Authentication
Password Thoughts

• A secret should be chosen from about a $2^{64}$ space to be considered secure

• Rule of thumb
  – human-generated password has about 2 bits per character
  – computer generated pronouncable, 4 bits

• don’t echo, because of someone watching

• But why not echo the token card value!!!
More password thoughts

- Dangerous to use `pwd` in multiple places
  - how could we make it less dangerous?
- login Trojan horse trick (and equivalents)
- initially configuring the user’s password
- variant on S/Key: list, and ask for #’s
Using same password on multiple systems

- How can user type (name,pwd) to use on multiple systems without one system being able to impersonate her to another?
- What about choosing a name that won’t collide on any systems (so you don’t have to remember what name you are on each system)
Passport/Liberty

- authenticate to “identity provider”
- other sites affiliate with it
- get information from the identity provider (who you are, your shipping address, your credit card number, …)
How it works

• go to merchant. Click on “pay with Liberty”
  Get redirected to liberty (where Liberty will be told where the user came from). Liberty asks for name/password, and puts parameter into URL, and gives browser cookie.

• Passport: one identity for all sites. Liberty, different identity. But can still correlate. (web bugs)
Cryptographic Handshakes

• Once keys are known to two parties, need a handshake to authenticate

• Goals:
  – Mutual authentication
  – Immune from replay and other attacks
  – Minimize number of messages
  – Establish a session key as a side effect
Challenge/Response vs Timestamp

Alice

I’m Alice

compare:

R

f(R,K)

Bob

I’m Alice

compare:

K{R}

R
Challenge/Response vs Timestamp

Alice

I’m Alice

Bob

R

\{R\}_K

I’m Alice, \{timestamp\}_K

vs:

compare:
Issues

- Suppose a server has millions of connect requests, how to ensure no replay?
- Suppose want to be stateless?
- Suppose R is predictable?
- Suppose simultaneously giving out R’s, but don’t yet know who they’re going to?
Pitfalls with Public Key

I’m Alice

R

R signed with private key

This might trick Alice into signing something, or possibly decrypting something
Eavesdropping/Server Database Stealing

- pwd-in-clear, if server stores \( h(pwd) \), protects against database stealing, but vulnerable to eavesdropping
- Standard challenge/response, using \( K = h(pwd) \), foils eavesdropping but \( K \) is pwd-equivalent so server database vulnerable
- Lamport’s hash solves both
Mutual Authentication

Alice  Bob

I’m Alice

→

R1

←

{R1}_K

←

R2

→

{R2}_K
More Efficient Mutual Authentication

Alice
I'm Alice, R2
\{R1\}_K

Bob
R1, \{R2\}_K
\{R1\}_K
Reflection Attack

Trudy

I’m Alice, R2

R1, \{R2\}_K

start a second parallel connection

I’m Alice, R1

R3, \{R1\}_K

complete the first \{R1\}_K
Timestamp Based Mutual Authentication

Alice

Bob

I’m Alice, \(\{\text{timestamp}\}_K\)

I’m Bob, \(\{\text{timestamp}\}_K\)

Two messages instead of three
Must assure Bob’s timestamp is different
How does server know Alice’s secret key?

- individually configured into each server
- authentication storage node (servers retrieve it from there)
- authentication facilitator node (does the authentication and answers yes/no)
- hopefully doesn’t store password or password-equivalent
Predictable vs unpredictable nonce: both cases

Alice

I’m Alice

R

{R}_K

Bob

I’m Alice

{R}_K

R
Can be predictable

Alice           Bob

\[ \{R\}_K \]
\[ \{R\}_{K+1} \]
Generating Session Key

- Could use $K$ all the time (frowned on)
- Couldn’t use $\{R\}_K$
- Could use $\{R\}_{K+1}$
- What about $\{R+1\}_K$
- Good if both sides contribute
  - avoid replay
  - works if either side has good random # gen
Needham-Schroeder

Alice     KDC     Bob

Alice wants Bob, \( N_1 \)

\[ \{ N_1, \text{"Bob"}, K_{ab}, \{\text{"Alice"}, K_{ab}\}K_b \} K_a \]

\[ \{\text{"Alice"}, K_{ab}\}K_b, \{N_2\}K_{ab} \]

\[ \{N_2-1, N_3\}K_{ab} \]

\[ \{N_3-1\}K_{ab} \]
Perfect Forward Secrecy

- Also escrow-foilage
- One way to do it: Diffie-Hellman
- Any others?
Anti-clogging protection

• cookies; stateless based on?
  – IP address
  – timestamp
  – sequence number

• puzzles
Hiding endpoint identifier

- How with different kinds of keys?
- If hide one from active attacker, which one?
Reusing D-H exponentials

• OK to do that, at some loss of PFS
• To avoid replays, have nonces
• Stateless cookie isn’t usually a nonce (since will be reused). But lots of ways to make stateless cookies work.
Adding a message to speed things up!

Alice

\[ g^A \mod p \]

Bob

\[ g^B \mod p \]

\[ [h(g^{AB} \mod p)]_{Bob} \]

\[ [h(g^{AB} \mod p)]_{Alice} \]
Session Resumption

- Initial handshake is expensive (with public keys)
- How to resume a session. When do these work?
  - SSL way--session #
  - Lotus Notes way--f(name)
  - DASS way--send session key signed and encrypted
Plausible Deniability

• What if want to deny conversation happened?
  – secret keys
  – public encryption keys
  – signature keys (depends on if sign other side’s identity)
Crypto Negotiation

- Why: exportability, security/performance, new algorithms, vanity algorithms
- Have to make sure there’s no downgrade attack by an active attacker
Exportability

• Old rules: only allow 40 bit encryption keys, only allow 512-bit RSA keys for sending it

• But want to be interoperable, and use strong crypto where possible (two domestic versions)
SSL/TLS

- Secure Socket Layer/Transport Layer Security
- Used with http and recently others
- As deployed implements server side authentication with X.509 certs
- Client side starting to appear
- CA infrastructure issues not worked out
SSL/TLS (simplified)

Client → **Initiate Request** → Server

- **Server Certificate**
- \( \text{Server’s public key} \)
- \( \{\text{Session key}\} \)
- \( \{\text{Data}\} \)
SSLv3/TLS (simplified)

Client → \text{Hello, ciphers I support, } R_A \rightarrow \text{Server}

\text{Certificate, cipher I choose, } R_B, \text{ session-id}

\{S\}_{\text{Server's public key, keyed hash of prev msgs}}

\{\text{keyed hash of handshake msgs}\}

\{\text{Data}\}_{\text{keys}}
Session Resumption

- Bob sends session-id in msg 2
- If Alice returns it in msg 1:
  - session-ID, ciphers I support, $R_A$
  - session-id, cipher I choose, $R_B$, \{keyed hash\}
  - \{keyed hash of prev msgs\}
  - \{data\}
Exportability, v2

- Always use 128-bit client master secret
- In exportable suite, send 88 bits in the clear, other 40 encrypted with other side’s RSA key
- But domestic servers had 1024-bit RSA keys, technically illegal to send 40-bit key in 1024-bit key
Exportability, v3

- computing 6 keys (IV, encryption, integrity, in each direction)
- use 40 bits of seed, hashed with nonces, to obtain 128-bit encryption keys
- use ephemeral small key
- Ironically, stronger than v2 since gives weak-key PFS
Allowing sometimes strong exportable crypto

- Server Gated Crypto/Step-Up, for things like banks
- Special cert, has to be signed by Verisign (not any of other trust anchors)…it’s who US govt trusts, not who client trusts
- do second handshake when Alice noticed Bob’s cert is a step-up
- awkward protocol allows server just to have cert, not special code
Client Auth

Client ———— Initiate Request ———— Server

server cert

{Session key}, Client cert, [MD all prev msgs]_{client PK}

{Data}_{Session key}
TLS

- SSL was Netscape
- Microsoft designed similar thing called PCT
- IETF, realizing bad for industry to have 2 similar things designed a third called TLS
- Based on SSL, but changed crypto suite to avoid encumbered technology
- SSL more widely deployed than TLS
Other details

• crypto negotiation is with suites, e.g.,
  – SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
  – version 2: Alice sends choices, Bob narrows it down, Alice makes final choice. version 3: Bob chooses

• version # kind of broken
  – no integrity protection on client hello, so attacker could change version # to 2
  – version 3 does very clever thing with PKCS1 padding (bottom 8 octets of padding=3)
Million Message Attack

• PKCS1 of pre-master secret is formatted with top 2 bytes 0002.
• Capture the encrypted premaster secret = c
• Pick a million s’s ($2^{20}$)
• Send server $c^s e$ for each s.
• it will let you know if the padding is correct (about $2^{16}$ messages will be)
• then (by magic) you can find the premaster secret if you have about 16 s’s