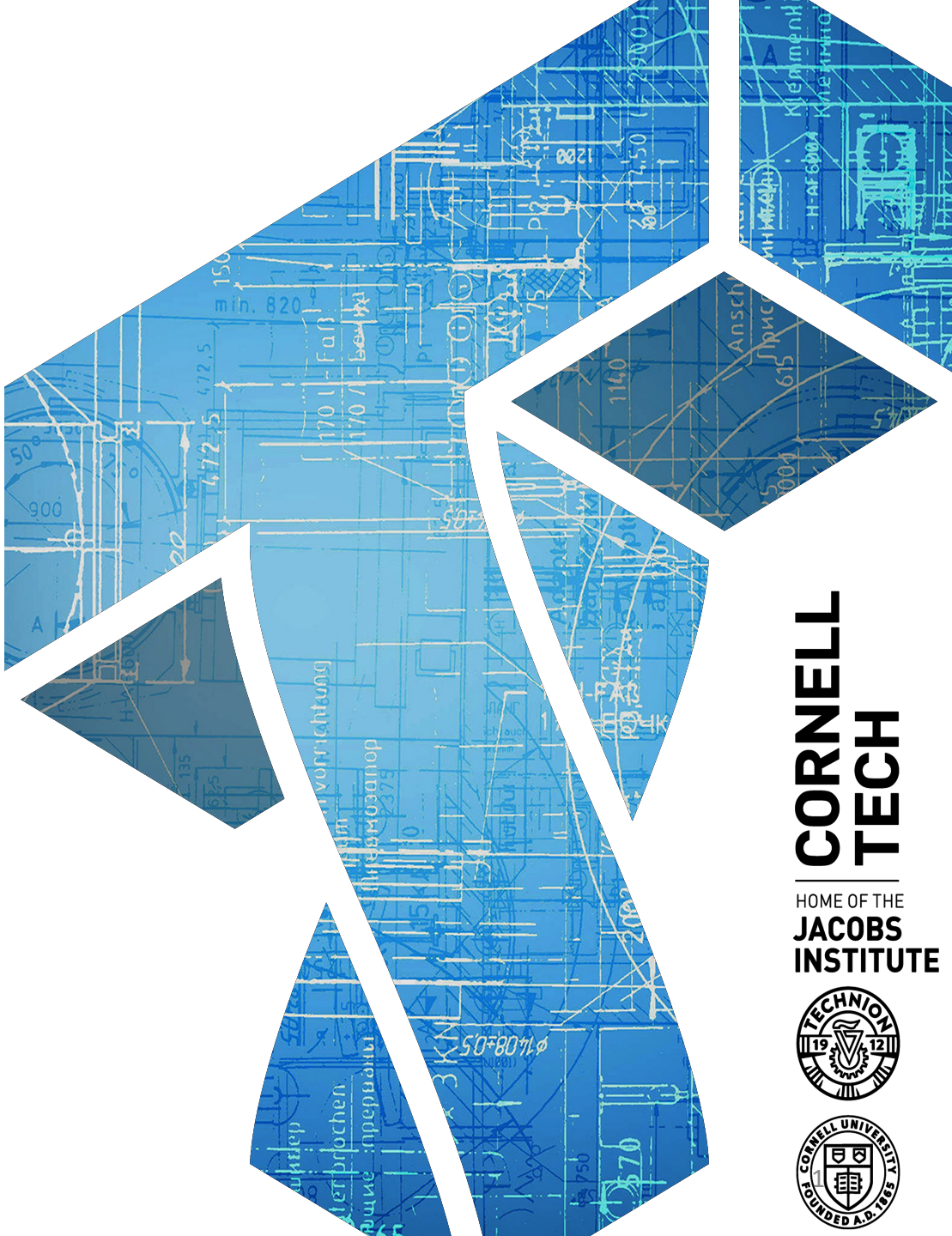
CS 5830
Cryptography



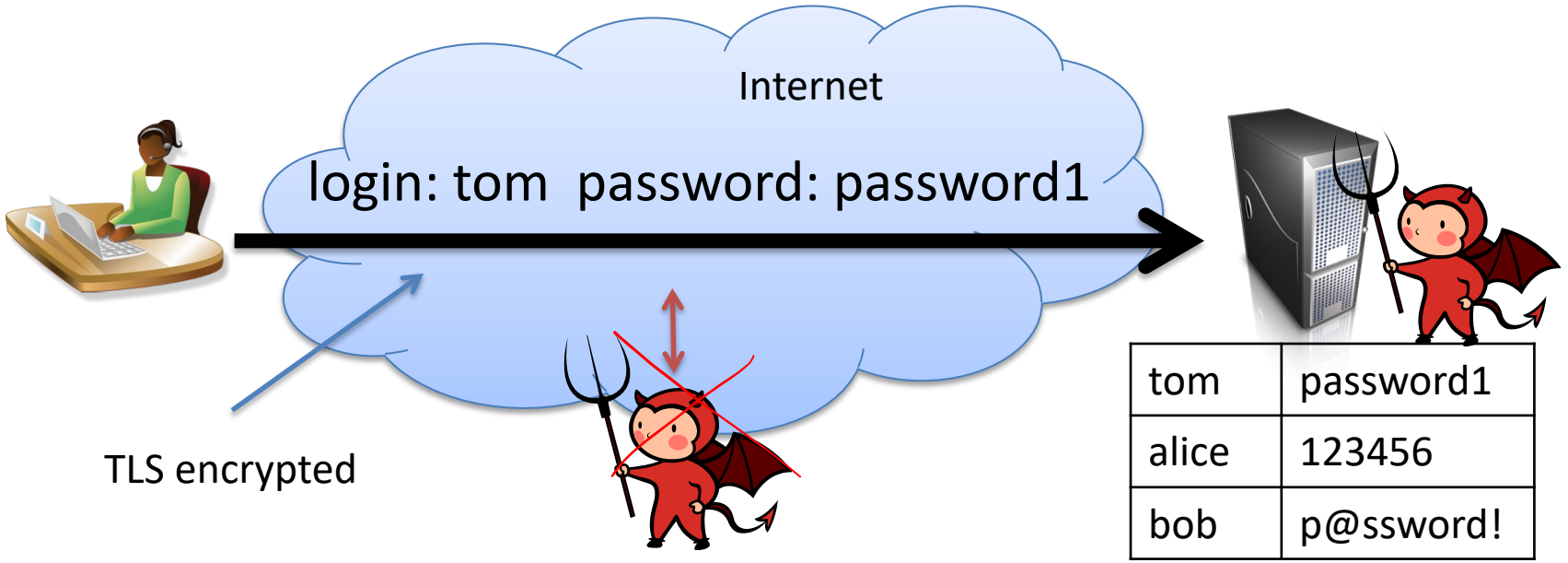
CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Where we're at

- We have seen authenticated encryption
  - We can encrypt data with shared, high-entropy secret so that no one can learn about plaintexts or modify messages
- Today's lecture:
  - Human-chosen/memorizable passwords
  - Password-based encryption
- Next time: random number generation

Passwords


Internet
login: tom password: password1
TLS encrypted
tom password1
alice 123456
bob p@ssword!

Breaches are ubiquitous

https://en.wikipedia.org/wiki/List_of_data_breaches

Entity	Year	Records
International Committee of the Red Cross	2022	515,000
50 companies and government institutions	2022	6,400,000
Ancestry.com	2021	300,000
Ankle & Foot Center of Tampa Bay, Inc.	2021	156,000
AOL	2021	20,000,000
Apple, Inc./BlueToad	2021	12,367,232
Apple	2021	275,000
Apple Health Medicaid	2021	91,000
Atraf	2021	unknown
CyberServe	2021	1,107,034
Health Service Executive	2021	unknown
Microsoft Exchange servers	2021	unknown
NEC Networks, LLC	2021	1,6000,000
T-Mobile	2021	45,000,000
Twitch	2021	unknown

Credential stuffing attacks

tom, password1

Login server

tom	$salt_1$, $H(salt_1$, password1)
alice	$salt_2$, $H(salt_2$, 123456)
bob	$salt_3$, $H(salt_3$, p@ssword!)

tom	password1
sally	hell0w0rlD4
nick	blahblahblah
…	…

- Simulations suggest works **~40%** of time (people reuse passwords!)
- Some evidence that ***majority*** of account compromises due to credential stuffing

December 18, 2017

4iQ Discovers 1.4 Billion Clear Text Credentials in a Single Database

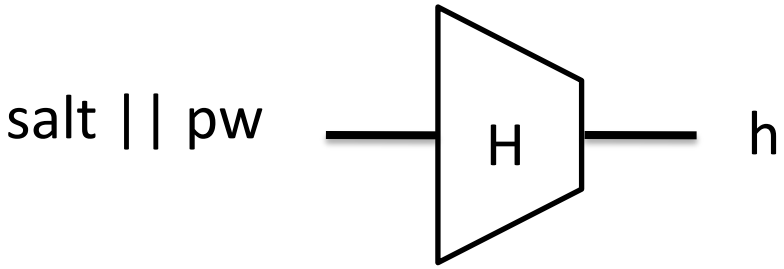
Preventing credential stuffing

- Store passwords in more secure form (password hashing)

- Build privacy-preserving protocols to alert users about breaches (password breach alerting)

Password hashing

Password hashing. Choose random salt and store (salt,h) where:



The idea: Attacker, given (salt,h), should not be able to recover pw

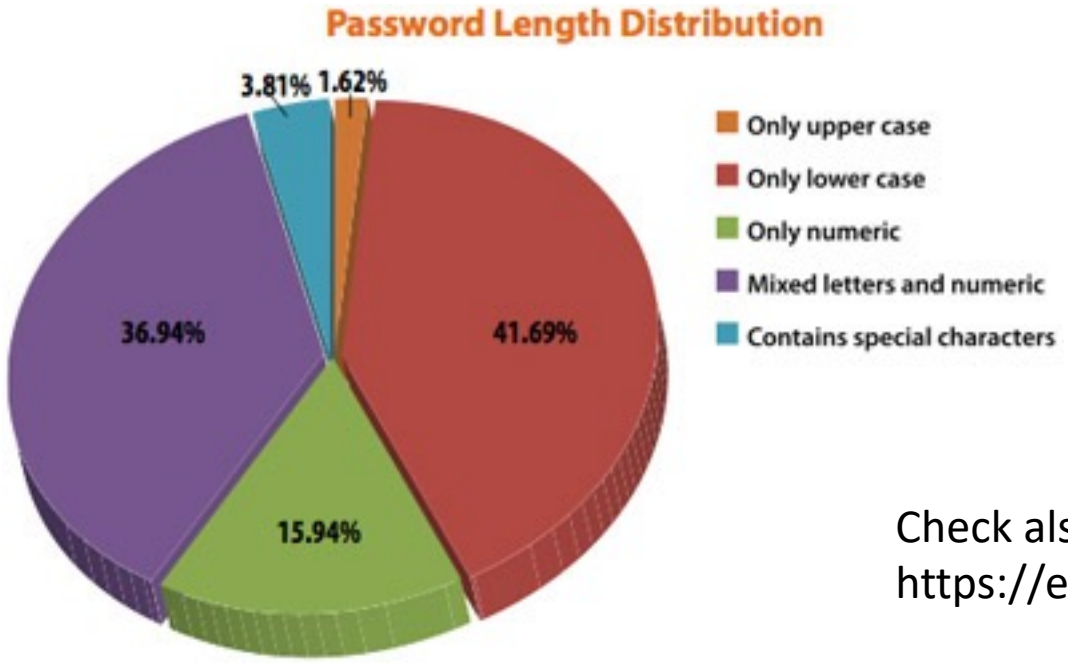
Or can they?

For each guess pw':
    If H(salt||pw') = h then
        Ret pw'

Rainbow tables speed this up in practice by way of precompution. Large salts make rainbow tables impractical

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Rank	Password	Number of Users with Password (absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856



From an Imperva study of released rockyou.com password database 2010

Check also:
https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

Brute-force password cracking

Given hash h = H(salt ||pw) and salt for unknown pw, find pw

JohnTheRipper: rule-based password guess generation. Can use dictionary of common (pass)words

Machine learning approach:

Count	password
290729	123456
79076	12345
76789	123456789
59462	password
49952	iloveyou
33291	princess
…	

Training alg. for language model

Model p

Generate sequence of guesses

$pw_1, pw_2, pw_3, \ldots$ s.t.

$p(pw_1) \geq p(pw_2) \geq p(pw_3) \ldots$

$h_1 = H(salt \,||\, pw_1)$

$h_2 = H(salt \,||\, pw_2)$

$h_3 = H(salt \,||\, pw_3)$

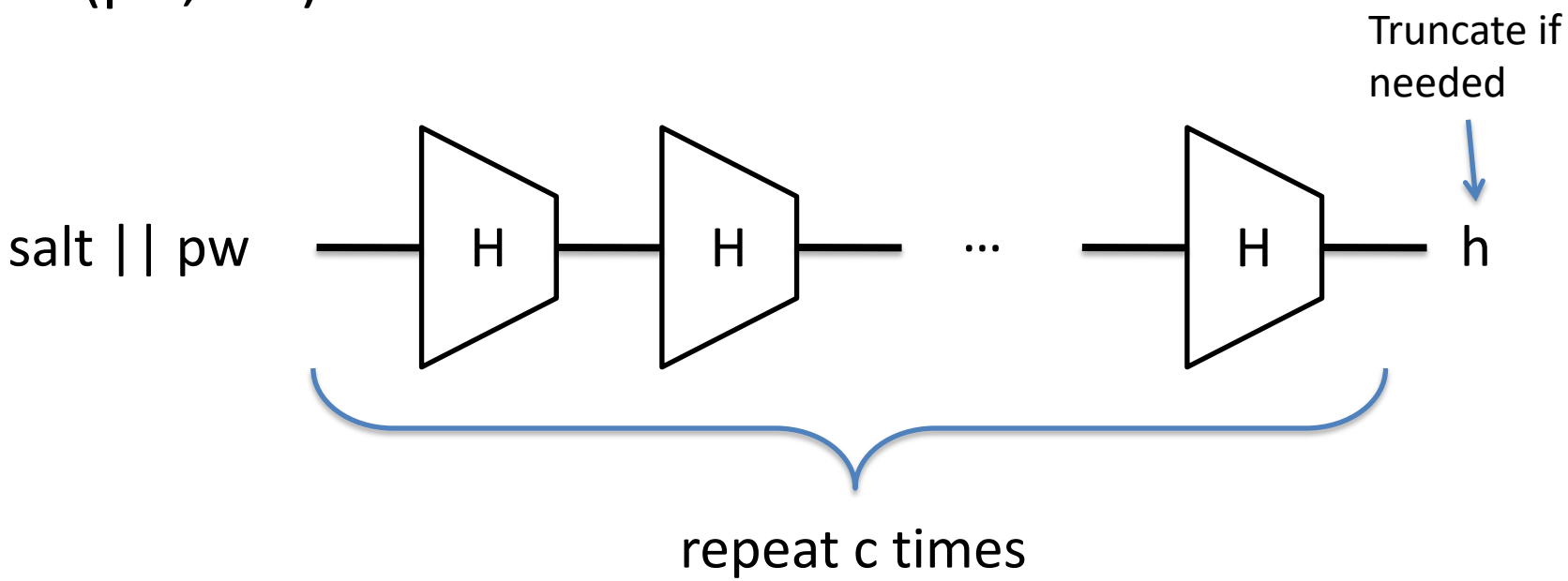
…

If $h = h_i$ then $pw = pw_i$

p is a probability model:
p(pw) is estimate of likelihood someone picks pw

Password-based Key Derivation (PBKDF)

PBKDF(pw,salt):



PKCS#5 standardizes PBKDF1 and PBKDF2, which are both hash-chain based. (PBKDF2 uses HMAC)

Slows down cracking attacks by a factor of c. Similar approach taken by bcrypt

```
06/16/21 7:52:58 ~/Dropbox/work/teaching/cs5830-summer2021/repo/slides$ openssl speed sha256
Doing sha256 for 3s on 16 size blocks: 11449960 sha256's in 2.99s
Doing sha256 for 3s on 64 size blocks: 6102397 sha256's in 2.99s
Doing sha256 for 3s on 256 size blocks: 2543220 sha256's in 2.99s
Doing sha256 for 3s on 1024 size blocks: 765381 sha256's in 2.99s
Doing sha256 for 3s on 8192 size blocks: 101501 sha256's in 2.99s
```

Say c = 4096. Generous back of envelope suggests that in 1 second, can test 931 passwords and so single-threaded naïve brute-force:

6 numerical digits	10^6 = 1,000,000	~ 17 minutes
6 lower case alphanumeric digits	36^6 = 2,176,782,336	~ 27 days
8 alphanumeric + 10 special symbols	72^8 = 722,204,136,308,736	~ 9 million days

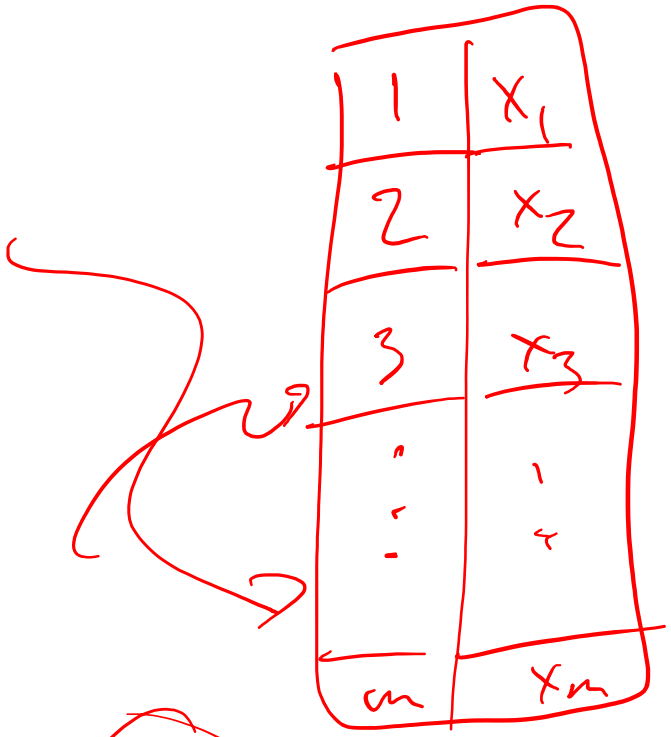
Special-purpose hashing chips (built for Bitcoin)

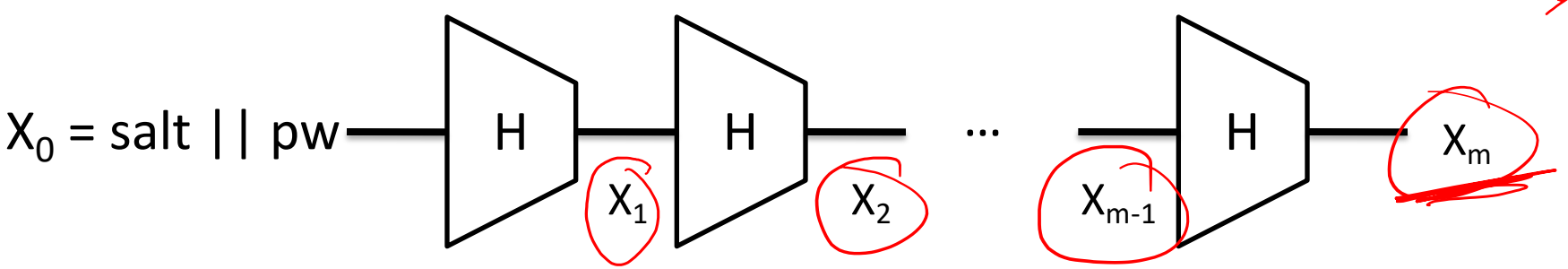
14,000,000,000,000 hashes / second

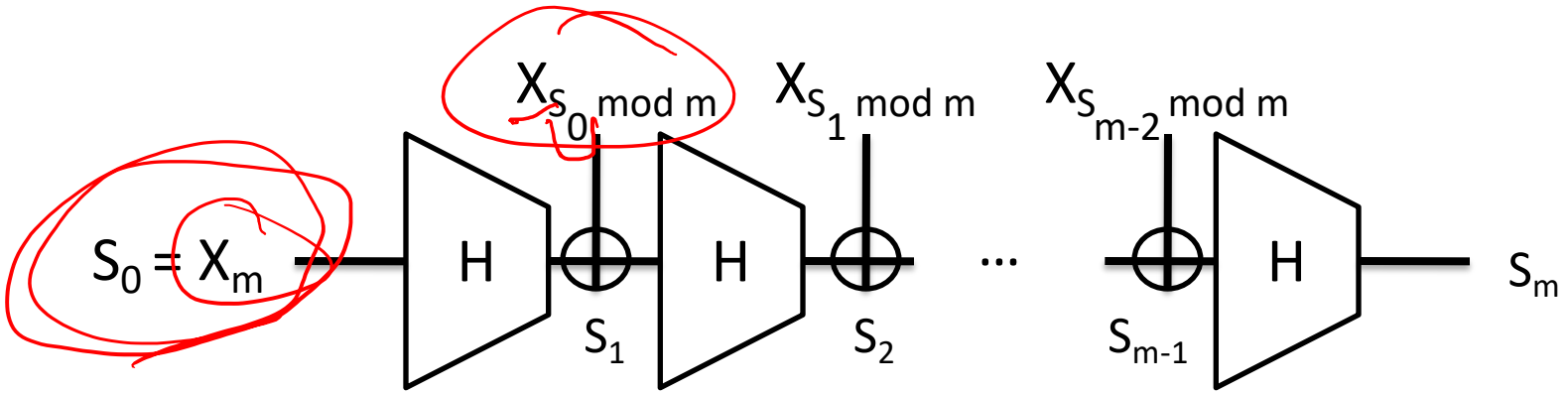
Can't be used easily for password cracking, but concern about ASICs (application-specific integrated circuits) remains

Scrypt and memory-hard hashing

- Increase time & memory needed to compute hash.
  - This makes ASIC implementations harder
- Scrypt



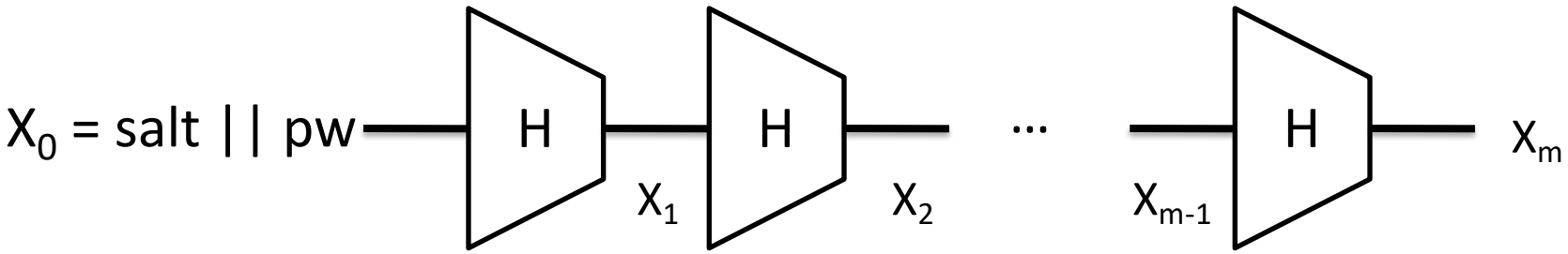


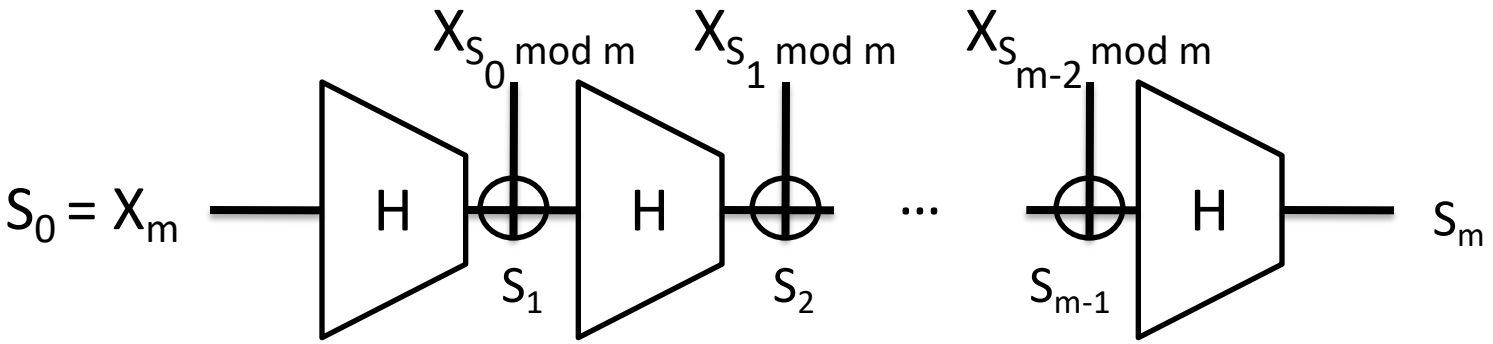


Scrypt and memory-hard hashing

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

data-indept
argon2i
independent
independent
concern about side-channel attacks

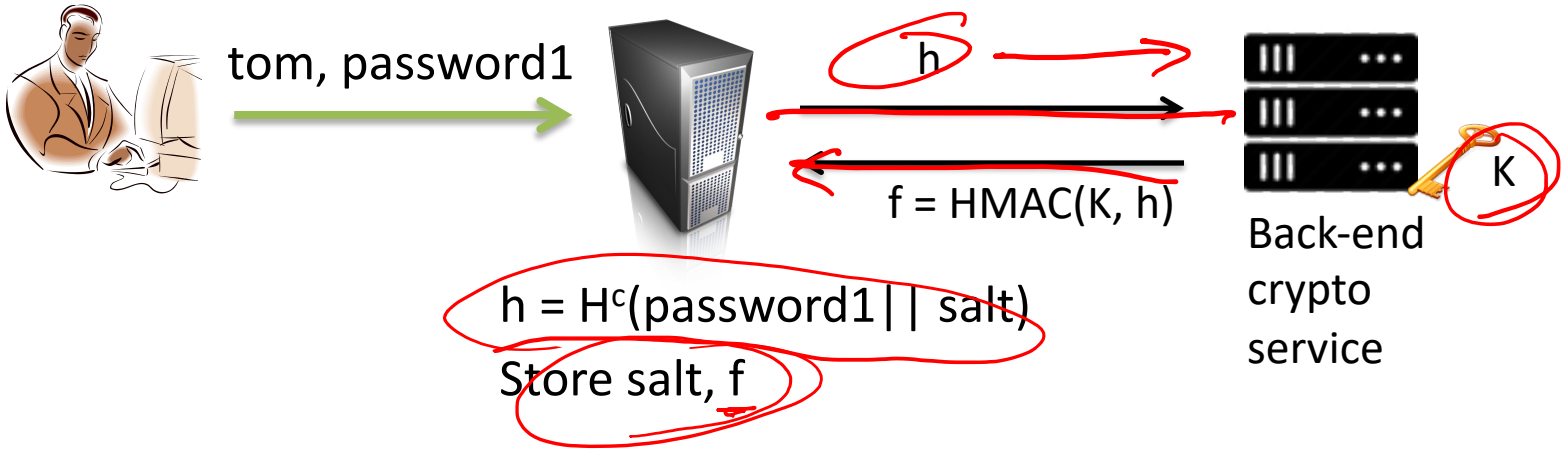


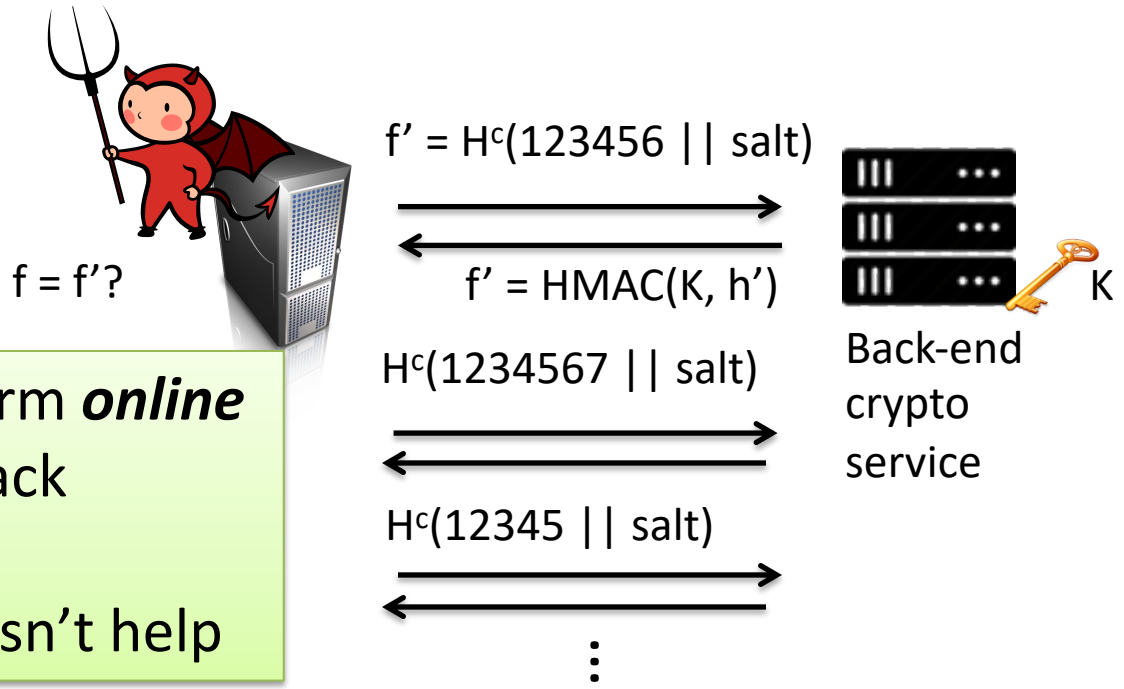


Facebook password onion

```
$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)
```

Strengthening password hash storage





Must still perform ***online*** brute-force attack

Exfiltration doesn't help

Credential stuffing attacks

tom, password1

Login server

tom	$salt_1$, $H(salt_1$, password1)
alice	$salt_2$, $H(salt_2$, 123456)
bob	$salt_3$, $H(salt_3$, p@ssword!)

tom	password1
sally	hell0w0rlD4
nick	blahblahblah
…	…

Breach alerting service

alert: tom has compromised credential!

';--have i been pwned?

Password Checkup

Offered by: google.com

Protocols for checking if password is in breach with third-party service

[Li et al. 2019] [Thomas et al. 2019]

Google & Li et al. credential checking protocol

Login server

tom, password1

Breach alerting service

tom	password1
sally	hell0w0rlD4
nick	blahblahblah
…	…

Holds secret key K for ***oblivious PRF***

Private set membership protocol:

- Client learns if (tom, password1) in breach database
- Server learns nothing about password1

We'll come back to it

Credential *tweaking* attacks

tom, password1

tom, Password1

tom, password123

Login server

tom	$salt_1$, $G_K(salt_1$, password123)
alice	$salt_2$, $G_K(salt_2$, 123456)
bob	$salt_3$, $G_K(salt_3$, p@ssword!)

tom	password1
sally	hell0w0rlD4
nick	blahblahblah
…	…

Breach alerting service

alert: tom has compromised credential!

Unclear if attackers using in wild yet

Early academic work suggests efficacy:

[Das et al. 2014]

[Wang et al. 2016]

} Ad hoc mangling rules

Our approach: password similarity models

[Pal et al. Oakland 2019]

User	Password Lists
tom	password1, cornelltechYay!
sally	hell0w0rlD1 helloworld123
nick	cispalsAwesome, Rugby123, rugby1
...	...

Lots of info about password similarity

First discovered by 4iQ on the "Dark Web"

1.4 billion email, password pairs
1.1 billion unique emails
463 million unique passwords

More than **150 million** users with **2 or more passwords**

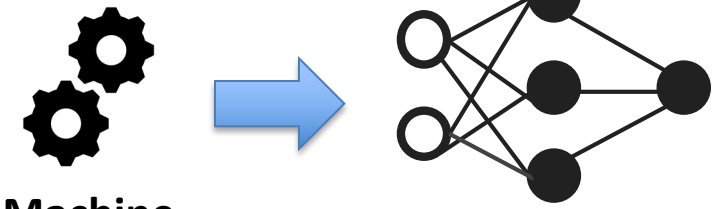
Around **10%** of distinct password pairs of same user are within **1 edit distance**

Our approach: password similarity models

[Pal et al. Oakland 2019]

User	Password Lists
tom	password1, cornelltechYay!
sally	hell0w0rlD1, helloworld123
nick	cispaIsAwesome, Rugby123, rugby1
…	…

Lots of info about password similarity



Machine learning

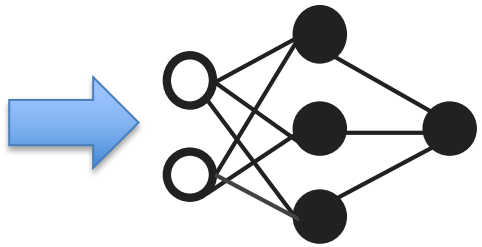
Similarity model:

$$P(w' \mid w)$$

Models probability user selects w' given they also chose password w

Good similarity model = good credential tweaking attack

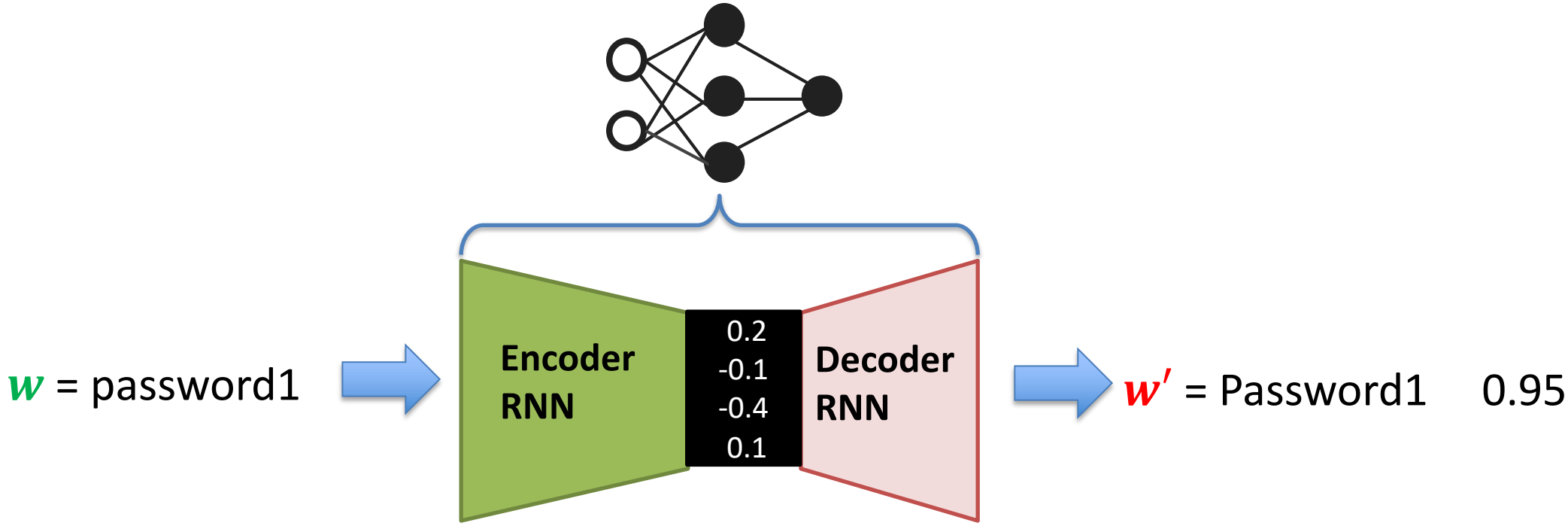
w = password1



Requires algorithm to enumerate
Use beam search

w'	$P(w' \mid w)$
Password1	0.95
password123	0.80
pASSWORD1	0.73
…	…

Password similarity models via deep learning

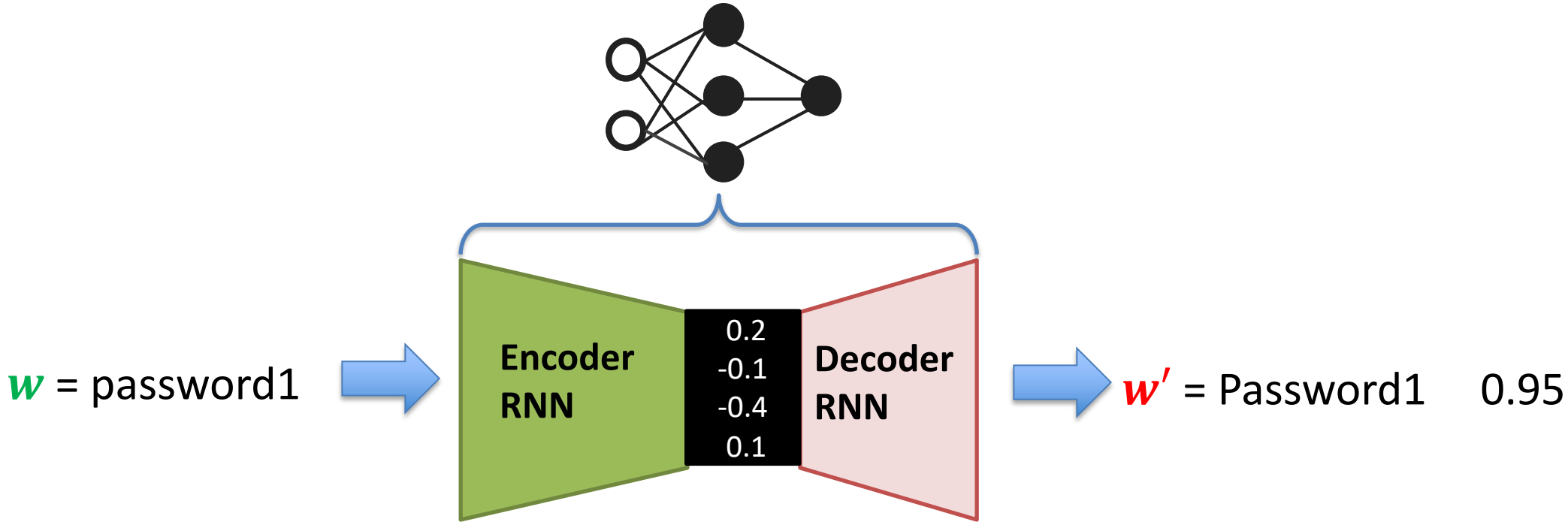


Recurrent neural network (RNN) encoder-decoder architectures

pass2pass: obvious approach like seq2seq [Sutskever et al. 2012] worse than prior work

pass2path: decode to a path (sequence of changes to w that yields w')
seems to better focus network on learning similarities

Password similarity models via deep learning



pass2path:

- Trained on **144 million** password pairs
- Took 2 days on Nvidia GTX 1080 GPU and Intel Core i9 processor
- Model has 2.4 million parameters, 60 MB on disk
- ~100 milliseconds to generate 1,000 guesses

Evaluating danger of credential tweaking

Audit of Cornell accounts in 2018

- ~500,000 accounts at Cornell
- Uses breach notification service
- Requires 8 character passwords with 3 different character classes

[Pal et al. 2019]

- 15,776 active accounts in leak dataset
- **Machine-learning based pass2path attack algorithm cracked 8.4% in ≤1,000 guesses**
- Put 1,374 vulnerable accounts on watch list

First real-world evaluation of credential tweaking!

Defending against credential tweaking attacks

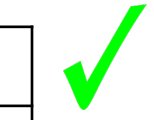
tom, password1

tom, Password1

tom, password123

Login server

tom	$salt_1$, $G_K(salt_1$, password123)
alice	$salt_2$, $G_K(salt_2$, 123456)
bob	$salt_3$, $G_K(salt_3$, p@ssword!)



tom	password1
sally	hell0w0rlD4
nick	cispaIsAwesome
…	…

Breach notification services

Tom please reset password!

Can we guide users away from picking passwords similar to those in breaches?

Personalized password strength meter (PPSM)
Take into account previous passwords when giving strength score

[Pal et al. Oakland 2019]

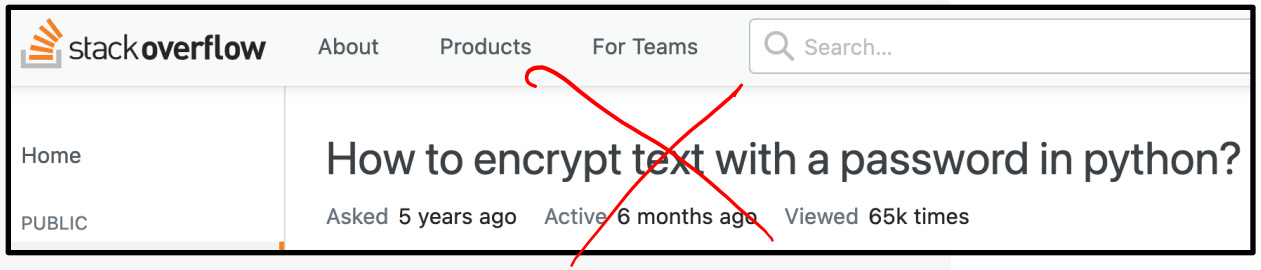
Can we build similarity-aware breach alerting services?

Might I Get Pwned (MIGP)
Deployment reduces efficacy of best 1,000-guess credential tweaking attacks by 94%

[Pal et al. Security 2022]

Password-based encryption

- Sometimes want to encrypt data using password
- Examples:
  - Password vaults (e.g., LastPass)
  - WhatsApp chat backups
  - SSH private key files
  - …




```python
import base64
from Crypto.Cipher import AES
from Crypto.Hash import SHA256
from Crypto import Random

def encrypt(key, source, encode=True):
    key = SHA256.new(key).digest()  # use SHA-256 over our key to get a proper-size
    IV = Random.new().read(AES.block_size)  # generate IV
    encryptor = AES.new(key, AES.MODE_CBC, IV)
    padding = AES.block_size - len(source) % AES.block_size  # calculate needed pad
    source += bytes([padding]) * padding  # Python 2.x: source += chr(padding) * pa
    data = IV + encryptor.encrypt(source)  # store the IV at the beginning and encr
    return base64.b64encode(data).decode("latin-1") if encode else data

def decrypt(key, source, decode=True):
    if decode:
        source = base64.b64decode(source.encode("latin-1"))
    key = SHA256.new(key).digest()  # use SHA-256 over our key to get a proper-size
    IV = source[:AES.block_size]  # extract the IV from the beginning
    decryptor = AES.new(key, AES.MODE_CBC, IV)
    data = decryptor.decrypt(source[AES.block_size:])  # decrypt
    padding = data[-1]  # pick the padding value from the end; Python 2.x: ord(data
    if data[-padding:] != bytes([padding]) * padding:  # Python 2.x: chr(padding) *
        raise ValueError("Invalid padding...")
    return data[:-padding]  # remove the padding
```

Problems in that stackoverflow example

- No authentication mechanism
  - Likely will be vulnerable to padding oracle attacks
- Using just SHA256 for key derivation
  - Should use a slow hash instead

PW-based Encryption

```
PWEnc(pw,M):
salt <-$  {0,1}^256
K <- PBKDF(pw,salt)
C <- Enc(K,M)
Return (salt,C)
```

```
PWDec(pw,salt||C):
K <- PBKDF(pw,salt)
M <- Dec(K,C)
Return M
```

Enc need only be *one-time-secure* AE scheme, such as CTR-then-HMAC with constant IV for CTR

Ok to use random IV's as well

Using scrypt instead of PBKDF also good

Summary

- Passwords widely used in practice
- Never store passwords in clear, should at least be salted and slow hashed (PBKDF)
  - Memory-hard hashes are state-of-the art (scrypt)
- Password breach alerting increasingly used to warn users about exposed passwords
  - See what I worked on at Cloudflare: https://blog.cloudflare.com/privacy-preserving-compromised-credential-checking/
- Password-based encryption should derive key for AEAD scheme using PBKDF
  - Beware of what you see on internet about applied crypto