CS 578 – Cryptography

Prof. Michael Backes

Authentication Methods and SSL

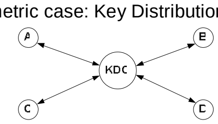
June 27, 2006

Administrative Announcements

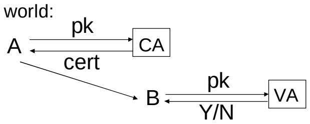
- Handouts:
  - New exercise sheet
  - Lecture notes on the web tomorrow; handed out on Friday

Recall: Distribution of Keys

- Comparison: distribution of symmetric / public keys.
- Recall symmetric case: Key Distribution Center (KDC)



- Public-key world:



Recall: Distribution of Keys

- KDC
  - is online: needed for every new session
  - is compromised → all past and future sessions are exposed (no forward secrecy)
  - fast
- CA
  - is offline, but VA is online
  - is compromised → then only future sessions exposed (forward secrecy)
  - slow

Authentication/Identification

- User Alice —Authentication protocol→ Server
- Obvious attacks:
  1. Eavesdropping (good solution: SSL)
  2. Expose secrets on server

Authentication/Identification

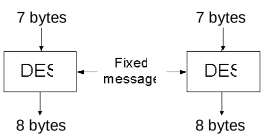
- Note: authentication protocol often also does key exchange
  without key exchange → session hijacking
- Authentication methods:
  - Passwords, one-time passwords, challenge-response authentication, STS, EKE

Authentication/Identification

1. Passwords:
   - Low entropy
   - Protection against replay
   - Protection of server
- On server: Do not store plaintext passwords
- [Alice, $H(P_A)$], [Bob, $H(P_B)$], …
- Attacker goal: Find y such that $H(y) = H(P_A)$
- → H has to be a one-way function

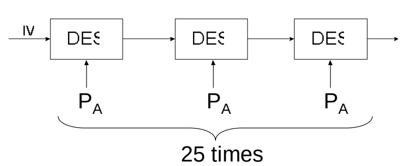
Authentication/Identification

- Windows:
  - MD4: Outputs 16 byte hash
  - LANMAN:
    - Accepted only 14 bytes of the passwords
    - Converts all characters into upper case (big no!)
    - Hash to 16 bytes as follows



Authentication/Identification

- Unix:



- User Password: limited to 7 bytes
- Hash: 8 bytes

Dictionary Attacks

- Attacker has password file
- Dictionary file: W_1, W_2, W_3, … W_N
- Compute $T_1 := H(W_1)$, $T_2 := H(W_2)$, …
- Then match T_i against the stored passwords in the password file for U users, i.e., Attacker intersects the two lists
  → recover all passwords in the dictionary
- Takes time:
  O(N) to hash dictionary
  + O(U·logU+N·logN) to compute intersection

Saarland University

Dictionary Attacks (cont'd)

- Salting:
  [Alice, $salt_A$, H(P_A || $salt_A$)],
  [Bob, $salt_B$, H(P_B || $salt_B$)], …
- Salt is random for every user
- UNIX – 12 bit salts
- Windows – no salts…
- With salting: Time to recover all passwords in dictionary: O(U·logU +N·S·log(N·S))

Saarland University

Dictionary Attacks (cont'd)

- Secret Salt (Pepper):
  [Alice, $salt_A$, H(P_A || $salt_A$ || $salt^*_A$)],
  [Bob, $salt_B$, H(P_B || $salt_B$ || $salt^*_B$)],
- $salt^*_A$ – 8-bit value not stored in password file
- Server tries all 256 $salt^*_A$ to validate password
- Biometric passwords:
  - Not secret
  - No revocation

One-time Passwords

2. One-time Password (Lamport)
- First mechanism: S-Key
- Setup: Alice generates password P_A

$$P_A \xrightarrow{H} \xrightarrow{H} \xrightarrow{H} \xrightarrow{H} \xrightarrow{H} \xrightarrow{H} \xrightarrow{H} W_A$$

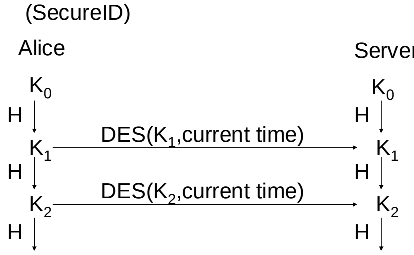
- $W_A := H(H(H(\ldots(H(P_A))\ldots)) = H^n(P_A)$
- Server stores W_A
- Alice sets cnt := n

One-time Passwords (cont'd)

- Authentication:
  - Decrease cnt
  - Send $A = H^{cnt}(P_A)$ to server
  - Server verifies $H(A) = W_A$. If so, it sets $W_A := A$.
- Prevents eavesdropping
- No secrets on server
- Limited number of passwords
- Vulnerable to preplay attack, e.g., phishing

One-time Passwords (cont'd)

- Second mechanisms: Secure Tokens (SecureID)



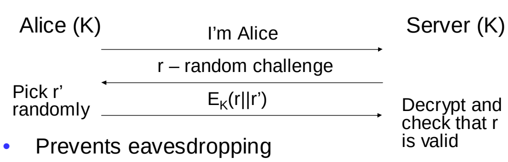
One-time Passwords (cont'd)

- Require secrets on server
- Tamper resistant
- Prevents eavesdropping
- Unlimited passwords
- Prevents phishing?
- No, but at least online phishing with on-the-fly usage of passwords

Challenge-Response Mechanisms

3. Challenge-Response Mechanisms

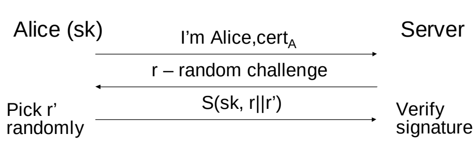
- First mechanism: based on symmetric ciphers



- Prevents eavesdropping
- Requires secrets on the server

Challenge-Response Mechanisms

- Second mechanism: Based on public-key crypto



- Prevents eavesdropping
- No secrets on the server

STS – Station-to-Station

- STS (mutual authentication + session key generation)
- Setup:
  - Publish prime p and generator g of Z^*_p
  - Alice selects signature keys pk, sk
  - Alice obtains certificate on pk

STS: Station-to-Station

- Auth:

Alice — Bob

$x \leftarrow_R \{1,\ldots,p-1\}$ — $y \leftarrow_R \{1,\ldots,p-1\}$

Alice → Bob: $cert_A$, $g^x \bmod p$

Bob → Alice: $cert_B$, $g^y \bmod p$, $E_K[S_B(g^x, g^y)]$

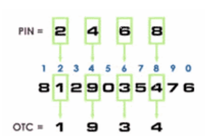
Alice → Bob: $E_K[S_A(g^x, g^y)]$

$K := H(g^{xy})$

- Then K is the shared key

Challenge-Response Mechanisms

- Challenge-response protocols used in real life to defend against key loggers
- Bank of Adelaide [show in browser]
- Swivel PINSafe



Source: Swivel

EKE – Encrypted Key Exchange

- Encrypted Key Exchange protocol (EKE)
- Key exchange + mutual authentication
- Problem: Alice typically only has a password, but EKE has to withstand dictionary attacks
- Negative example:

Alice $\xleftarrow{\quad r \quad}$ Server

Alice $\xrightarrow{\quad E_{pwd}(r) \quad}$ Server

Not secure against dictionary attacks

EKE – Encrypted Key Exchange

- EKE (Bellovin, Merritt 1992)

A (pwd) | B (pwd)

$x \leftarrow_R \{1,\ldots,p-1\}$ | $y \leftarrow_R \{1,\ldots,p-1\}$

$\xrightarrow{\quad h = AES_{pwd}(g^x) \quad}$

$\xleftarrow{\quad i = AES_{pwd}(g^y) \quad}$

$K = i^x = (g^y)^x = g^{xy}$ | $K = h^y = (g^x)^y = g^{xy}$

Then:

$\xrightarrow{\quad r \quad}$

$\xleftarrow{\quad E_K(r, r') \quad}$

EKE – Encrypted Key Exchange

- Main ideas of EKE:
  - Low entropy shared secret becomes a high entropy shared key
  - Prevents dictionary attack against the password
  - Prevents man-in-the-middle attack
  - Provides forward secrecy

Authentication by Jablon

- Jablon 1996

A (pwd) — B (pwd)

$x \leftarrow_R \{1,\ldots,p-1\}$ — $y \leftarrow_R \{1,\ldots,p-1\}$

A → B: pwd^x

B → A: pwd^y

$K = pwd^{xy}$ — $K = pwd^{xy}$

Schematic SSL

- SSL 3.0 → TLS (SSL 3.1)
- Basic schematic SSL

Browser — Server

Browser → Server: Client_Hello

session-id, c-rand (28 bytes), cipher-specs

Server → Browser: Server_Hello

session-id, s-rand (28 bytes), cipher-spec, server-cert

Schematic SSL

Browser — Server

Browser ↔ Server: Key exchange

→ pre-master secret (PMS): 40 bytes

- master-secret = Hash(PMS || c-rand || s-rand)
- master-secret used to derive
  [S-EK, S-IV, S-MK, C-EK, C-IV, C-MK]

Server → Browser: Finished

Browser → Server: Finished

Browser ↔ Server: Secure link (Enc+Mac)

- Ensures that both sides agree on some keys

SSL – Key Exchange Types

1. RSA

Browser — Server

← server-cert on pk = (N,e)

- Browser picks random K (48 bytes)
- Browser computes $c = [PKCS1(K)]^e \bmod N$

client-key-exchange →
c

K — decrypts c → K

- Problem: no forward secrecy

Saarland University

SSL – Key Exchange Types

2. EDH (Ephemeral Diffie-Hellman)

Browser — Server

← server-cert on pk = (N,e)

- Server picks random a
- Server computes $z_1 = g^a \bmod p$
- Server signs (p,g,z_1) with RSA key d (using, e.g., RSA-FDH) yielding sig

← server-key-exchange
p,g,z_1,sig

Saarland University

SSL – Key Exchange Types

- Browser verifies sig
- Browser picks random b
- Browser computes $z_2 = g^b \bmod p$

client-key-exchange →
z_2

- Browser and server and compute PMS $K := g^{ab}$
- Provides forward secrecy
- Problem: EDH three times slower than RSA key exchange

Performance of SSL (SSL Resume)

- Reducing number of key exchanges: SSL resume

Browser —— Client_Hello ——→ Server

session-id = CSID, c-rand, cipher-specs

- Server checks: does CSID exist in my session cache?

Yes: ←—— Server_Hello ——
session-id = CSID, s-rand, cipher-spec

No: ←—— Server_Hello ——
session-id = random, s-rand, cipher-spec

Performance of SSL (SSL Resume)

- If SSL resume (old session is reused). Then only termination:

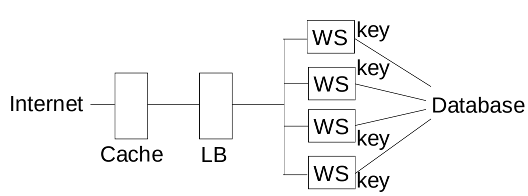
—— Finished ——→

←—— Finished ——

- Reuse the old PMS
- Note: new c-rand, s-rand

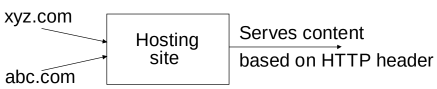
Problems with Deploying SSL

- Common web site design



Problems with SSL

- Slows down web server (also: use of RSA unfortunate)
- Secret key sk replicated across many servers
- Defeats purpose of caching
- Break virtual hosting

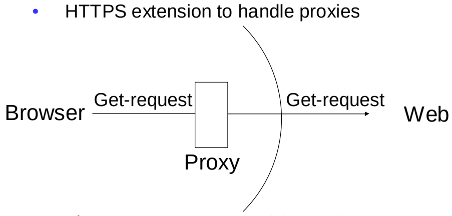


- Fixed in TLS 1.1
- TLS Client-Hello Extensions

Client_Hello →
session-id, c-rand (28 bytes), …, Host Hdr

Problems with SSL

- HTTPS extension to handle proxies



- First connect message: CONNECT Host-name, port
- Then SSL messages as usual

User Authentication

- User authentication: Key-based authentication

Browser (cert$_C$, sk$_C$) Server (cert$_S$, sk$_S$)

Client_Hello →
← Server_Hello, S-cert, cert-req, list of acc. CA's

c-cert (cert$_C$), client-key-exchange
cert-verify = Signature (using sk$_C$) on all protocol data →

← Finished
Finished →

Brief Overview of PGP

- Program for securing email, also file encryption
- Very popular in the private sector; free-of-charge for private use
- Uses a “web-of-trust” for key exchange
- Uses RSA, ElGamal, DSA, IDEA, 3DES, MD5, SHA-1, and SHA-256

Brief Overview of IPSEC

- Cryptographic protocol for securing IP communication plus successive key exchange
- Operates on network layer
  → more versatile than other protocols
- Headers are authenticated, variable fields such as hop-count are set to 0 before authenticating
- Two modes:
  - Transport mode, where only payload data is encrypted
  - Tunnel mode, where the whole package is encrypted, and new headers are added

Brief Overview of SSH

- First version in 1995: Response of a password-sniffing attack replacing telnet, rlogin, … which transmit passwords in plaintext
- Functionality includes
  - Secure shell,
  - SCP as a replacement for the rcp command,
  - tunneling protocols
- Various authentication methods: password, public-key based, …

Brief Overview of Kerberos

- Best-known system for symmetric key exchange with a central authority
- Originates from the “Athena” project at MIT (around 1987)
- More or less deprecated nowadays
- protocol standard as well as library of commands