Lecture 14: Passwords

CS 181S

Spring 2024

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**

Exercise 1: Choosing Passwords

Guess the top five most common US passwords in 2023

Who creates?

- **User**
- **System**
- **Administrator**

Strong passwords

- How to characterize strength?
- **One Approach:** 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)

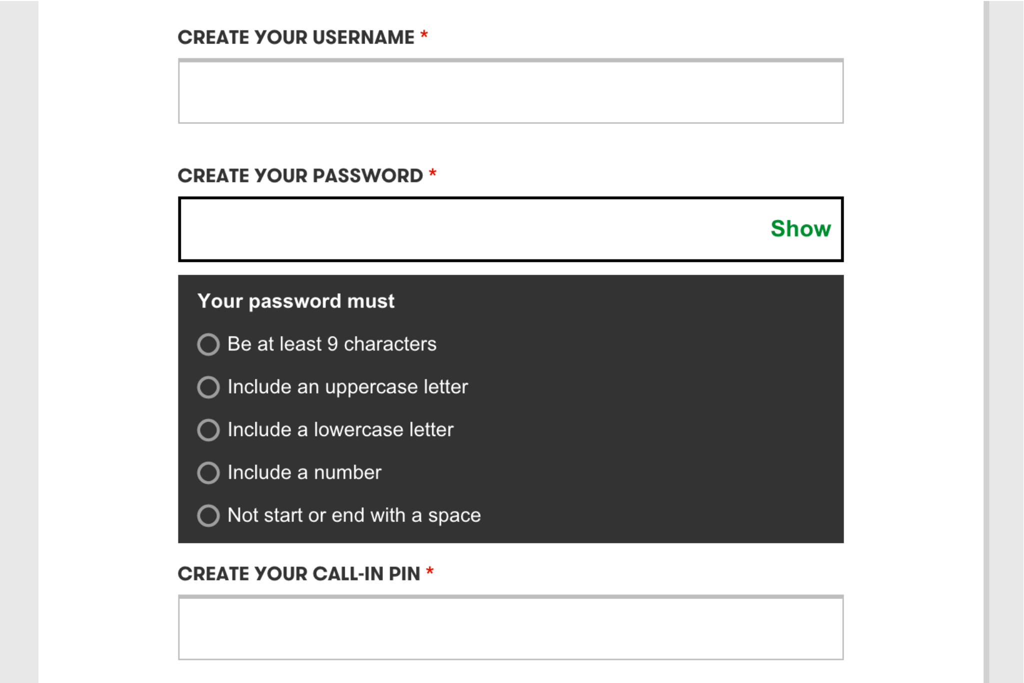
Exercise 2: Entropy of passwords

- Option A: 8 character passwords chosen uniformly at random from 26 character alphabet

- Option B: 1 word chosen at random from entire vocabulary

Password Recipes

- **Problem:** guide users into choosing strong passwords
- **Solution:** **password recipes** are rules for composing passwords
  - e.g., must have at least one number and one punctuation symbol and one upper case letter



Entropy estimation

- 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 estimation

But:

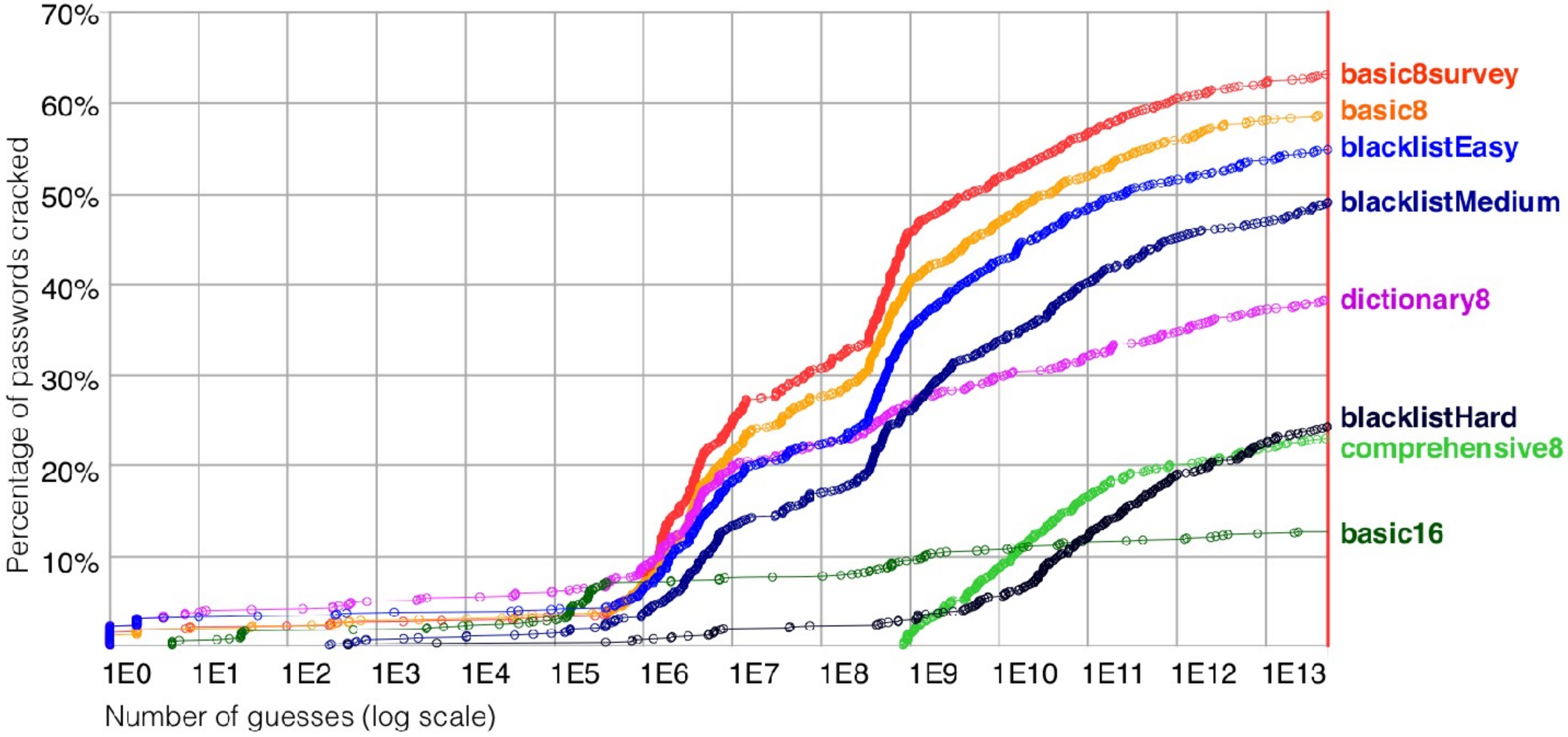
- *"[NIST's] notion of password entropy...does not provide a valid metric for measuring the security provided by password creation policies."*
- Underlying problem: Shannon entropy not a good predictor of how quickly attackers can crack passwords

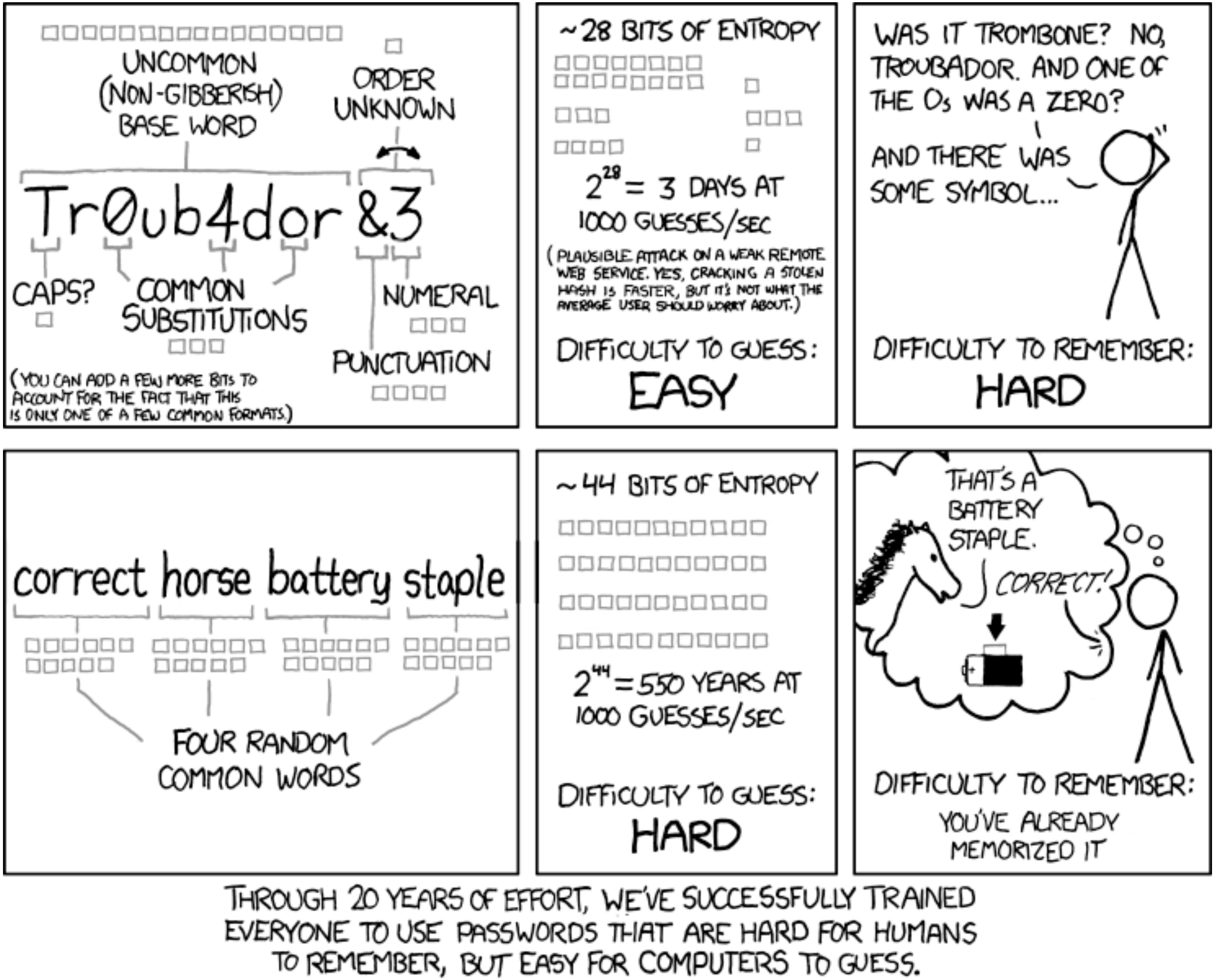
Password Cracking

- Evaluate recipes based on
  - percentage of passwords cracked
  - number of guesses required to crack

- Example recipes:
  1. ≥ 8 characters
  2. ≥ 8 characters, no blacklisted words ...with various blacklists
  3. ≥ 8 characters, no blacklisted words, one uppercase, lowercase, symbol, and digit ("comprehensive", c8)
  4. ≥ 16 characters ("passphrase", b16)
- Results...

Recipe comparison




UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
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, updated 2020) recommends:

- minimum of 8 characters
- up to 64 characters should be accepted
- all printable ASCII characters and Unicode should be accepted
- blacklist compromised values, dictionary words, repetitive characters, and context-specific words
- no other security requirements

Should provide guidance on picking a good password (e.g., password meter

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
  - ...

Threat Model: Offline Attack

- Adversary can read files from disk

BRANDS WITH 100M+ LEAKED RECORDS

BRAND NAME	RECORDS LEAKED
Tencent	1.5B
Weibo	504M
MySpace	360M
Twitter	281M
Wattpad	271M
NetEase	261M
Deezer	258M
LinkedIn	251M
AdultFriendFinder	220M
Zynga	217M
Luxottica	206M
Evite	179M
Zing	164M
Adobe	153M
MyFitnessPal	151M
Canva	143M
JD.com	142M
Badoo	127M
VK	101M
Youku	100M

cybernews

- 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 work!
  - 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)

Exercise 3: Hashed Passwords

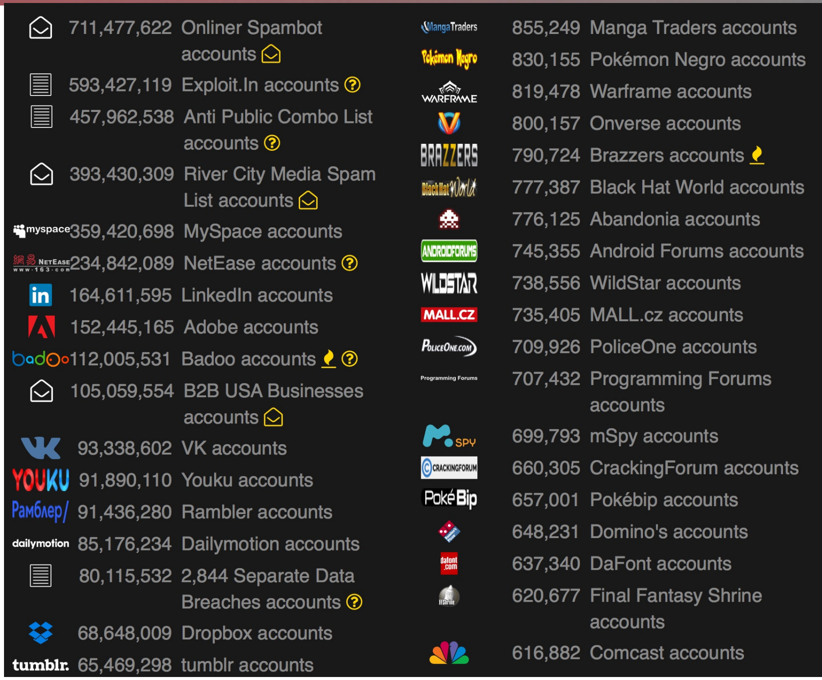
- Consider an alternative authentication protocol where user sends uid, H(p) and the service compares H(p) to the stored hash. Would this be more or less secure than sending the plaintext password? Why?

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 value as part of each user's password

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

Salt and pepper

- Each user has:
  - username uid
  - unique salt s1
  - unique pepper s2
  - password p
- System stores: uid, s1, H(s1, s2, p)

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)

4. PASSWORD CHANGE

Password change

Motivated by...

- **User** forgets password (maybe just *recover* password)
- **System** forces password expiration
  - Naively seems wise
  - Research suggests otherwise
- **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 reset

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 reset

- Might be your old password, a new temporary password, or just a reset link
  - **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:
  - Security
  - Usability
  - Deployability

Schemes to replace passwords

- Graphical
- Cognitive
- Visual cryptography

Schemes to improve passwords

- Password managers
- Single Sign-On
- Two-factor authentication

Exercise 4: Authentication Examples

- Choose an example website (e.g., email provider, social network, or a payments app) and investigate how how they handle authentication.

- What are their restrictions on password selection? Do they support SSO? How do they handle recovery? Do they rely exclusively on passwords?

Something you know