Authentication

Lecture 04

Software Security Engineering

Winter 2023
Thompson Rivers University

Authentication: process of using supporting evidence to corroborate an asserted identity

Authentication: process of using supporting evidence to corroborate an asserted identity

Identification (recognition): establish identity from available information (without assertion)

Authentication: process of using supporting evidence to corroborate an asserted identity

Identification (recognition): establish identity from available information (without assertion)

Authorization: determining if a request should be granted based on an entity

User Authentication

- something you know
  - e.g., password, pin code
- something you have
  - e.g., hardware token, bank card
- something you are
  - i.e., biometrics
  - e.g., fingerprint, iris

Password

- a secret associated with a (public) user identity (userid)
- to authenticate:
  - user sends userid and password
  - server authenticates if password is correct for userid

Attacks of password-based authentication systems

- online guessing attacks
  - attacker tries logging in by guessing password
- eavesdropping
  - attacker on the network intercepts the password
- server compromise
  - attacker compromises server and reads stored password
- social engineering and phishing
  - attacker fools user into revealing password
- client-side malware
  - keylogging or other malware captures password

Mitigation for online-guessing

- rate-limiting
  - timeout, lockout
- “Completely Automated Public Turing test to tell Computers and Humans Apart” (CAPTCHA)
  - prevent automated guessing
- then make the password hard to guess in the allowed tries
  - password requirements
  - length, caps, punctuation
  - force “good” passwords

Mitigation for eavesdropping:

use encryption, etc., to secure the network communication.

Mitigation for server compromise …

Approach 1: server stores

<userX, passwordX>

for all users.

problem?

Approach 1: server stores
<userX, passwordX>
for all users.

problem?

Attackers who compromise the server,
get access to all <user, password>s

Approach 2: server stores

<userX, E_k(passwordX)>

for some key k

Approach 2: server stores

<userX, E_k(passwordX)>

for some key k

any problem?

Approach 2: server stores

<userX, E_k(passwordX)>

for some key k

any problem?

The same problem. The server needs to keep the key k in memory.

attackes can have access to the key

Approach 3: server stores
<userX, H(passwordX)>
for all users

Approach 3: server stores

<userX, H(passwordX)>

for all users

problems? (think of the magician exmple)

Approach 3: server stores

<userX, H(passwordX)>

for all users

problems? (think of the magician exmple)

attackers can burte force possible passwords

Approach 4: server stores

<userX, H(H(H(...(passwordX))))>

for all users

Approach 4: server stores
<userX, H(H(H(...(passwordX))))>
for all users

problems?

Approach 4: server stores

<userX, H(H(H(...(passwordX))))>

for all users

effectively use a slow hash that takes a while to compute

Approach 4: server stores

<userX, H(H(H(...(passwordX))))>

for all users

effectively use a slow hash that takes a while to compute

if H is 1000*slower, a day-long guessing attack now takes 3 years

Hash Chain

- this repeated application of hashing is called a hash chain
  - it is used to perform key strengthening
- you repeat hashing so it’s still fast to execute in practice when checking passwords
- but if you do it 10000 tiems it takes the adversary 10000 times longer to compute all password’s hash values
  - this helps, but what if the attacker just stored a giant precomputed table of hashes
  - they would pay this cost once but be able to break passwords in constant time

Table of All Password Hashes

- the number of possible passwords is huge
  - how can you except to store this?
  - you can store the password and hash for the first hundred billion passwords
    - around 3 TiB
    - but you need to still run to check the rest
- solution: use hash chains
  - sadly this is not the kind of hash chains we just talked about
  - Hellman, 1980, “A cryptographic Time Memory Tradeoff” (at that time, 3 TiB was impossible to imagine)
  - Hellman introduced a solution to efficiently store passwords and hashes

Concise Password Table

- use a function R that maps hashed back to the domain space
  - this doesn’t need to be a hash function, or the reverse function, just any bona fide random mapping
  - e.g., interpret the hash as a number and have the passwords ordered
- pick a random password P
- compute H(P), R(H(P)), H(R(H(P))), R(H(R(H(P)))), …
  - R is function that maps a Hash back to a possible password
- every so often stop doing this and record the P and the last value E as (P, E).
  - E = R(H(… H(P) …)
  - given P, you can compute E by hashing and returning
  - each (P, E) pais “stores” the entire chain between them

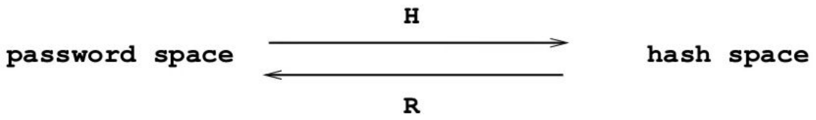
Concise Password Table

- given a hashed password x, run this computation forward until you find a know E, then run it forward from P until you find x
  - the value right before is the password
- allows you to only store some number of (P, E)
  - the length of the chain is the amount of work you'll have to redo
  - the number of (P, E) pairs is the amount of space you'll need
- the choice of the reverse function is important
  - we still want to prioritize likely passwords since all passwords is too large
  - but if R has collisions to the passwords, we won't notice right away (how we do?)
  - collisions waste time and space! (why is this?)

password space

hash space

password space $\xrightarrow{\mathbf{H}}$ hash space


H
password space
hash space
R

H

password space → hash space

← R

cat

12345

p@ssword

123456

123456789

qwerty

zxasqw

topsecret

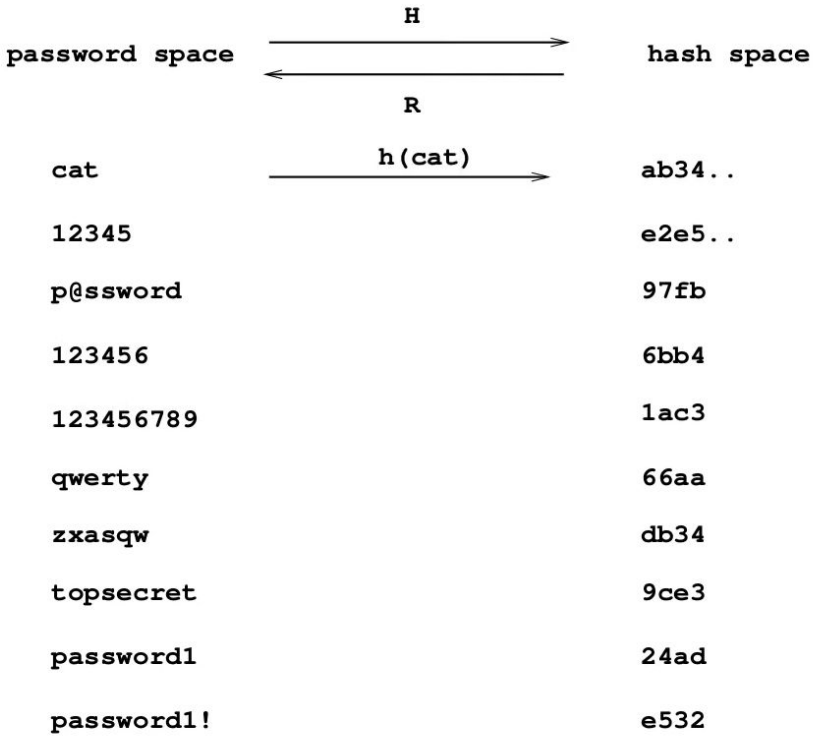
password1

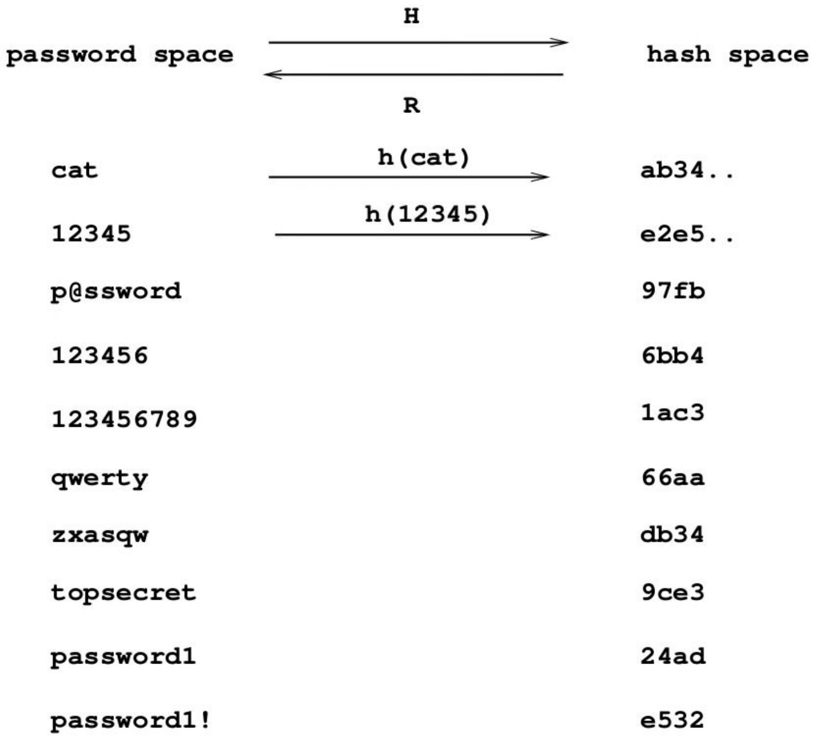
password1!

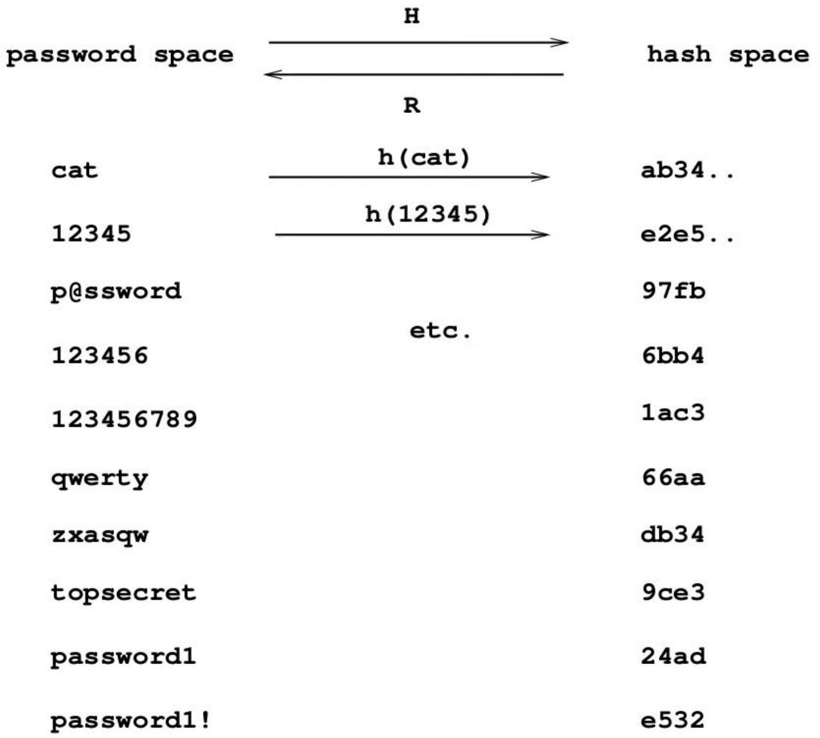
password space $\xrightarrow{H}$ hash space

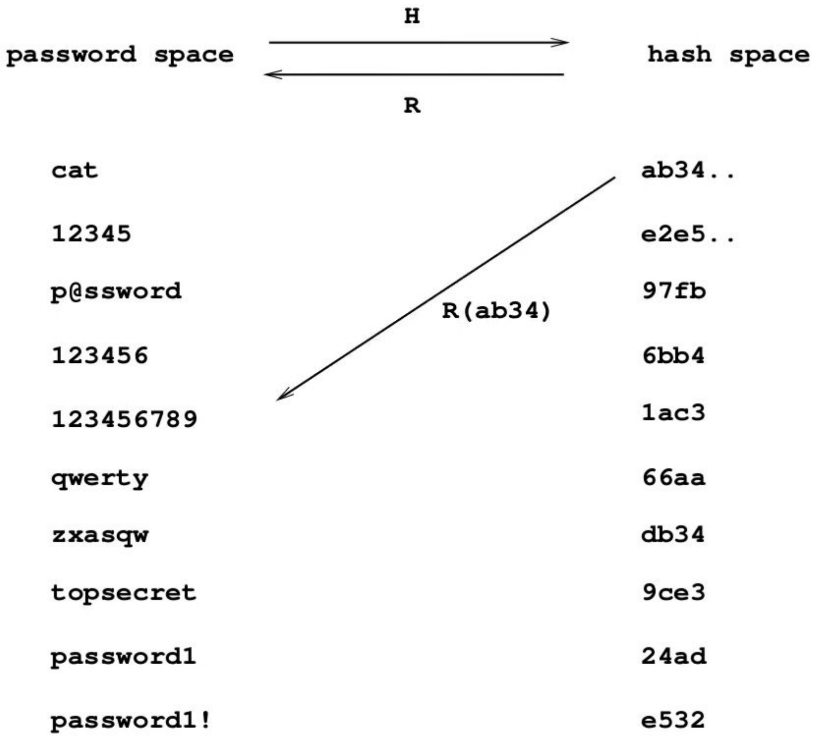
password space $\xleftarrow{R}$ hash space

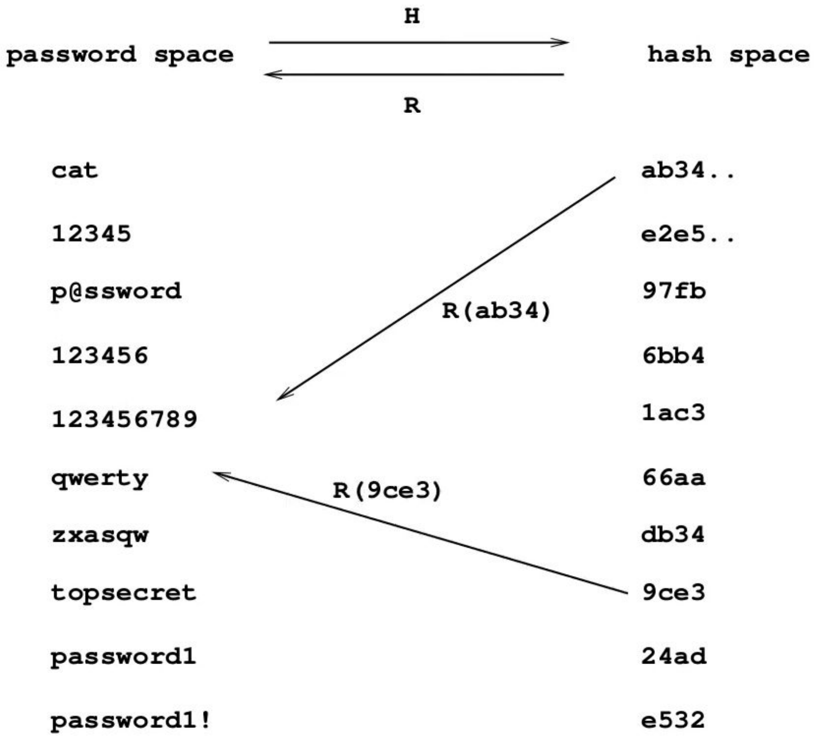
cat	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

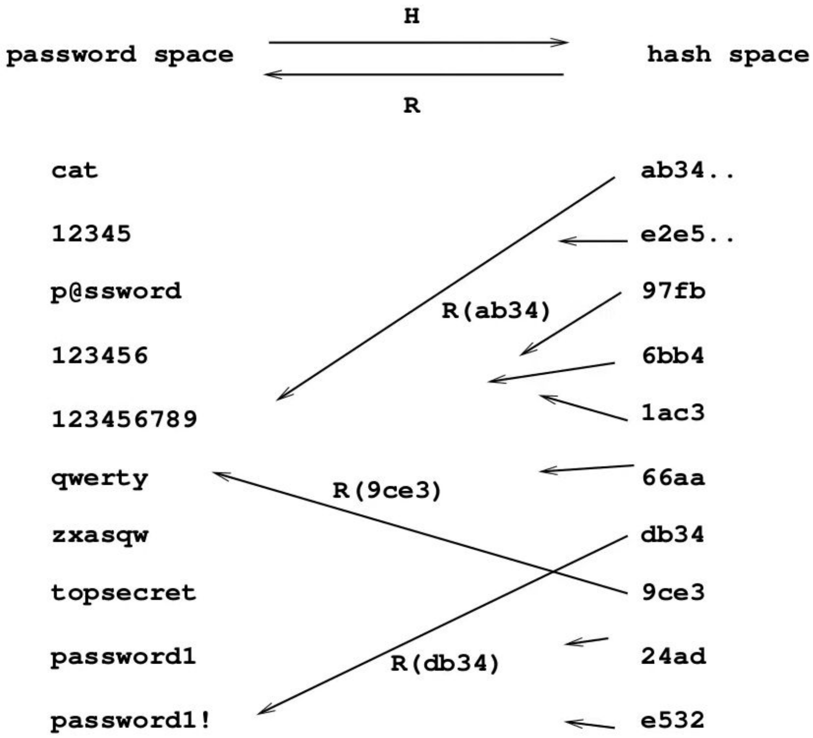

H
password space
hash space
R
cat
h(cat)
ab34..
12345
e2e5..
p@ssword
97fb
123456
6bb4
123456789
1ac3
qwerty
66aa
zxasqw
db34
topsecret
9ce3
password1
24ad
password1!
e532


H
password space
hash space
R
cat
h(cat)
ab34..
12345
h(12345)
e2e5..
p@ssword
97fb
123456
6bb4
123456789
1ac3
qwerty
66aa
zxasqw
db34
topsecret
9ce3
password1
24ad
password1!
e532


password space
H
R
hash space
cat
h(cat)
ab34..
12345
h(12345)
e2e5..
p@ssword
97fb
etc.
123456
6bb4
123456789
1ac3
qwerty
66aa
zxasqw
db34
topsecret
9ce3
password1
24ad
password1!
e532


H
password space
hash space
R
cat
12345
p@ssword
123456
123456789
qwerty
zxasqw
topsecret
password1
password1!
ab34..
e2e5..
97fb
6bb4
1ac3
66aa
db34
9ce3
24ad
e532
R(ab34)


H
password space
hash space
R
cat
12345
p@ssword
123456
123456789
qwerty
zxasqw
topsecret
password1
password1!
ab34..
e2e5..
97fb
6bb4
1ac3
66aa
db34
9ce3
24ad
e532
R(ab34)
R(9ce3)


H
password space
hash space
R
cat
12345
p@ssword
123456
123456789
qwerty
zxasqw
topsecret
password1
password1!
ab34..
e2e5..
97fb
6bb4
1ac3
66aa
db34
9ce3
24ad
e532
R(ab34)
R(9ce3)
R(db34)

password space	H → / ← R	hash space
cat		ab34..
12345		e2e5..
p@ssword		97fb
123456		6bb4
123456789		1ac3
qwerty		66aa
zxasqw		db34
topsecret		9ce3
password1		24ad
password1!		e532

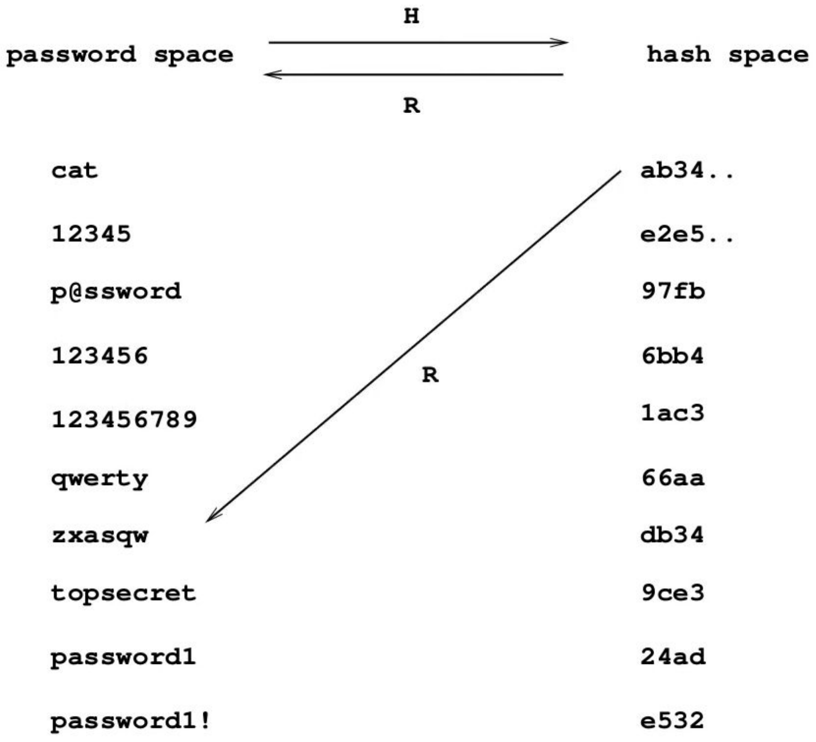
cat

password space —H→ hash space

password space ←R— hash space

cat —H→	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

cat -> ab34

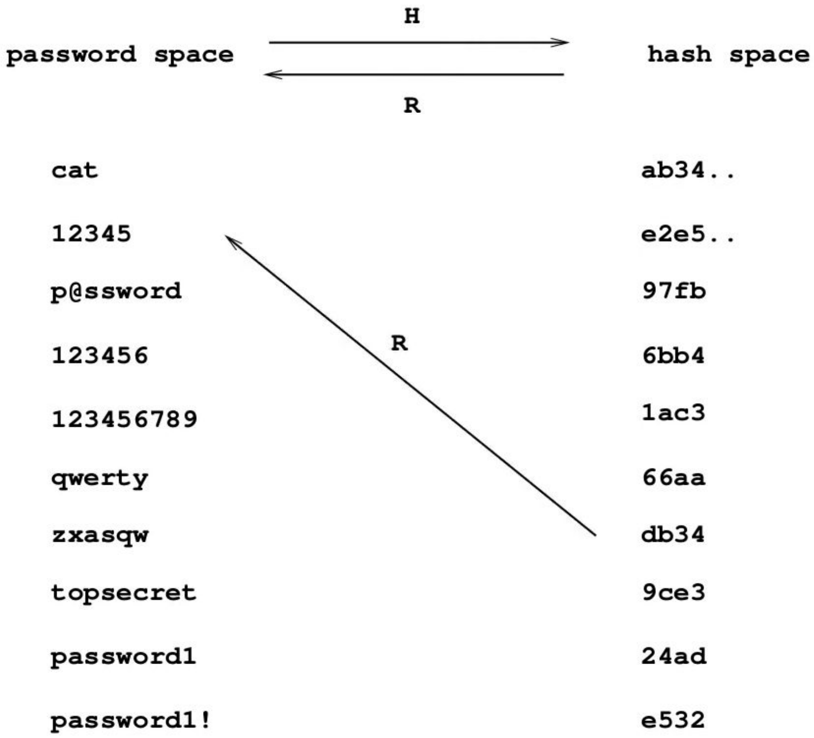


cat -> ab34 -> zxasqw

password space —H→ hash space
password space ←R— hash space

cat	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw —H→	db34
topsecret	9ce3
password1	24ad
password1!	e532

cat -> ab34 -> zxasqw -> db34



cat -> ab34 -> zxasqw -> db34 -> 12345

password space —H→ hash space

password space ←R— hash space

cat	ab34..
12345 —H→	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

password space → (H) → hash space

password space ← (R) ← hash space

password	hash	start .. end
cat	ab34..	[cat .. e2e5]
12345	e2e5..	
p@ssword	97fb	
123456	6bb4	
123456789	1ac3	
qwerty	66aa	
zxasqw	db34	
topsecret	9ce3	
password1	24ad	
password1!	e532	

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

H

password space ⇄ hash space

R

		start	end
cat	ab34..	[cat .. e2e5]	
12345	e2e5..		
p@ssword	97fb		
123456	6bb4		
123456789	1ac3		
qwerty	66aa		
zxasqw	db34		
topsecret	9ce3		
password1	24ad		
password1!	e532		

2 pieces
of data

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

H

password space ⟶ / ⟵ hash space

R

start end

password space	hash space
cat	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

[cat .. e2e5]

2 pieces
of data

arbitrarily
long chain

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

H

password space → hash space

← R

password space	hash space	start	end
cat	ab34..	[cat ..	e2e5]
12345	e2e5..		
p@ssword	97fb		
123456	6bb4		
123456789	1ac3		
qwerty	66aa		
zxasqw	db34		
topsecret	9ce3		
password1	24ad		
password1!	e532		

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

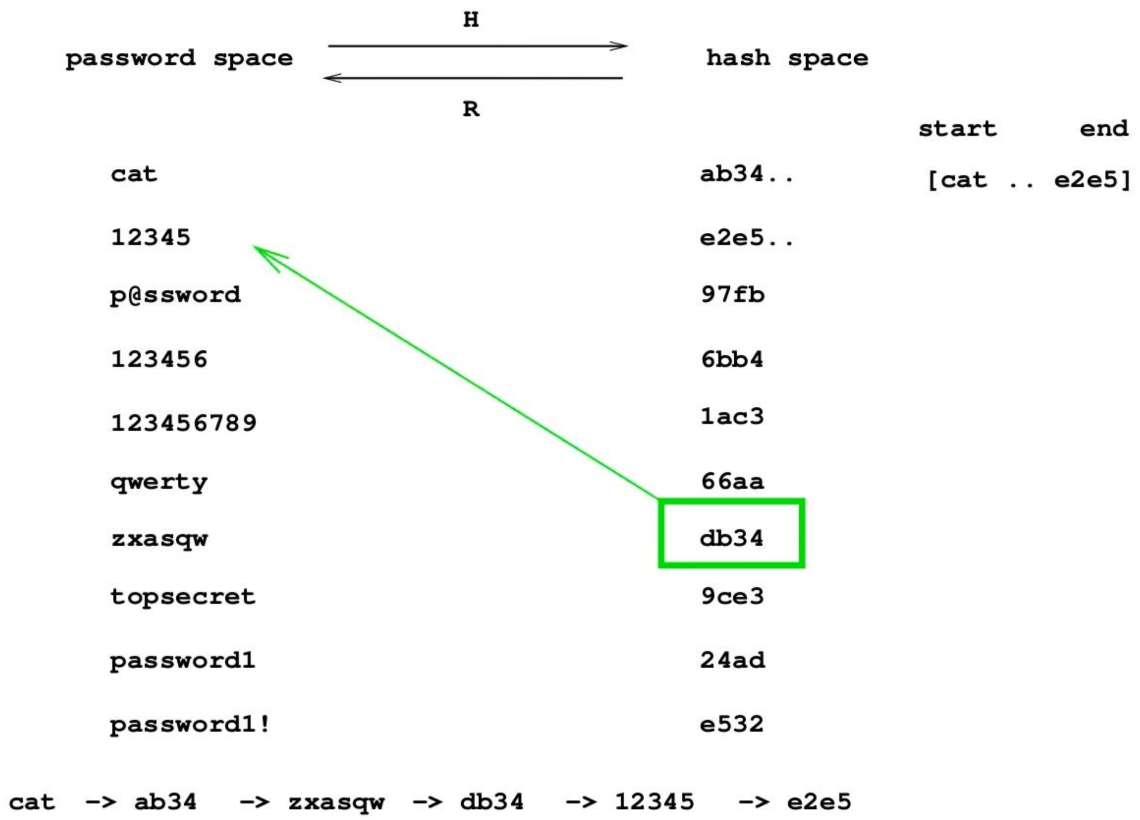
H

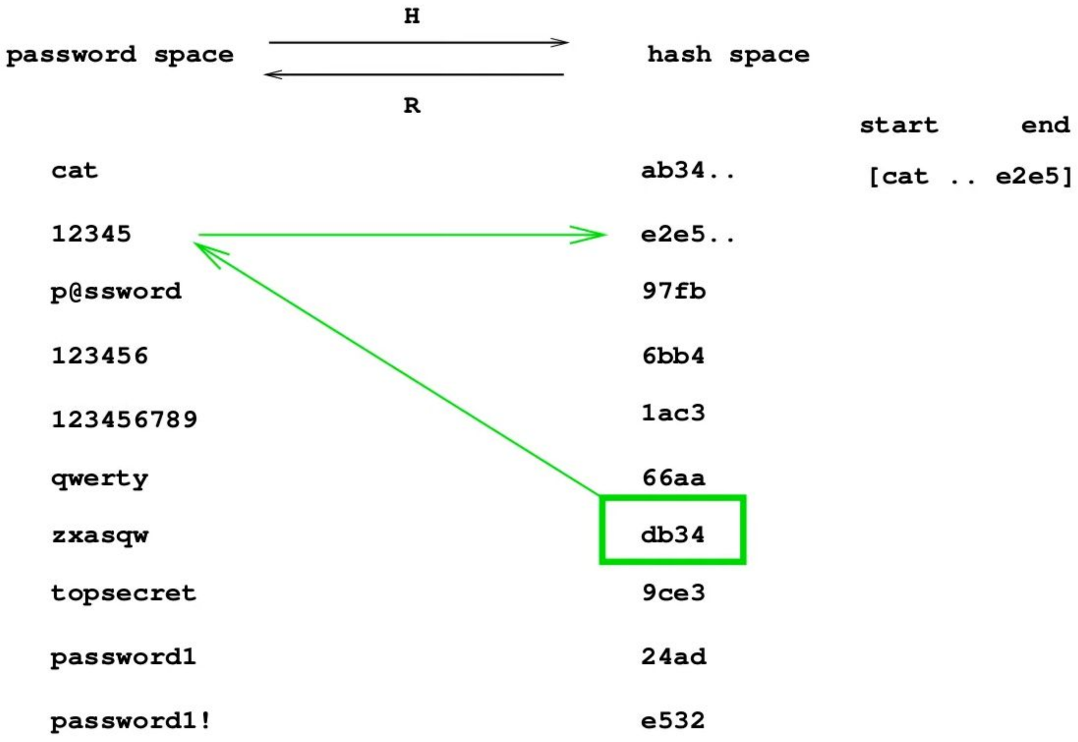
password space → hash space

← R

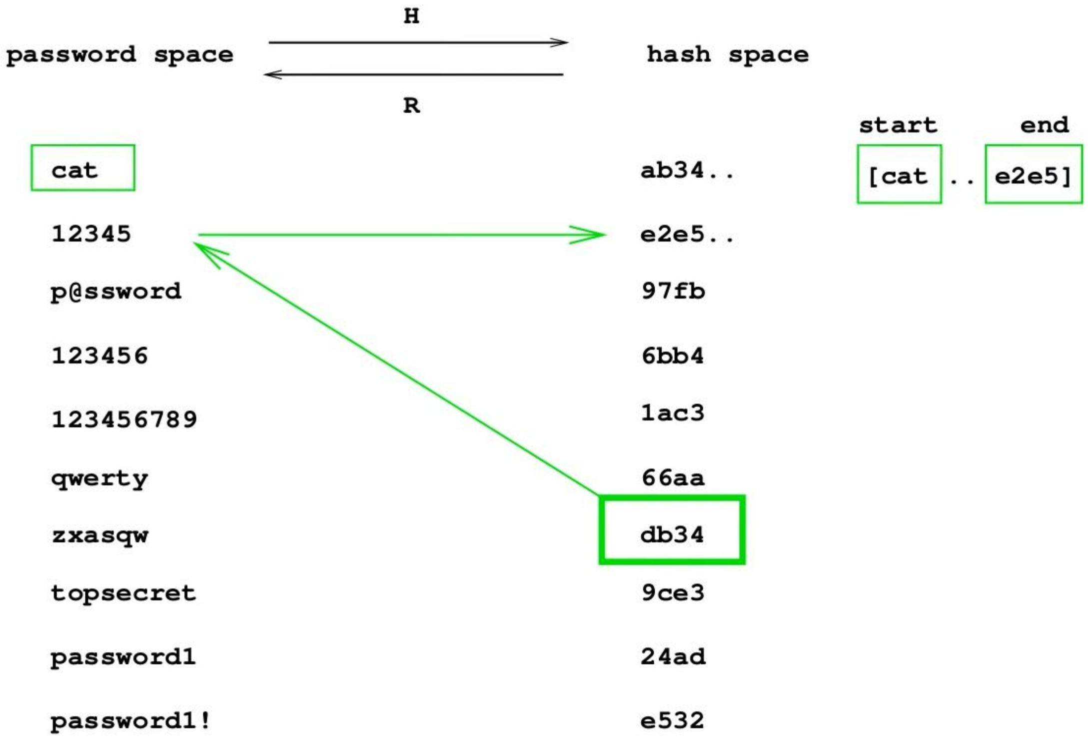
		start	end
cat	ab34..	[cat ..	e2e5]
12345	e2e5..		
p@ssword	97fb		
123456	6bb4		
123456789	1ac3		
qwerty	66aa		
zxasqw ←	db34		
topsecret	9ce3		
password1	24ad		
password1!	e532		

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

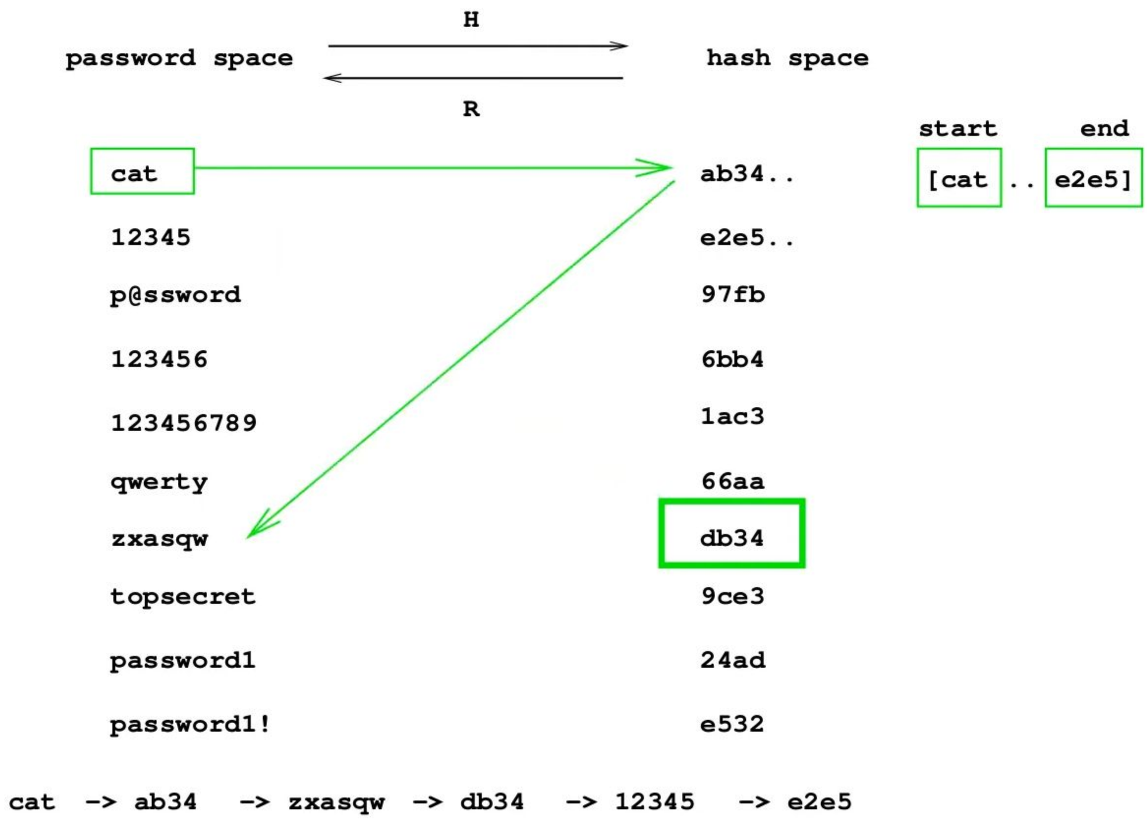

H
password space
hash space
R
start end
[cat .. e2e5]
cat
ab34..
12345
e2e5..
p@ssword
97fb
123456
6bb4
123456789
1ac3
qwerty
66aa
zxasqw
db34
topsecret
9ce3
password1
24ad
password1!
e532
cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

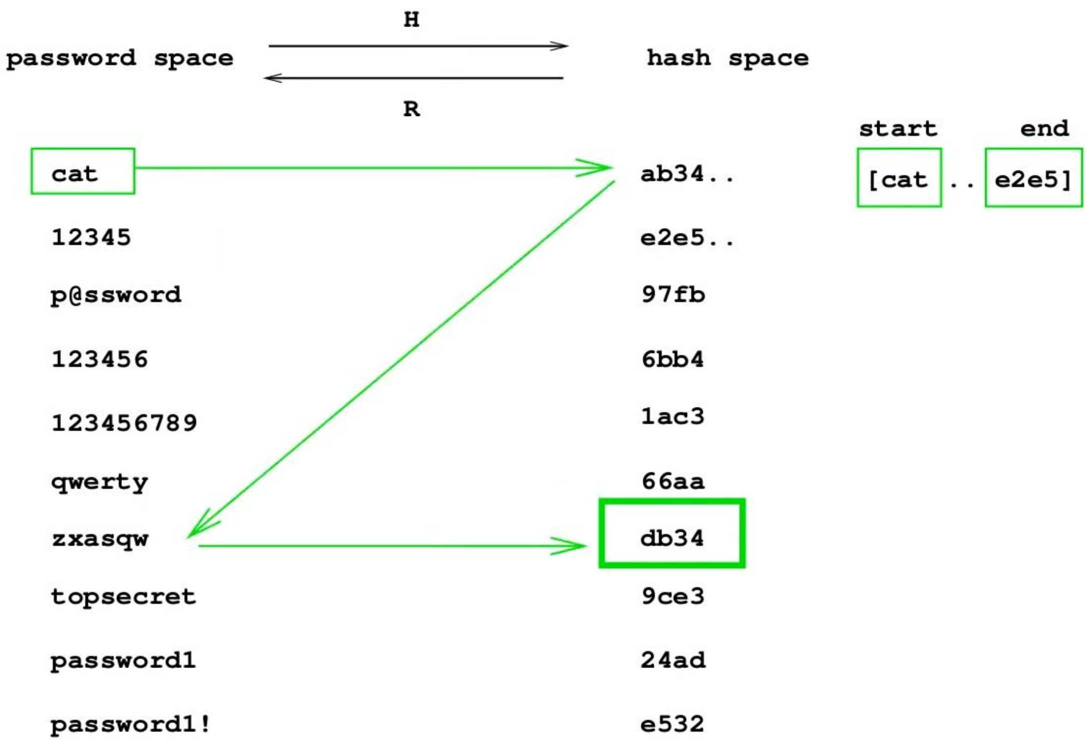


cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

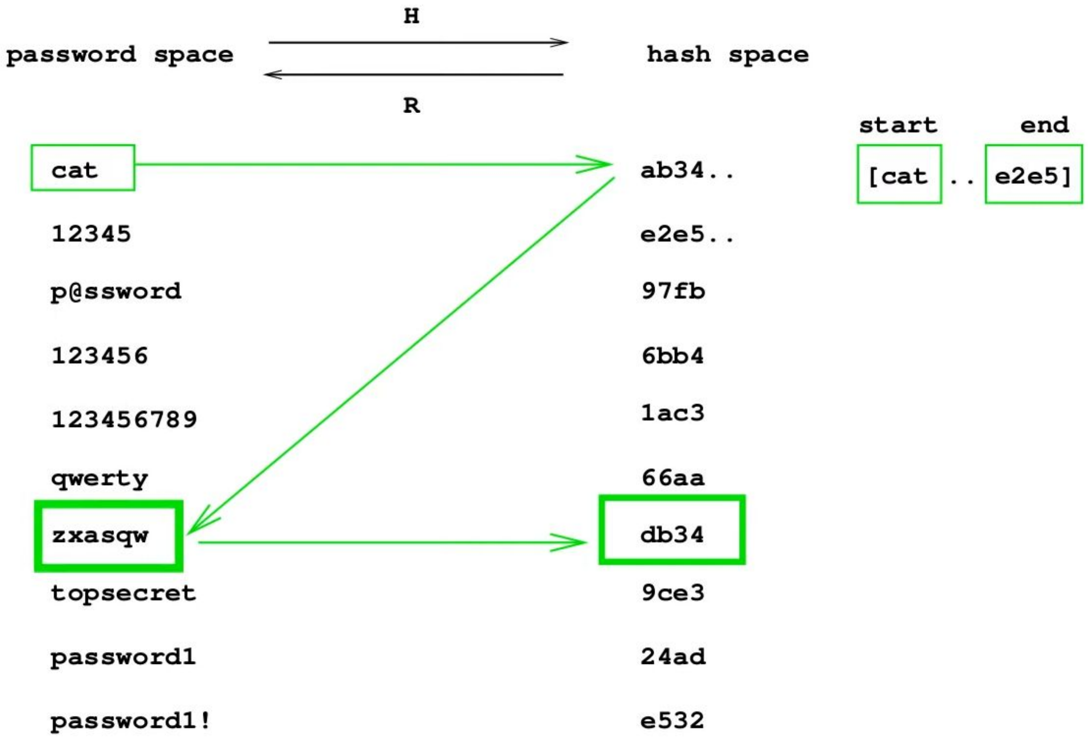


cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5


H
password space
hash space
R
start
end
cat
ab34..
[cat
..
e2e5]
12345
e2e5..
p@ssword
97fb
123456
6bb4
123456789
1ac3
qwerty
66aa
zxasqw
db34
topsecret
9ce3
password1
24ad
password1!
e532
cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5



cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5



cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

password space —H→ hash space
password space ←R— hash space

password space	hash space	start .. end
cat	ab34..	[cat .. e2e5]
12345	e2e5..	
p@ssword	97fb	
123456	6bb4	
123456789	1ac3	
qwerty	66aa	
zxasqw	db34	
topsecret	9ce3	
password1	24ad	
password1!	e532	

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

topsecret -> 9ce3 -> qwerty -> 66aa -> 12345 -> e2e5

password space —H→ hash space
password space ←R— hash space

cat	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

start	end
[cat ..	e2e5]
[topsecret ..	e2e5]

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

topsecret -> 9ce3 -> qwerty -> 66aa -> 12345 -> e2e5

password space —H→ hash space

password space ←R— hash space

password space	hash space
cat	ab34..
12345	e2e5..
p@ssword	97fb
123456	6bb4
123456789	1ac3
qwerty	66aa
zxasqw	db34
topsecret	9ce3
password1	24ad
password1!	e532

start	end
[cat	.. e2e5]
[topsecret	.. e2e5]

if we end on e2e5
we have to spawn
two seaches,
one from cat
and one from
topsecret

cat -> ab34 -> zxasqw -> db34 -> 12345 -> e2e5

topsecret -> 9ce3 -> qwerty -> 66aa -> 12345 -> e2e5

Rainbow Tables

- a password table that solves the collision problem
- instead of using one return function R, use a family of them: $R_1, \ldots, R_k$
- the chains are built using R_1 first, then R_2, etc.
- wasteful collisions only happen if they are one the same round
  - i.e., if $R_i(h_1) = R_i(h_2)$ then the chains will agree after
  - but if $R_i(h_1) = R_j(h_2)$, it is not guaranteed that R_{i+1} and R_{j+1} will continue to agree

These rainbow table already exist.

So how can we defeat them?

Rainbow Table Specification

Algorithm	Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files
LM	lm_ascii-32-65-123-4#1-7	ascii-32-65-123-4	1 to 7	7,555,858,447,479 ≈ $2^{42.8}$	99.9 %	27 GB	Files
NTLM	ntlm_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495 ≈ $2^{46.0}$	99.9 %	52 GB	Files
NTLM	ntlm_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120 ≈ $2^{52.6}$	96.8 %	460 GB	Files
NTLM	ntlm_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090 ≈ $2^{47.7}$	99.9 %	127 GB	Files
NTLM	ntlm_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642 ≈ $2^{53.6}$	96.8 %	690 GB	Files
NTLM	ntlm_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084 ≈ $2^{46.6}$	99.9 %	65 GB	Files
NTLM	ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files
MD5	md5_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495 ≈ $2^{46.0}$	99.9 %	52 GB	Files
MD5	md5_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120 ≈ $2^{52.6}$	96.8 %	460 GB	Files
MD5	md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090 ≈ $2^{47.7}$	99.9 %	127 GB	Files
MD5	md5_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642 ≈ $2^{53.6}$	96.8 %	690 GB	Files
MD5	md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084 ≈ $2^{46.6}$	99.9 %	65 GB	Files
MD5	md5_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files
SHA1	sha1_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495 ≈ $2^{46.0}$	99.9 %	52 GB	Files
SHA1	sha1_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120 ≈ $2^{52.6}$	96.8 %	460 GB	Files
SHA1	sha1_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090 ≈ $2^{47.7}$	99.9 %	127 GB	Files
SHA1	sha1_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642 ≈ $2^{53.6}$	96.8 %	690 GB	Files
SHA1	sha1_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084 ≈ $2^{46.6}$	99.9 %	65 GB	Files
SHA1	sha1_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files

Approach 5: server stores

<userX, saltX, H(H(H(...(H(passwordX, saltX))))>
for all users

Approach 5: server stores

<userX, saltX, H(H(H(...(H(passwordX, saltX))))>

for all users

This is the best approach to store user passwords.

What do you use to hash a password?

Hashing Passwords

- MD5 and SHA-1 are designed to be collision and preimage resistant and to run as fast as possible
- this helps offline guessing attacks
  - this is why we had the (H(H(H(H(...)))) construction
- GPUs and specialized hardware can make this much faster
  - expensive for every log-in server to have to buy
  - cheap for one attacker
  - this is sunk-cost / all-front problem
- instead, use hash functions that aren't GPU solvable
- Argon2 is preferred (won competition)
  - uses memory in hashing to stop GPU attacks
  - take number of iterations, salt, and memory required as arguments

Why do we use passwords?

Forbes

FORBES › MONEY › FINTECH

Thanks To Apple, Microsoft And Google Passwords Will Finally Die

People have said that the end of password era is upon us, but we still use passwords all the time

Disadvantage of Passwords?

Disadvantage of Passwords?

must memorize; inconsistent composition policies;
cannot re-use; change every so often;
impossible balance between easy to remember and hard to guess;
vulnerable to capture and reply;
vulnerable to online and offline guessing attacks.

Advantages of Passwords?

Advantages of Passwords?

simple to use and understand; no extra hardware;
nothing to carry; quick login; easy to change if lost;
failure mode is clear; no trust in third party;
easily delegated (though hard to undelegate).

Example Password Guessing Attack

President Trump's Twitter accessed by security expert who guessed password 'maga2020!'

Zack Whittaker @zackwhittaker / 9:37 AM MDT • October 22, 2020

Comment

Dutch prosecutors believe a security researcher hacked President Donald Trump's Twitter account in October, despite earlier denials from the White House and Twitter. According to *The Guardian*, a specialist police team investigated hacker Victor Gevers, who claimed to have guessed Trump's password as "maga2020!" and breached his account. "We believe the hacker has actually penetrated Trump's Twitter account, but has met the criteria that have been developed in case law to go free as an ethical hacker," a public prosecutor's office spokesperson told *The Guardian*.

Password Recovery

- used if password if forgotten
  - major failure mode of passwords
  - what’s the other failure mode?
- principle:
  - server authenticated the user some other way
  - a working password is then delivered to that user
- password reset
  - ideally server doesn’t actually know password
    - e.g., is stored hashed
  - user is given opportunity to reset the password

Password Recovery

- typically send password or a link over email
  - account created with an email address
  - uses the fact that I still know my password to email
- this means there are now two ways of logging in
  - i.e., either of two passwords can work
  - `WEAKEST LINK SECURITY`

Default Passwords

- Pennsylvania ice cream shop phone scam
  - voicemail PIN default to last four digits of phone number
    - SAFE DEFAULTS
  - criminals change message to “I accept collect call” and make $8600 call
- A US courthouse server: “public” / “public”
- NY Times employee DB: password = last 4 SSN digits

Gary McKinnon: Scottish sysadmin and hacker

Gary McKinnon: Scottish sysadmin and hacker

In 2001/2: hacked 97 US military and NASA computers

Gary McKinnon: Scottish sysadmin and hacker

In 2001/2: hacked 97 US military and NASA computers

Goal: find evidence of UFO coverups
and free energy tech suppressions

Gary McKinnon: Scottish sysadmin and hacker

In 2001/2: hacked 97 US military and NASA computers

Goal: find evidence of UFO coverups
and free energy tech suppressions

Method: perl script randomly looking for
blank and default passwords to administrator accounts

Rockyou Hack (2009)

- “social gaming” company
- database with 32 million user passwords from partner social networks
- passwords stored in the clear (plaintext)
- December 2009: entire database hacked using an SQL injection attack
  - more on this later!

Top Passwords

Top Passwords

123456

Top Passwords

123456

12345

Top Passwords

123456

12345

123456789

Top Passwords

123456

12345

123456789

Password

Top Passwords

123456

12345

123456789

Password

iloveyou

Top Passwords

123456
12345
123456789
Password
iloveyou
princess

Top Passwords

123456
12345
123456789
Password
iloveyou
princess
rockyou

Top Passwords

123456
12345
123456789
Password
iloveyou
princess
rockyou
1234567

Top Passwords

123456
12345
123456789
Password
iloveyou
princess
rockyou
1234567
12345678

Top Passwords

123456
12345
123456789
Password
iloveyou
princess
rockyou
1234567
12345678
abc123

Adobe Passwords (2013)

- leaked about 38 million active user accounts
- encrypted with 3DES in ECB mode
- the key was not leaked
- included user-settable password hints

Linkedin Hack (2012)

- 177 million unsalted SHA1 password hashes
- most common
  - 123456
  - linkedin
  - password
  - 123456789
  - 12346578
  - 111111
  - 1235467
  - 654321
  - qwerty
  - sunshine
  - 000000

Poring through the database, the trio found an entry for Trump as well as the hash for Trump's password:

```
07b8938319c267dcdb501665220204bbde87bf1d
```

Generate the hash of the string you input.

yourefired

Checksum type: ○ MD5 ● SHA1 ○ SHA-256

String hash: 07B8938319C267DCDB501665220204BBDE87BF1D

Calculate

First part is a dictionary attack to get password

First part is a dictionary attack to get password

second part is credential stuffing

First part is a dictionary attack to get password

second part is credential stuffing

using a username/password found in one place
and trying to log into somewhere else with it

First part is a dictionary attack to get password

second part is credential stuffing

using a username/password found in one place
and trying to log into somewhere else with it

why they say use different passwords for different sites

First part is a dictionary attack to get password

second part is credential stuffing

using a username/password found in one place
and trying to log into somewhere else with it

why they say use different passwords for different sites

you password is as safe as the least competent place that stores it

Sarah Palin’s Email Hack

- reset password for gov.palin@yahoo.com
  - no secondary email needed
  - data of birth? (Wikipedia)
  - ZIP code? (Wasilla has 2)
  - where did you meet your spouse? (somewhere in Alaska?)
- changed password to “popcorn”
- Hacker sentenced to 1 year prison, 3 years supervision

Security Questions

- ideal: only Alice knows and tells Bob answer to
  - Bob can ask a question of Alice to authenticate
  - “What is your password” is essentially a security question
- in practice: terrible, doesn’t work at all, easily guessable

Security Questions Flaws

- inapplicable
  - what highschool did your spouse attend?
- not memorable
  - name of kindergarten teacher?
- ambiguous
  - name of university you applied to but did not attend?
- easily guessable
  - favourite colour?
- public record
  - mother’s maiden name?

Mother’s Maiden Name is a fact, not a secret.

Mother’s Maiden Name is a fact, not a secret.

A study found MMN for one fifth of Texans using only free public source of information.

Attached is your 2019 W-2. The password to open the file is your last 4SSN.

[redacted]

•••

Encrypted attachment warning – Be careful with this attachment. This

PDF

```
for i in range(0, 10000):
    s = ''
    if i < 10: s += '0'
    if i < 100: s += '0'
    if i < 1000: s += '0'
    s += str(i)
    print "echo " + s
    print "xpdf -upw " + s + " file.pdf"
```

So we have these widely used passwords that have certain flaws,
so what we can do?

Public-Key Authentication

- instead of giving a password, I prove knowledge of private key
    - but just giving the private key is bad
- assume Alice wants to connect to Bob's computer remotely
    - Bob knows Alice's public key
    - Bob wants Alice to connect, but needs to know it is Alice
- how can this work?

Public-Key Authentication

- Bob issues Alice a challenge
  - some message that Alice needs to sign
  - Alice signs the message and gives signature to Bob
  - Alice must therefore have this key

What can go wrong?

Replay attack

- Eve monitors all communication
- Bob reuses challenges
- Eve already has the answer and provides it

Solution: don’t reuse challenges!

Solution: don’t reuse challenges!

include timestamps and random numbers

(DEFENCE IN DEPTH)

Mafia Fraud

- Alice connects to Eve willingly
  - Eve runs some Website (like an illegal downloading site)
- Eve then connects to Bob pretending to be Alice
- Bob issues “Alice” (really Eve) a challenge to sign
- Eve uses that as a challenge for Alice pretending it’s Eve’s Website

This is also called the chess grandmaster’s problem:

how was a young girl named Anne-Louise
able to defeat a grandmaster in chess?

This is also called the chess grandmaster's problem:

how was a young girl named Anne-Louise
able to defeat a grandmaster in chess?

So why is it called Mafia Fraud and not MitM attack?

Mafia fraud / grandmaster problem the victim
willingly and knowingly communicates
with the attacker and
unknowingly communicates with the other victim.

Mafia fraud / grandmaster problem the victim willingly and knowingly communicates with the attacker and unknowingly communicates with the other victim.

MitM involves the victims unknowingly communicating with the attacker.

Phone Code Authentication

- assume that people carry their phone
    - something they have
- when logging in, send a text to their phone
    - text message has a code like GKDFTM
- user enters GKDFTM to continue logging in

What is wrong with:

YOUR LOGIN CODE IS: 24

What is wrong with:

YOUR LOGIN CODE IS:

bFdhb8mnYtLuab/pq0mL+vzZ0stZE/S8X9H6nx
IcHpW8um0k8MFMLrNk7kp4js5eCBEU1pglL1qS
eFGMwVo6abagD6uE0Ishb9FFEd0iNi6MxJg1t2
xaysN64vR8+o8zMWmk6RMGU1ashX/hNViKrvsR
bGWj6n0MzZ8ToseJtF34zHbegJTX6IfgnSZaja
cEd2HKbnPnSTdKxjxXDz8tP//B18+Jbb/ySJ4

What is wrong with:

YOUR LOGIN CODE IS: 52373798

What is wrong with:

YOUR LOGIN CODE IS: 52373798

no information about who is sending this number

potential mafia fraud

One-Time Password Authentication

- Bob issues Alice a sheet of passwords
- each can be used once
- Bob asks for a particular one or accept any of the list
  - problem here with accepting any?

Symmetric Token

- Alice and Bob both have a shared key K
- at time T, Alice uses $E_k(T|A|B)$ as password
- if a hardware token then noticed when missing
    - something you have

These non-password authentications can be as a second factor

Two-Factor Authentications (2FA)

- in addition to a password, use a second factor too
  - e.g., token, list, question, phone code
- can be everytime or under certain circumstances
  - unusual activity
    - e.g., logging in from another country
    - e.g., outside of work hours
  - extraordinary actions
    - e.g., access HR records, tax information, etc.
    - e.g., perform a stock trade
    - e.g., buy something (amazon)

Password Security

- 2FA is important because passwords aren’t enough
  - in 2012, 76% of network intrusions exploited weak or stolen credentials
  - keystroke loggers
  - shoulder surfing
  - same passwords at multiple places
- so demand 2FA!