Revocation

- What happens if someone steals your private key?
- They can impersonate you and read messages encrypted to you
- Key expiration helps with this but not enough
- CA and PGP PKIs support revocation
  - Owner says: “I, Alice, revoke my public key … do not use it.”
  - Signs revocation with her private key
  - Others can verify Alice’s signature, stop using key
Revocation

- In CA model, Alice asks CA to revoke certificate
  - Alice does not need private key to do this
  - CAs publish a Certificate Revocation List (CRL)

- In PGP model, only Alice can revoke her own key
  - If Alice loses her private key, she can’t revoke
  - Do not lose private key
  - Option: generate revocation with key, store in secure place
Revocation

- How does Bob know if Alice’s key has been revoked?
- Bob asks Alice: “Has your key been revoked?”
- Alice sends signed message: “No.”
- Does not work: if Alice’s key has been compromised, then Eve could have forged the message “No.”
- Availability of revocation list critical
Revocation Today

- Two Mechanisms: CRL and OCSP
- **Published CRL:** certificate also says whereto get CRL
  - What if CRL server is down?
- **Online Certificate Status Protocol:** Query CA about cert
- **OCSP stapling:** Web server includes recent OCSP cert
X509v3 extensions:
  X509v3 Authority Key Identifier:

X509v3 Subject Key Identifier:

X509v3 Subject Alternative Name:
  DNS:*.ucsd.edu, DNS:ucsd.edu

X509v3 Key Usage: critical
  Digital Signature, Key Encipherment

X509v3 Extended Key Usage:
  TLS Web Server Authentication, TLS Web Client Authentication

X509v3 CRL Distribution Points:
  URI:http://crl3.digicert.com/ca3-g14.crl
  URI:http://crl4.digicert.com/ca3-g14.crl

X509v3 Certificate Policies:
  Policy: 2.16.840.1.114412.1.1
    CPS: http://www.digicert.com/ssl-cps-repository.htm
User Notice:
  Explicit Text:

Authority Information Access:
  OCSP - URI:http://ocsp.digicert.com

X509v3 Basic Constraints: critical
“On the Internet, nobody knows you’re a dog.”
The Problem

- Security policies refer to persons
  - Legal deterrents apply to persons
  - Systems built for people not machines
- People don’t exist at machine level
- How do we attribute actions to persons?
Identity

- **Identity**: the totality of characteristics that define an object in a particular context

- **Identifier**: a name uniquely determining an object in a particular context

- We use identifiers to name objects in a system
  - *Objects* includes *subjects*
Identity Example

- **Context:** California Department of Motor Vehicles
- **Identifier:** driver license number
- **Identity:** what defines a driver from DMV point of view: driver license number, name, birth date, height, weight, eye color, hair color, photo, fingerprint, etc.
Authentication

- **Authentication**: Associating the subject of an interaction with an identity known to the system

- **Example**: Showing police officer your driver license when you get pulled over

- **Example**: Providing your user name and password to a Web site
Authentication Goal

- Only the intended subject can authenticate to the system as that subject.
- If a less privileged subject acts as more privileged subject, security policy may be violated.
Authentication Kinds

- Something you know
  - Password, other identifiers

- Something you have
  - Card, key, one-time password device

- Something you are
  - Biometrics
Many authentication schemes have two elements:

- Provide an identifier to the system
- Prove that you are the subject associated with identifier
- Example: user name and password
Popular Auth. Mechanisms

- Username and password
  - Two factor: secondary verification, e.g., via phone

- Card and PIN
  - No need to explicitly enter identifier

- Biometric
  - E.g. fingerprint, voice, face, retina scan
Passwords

- Password is a shared secret

- Commonly defeated by:
  - Password compromise at subject (e.g. keystroke logger)
  - Password compromise in system (e.g. database leak)
  - Brute force attack
User-Side Compromise

❖ **Old school:** post-it note with password stuck to monitor
  • Targeted attacks against particular user

❖ **Modern:** electronic capture during entry
  • Insecure Internet connection
    • Man-in-the-middle
    • Unencrypted
  • Keystroke logger
    • Targeted attack
    • Generic malware
System-Side Compromise

- System may be compromised and passwords stolen
  - Huge yield compared to user-side attacks

- Defenses:
  - Protect the password database
  - Make password recovery from database difficult
Checking Password

- System does not need to *know* password, only *check* it
- **One-way function:** easy to compute, hard to invert
  - Also called pre-image resistance
- **Solution:** store cryptographic hash of password
  - Vulnerable to rainbow table attacks *(discussed next)*
  - Pre-image resistance implies hard to get password
Rainbow Table Attack

- Attacker generates hashes of all dictionary words and/or all strings up to certain length

- **Example:**
  - Each password character upper or lower case letter or digit
  - Roughly 64 possible values per character
  - $64^n$ possible passwords of length $n$
  - For $n = 6$: $2^{36}$ possible passwords strings:
    - 10TB to store 6-character password and SHA1 hash
Rainbow Table Attack

- **Rainbow table:** list of all possible passwords of certain form and their hashes

- Compare all possible passwords (rainbow table) against password database
  - For \( n = 6 \): 10TB sort—feasible

- Very common attack today
Free Password Hash Cracker

Enter up to 10 non-salted hashes:

Supports: LM, NTLM, md2, md4, md5, md5(md5), md5-half, sha1, sha1(sha1_bin()), sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing_security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with...
Rainbow Table Attack

- What is the problem? Work hashing one possible password (when generating table) amortized over all password we compare against

- Total work $O(m+n)$
  - $m$ — size of rainbow table
  - $n$ — size of password database trying to crack

- Want: $O(m \times n)$
  - Have to brute force every password
Salting

- Salting: add additional random value to password
- Generate random salt $s$
- Store $\langle s, \text{MAC}_p(s) \rangle$ in the database
  - Sometimes additional hashing added to slow down brute forcing
- Attacker would need to precompute rainbow tables for every salt
- Or attack each password individually
Brute Forcing

- **Simplest idea:** try every password
  - Against salted table
  - Interactively if the system does not limit attempts

- **Better:** try common passwords first
  - Dictionary words, common passwords
<table>
<thead>
<tr>
<th>Rank</th>
<th>Password</th>
<th>Change from 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
<td>Up 1</td>
</tr>
<tr>
<td>2</td>
<td>password</td>
<td>Down 1</td>
</tr>
<tr>
<td>3</td>
<td>12345678</td>
<td>Unchanged</td>
</tr>
<tr>
<td>4</td>
<td>qwerty</td>
<td>Up 1</td>
</tr>
<tr>
<td>5</td>
<td>abc123</td>
<td>Down 1</td>
</tr>
<tr>
<td>6</td>
<td>123456789</td>
<td>New</td>
</tr>
<tr>
<td>7</td>
<td>111111</td>
<td>Up 2</td>
</tr>
<tr>
<td>8</td>
<td>1234567</td>
<td>Up 5</td>
</tr>
<tr>
<td>9</td>
<td>iloveyou</td>
<td>Up 2</td>
</tr>
<tr>
<td>10</td>
<td>adobe123</td>
<td>New</td>
</tr>
<tr>
<td>11</td>
<td>123123</td>
<td>Up 5</td>
</tr>
<tr>
<td>12</td>
<td>admin</td>
<td>New</td>
</tr>
<tr>
<td>13</td>
<td>1234567890</td>
<td>New</td>
</tr>
<tr>
<td>14</td>
<td>letmein</td>
<td>Down 7</td>
</tr>
<tr>
<td>15</td>
<td>photoshop</td>
<td>New</td>
</tr>
<tr>
<td>16</td>
<td>1234</td>
<td>New</td>
</tr>
<tr>
<td>17</td>
<td>monkey</td>
<td>Down 11</td>
</tr>
<tr>
<td>18</td>
<td>shadow</td>
<td>Unchanged</td>
</tr>
<tr>
<td>19</td>
<td>sunshine</td>
<td>Down 5</td>
</tr>
<tr>
<td>20</td>
<td>12345</td>
<td>New</td>
</tr>
<tr>
<td>21</td>
<td>password1</td>
<td>Up 4</td>
</tr>
<tr>
<td>22</td>
<td>princess</td>
<td>New</td>
</tr>
</tbody>
</table>
Centralized Authentication

- **Problem**: lots of different passwords
- People will re-use passwords
- Sites with poor security practices will leak passwords
- Can we centralize authentication?
- OpenID and OAuth