

CSE 127 Computer Security

Stefan Savage, Fall 2024, Lecture 15

User Authentication

Quick announcement

PA5 is optional because we're short on time

- We're going to try to have it up on Friday
 - We'll provide some modest amount of extra-credit for it (e.g., 2%)
 - So it'll help if you do it, but its going to be super important for your grade
- Its super-fun – local environment where you have a hacking mission; you learn the objectives along the way

Today: user authentication

Change of Focus

- Thus far we have largely focused entirely on **computers** (i.e., how we try to protect ourselves from attacks on code or the OS or the network by an untrusted party)
- Today, we're going to start talking about **people** too
- Today's issue:
 - How do we determine if a process is running on behalf of a *particular* trusted party?

The goal is to understand:

- Common techniques for authenticating users, locally and remotely;
- Security challenges associated with different authentication methods;
- Mitigations designed to address some of the above security challenges.

Authentication

Using cryptography Alice and Bob can authenticate each other by proving they know respective secret keys

- Alice sends a random challenge to Bob. Bob signs (or MACs) the challenge.
- Switch roles, repeat.

What exactly did we authenticate?

- Have Alice and Bob really committed their secret keys to memory?
- Did they manually perform cryptographic signing operations?



Authentication

Using cryptography Alice and Bob can authenticate each other by proving they know respective secret keys

- Alice sends a random challenge to Bob. Bob signs (or MACs) the challenge.
- Switch roles, repeat.

What exactly did they authenticate?

- Have Alice and Bob really committed their secret keys to memory?
- Did they manually perform cryptographic signing operations?

They authenticated each other's computers.



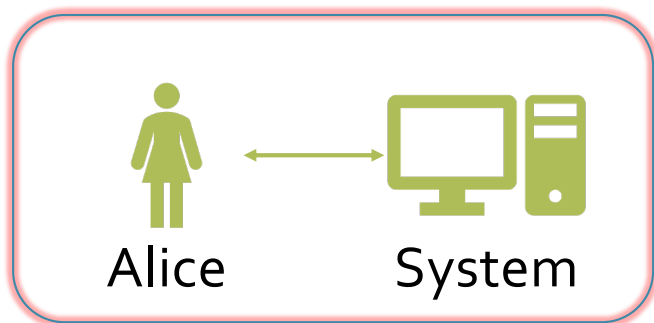
Authentication

How do we authenticate a human user to a system?

- System is often remote server

Authenticate: ascertain who is interacting with the system

- Necessary to apply appropriate security policy
- Only the intended subject should be able authenticate to the system as that subject



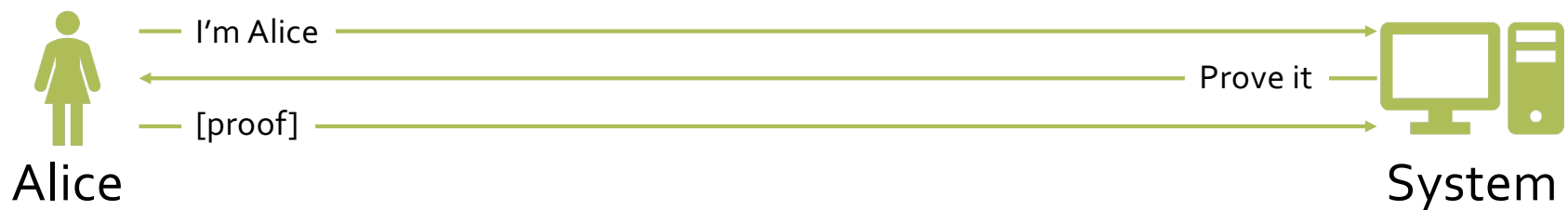
Authentication

How do we authenticate a human user to a machine?

Provide ***identity*** and proof of identity.

Identity examples:

- Name, Username, Student ID, Others?



Authentication

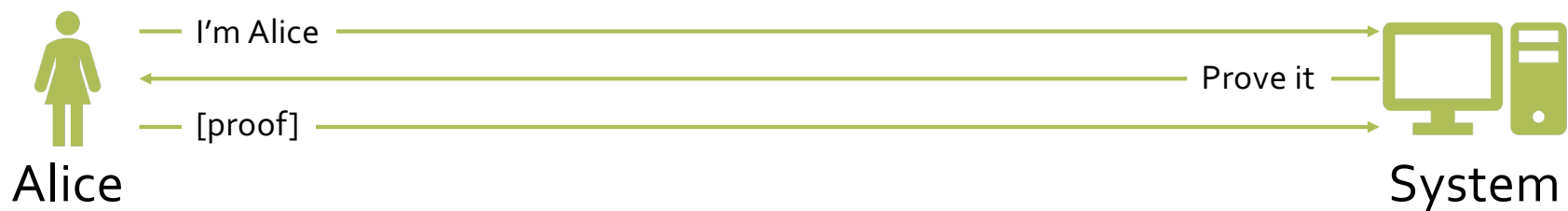
How can Alice prove that she's really Alice?

3 types of authentication factors

- Password: ***Something you know***
- Token: ***Something you have***
- Biometrics: ***Something you are***

Each factor can be used independently, or combined for ***multi-factor*** authentication.

- Typically 2-factor



Something You Know

Something You Know

A secret that only the real Alice should know

- A secret passcode

Examples: PIN*, password

- A secret about Alice

Examples: mother's maiden name, first pet, mortgage payment

Technically, only proves knowledge of secret, not that it's really Alice

- Secrets leak, can be shared, guessed.

* PIN: Personal Identification Number

- Misnomer. Usually used for authentication, not identification.

Passwords

How does Alice prove she knows the password?

Simplest: Alice provides the password to the system

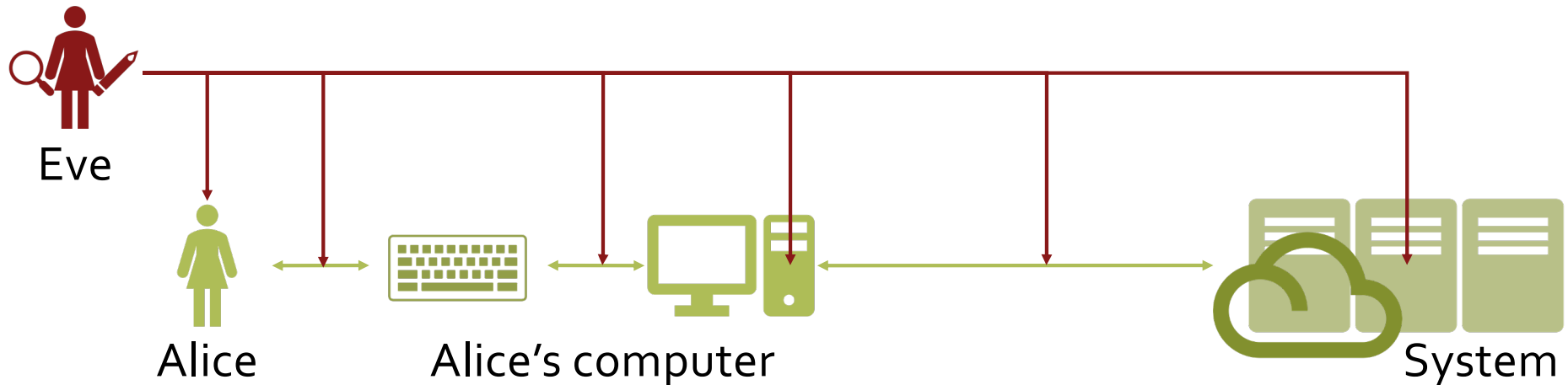
Problems?

- Passive adversary may observe password in transit
Need secure channel to protect confidentiality
- Active adversary may impersonate the system
Alice needs a way of authenticating the system

Setting

Alice uses a keyboard to type her password into client software that sends it on to the remote system for authentication.

Which points can Eve attack?

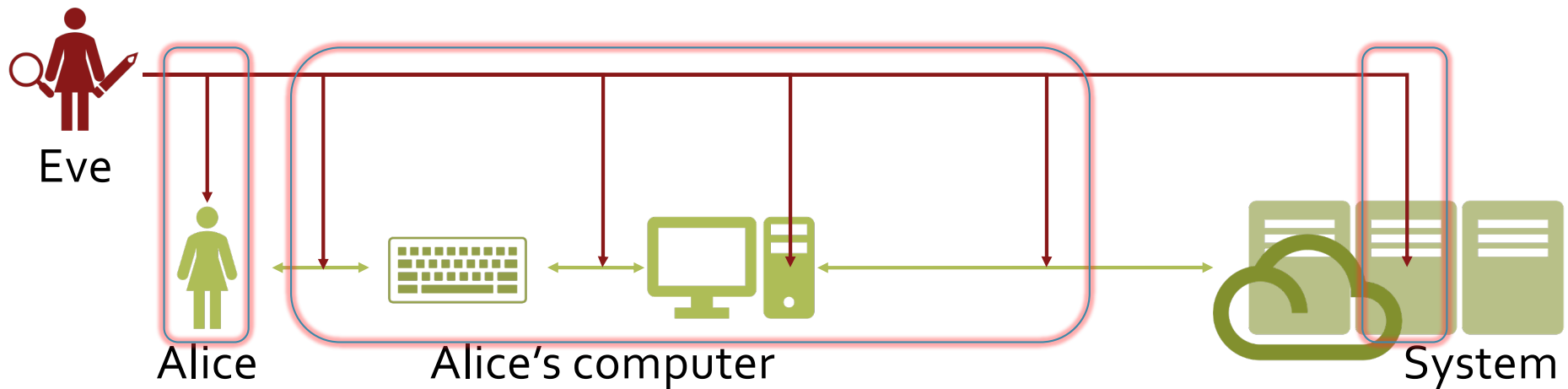


Attacking Passwords

Get it from Alice

Intercept it

Get it from the system



Attacking Passwords

\$5 wrench?

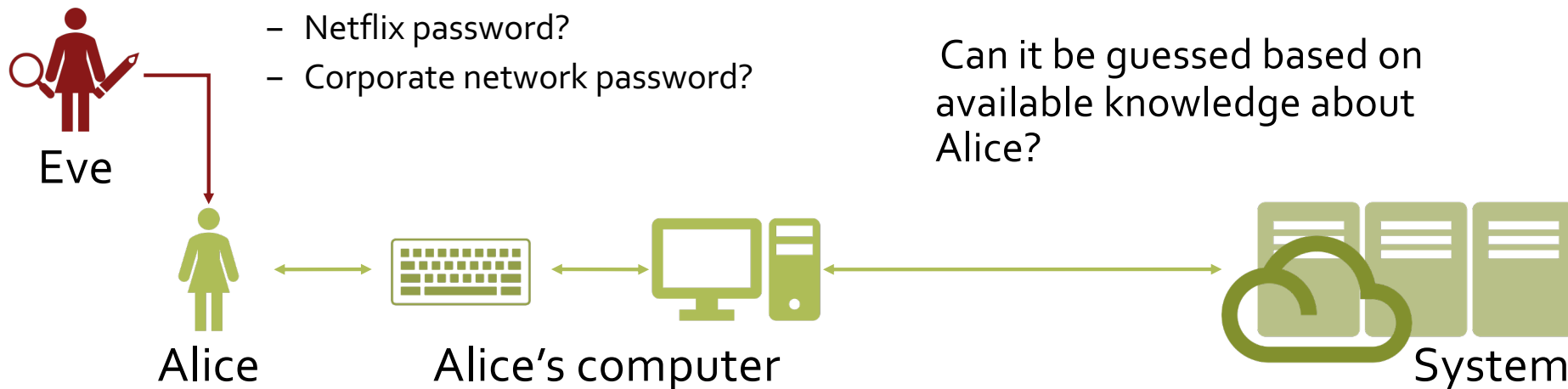
Is Alice vested in keeping it secret?

- Debit card PIN number?
- Personal email password?
- Netflix password?
- Corporate network password?

Is it written down somewhere?

- Good against remote attackers
- Not good against targeted local attacks (co-workers, family, abusers)
- Know your threat model!

Can it be guessed based on available knowledge about Alice?



Strong Passwords

The challenge is to come up with passwords that are hard to guess, but easy to remember.

Common password rules:

- Composition
 - Letters and numbers, mixed case, symbols
 - Banned dictionary
- Length
- Lifetime

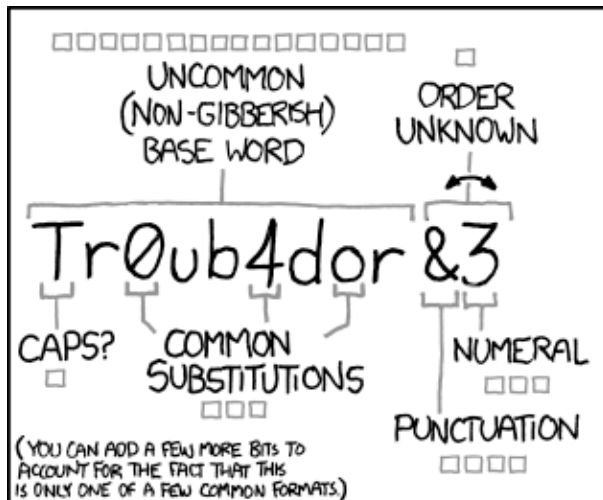
But... unintended consequences

- Required letters/symbols -> ?
- Monthly change requirement -> ?

Top 25 most common passwords by year according to SplashData

Rank	2011 ^[4]	2012 ^[5]	2013 ^[6]	2014 ^[7]	2015 ^[8]	2016 ^[3]	2017 ^[9]
1	password	password	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password
3	12345678	12345678	12345678	12345	12345678	12345	12345678
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty
5	abc123	qwerty	abc123	qwerty	12345	football	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789
7	1234567	letmein	111111	1234	football	1234567890	letmein
8	letmein	dragon	1234567	baseball	1234	1234567	1234567
9	trustno1	111111	iloveyou	dragon	1234567	princess	football
10	dragon	baseball	adobe123 ^[a]	football	baseball	1234	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin
12	111111	trustno1	admin	monkey	1234567890	welcome	welcome
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey
14	master	sunshine	letmein	abc123	111111	abc123	login
15	sunshine	master	photoshop ^[a]	111111	1qaz2wsx	admin	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars
17	bailey	welcome	monkey	access	master	flower	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd
20	123123	football	12345	michael	login	sunshine	master
21	654321	jesus	password1	superman	princess	master	hello
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom
23	qazwsx	ninja	azerty	123123	solo	loverme	whatever
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx
25	Football	password1	000000	trustno1	starwars	password1	trustno1

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



~28 BITS OF ENTROPY

$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$

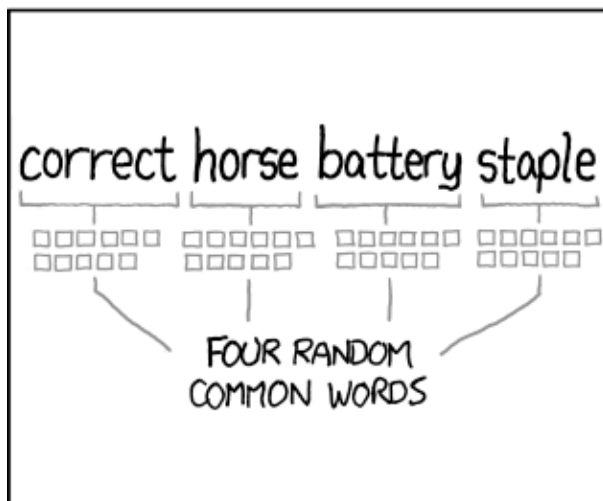
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER: **HARD**



~44 BITS OF ENTROPY

$2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$

DIFFICULTY TO GUESS: **HARD**

THAT'S A BATTERY STAPLE.

CORRECT!

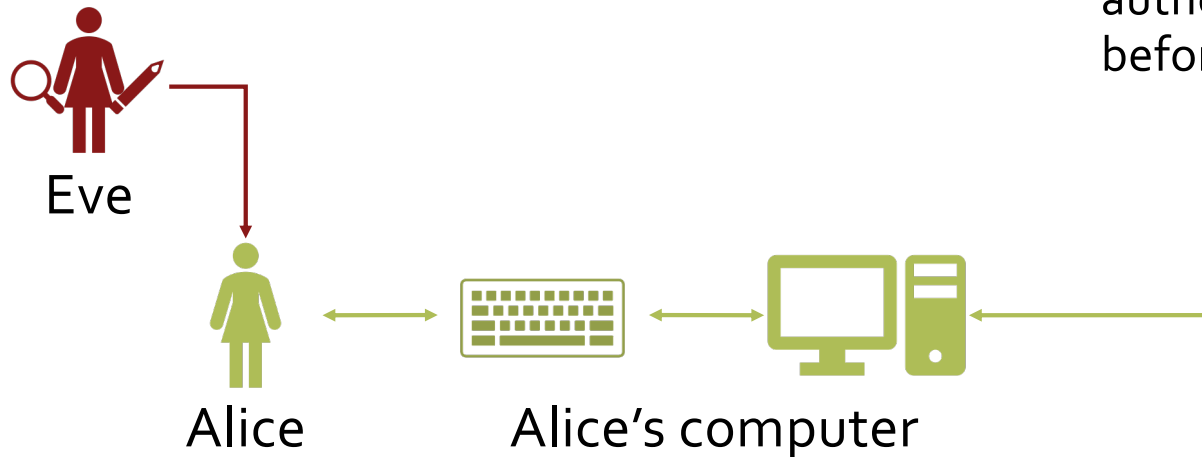
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Attacking Passwords

Can Eve trick Alice into revealing her password?

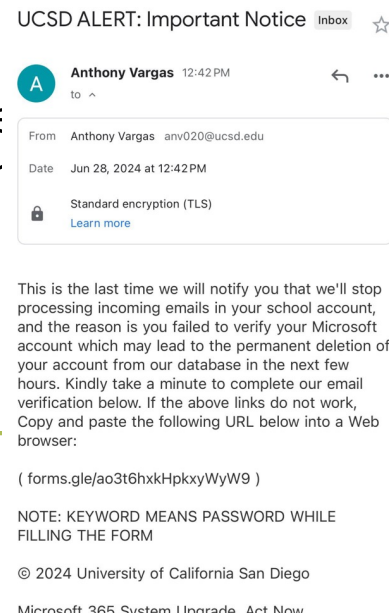
How does Alice know that she is logging into the real System?



Phishing!

- Tricking Alice into revealing her password by impersonating the system she is trying to access.

Alice authenticates before



password.



Phishing

How can Alice authenticate the System?

HTTPS certificates validate the URL.

What does it really tell you?

- That you are communicating to a server owned by UCSD?
- No. Only that you are communicating to www.ucsd.edu and your connection is secure (confidentiality and integrity are protected) against passive and active attackers on the link.

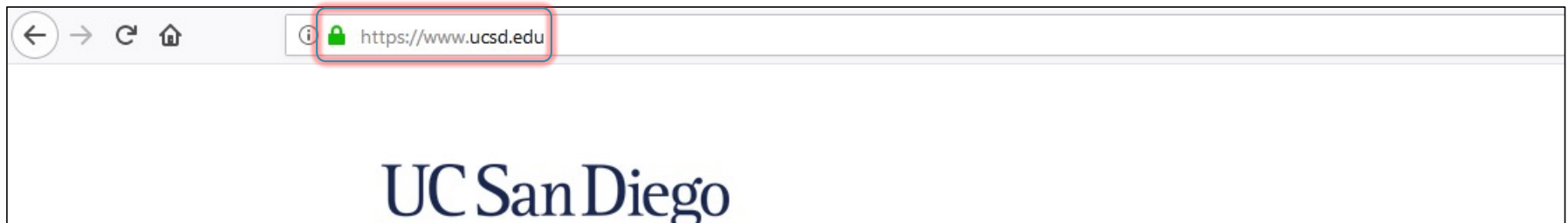


Phishing

How do you know www.ucsd.edu is a legitimate UCSD web site?

What about:

- www.cse.ucsd.edu
- www.ucsd.cse.edu
- www.cse-ucsd.edu
- www.cs.ucsd.e.duck
- www.cs.ucsd.education



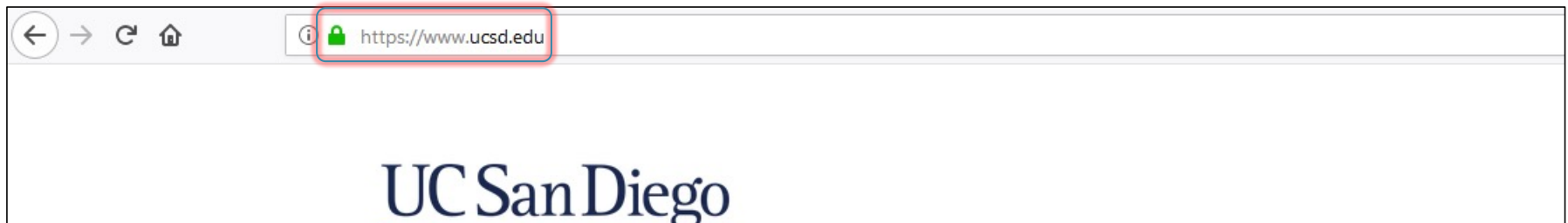
Phishing

How do you know www.ucsd.edu is a legitimate UCSD web site?

A user is expected to know which domains are associated with the entity they are trying to interact with.

- And how to properly parse the URL

Some browsers now highlight the domain portion




Phishing


What if the user knows which domain is real?

Homoglyphs: symbols that appear identical or very similar

Attack: register domain names that look just like the victim domain, but using a different character set.



 *Latin Small Letter A*
Unicode U+0061

 *Mathematical Sans-Serif Small A*
Unicode U+1D5BA

Both set in Helvetica Neue

Phishing

<https://www.irongeek.com/homoglyph-attack-generator.php>

– ucsd.edu

Homoglyph Attack Generator

This app is meant to make it easier to generate homoglyphs based on homoglyphs that having to search for look-a-like character in Unicode, then copying and pasting. Please use only for legitimate pen-test purposes and user awareness training. I also recommend website developers use it to find out possible user impersonation attacks in their code. This is still a work in progress, so please send me suggestions (especially for new homoglyphs to add). While this tool was designed with making DNS/Punycode names for pulling into DNS to display foreign characters in a browser's URL bar, it can be used for other things. Try guessing the DNS/Punycode stuff and just making look-alike user names for systems that accept Unicode. I made this tool to easily generate homoglyphs based on homoglyphs in Unicode and to test out how different apps display them. It seems like a lot of modern browsers have gotten better at warning the users of attack, but it's nice to have experience about other apps that accept Unicode/Punycode/Internationalized Domain Names, especially weblogs.

For more information see my [Paper Proposal for "List of Character Use of Punycode and Homoglyph Attacks to Obfuscate URLs for Phishing"](#)

1st: Type in a name to look like:

ucsd.edu

2nd: Choose Homoglyphs to add:

3rd: Output will be something like this:

This one is for testing/learning:
ucsd.edu

This one is for you can copy & paste:
ucsd.edu



The screenshot shows a web application titled 'Homoglyph Attack Generator'. It contains a text input field with 'ucsd.edu' and a button labeled 'Generate'. Below the input, there is a section titled '2nd: Choose Homoglyphs to add:' which displays a grid of various Unicode characters that look similar to the characters in 'ucsd.edu'. These characters are organized by the original character they replace (e.g., 'u', 'c', 's', 'd', 'e', '.'). At the bottom, there are two text areas: '3rd: Output will be something like this:' followed by 'ucsd.edu' (labeled 'This one is for testing/learning:') and another 'ucsd.edu' (labeled 'This one is for you can copy & paste:').

Homoglyph Attack Generator

This app is meant to make it easier to generate homographs based on Homoglyphs than having to search for look-a-like character in Unicode, then coping and pasting. Please use only for legitimate pen-test purposes and user awareness training. I also recommend webapp developers use it to test out possible user impersonation attacks in their code. This is still a work in progress, so please send me suggestions (especially for new Homoglyphs to add). While this tool was designed with making IDNA/Punycode names for putting into DNS to display foreign characters in a browsers URL bar, it can be used for other things. Try ignoring the IDNA/Punycode stuff and just making look alike user names for systems that accept Unicode. I made this tool to easily generate homographs based on homoglyphs in Unicode and to test out how different apps display them. It seems like a lot of modern browsers have gotten better at warning the users of attack, but I'd love to hear experiences about other apps that accept Unicode/Punycode/Internationalized Domain Names, especially webapps.

For more information see my [Paper Proposal for "Out of Character: Use of Punycode and Homoglyph Attacks to Obfuscate URLs for Phishing"](#).

1st, type in a name to look like:

ucsd.edu

○ Rev^{202E} ○ U ○ C ○ S ○ D ○ . ○ E ○ D ○ U
○ u ○ c ○ s ○ d ○ .660 ○ e ○ d ○ u
○ μ^{3bc} ○ c^{3f2} ○ S⁴⁰⁵ ○ D^{10e} ○ 6.d4 ○ È ○ D^{10e} ○ μ^{3bc}
○ U^{3c5} ○ C^{3f9} ○ S⁴⁵⁵ ○ d^{10f} ○ 701 ○ É ○ d^{10f} ○ U^{3c5}
○ U⁵³¹ ○ C⁴²¹ ○ S^{54f} ○ D¹¹⁰ ○ 702 ○ È ○ D¹¹⁰ ○ U⁵³¹
○ U^{54d} ○ c⁴⁴¹ ○ S^{10bd} ○ d¹¹¹ ○ .2024 ○ È ○ d¹¹¹ ○ U^{54d}
○ U^{22c3} ○ C^{13df} ○ S^{13da} ○ d⁵⁰¹ ○ .2027 ○ é ○ d⁵⁰¹ ○ U^{22c3}
○ U^{#35} ○ C^{216d} ○ d⁸⁰¹ ○ d^{56a} ○ .3002 ○ â ○ d^{56a} ○ U^{#35}
○ u^{#55} ○ c^{217d} ○ d^{c96} ○ D^{13a0} ○ .#0e ○ ê ○ D^{13a0} ○ u^{#55}
○ d⁸⁰¹ ○ S^{#33} ○ d^{1e0d} ○ .#61 ○ È¹¹² ○ d^{1e0d}
○ d^{ca8} ○ s^{#53} ○ D^{216e} ○ È¹¹³ ○ D^{216e}
○ c^{#23} ○ d^{217e} ○ È¹¹⁴ ○ d^{217e}
○ c^{#43} ○ D^{#24} ○ È¹¹⁵ ○ D^{#24}
○ d^{#44} ○ È¹¹⁶ ○ d^{#44}
○ È¹¹⁷
○ È¹¹⁸
○ È^{11a}
○ È^{11b}
○ È³⁹⁵
○ È⁴¹⁵
○ e⁴³⁵
○ È^{13ac}
○ E^{#25}
○ e^{#45}

2nd, choose homoglyphs to use:

This one is for testing linking:

[ucsd.edu](#)

3rd, Output will be something like this:

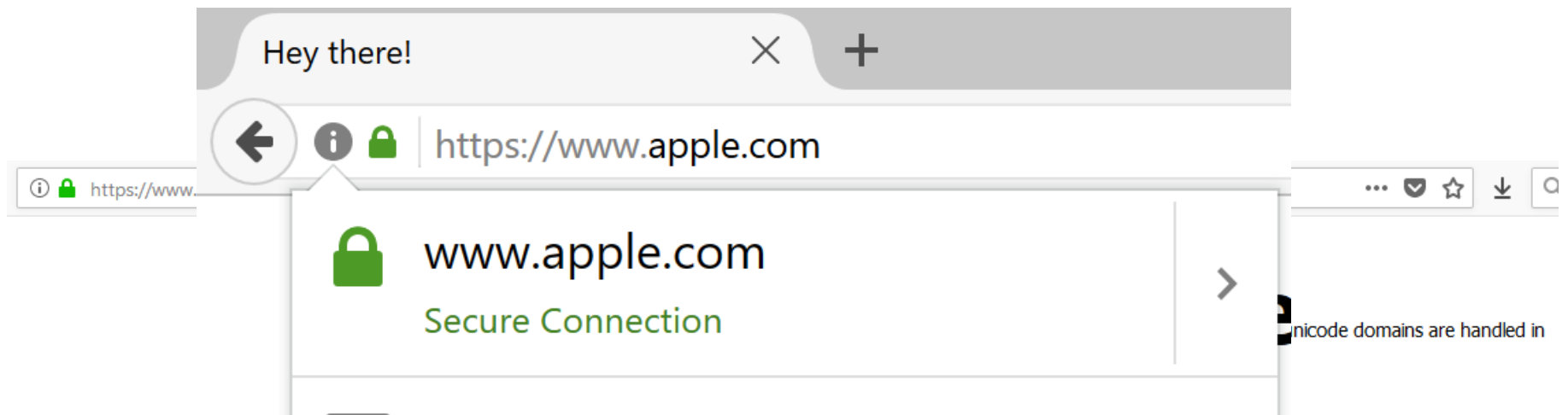
This one is so you can copy & paste:

ucsd.edu

Phishing

Better Example: <https://www.apple.com/>

- <https://www.xudongz.com/blog/2017/idn-phishing/>



Phishing

Related: When logging into a machine locally, how does Alice know that she is entering the password into the **real** login program?

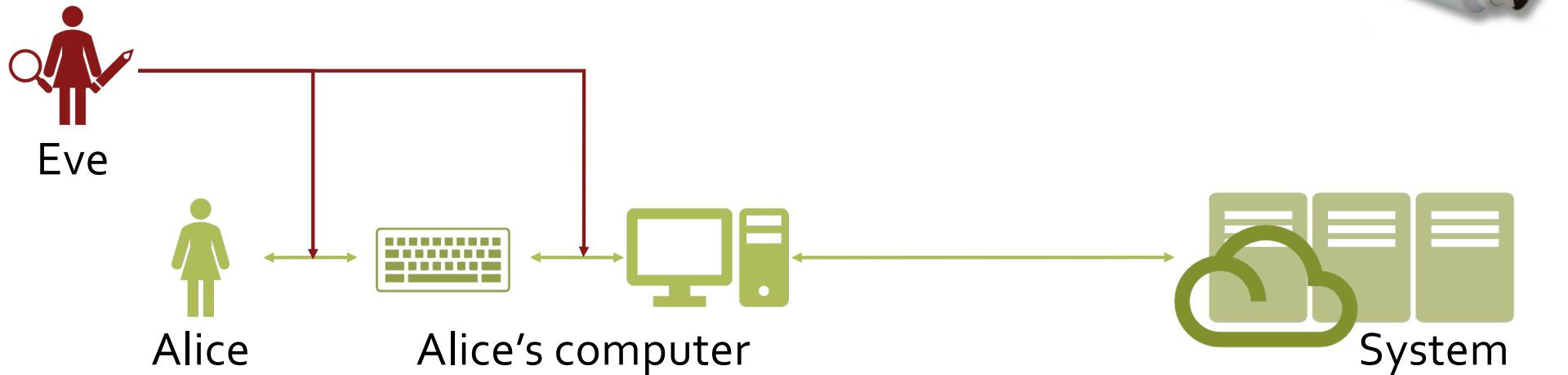
- ***Trusted path***: mechanism that guarantees user is interacting with intended component
- CTRL+ALT+DEL on Windows

Attacking Passwords

Shoulder-surf

Side channels

Hardware keyloggers



Attacking Passwords

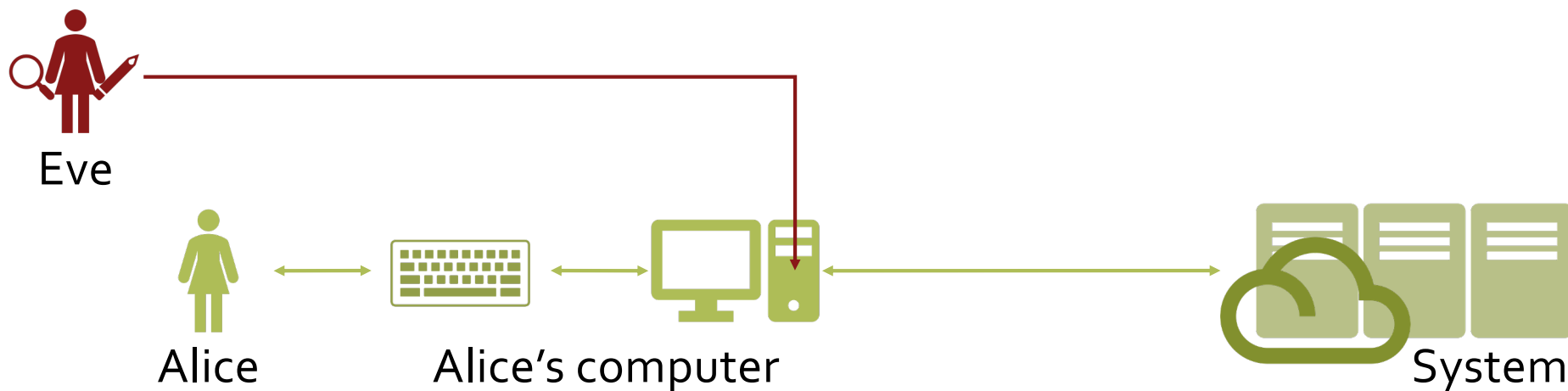
Software keyloggers

Passwords in memory

- Internal buffers
- Clipboard

Stored passwords

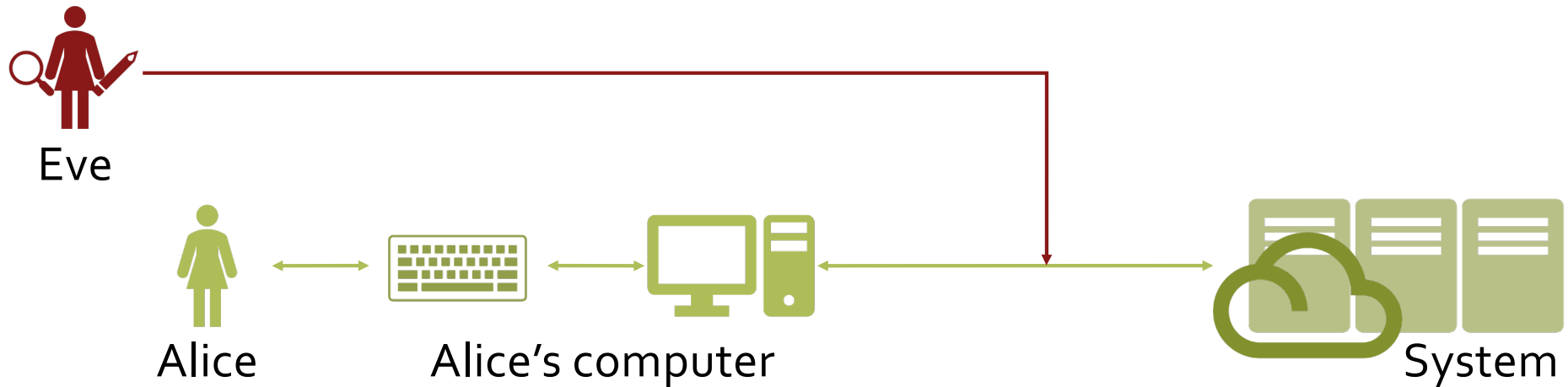
- Cached passwords (e.g. browsers)
- Password managers
 - Good ones are well protected by master passwords
- AlicePasswords.txt



Attacking Passwords

Monitoring the transmission channel

- Channel should be encrypted to protect password confidentiality
- Examples: TLS/SSH/HTTPS



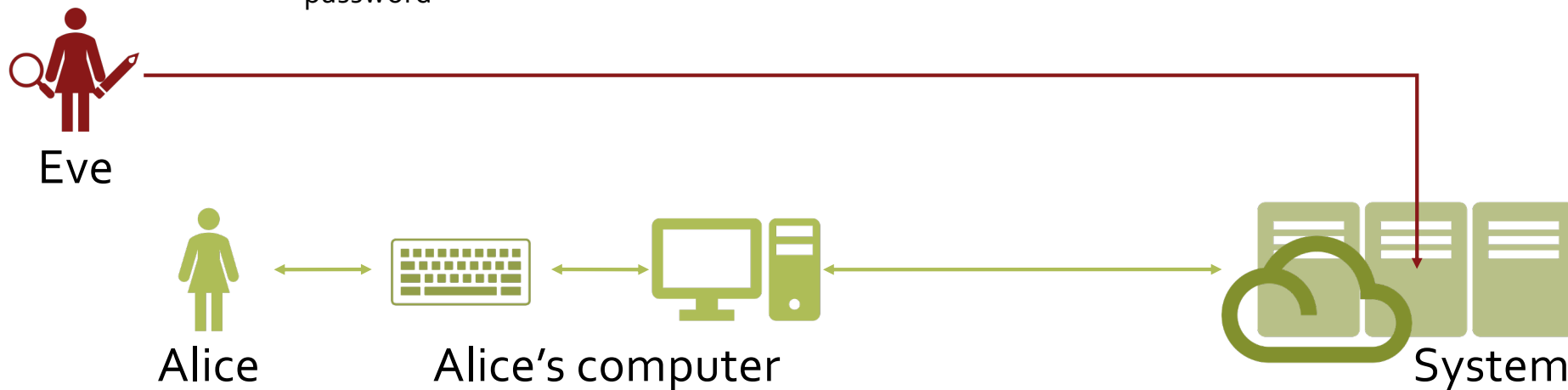
Attacking Passwords

Use system as an oracle: try to log in with different passwords

- Defense: Minimize error information
- Defense: Limit number of login attempts per user
- Attack: Try different users for common password

Compromise password database

- Huge yield compared to user-side attacks
- <https://haveibeenpwned.com/>
- ***Password reuse issues...***



Protecting Passwords

How can the system verify that the password Alice entered is correct?

Naïve solution:

- Store a copy of the password and compare provided copy to the stored one.

Problem?

If system is compromised, passwords are revealed.

- Same passwords may be used on other systems.

Protecting Passwords

Other solutions?

Hint: System does not need to know the password, only be able to verify it is correct.

What if the system stores a cryptographic hash of the password?

- $H(p)$
- Hash must be pre-image resistant.

Better...

- ... but still problematic

Protecting Passwords

Given a hash of a password, Eve can use it to validate guesses.

- Also, obvious which users have identical passwords

Dictionary Attacks

- Dictionary: collection of possible, or likely, password strings
- Try every string in the dictionary until the correct entry is found.

Pre-compute hashes of all strings in the dictionary, then perform reverse look-ups by hash to find corresponding password.

Protecting Passwords

Dictionary Attack Example:

- Assume passwords are composed of upper or lower case letters or digits
 $26+26+10 = 62$ possible values per character (round off to 64, so we have a power of 2)
- $64^n = 2^{6n}$ possible passwords of length n
- For $n = 6$: 2^{36} possible password strings
~10TB to store all possible 6-character passwords and respective SHA1 hashes

Protecting Passwords

How do we make dictionary attacks harder?

Note, the attacker only had to compute one dictionary of hashes that could then be used for any user's password hash from any system.

We can parameterize, or "**salt**", password hashes with unique random numbers

- Instead of storing $H(p)$, store $(r, H(r||p))$, where r is random salt of n bits
- Precomputation is no-longer possible. Attacker must compute unique hashes for every target with different r .

Better...

- ... but still problematic

Be careful about computation assumptions

From 2012, Gosney's 25GPU password cracking cluster

- 350B NTLM hashes (used by Windows) per sec
- 180B MD5 hashes/sec, 63B SHA-1 hashes/sec
- Modern rigs are even faster
(e.g., KnowBe4 7.5T /sec [NTLM], 4.1T/sec

For state actors, custom hardware feasible



Remember, people don't pick random passwords

We pick strings we can remember...

Real dictionary attacks use password dictionaries

- Combinations of words, numbers, letter translations, inversions, etc that are in common use
- Can typically guess 20-50% of passwords in the first billion guesses
 - Remember how long a billion NTLM, MD5 or SHA1 hashes takes?

Protecting Passwords

How do we make dictionary attacks even harder?

Hint: the computation to verify a password for a given user on a legitimate system happens relatively infrequently, but an attacker attempting to crack a password hash must perform many, many attempts.

So, we can use a deliberately slow and resource-consuming hashing function.

- PBKDF2, bcrypt, scrypt

Protecting Passwords

Building blocks for password protection

- Hash
- Salt
- Slow down

Use one of:

- PBKDF2
- bcrypt
- scrypt

Something You Have

Something You Have

Something only Alice should have

- Examples: key, smartcard, RFID badge, SecurID token

Frequently used as a second factor (in combination with a passcode)

- 2FA token

Technically, only proves possession of the token, not that it's really Alice

- Tokens get shared, lost, stolen, duplicated.

Smartcards

Idea: Put a secret key into a tiny computer that Alice can carry with her.

- Plastic card with an embedded integrated circuit
- Provisioned with secret key(s)
- Interacts with readers through contact pads or short range wireless (NFC)

Many uses beyond user authentication

- Stored value payment and transit, SIM cards, satellite TV

Sample authentication protocol:

- Interrogate with a random challenge and verify signed response



One Time Passcode Tokens

Same basic idea as smart card: a tiny computer with a secret

- But, typically without direct computer interface.

How to provide challenge and get response?

Response is displayed on token screen, user types it into the authentication system.

- Typically using current time instead of challenge (requires time sync)
 - Some versions simply generate random OTPs that can be validated (no time sync needed)
- Some variants have keypads to allow the user to type in a challenge as well.



One Time Passcode Tokens

Typical protocol:

- Based on symmetric key cryptography (shared secret between token and authenticating server)
- Periodically (e.g. once a minute) token generates a new single-use code by MACing current time.
- To authenticate, Alice types in her password and current code (two-factor).



Strengths:

- Knowing the password is not enough to impersonate Alice.
- Each code is single-use. Eavesdropping (shoulder-surfing, keylogging, etc.) does not enable Eve to impersonate Alice in the future.
- Observing any number of codes does not help in predicting future ones.

One Time Passcode Tokens

Weaknesses:

- Vulnerable to man-in-the-middle and phishing attacks.
- Server needs to know the secret key to validate token codes. Single point of failure.
 - E.g., RSA Breach
- Does not scale well to multiple accounts.



USB/NFC tokens

Same basic idea but ...

- Device to device

User never gets to see code so can't be socially engineered

Compromised OS/hardware can't play MiTM without being detected



- Interfaces to tie into a variety of authentication infrastructures

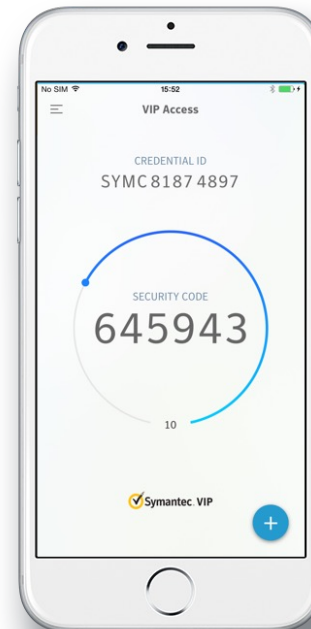
In principal, can avoid the "lots of tokens" problem

Ongoing standardization work, but the heir apparent to remote passwords

One Time Passcode Without Tokens

Virtual edition

- Everybody (in some parts of the world) already carries a tiny computer. Let's just use that.
- Strength: better scaling, support multiple keys with the same physical device.
- Weakness: the two authentication factors are not as isolated anymore.



One Time Passcode Without Tokens

Extending the idea of using [possession of] your phone as an authentication factor.

Authenticating server can send Alice a one-time code via SMS.

- Alice logs in with her password and received code.

Often used for step-up authentication or account recovery.

- **Step-up authentication**: secondary [stronger] authentication mechanism invoked based on risk level

Examples: when attempting to access more sensitive resources, or when behavior pattern does not match routine.

- Similar solutions use email in place of SMS.

Proof that Alice has access to the email account she registered with.

Widespread use, but weaker against range of threat models
(SMS not very secure even against criminals... SIM swapping attacks)

Something You Are

Something You Are

Something unique identifying characteristic that only Alice has
(**biometrics**)

- Physical feature: fingerprint, iris print
- Behavioral characteristic: handwriting, typing
- Combination thereof: voice, gait

How do you know that I am the same person that was here last week?

- Did I provide a password?
- Did I provide a badge?

Pretty much all trust boils down to biometric authentication of one human by another

Biometrics

The only authentication factor that is not designed to be transferable

- Clear separation of authentication and authorization

Nothing to remember, nothing to carry around

Can be very strong differentiator

- Unique-ish

Biometrics

Fingerprint

Face recognition

Handprint

Retina

Iris

Vein

- vascular pattern in back of hand

Voiceprint

Signature (special pen)

Typing

- timing between character sequences

Gait recognition

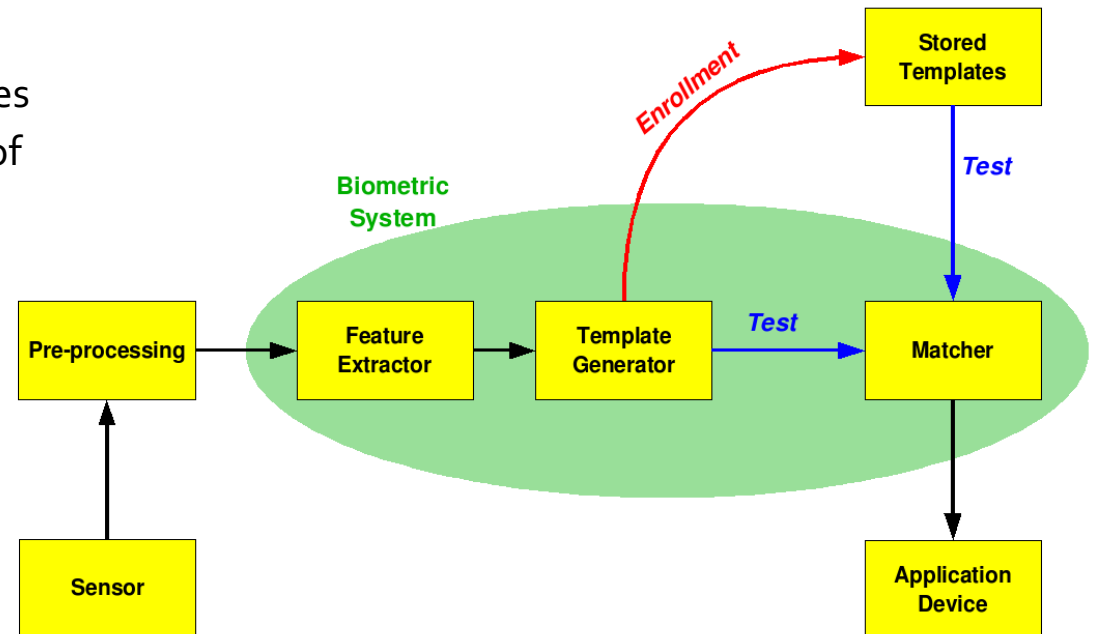
Heartbeat

DNA

Biometrics

General approach

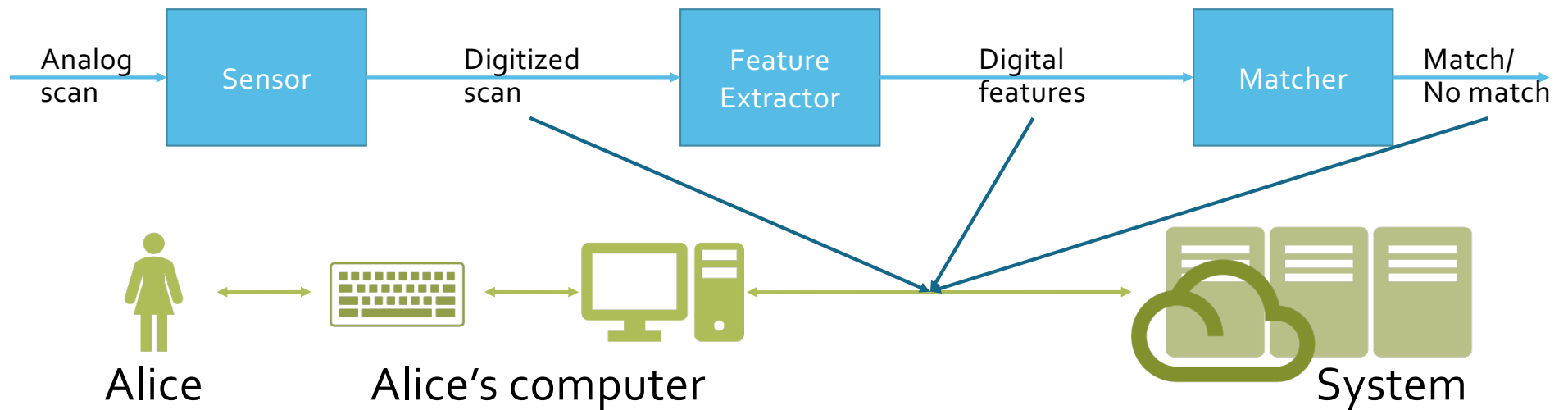
- Scan an analog sample
- Convert to set of digital features
- On enrollment save template of identifiable features
- On authentication, attempt approximate match against saved features



Biometrics

What happens in a remote authentication setting?

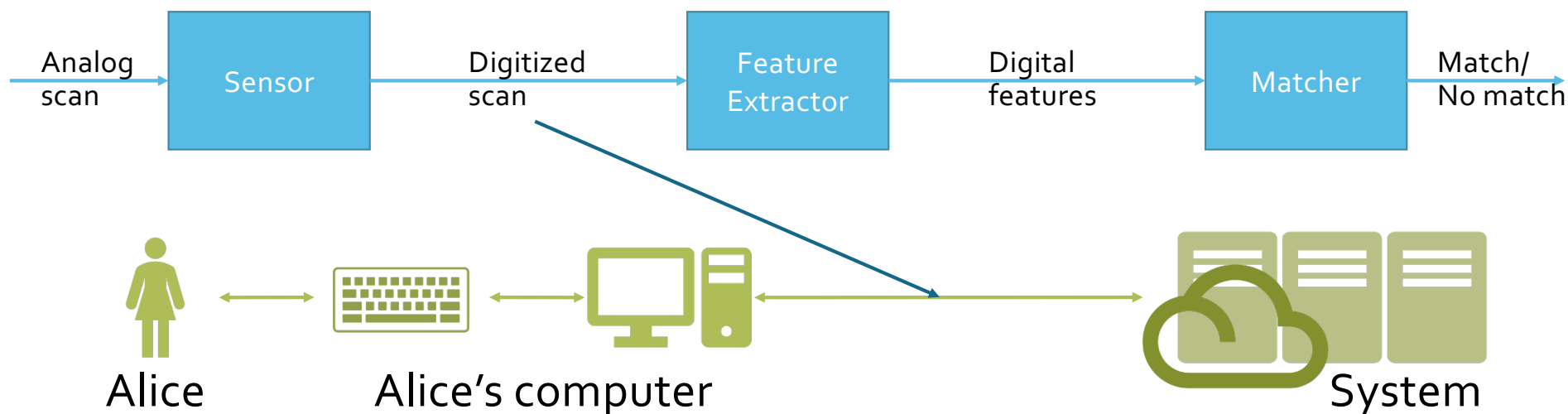
What does the authenticating system actually get?



Biometrics

Scenario A: only the sensor is local to user. Feature extraction and matching happen on authenticating system.

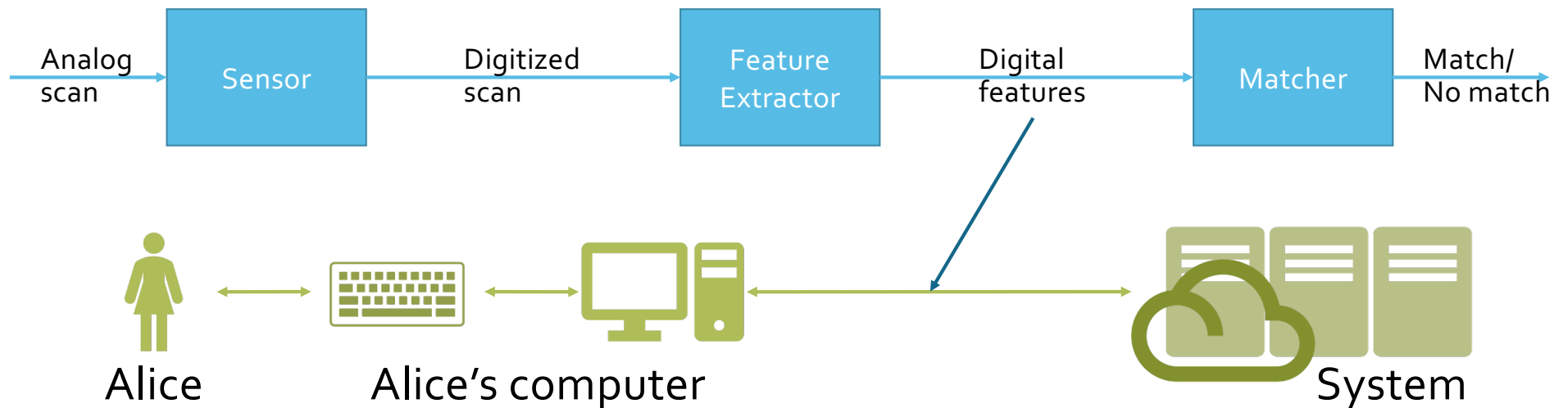
- Authenticating system has to trust Alice's computer to provide fresh, unspoofed sensor data.
- All biometric features and template data are on a central server.



Biometrics

Scenario B: Sensing and feature extraction are local to user.
Matching happens on authenticating system.

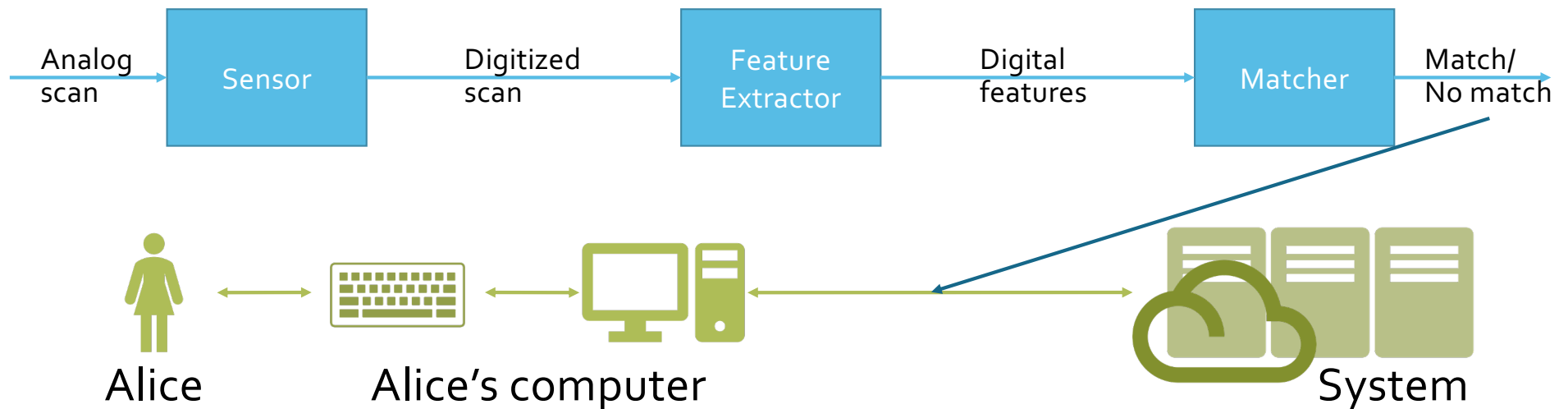
- Authenticating system has to trust Alice's computer to provide authentic, fresh, unspoofed feature data.
- All biometric features and template data are still on a central server.



Biometrics

Scenario C: Sensing, feature extraction, and matching are local to user. Only the result is communicated to the authenticating system.

- Authenticating system has to trust Alice's computer to perform authentication.
- All biometric features and template data are isolated on end users' devices.



Matching is not exact

Enrollment issues

- Unlike passwords, hard to pre-enroll user
- Users must be enrolled interactively
- For many biometrics, getting good accuracy requires multiple readings
 - Build templates and test against registration
 - Repeat
 - Some templates simply tough (e.g., smooth fingerprint); “goats”

What does accuracy in matching mean?

- False accept rate (FAR); lower FAR is harder to attack
- False reject rate (FRR); lower FRR is less finicky, easier to use
 - This is generally a tradeoff – if you want to reduce the probability of mismatching valid biometrics, you typically need to increase the probability of succeeding on invalid ones

Adversarial Biometrics Spoofing

Biometrics are private, but not secret

Users expose biometric instances everywhere

- Fingerprints, hand geometry, face, handwriting, iris, gait, etc.

Allows attacker to create biometric forgery

Very hard to replace a biometric identifier

Biometrics Spoofing

There are spoofing techniques for virtually all biometrics.

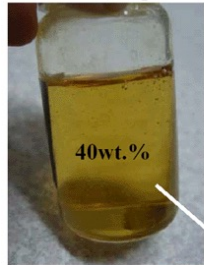


Live Image

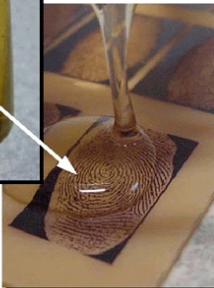


Cadaver Image

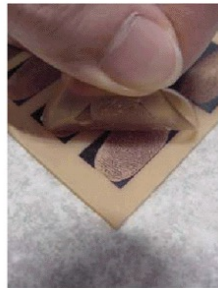
Gelatin Liquid



Drip the liquid
onto the mold.



Put this mold into
a refrigerator to cool,
and then peel carefully.



Chaos Computer Club



CCC researchers use iris image to breach Samsung Galaxy S8 scanner

🕒 May 23, 2017

Researchers from the Chaos Computer Club (CCC) have successfully breached the Samsung Galaxy S8's iris recognition system to unlock the...

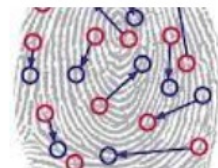
[Continue Reading](#)



Spoofing iris recognition technology with pictures

🕒 Mar 9, 2015

In a recent report by Forbes, Chaos Computer Club security researcher Jan "Starbug" Krissler highlighted the vulnerabilities behind some iris-scanning...



German researcher reverse-engineers a fingerprint using photos

🕒 Dec 30, 2014

A German researcher from European hackers association Chaos Computer Club recently demonstrated a method to fool standard biometric security software...



Chaos Computer Club claims Touch ID fake fingerprint spoof

🕒 Sep 23, 2013

Well that was fast. German hacker collective, Chaos Computer Club, has claimed that it has already spoofed the iPhone 5S's Touch...

Privacy Issues

Biometric identifier can track your physical activities as well as your virtual activities

- Some with crisp legal standing (fingerprint, DNA)

Easy to match (even if can't spoof)

Very hard to obscure



Review

3 types of authentication factors

- Password: ***Something you know***
- Token: ***Something you have***
- Biometrics: ***Something you are***

Each factor can be used independently, or combined for ***multi-factor*** authentication.

- Typically 2-factor

Use a ***slow salted hash*** to store passwords

- PBKDF2, bcrypt, or scrypt
- Don't make up your own!

Next time

