Operating System Principles: Security and Privacy
CS 111
Operating Systems

Peter Reiher

Outline

- Introduction
- Authentication
- Access control
- Cryptography

Introduction

- Operating systems provide the lowest layer of software visible to users
- Operating systems are close to the hardware
  - Often have complete hardware access
- If the operating system isn't protected, the machine isn't protected
- Flaws in the OS generally compromise all security at higher levels

Why Is OS Security So Important?

- The OS controls access to application memory
- The OS controls scheduling of the processor
- The OS ensures that users receive the resources they ask for
- If the OS isn't doing these things securely, practically anything can go wrong
- So almost all other security systems must assume a secure OS at the bottom

Some Important Definitions

- Security
- Protection
- Vulnerabilities
- Exploits
- Trust
- Authentication and authorization

Security and Protection

- *Security* is a policy
  - E.g., “no unauthorized user may access this file”
- *Protection* is a mechanism
  - E.g., “the system checks user identity against access permissions”
- Protection mechanisms implement security policies

Vulnerabilities and Exploits

- A *vulnerability* is a weakness that can allow an attacker to cause problems
  - Not all vulnerabilities can cause all problems
  - Most vulnerabilities are never exploited
- An *exploit* is an actual incident of taking advantage of a vulnerability
  - Allowing attacker to do something bad on some particular machine
  - Term also refers to the code or methodology used to take advantage of a vulnerability

Trust

- An extremely important security concept
- You do certain things for those you trust
- You don’t do them for those you don’t
- Seems simple, but . . .
  - How do you express trust?
  - Why do you trust something?
  - How can you be sure who you’re dealing with?
  - What if trust is situational?
  - What if trust changes?

Trust and the Operating System

- You pretty much <u>have</u> to trust your operating system
- It controls all the hardware, including the memory
- It controls how your processes are handled
- It controls all the I/O devices
- If your OS is out to get you, you're gotten
- Which implies compromising an OS is a big deal

Authentication and Authorization

- In many security situations, we need to know who wants to do something
  - We allow trusted parties to do it
  - We don't allow others to do it
- That means we need to know who's asking
  - Determining that is *authentication*
- Then we need to check if that party should be allowed to do it
  - Determining that is *authorization*
  - Authorization usually requires authentication

Authentication

- Security policies tend to allow some parties to do something, but not others
- Which implies we need to know who's doing the asking
- For OS purposes, that's a determination made by a computer
- How?

Real World Authentication

- Identification by recognition
  - I see your face and know who you are
- Identification by credentials
  - You show me your driver's license
- Identification by knowledge
  - You tell me something only you know
- Identification by location
  - You're behind the counter at the DMV
- These all have cyber analogs

Authentication With a Computer

- Not as smart as a human
  - Steps to prove identity must be well defined
- Can’t do certain things as well
  - E.g., face recognition
- But lightning fast on computations and less prone to simple errors
  - Mathematical methods are acceptable
- Often must authenticate non-human entities
  - Like processes or machines

Identities in Operating Systems

- We usually rely primarily on a user ID
  - Which uniquely identifies some user
  - Processes run on his behalf, so they inherit his ID
    - E.g., a forked process has the same user associated as the parent did
- Implies a model where any process belonging to a user has all his privileges
  - Which has its drawbacks
  - But that's what we use

Bootstrapping OS Authentication

- Processes inherit their user IDs
- But somewhere along the line we have to create a process belonging to a new user
  - Typically on login to a system
- We can't just inherit that identity
- How can we tell who this newly arrived user is?

Passwords

- Authenticate the user by what he <u>knows</u>
  - A secret word he supplies to the system on login
- System must be able to check that the password was correct
  - Either by storing it
  - Or storing a hash of it
    - That's a much better option
- If correct, tie user ID to a new command shell or window management process

Problems With Passwords

- They have to be unguessable
  – Yet easy for people to remember
- If networks connect remote devices to computers, susceptible to password sniffers
  – Programs which read data from the network, extracting passwords when they see them
- Unless quite long, brute force attacks often work on them
- Widely regarded as an outdated technology
- But extremely widely used

Proper Use of Passwords

- Passwords should be sufficiently long
- Passwords should contain non-alphabetic characters
- Passwords should be unguessable
- Passwords should be changed often
- Passwords should never be written down
- Passwords should never be shared
- Hard to achieve all this simultaneously

Challenge/Response Systems

- Authentication by what questions you can answer correctly
  - Again, by what you know
- The system asks the user to provide some information
- If it's provided correctly, the user is authenticated
- Safest if it's a different question every time
  - Not very practical

Hardware-Based Challenge/ Response

- The challenge is sent to a hardware device belonging to the appropriate user
  - Authentication based on what you <u>have</u>
- Sometimes mere possession of device is enough
  - E.g., text challenges sent to a smart phone to be typed into web request
- Sometimes the device performs a secret function on the challenge
  - E.g., smart cards

Problems With Challenge/Response

- If based on what you know, usually too few unique and secret challenge/response pairs
- If based on what you have, fails if you don't have it
  – And whoever does have it might pose as you
- Some forms susceptible to network sniffing
  – Much like password sniffing
  – Smart card versions usually not susceptible

Biometric Authentication

- Authentication based on what you <u>are</u>
- Measure some physical attribute of the user
  - Things like fingerprints, voice patterns, retinal patterns, etc.
- Convert it into a binary representation
- Check the representation against a stored value for that attribute
- If it’s a close match, authenticate the user

Problems With Biometric Authentication

- Requires very special hardware
  - With some minor exceptions
- Many physical characteristics vary too much for practical use
- Generally not helpful for authenticating programs or roles
- Requires special care when done across a network

Errors in Biometric Authentication

- False positives
  - You identified Bill Smith as Peter Reiher
  - Probably because your biometric system was too generous in making matches
  - Bill Smith can pretend to be me
- False negatives
  - You didn't identify Peter Reiher as Peter Reiher
  - Probably because your biometric system was too stingy in making matches
  - I can't log in to my own account

Biometrics and Remote Authentication

- The biometric reading is just a bit pattern
- If attacker can obtain a copy, he can send the pattern over the network
  - Without actually performing a biometric reading
- Requires high confidence in security of path between biometric reader and checking device
  - Usually OK when both are on the same machine
  - Problematic when the Internet is between them

Access Control in Operating Systems

- The OS can control which processes access which resources
- Giving it the chance to enforce security policies
- The mechanisms used to enforce policies on who can access what are called access control
- Fundamental to OS security

Goals for Access Control

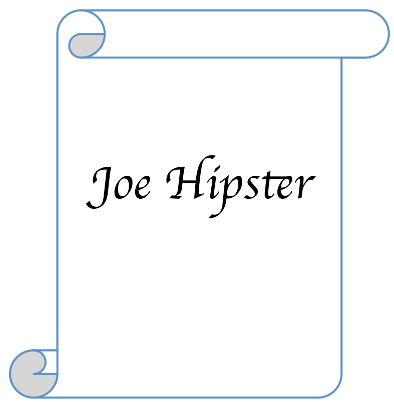
- Complete mediation
- Least privilege
- Useful in a networked environment
- Scalability
- Cost and usability

Access Control Lists

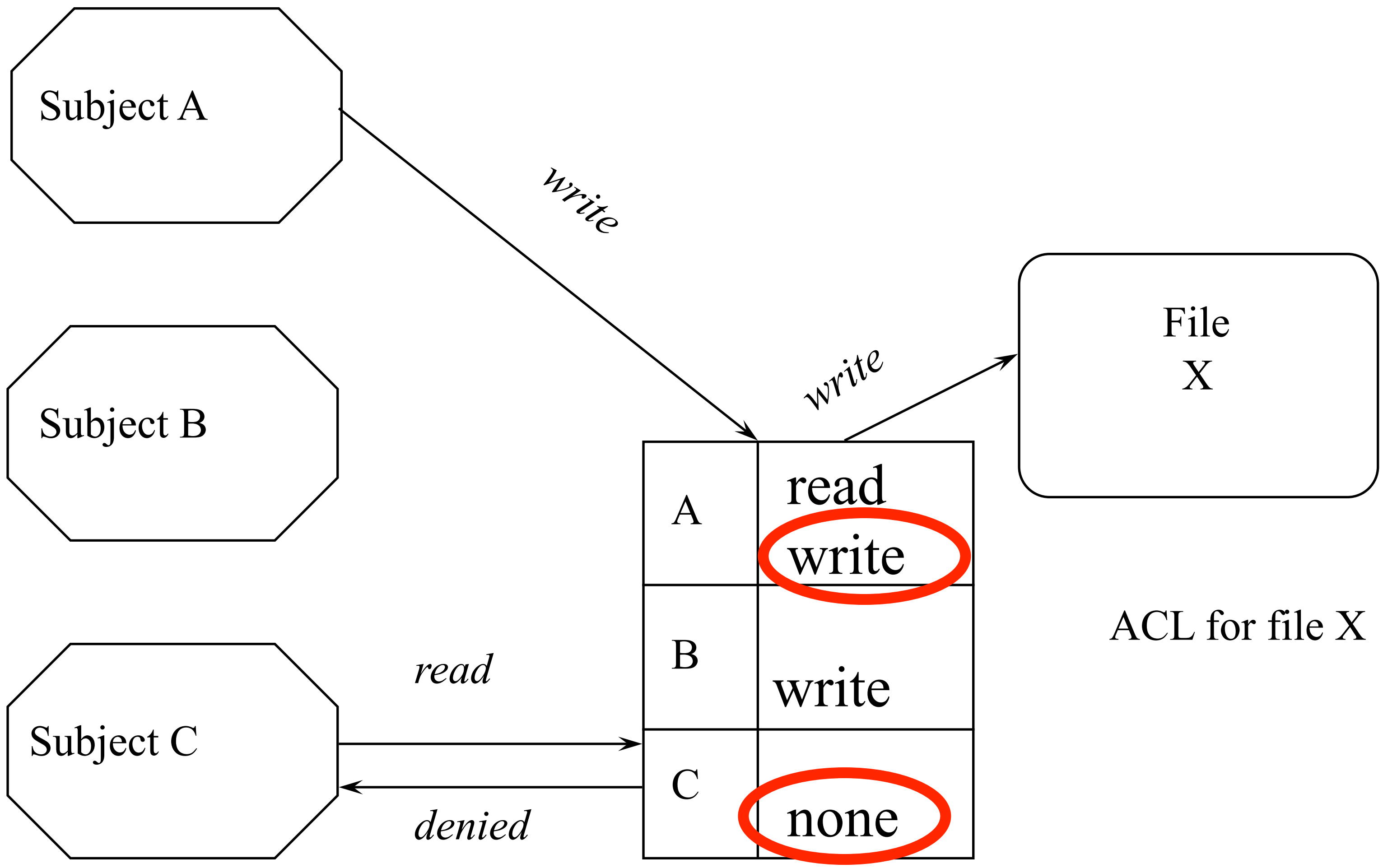
- ACLs
- For each protected object, maintain a single list
  - Managed by the OS, to prevent improper alteration
- Each list entry specifies who can access the object
  - And the allowable modes of access
- When something requests access to a object, check the access control list

An Analogy

This is an access control list



An ACL Protecting a File



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

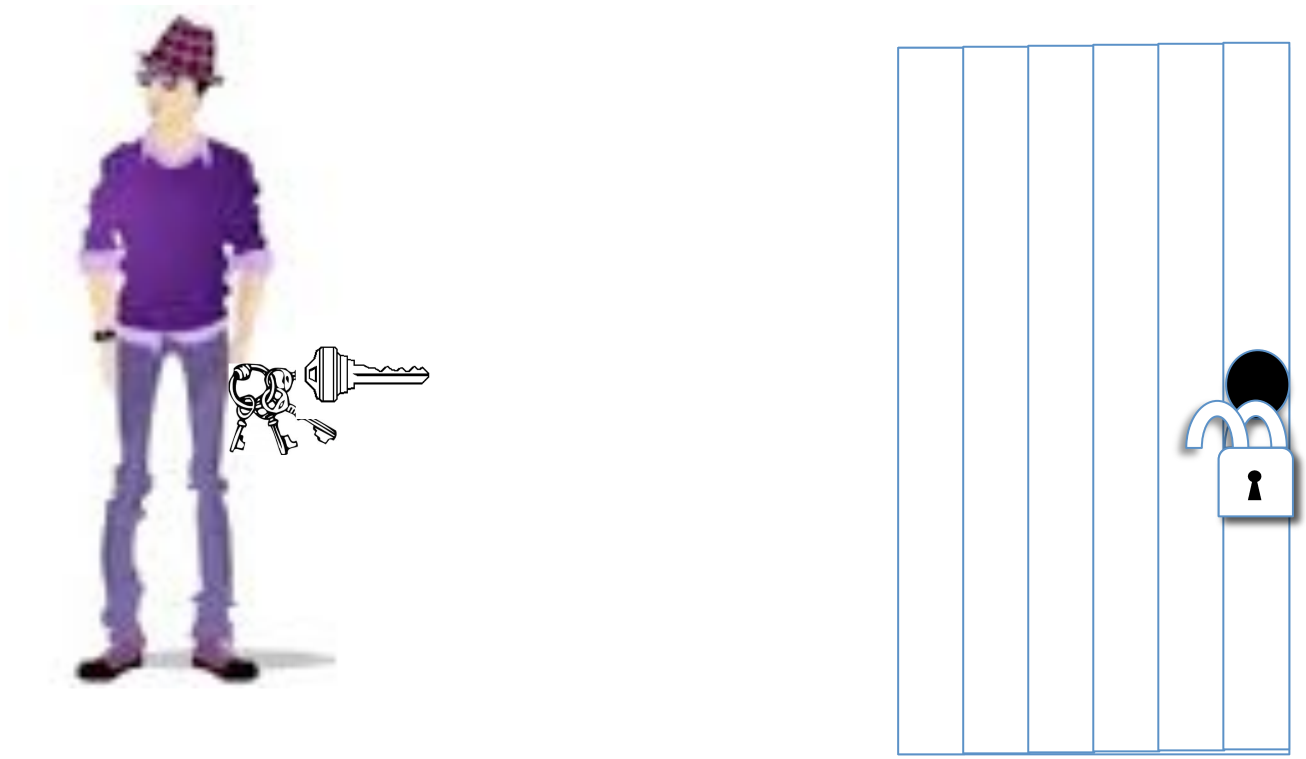
Pros and Cons of ACLs

+ 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

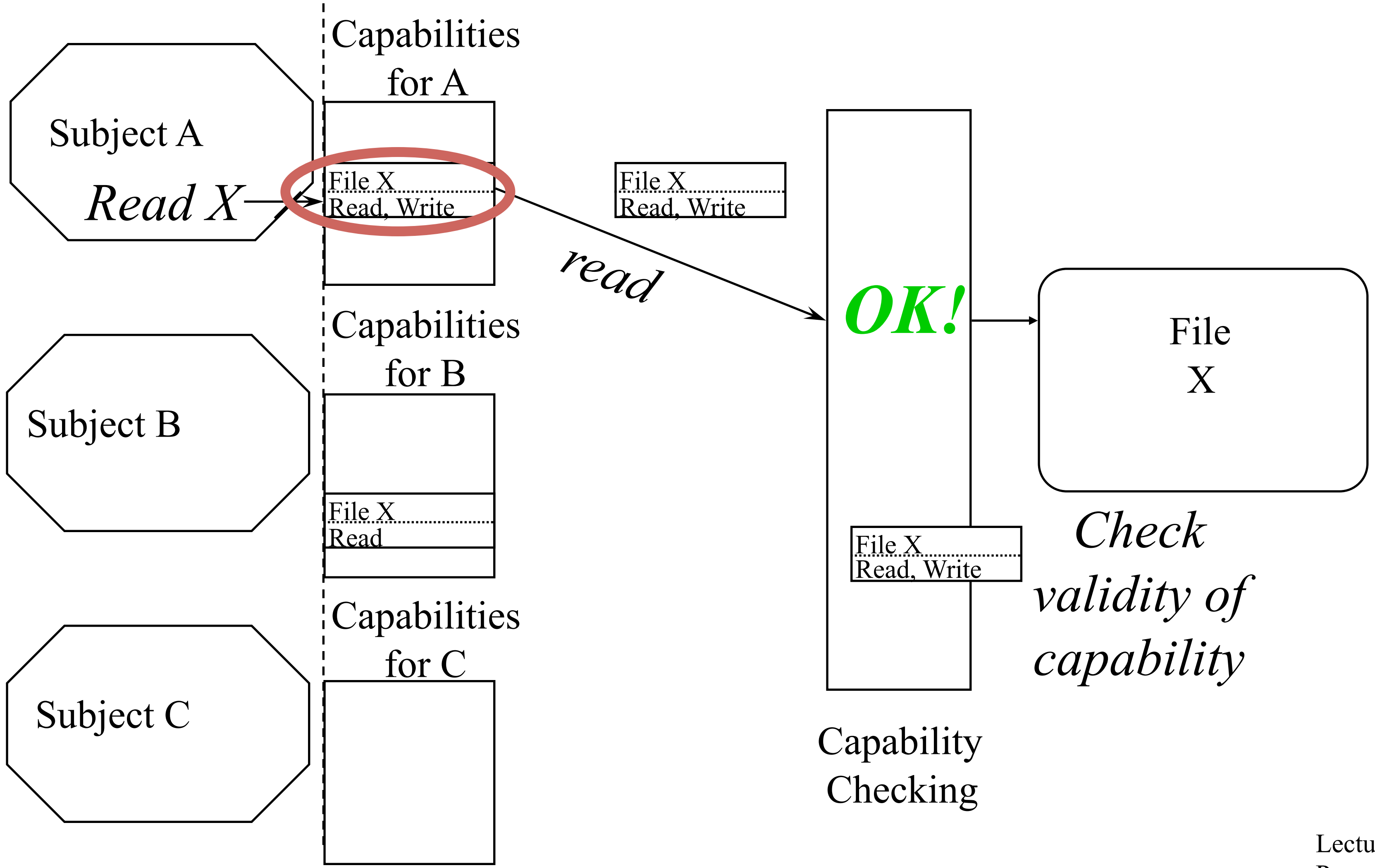
- Each entity keeps a set of data items that specify his allowable accesses
- Essentially, a set of tickets
- To access an object, present the proper capability
- Possession of the capability for an object implies that access is allowed

An Analogy



The key is a capability

Capabilities Protecting a File



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

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

OS Use of Access Control

- Operating systems often use both ACLs and capabilities
  - Sometimes for the same resource
- E.g., Unix/Linux uses ACLs for file opens
- That creates a file descriptor with a particular set of access rights
  - E.g., read-only
- The descriptor is essentially a capability

Enforcing Access in an OS

- Protected resources must be inaccessible
  - Hardware protection must be used to ensure this
  - So only the OS can make them accessible to a process
- To get access, issue request to OS
  - OS consults access control policy data
- Access may be granted directly
  - Resource manager maps resource into process
- Access may be granted indirectly
  - Resource manager returns a “capability” to process

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
  - Transforming bit patterns in controlled ways to obtain security advantages

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 Cryptosystems

- $C = E(K,P)$
- $P = D(K,C)$
- $P = D(K, E(K,P))$
- *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
  - For N possible keys, attack must try N/2 keys, on average, before finding the right one
- DES uses 56 bit keys
  - Too short for modern brute force attacks
- AES uses 128 or 256 bit keys
  - Long enough

Asymmetric Cryptosystems

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

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

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
- Also implies choosing key pairs is complex and expensive

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

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

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

M

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

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