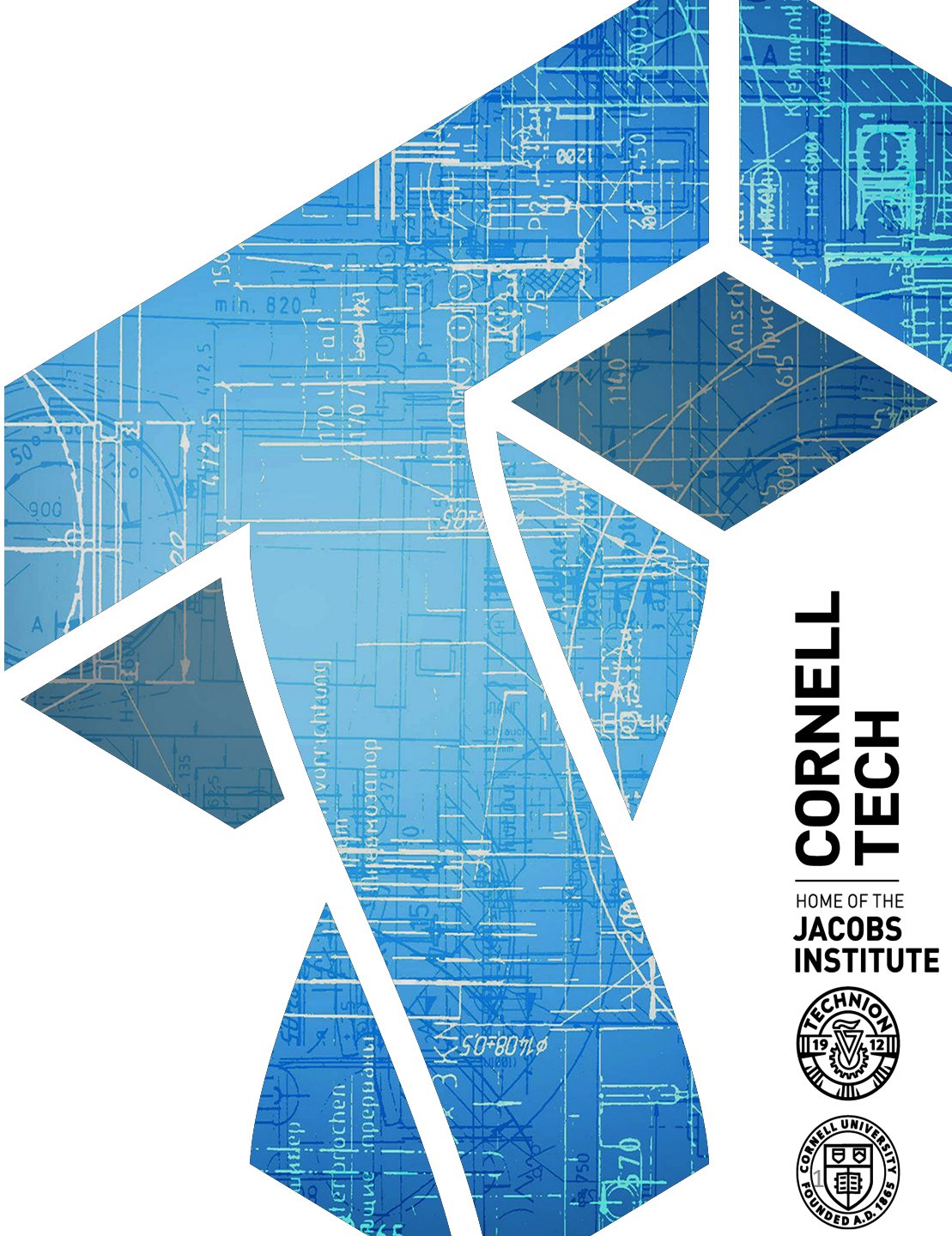
CS 6431: Passwords



Instructor: Tom Ristenpart

https://github.com/tomrist/cs6431-fall2021

CORNELL TECH
HOME OF THE JACOBS INSTITUTE

The game plan

- Basics, history
  - Morris and Thompson 1979
  - The research landscape
- Password hashing, hardening
- Understanding password strength
  - Morris & Thompson (server-side measurement)
  - Florencio & Herley (client-side measurement)
  - Bonneau (server-side measurement)
  - Evaluating password strength metrics

Password use cases

- OS login
- Website / service login
- PINs
- Encryption

Authentication

Confidentiality

Authentication

Authentication service

Registration

Register:
tom, pw

Store tom, pw in some form.

Login

Login:
tom, pw’

Check that pw’ = pw

Recover

Recover:
tom

Somehow confirm that it is tom, reset password

Threat models for passwords

Login:
tom, pw’

Check that pw’ = pw

- Threat model is description of adversary, their goal(s), their capabilities, and their target

Brute-force attacks

- **Offline brute-force attacks**
  - Compromise database
  - E.g.: “cracking” via dictionary attacks
  - *Countermeasures*: hash passwords with purposefully slow-to-compute cryptographic hash function

    (was: UNIX hash, MD5, SHA-1 now: argon2, scrypt)
- **Online brute-force attacks**
  - E.g: Submit guesses to web site
  - *Countermeasures*: Rate limit, account lockout

- **Shoulder surfing, compelled password disclosure, malware, side-channels, ...**

Brute-force attacks

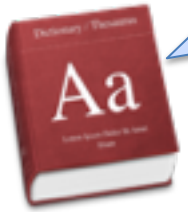
- **Offline brute-force attacks**
  - Compromise database
  - E.g.: “cracking” via dictionary attacks
  - *Countermeasures*: hash passwords with purposefully slow-to-compute cryptographic hash function

    (was: UNIX hash, MD5, SHA-1 now: argon2, scrypt)
- **Online brute-force attacks**
  - E.g: Submit guesses to web site
  - *Countermeasures*: Rate limit, account lockout

- **Shoulder surfing, compelled password disclosure, malware, side-channels, ...**

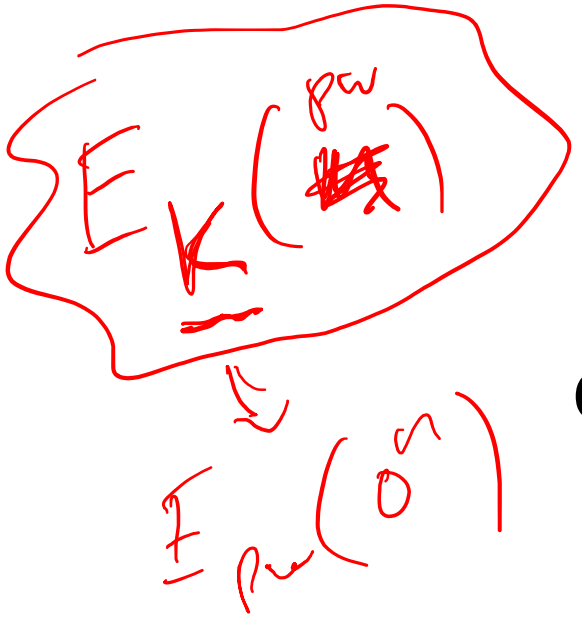
Offline brute-force attacks

$h_1 = H(pw_1)$
$h_2 = H(pw_2)$
…
$h_m = H(pw_m)$



Nowadays:
Use password leaks to inform dictionary

Dictionary:
List of probable passwords
or way of generating them



$H(guess_1)$, $H(guess_2)$, …

Check if any guesses equal any of h_1, … , h_m

“Human beings being what they are, there is a strong tendency for people to choose relatively short and simple passwords that they can remember. “

[Morris, Thompson 1979]

“The results were disappointing, except to the bad guy.”

“In a collection of 3,289 passwords gathered from many users over a long period of time,

15 were a single ASCII character;
72 were strings of two ASCII characters;
464 were strings of three ASCII characters;
477 were strings of four alphamerics;
706 were five letters, all upper-case or all lower-case;
605 were six letters, all lower-case.”

[Morris, Thompson 1979]

86% crackable

Improvements suggested

Morris, Thompson suggest:

- Slow hashing (they called it encryption)
- Less predictable passwords (pw requirements)
- Salt before hashing
- Use custom version of DES to avoid hardware
- Avoid timing attacks to distinguish bad login

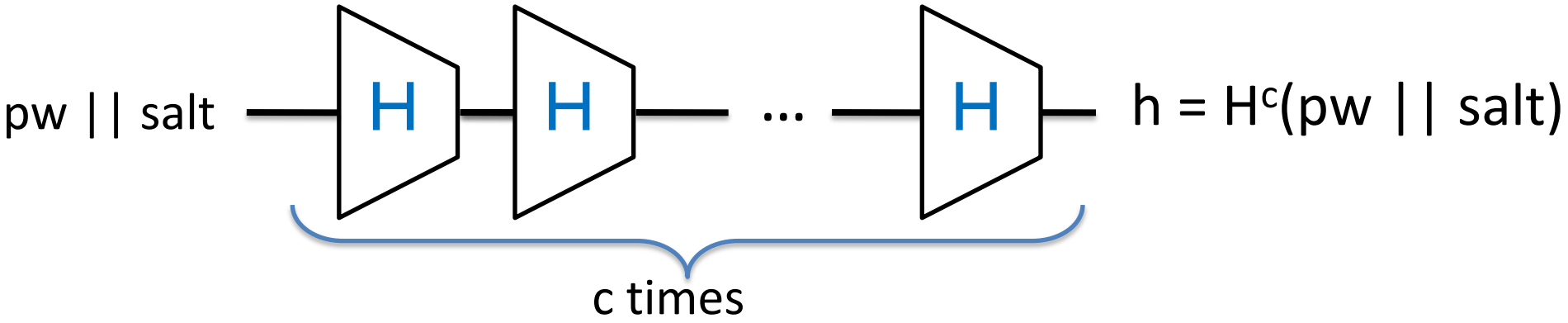
> "We did not attempt to hide the security aspects of the operating system, thereby playing the customary make-believe game in which weaknesses of the system are not discussed no matter how apparent."

The research landscape since 1979…

- **Understanding user password selection**
  - Measuring password strength [see citations in Bonneau paper], [Li, Han `14], [CMU papers]
  - Measuring password reuse [Das et al. `14] [Pearmen et al. `17]
- **Usability**
  - Strength meters, requirements, etc. [Komanduri et al. ‘11] [Dell’Amico, Filippone ‘15] [Wheeler ‘16] [Melicher et al. ‘16]
  - Password expiration [Zhang et al. ‘12]
  - Typo-tolerance [Chatterjee et al. `16]
- **Password transmission, login logic**
  - Single sign-on (SSO) technologies
  - Password-based authenticated key exchange [Bellovin, Merritt ‘92]
- **Password hashing**
  - New algorithms [PKCS standards], [Percival ’09], [Biryukov, Khovratovich ‘15]
  - Proofs [Wagner, Goldberg ‘00] [Bellare, Ristenpart, Tessaro ‘12] [Alwen et al. 2016]
- **Improving offline brute-force attacks**
  - Time-space trade-offs (rainbow tables) [Hellman ’80], [Oeschlin ‘03], [Narayanan, Shmatikov ‘05]
  - Better dictionaries [JohntheRipper], [Weir et al. ‘09], [Ma et al. ‘14]
- **Password managers**
  - Decoy-based [Bojinov et al. ’10], [Chatterjee et al. ‘15]
  - Breaking password managers [Li et al. ‘14] [Silver et al. ’15]
  - Stateless password managers [Ross et al. ’05]

Password hashing

- Recall Morris, Thompson goal: slow down brute-force attacks
- PKCS#5 approach:



H : $\{0,1\}^*$ -> $\{0,1\}^n$ is cryptographic hash function (e.g., SHA-256)

salt should be random bit string large enough to be unpredictable

Password hashing

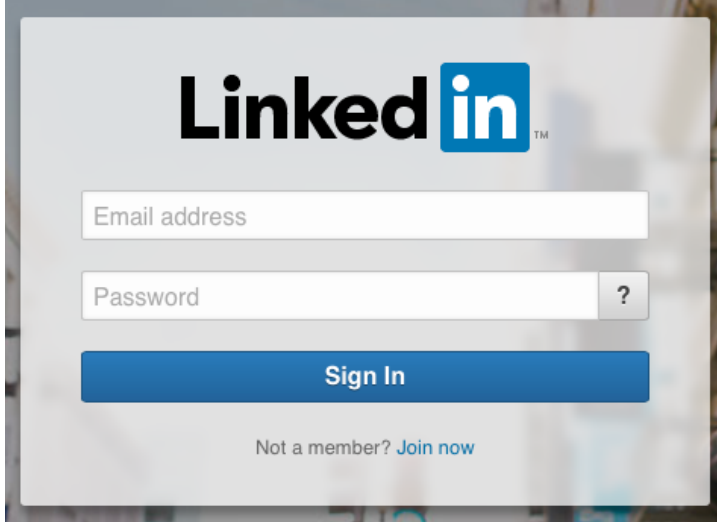
- Recall Morris, Thompson goal: slow down brute-force attacks
- The role of salts:
  - Prevents use of time-memory trade-offs (rainbow tables)
  - Cracking m accounts requires m times the work

$$h_1 = H^c(pw_1, salt_1)$$
$$h_2 = H^c(pw_2, salt_2)$$
$$\ldots$$
$$h_m = H^c(pw_m, salt_m)$$

Proofs:
See [Bellare, Ristenpart, Tessaro '12]

Linked in circa 2012 stored passwords as which of:

- pw
- MD5(pw)
- H(salt,pw)
- H^c(salt,pw)



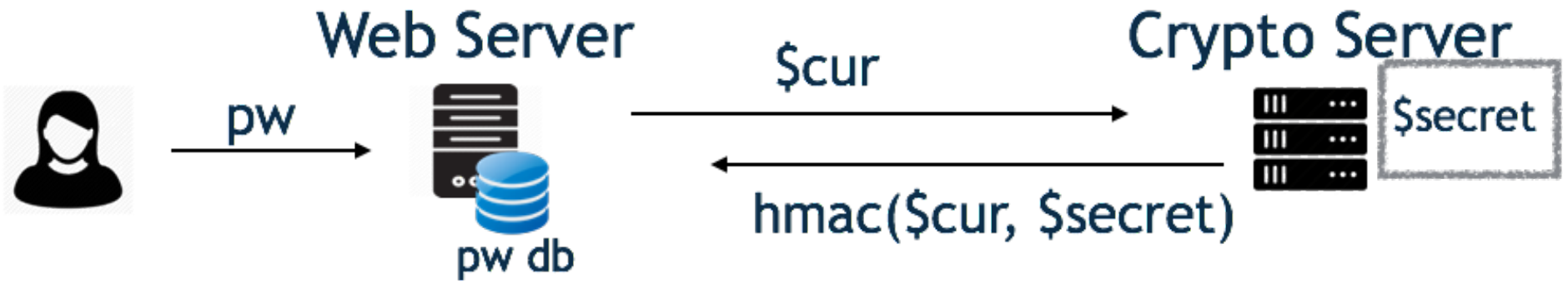
2012: 6.5 million hashes leaked onto Internet
90% cracked in 2 weeks

2016: 177.5 million more hashes leaked
98% cracked in 1 week

http://arstechnica.com/security/2016/06/how-linkedins-password-sloppiness-hurts-us-all/

Facebook password “onion”

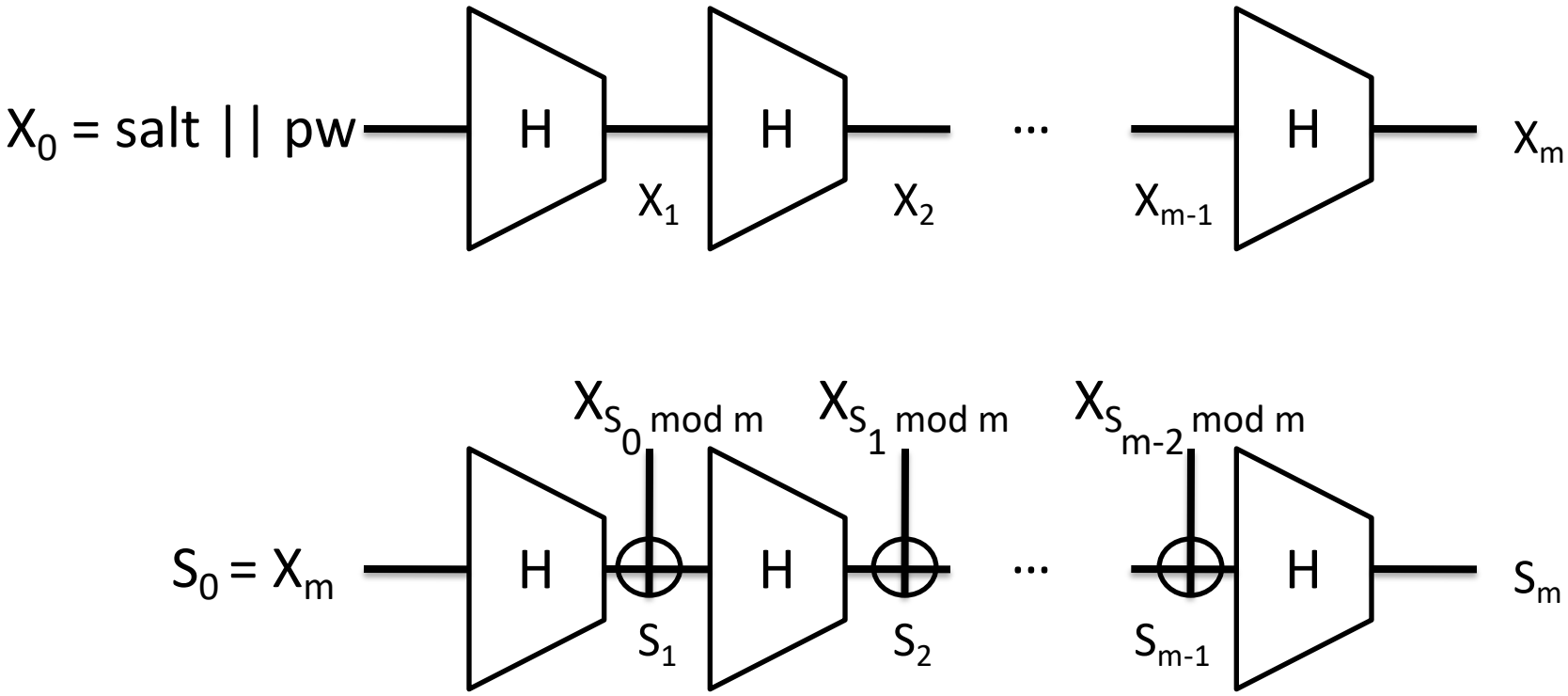
```
$cur  = ‘password’
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```



Split-trust model:
must compromise both servers to mount offline brute-force

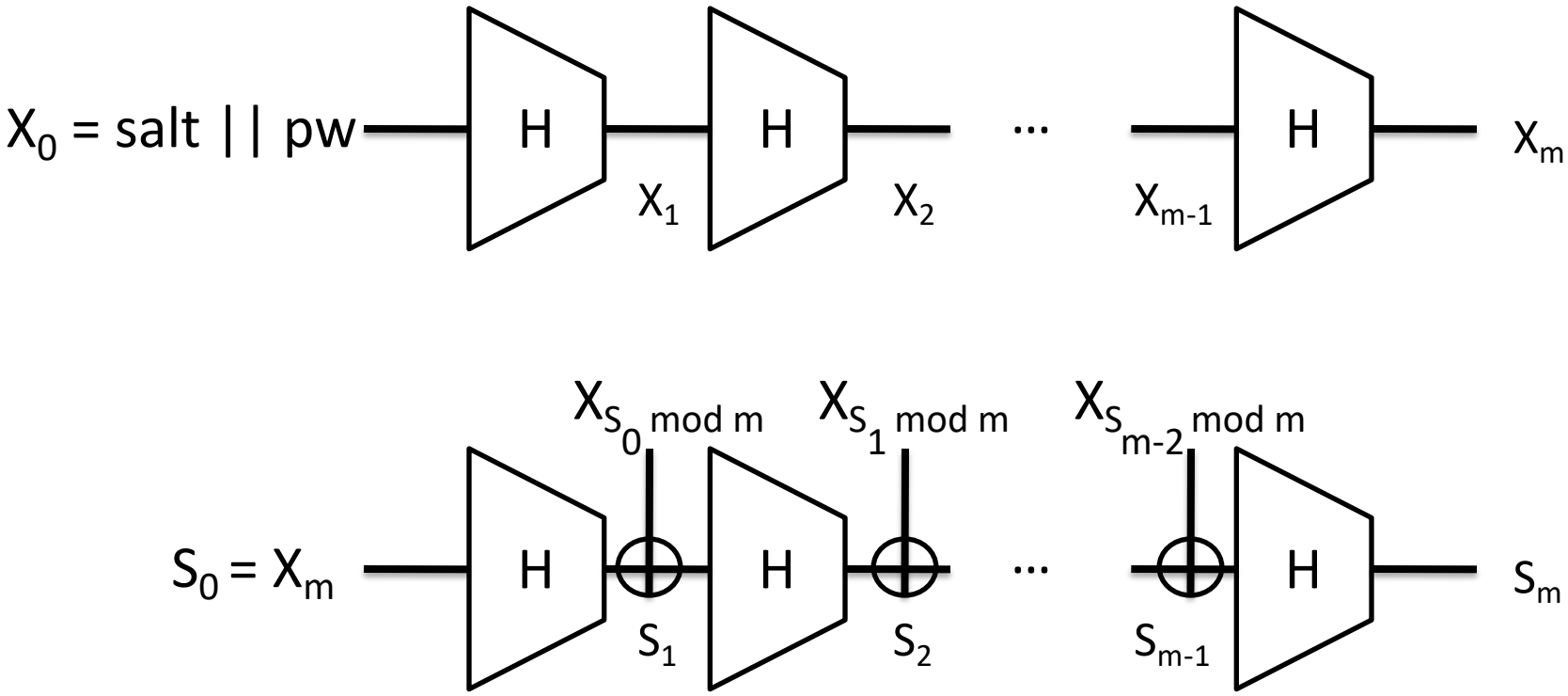
Scrypt and memory-hard hashing

- What did Morris, Thompson suggest for slowing down DES chips?
- Increase time & memory needed to compute hash.
  - This makes ASIC implementations harder
- Scrypt



Scrypt and memory-hard hashing

- How much time-space required to compute?
  - Obvious algorithm: m space and 2m time
  - Time-space complexity = Time * Space $\in O(m^2)$
  - Recent proof that no shortcuts exist [Alwen et al. 2016]



Understanding password strength

Methods for measuring password strength?

Understanding password strength

(1) Empirical studies of user passwords

Password database leaks
Instrumentation of large web systems (Bonneau paper)
Instrumentation of clients (Florencio & Herley)
In-lab studies, online studies, surveys

(2) Develop probabilistic model of passwords

$pw_1, pw_2, \ldots, pw_N$
$p(pw_i) = p_i$ = probability user selects password pw_i

$$\sum_i p_i = 1$$

(3) Use p to educate brute-force crackers, strength meters

Internet users ditch “password” as password, upgrade to “123456”

Contest for most commonly used terrible password has a new champion.

by **Jon Brodkin** - Jan 20 2014, 4:00pm GMT

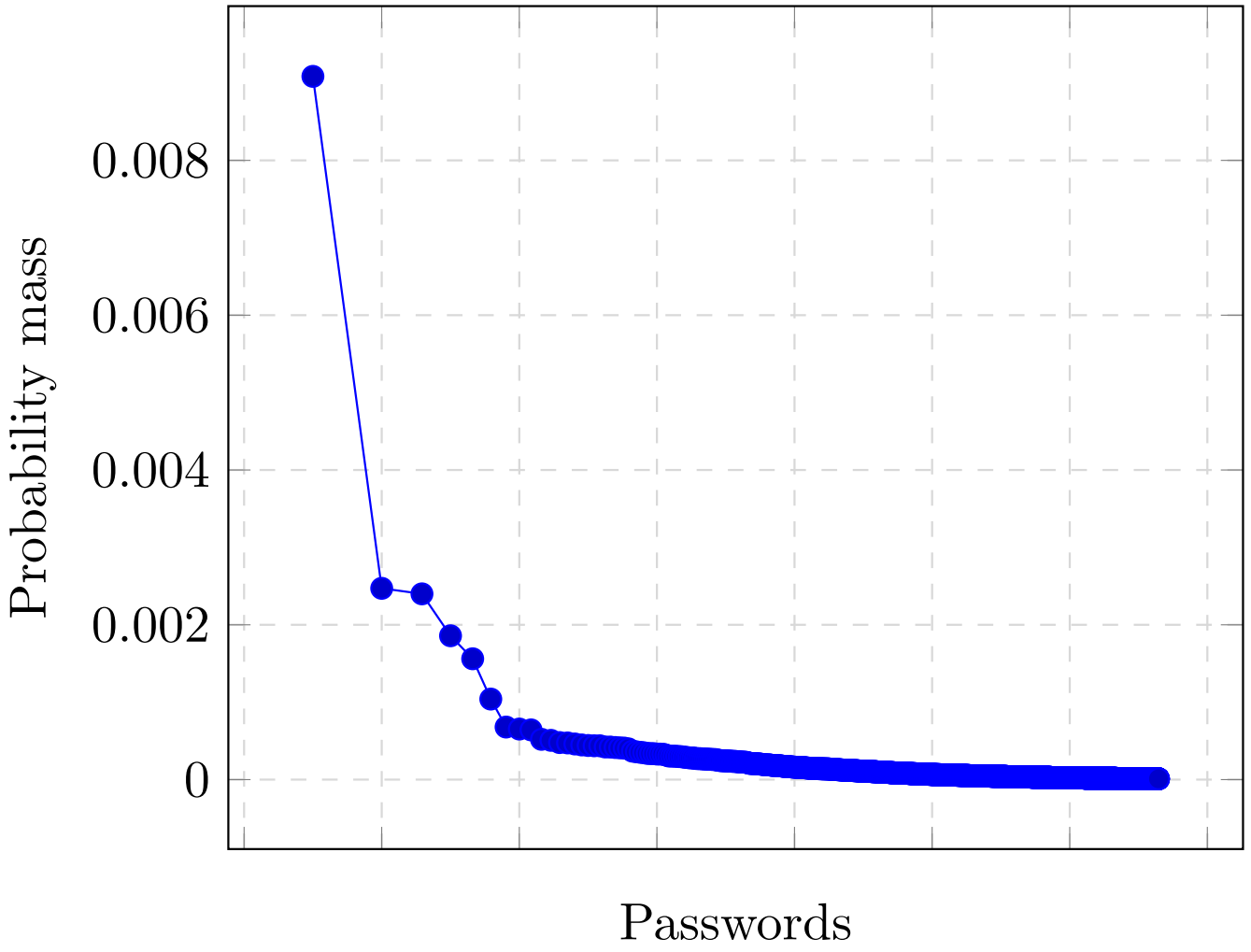
290729 123456
79076 12345
76789 123456789
59462 password
49952 iloveyou
33291 princess
21725 1234567
20901 rockyou
20553 12345678
16648 abc123
16227 nicole
15308 daniel
15163 babygirl
14726 monkey
14331 lovely

Rockyou data breach:
32 million social gaming accounts

Most common password used by almost 1%

[Bonneau 2012]
69 million Yahoo! Passwords
1.1% of users pick same password

Rockyou empirical probability mass function



(Only first 5,000 points shown)

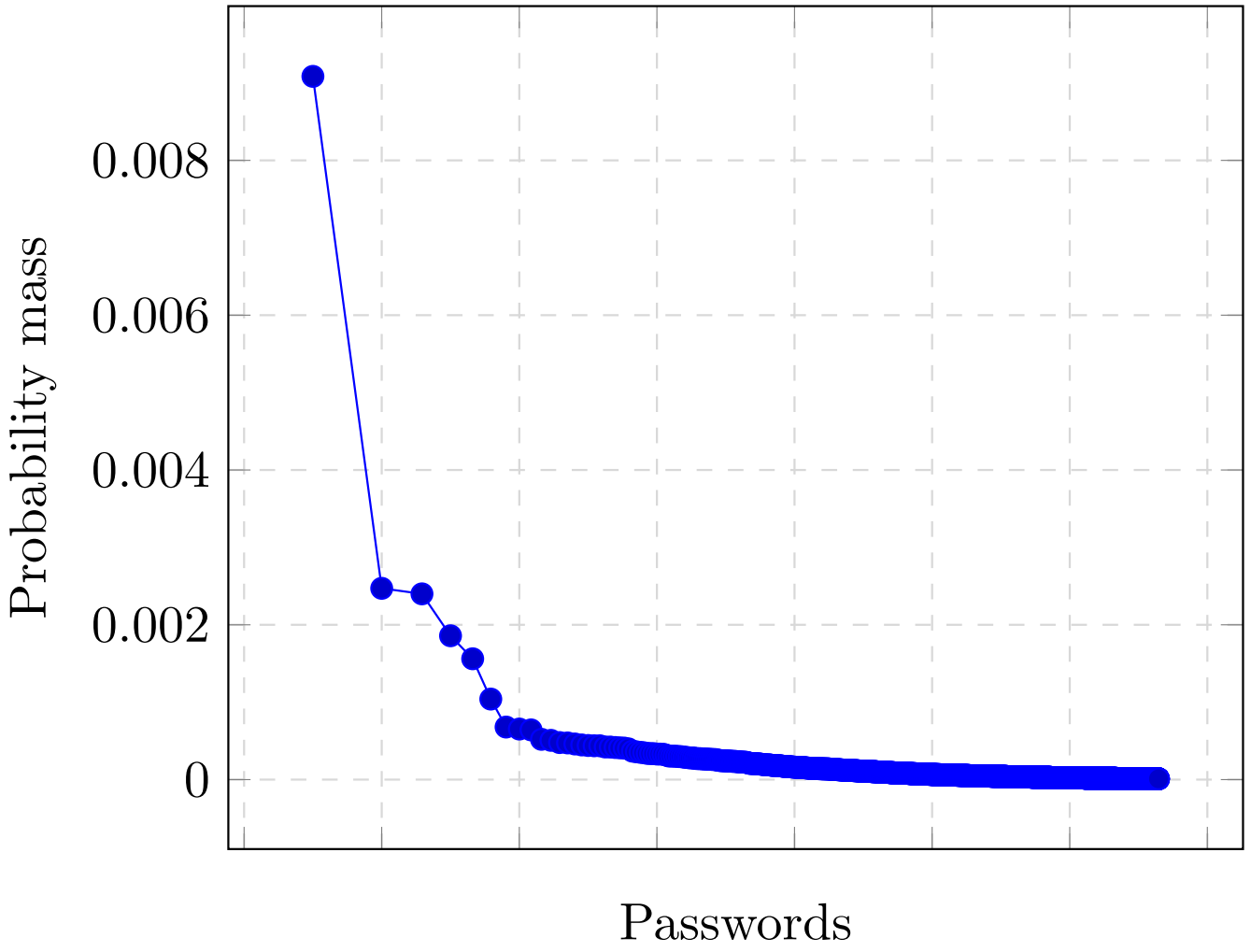
Florencio & Herley 2007

- Instrument Windows Live toolbar
  - 544,960 clients opted-in to study
- Captured passwords typed into browser
  - Hashed and stored locally
  - Sent report to server about (quantized) password strength, associated URL, etc.

Florencio & Herley 2007

- Avg user:
  - Has 6.5 passwords, each used at 3.9 different sites
  - Has 25 accounts requiring passwords
  - Types 8 passwords per day
  - Selects 40.54 “bitstring” password
- ~1.5% of Yahoo users forget their passwords each month (!)

Rockyou empirical probability mass function



(Only first 5,000 points shown)

Password strength metrics

- Florencio and Herley approach?
  - Alphasize(pw) = sum of the sizes of character classes observed in password
    - Hello12! Has alphabet size = 26 + 26 + 10 + 22 = 84
  - Bitstrength(pw) = $\text{Alphasize(pw)}^{\text{len(pw)}}$
- Simpler than classical NIST entropy estimate

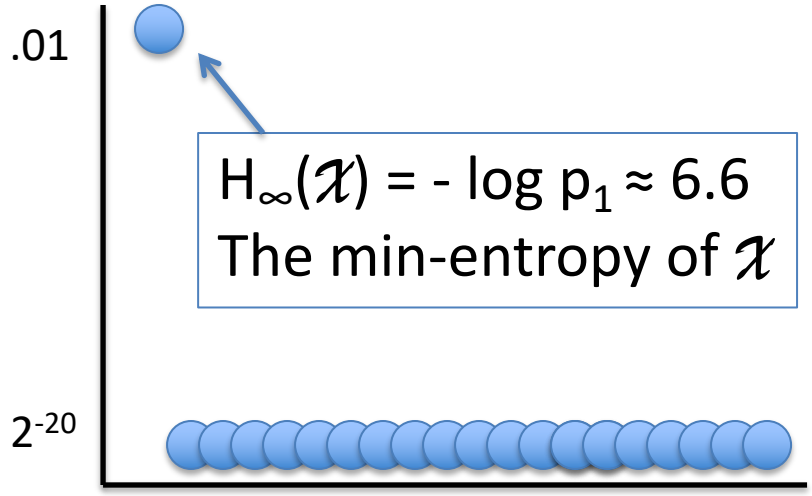
Password strength metrics

Let $\mathcal{X}$ be password distribution.
Passwords are drawn iid from $\mathcal{X}$
N is size of support of $\mathcal{X}$
p_1 , p_2 , …, p_N are probabilities of passwords in decreasing order

Shannon entropy: $$H_1(\mathcal{X}) = \sum_{i=1}^{N} -p_i \log p_i$$

Shannon entropy is poor measure (for password unpredictability)

N = 1,000,000
$p_1 = 1 / 100$
$p_2 = (1 - 1/100)/999{,}999 \approx 1 / 2^{20}$
…
$p_N = (1 - 1/100)/999{,}999 \approx 1 / 2^{20}$



$H_1(\mathcal{X}) \approx 19$

19 bits of "unpredictability". Probability of success about $1/2^{19}$

What is probability of success if attacker makes one guess?

Shannon entropy is almost never useful measure for security

Summary

- Passwords key security technology
- Many deficiencies already discussed back in 1970s
- Measurement techniques challenging
- Early metrics for password strength crude / misleading

- Next time: Bonneau & Melicher et al. papers. Segue to credential stuffing & other types of attacks & mitigations