# Security in Operating Systems
# CS 111
# Operating System Principles

Peter Reiher

# Outline

- Security goals
- Access control
- Cryptography
  - Symmetric cryptography
  - Asymmetric cryptography

# Security Goals

- Confidentiality
  - If it's supposed to be secret, be careful who hears it
- Integrity
  - Don't let someone change something they shouldn't
- Availability
  - Don't let someone stop others from using services
- Exclusivity
  - Don't let someone use something he shouldn't
- Note that we didn't mention "computers" here
  - This classification of security goals is very general

# Access Control

- Security could be easy
  - If we didn't want anyone to get access to anything
- The trick is giving access to only the right people
- How do we ensure that a given resource can only be accessed by the proper people?
- The OS plays a major role in enforcing access control

# Common Mechanisms for Access Control in Operating Systems

- Access control lists
  - Like a list of who gets to do something
- Capabilities
  - Like a ring of keys that open different doors
- They have different properties
- And are used by the OS in different ways

# The Language of Access Control

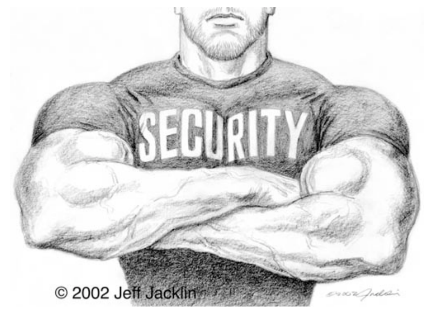
- *Subjects* are active entities that want to gain access to something
  - E.g., users or programs
- *Objects* represent things that can be accessed
  - E.g., files, devices, database records
- *Access* is any form of interaction with an object
- An entity can be both subject and object

# Access Control Lists

- ACLs
- For each protected object, maintain a single list
- Each list entry specifies a subject who can access the object
  - And the allowable modes of access
- When a subject requests access to a object, check the access control list

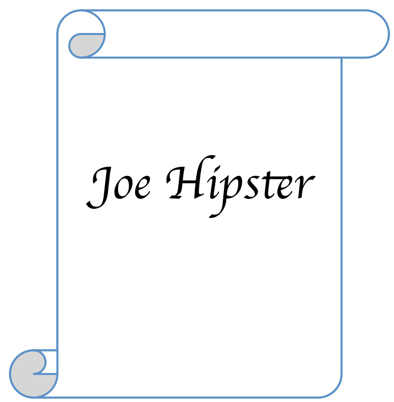
# An Analogy

**You're Not On the List!**





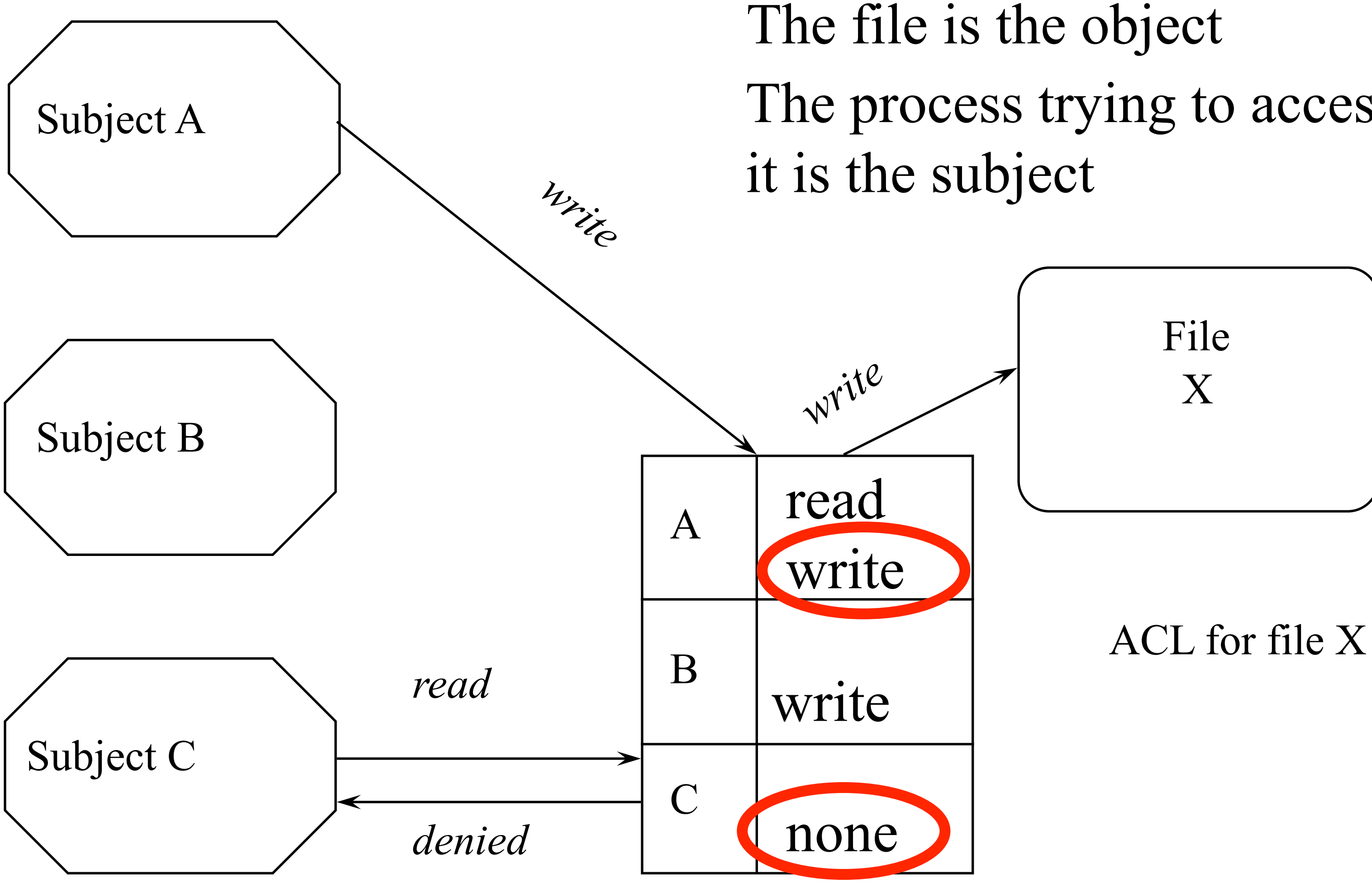
This is an access control list



# An ACL Protecting a File

The file is the object

The process trying to access it is the subject



# Issues For Access Control Lists

- How do you know the requestor is who he says he is?
- How do you protect the access control list from modification?
- How do you determine what resources a user can access?

# An Example Use of ACLs: the Unix File System

- An ACL-based method for protecting files
  - Developed in the 1970s
- Still in very wide use today
  - With relatively few modifications
- Per-file ACLs (files are the objects)
- Three subjects on list for each file
    - Owner, group, other
- And three modes
  - Read, write, execute
  - Sometimes these have special meanings

# Storing the ACLs

- They can be very small
  - Since there are only three entries
  - Basic ACL is only 9 bits
- Therefore, kept inside the file descriptor
- Makes it easy to find them
  - Since trying to open the file requires the file descriptor, anyway
- Checking this ACL is not much more than a logical AND with the requested access mode

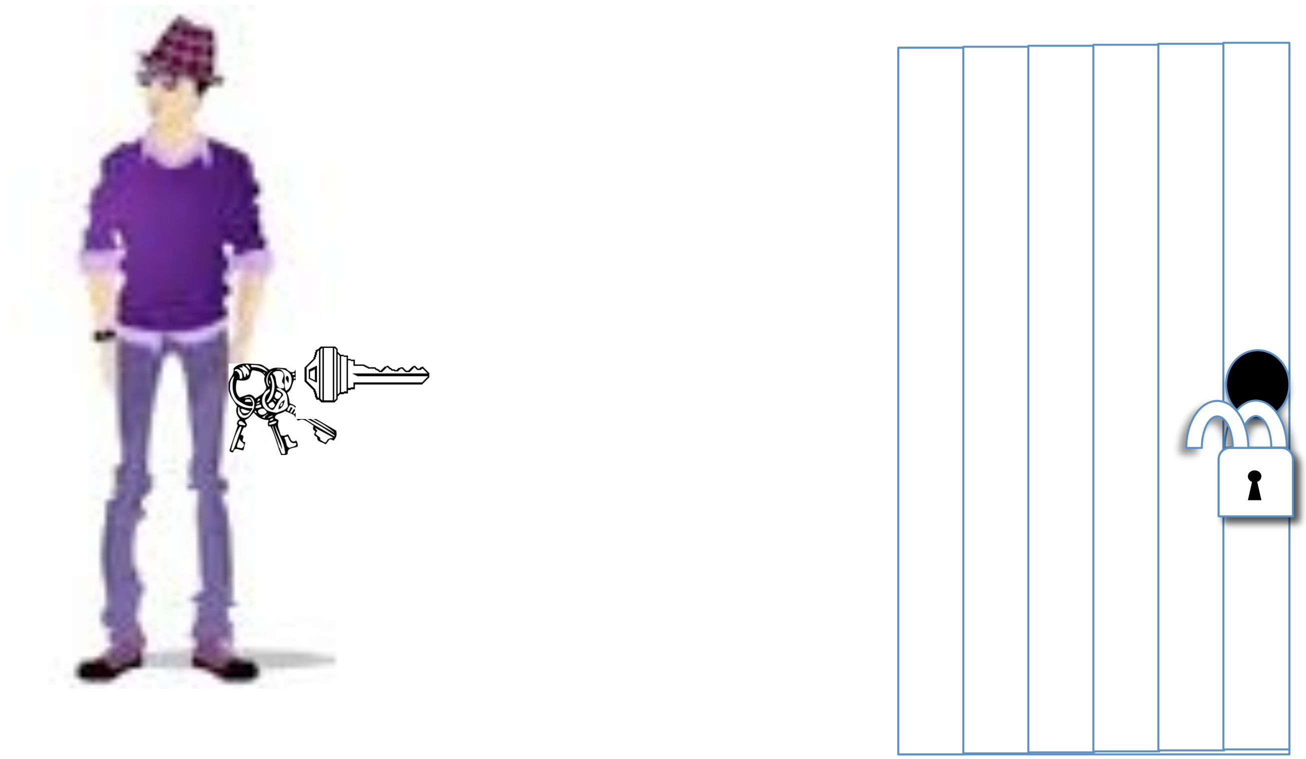
# Pros and Cons of ACLs

+ Easy to figure out who can access a resource
+ Easy to revoke or change access permissions
– Hard to figure out what a subject can access
– Changing access rights requires getting to the object

# Capabilities

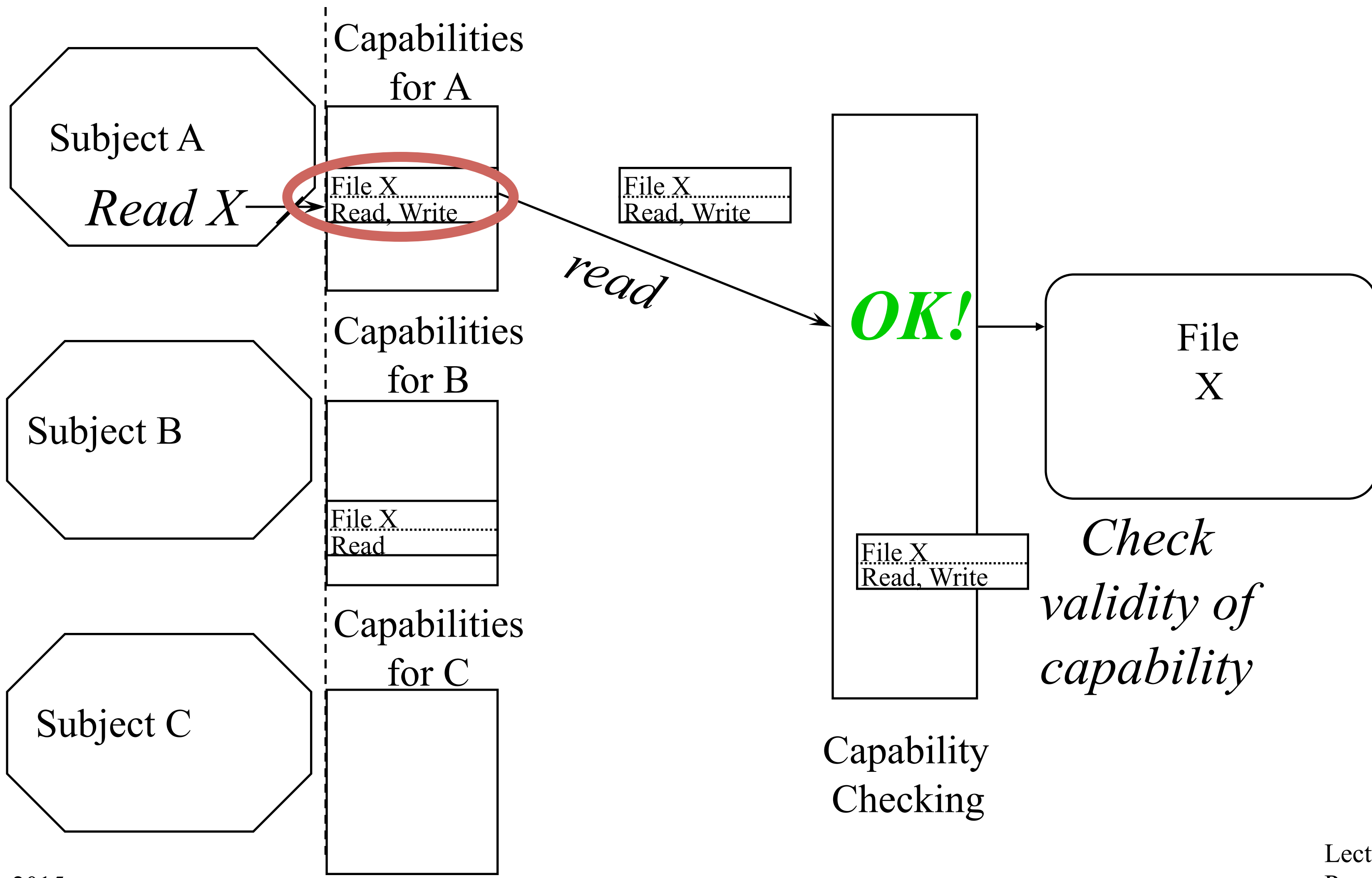
- Each subject keeps a set of data items that specify his allowable accesses
- Essentially, a set of tickets
- To access an object, present the proper capability
- Possession of the capability for an object implies that access is allowed

# An Analogy

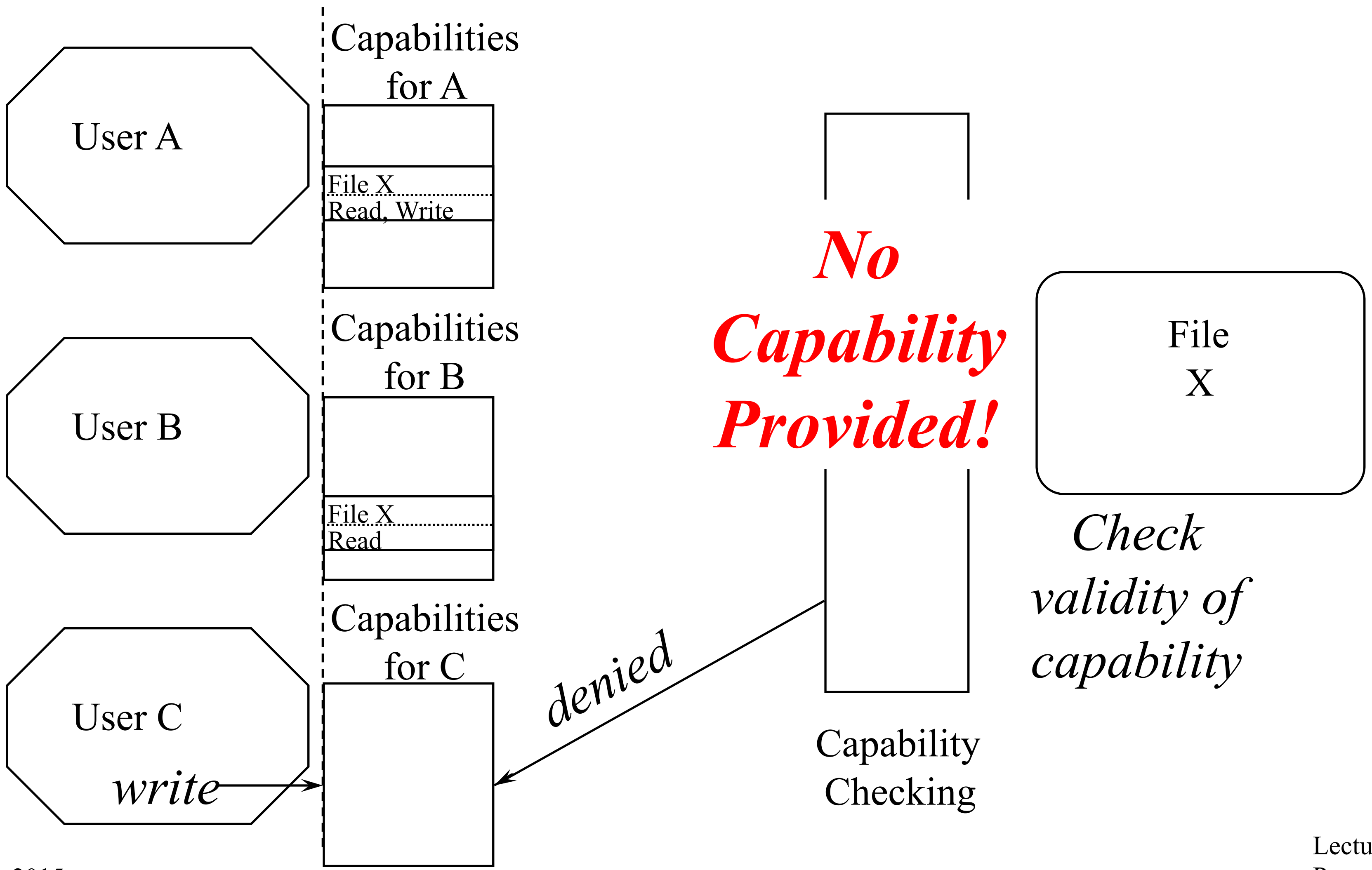


The key is a capability

# Capabilities Protecting a File



# Capabilities Denying Access



# Properties of Capabilities

- Capabilities are essentially a data structure
  - Ultimately, just a collection of bits
- Merely possessing the capability grants access
  - So they must not be forgeable
- How do we ensure unforgeability for a collection of bits?
- One solution:
  - Don't let the user/process have them
  - Store them in the operating system

# Revoking Capabilities

- A simple problem for capabilities stored in the operating system
  - Just have the OS get rid of it
- Much harder if it's not in the operating system
  - E.g., in a network context
- How do we make the bundle of bits change from valid to invalid?
- Consider the real world problem of a door lock
- If several people have the key, how do we keep one of them out?

# Pros and Cons of Capabilities

+ Easy to determine what objects a subject can access
+ Potentially faster than ACLs (in some circumstances)
+ Easy model for transfer of privileges
– Hard to determine who can access an object
– Requires extra mechanism to allow revocation
– In network environment, need cryptographic methods to prevent forgery

# Cryptography

- Much of computer security is about keeping secrets
- One method of doing so is to make it hard for others to read the secrets
- While (usually) making it simple for authorized parties to read them
- That's what cryptography is all about

# What Is Encryption?

- Encryption is the process of hiding information in plain sight
- Transform the secret data into something else
- Even if the attacker can see the transformed data, he can't understand the underlying secret
- Usually, someone you want to understand it can

# Cryptography Terminology

- Typically described in terms of sending a message
  - Though it's used for many other purposes
- The sender is *S*
- The receiver is *R*
- *Encryption* is the process of making message unreadable/unalterable by anyone but *R*
- *Decryption* is the process of making the encrypted message readable by *R*
- A system performing these transformations is a *cryptosystem*
  - Rules for transformation sometimes called a *cipher*

# Plaintext and Ciphertext

- *Plaintext* is the original form of the message (often referred to as *P*)

| |
|---|
| Transfer $100 to my savings account |

- *Ciphertext* is the encrypted form of the message (often referred to as *C*)

| |
|---|
| Sqzmredq #099 sn lx rzuhmfr zbbntms |

# Cryptographic Keys

- Most cryptographic algorithms use a *key* to perform encryption and decryption
  - Referred to as *K*
- The key is a secret
- Without the key, decryption is hard
- With the key, decryption is easy
- Reduces the secrecy problem from your (long) message to the (short) key
  - But there's still a secret

# More Terminology

- The encryption algorithm is referred to as *E()*
- $C = E(K,P)$
- The decryption algorithm is referred to as *D()*
- The decryption algorithm also has a key
- The combination of the two algorithms are often called a *cryptosystem*

# Symmetric and Asymmetric Cryptosystems

- Symmetric cryptosystems use the same keys for E and D :

  $P = D(K, C)$

  – Expanding, $P = D(K, E(K,P))$
- Asymmetric cryptosystems use different keys for E and D:

  $C = E(K_E, P)$

  $P = D(K_D, C)$

  – Expanding, $P = D(K_D , E(K_E ,P))$

# Desirable Characteristics of Keyed Cryptosystems

- If you change only the key, a given plaintext encrypts to a different ciphertext
- Same applies to decryption
- Changes in the key ideally should cause unpredictable changes in the ciphertext
- Decryption should be hard without knowing the key
- The less a given key is used, the better (in security terms)

# Cryptography and Operating Systems

- Cryptography doesn't solve all of an OS' security problems
- But it helps with many:
  - Secrecy
    - Encrypt data you don't want to lose
  - Integrity
    - Encrypt data you don't want to change
  - Authentication
    - Use crypto as part of your authentication mechanism

# Symmetric Cryptosystems

- $C = E(K,P)$
- $P = D(K,C)$
- *E()* and *D()* are not necessarily the same operations

# Advantages of Symmetric Cryptosystems

+ Encryption and authentication performed in a single operation
+ Well-known (and trusted) ones perform much faster than asymmetric key systems
+ No centralized authority required
  - Though key servers help a lot

# Disadvantages of Symmetric Cryptosystems

- Encryption and authentication performed in a single operation
  - Makes signature more difficult
- Non-repudiation hard without servers
- Key distribution can be a problem
- Scaling
  - Especially for Internet use

# Some Popular Symmetric Ciphers

- The Data Encryption Standard (DES)
  - The old US encryption standard
  - Still fairly widely used, due to legacy
  - Weak by modern standards
- The Advanced Encryption Standard (AES)
  - The current US encryption standard
  - Probably the most widely used cipher
- Blowfish
- There are many, many others

# Symmetric Ciphers and Brute Force Attacks

- If your symmetric cipher has no flaws, how can attackers crack it?
- *Brute force* – try every possible key until one works
- The cost of brute force attacks depends on key length
  - Assuming random choice of key
  - For N possible keys, attack must try N/2 keys, on average, before finding the right one

# How Long Are the Keys?

- DES used 56 bit keys
  - Brute force attacks on that require a lot of time and resources
  - But they are demonstrably possible
  - Attackers can thus crack DES, if they really care
- AES uses either 128 bit or 256 bit keys
  - Even the shorter key length is beyond the powers of brute force today
  - $2^{127}$ decryption attempts is still a lot, by any standard

# Asymmetric Cryptosystems

- Often called *public key cryptography*
  - Or PK, for short
- The encrypter and decrypter have different keys
  - $C = E(K_E,P)$
  - $P = D(K_D,C)$
- Often works the other way, too
  - $C' = E(K_D,P)$
  - $P = D(K_E,C')$

# Using Public Key Cryptography

- Keys are created in pairs
- One key is kept secret by the owner
- The other is made public to the world
  - Hence the name
- If you want to send an encrypted message to someone, encrypt with his public key
  - Only he has private key to decrypt

# Authentication With Public Keys

- If I want to "sign" a message, encrypt it with my private key
- Only I know private key, so no one else could create that message
- Everyone knows my public key, so everyone can check my claim directly
- Much better than with symmetric crypto
  - The receiver could not have created the message
  - Only the sender could have

# PK Key Management

- To communicate via shared key cryptography, key must be distributed
  - In trusted fashion
- To communicate via public key cryptography, need to find out each other's public key
  - "Simply publish public keys"
- Not really that simple, for most cases

# Issues With PK Key Distribution

- Security of public key cryptography depends on using the right public key
- If I am fooled into using wrong one, that key's owner reads my message
- Need high assurance that a given key belongs to a particular person
  - Either a *key distribution infrastructure*
  - Or use of *certificates*
- Both are problematic, at high scale and in the real world

# The Nature of PK Algorithms

- Usually based on some problem in mathematics
  - Like factoring extremely large numbers
- Security less dependent on brute force
- More on the complexity of the underlying problem

# Choosing Keys for Asymmetric Ciphers

- For symmetric ciphers, the key can be any random number of the right size
  - You can't do that for asymmetric ciphers
- Only some public/private key pairs "work"
  - Generally, finding a usable pair takes a fair amount of time
  - E.g., for RSA you perform operations on 100-200 digit prime numbers to get keys
- You thus tend to use one public/private key pair for a long time
  - Issues of PK key distribution and typical usage also suggest long lifetimes for these keys

# Example Public Key Ciphers

- RSA
  - The most popular public key algorithm
  - Used on pretty much everyone's computer, nowadays
- Elliptic curve cryptography
  - An alternative to RSA
  - Tends to have better performance
  - Not as widely used or studied

# Security of PK Systems

- Based on solving the underlying problem
  - E.g., for RSA, factoring large numbers
- In 2009, a 768 bit RSA key was successfully factored
- Research on integer factorization suggests keys up to 2048 bits may be insecure
  - In 2013, Google went from 1024 to 2048 bit keys
- Size will keep increasing
- The longer the key, the more expensive the encryption and decryption

# Combined Use of Symmetric and Asymmetric Cryptography

- Very common to use both in a single session
- Asymmetric cryptography essentially used to “bootstrap” symmetric crypto
- Use RSA (or another PK algorithm) to authenticate and establish a *session key*
- Use DES or AES with session key for the rest of the transmission

# For Example,

Alice

$K_{EA}$ $K_{DA}$

$K_{DB}$

Alice wants to share $K_S$ only with Bob

Bob wants to be sure it's Alice's key

Only Bob can decrypt it

Only Alice could have created it

Bob

$K_{EB}$ $K_{DB}$

$K_{DA}$

$K_S$

$C=E(K_S,K_{DB})$

$M=E(C,K_{EA})$

$C=D(M,K_{DA})$

$K_S=D(C,K_{EB})$

# Authentication for Operating Systems

- What is authentication?
- How does the problem apply to operating systems?
- Techniques for authentication in operating systems

# What Is Authentication?

- Determining the identity of some entity
  - Process
  - Machine
  - Human user
- Requires notion of identity
  - One implication is we need some defined name space
- And some degree of proof of identity

# Where Do We Use Authentication in the OS?

- Typically users authenticate themselves to the system
- Their identity tends to be tied to the processes they create
  - OS can keep track of this easily
- Once authenticated, users (and their processes) typically need not authenticate again
  - One authentication per session, usually
- Distributed systems greatly complicate things

# Authentication Mechanisms

- Something you know
  - E.g., passwords
- Something you have
  - E.g., smart cards or tokens
- Something you are
  - Biometrics
- Somewhere you are
  - Usually identifying a role

# Passwords

- Authentication by what you know
- One of the oldest and most commonly used security mechanisms
- Authenticate the user by requiring him to produce a secret
  - Usually known only to him and to the authenticator

# Problems With Passwords

- They have to be unguessable
  - Yet easy for people to remember
- If sent over the network, susceptible to password sniffers
- Unless fairly long, brute force attacks often work on them

# Handling Passwords

- The OS must be able to check passwords when users log in
- So must the OS store passwords?
- Not really
  - It can store an encrypted version
- Encrypt the offered password
  - Using a *one-way function*
  - E.g., a secure hash algorithm like SHA1
- And compare it to the stored version
- Real security requires a little more

# Is Encrypting the Password File Enough?

- What if an attacker gets a copy of your password file?
- No problem, the passwords are encrypted
  - Right?
- Yes, but . . .

# Dictionary Attacks

| | |
|---|---|
| Harpo | 2st6'sG0 |
| Zeppo | G>I5{as3 |
| Chico | |
| Karl | |
| Groucho | |
| Gummo | 3(;wbnP] |

sY(34,ee


sY(34,ee

abaca is Karl Marx's password!

**Rats!!!!**

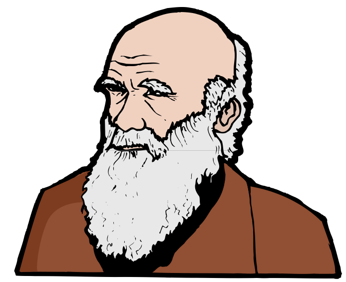
Now you can hack the Communist Manifesto!

# Salted Passwords

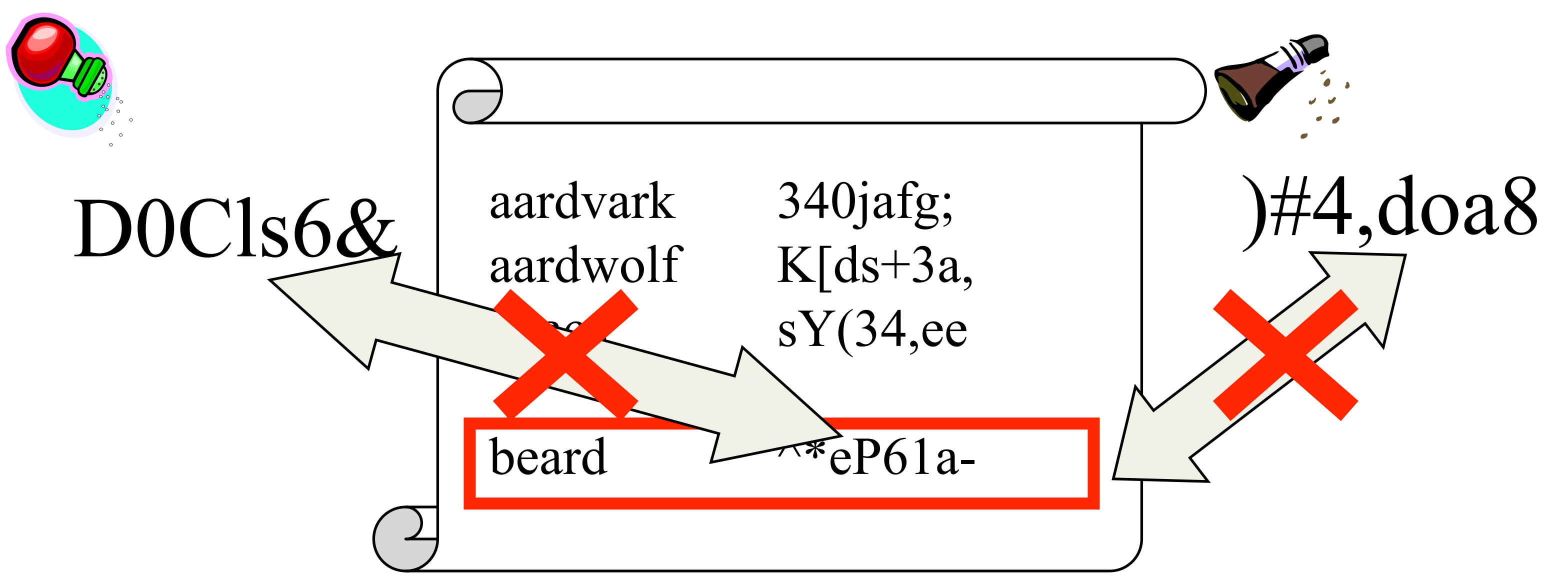
- A technique to combat dictionary attacks
- Combine the plaintext password with a random number
  - Then run it through the one-way function
- The random number need not be secret
- It just has to be different for different users
- You store the salt integer with the password
  - Generally in plaintext

# Did It Fix Our Problem?

Karl Marx



Charles Darwin



# Are My Passwords Safe Now?

- If I salt and encrypt them, am I OK?
- Depends on the quality of the passwords chosen
- Attacker can still perform dictionary attacks on an individual password, with its salt
- If the password isn't in the dictionary, no problem
- If it is, the attack succeeds
- Which is why password choice is important

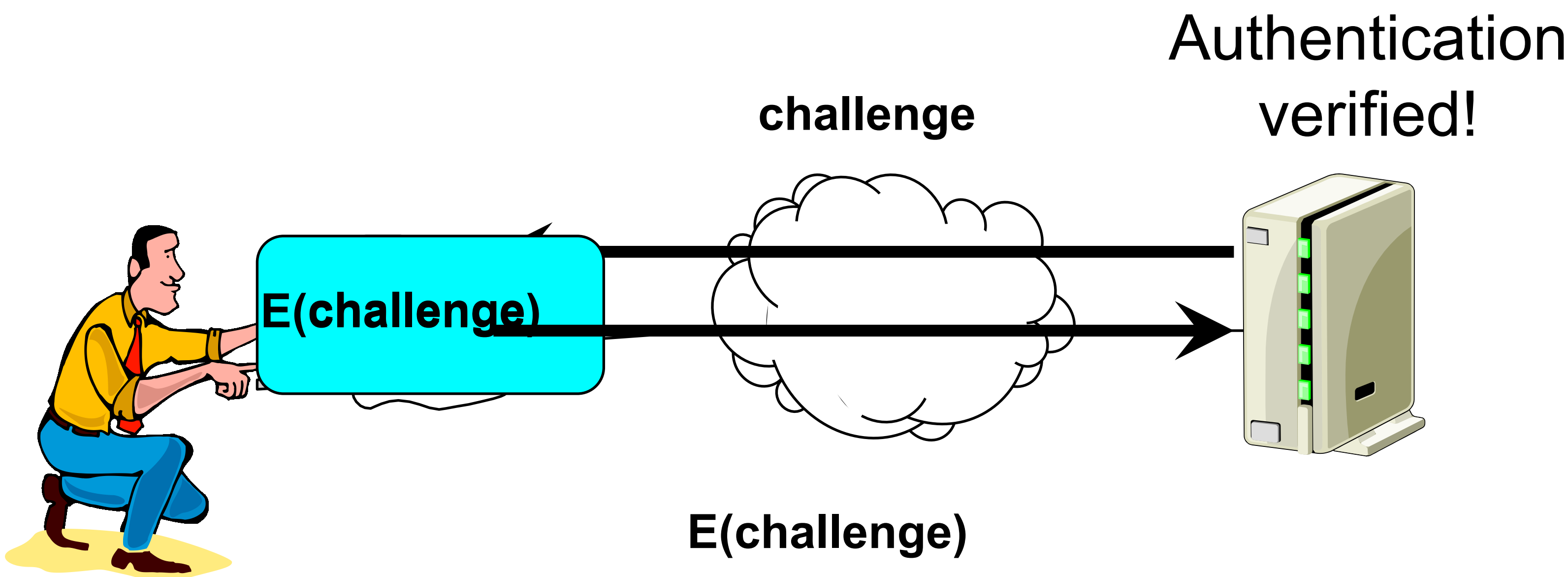
# Password Selection

- Generally, long passwords chosen from large character sets are good
- Short passwords chosen from small character sets are bad
- How long?
  - A matter of time
  - Moore's law forces us to make them ever longer
- What's a large character set?
  - Upper and lower case letters, plus numbers, plus symbols (like ^ and @)

# Authentication Devices

- Authentication by what you have
- A smart card or other hardware device that is readable by the computer
  - Safest if device has some computing capability
  - Rather than just data storage
- Authenticate by providing the device to the computer
- More challenging when done remotely, of course

# Authentication With Smart Cards




How can the server be sure of the remote user's identity?

By proper use of cryptography

# Problems With Authentication Devices

- If lost or stolen, you can't authenticate yourself
  - And maybe someone else can
  - Often combined with passwords to avoid this problem
- Unless cleverly done, susceptible to sniffing attacks
- Requires special hardware
- There have been successful attacks on some smart cards

# Biometric Authentication

- Authentication based on who you are
- Things like fingerprints, voice patterns, retinal patterns, etc.
- To authenticate, allow the system to measure the appropriate physical characteristics
- Biometric measurement converted to binary and compared to stored values
  – With some level of match required

# Problems With Biometrics

- Requires very special hardware
- May not be as foolproof as you think
- Many physical characteristics vary too much for practical use
  - Day to day or over long periods of time
- Generally not helpful for authenticating programs or roles
- What happens when it's cracked?
  - You only have two retinas, after all