CS 88: Security and Privacy

14: Authentication

03-19-2023

slides courtesy Christo Wilson, Vitaly Shmatikov

SWARTHMORE COLLEGE


Microsoft Windows xp
To begin, click your user name
Administrator
Type your password
Turn off computer
After you log on, you can add or change accounts.
Just go to Control Panel and click User Accounts.

Authentication

- Authentication is the process of verifying an actor's identity
- Critical for security of systems
  - Permissions, capabilities, and access control are all contingent upon knowing the identity of the actor
- Typically parameterized as a username and a secret
  - The secret attempts to limit unauthorized access
- Desirable properties of secrets include being *unforgeable*, *unguessable*, and *revocable*

Types of Secrets

- Actors provide their secret to log-in to a system
- Three classes of secrets:
  1. Something you know
     - Example: a password
  2. Something you are
     - Examples: fingerprint, voice scan, iris scan
  3. Somewhere you are
     - IP address, geolocation
  4. Something you have
     - Examples: a smart card or smart phone, 2FA

Today’s Topics

- Password Storage
  - How should you securely store password to prevent cracking?
  - Are there ways to help detect password breaches?
- Password Cracking
  - Basic attacks: brute forcing and dictionary
  - Hash chains
  - Rainbow tables
- Local and Distributed Authentication Systems
  - Unix/Linux PAM system
  - NIS
  - Needham-Schroeder
  - Kerberos

Password Storage

Hashing and Salting

Key Stretching and Work Factor

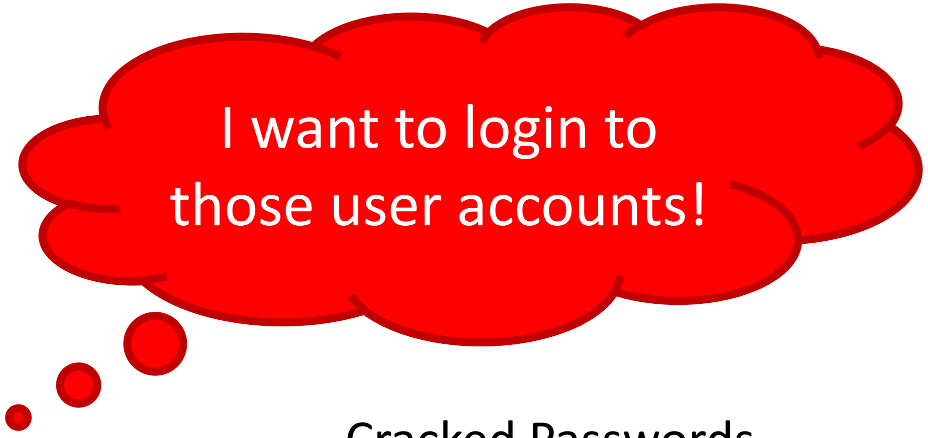
Honeywords

Attacker Goals and Threat Model

- Assume we have a system storing usernames and passwords
- The attacker has access to the password database/file
- Our goal: even if the database is stolen, attacker should learn as little as possible about the passwords.

Database

User	Password
cbw	p4ssW0rd
sandi	puppies
amislove	3spr3ss0



Cracked Passwords

User	Password
cbw	p4ssW0rd
sandi	puppies
amislove	3spr3ss0

Checking Passwords

- System must validate passwords provided by users
- Thus, passwords must be stored somewhere
- Basic storage: plain text

password.txt	
cbw	p4ssw0rd
sandi	i heart doggies
amislove	93Gd9#jv*0x3N
bob	security

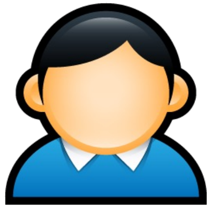
Problem: Password File Theft

- Attackers often compromise systems
- They may be able to steal the password file
  - Linux: /etc/shadow
  - Windows: c:\windows\system32\config\sam
- If the passwords are plain text, what happens?
  - The attacker can now log-in as any user, including root/administrator
- **Passwords should never be stored in plain text**

Hashed Passwords

- Key idea: store hashed versions of passwords
  - Use one-way cryptographic hash functions
  - Examples: MD5, SHA1, SHA256, SHA512, bcrypt, PBKDF2, scrypt
- Cryptographic hash function transform input data into scrambled output data
  - Deterministic: hash(A) = hash(A)
  - High entropy:
    - MD5(‘security’) = e91e6348157868de9dd8b25c81aebfb9
    - MD5(‘security1’) = 8632c375e9eba096df51844a5a43ae93
    - MD5(‘Security’) = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A’ such that hash(A) = hash(A’) takes a long time (hopefully)
    - Example: 2^{21} tries for md5

Hashed Password Example



User: cbw

MD5(‘p4ssw0rd’) =
2a9d119df47ff993b662a8ef36f9ea20

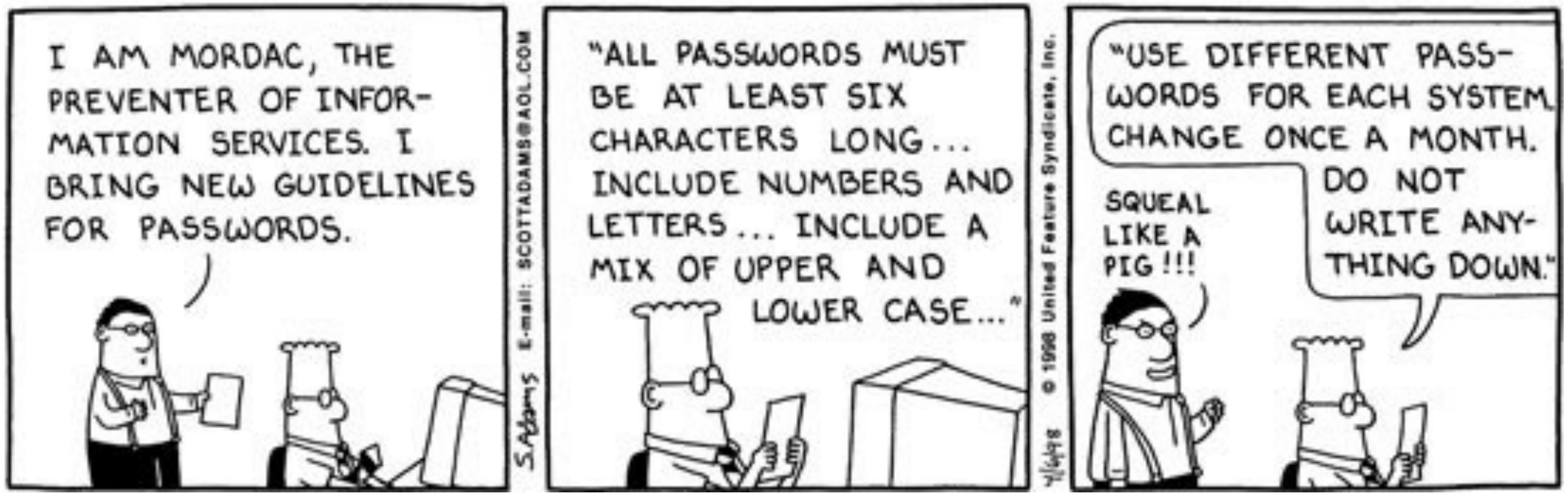
MD5(‘2a9d119df47ff993b662a8ef36f9ea20’)
= b35596ed3f0d5134739292faa04f7ca3

hashed_passw[illegible]xt	
cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Discussion Question: Are hashed password secure from cracking?

A. Yes (discuss why/under what circumstances)
B. No (discuss why/under what circumstances)

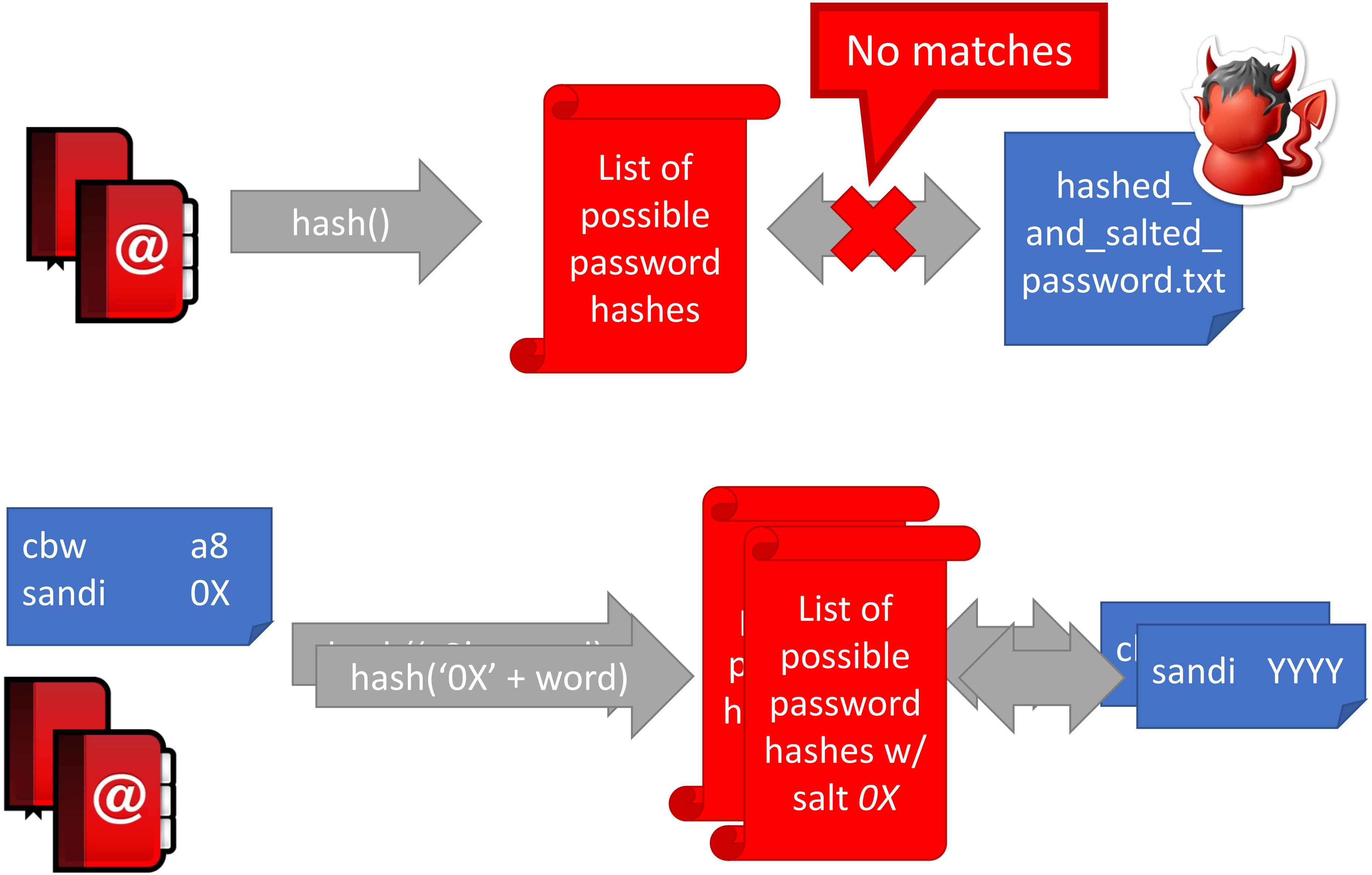
Password Usability



Discussion Question: Hardening Password Hashes

- Key problem: cryptographic hashes are deterministic
  - hash(‘p4ssw0rd’) = hash(‘p4ssw0rd’)
  - This enables attackers to build lists of hashes
- What might we do to make the passwords we store be “unique”?
  - HINT: analogous problem to symmetric key encryption and block ciphers

Attacking Salted Passwords



Breaking Hashed Passwords

- **Stored passwords should always be salted**
  - Forces the attacker to brute-force each password individually
- Problem: it is now possible to compute hashes very quickly
  - GPU computing: hundreds of small CPU cores
  - nVidia GeForce GTX Titan Z: 5,760 cores
  - GPUs can be rented from the cloud very cheaply
    - $0.2 per hour (2024 prices)

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours (assuming md5 hashing)
  - Upper and lowercase letters, numbers, symbols
  - $(26+26+10+32)^6$ = 690 billion combinations
- A modern GPU can do the same thing in 16 minutes
- Most users use (slightly permuted) dictionary words, no symbols
  - Predictability makes cracking much faster
  - Lowercase + numbers → $(26+10)^6$ = 2 billion combinations
  - less than a minute!

Hardening Salted Passwords

- Problem: typical hashing algorithms are too fast
  - Enables GPUs to brute-force passwords
- Old solution: hash the password multiple times
  - Known as key stretching
  - Example: *crypt* used 25 rounds of DES
- New solution: use hash functions that are designed to be **slow**
  - Examples: bcrypt, PBKDF2, scrypt
  - These algorithms include a work factor that increases the time complexity of the calculation
  - scrypt also requires a large amount of memory to compute, further complicating brute-force attacks (CPU + memory bottlenecked)

bcrypt Example

- Python example; install the *bcrypt* package

```
[cbw@localhost ~] python
>>> import bcrypt
>>> password = “my super secret password”
>>> fast_hashed = bcrypt.hashpw(password, bcrypt.gensalt(0))
>>> slow_hashed = bcrypt.hashpw(password, bcrypt.gensalt(12))
>>> pw_from_user = raw_input(“Enter your password:”)
>>> if bcrypt.hashpw(pw_from_user, slow_hashed) == slow_hashed:
…       print “It matches! You may enter the system”
…   else:
…       print “No match. You may not proceed”
```

Work factor

Discussion Question: Dealing With Breaches

- Suppose you build an extremely secure password storage system
  - All passwords are salted and hashed by a high-work factor function
- Is it still possible for a dedicated attacker to steal and crack passwords?

A. Yes

B. No

C. Maybe (be prepared to explain each choice ☺)

Password Storage Summary

1. Never store passwords in plain text
2. Always salt and hash passwords before storing them
3. Use hash functions with a high work factor

- These rules apply to any system that needs to authenticate users
  - Operating systems, websites, etc.

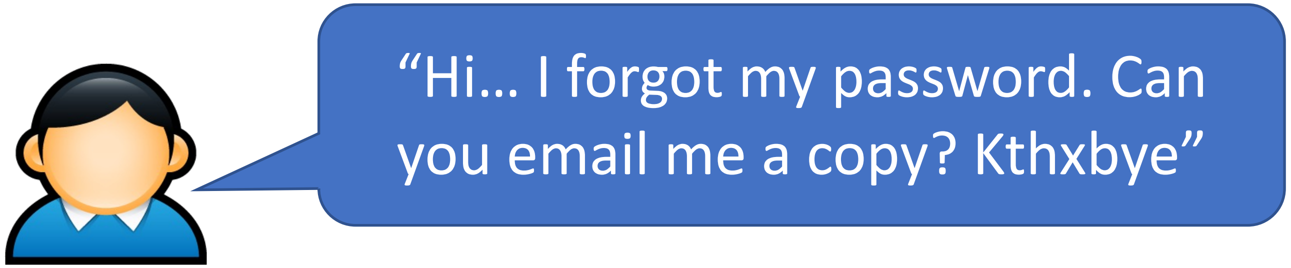
Password Recovery and Reset

Password Reuse

- People have difficulty remembering >4 passwords
  - Thus, people tend to reuse passwords across services
  - What happens if any one of these services is compromised?
- Service-specific passwords are a beneficial form of compartmentalization
  - Limits the damage when one service is inevitably breaches
- Some service providers now check for password reuse
  - Forbid users from selecting passwords that have appeared in leaks

Password Recovery/Reset

- Problem: hashed passwords cannot be recovered (hopefully)



- This is why systems typically implement password reset
  - Use out-of-band info to authenticate the user
  - Overwrite hash(old_pw) with hash(new_pw)
- Be careful: its possible to crack password reset

Knowledge-based Rest

- Typical implementations use Knowledge Based Authentication (KBA)
    - What was your mother's maiden name?
    - What was your prior street address?
    - Where did you go to elementary school
- Problems?
    - This information is widely available to anyone
    - Publicly accessible social network profiles
    - Background-check services like Spokeo
- Experts recommend that services not use KBA
    - When asked, users should generate random answers to these questions

HealthCare.gov

<u>Federal:</u>

What is a relative's telephone number that is not your own?
Type a significant date in your life?
What is the name of the manager at your first job?

<u>Individual states:</u>

What is your youngest child's birth weight?
What color was your first bicycle?
If you needed a new first name, what would it be?
What band poster did you have on your wall in high school?
How many bones have you broken?

Account-based Reset

- Idea: authenticate a user by sending a code to their contact address
  - Typically e-mail address or phone number
- Security rests on the assumption that the person’s contact address is also secure
  - E-mail account takeover
  - SIM hijacking

Challenges of Password Reset

- Password reset mechanisms are often targeted and are quite vulnerable
- Best practice: implement a layered mechanism
  - Knowledge-based
  - Secondary account
  - Second factor authentication: biometric or tokens
- Warning: more secure = less usable
  - Password loss is common, people will be frustrated by onerous reset mechanisms

Choosing Passwords

Bad Algorithms

Better Heuristics

Password Reuse

Mental Algorithms

Years of security advice have trained people to generate "secure" passwords

1. Pick a word
2. Capitalize the first or last letter
3. Add a number (and maybe a symbol) to the beginning or end

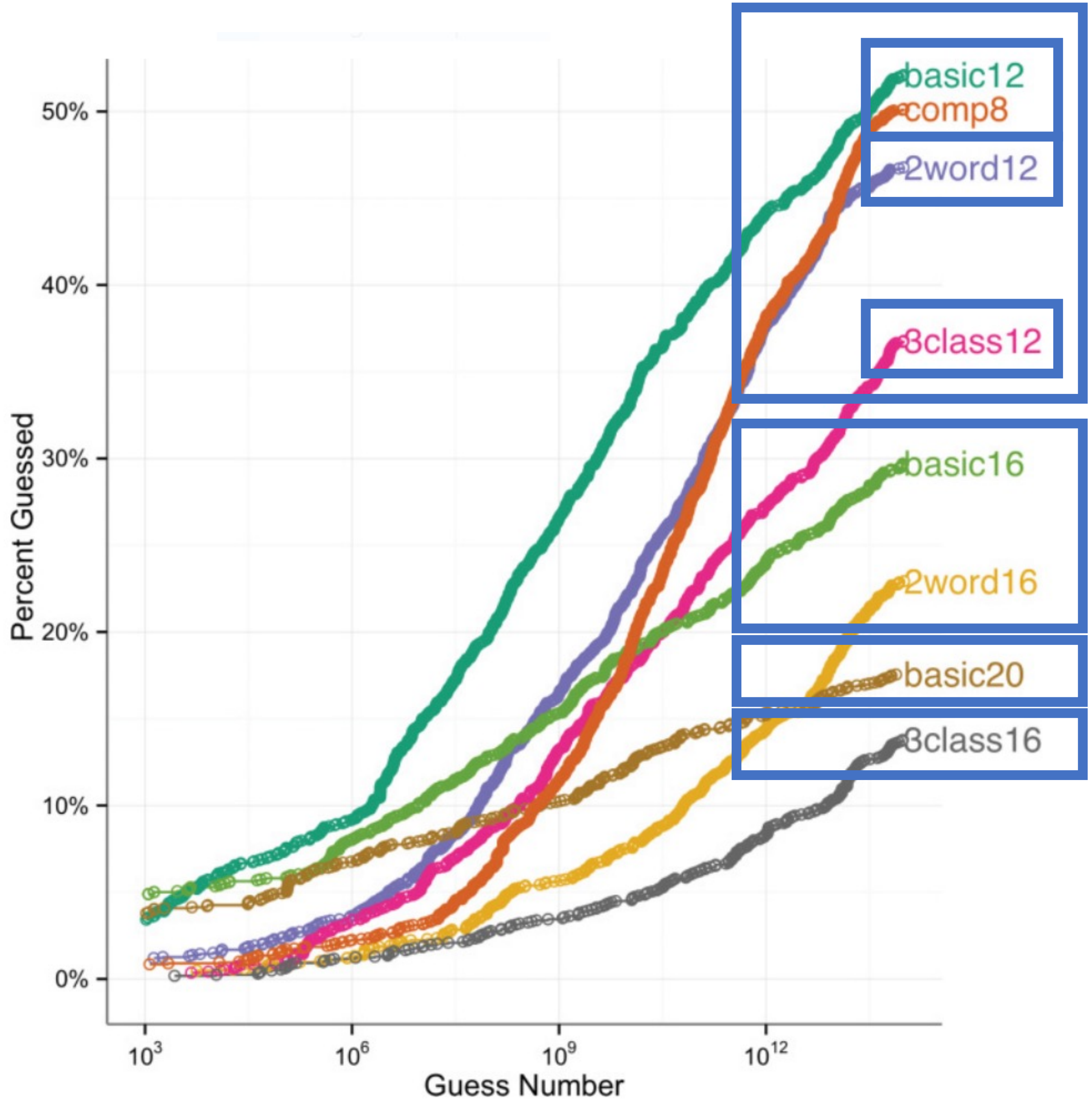
1. Pick a word
2. Replace some of the letters with symbols (a → @, s → $, etc.)
3. Maybe capitalize the first or last letter

Human Generated Passwords

Password
Computer3@
cl4ssr00m
7Dogsled*
Tjw1989&6
B4nk0f4m3r1c4!

Modern attackers are sophisticated

- No need for brute force cracking!
- Use dictionaries containing common words and passwords from prior leaks
- Apply common "mental" permutations



Password Requirements

- comp *n* and basic *n*: use at least *n* characters
- *k* word *n*: combine at least *k* words using at least *n* characters
- *d* class *n*: use at least *d* character types (upper, lower, digit, symbol) with at least *n* characters

Plot from Shay et al.
https://www.blaseur.com/papers/tissec_1026.pdf

Better Heuristics

Notice that in $S = L * log_2 N$, length matters more than symbol types

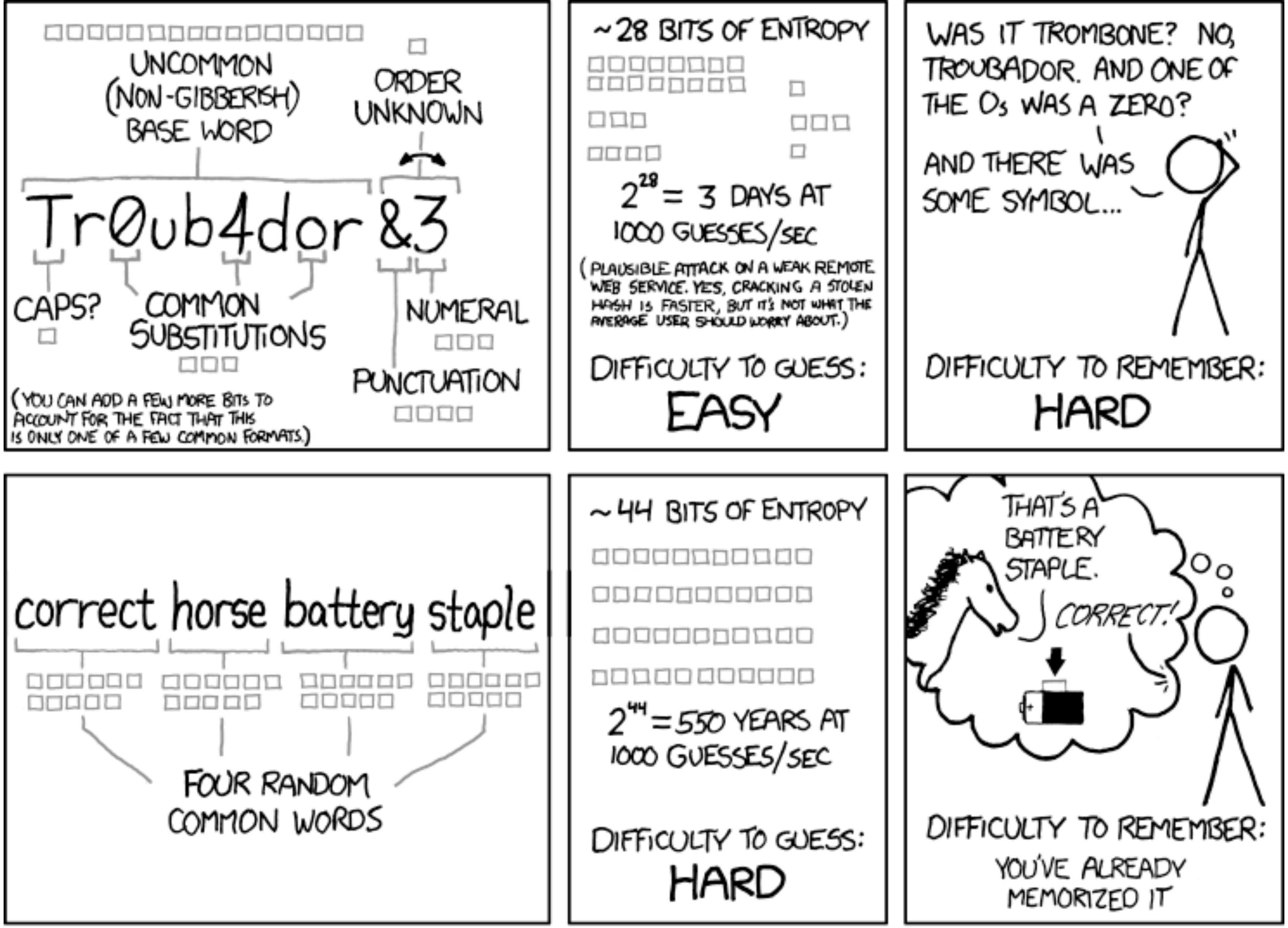
- Choose longer passwords (16+ characters)
- Don't worry about uppercase, digits, or symbols

Use mnemonics

- Choose a sentence or phrase
- Reduce it to the first letter of each word
- Insert random uppercase, digits, and symbols

I double dare you, say "what" one more time

i2Dy,s"w"omt



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.

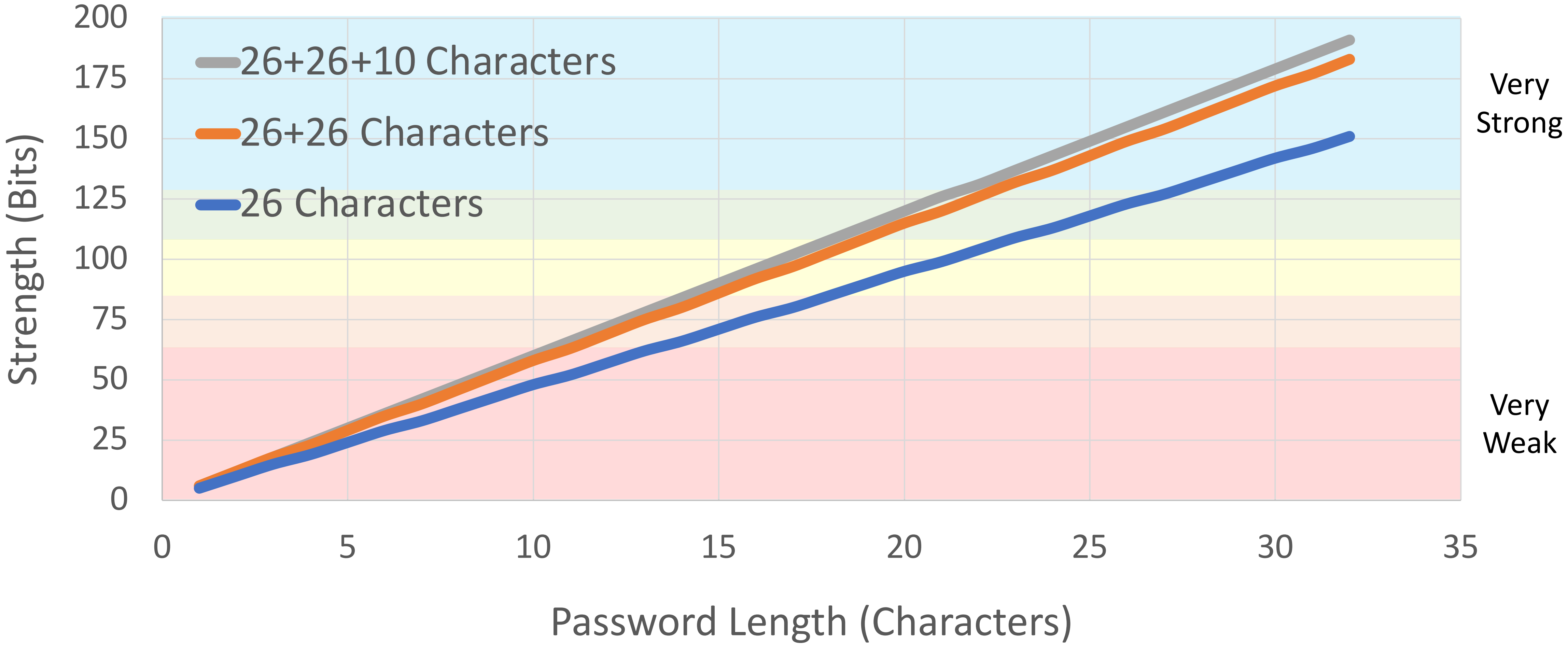
Password Quality

$$S = \log_2 N^L \rightarrow L = \frac{S}{\log_2 N}$$

- How do we measure password quality? Entropy
  - *N* – the number of possible symbols (e.g. lowercase, uppercase, numbers, etc.)
  - *L* – the length of the password
  - *S* – the strength of the password, in bits
- Formula tells you length *L* needed to achieve a desired strength S…
  - … for **randomly generated** passwords
- Is this a realistic measure in practice?

The Strength of Random Passwords

$$S = L * log_2 N$$



Password Cracking

Password Theory

Hash Chains

Rainbow Tables

Attacker Goals and Threat Model

Assume we have a system storing usernames and passwords

The attacker has access to the password database/file

Database

User	H(PW)
cbw	iuafNas
sandi	23asZR
amislove	9xgGw/

I want to login to those user accounts!

Cracked Passwords

User	Password
cbw	p4ssW0rd
sandi	puppies
amislove	3spr3ss0

Basic Password Cracking

Problem: humans are terrible at generating/remembering random strings

Passwords are often weak enough to be brute-forced

- Naïve way: systematically try all possible passwords
- Slightly smarter way: take into account non-uniform distribution of characters

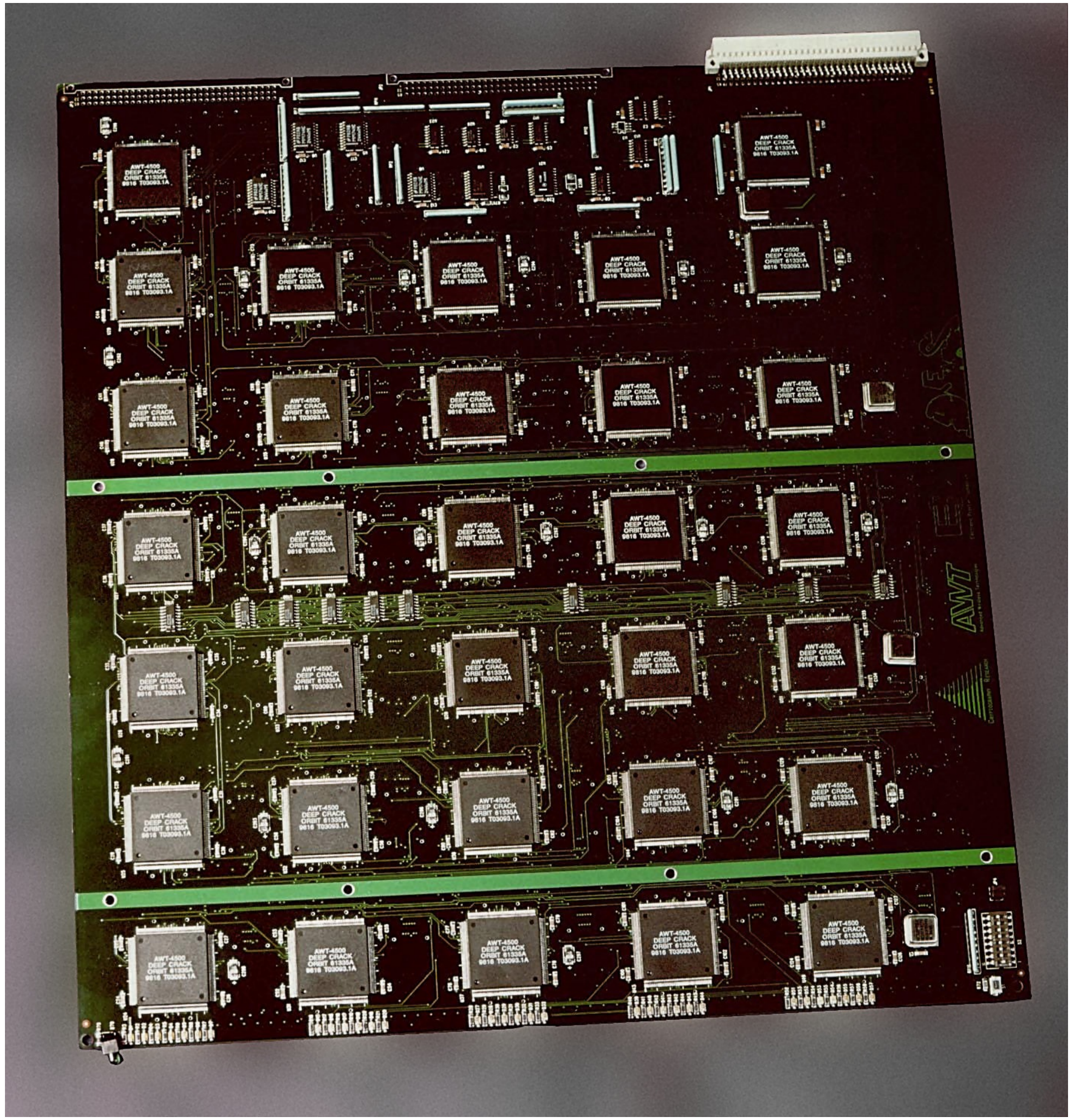
Dictionary attacks are also highly effective

- Select a baseline wordlist/dictionary full of **likely** passwords
  - Today, the best wordlists come from lists of breached passwords
- Rule-guided word mangling to look for slight variations
  - E.g. password → Password → p4ssword → passw0rd → p4ssw0rd → password1 → etc.

Many password cracking tools exist (e.g. John the Ripper, hashcat)

“Deep Crack”: The Electronic Frontier Foundation DES Cracker

- DES uses a 56-bit key
- $250K in 1998, capable of brute-forcing DES keys in 56 hours
  - Uses 1856 custom ASIC chips
  - Similar attacks have been demonstrated against MD5, SHA1
- Modern equivalent?
  - Bitcoin mining ASICs



Speeding Up Brute-Force Cracking

Brute force attacks are slow because hashing is CPU intensive

- Especially if a strong function (SHA512, bcrypt) is used

Idea: why not pre-compute and store all hashes?

- You would only need to pay the CPU cost once…
- … for a given salt

Given a hash function *H*, a target hash *h*, and password space *P*, goal is to recover $p \in P$ such that $H(p) = h$

Problem: naïve approach requires $\Theta(|P|n)$ bits, where *n* is the space of the output of *H*

Hash Chains

Hash chains enable time-space efficient reversal of hash functions

Key idea: pre-compute chains of passwords of length *k*…

- … but only store the start and end of each chain
- Larger *k* → fewer chains to store, more CPU cost to rebuild chains
- Small *k* → more chains to store, less CPU cost to rebuild chains

Building chains require *H*, as well as a reduction *R : H* $\mapsto$ *P*

- Begin by selecting some initial set of password $P' \subset P$
- For each $p' \in P$', apply $H(p') = h', R(h') = p''$ for *k* iterations
- Only store p' and p'^k

To recover hash *h*, apply R and H until the end of a chain is found

- Rebuild the chain using p' and p'^k
- *H(p) = h* may be within the chain

Uncompressed Hash Chain Example

Link 1 | Link 2 | Link 3

p'
abcde
passw
12345
secrt

K = 3

Only these two columns get stored on disk

Hash Chain Example

p'	p*
abcde	cjldar
passw	zvxscs
12345	wjizbn
secrt	eivlqc

K = 3

- Size of the table is dramatically reduced…
- … but some computation is necessary once a match is found

p'	H(p’) = h’	R(h’) = p”	H(p”) = h”	R(h”) = p’’’	H(p’’’) = h’’’	R(h’’’) = p*
12345						wjizbn

p	H(p) = h	R(h) = p’	H(p’) = h’	R(h’) = p”	H(p”) = h”	R(h”) = p’’’	H(p’’’) = h’’’
	RHKP_D						

Desired password

Hash to recover

Problems with Hash Chains

Hash chains are prone to collisions

- Collisions occur when *H(p')* = *H(p")* or *R(h')* = *R(h")* (the latter is more likely)
- Causes the chains to merge or overlap

Problems caused by collisions

- Wasted space in the file, since the chains cover the same password space
- False positives: a chain may not include the password even if the end matches

Proper choice of *R()* is critical

- Goal is to cover ***likely*** password space, not entire password space
- *R* cannot be collision resistant (like *H*) since it has to map into likely plaintexts
- Difficult to select *R* under this criterion

Rainbow Tables

Rainbow tables improve on hash chains by reducing the likelihood of collisions

Key idea: instead of using a single reduction *R*, use a family of reductions $\{R_1, R_2, \ldots, R_k\}$

- Usage of *H* is the same as for hash chains
- A collisions can only occur between two chains if it happens at the same position (e.g. R_i in both chains)

Final Thoughts on Rainbow Tables

Caveats

- Tables must be built for each hash function and character set
- Salting and key stretching defeat rainbow tables

Rainbow tables are effective in some cases, e.g. MD5 and NTLM

- Precomputed tables can be bought or downloaded for free

Password Management

Password Reuse

People have difficulty remembering >4 passwords

- Thus, people tend to reuse passwords across services
- What happens if any one of these services is compromised?

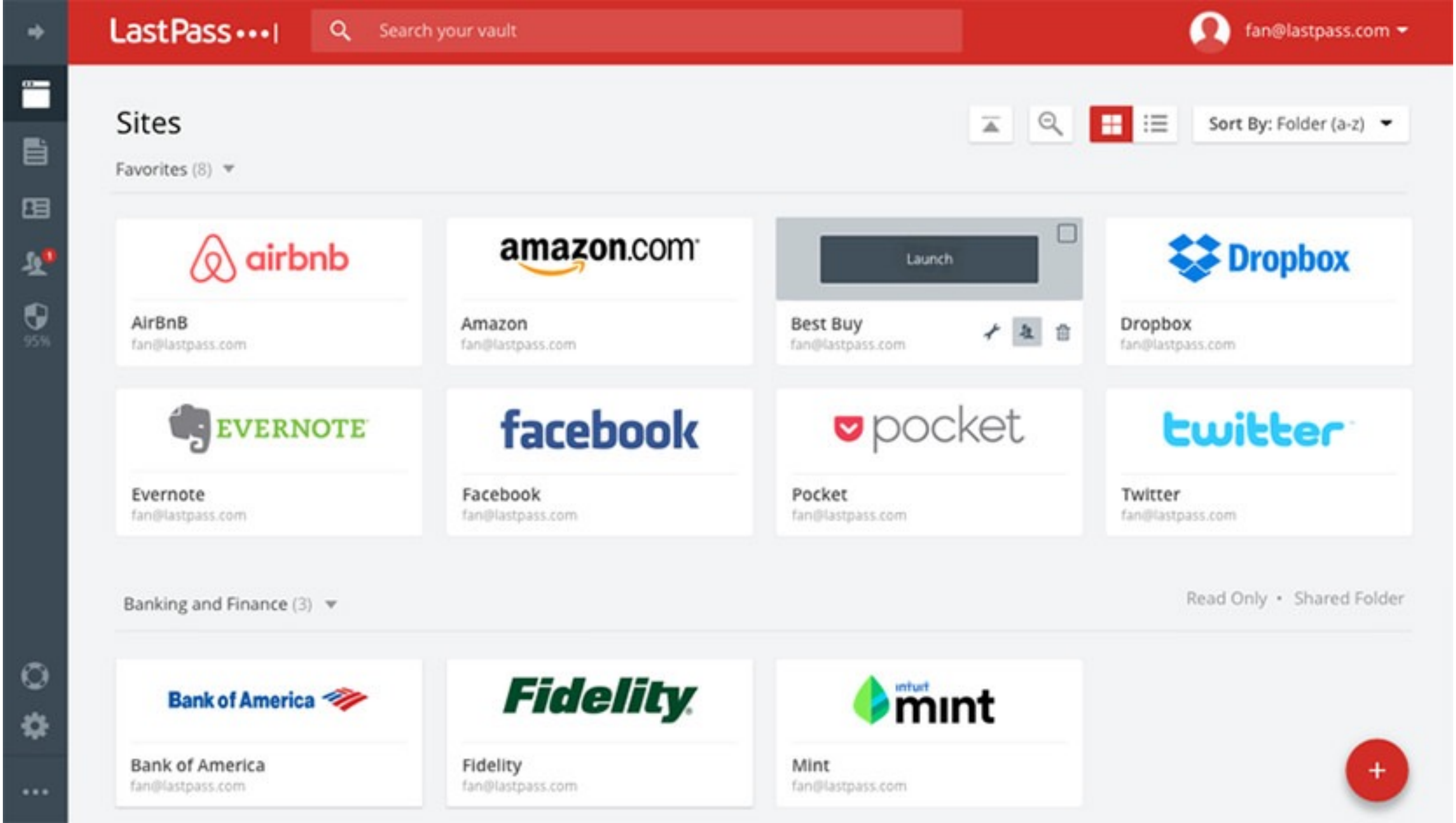
Service-specific passwords are a beneficial form of compartmentalization

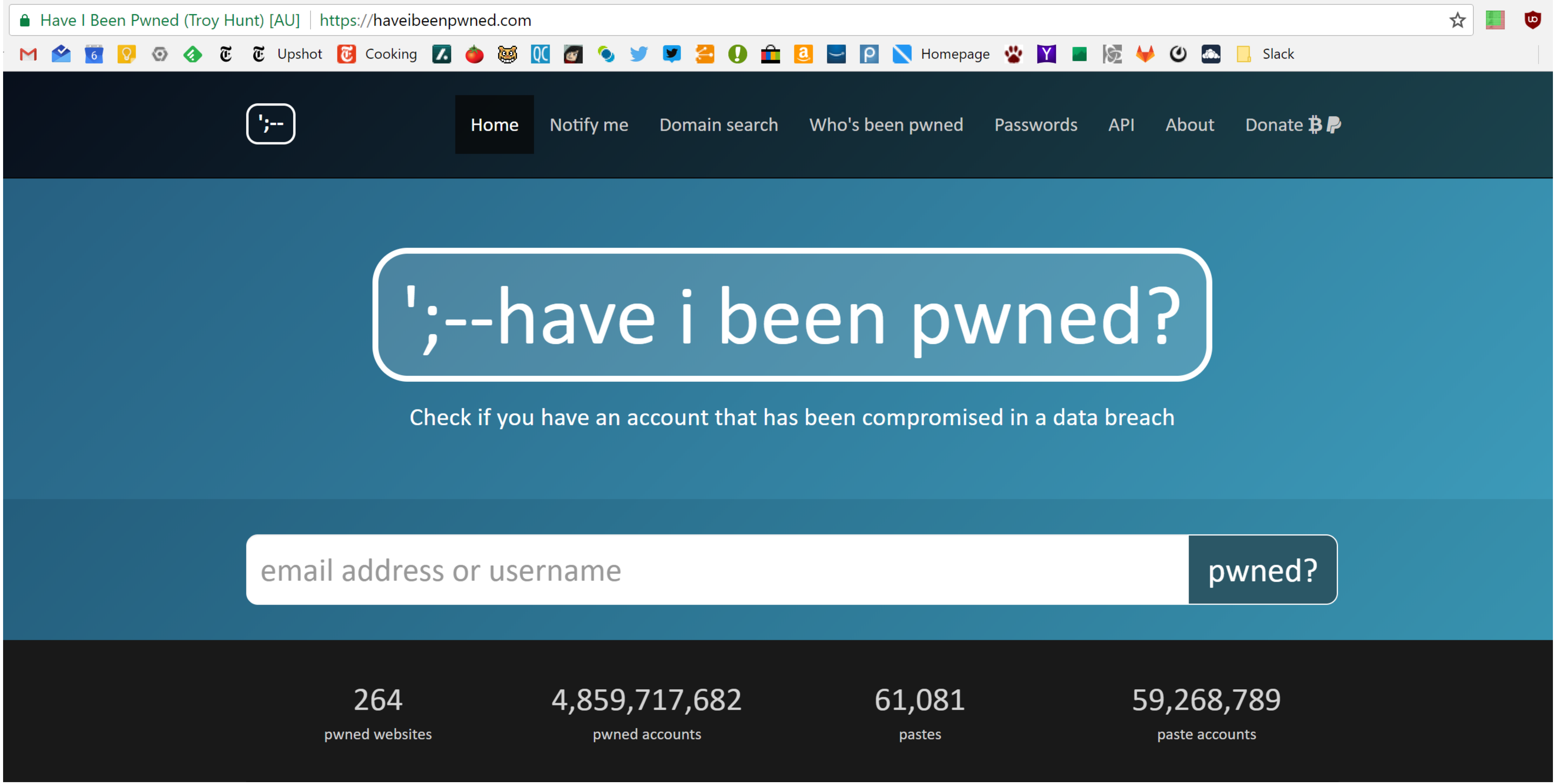
- Limits the damage when one service is inevitably breaches

Use a password manager

Some service providers now check for password reuse

- Forbid users from selecting passwords that have appeared in leaks


LastPass
Search your vault
fan@lastpass.com
Sites
Sort By: Folder (a-z)
Favorites (8)
AirBnB
fan@lastpass.com
Amazon
fan@lastpass.com
Launch
Best Buy
fan@lastpass.com
Dropbox
fan@lastpass.com
Evernote
fan@lastpass.com
Facebook
fan@lastpass.com
Pocket
fan@lastpass.com
Twitter
fan@lastpass.com
Banking and Finance (3)
Read Only • Shared Folder
Bank of America
fan@lastpass.com
Fidelity
fan@lastpass.com
Mint
fan@lastpass.com
95%



Have I Been Pwned (Troy Hunt) [AU] | https://haveibeenpwned.com
Upshot
Cooking
Homepage
Slack
';--
Home
Notify me
Domain search
Who's been pwned
Passwords
API
About
Donate
';--have i been pwned?
Check if you have an account that has been compromised in a data breach
email address or username
pwned?
264
pwned websites
4,859,717,682
pwned accounts
61,081
pastes
59,268,789
paste accounts

Biometric Two Factor Authentication

Biometrics

SMS

Authentication Codes

Smartcards & Hardware Tokens

Types of Secrets

Actors provide their secret to log-in to a system

Three classes of secrets:

1. Something you know
   - Example: a password
2. Something you are
   - Examples: fingerprint, voice scan, iris scan
3. Something you have
   - Examples: a smart card or smart phone

Biometrics

ancient Greek: bios ="life", metron ="measure“

Physical features

- Fingerprints
- Face recognition
- Retinal and iris scans
- Hand geometry

Behavioral characteristics

- Handwriting recognition
- Voice recognition
- Typing cadence
- Gait

Biometric Authentication

- Nothing to remember
- Passive
  - Nothing to type, no devices to carry around
- Can’t share (usually)
- Can be fairly unique
  - … if measurements are sufficiently accurate

Identification vs. Authentication

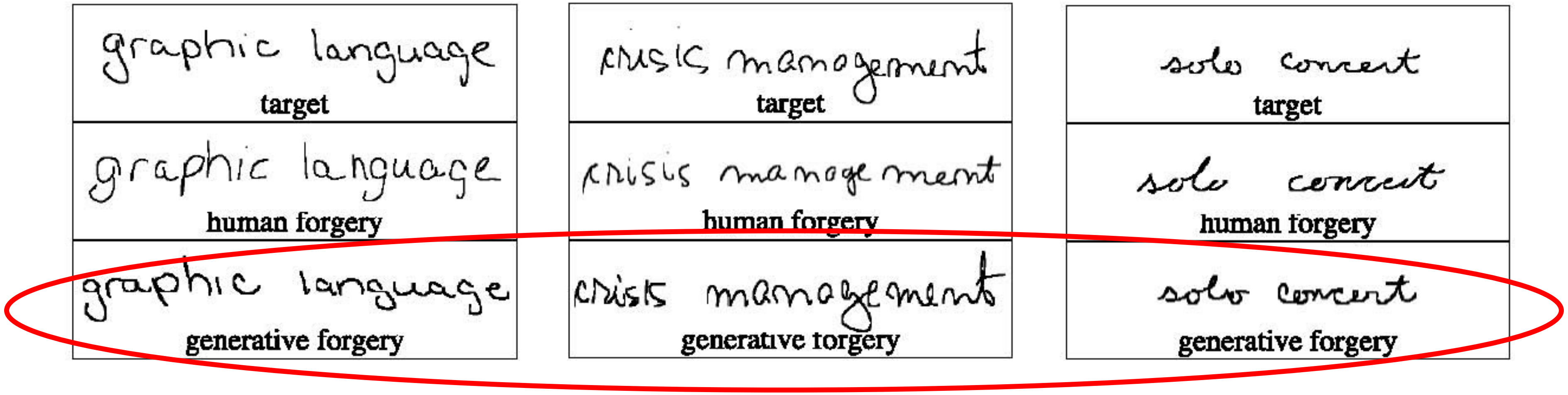
- Goal: associate an identity with an event
  - Example: a fingerprint at a crime scene
  - Key question: given a particular biometric reading, does there exist another person who has the same value of this biometric?
- Goal: verify a claimed identity
  - Example: fingerprint scanner to enter a building
  - Key question: do there exist any two persons who have the same value of this biometric?
    - Birthday paradox!

Problems with Biometrics

- Private, but not secret
  - Biometric passports, fingerprints and DNA on objects…
- Even random-looking biometrics may not be sufficiently unique for authentication
  - Birthday paradox!
- Potentially forgeable
- Revocation is difficult or impossible

Forging Handwriting

[Ballard, Monrose, Lopresti]



Generated by computer algorithm trained on handwriting samples

Biometric Error Rates (Benign)

- “Fraud rate” vs. “insult rate”
  - Fraud = system accepts a forgery (false accept)
  - Insult = system rejects valid user (false reject)
- Increasing acceptance threshold increases fraud rate, decreases insult rate
- For biometrics, U.K. banks set target fraud rate of 1%, insult rate of 0.01% [Ross Anderson]
  - Common signature recognition systems achieve equal error rates around 1% - not good enough!

Biometrics

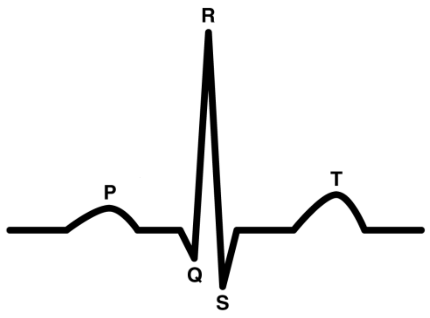
- Face recognition (by a computer algorithm)
  - Error rates up to 20%, given reasonable variations in lighting, viewpoint and expression
- Fingerprints
  - Traditional method for identification
  - 1911: first US conviction on fingerprint evidence
  - U.K. traditionally requires 16-point match
    - Probability of a false match is 1 in 10 billion
    - No successful challenges until 2000
  - Fingerprint damage impairs recognition
    - Ross Anderson’s scar crashes FBI scanner

Biometrics

- Iris scanning
  - Irises are very random, but stable through life
    - Different between the two eyes of the same individual
  - 256-byte iris code based on concentric rings between the pupil and the outside of the iris
  - Equal error rate better than 1 in a million
- Hand geometry
  - Used in nuclear premises entry control, INSPASS (discontinued in 2002)
- Voice, ear shape, vein pattern, face temperature

Biometrics

Identifies wearer by his/her unique heartbeat pattern



Biometrics


I AM MORDAC, THE PREVENTER OF INFOR-MATION SERVICES, AND I BRING YOU MY NEWEST BIOMETRIC SCANNER.
scottadams@aol.com
www.dilbert.com
INSTEAD OF A PASS-WORD, I PUT THIS ON YOUR HEAD AND SQUEEZE UNTIL YOU SCREAM IN A WAY THAT ONLY YOU CAN SCREAM.
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.
NO, THAT'S NOT YOU.
GAAA! GAAA! GAAA!

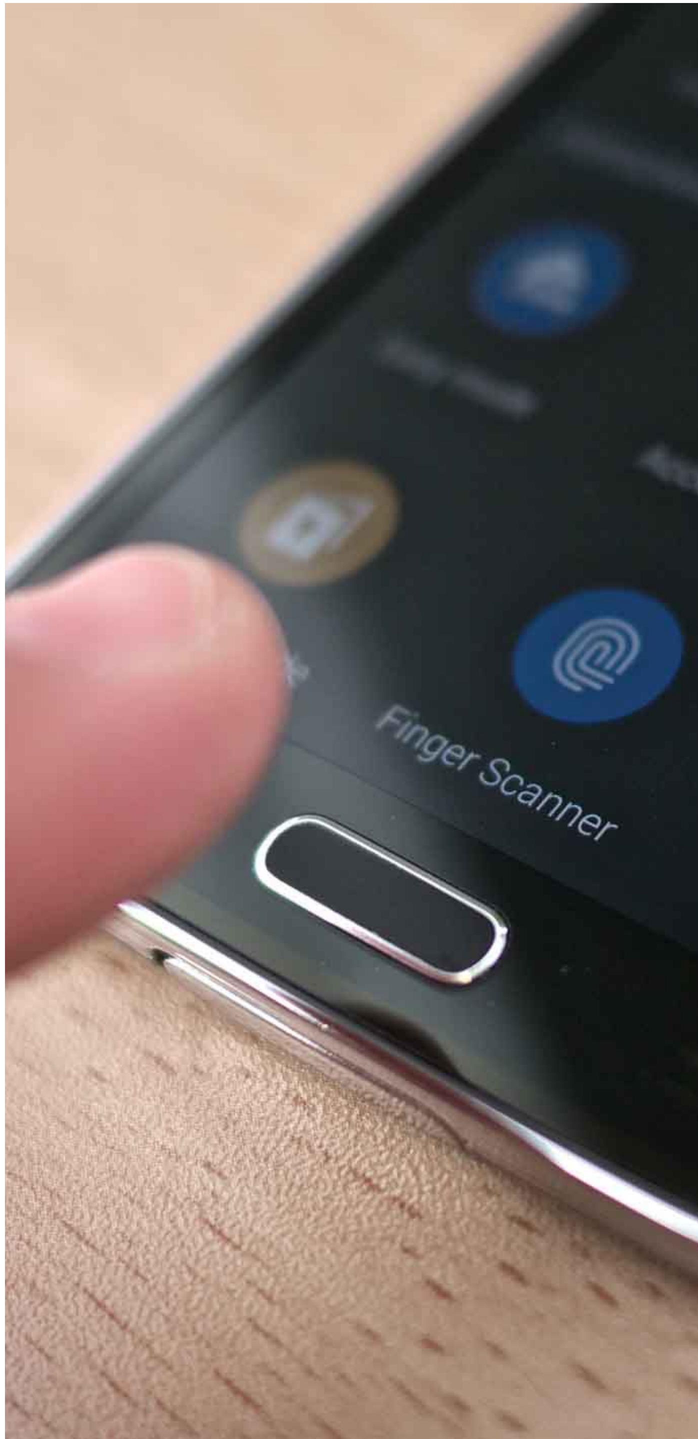
Fingerprints

Ubiquitous on modern smartphones, some laptops

Secure?

- May be subpoenaed by law enforcement
- Relatively easy to compromise
    1. Pick up a latent fingerprint (e.g. off a glass) using tape or glue
    2. Photograph and enhance the fingerprint
    3. Etch the print into gelatin backed by a conductor
    4. Profit ;)

https://www.theregister.co.uk/2002/05/16/gummi_bears_defeat_fingerprint_sensors/



Facial Recognition

Popularized by FaceID on the iPhone X

Secure?

- It depends

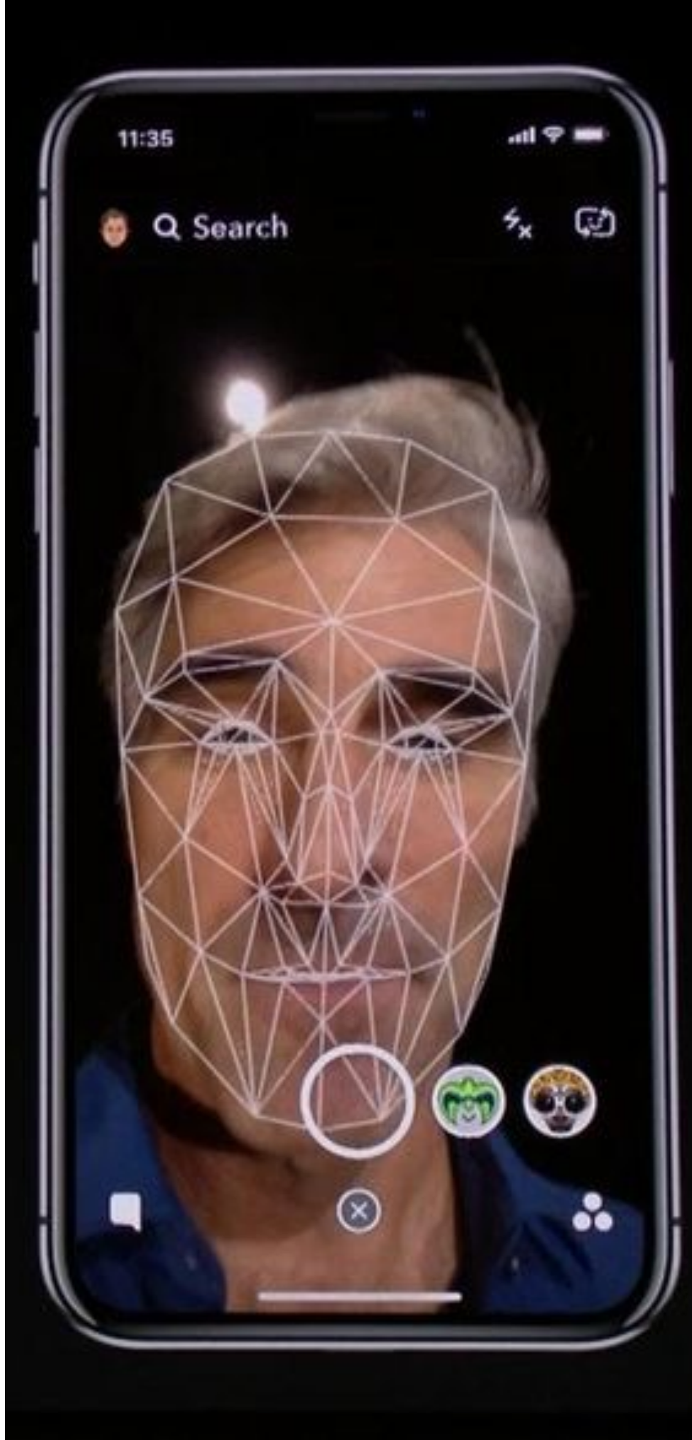
Vulnerable to law enforcement requests

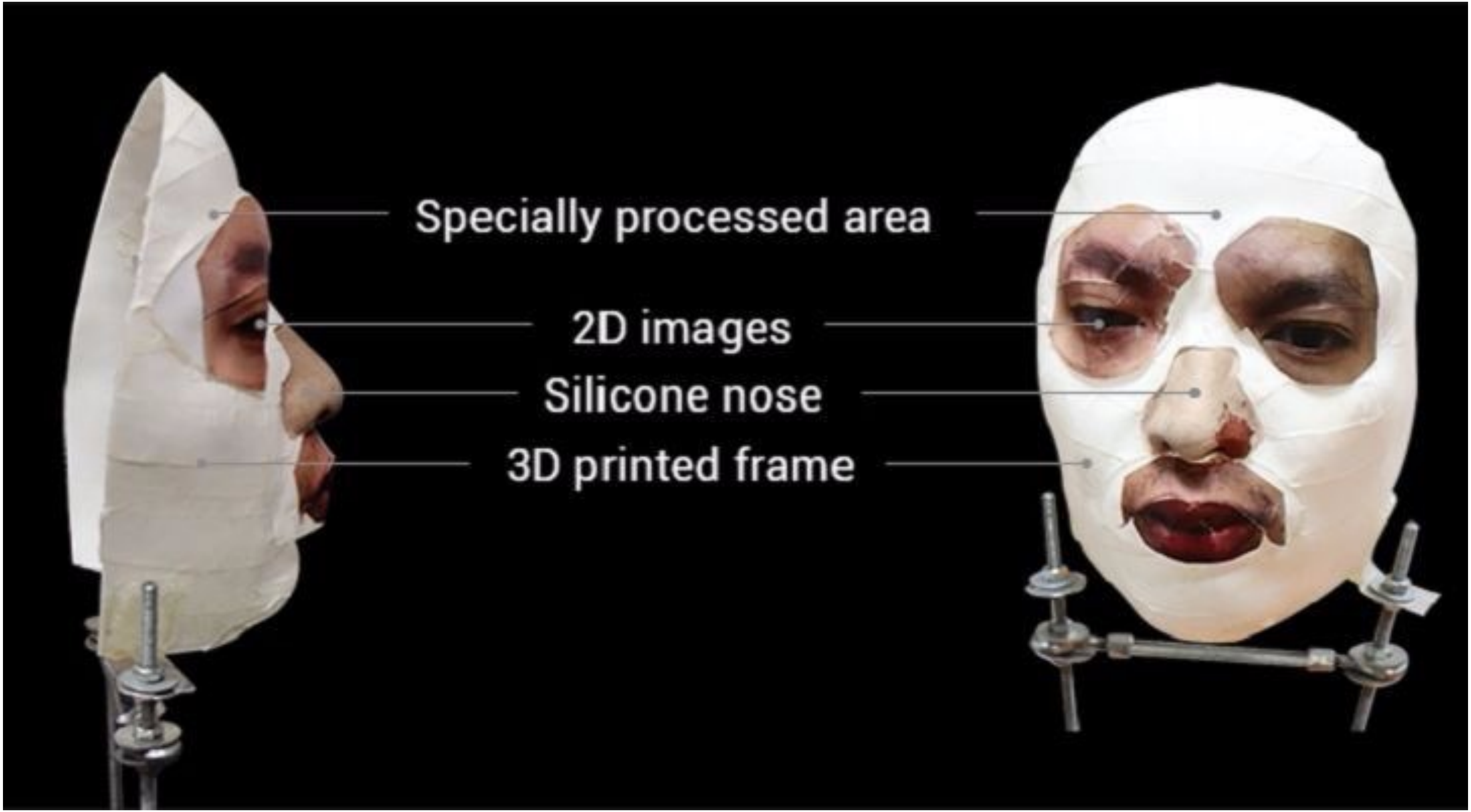
Using 2D images?

- Not secure
- Trivial to break with a photo of the target's face

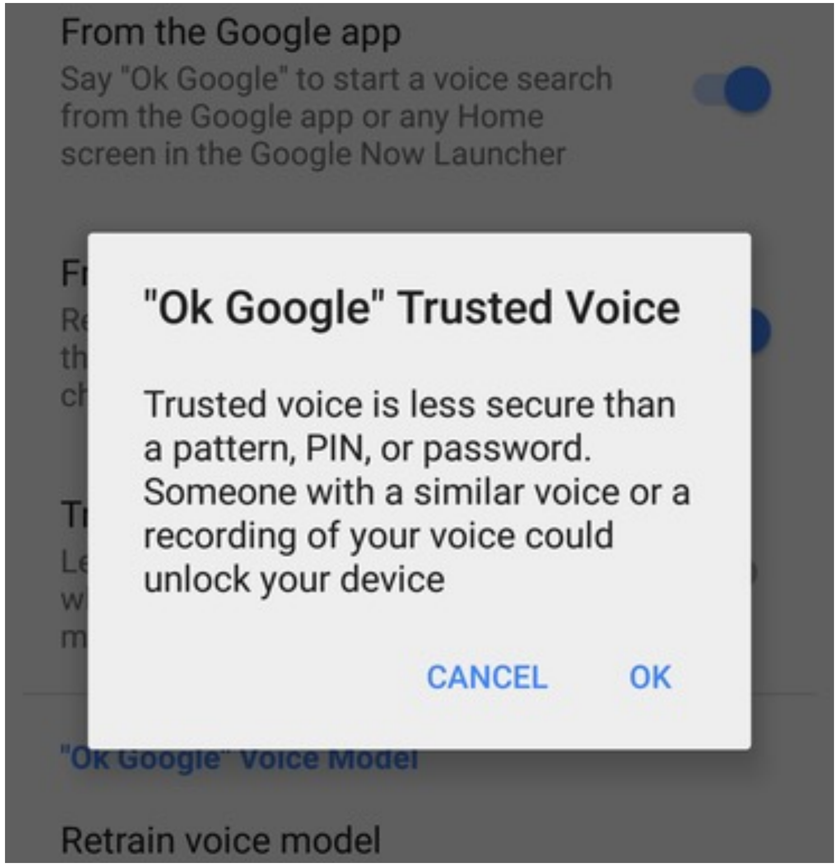
Using 2D images + 3D depth maps?

- More secure, but not perfect
- Can be broken by crafting a lifelike mask of the target




Specially processed area
2D images
Silicone nose
3D printed frame

Voice Recognition



Secure?

- Very much depends on the implementation

Some systems ask you to record a static phrase

- E.g. say "unlock" to unlock
- This is wildly insecure
  - Attacker can record and replay your voice

Others ask you to train a model of your voice

- Train the system by speaking several sentences
- To authenticate, speak several randomly chosen words
- Not vulnerable to trivial replay attacks, but still vulnerable
  - Given enough samples of your voice, an attacker can train a synthetic voice AI that sounds just like you

Fundamental Issue With Biometrics

Biometrics are immutable
- You are the password, and you can't change
- Unless you plan on undergoing plastic surgery?

Once compromised, there is no reset
- Passwords and tokens can be changed

Example: the Office of Personnel Management (OPM) breach
- US gov agency responsible for background checks
- Had fingerprint records of all people with security clearance
- Breached by China in 2015, all records stolen :(

Play-Doh Fingers

[Schuckers]

- Alternative to gelatin
- Play-Doh fingers fool 90% of fingerprint scanners
  - Clarkson University study
- Suggested perspiration measurement to test "liveness" of the finger

Token-based Two Factor Authentication

Types of Secrets

Actors provide their secret to log-in to a system

Three classes of secrets:

1. Something you know
   - Example: a password
2. Something you are
   - Examples: fingerprint, voice scan, iris scan
3. Something you have
   - Examples: a smart card or smart phone

Something You Have

Two-factor authentication has become more commonplace

Possible second factors:

- SMS passcodes
- Time-based one time passwords
- Hardware tokens

SMS Two Factor

Relies on your phone number as the second factor

- Key assumption: only your phone should receive SMS sent to your number

SMS two factor is deprecated. Why?

Social engineering the phone company

1. Call and pretend to be the victim
2. Say “I got a new SIM, please activate it”
3. If successful, phone calls and SMS are now sent to your SIM in your phone, instead of the victim

Not hypothetical: successfully used against many victims

One Time Passwords

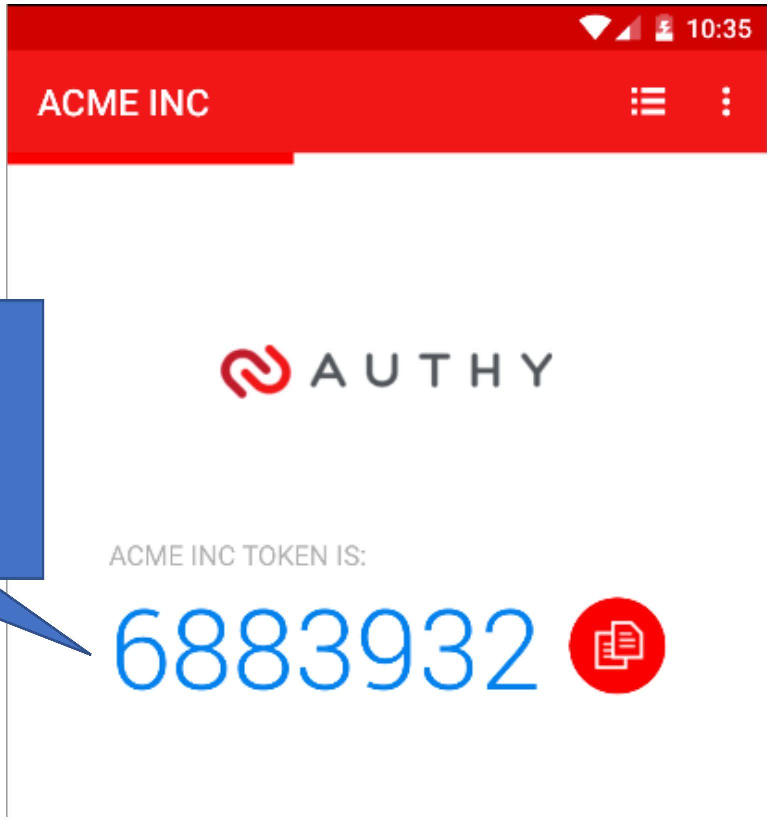
Generate ephemeral passcodes that change over time

To login, supply normal password and the current one time password

Relies on a shared secret between your mobile device and the service provider

- Shared secret allows both parties to know the current one time password

Changes every few minutes



Duo Mobile

Lastpass Authenticator

Google Authenticator

Time-based One-time Password Algorithm

T0 = <the beginning of time, typically Thursday, 1 January 1970 UTC>

TI = <length of time the password should be valid>

K = <shared secret key>

d = <the desired number of digits in the password>

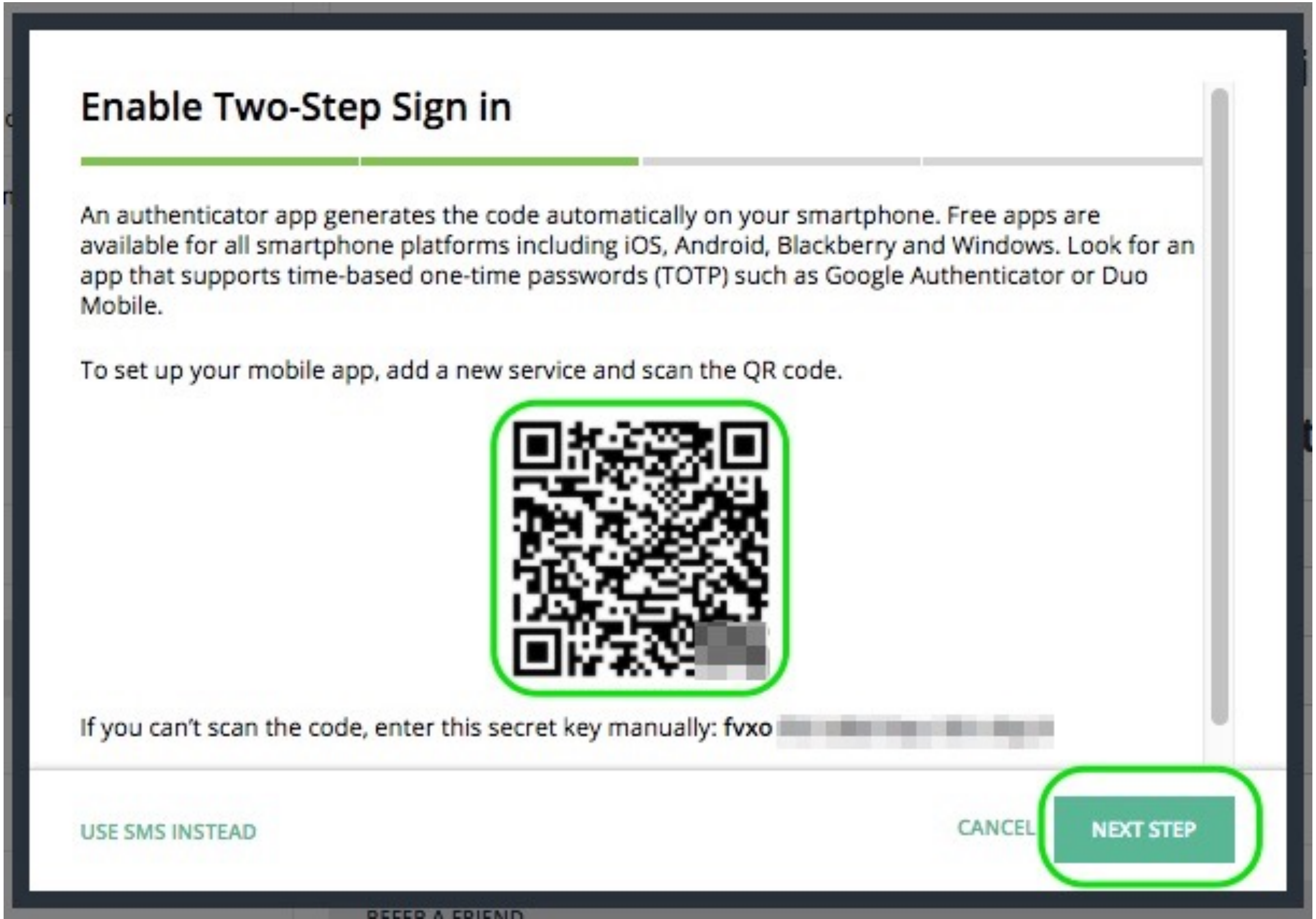
TC = floor((unixtime(now) − unixtime(*T0*)) / *TI*),

TOTP = HMAC(*K*, *TC*) % 10^d

Specially formatted SHA1-based signature

Given *K*, this algorithm can be run on your phone and by the service provider

Secret Sharing for TOTP



Hardware Two Factor

Special hardware designed to hold cryptographic keys

Physically resistant to key extraction attacks

- E.g. scanning tunneling electron microscopes

Uses:

- 2nd factor for OS log-on
- 2nd factor for some online services
- 2nd factor for password manager
- Storage of PGP and SSH keys

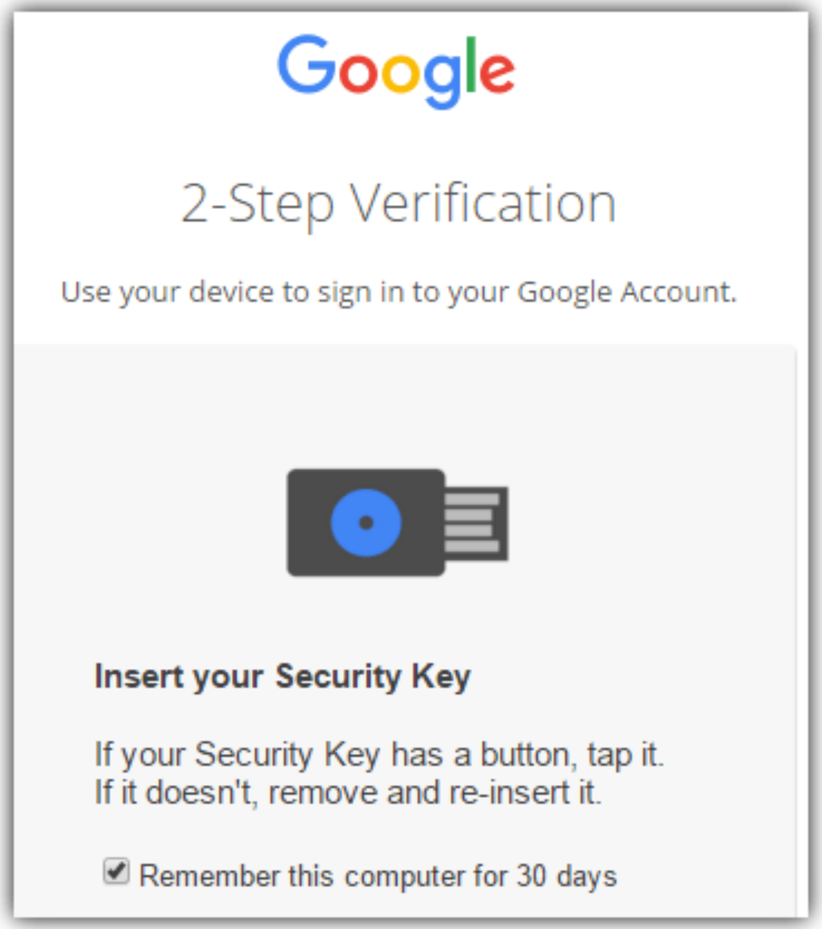
Universal 2nd Factor (U2F)

Supported by Chrome, Opera, and Firefox

Works with Google, Dropbox, Facebook, Github, Gitlab, etc.

Pro tip: always buy 2 security keys

- Associate both with your accounts
- Keep one locked in a safe, in case you lose your primary key ;)



Authentication in Linux

Unix, PAM, and crypt

Network Information Service (NIS, aka Yellow Pages)

Needham-Schroeder and Kerberos

Status Check

- At this point, we have discussed:
  - How to securely store passwords
  - Techniques used by attackers to crack passwords
  - Biometrics and 2^{nd} factors
- Next topic: building authentication systems
  - Given a user and password, how does the system authenticate the user?
  - How can we perform efficient, secure authentication in a distributed system?

Authentication in Unix/Linux

- Users authenticate with the system by interacting with *login*
  - Prompts for username and password
  - Credentials checked against locally stored credentials
- By default, password policies specified in a centralized, modular way
  - On Linux, using Pluggable Authentication Modules (PAM)
  - Authorizes users, as well as environment, shell, prints MOTD, etc.

Example PAM Configuration

```
# cat /etc/pam.d/system-auth
#%PAM-1.0

auth required pam_unix.so try_first_pa
auth optional pam_permit.so
auth required pam_env.so

account required pam_unix.so
account optional pam_permit.so
account required pam_time.so

password required pam_unix.so try_first_pass nullok sha512 shadow
password optional pam_permit.so

session required pam_limits.so
session required pam_unix.so
session optional pam_permit.so
```

- Use SHA512 as the hash function
- Use /etc/shadow for storage

Unix Passwords

- Traditional method: *crypt*
  - First eight bytes of password used as key (additional bytes are ignored)
  - 12-bit salt
  - 25 iterations of DES on a zeroed vector
- Modern version of *crypt* are more extensible
  - Full password used
  - Up to 16 bytes of salt
  - Support for additional hash functions like MD5, SHA256, and SHA512
  - Key lengthening: defaults to 5000 iterations, up to $10^8 - 1$

Password Files

- Password hashes used to be in */etc/passwd*
  - World readable, contained usernames, password hashes, config information
  - Many programs read config info from the file…
  - But very few (only one?) need the password hashes
- Turns out, world-readable hashes are **Bad Idea**
- Hashes now located in */etc/shadow*
  - Also includes account metadata like expiration
  - Only visible to root

Password Storage on Linux

/etc/passwd

username:x:UID:GID:full_name:home_directory:shell

cbw:x:1001:1000:Christo Wilson:/home/cbw/:/bin/bash
n Mislove:/home/amislove/:/bin/sh

$<algo>$<salt>$<hash>
Algo: 1 = MD5, 5 = SHA256, 6 = SHA512

/etc/shadow

ername:password:last:may:must:warn:expire:disable:reserved

cbw:$1$0nSd5ewF$0df/3G7iSV49nsbAa/5gSg:9479:0:10000::::
amislove:1l3RxU5F1$:8172:0:10000::::

Distributed Authentication

Distributed Authentication

- Early on, people recognized the need for authentication in distributed environments
  - Example: university lab with many workstations
  - Example: file server that accepts remote connections
- Synchronizing and managing password files on each machine is not scalable
  - Ideally, you want a centralized repository that stores policy and credentials

The Yellow Pages

- Network Information Service (NIS), a.k.a. the Yellow Pages
  - Developed by Sun to distribute network configurations
  - Central directory for users, hostnames, email aliases, etc.
  - Exposed through *yp** family of command line tools
- For instance, depending on */etc/nsswitch.conf*, hostname lookups can be resolved by using
  - */etc/hosts*
  - DNS
  - NIS
- Superseded by NIS+, LDAP

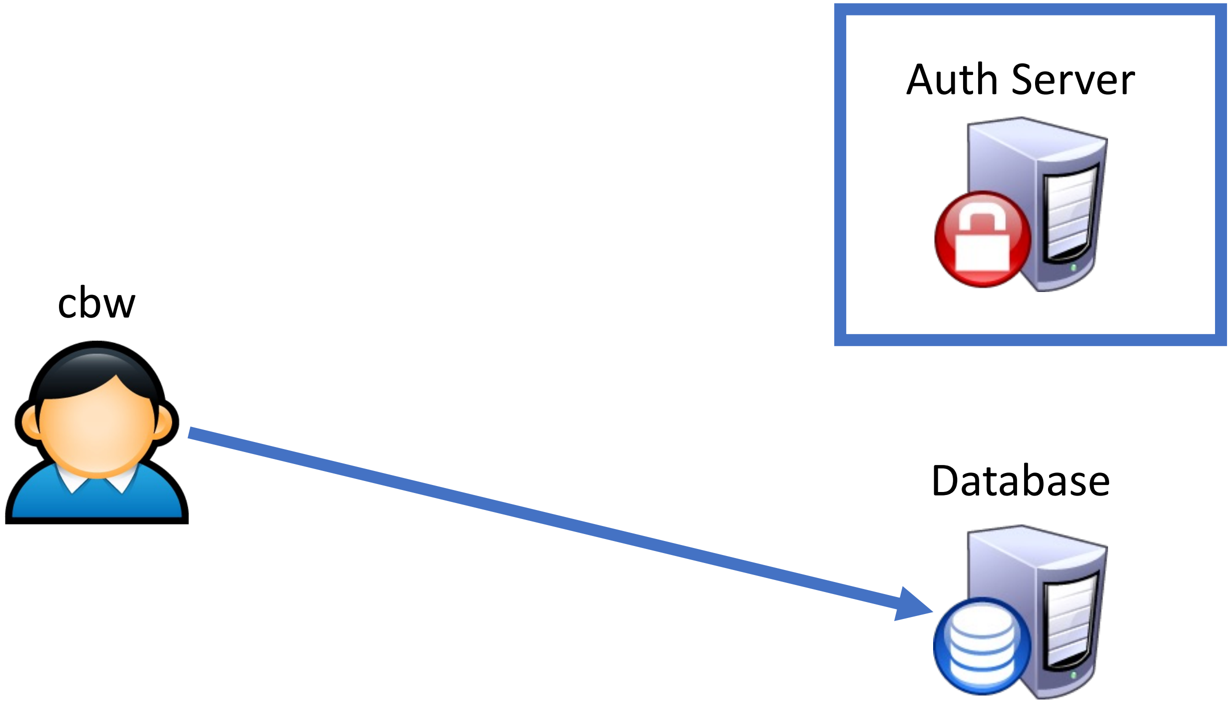
NIS Password Hashes

- *Crypt* based password hashes
- Can easily be cracked
- Many networks still rely on insecure NIS

```
[cbw@workstation ~] ypcat passwd
afbjune:qSAH.evuYFHaM:14532:65104::/home/afbjune:/bin/bash
philowe:T.yUMej3XSNAM:13503:65104::/home/philowe:/bin/bash
bratus:2omkwsYXWiLDo:6312:65117::/home/bratus:/bin/tcsh
adkap:ZfHdSwSz9WhKU:9034:65118::/home/adkap:/bin/zsh
amitpoon:i3LjTqgU9gYSc:8198:65117::/home/amitpoon:/bin/tcsh
kcole:sgYtUsOtyk38k:14192:65104::/home/kcole:/bin/bash
david87:vA06wxjJEUgBE:13055:65101::/home/david87:/bin/bash
loch:6HgIQrVkcBeiw:13729:65104::/home/loch:/bin/bash
ppkk315:s6CTSAkqqr/nU:14061:65101::/home/ppkk315:/bin/bash
haynesma:JYWaQUARSqDQE:14287:65105::/home/haynesma:/bin/bash
ckubicek:jYpwYhqqvr3tA:10937:65117::/home/ckubicek:/bin/tcsh
mwalz:wPIa5Bv/tFVb2:9103:65118::/home/mwalz:/bin/tcsh
sushma:G6XNe18GpeQj.:13682:65104::/home/sushma:/bin/bash
guerin1:n0Da2TmO9MDBI:14512:65105::/home/guerin1:/bin/bash
```

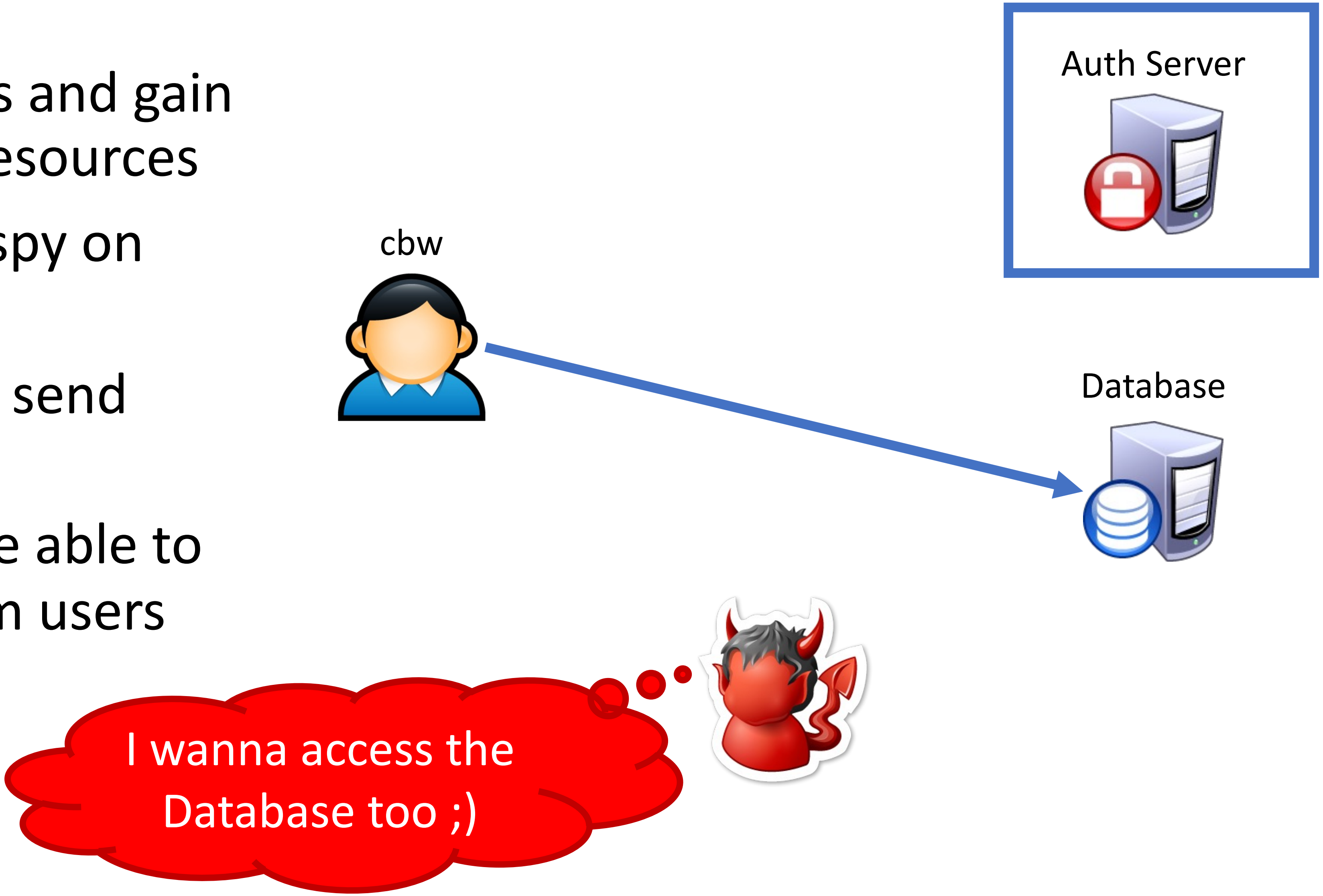
Distributed Authentication Revisited

- Goal: a user would like to use some resource on the network
  - File server, printer, database, mail server, etc.
- Problem: access to resources requires authentication
  - Auth Server contains all credential information
  - You do not want to replicate the credentials on all services



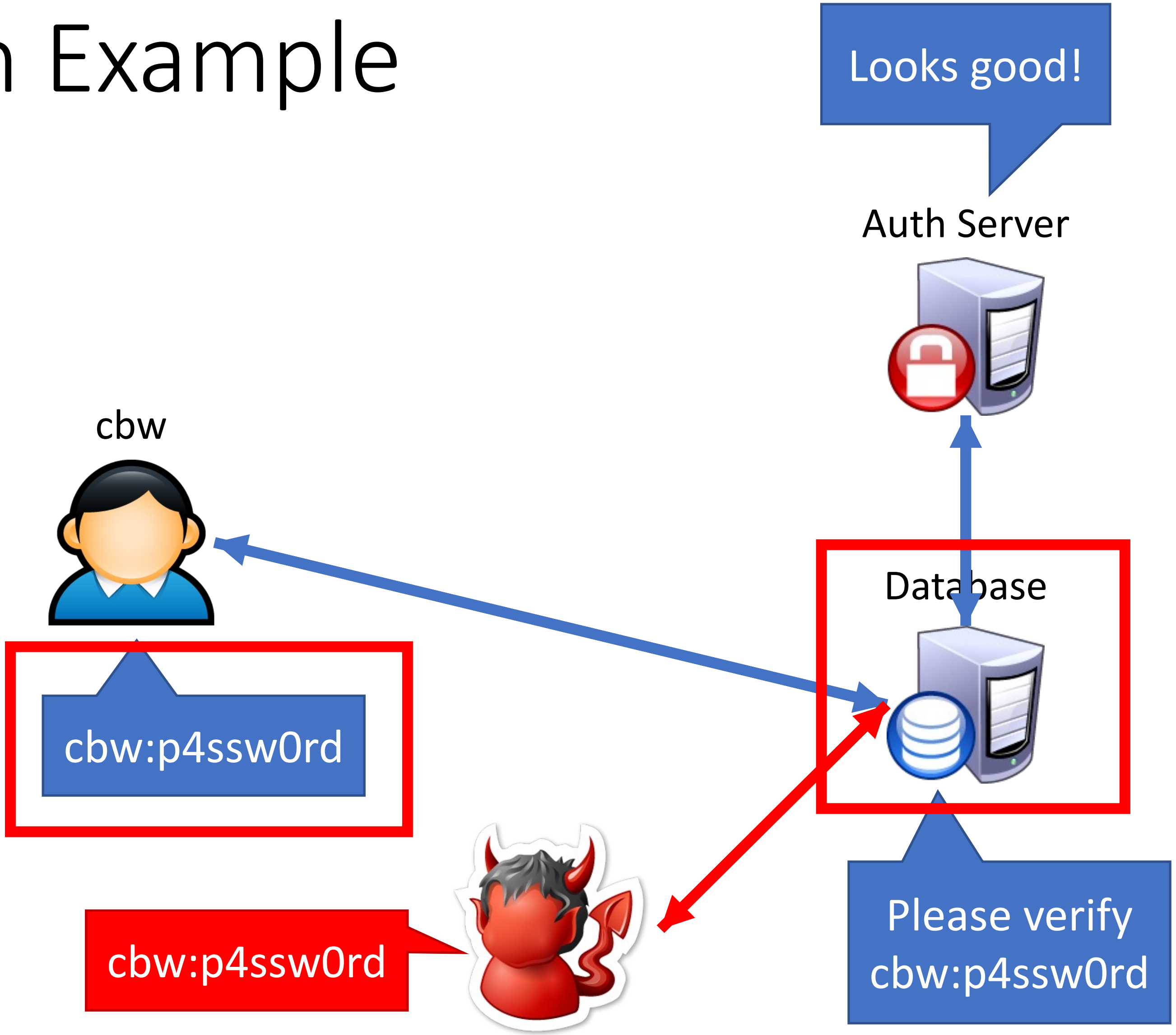
Attacker Goals and Threat Model

- Goal: steal credentials and gain access to protected resources
- Local attacker – may spy on traffic
- Active attacker – may send messages
- In some cases, may be able to steal information from users



(Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server
- Problems:
  - Passwords being sent in the clear
    - Attacker can observe them!
    - Clearly we need encryption
  - Database learns about passwords
    - Additional point of compromise
    - Ideally, only the user and the Auth Server should know their password



Symmetric Key Agreement among Multiple Parties

- For a group of N parties, every pair needs to share a different symmetric key
  - What is the number of keys?
  - What secure channel to use to establish the keys?

- How to establish such keys
  - Symmetric Encryption - Use a central authority, a.k.a. (TTP).
  - Asymmetric Encryption – PKI.

Needham-Schroeder Protocol

- Let Alice A and Bob B be two parties that trust server S
- K_{AS} and K_{BS} are shared secrets between [A, S] and [B, S]
- K_{AB} is a negotiated session key between [A, B]
- N_i and N_j are random nonces generated by A and B

1) $A \rightarrow S: A, B, N_i$
2) $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3) $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
4) $B \rightarrow A: \{N_j\}_{K_{AB}}$
5) $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

K_{AS} is not sent in the clear, authenticates S and A

K_{BS} is not sent in the clear, authenticates B

Challenge nonce forces A to acknowledge they have K_{AB}

Needham-Schroeder Example

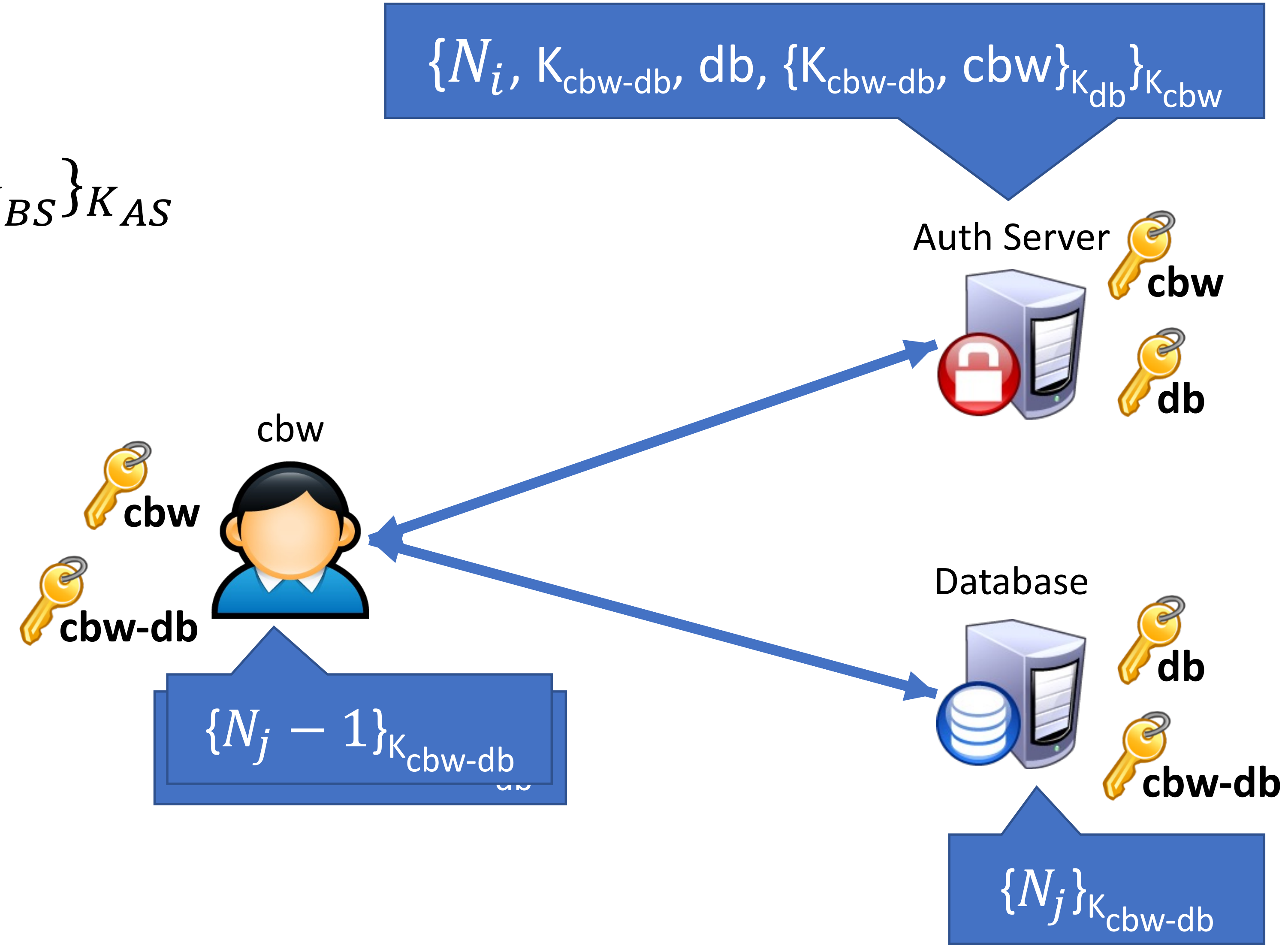
1) $A \rightarrow S: A, B, N_i$

2) $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3) $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

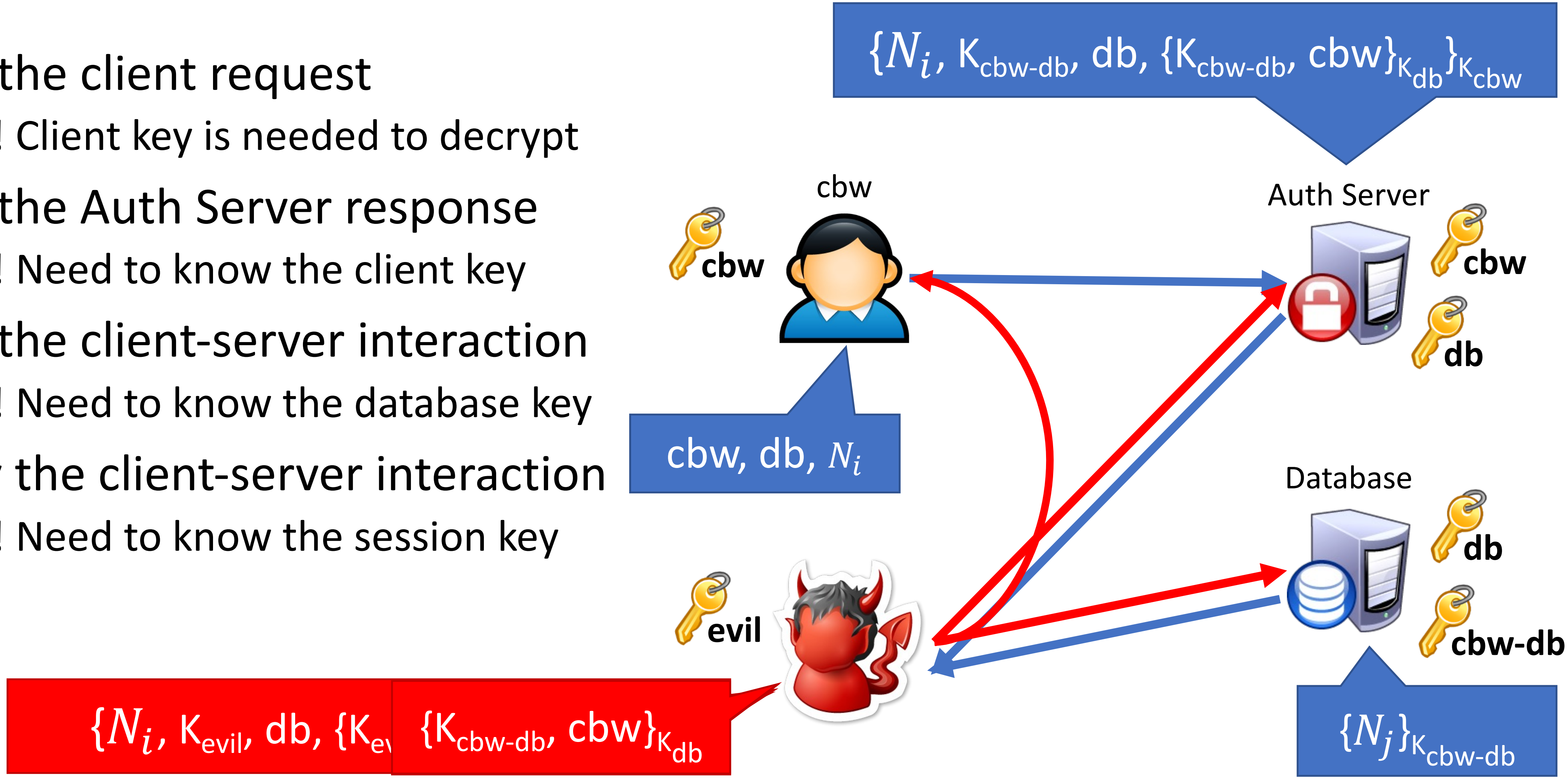
4) $B \rightarrow A: \{N_j\}_{K_{AB}}$

5) $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$



Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key
- Replay the client-server interaction
  - Fail! Need to know the session key



Replay Attack

Typical, Benign Protocol

1) $A \rightarrow S: A, B, N_i$
2) $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3) $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
4) $B \rightarrow A: \{N_j\}_{K_{AB}}$
5) $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

Replay Attack

1) $M \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
2) $B \rightarrow M: \{N_j\}_{K_{AB}}$
3) $M \rightarrow B: \{N_j - 1\}_{K_{AB}}$

- Attacker must hack *A* to steal K_{AB}
  - So the attacker can also steal K_{AS}
- However, what happens after *A* changes K_{AS}
  - Attacker can still conduct the replay attack! Only is K_{AB} necessary!

Fixed Needham-Schroeder Protocol

- Let Alice A and Bob B be two parties that trust server S
- K_{AS} and K_{BS} are shared secrets between [A, S] and [B, S]
- K_{AB} is a negotiated session key between [A, B]
- N_i and N_j are random nonces generated by A and B
- T is a timestamp chosen by S

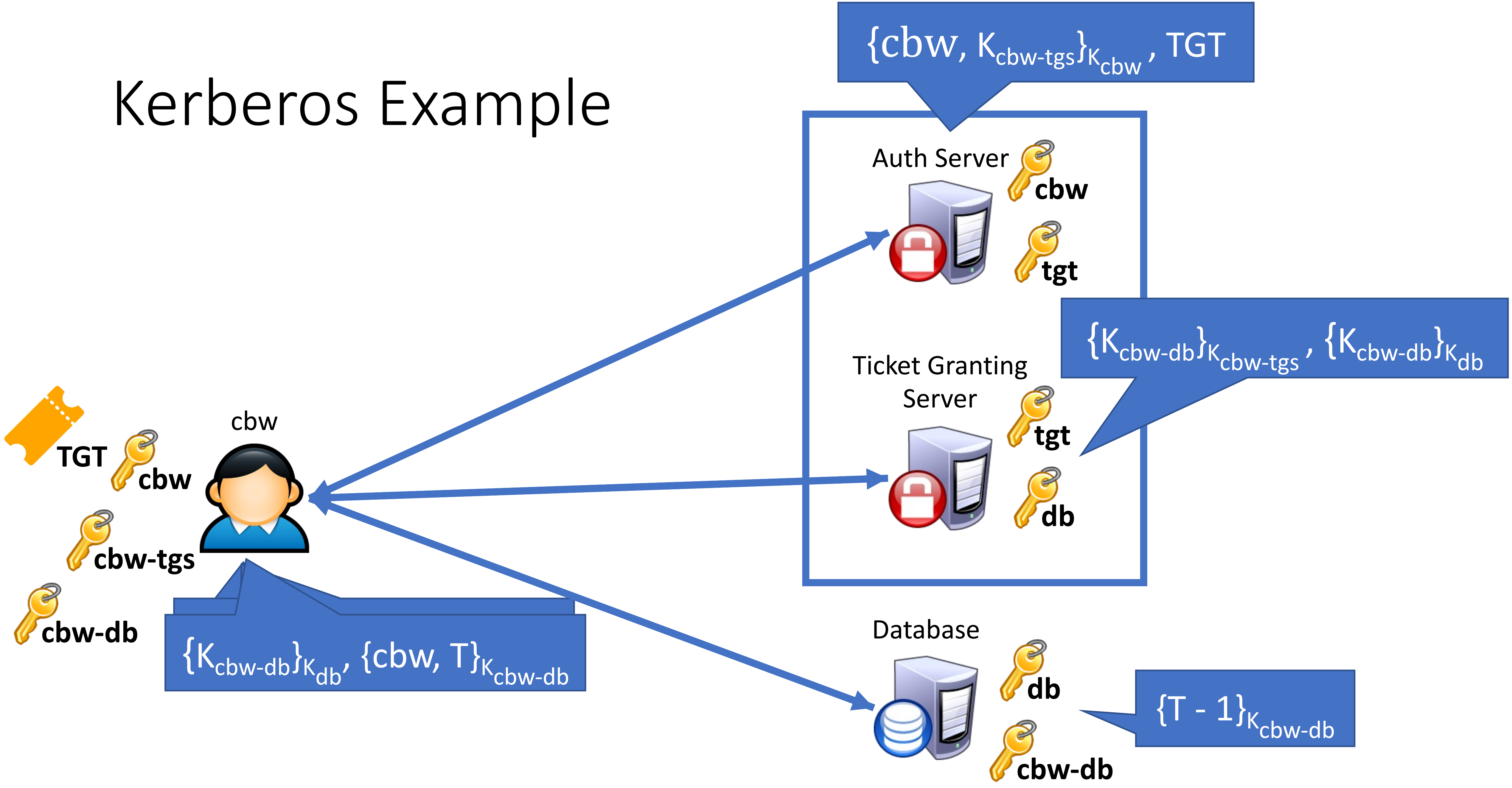
1) $A \rightarrow S: A, B, N_i$
2) $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A, T\}_{K_{BS}}\}_{K_{AS}}$
3) $A \rightarrow B: \{K_{AB}, A, T\}_{K_{BS}}$
4) $B \rightarrow A: \{N_j\}_{K_{AB}}$
5) $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

B only accepts requests with fresh timestamps

Kerberos

- Created as part of MIT Project Athena
  - Based on Needham-Schroeder
- Provides mutual authentication over untrusted networks
  - Tickets as assertions of authenticity, authorization
  - Forms basis of Active Directory authentication
- Principals
  - Client
  - Server
  - Key distribution center (KDC)
    - Authentication server (AS)
    - Ticket granting server (TGS)

Kerberos Example



Attacking Kerberos

- Don’t put all your eggs in one basket
  - The Kerberos Key Distribution Server (KDS) is a central point of failure
  - DoS the KDS and the network ceases to function
  - Compromise the KDS leads to network-wide compromise
- Time synchronization
  - Inaccurate clocks lead to protocol failures (due to timestamps)
  - Solution?
  - Use NTP ;)

Sources

1. Many slides courtesy of Wil Robertson: https://wkr.io
2. Honeywords, Ari Juels and Ron Rivest: http://www.arijuels.com/wp-content/uploads/2013/09/JR13.pdf

- For more on generating secure passwords, and understanding people's mental models of passwords, see the excellent work of Blas Ur: http://www.blaseur.com/pubs.htm