# Methods for Knowledge-Based Authentication

David Jablon

KBA Symposium
February 9-10, 2004

# David Jablon

- CTO, Phoenix Technologies
  - BIOS & beyond
- Developed SPEKE
  - A zero-knowledge password method
- Editor of IEEE P1363.2
  - Proposed standard for password-based cryptography

# Characteristics of knowledge

- Not <u>just</u> passwords & PINs
  - Also “relative secrets”: SSN, Mother’s name, …
- Quality of secrecy
  - Who shares it and how?
  - Can it be bought?
- Lifetime
  - Changed periodically vs. Fixed for life
- Size of secret
  - Length and randomness
  - Typically NOT equivalent to an 80-bit key.

# Trusted path

*Secret knowledge is precious*

- Don’t expose knowledge at Client
- Don’t expose knowledge on Server
- Don’t expose knowledge in Transit

Don’t reveal secrets to any wrong party.
Insure that what you see is from who you expect.

# Ancient history of KBA methods

- Standalone: password file **`/etc/passwd`**
  - Stored public hashed password
- Standalone: shadow password file
  - Stored secret hashed password
- Network: CHAP, Kerberos
  - Transmitted public hashed password

***Moral***: Hashed password ≅ password, due to dictionary / brute-force attacks.

# Recent history of KBA methods

- Using SSL/TLS tunnels
- **Zero-knowledge password proofs**
  - IEEE P1363.2: AMP, PAK, SPEKE, SRP, ...
- Multi-server password systems
  - Ford & Kaliski, Jablon, Nightingale
  - Refinements: more groups, error handling, ...
- Hardened clients
  - Other work in progress (not covered here)

# What’s wrong with a browser’s server-only SSL tunnel?

- User might not check SSL icon
- User might not check certificate
- User might not see a misspelled name or URL
  - Server spoofing attacks
- Mistakes in trust interpretation
- User might enter the wrong password

***What’s good about SSL?***

  - Can help protect usernames, challenge questions.

# The SPEKE trick

| *Client* | *Server* |
|---|---|
| small secret $\pi$ | small secret $\pi$ |
| one-time big secret $X$ | one-time big secret $Y$ |
| $\pi^X \rightarrow$ | |
| | $\leftarrow \pi^Y$ |
| $K = (\pi^Y)^X$ | $K = (\pi^X)^Y$ |

Converts small secret $\pi$ into big secret $K$.

Uses a big prime order group (e.g. integer multiply mod $q$).

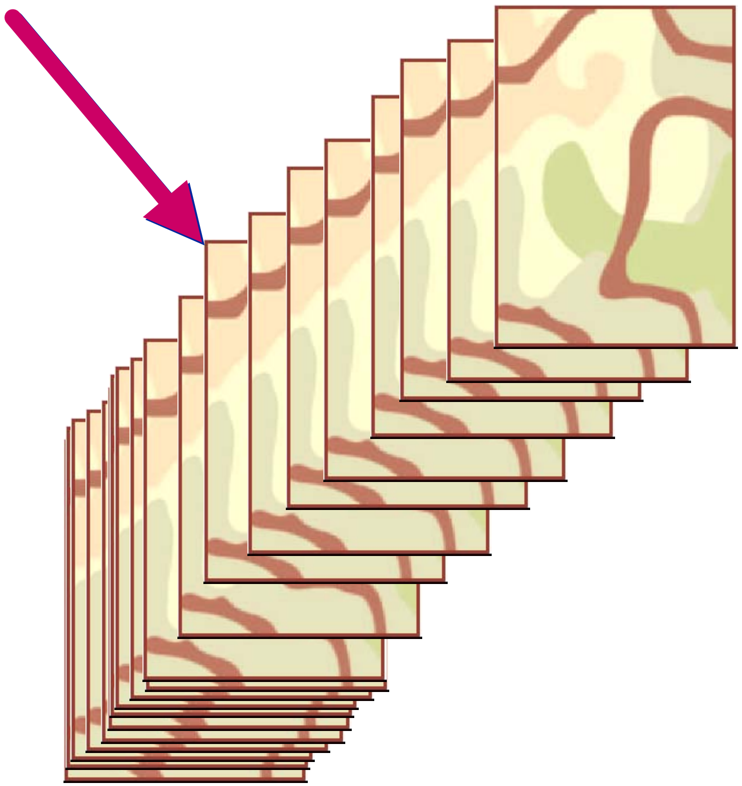
# A zero knowledge password proof demo with a deck of cards

- Choose a fair deck of cards
- Alice and Bob jointly shuffle it
- They share knowledge of a small secret number
  - (or maybe not)
- Commitment:
  - They blindly select cards based on their numbers
- Revelation:
  - They disclose the selected face values

*Objective*: Prove whether they know the same number without revealing their knowledge.

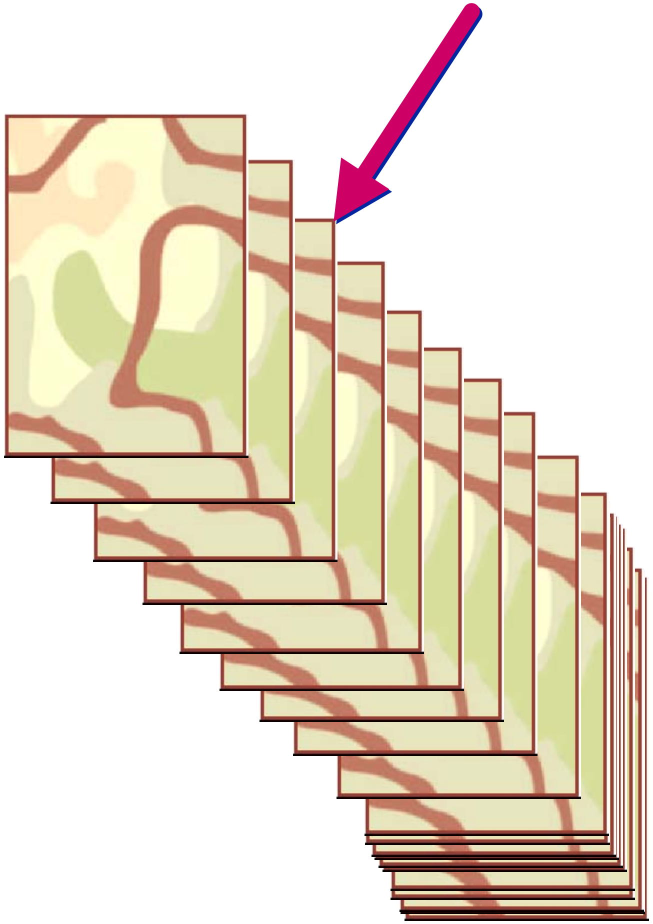
# Commitment phase (1)

Alice's password is "8", so she secretly peeks at the 8th card only.

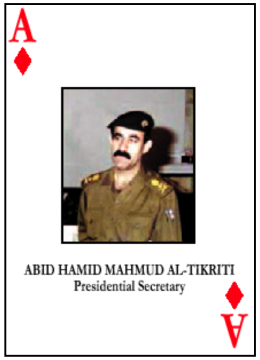


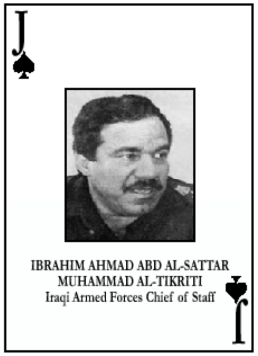
# Commitment phase (2)

Bob guesses “3”, so he secretly peeks at the 3rd card only.



# Revelation phase:  Do they match?




Sorry. “Abid Hamid Mahmud Al-Tikriti” ≠
“Ibrahim Ahmad Abd Al-Sattar Muhammad Al-Tikriti”.

- Alice learns only that Bob’s secret isn’t 8.
- Bob learns only that Alice’s secret isn’t 3.

The most efficient ZKPPs allow only 1 guess per run.
(The ZKPP crypto protocols don’t need a Referee.)

# Benefits of zero knowledge password methods

- Simultaneous mutual authentication
  - Eliminates trust gap
- Active authentication
  - A step that can’t be skipped
- Even small secrets are not revealed
  - A wrong party won’t get knowledge intended for another

# Combining benefits

- ZKPP network protocol
  - User's secret knowledge doesn't need to be transmitted to mutually authenticate.
- Hardened server
  - Multiple-servers removes single server point of failure.
- Hardened client
  - Knowledge may be contained by a secure keypad and CPU.

# Split a password among multiple servers

- Knowledge verification data for $\pi$ is split into multiple parts ( $S_1$, $S_2$, ... ).
  - All Servers must collude to get a chance to crack $\pi$.

- Client uses master key $K_\mathrm{m}$ to encrypt stuff
  - Leverages secure memorized knowledge to protect non-memorized stuff.

# Example of a zero-knowledge multiserver method

- User chooses a secret password, call it $\pi$.
- Stores key "shares", $S_1$, $S_2$ , ... with different servers.
- Protocol retrieves user's key $K_m = f(\pi, S_1, S_2, \ldots)$
- User needs $\pi$ to retrieve $K_m$
- Does not need other keys or certificates.
- Does not reveal $\pi$, $S_1$, $S_2$ , ..., or $K_m$.

# Another neat trick ...

*Client knows*

$\pi$ = a small secret (password)

$X$ = a one-time big secret

*Server knows*

$S$ = a big secret

$\pi^{X} \quad \rightarrow$

$\leftarrow \; \pi^{X\,S}$

$K = \pi^{X\,S\,1/X}$

$K = \pi^{S}$

Converts a small secret $\pi$ into big secret $K$.

Uses a big prime order group (e.g. integer multiply mod $q$).

# ... do it twice

[Ford & Kaliski 2000]

*Client* $\pi$

$\pi^X \rightarrow$

$K_1 = \pi^{X S_1 \, 1/X} = \pi^{S_1}$

$K_2 = \pi^{X S_2 \, 1/X} = \pi^{S_2}$

$K_m = hash(K_1 \,||\, K_2)$

*Server 1* $S_1$

$\leftarrow \pi^{X S_1}$

*Server 2* $S_2$

$\leftarrow \pi^{X S_2}$

# ... test $K_m$ before using it

*Client* $\pi$

*Servers* $S_1$ $S_2$

$\pi^X \rightarrow$

$\leftarrow \pi^{X S_1}$

$\leftarrow \pi^{X S_2}$

$\leftarrow V = hash(K_m)$

$K_m = hash(\pi^{S_1} \,||\, \pi^{S_2})$

If $hash(K_m) \neq V$, abort.

(Must not reveal any $f(K_m)$.)

# ... then, sign $\{\pi^X\}$

*Client* $\pi$

*Servers* $S_1$ $S_2$

$\pi^X \rightarrow$

$\leftarrow \pi^{X S_1}$

$\leftarrow \pi^{X S_2}$

$K_m = hash(\pi^{S_1} \| \pi^{S_2})$

$\leftarrow V = hash(K_m)$

If $hash\,(K_m) \neq V$, abort.

(... tick, tock, tick tock, ...)

$Sign_{\mathrm{UserPrivateKey}}\{\pi^X\} \rightarrow$

If no valid signature received, log failure.

# *Other refinements:* Extended to use other groups

- Smaller subgroups of $GF(p)^*$
- $GF(2^m)$
- $GF(p^m)$
- Elliptic curve groups
- P1363.2 leverages earlier and ongoing work
  - IEEE 1363-2000, P1363a

# *Other refinements:* Forgiveness protocol

Scene: User mis-types a few bad passwords, $\pi_1$, $\pi_2$, ..., but eventually gets it right.

Goals:

(1) Limit number of guesses an attacker can make over a long term.

(2) Don't punish a clumsy user by counting all mistakes against a long-term limit.

# *Other refinements:* Forgiveness protocol

Method:

- Client digitally signs & sends prior mistaken values to each Server.
  - $privateKey_{\mathrm{User}}\{\pi_1^{X_1}, \pi_2^{X_2}, \ldots\} \rightarrow$ Server 1, Server 2, ...
- Each Server forgives mistakes made within a session that eventually succeeds.

Forgiven mistakes don't count against a long-term bad guess limit.

# Summary of features and benefits

- One example of a multi-server ZKPP
- Client tests $K_m$ before using it in public and signs $\pi^X$ to prove she's real
    - No need for SSL to protect $\pi$
- Extended to use other groups
    - More options for security & performance
- Forgiveness protocol
    - Improved error handling

# Metrics for KBA methods

- Performance metrics may be qualitative
  - Does it resist attack $x$? [*yes*/*no*]
  - If needed, how many users check certs? [$x$%]
- Metrics may drive authentication policy
  - Est. knowledge size $\rightarrow$ Limit # of bad guesses
  - Est. knowledge lifetime $\rightarrow$ Min. required size
- Limits for short-term and long-term errors may depend on the method

# For more information ...

- IEEE P1363.2
  - Proposed standard for password-based techniques
- Research papers
  - www.integritysciences.com/links.html
- Phoenix Technologies
  - www.phoenix.com
- David Jablon
  - dpj@theworld.com
  - +1 508 898 9024