroeder – II

- 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 $\mathcal{E}(n_B, K_B)$
- A sends $[n_1, \mathcal{E}(n_B, K_B)]$ to KDC
- KDC generates
  - session key S
  - $t_B \leftarrow enc([A, B, S, n_B],\ K_B)$ // ticket
  - 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
- ...

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 $[n_C, enc([n_A, S], K_A), enc([n_B, S], K_B)]$ to B
- B: decrypts $enc([n_B, S], K_B)$
  - if n_B matches, sends $enc([n_A, S], K_A)$ to A
- A decrypts $enc([n_A, S], K_A)$
  - if n_A matches, sends $enc(\text{"}hello\text{"}, S)$ 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 // $cert_{X,z}$
  - 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 // crl_X
  - issuer: X's name, address, ...
  - issue time // frequent: hourly, daily, ...
  - list of serial numbers // revoked unexpired certificates
  - X's signature on above

CA – II

- 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 $\mathcal{E}_{\mathcal{P}}(pwd, pub_B)$

CA – III

- 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 // anchor
  - $[cert_j, crl_j]$ verifies public-key of $cert_{j+1}$'s issuer
  - $cert_n$'s subject is B // 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 $[W, enc(Z, W')]$

- A: random a,
  $T_A \leftarrow g^a$ mod-p
  $W \leftarrow \mathcal{H}(pw)$
  send $enc(T_A, W)$
- B: extract T_A, random b, $T_B \leftarrow g^b$ mod-p,
  $K_B \leftarrow T_A^b$ mod-p // $g^{a \cdot b}$ mod-p
  send $[T_B, \mathcal{E}(Y, K_B)]$
- A: extract T_B, $K_A \leftarrow T_B^a$ mod-p // $g^{a \cdot b}$ mod-p
  $Z \leftarrow \mathcal{D}(Y, K_A)$

****