Password-Based Private-Key Download Protocols

Radia Perlman **Charlie Kaufman**

radia.perlman@sun.com ckaufman@iris.com

Goals

- "The network is the computer"
- WS has some minimal amount of (trusted) software installed
- No user-specific info configured on WS
- User Alice's private key and other info stored in central place "Bob" (e.g., the directory)
- "Log into the network" means get Alice's private key, so WS can authenticate as Alice
- Download the key knowing only a password
  - no off-line pwd guessing by eavesdropper, Alice-impersonator, or Bob-impersonator
  - minimal messages
  - minimize computation for Bob, stateless Bob
  - allow for "salt"

Unavoidable Vulnerabilities

- Someone that knows (or can guess) Alice's password can impersonate Alice to Bob, get Alice's security info from Bob and then impersonate Alice to the world
- Someone that knows (or can guess) Alice's password can impersonate Bob to Alice and trick her into using the wrong private key, trusting the wrong CA's, etc.
- Someone who can read Bob's database can do off-line, unaudited, password guessing
- Alice-impersonator can do on-line guessing. Bob can't distinguish this from user mistyping
- One side, X, discovers first whether the other side is legitimate. X-impersonator can get one "free" on-line pwd guess by breaking off communication

Building Blocks

- Diffie-Hellman
- EKE (Bellovin-Merritt)
  - encrypt Diffie-Hellman exchange with W (W=h(pwd))
- SPEKE (Jablon)
  - replace base in Diffie-Hellman exchange with W
- Both EKE and SPEKE are 4-message protocols designed for mutual authentication and agreeing on a strong session key S

EKE

W=h(pwd), g, p

Alice ◄- - - - - - - - - - - - - ► Bob

pick A

$\{g^A \bmod p\}W$ →

pick B
decrypt $\{g^A \bmod p\}W$
calculate $K=g^{AB} \bmod p$
choose challenge C1

← $\{g^B \bmod p\}W, \{C1\}K$

choose challenge C2

$\{C1,C2\}K$ →

← $\{C2\}K$

SPEKE

W, g, p

Alice Bob

pick A

W^A mod p

pick B

calculate $K=W^{AB}$ mod p

choose challenge C1

W^B mod p, {C1}K

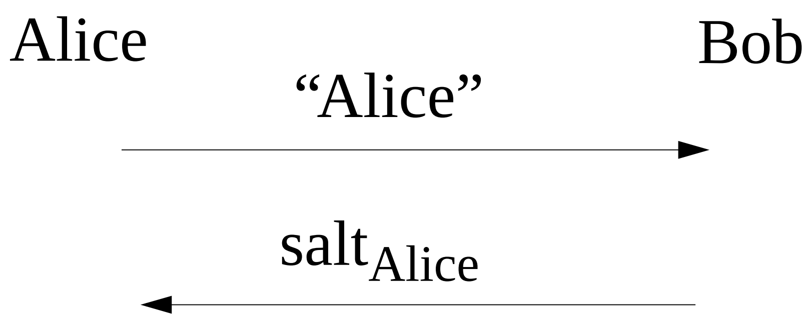
choose challenge C2

{C1,C2}K

{C2}K

Downloading Private Key

- Call private key encrypted with password "Y"
- Bob could send {Y}K in 4th message
- No salt: Reading Bob's database, can check W for many users against one database of guessed passwords
- Could add salt by adding two messages to the front



- Simplifications for our use:
  - no need to authenticate Bob
  - get rid of C1 (just prove knowledge of K)

Basic EKE-Based 4-msg

W, g, p

Alice: W, Y

Alice — Bob

pick A

$\{g^{A} \bmod p\}W$

pick B

decrypt $\{g^{A} \bmod p\}W$

calculate $K=g^{AB} \bmod p$

$\{g^{B} \bmod p\}W$

calculate $K=g^{AB} \bmod p$

h(K)

verify h(K)

{Y}K

Basic SPEKE-based 4-msg

Alice: W, Y

W, g, p

Alice ⟵ - - - - - - ⟶ Bob

pick A

W^A mod p ⟶

pick B

calculate $K=W^{AB}$ mod p

⟵ W^B mod p

calculate $K=W^{AB}$ mod p

h(K) ⟶

verify h(K)

⟵ {Y}K

Notes

- Ted, impersonating Bob, gets one unaudited guess by breaking off communication after msg 3
- An eavesdropper gains no information
- Someone impersonating Alice gets one "on-line" guess, i.e., can't verify guess without letting Bob know she guessed wrong
- Salt would be nice, so computation to produce W must be done per user, per password
- A "cookie" would be nice, so Bob does no significant computation if you can't receive from the IP address you sent from
- Both salt and cookie can be added easily to produce a 6-message protocol

SPEKE-based, 6-msgs

Alice: $salt_{Alice}$, W, Y

W=f(pwd, salt), g, p

Alice ⟷ Bob

"Alice" →

choose cookie

$salt_{Alice}$, cookie ←

pick A

cookie, W^A mod p →

pick B

calculate $K=W^{AB}$ mod p

W^B mod p ←

calculate $K=W^{AB}$ mod p

h(K) →

verify h(K)

{Y}K ←

Notes

- No security lost if Bob uses same B every time for a particular user (but different B per user)
- If we're picking B per user in advance, we can precompute $\{g^B \bmod p\}W$ for EKE and $W^B \bmod p$ for SPEKE
- EKE: if Alice sends $g^A \bmod p$ unencrypted rather than $\{g^A \bmod p\}W$, then Bob can store B, $\{g^B \bmod p\}W$ and does not need to store W
- SPEKE: If Bob stores B, $W^B \bmod p$, then he doesn't need to store W
- This accomplishes the same thing as salt!
- And it saves computation for Bob!
- And it avoids the first two messages!

Making Bob Stateless

- Bob doesn’t have to remember what B he used, since it’s always the same one for Alice
- Include Alice’s name in each msg from Alice
- Make cookie a function of Alice’s IP address and a secret known only to Bob
  - cookie=h(IP address, Bob’s secret)
- Bob can change his secret often (like every 5 minutes) and always accept one of 2 values

Stateless, precompute, EKE-based, 4-msgs

“Alice”, B, $\{g^B \bmod p\}W$, Y

Alice → Bob

“Alice” →

calculate cookie=h(IP add, Bob’s secret)

← $\{g^B \bmod p\}W$, cookie

choose A

calculate $K=g^{AB} \bmod p$

“Alice”, cookie, $g^A \bmod p$, h(K) →

verify cookie

calculate $K=g^{AB} \bmod p$

verify h(K)

← {Y}K

Stateless, precompute, SPEKE-based, 4-msgs

"Alice", B, W^B mod p, Y

Alice → Bob: "Alice"

calculate cookie=h(IP add, secret)

Bob → Alice: W^B mod p, cookie

choose A
calculate $K=W^{AB}$ mod p

Alice → Bob: "Alice", cookie, W^A mod p, h(K)

verify cookie
calculate $K=W^{AB}$ mod p
verify h(K)

Bob → Alice: {Y}K

Two Messages, EKE-based

- Bob stores, for Alice:
  - "Alice"
  - W=h(pwd)
  - Y=private key encrypted with password
  - B ;to save computation
  - $\{g^B \bmod p\}W$;to save computation

Alice Bob

"Alice", $\{g^A \bmod p\}W$ →

← $\{g^B \bmod p\}W$, $\{Y\}K$

Notes

- No salt. Bob needs to store W
  - If Alice sent g^A mod p, someone impersonating Alice could do off-line password guessing
  - So Bob needs W do decrypt $\{g^A \bmod p\}W$
  - Conceivably the user's name can act as salt W=h(name,pwd), but problematic if user has aliases, or name changes
- No cookie. Every message requires Bob to compute K
- Someone impersonating Bob gets no info about W
- Someone impersonating Alice gets one unaudited on-line password guess

Two Msgs, SPEKE-based

- Bob stores, for Alice:
  - "Alice"
  - B
  - W^B mod p
  - Y=private key encrypted with password

Alice → Bob: "Alice", W^A mod p

Bob → Alice: W^B mod p, {Y}K

Notes

- Better than 2-msg EKE-based because it gets the advantage of salt!
- Same other disadvantages (relative to 4-msg protocols) as 2-msg EKE-based
  - Alice gets a single unaudited on-line guess
  - no cookie

Retrieving User's Security Context

- WS needs other info, like keys of CAs she trusts, her certificate, etc.
- Store it signed (and encrypted if necessary) with Alice's key
- If info changes, include a timestamp or version number and display to the user (to prevent tricking WS into using old security context info)
- Or Alice can sign certificate trusting some administrator's signature on her security context. Store info signed by admin's key and encrypted (if necessary) with Alice's key

Summary

- We present 4-msg protocols for downloading private key and user's security context:
  - no off-line guessing
  - minimize server computation
  - denial of service protection of cookie
  - equivalent advantage of salt
  - stateless server (Bob can act in request-response mode)
  - Bob gets one unaudited on-line guess
- We present 2-msg protocols:
  - lose cookie protection
  - Alice gets one unaudited on-line guess
  - in EKE-based, lose salt. SPEKE-based we keep salt advantage