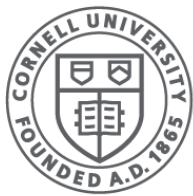


# Security

## CS 4410 Operating Systems

[E. Birrell, A. Bracy, E. Siroer, R. Van Renesse]



**Cornell CIS**  
COMPUTING AND INFORMATION SCIENCE

References: [Security Introduction](#) and [Access Control](#) by Fred Schneider

# Plan of Attack

- Protection
  - Authorization: *what are you permitted to do?*
  - Access Control Matrix
- Security
  - Authentication: *how do we know who you are?*
  - Threats and Attacks

# Authentication

Establish the identity of user/machine by

- **Something you are:**  
retinal scan, fingerprint
- **Something you have:**  
physical key, ticket, credit card, smart card
- **Something you know:**  
password, secret, answers to security questions, PIN

In the case of an OS this is done during login

- OS wants to know who the user is

# Multiple Factors

**Two-factor Authentication:** authenticate based on two independent methods

- ATM card + PIN
- password + secret Q
- password + registered cell phone



**Multi-factor Authentication:** two or more independent methods

Best to combine separate categories

- 2 passwords from a same person? arguably not independent

# Biometrics: something you are

- System has 2 components:
  - **Enrollment:** measure & store characteristics
  - **Identification:** match with user supplied values
- What are good characteristics?
  - Finger length, voice, hair color, retinal pattern, blood

**Pros:** user carries around a good password

**Cons:** difficult to change password, can be subverted

# Authentication with Physical Objects

**Door keys** have been around long

**Plastic cards** inserted into reader associated with comp

- Also a password known to user, to protect against lost card

**Magnetic stripe cards:** ~140 bytes info glued to card

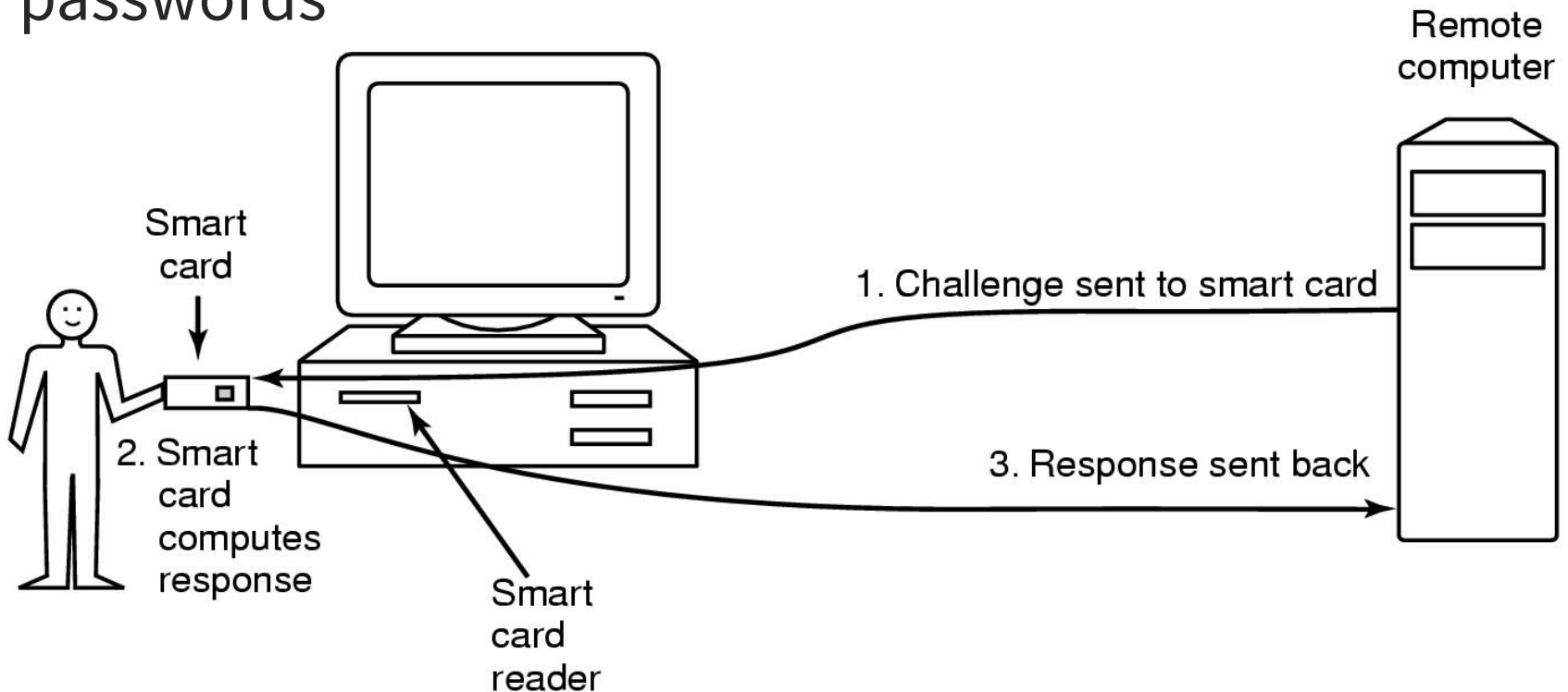
- Is read by terminal and sent to computer
- Info contains encrypted user password (only bank knows key)

**Chip cards:** have an integrated circuit

- Stored value cards: have EEPROM memory but no CPU
  - Value on card can only be changed by CPU on another comp
- Smart cards: 4 MHz 8-bit CPU, 16 KB ROM, 4 KB EEPROM, 512 bytes RAM, 9600 bps comm. channel

# Smart Cards

- Better security than stored value cards
  - Card sends a small encrypted msg. to merchant, who can later use it to get money from the bank
  - **Pros:** no online connection to bank required
- Perform local computations, remember long passwords



# Challenge Response Scheme

New user provides server with list of Q/A pairs

- Server asks one of them at random
- Requires a long list of question answer pairs

Prove identity by computing a secret function

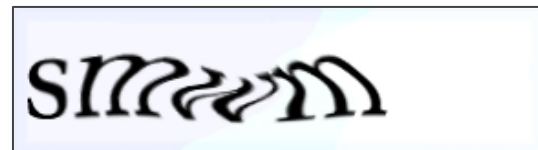
- User picks an algorithm, e.g.  $x^2$
- Server picks a challenge, e.g.  $x=7$
- User sends back 49
- Should be difficult to deduce function by looking at results

In practice

- Algorithm is fixed, e.g. one-way hash, but user selects a key
- The server's challenge is combined with user's key to provide input to the function

Authenticate yourself as a human:

CAPTCHA, image tasks, *etc.*





# Passwords

Secret known only to the subject

Top 10 passwords in 2017:

[SplashData]

- |             |              |
|-------------|--------------|
| 1. 123456   | 6. 123456789 |
| 2. password | 7. letmein   |
| 3. 12345678 | 8. 1234567   |
| 4. qwerty   | 9. football  |
| 5. 12345    | 10. iloveyou |

16: starwars, 18: dragon, 27: jordan23

Top 20 passwords suffice to compromise 10% of accounts

[Skyhigh Networks]

# Verifying Passwords

How does OS know that the password is correct?

Simplest implementation:

- OS keeps a file with ⟨login, password⟩ pairs
- User types password
- OS looks for a login → password match

Goal: make this scheme as secure as possible

- display the password when being typed?

# Storing Passwords

## 1. Store username/password in a file

- Attacker only needs to read the password file
- Security of system now depends on protection of this file!  
Need: perfect authorization & trusted system administrators



**Claudia Pellegrino**

@c\_pellegrino

Follow



# Does T-Mobile Austria in fact store customers' passwords in clear text @tmobileat? @PWTooStrong @Telekom\_hilft

**SeloX** @SeloX\_AUT

Replying to @c\_pellegrino @PWTooStrong @Telekom\_hilft

Had the same issue with T-Mobile Austria. Apparently they are saving the password in clear because employees have access to them (you have tell them your password when you're talking to them on the phone or in a shop) and they are not case sensitive

10:53 PM - 3 Apr 2018

908 Retweets 2,015 Likes



113 908 2.0K



**T-Mobile Austria** @tmobileat · Apr 3

Replying to @c\_pellegrino @PWTooStrong @Telekom\_hilft

Hello Claudia! The customer service agents see the first four characters of your password. We store the whole password, because you need it for the login for [mein.t-mobile.at](http://mein.t-mobile.at) ^Andrea

319 624 589



**Claudia Pellegrino** @c\_pellegrino · Apr 4

Thanks for your reply Andrea! Storing cleartext passwords in a database is a naughty thing to do. [plaintextoffenders.com/faq/devs](http://plaintextoffenders.com/faq/devs) What can we do to get your devs to fix that?



**Tumblr**

plaintextoffenders

4.0/5.0 stars - 381,908 ratings



**T-Mobile Austria** @tmobileat · Apr 4

Hi @c\_pellegrino, I really do not get why this is a problem. You have so many passwords for every app, for every mail-account and so on. We secure all data very carefully, so there is not a thing to fear. ^Käthe

319 370 353



**Eric™** @Korni22 · Apr 6

Well, what if your infrastructure gets breached and everyone's password is published in plaintext to the whole wide world?

5 87 1.8K



**T-Mobile Austria** @tmobileat · Apr 6

@Korni22 What if this doesn't happen because our security is amazingly good? ^Käthe

460 701 917



**Eric™** @Korni22 · Apr 6

Bad news for you Käthe, nobody's security is that good. No, not even yours. It's not that I say you are 100% getting hacked - what if an employee accesses the database directly?

10 84 2.7K



**Arkan Hadi** @c0derr0r · Apr 7

Pretty sure after this they will be penetrated in days if not hours, let's just hope its a white hat that have good intentions

1 3

# Storing Passwords

1. Store username/password in a file
  - Attacker only needs to read the password file
  - Security of system now depends on protection of this file!  
Need: perfect authorization & trusted system administrators
  
2. Store login/encrypted password in file
  - Access to password file  $\neq$  access to passwords

# Hashing

Want a function  $f$  such that:

1. Easy to compute and store  $h(p)$
2. Hard to compute  $p$  given  $h(p)$
3. Hard to find  $q$  such that  $q \neq p, h(q) = h(p)$

Cryptographic hash functions to the rescue!

$h(\text{password}) = \text{encrypted-password}$

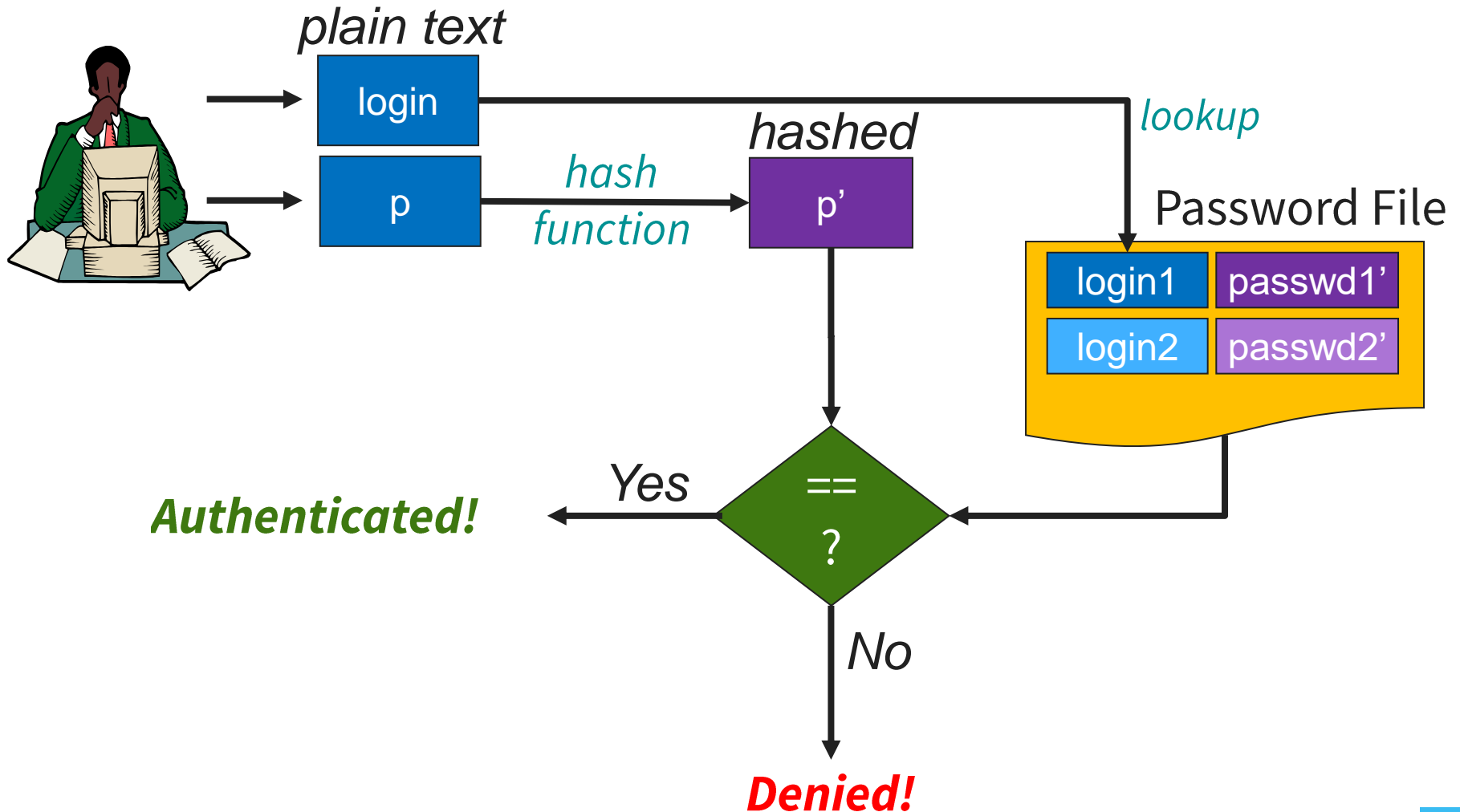
e.g., SHA

(but don't use SHA...)

- one-way property gives (1) and (2)
- collision resistance gives (3)

**Remember:**  $h(\text{encrypted-password}) \neq \text{password}$   
 $h^{-1}(\text{encrypted-password}) = \text{password}$   
 $h^{-1}$  hard to compute (hard  $\approx$  impossible)

# Storing and Checking Passwords

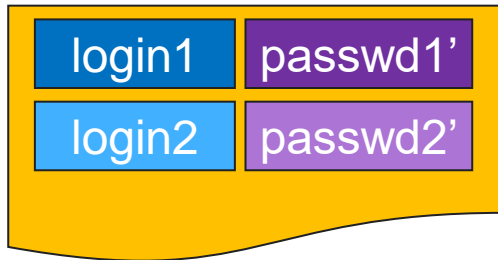


# Hashed passwords still vulnerable

Suppose attacker obtains password file:

/etc/passwd public, known hash fn known  
+ hard to invert → hard to obtain **all** the passwords

Password File



login1	passwd1'
login2	passwd2'

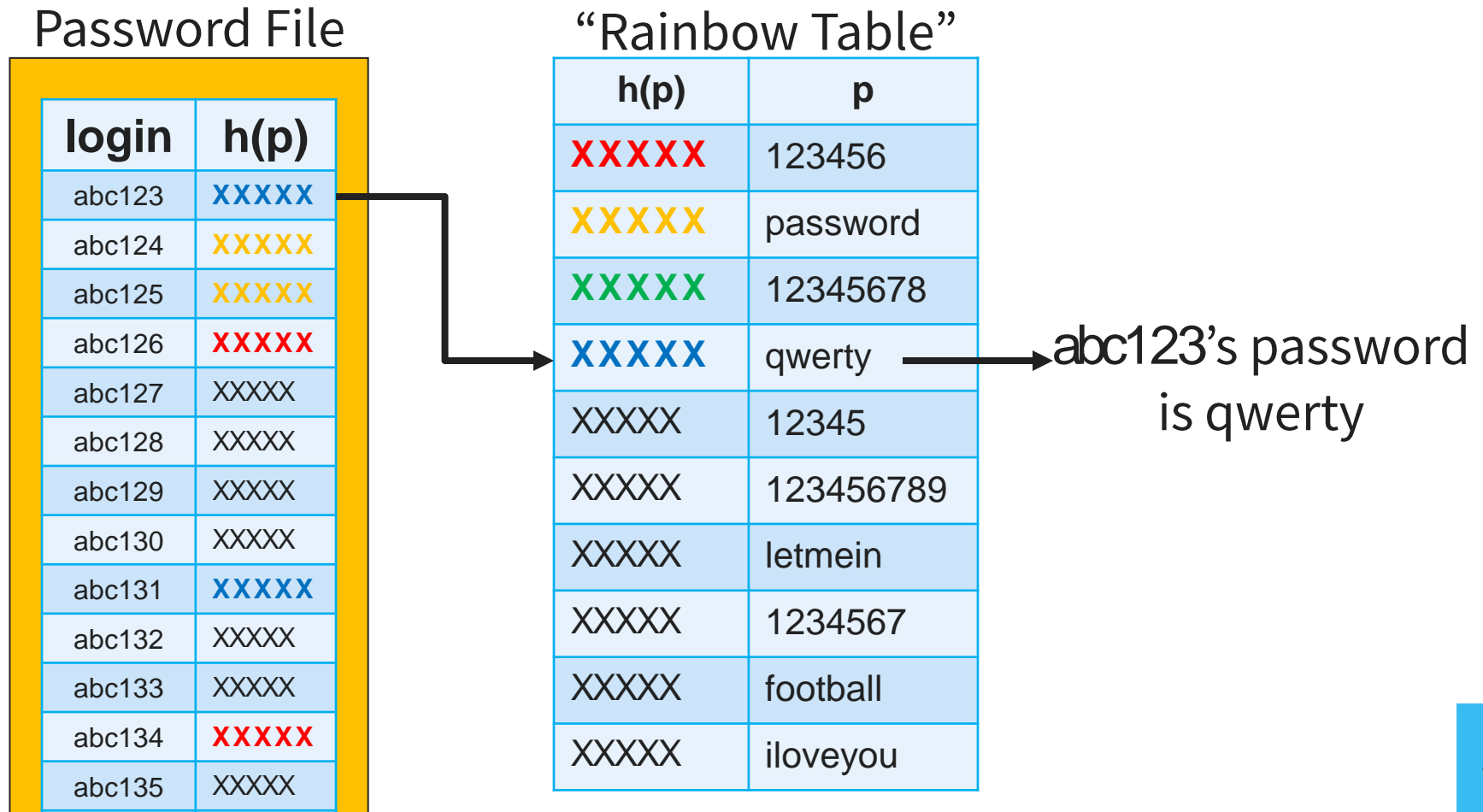
How else can I crack this file?

- Brute Force Attack:
  - Enumerate all **possible** passwords  $p$ , calculate  $h(p)$  and see if it matches an entry in the file
- Dictionary Attack
  - List all the **likely** passwords  $p$ , calculate  $h(p)$  and check for a match. (recall: top 20 passwords can compromise 10% of accounts)



# Rainbow Table Attack

- Pre-compute the dictionary hashes (need space, not time), use hashed passwords as key
- Quick attack: look up each hashed password 1-by-1



# Salting

## **Vulnerabilities:**

- single dictionary compromises all users
- passwords chosen from small space

**Countermeasure:** include a **unique system-chosen nonce** as part of each user's password

- make every user's stored hashed password different, even if they chose the same password
- now passwords come from a larger space

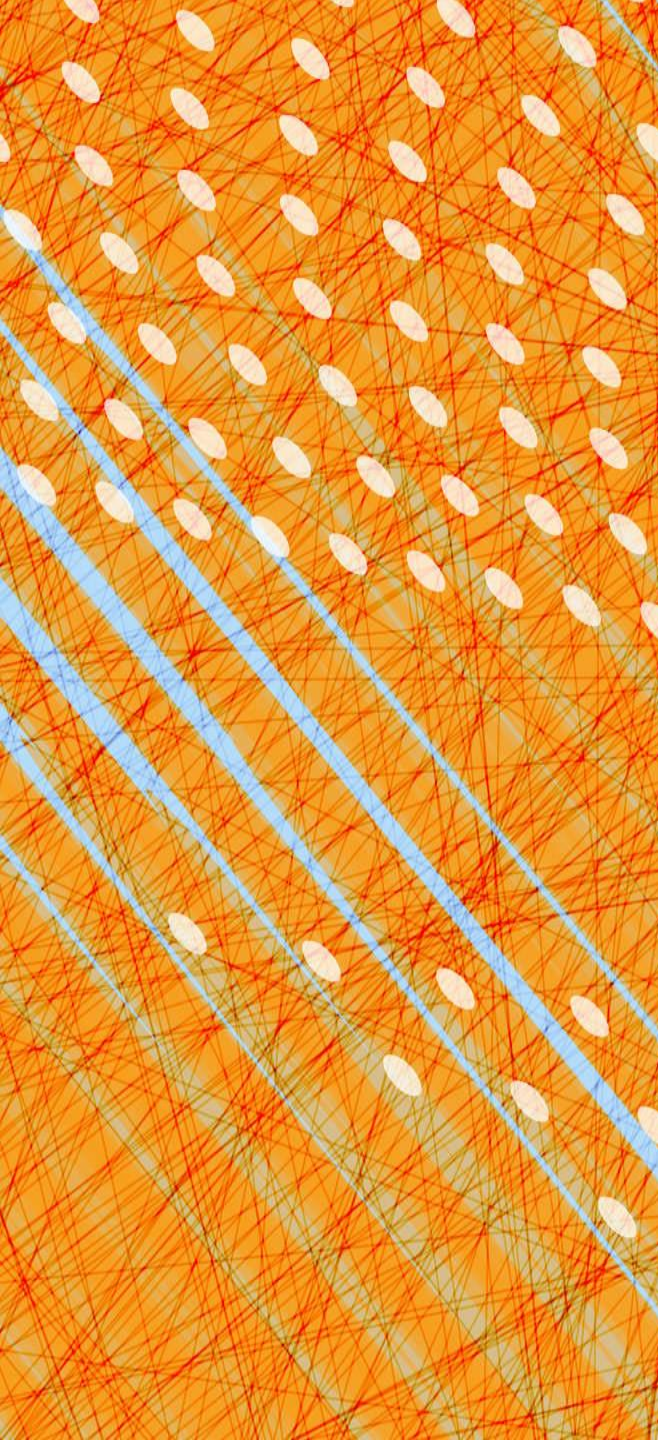
Each user has: **login**, unique salt **s**, passwd **p**

System stores: login, s,  $H(s, p)$

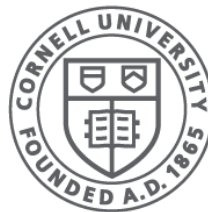
# Salting Example

login	salt	h(p  s)
abc123	4238	h(423812345)
abc124	2918	h(2918password)
abc125	6902	h(6902LordByron)
abc126	1694	h(1694qwerty)
abc127	1092	h(109212345)
abc128	9763	h(97636%%TaeFF)
abc129	2020	h(2020letmein)

- If the hacker guesses `qwerty`, has to try: `h(0001qwerty)`, `h(0002qwerty)`, `h(0003qwerty)` ...
- UNIX adds 12-bit of salt
- Also, passwords should be secure:
  - Length, case, digits, not from dictionary
  - Can be imposed by the OS! This has its own tradeoffs



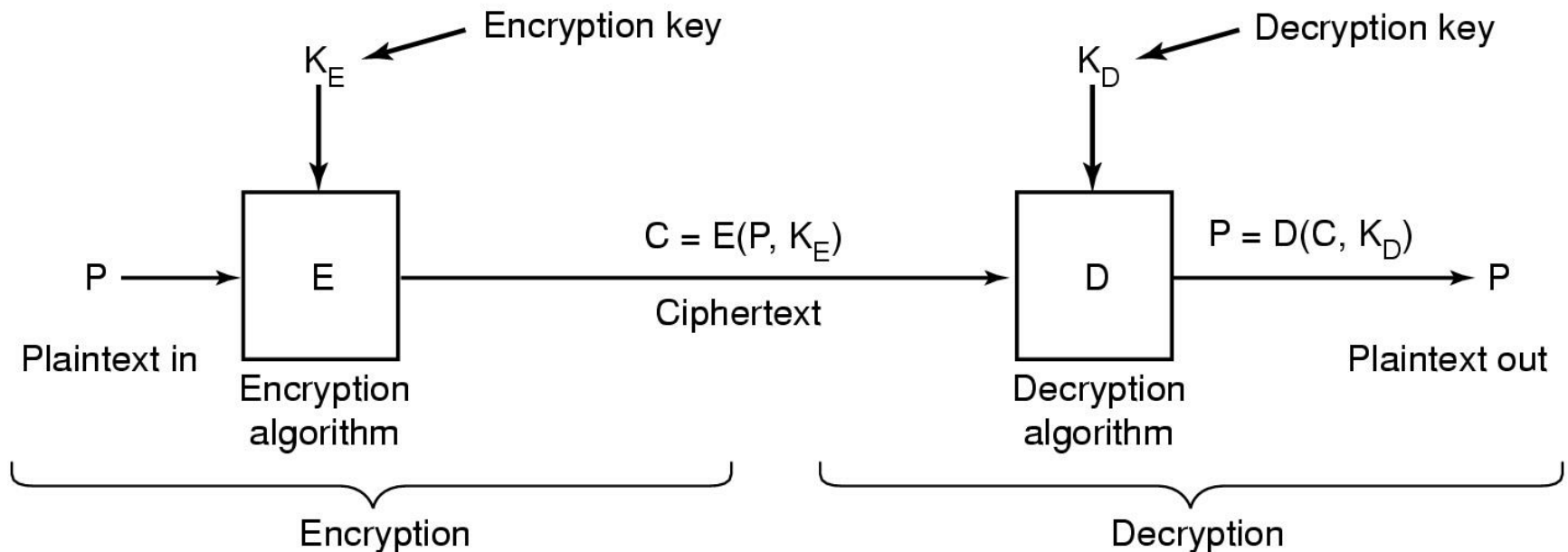
# Cryptography



**Cornell CIS**  
COMPUTING AND INFORMATION SCIENCE

# Cryptography Overview

- Encrypt data so it only makes sense to authorized users
  - Input data is a message or file called plaintext
  - Encrypted data is called ciphertext
- Encryption and decryption functions should be public
  - Security by obscurity is not a good idea!



# Secret-Key Cryptography

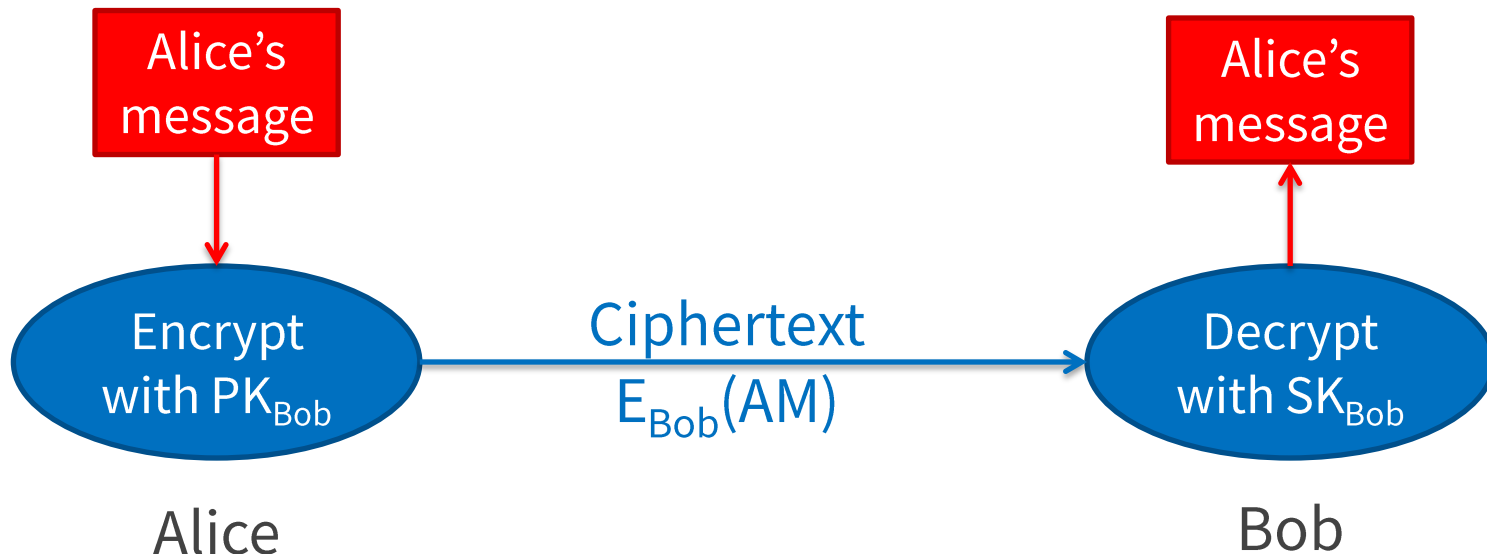
- Also called *symmetric cryptography*
  - Encryption algorithm is publicly known
  - $E(\text{message}, \text{key}) = \text{ciphertext}$   
 $D(\text{ciphertext}, \text{key}) = \text{message}$
- Naïve scheme: monoalphabetic substitution
  - Plaintext : ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Ciphertext: QWERTYUIOPASDFGHJKLZXCVBNM
  - So, *attack* is encrypted to: *qzzqea*
  - 26! possible keys  $\sim 4 \times 10^{26}$  possibilities
    - 1  $\mu\text{s}$  per permutation  $\Rightarrow$  10 trillion years to break
  - easy to break this scheme! How?
    - 'e' occurs 14%, 't' 9.85%, 'q' 0.26%

# Symmetric Key Cryptography

- Which encryption algorithm is good?
  - DES was proposed in the 1970s
    - Encrypts 64 bits of data with 56 bit key to give 64-bit ciphertext
    - Uses 16 rounds of substitution and permutation
    - EFF invested \$250000 to break DES message in 56 hours
    - DES made powerful by encrypting message 3 times (DES3)
  - Current standard is AES
    - A result of 3-year competition with entries from 12 countries
    - Winning entry was from Belgium, called 'Rijndael'

# Public Key Cryptography

- Diffie and Hellman, 1976
- All users get a public key and a private key
  - Public key is published
  - Private key is not known to anyone else
- If Alice has a packet to send to Bob,
  - She encrypts the packet with Bob's public key
  - Bob uses his private key to decrypt Alice's packet





# Public Key Cryptography

- Diffie and Hellman, 1976
- All users get a public key and a private key
  - Public key is published
  - Private key is not known to anyone else
- If Alice has a packet to send to Bob,
  - She encrypts the packet with Bob's public key
  - Bob uses his private key to decrypt Alice's packet
- Private key linked mathematically to public key
  - Make it computationally infeasible to derive (RSA)
- Pros: more security, convenient, digital signatures
- Cons: slower

# Digital Signatures

- Choose hashing function hard to invert, e.g. MD5, SHA
- Hash document to sign
- Apply **private** key to hash (decrypt hash)
  - Called signature block
- Receiver uses sender's **public** key on signature block
  - $E(D(x)) = x$  should work (works for RSA)
- If signature block “encrypts” to hash of document, signature is valid

# Digital Signatures

