CS 465
Computer Security

Passwords

Daniel Zappala, adapted from Kent Seamons
Fall 2018

Goals

- Understand how passwords are hashed and salted
- Understand basic attacks on password databases
- Understand Lamport's hash and its vulnerabilities
- Understand the objectives of PAKE protocols and how they work

How to Attack Password Systems

- Guess the user’s password
    - Online attack: attempt to login as the user would
    - Offline attack: repeated guessing involving an encrypted form of the user’s password
- Shoulder surfing
- Users write down their passwords
- Users give away their passwords
    - Phishing, social engineering

Problems with Passwords

- Users have too many passwords
    - Encourages password reuse
    - Leads to forgotten passwords
    - Burdens users and administrators
- Attempts to increase password strength inconvenience users

Time estimates

- What is the maximum number of attempts to guess a password?
    - Password length = 8 characters
    - Assume password is alphanumeric (26+26+10)
    - $(26+26+10)^8 = 62^8$
- How many attempts on average? Divide maximum number by 2 (this assumes brute force attack and passwords chosen randomly)

Unix Passwords

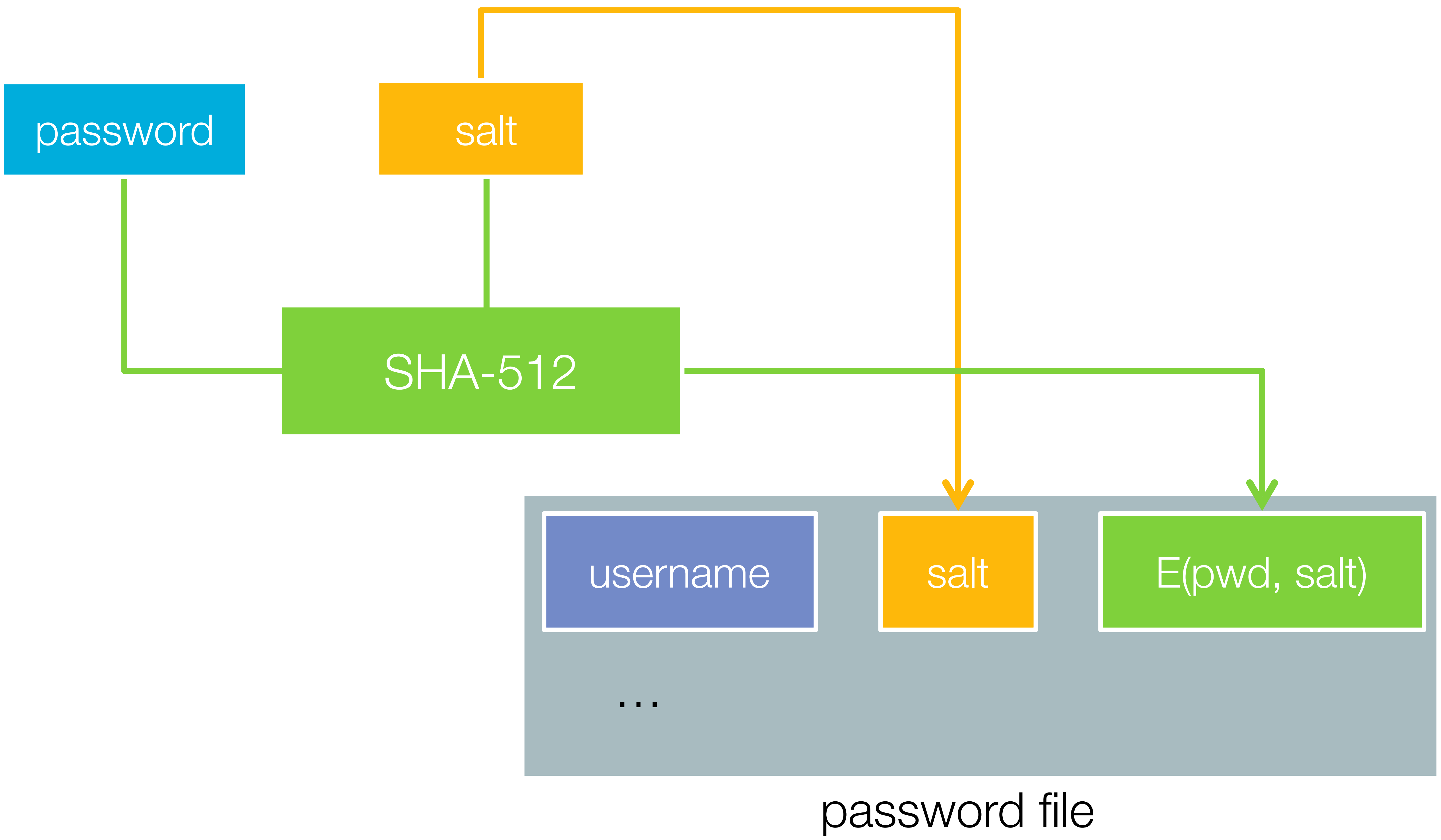
History of UNIX Passwords

- Version 1: passwords stored in plaintext: /etc/passwords
  - Anyone who can seal the password file gets all the passwords
- Version 2: encrypt passwords in the file
  - Originally, the password file was world readable
  - Anyone could copy the file offline and perform a dictionary attack
  - You could find sample files on Google courtesy of naïve system admins!

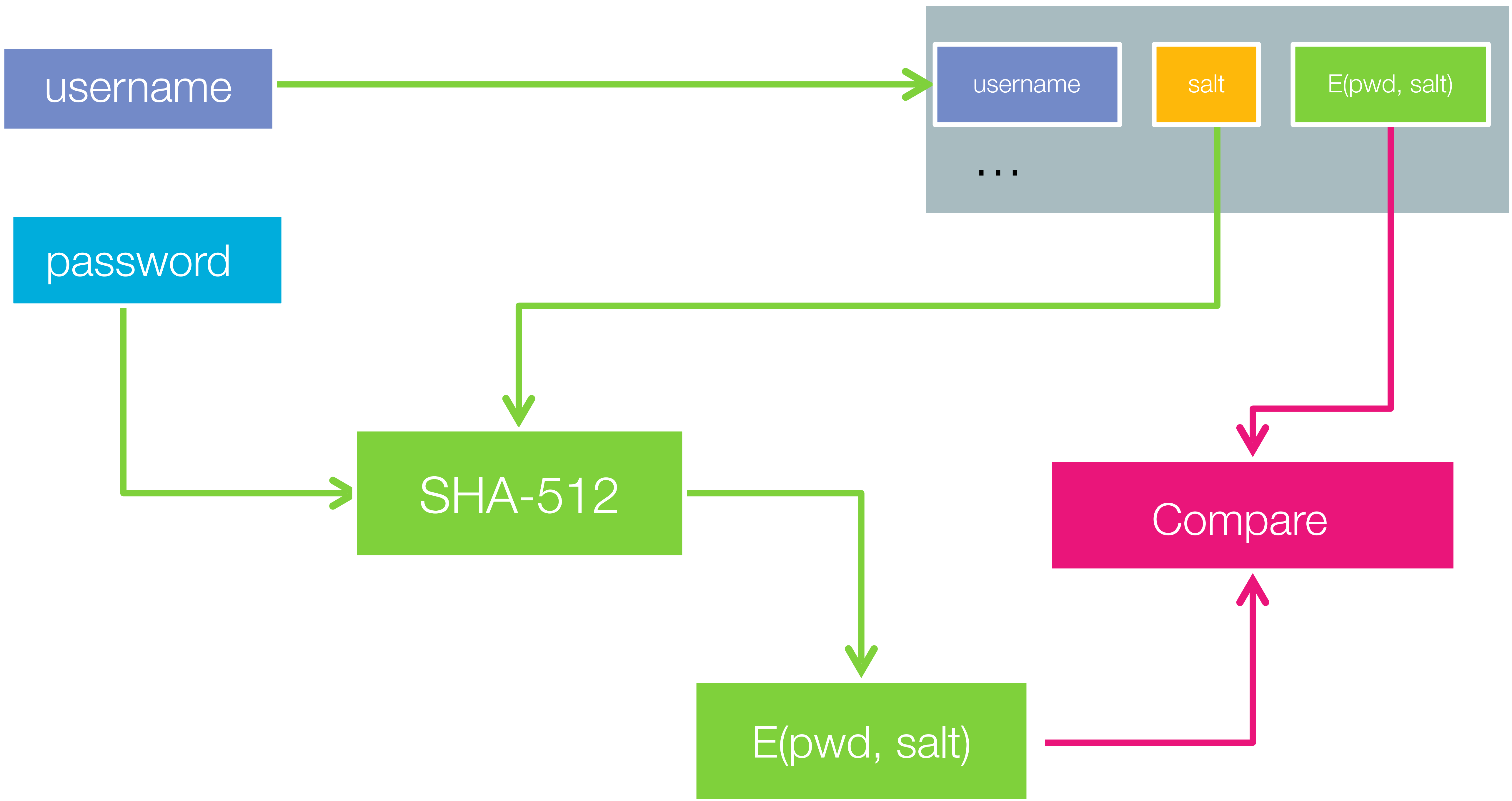
History of UNIX Passwords

- Version 3: separate shadow password file
  - /etc/passwords is world readable but does not have passwords
  - /etc/shadow is readable only by root, contains encrypted passwords

Unix Password File Creation



Verifying a Password



SHA-512

- cryptographic hash function
- the salt adds randomness and is different per user, so that even if two users choose the same password their encrypted passwords differ
  - guess made with one user's salt aren't helpful for another, increases the cost of offline attack to crack any password in the file, increases the size requirement for a pre-computed database of hashed passwords
- https://www.slashroot.in/how-are-passwords-stored-linux-understanding-hashing-shadow-utils

Passwords on other systems

- Mac OS: shadow file per user
  - /var/db/dslocal/nodes//Default/users
- Windows: shadow file
  - c:\Windows\System32\Config\

Password Cracking

Basic Password Cracking Attacks

- Brute force
  - Go through every possible password, use the salt (from the stolen shadow file), hash them, and see if the hash matches — must repeat separate for each user
- Dictionary
  - Same but use dictionary words
  - See also: https://arstechnica.com/information-technology/2013/10/how-the-bible-and-youtube-are-fueling-the-next-frontier-of-password-cracking/
- Substitution
  - Try common patterns, like password, passw0rd

Rainbow Table Attack

- Create a massive table of precomputed tables of hashed values
- If you find a match with a given user’s hash, it may not be their actual password (due to a collision), but this doesn’t matter
- Trades storage for computation time

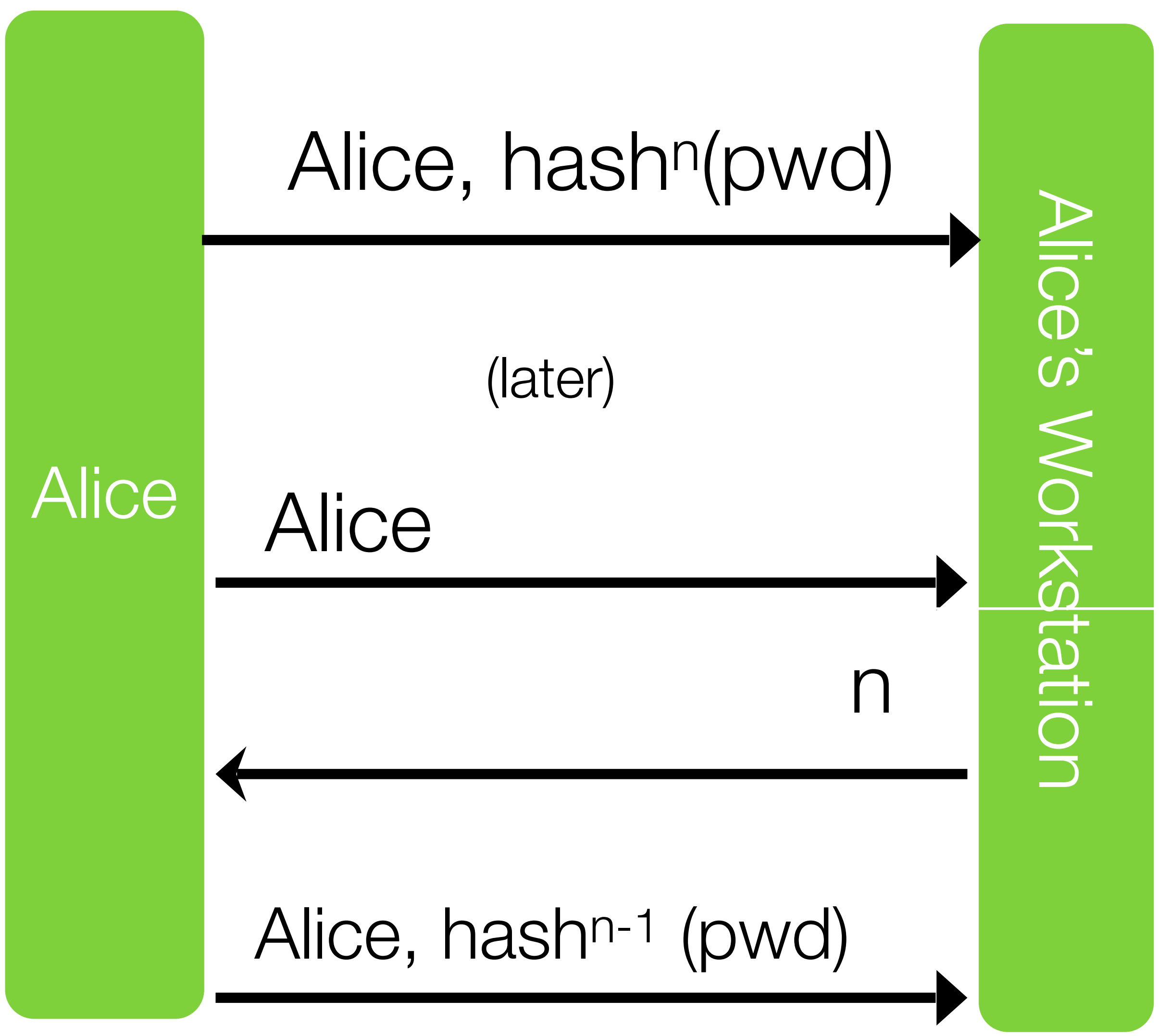
Impact of Salt on Attacks

- How many guesses do password attacks need when a salt is used?
  - Off-line attack – one attempt for each unique salt in the file
- How does the salt impact on-line attacks?
  - It doesn't
- How does the salt impact an attempt to crack a specific user's password in the file?
  - It doesn't change the number of attempts, but it does increase the size of a pre-computed database of passwords or rainbow table

Lamport's Hash

Lamport's Hash

- One time password scheme



Workstation checks for $hash(hash^{n-1}(pwd)) = hash^n(password)$

http://merlot.usc.edu/cs530-s07/papers/Lamport81a.pdf

Attacks on Lamport's Hash

- Small n attack
  - Active attacker intercepts server's reply message with n and changes it to a smaller value
  - Attacker can easily manipulate the response (repeatedly) to impersonate Alice
- Eavesdropper captures Alice's hashed reply and conducts off-line attack
- Replay Alice's response to other servers where Alice may use the same password
  - Thwart using salt at the server – server hashes pwd || salt and sends n and the salt to Alice during login
  - Salt also permits automatic password refresh when n reaches 1
- How many of these are thwarted by TLS?

Related articles (optional)

- The Curse of the Secret Question
  - http://www.schneier.com/essay-081.html
- Sarah Palin Yahoo! account hacked
  - http://www.informationweek.com/news/security/cybercrime/showArticle.jhtml?articleID=210602271
- Secret Questions Too Easily Answered
  - http://www.technologyreview.com/web/22662/
- Scientists claim GPUs make passwords worthless
  - http://www.pcpro.co.uk/news/security/360313/scientists-claim-gpus-make-passwords-worthless
- How the Bible and Youtube are fueling the next frontier of password cracking
  - http://arstechnica.com/security/2013/10/how-the-bible-and-youtube-are-fueling-the-next-frontier-of-password-cracking/
- 32 million passwords show most users careless about security
  - http://arstechnica.com/security/2010/01/32-million-passwords-show-most-users-careless-about-security/

PAKE Protocols

Password-authenticated key agreement (PAKE)

- Two parties establish a cryptographic key based on knowledge of a password
- Eavesdropper or man-in-the-middle cannot gain enough info to be able to brute-force guess a password
- Strong security even with weak passwords
- When used for login, password is **not** revealed to server, and server stores only a hash
- Numerous PAKE protocols proposed

How PAKE Protocols Work

- password, or hash of password, known by server
- after a login attempt (valid, or invalid) both the client and server should learn only whether the client's password matched the server's expected value, and no additional information
- standard protocol includes a key exchange (like DH)
- a "login" protocol can simply check that both parties have arrived at the same key, e.g. by having the parties compute some cryptographic function with it and check the results

Secure Remote Password (SRP) protocol

Registration

- Client computes

```
x = Hash(salt, passwd)    (salt is chosen randomly)
v = g^x                   (computes password verifier)
```

- Server stores (userID,v,salt) in database

https://blog.cryptographyengineering.com/should-you-use-srp/

Secure Remote Password (SRP) protocol

Compute Key

```
Client -> Svr:  User ID, A = g^a           (identifies self, a = random number)
Svr -> Client:  salt, B = kv + g^b         (sends salt, b = random number)

         Both:  u = H(A, B)

       Client:  x = Hash(salt, passwd)        (user enters password)
       Client:  S = (B - kg^x) ^ (a + ux)     (computes session key)
       Client:  K = H(S)

          Svr:  S = (Av^u) ^ b             (computes session key)
          Svr:  K = H(S)
```

both parties compute g^(ab + bux)

(g^a is mod p)

https://blog.cryptographyengineering.com/should-you-use-srp/

Secure Remote Password (SRP) protocol

Verify Key

```
Client -> Svr:  M = H(H(N) xor H(g), H(I), salt, A, B, K)
Svr -> Client:  H(A, M, K)
```

https://blog.cryptographyengineering.com/should-you-use-srp/

OPAQUE

- does not reveal salt to the attacker (this avoids pre-computation attacks)
- can be implemented using efficient elliptic curves
- works with any hashing function
- all hashing done on the client
- security proof

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

OPAQUE

Generating Secret

- keeping the salt secret, while ensuring only the client has the password

```
The server stores "salt", and the client has the password.

salt2 = PRF(salt, password) // This is calculated between the
                            // client and server, using an oblivious
                            // protocol where the client never learns
                            // salt, and the server never learns
                            // the password. The client obtains salt2

K      = PasswordHash(salt2, password) // This is done on the client
```

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

OPAQUE

Registration

- client generates a public and private key
- server also has a public, private key
- client computes
  C = Encrypt(**K**, client private key | server public key)
- server stores **C**, server's private key, salt

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

OPAQUE

Authentication

- server and client run the PRF protocol so that client generates **K**, only server knows salt, only client knows password
- server sends **C** to client, client uses **K** to decrypt **C**
- server and client run a key exchange protocol using their keys

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/