Lecture 12: Passwords

CS 5430

3/14/2018

Where we were…

- **Authentication:** mechanisms that bind principals to actions

- **Authorization:** mechanisms that govern whether actions are permitted

- **Audit:** mechanisms that record and review actions

Where we were…

- **Authentication:** mechanisms that bind principals to actions
  - Authenticating Humans
  - Authenticating Machines
  - Authenticating Programs

Where we were…

- Something you are
    fingerprint, retinal scan, hand silhouette, a pulse
- Something you know
    password, passphrase, PIN, answers to security questions
- Something you have
    physical key, ticket, {ATM, prox, credit} card, token

Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

1. PASSWORD CREATION

Who creates?

- **User:** typically guessable passwords
- **System:**
  - can produce hard-to-guess passwords (e.g., random ASCII character strings)
  - but users can't remember them
- **Administrator:** reduces to one of the above

Weak passwords

Top 10 passwords in 2017: [SplashData]

1. 123456
2. password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. letmein
8. 1234567
9. football
10. iloveyou

16: starwars, 18: dragon, 27: jordan23

Top 20 passwords suffice to compromise 10% of accounts [Skyhigh Networks]

Strong passwords

- How to characterize strength?
- Difficulty to brute force—"strength" or "security level"
  - Recall: if 2^X guesses required, strength is X
- Suppose passwords are L characters long from an alphabet of N characters
  - Then N^L possible passwords
  - Solve for X in 2^X = N^L
  - Get X = L $\log_2$ N
  - This X is aka entropy of password
    - Assuming every password is equally likely, X is the *Shannon entropy of the probability distribution* (cf. Information Theory)

Entropy estimation

- **Problem:** guide users into choosing strong passwords
- Entropy estimates [NIST 2006 based on experiments by Shannon]:
  - (assuming English and use of 94 characters from keyboard)
  - 1st character: 4 bits
  - next 7 characters: 2 bits per character
  - characters 9..20: 1.5 bits per character
  - characters 21+: 1 bit per character
  - user forced to use lower & upper case and non-alphabetics: flat bonus of 6 bits
  - prohibition of passwords found in a 50k word dictionary: 0 to 6 bits, depending on password length

Entropy of passwords

- Option A:
  - 8 character passwords chosen uniformly at random from 26 character alphabet
  - entropy of $8 \log_2 26 \approx 37$ bits
  - but that means abcdefgh equally likely as ifhslgqz
- Option B:
  - 1 word chosen at random from entire vocabulary
  - average high-school graduate: 50k word vocabulary
  - entropy of $\log_2 50\text{k} \approx 16$ bits

Entropy estimation

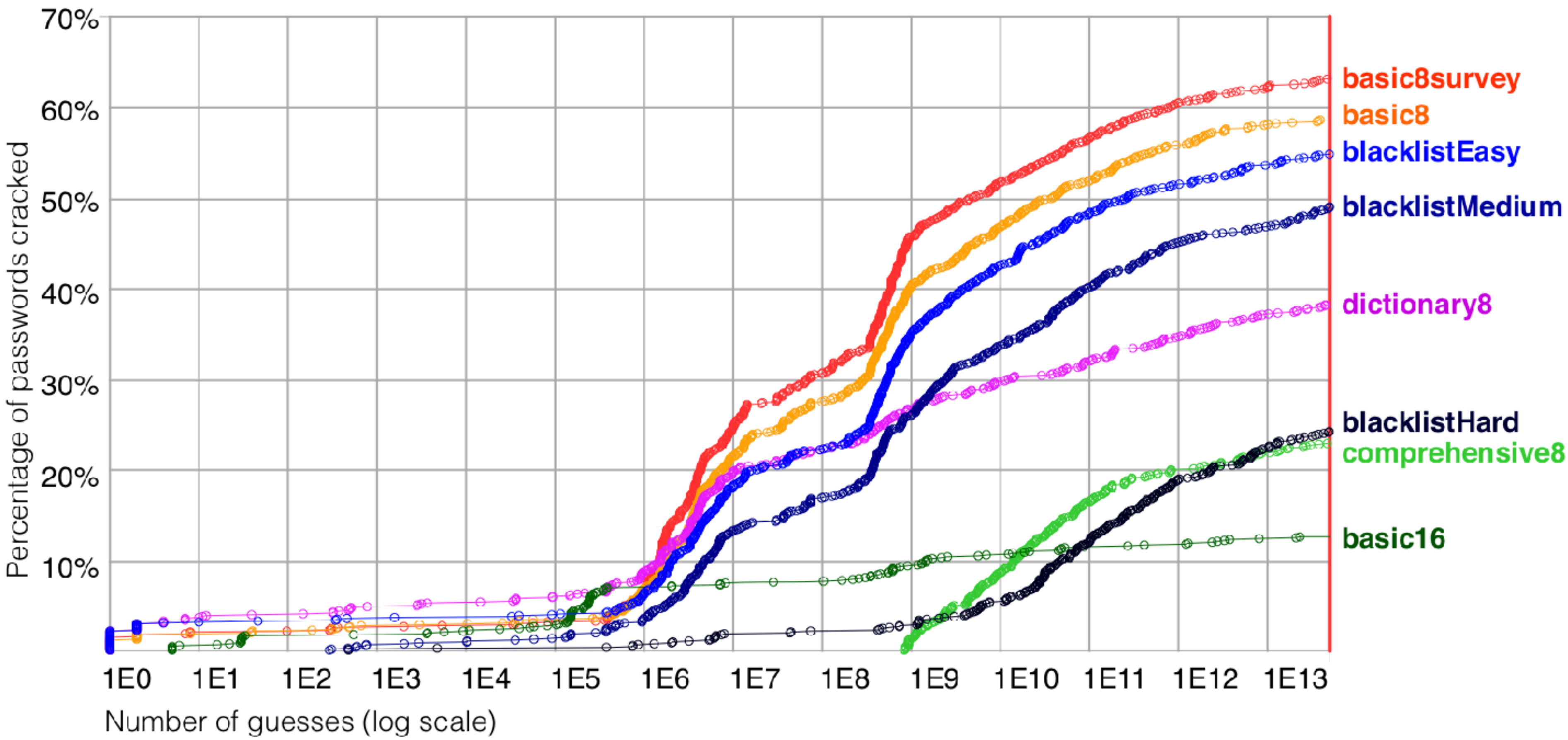
But:

- [Weir et al. 2010] based on cracking real-world passwords conclude *"[NIST's] notion of password entropy...does not provide a valid metric for measuring the security provided by password creation policies."*
- Underlying problem: Shannon entropy not a good predictor of how quickly attackers can crack passwords

Password Recipes

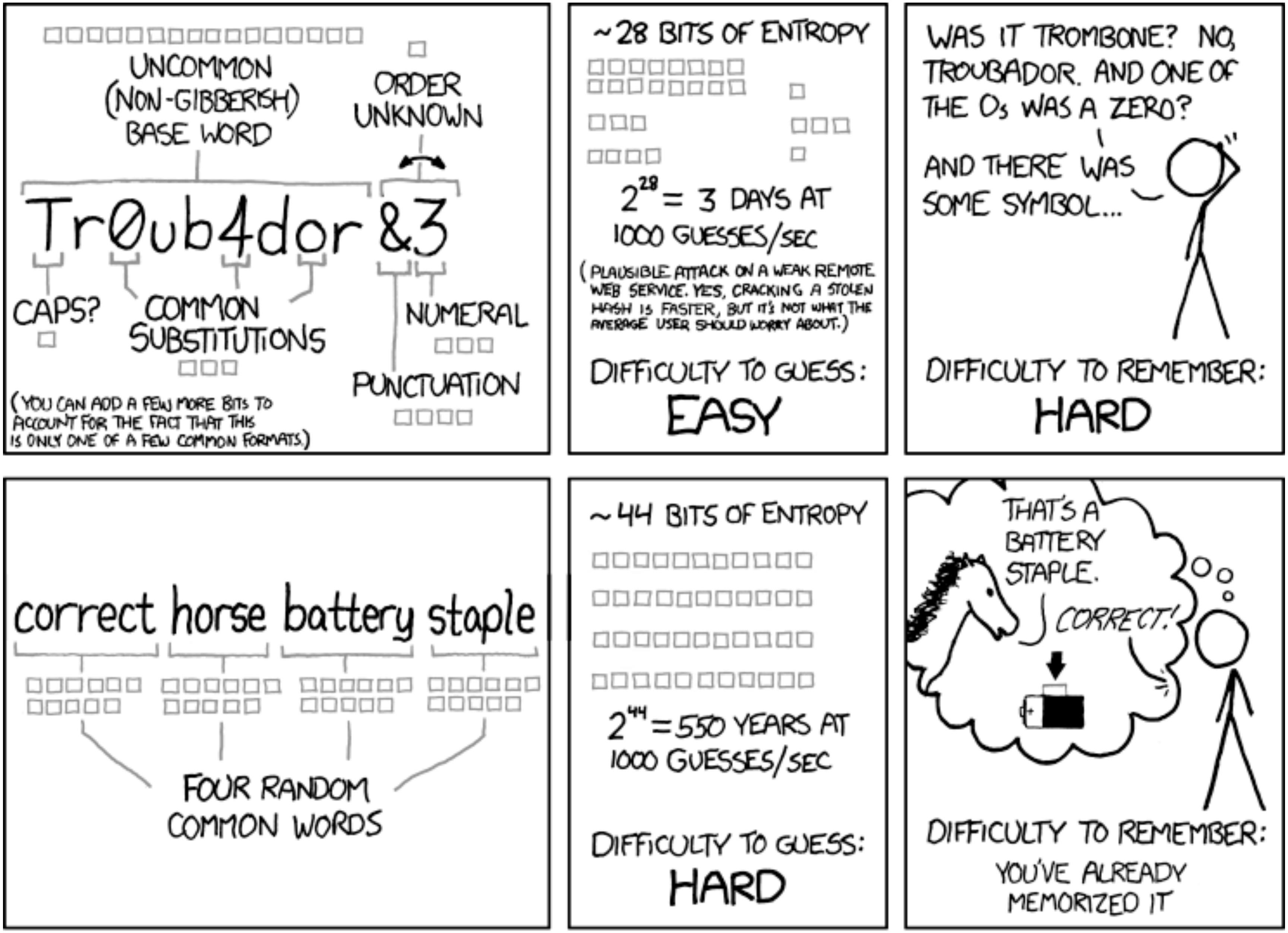
- Recipes: rules for composing passwords
  - e.g., must have at least one number and one punctuation symbol and one upper case letter
- [Kelley et al. 2012] Evaluate recipes based on
  - percentage of passwords cracked
  - number of guesses required to crack
  - for two state-of-the-art cracking algorithms, one of which is from [Weir et al. 2010] (same paper that invalidates Shannon entropy)
- Selected recipes:
  1. ≥ 8 characters
  2. ≥ 8 characters, no blacklisted words ...with various blacklists
  3. ≥ 8 characters, no blacklisted words from freely available 4M word common password + dictionary word list, one uppercase, lowercase, symbol, and digit ("comprehensive", c8)
  4. ≥ 16 characters ("passphrase", b16)
- Results...

Recipe comparison



Recipe comparison

- Comprehensive recipe (comprehensive8) makes it hard to crack passwords
  - Doesn't that contradict [Weir 2010]?
  - No: even if NIST's Shannon entropy estimates are quantitatively invalid **in general**, c8 in particular is hard to crack
- But blacklists make passwords almost as hard to crack
- And passphrases (basic16) are hard to crack and are more usable [Komanduri et al. 2011]:
  - Easier to create
  - Easier to remember
  - Threat to validity: maybe state-of-art crackers would improve to handle passphrases if people were required to use them



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Passwords

NIST (2017) recommends:

- minimum of 8 characters
- up to 64 characters should be accepted
- blacklist compromised values
- no other security requirements

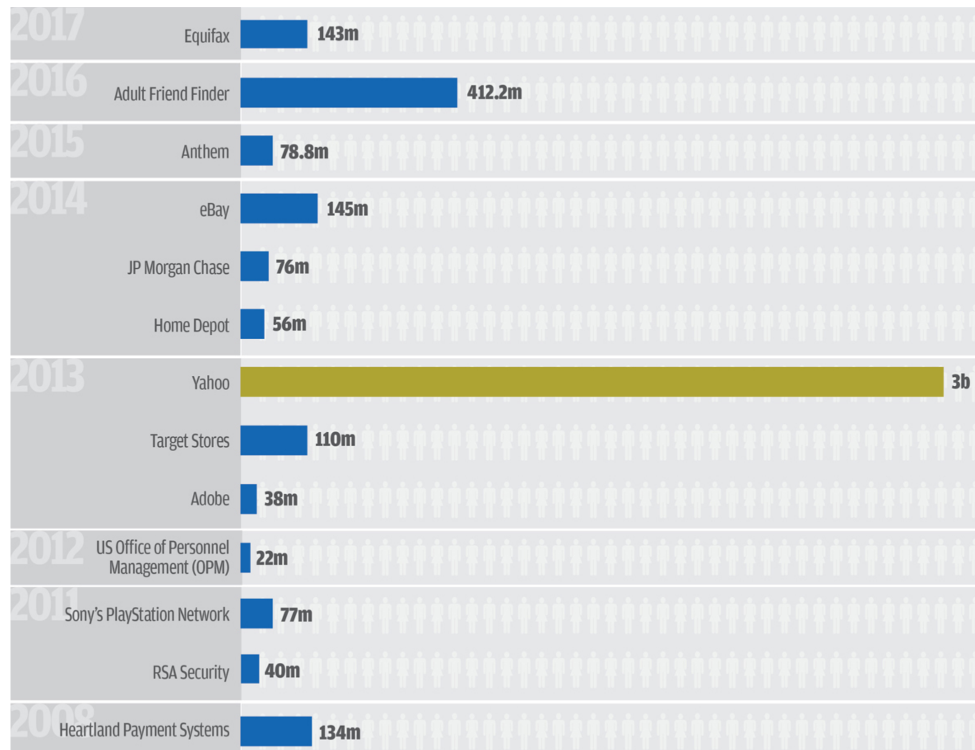
2. PASSWORD STORAGE

Password Storage

- Passwords typically stored in a file or database indexed by username
- **Strawman idea:** store passwords in plaintext
  - requires perfect authorization mechanisms
  - requires trusted system administrators
  - ...
- In the real world, password files get stolen

Threat Model: Offline Attack

- Adversary can read files from disk



- Adversary can read process memory

Note: users make this worse by reusing passwords across systems.

Password Storage

- **Want:** a function f such that...
  1. easy to compute and store f(p) for a password p
  2. hard given disclosed f(p) for attacker to recover p
  3. hard to trick system by finding password q s.t. q != p yet f(p) = f(q)
- Encryption would work, but then the key has to live somewhere
- Cryptographic hash functions suffice!
  - one-way property gives (1) and (2)
  - collision resistance gives (3)

Hashed passwords

- Each user has:
  - username uid
  - password p
- System stores: uid, H(p)

To authenticate Hu to remote server S using local machine L:

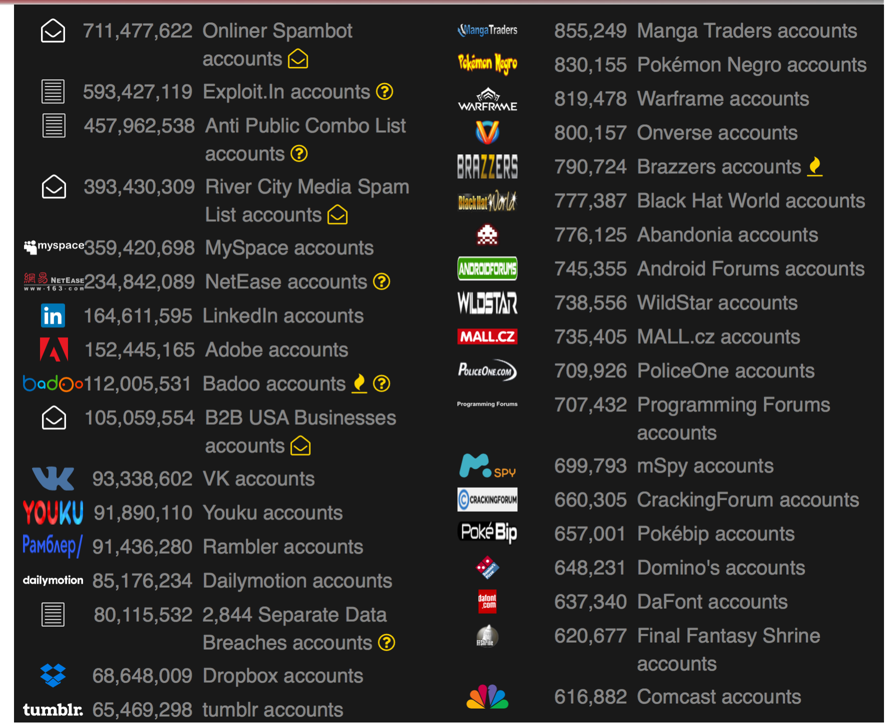
```
1. Hu->L: uid, p
2. L and S: establish secure channel
3. L->S: uid, p
4. S: let h = stored hashed password for uid;
      if h = H(p)
      then uid is authenticated
```

Hashed passwords are still vulnerable

Assume: attacker does learn password file *(offline guessing attack)*

- Hard to invert: i.e., given H(p) to compute p
- But what if attacker didn't care about inverting hash on arbitrary inputs?
  - i.e., only have to succeed on a small set of p's: p1, p2, ..., pn
- Then attacker could build a dictionary...

Dictionary attacks



Dictionary:

- p1, H(p1)
- p2, H(p2)
- ...
- pn, H(pn)

- Dictionary attack: lookup H(p) in dictionary to find p
- And it works because most passwords chosen by humans are from a relatively small set

Typical passwords

[Schneier quoting AccessData in 2007]:

- 7-9 character root plus a 1-3 character appendage
  - Root typically pronounceable, though not necessarily a real word
  - Appendage is a suffix (90%) or prefix (10%)
- Dictionary of 1000 roots plus 100 suffixes (= 100k passwords) cracks about 24% of all passwords
- More sophisticated dictionaries crack about 60% of passwords within 2-4 weeks
- Given biographical data (zip code, names, etc.) and other passwords of a user...
  - success rate goes up a little
  - time goes down to days or hours

Salted hashed passwords

- **Vulnerability:** one dictionary suffices to attack every user
- **Vulnerability:** passwords chosen from small space
- **Countermeasure:** include a unique system-chosen nonce as part of each user's password
  - make every user's stored hashed password different, even if they chose the same password
  - make passwords effectively be from larger space

Salted hashed passwords

- Each user has:
  - username uid
  - unique salt s
  - password p
- System stores: uid, s, H(s, p)

3. PASSWORD USAGE

Authenticating to a remote server

- Each user has:
  - username uid
  - unique salt s
  - password p
- System stores: uid, s, H(s, p)

```
1. Hu->L: uid, p
2. L and S: establish secure channel
3. L->S: uid, p
4. S: let h = stored hashed password for uid;
      let s = stored salt for uid;
      if h = H(s, p)
      then uid is authenticated
```

Threat Model: Online Attack

- Adversary can interact with the server as a user

When authentication fails

- **Guiding principle:** the system might be under attack, so don't make the attacker's job any easier
- Don't leak valid usernames:
  - Prompt for username and password in parallel
  - Don't reveal which was bad
- Record failed attempts and review
  - Perhaps in automated way by administrators
  - Perhaps manually by user at next successful login
- Lock account after too many attempts
- Rate limit login

Rate limiting

- **Vulnerability:** hashes are easy to compute
- **Countermeasure:** hash functions that are slow to compute
  - Slow hash wouldn't bother user: delay in logging hardly noticeable
  - But would bother attacker constructing dictionary: delay multiplied by number of entries
  - Ideally, enough to make constructing a large dictionary prohibitively expensive
- Examples: bcrypt, scrypt, Argon2,...

Slowing down fast hashes

- Given a fast hash function...
- Slow it down by iterating it many times:

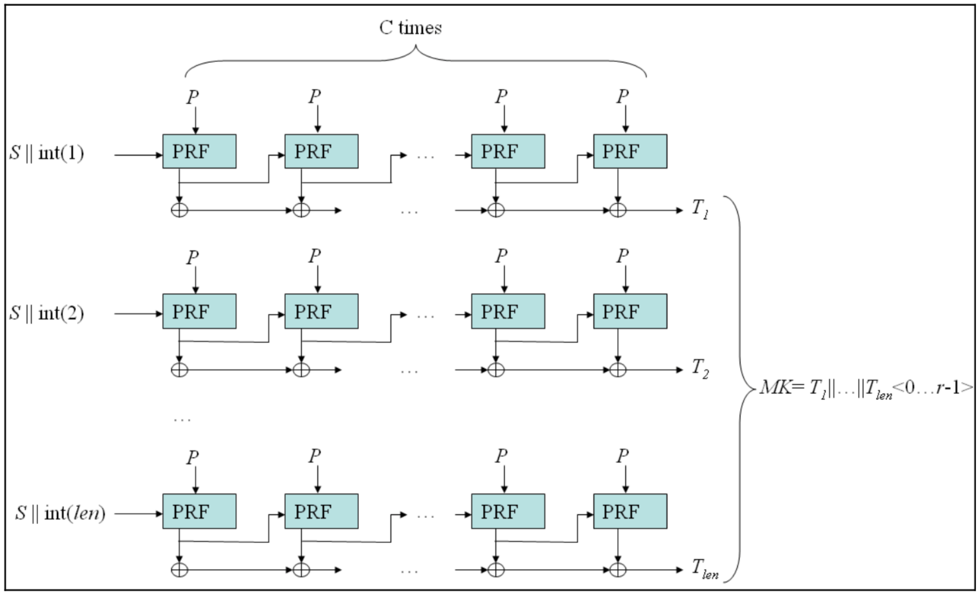
```
z1 = H(p);
z2 = H(p, z1);
...
z1000 = H(p, z999);
output z1 XOR z2 XOR ... XOR z1000
```

- Number of iterations is a parameter to control slowdown
  - originally thousands
  - current thinking is 10s of thousands
- Aka key stretching

Password-Based Encryption

- PBKDF2: Password-based key derivation function [RFC 8018]
- **Output:** derived key k
- **Input:**
  - Password p
  - Salt s
  - Iteration count c
  - Key length len
  - Pseudorandom function (PRF): "looks random" to an adversary that doesn't know an input called the *seed* (commony instantiated with an HMAC)

PBKDF2



Algorithm:

- k = T(1) || T(2) || ... || T(n)
  - enough T's to achieve desired len
  - || denotes bit concatenation
- T(i) = F(p, s, c, i)
  - F is in essence a salted iterated hash...
- F(p, s, c, i) = U(1) XOR ... XOR U(c)
  - U(1) = PRF(s, i; p)
  - U(j) = PRF(U(j-1); p)

4. PASSWORD CHANGE

Password change

Motivated by...

- **User** forgets password (maybe just *recover* password)
- **System** forces password expiration
  - Naively seems wise
  - Research suggests otherwise [see Cranor 2016]:
    - When users do change passwords, they change them predictably
    - Foreknowledge of expiration causes users to choose weaker passwords
- **Attacker** learns password:
  - Social engineering: deceitful techniques to manipulate a person into disclosing information
  - Online guessing: attacker uses authentication interface to guess passwords
  - Offline guessing: attacker acquires password database for system and attempts to crack it

Change mechanisms

- Tend to be **more vulnerable** than the rest of the authentication system
  - Not designed or tested as well
  - Have to solve the authentication problem without the benefit of a password
- Two common mechanisms:
  - Security questions
  - Emailed passwords

Security questions

- Something you know: attributes of identity established at enrollment
- **Pro:** you are unlikely to forget answers
- **Assumes:** attacker is unlikely to be able to answer questions
- **Con:** might not resist targeted attacks
- **Con:** linking is a problem; same answers re-used in many systems

Emailed password

- Might be your old password or a new temporary password
  - one-time password: valid for single use only, maybe limited duration

- **Assumes:** attacker is unlikely to have compromised your email account
- **Assumes:** email service correctly authenticates you

Password lifecycle

1. **Create:** user chooses password
2. **Store:** system stores password with user identifier
3. **Use:** user supplies password to authenticate
4. **Change/recover/reset:** user wants or needs to change password

Beyond passwords?

- Passwords are tolerated or hated by users
- Passwords are plagued by security problems
- **Can we do better?**
- Criteria: [Bonneau et al. 2012]
  - Security
  - Usability
  - Deployability

...criteria are worth studying for security in general

Schemes to replace passwords

- Password managers
- Proxies
- Federated identity management
- Graphical
- Cognitive
- Paper tokens
- Visual cryptography
- Hardware tokens
- Phone-based
- Biometric

Schemes to replace passwords

[Bonneau et al. 2012]:

- Most schemes do better than passwords on security
- Some schemes do better and some worse on usability
- Every scheme does worse than passwords on deployability
- **Passwords are here to stay, for now**
- Schemes offering some variation of **single sign on** seem to offer best improvements in security and usability...