User Authentication: ID protocols

D. Boneh
The Setup

Alg. G

User P (prover)

Server V (verifier)

vk either public or secret

yes/no

no key exchange
Applications

- Physical locks: (friend-or-foe)
  - Wireless car entry system (e.g. KeeLoq)
  - Opening an office door or a garage door

- Login at a bank ATM or a desktop computer

- Login to a remote web site once key-exchange with one-sided authentication completes (e.g. SSL)
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

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 to 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 secure against **direct** attacks

a.k.a Password Systems
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)

$sk$ from $P$ to $V$:
- $yes$ 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>Alice</th>
<th>$pw_{\text{alice}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>$pw_{\text{bob}}$</td>
</tr>
<tr>
<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)”

Password file on server

User P (prover)

Server V (verifier)

\[ sk \]

\[ vk = H(sk) \]

yes iff \( H(sk)=vk \)

Alice  \( H(pw_A) \)
Bob  \( H(pw_B) \)
…  …
Weak Passwords and Dictionary Attacks

People often choose passwords from a small set:

- The 6 most common passwords (sample of $32 \times 10^6$ pwds):
  
  123456, 12345, Password, iloveyou, princess, abc123

  (‘123456’ appeared 0.90% of the time)

- 23% of users choose passwords in a dictionary of size 360,000,000

Online dictionary attacks:

- Defeated by doubling response time after every failure
- Harder to block when attacker commands a bot-net
Suppose attacker obtains \( vk = H(pw) \) 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**

- 2,000,000 guesses/sec
- Scan through 360,000,000 guesses in few minutes
  - Will recover 23% of passwords
Password Crackers

Many tools for this
- John the ripper
- Cain and Abel
- Passware(Commercial)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>2 383 000</td>
</tr>
<tr>
<td>MD5</td>
<td>4 905 000</td>
</tr>
<tr>
<td>LanMan</td>
<td>12 114 000</td>
</tr>
</tbody>
</table>
Batch Offline Dictionary Attacks

Suppose attacker steals pwd file F
  • Obtains hashed pwds for all users

Batch dict. attack:
  • Build list L containing \((w, H(w))\) for all \(w \in \text{Dict}\)
  • Find intersection of L and F

Total time: \(O( |\text{Dict}| + |F| )\)

Much better than a dictionary attack on each password
Preventing Batch Dictionary Attacks

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

Recommended salt length, $n = 64$ bits
- Pre-hashing dictionary does not help.

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

Slow hash function $H$: (0.1 sec to hash pw)

- Example: $H(pw) = SHA1(SHA1( ... SHA1(pw) ...))$
- Unnoticeable to user, but makes offline dictionary attack harder

Secret salts:

- When setting pwd choose short random $r$ (8 bits)
- When verifying pw for $A$, try all values of $r_A$: 128 times slow down on average
- 256 times slow down for attacker
**Case study: UNIX and Windows**

**UNIX:** 12-bit public salt
- Hash function H:
  - Convert pw and salt and a DES key \( k \)
  - Iterate DES (or DES') 25 times:

```
0 -> DES \( k \) -> DES \( k \) -> DES \( k \) -> h
```

**Windows:** NT and later use MD4
- Outputs a 16 byte hash
- No public or secret salts
Biometrics

Examples:
- Fingerprints, retina, facial recognition, …
- Benefit: hard to forget

Problems:
- Biometrics are not generally secret
- Cannot be changed, unlike passwords

⇒ Primarily used as a second factor authentication
The Common Password Problem

Users tend to use the same password at many sites
  • Password at a high security site can be exposed by a break-in at a low security site

Standard solution:
  • Client side software that converts a common password $pw$ into a unique site password

$$pw' \leftarrow H(pw, \text{user-id, server-id})$$

$pw'$ is sent to server
ID protocols secure against eavesdropping attacks

a.k.a One-time Password Systems
Eavesdropping Security Model

Adversary is given:

- \( vk \)
- the transcript of several interactions between honest prover and verifier.

adv. goal is to then impersonate prover to verifier

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

The password protocol is clearly insecure

- We discuss two secure stateful protocols (one-time pwd), and
- one stateless protocol (challenge-response)
The SecurID system  
(secret vk,  stateful)

Algorithm G:  (setup)
- Choose random key  \( k \leftarrow K \)
- Output  \( \text{sk} = (k,0) \);  \( \text{vk} = (k,0) \)

Identification:

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

\( r_0 \leftarrow F(k,0) \)

Yes iff  \( r = F(k,0) \)
The SecurID system  (secret vk,  stateful)

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

RSA SecurID uses a custom PRF:

- 64 bit key → F
- 24 bit ctr → 6 digit output

Advancing state: \( sk \leftarrow (k, i+1) \)
- Time based: every 60 seconds
- User action: every button press

Both systems allow for skew in the counter value
The S/Key system  (public vk, stateful)

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

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

Identification:

\[ k \overset{H(k)}{\rightarrow} H^{(n-2)}(k) \overset{H^{(n-1)}(k)}{\rightarrow} H^{(n)}(k) \overset{H^{(n+1)}(k)}{\rightarrow} vk \]

pwd #1 pwd #2 pwd #3 pwd #4
The S/Key system  (public vk, stateful)

Identification  (in detail):
• Prover  (sk=(k,i)): send  t ← H^{(i)}(k) ;  set  sk ← (k,i-1)
• Verifier(  vk=H^{(i+1)}(k) ):  if  H(t)=vk  then  vk←t,  output “yes”

Notes:  vk can be made public;
but need to generate new sk after n logins  (n ≈ 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 auths
- often implemented using pencil and paper

SecurID:
- **secret** vk,  **unlimited** number of auths
- often implemented using secure token
ID protocols secure against active attacks

a.k.a Challenge-Response Protocols
Active Attacks

Offline fake ATM: interacts with user; later tries to impersonate to legit. ATM

Offline phishing: phishing site interacts with user; later authenticates to real site

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

User P (prover)

Server V (verifier)

k ← K

sk = k

vk = k

m ← M

t ← SMAC(k, m)

VMAC(k, m, t)

“Thm”:
Protocol is secure against active attacks (secret vk), provided (SMAC, VMAC) is a secure MAC
MAC-based Challenge Response

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

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)

"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 \((\geq 20\ \text{bytes})\)
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 vk, unbounded logins
    • S/Key: public vk, bounded logins
  
  • **Active attacks**: challenge-response