User Authentication: ID protocols
The Setup

User P (prover)  

Server V (verifier)  

Alg. G  

vk either public or secret  

no key exchange  

yes/no
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 the middle attack
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  $sk = vk = pw$. 

![Diagram showing the Basic Password Protocol]

- User P (prover)
- Server V (verifier)
- $sk$ is the secret key
- $vk$ is the verification key
- $sk = vk$ confirms the password

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

password file on server

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$p_{w_{\text{alice}}}$</td>
</tr>
<tr>
<td>Bob</td>
<td>$p_{w_{\text{bob}}}$</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 $H(y) = H(x)$”

User P (prover) → sk → Server V (verifier)

$vk = H(sk)$

yes iff $H(sk) = vk$

password file on server

<table>
<thead>
<tr>
<th>Alice</th>
<th>$H(pw_A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>$H(pw_B)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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
Offline Dictionary Attacks

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)

Using CUDA: 5x speed-up
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: \(O(\mid \text{Dict} \mid + \mid F \mid)\)

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: \( H(pw) = \text{SHA1}(	ext{SHA1}(\ldots \text{SHA1}(pw) \ldots )) \)
- 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

\[
\begin{array}{|c|c|c|}
\hline
\text{Alice} & S_A & H(pw_A, S_A, r_A) \\
\hline
\text{Bob} & S_B & H(pw_B, S_B, r_B) \\
\hline
\ldots & \ldots & \ldots \\
\hline
\end{array}
\]
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  →  DES  →  DES
   ↑  k     ↑  k       ↑  k
```

**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

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

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

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

Adversary is given:

- $vk$, and
- 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  $sk = (k,0)$ ;  $vk = (k,0)$

Identification:

<table>
<thead>
<tr>
<th>prover</th>
<th>verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk = (k,0)$</td>
<td>$vk = (k,0)$</td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$sk = (k,1)$</td>
<td>$vk = (k,1)$</td>
</tr>
</tbody>
</table>

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

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 $\rightarrow$ F $\rightarrow$ 6 digit output
- 24 bit ctr $\rightarrow$ 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: \( H^{(n)}(x) = H(H(\ldots H(x)\ldots)) \)  

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

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

pwd #4  pwd #3  pwd #2  pwd #1  vk
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”: \(S/\text{Key}_n\) 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)

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

Problems:

• \( vk \) must be kept secret on server
• dictionary attack when \( k \) is a human \( pwd \):
  • 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
Replace MAC with a digital signature:

\[
\begin{align*}
(sk, vk) & \leftarrow G_{\text{SIG}} \\

\text{User P (prover)} & \quad \text{Server V (verifier)} \\
\text{sk} & \quad \text{vk} \\
m & \leftarrow M \\
t & \leftarrow \text{Sign}(k, m) \\
\end{align*}
\]

“Thm”:
Protocol is secure against active attacks (public vk), provided \((G_{\text{SIG}}, \text{Sign}, \text{Verify})\) 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
Advanced Topics

- Anonymous digital cash
- Zero knowledge protocols and Dlog signatures
- Quantum computing
- Elliptic curve cryptography
- Factoring algorithms
- Advanced pub-key techniques: IBE, ABE, functional
THE END