ID protocols

Overview
The Setup

User P (prover) \( \rightarrow \) Alg. G \( \rightarrow \) Server V (verifier) \( \leftarrow \) yes/no

- sk
- vk

- vk either public or secret
- no key exchange
Applications: physical world

- Physical locks: (friend-or-foe)
  - Wireless car entry system
  - Opening an office door

- Login at a bank ATM or a desktop computer
Applications: Internet

Login to a remote web site after a key-exchange with one-sided authentication (e.g. HTTPS)
ID Protocols: how not to use

• ID protocol do not establish a secure session between Alice and Bob !!
  • Not even when combined with anonymous key exch.
  • Vulnerable to man in to the middle attacks

Prover k

Verifier k

ID protocol

anon. key exchange

sk

vk

Insecure!
ID Protocols: how not to use

• ID protocol do not set up a secure session between Alice and Bob !!
  • Not even when combined with anonymous key exch.
  • Vulnerable to man in the middle attack
ID Protocols: Security Models

1. **Direct Attacker**: impersonates prover with no additional information (other than vk)
   – Door lock

2. **Eavesdropping attacker**: impersonates prover after eavesdropping on a few conversations between prover and verifier
   – Wireless car entry system

3. **Active attacker**: interrogates prover and then attempts to impersonate prover
   – Fake ATM in shopping mall
ID protocols

Direct attacks
Basic Password Protocol (incorrect version)

- **PWD**: finite set of passwords

- Algorithm G (KeyGen):
  - choose $pw \leftarrow PWD$. output $sk = vk = pw$. 

```
User P (prover)
```

```
Server V (verifier)
```

$sk \rightarrow$ yes

$\text{iff } sk = vk$
Basic Password Protocol    (incorrect version)

Problem:  vk must be kept secret
  • Compromise of server exposes all passwords
  • Never store passwords in the clear!

password file on server

<table>
<thead>
<tr>
<th></th>
<th>Alice</th>
<th>pw_{alice}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td></td>
<td>pw_{bob}</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic Password Protocol: version 1

H: one-way hash function from PWD to X
  - “Given H(x) it is difficult to find y such that H(y)=H(x)”

User P (prover) sk → Server V (verifier) vk = H(sk)

yes iff H(sk)=vk

password file on server:

<table>
<thead>
<tr>
<th>User</th>
<th>H(password)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>H(pw_A)</td>
</tr>
<tr>
<td>Bob</td>
<td>H(pw_B)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Problem: Weak Password Choice

Users frequently choose weak passwords: (adobe list, 2013)

<table>
<thead>
<tr>
<th>Password</th>
<th>123456</th>
<th>123456789</th>
<th>password</th>
<th>adobe123</th>
<th>12345678</th>
<th>qwerty</th>
<th>1234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of users</td>
<td>5%</td>
<td>1.1%</td>
<td>0.9%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

Total: 8.8%

A common occurrence

• Example: the Rockyou password list, 2009  (6 most common pwds)

  123456, 12345, Password, iloveyou, princess, abc123

Dictionary of 360,000,000 words covers about 25% of user passwords
Online dictionary attack: Suppose an attacker obtains a list of usernames. For each username the attacker tries to login using the password ‘123456’.

<table>
<thead>
<tr>
<th>Password:</th>
<th>123456</th>
<th>123456789</th>
<th>password</th>
<th>adobe123</th>
<th>12345678</th>
<th>qwerty</th>
<th>1234567</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fraction of users:</td>
<td>5%</td>
<td>1.1%</td>
<td>0.9%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.5%</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

Success after 20 tries on average
Offline Dictionary Attacks

Suppose attacker obtains a single $vk = H(pw)$ from server

- **Offline** attack: hash all words in Dict until a word $w$ is found such that $H(w) = vk$

- Time $O(|Dict|)$ per password

Off the shelf tools (e.g. John the ripper):

- Scan through all 7-letter passwords in a few minutes
- Scan through 360,000,000 guesses in few seconds
  $\Rightarrow$ will recover 23% of passwords
Batch Offline Dictionary Attacks

Suppose attacker steals entire pwd file $F$

- Obtains hashed pwds for all users
- Example (2012): Linkedin ($6M$: $SHA1(pwd)$)

Batch dict. attack:

- For each $w \in Dict$: test if $H(w)$ appears in $F$ (using fast look-up)

Total time: $O(|Dict| + |F|)$  
[Linkedin: 6 days, 90% of pwds. recovered]

Much better than attacking each password individually!
Preventing Batch Dictionary Attacks

**Public salt:**

- When setting password, pick a random n-bit salt $S$
- When verifying pw for A, test if $H(pw_A, S_A) = h_A$

Recommended salt length, $n = 64$ bits
- Attacker must re-hash dictionary for each user

Batch attack time is now: $O(|Dict| \times |F|)$

<table>
<thead>
<tr>
<th>id</th>
<th>S</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$S_A$</td>
<td>$H(pw_A, S_A)$</td>
</tr>
<tr>
<td>Bob</td>
<td>$S_B$</td>
<td>$H(pw_B, S_B)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Further Important Defenses

Slow hash function $H$: (say 0.1 sec. to hash pw)
- Example: $H(pw) = SHA1(SHA1( ... SHA1(pw, S_A) ...))$
- Unnoticeable to user, but makes offline dictionary attack harder
- Use PBKDF2: tunable # iterations

Secret salts:
- When setting pwd choose short random $r$ (12 bits)
- When verifying pw for $A$, try all values of $r_A$. 2048 times slow down on average.
- 4096 times slow down for attacker
ID protocols

Security against eavesdropping attacks

(one-time password systems)
Eavesdropping Security Model

Adversary is given:

- Server’s vk, and
- the transcript of several interactions between honest prover and verifier.  
  (example: remote car unlock)

adv. goal is to impersonate prover to verifier

A protocol is “secure against eavesdropping” if no efficient adversary can win this game

The password protocol is clearly insecure!
One-time passwords (secret \(vk\), stateful)

Setup (algorithm G):
- Choose random key \(k\)
- Output \(sk = (k, 0)\); \(vk = (k, 0)\)

Identification:

\[
\begin{align*}
\text{prover} & \quad \text{server} \\
sk = (k, 0) & \quad vk = (k, 0) \\
\downarrow & \quad \downarrow \\
sk = (k, 1) & \quad \downarrow \\
\end{align*}
\]

- \(r_0 \leftarrow F(k, 0)\)
- \(r_1 \leftarrow F(k, 1)\)

Yes iff \(r = F(k, 0)\)

often, time-based updates: \(r \leftarrow F(k, \text{time})\) [stateless]
The SecurID system (secret vk, stateful)

“Thm”: if F is a secure PRF then protocol is secure against eavesdropping

RSA SecurID uses AES-128:

128 bit key
32 bit ctr

\[ F \]

\[ \rightarrow 6 \text{ digit output} \]

Advancing state: \( \text{sk} \leftarrow (k, i+1) \)

• Time based: every 60 seconds
• User action: every button press

Both systems allow for skew in the counter value
Google authenticator

• 6-digit timed one-time passwords (TOTP) based on [RFC 6238]
• Wide web-site adoption:
  – Evernote, Dropbox, WordPress, outlook.com, ...

To enable TOTP for a user: web site presents QR code with embedded data:

```
otpauth://totp/Example:alice@dropbox.com?
secret=JBSWY3DPEHPK3PXP & issuer=Example
```

(Subsequent user logins require user to present TOTP)

Danger: password reset upon user lockout
Server compromise exposes secrets

March 2011:
• RSA announced servers attacked, secret keys stolen
  ⇒ enabled SecurID user impersonation

Is there an ID protocol where server key $vk$ is public?
The S/Key system (public vk, stateful)

Notation: \( H^{(n)}(x) = H(H(...H(x)...)) \) \( n \) times

Algorithm G: (setup)
- Choose random key \( k \leftarrow K \)
- Output \( sk = (k,n) \); \( vk = H^{(n+1)}(k) \)

Identification:
The S/Key system (public vk, stateful)

Identification (in detail):

• Prover \((sk=(k,i))\): send \(t \leftarrow H^{(i)}(k)\); set \(sk \leftarrow (k,i-1)\)

• Verifier (\(vk=H^{(i+1)}(k)\)): if \(H(t)=vk\) then \(vk \leftarrow t\), output “yes”

Notes: \(vk\) can be made public; but need to generate new \(sk\) after \(n\) logins (\(n \approx 10^6\))

“Thm”: \(S/Key_n\) is secure against eavesdropping (public \(vk\)) provided \(H\) is one-way on \(n\)-iterates
SecurID vs. S/Key

S/Key:

- **public** vk, **limited** number of authentications
- Long authenticator t (e.g., 80 bits)

SecurID:

- **secret** vk, **unlimited** number of authentications
- Short authenticator (6 digits)
ID protocols

Security against active attacks

(challenge-response protocols)
Active Attacks

- Offline fake ATM: interacts with user; later tries to impersonate user to real ATM
- Offline phishing: phishing site interacts with user; later authenticates to real site

All protocols so far are vulnerable
MAC-based Challenge Response (secret vk)

“Thm”: protocol is secure against active attacks (secret vk), provided \((S_{MAC}, V_{MAC})\) is a secure MAC.
MAC-based Challenge Response

Problems:
• $vk$ must be kept secret on server
• dictionary attack when $k$ is a human $pwd$:
  
  \[
  \text{Given } [m, S_{\text{MAC}}(pwd, m)] \text{ eavesdropper can try all } pwd \in \text{Dict} \text{ to recover } pwd
  \]

Main benefit:
• Both $m$ and $t$ can be short
• CryptoCard: 8 chars each
Sig-based Challenge Response (public vk)

Replace MAC with a digital signature:

User P (prover)

Server V (verifier)

\[(sk, vk) \leftarrow G_{SIG}\]

random \(m \leftarrow M\)

\(t \leftarrow \text{Sign}(k, m)\)

\(\text{Verify}(k, m, t)\)

"Thm": Protocol is secure against active attacks (public vk), provided \((G_{SIG}, \text{Sign}, \text{Verify})\) is a secure digital sig.

but \(t\) is long (\(\geq 20\) bytes)
Summary

ID protocols: useful in settings where adversary cannot interact with prover during impersonation attempt

Three security models:

• **Direct:** passwords (properly salted and hashed)

• **Eavesdropping attacks:** One time passwords
  – SecurID: secret pk, unbounded logins
  – S/Key: public pk, bounded logins

• **Active attacks:** challenge-response
THE END