

Introduction
CS 136
Computer Security
Peter Reiher
January 5, 2021

Purpose of Class

- To introduce students to computer security issues
- To familiarize students with secure software development
- To learn to handle security in today's installations and systems

Description of Class

- Topics to be covered
- Prerequisites
- Grading
- Reading materials
- Homework
- Office hours
- Web page

Topics to Be Covered

- Cryptography and authentication
 - Use, not design and analysis
- Access control and security models
- Secure software design and programming
- Secure protocols
- Network security – threats and countermeasures
- Operating systems security
- Security analysis and forensics
- Malware, common attacks, and important defenses
- Privacy
- Practical computer security defenses

Prerequisites

- CS111 (Operating Systems)
- CS118 (Computer Networks)
- Or equivalent classes elsewhere
- If you aren't familiar with this material, you'll be at a disadvantage
 - People have had serious problems with this unfamiliarity recently

Teaching Assistant

- Evan Czyzycki
 - eczy@cs.ucla.edu
- Weekly recitation sections Fridays
 - Section 1A: 12-1:50 AM
 - Via Zoom
 - Link to be announced

TA Duties

- Won't cover new material in recitation sections
- Will help clarify problems with lectures
- Will present information on using Deter testbed and performing labs
- Will also handle all other lab issues
- Office hours:
 - TBA

Grading

- Midterm – 25%
- Exercises – 36%
- Final – 38%
- Evaluation – 1%
- Grading will be done on this basis
 - Do not expect me to alter percentages for your particular case

How Do I Grade?

- Grades are assigned based on each student's performance on all graded materials
- Your grade depends on how well you did compared to the rest of the class
- Not a formal curve
- But definitely NOT any guarantee that a particular percentage gets a particular letter grade
 - If someone's told you there's a “grade guarantee,” they're wrong

Extra Credit

- Some of the exercises have possibilities for extra credit
- That's the only extra credit available in the class
- If you blow off the midterm, I won't offer you other extra credit possibilities

Class Format

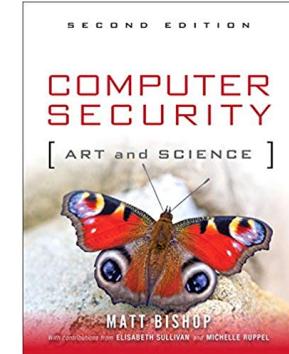
- A lecture class
- Questions and discussions always welcomed

Reading Materials

- Textbook
- Non-required supplemental text
- Links to papers and web pages

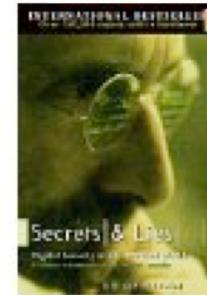
Textbook

- *Computer Security: Art and Science*
 - By Matt Bishop
- Second edition
 - Don't get a used first edition
- Should be available in the book store



Supplemental Text

- *Secrets and Lies*
 - By Bruce Schneier
- Not a textbook at all
- A philosophy of computer security
- Great for appreciating the field and problems
- Not great for depth of technical details
- Not required
 - No readings will be assigned from this book
 - But if you plan to work in this field, read it



Papers and Web Pages

- Mostly required reading material
 - Unless indicated otherwise
- Might or might not be assigned each week
- Made available electronically
 - Through class web page
- Generally relevant news stories or discussion of security topics

Exercises

- Five assignments
 - Plus a simple warmup
- Requiring practical work
- Performed on the Deter testbed
 - Accessible via the web from any connected location
 - Except exercise 5
- Individual, not group, assignments
 - Except exercise 5

Exercise Topics

0. Deter lab warmup
 - Week 1
1. Access control and permissions
 - Week 3
2. Exploits
 - Week 4
3. Analysis of attacks and forensics
 - Week 5
4. Man in the middle attacks
 - Week 7
5. Security analysis of a system
 - Week 10

More on Exercises

- Each exercise has an associated web page
 - With full instructions and pointers to necessary tools
- Due by midnight on indicated date
 - TAs may alter these dates
- Class TAs will provide advice and assistance on exercises

The Deter Testbed

- A set of machines devoted to security research and education
- Located at ISI and SRI
- Accessible remotely
- Special accounts set up for this class
- First discussion section will provide instructions on using Deter
 - With further assistance from TA

The Final Exercise

- A group exercise
 - Details of group composition to be provided
- Groups will perform a security evaluation of a piece of source code
- Resulting in a written report
- Full description provided later in the quarter

Tests

- Midterm – Tuesday, February 9
 - Via Zoom
- Final – Wednesday, March 17
 - Also via Zoom
- Open book/notes tests
- Students can take them during any time over a 24 hour period

Office Hours

- TTh 1-2
- Held via Zoom
 - Link will be provided in email
- Other times possible by appointment

Class Web Page

- On standard CCLE web site
- Slides for classes will be posted there
 - By 5 PM the previous afternoon
 - In Powerpoint and PDF
- Readings will be posted there
 - With links to web pages

A Few Words on Academic Honesty

- I expect all students to do their own work
 - Except on team projects
- I expect students not to cheat on tests
- All cases of suspected academic dishonesty will be reported to the Dean
 - Putting them out of my hands
- More details on class syllabus
- Or talk to me if in doubt

Introduction to Computer Security

- Why do we need computer security?
- What are our goals and what threatens them?

Why Is Security Necessary?

- Because people aren't always nice
- Because a lot of money is handled by computers
- Because a lot of important information is handled by computers
- Because our society is increasingly dependent on correct operation of computers

History of the Security Problem

- In the beginning, there was no computer security problem
- Later, there was a problem, but nobody cared
- Now, there's a big problem and people care
 - Only a matter of time before a real disaster
 - At least one company went out of business due to a DDoS attack
 - Identity theft and phishing claim vast number of victims
 - Stuxnet seriously damaged Iran's nuclear capability
 - Video showed cyberattack causing an electric transformer to fail
 - There's an underground business in cyber thievery
 - Increased industry spending on cybersecurity

Some Examples of Large Scale Security Problems

- Malicious code attacks
- Distributed denial of service attacks
- Vulnerabilities in commonly used systems

Malicious Code Attacks

- Multiple new viruses, worms, botnets, and Trojan horses appear every week
- Cryptojacking of everything from servers to web cameras
 - To mine cryptocurrency
- Stuxnet worm targeted at nuclear facilities
 - Unspecified amounts of damage done to Iran's nuclear program
- Increasing attacks on Internet of Things

Distributed Denial of Service Attacks

- Use large number of compromised machines to attack one target
 - By exploiting vulnerabilities
 - Or just generating lots of traffic
- Very common today
 - Major attack on Wikipedia in Europe and Middle East
 - 2018 attacks on online gaming companies
 - Attacks based on compromised Huawei routers
- In general form, an extremely hard problem

Vulnerabilities in Commonly Used Systems

- Recently, critical vulnerabilities in Android, Windows, iOS and macOS
- Many popular applications and middleware have vulnerabilities
 - Recent vulnerabilities in Webmin, Apache Solr, WordPress, Adobe Flash, Acrobat, Reader, etc.
- Many security systems have vulnerabilities
 - GlobalProtect Secure Socket Layer VPN, WPA3 Wifi security, Pulse Secure SSL VPN recently
- Critical hardware flaws in Intel and AMD processors
 - Fixes lead to slower processor

The SolarWinds Attack

- An extremely serious recent attack
- SolarWinds is a company that builds software to manage IT
- It was compromised and its Orion software corrupted
- Which then got deployed to its customers . . .

The Results of The Attack?

- The SolarWinds software controlled the customers' systems
- The corrupted version was controlled by the hackers
- So the hackers controlled the customers' systems

Who Were the Customers?

- US government agencies
 - Departments of Commerce, Treasury, Homeland Security, Energy
- Major corporations
 - Microsoft, Cisco, Intel, Nvidia, Deloitte LLC, VMWare, Belkin
- Computer security companies
 - Like FireEye (which discovered the problem)

Electronic Commerce Attacks

- As Willie Sutton said when asked why he robbed banks,
 - “Because that’s where the money is”
- Increasingly, the money is on the Internet
- Criminals have followed
- Common problems:
 - Identity theft (phishing is a common method)
 - Loss of valuable data from laptop theft
 - Extortion via DDoS attacks or threatened release of confidential data
 - Cryptocurrency mining on compromised machines
- Ponemon Institute estimates average cost of a data breach is \$3.9 million

Some Recent Statistics

- 2018 Verizon report found over 2200 data breaches from surveyed organizations
 - Two thirds of successful attacks only discovered months later
 - Health care organizations had $\frac{1}{4}$ of the breaches
- FBI Cybercrime report for 2017 showed over 300,000 reports
 - And losses of \$1.4 billion
- Predictions (possibly inflated) of losses of \$6 trillion worldwide in 2021

Cyberwarfare

- Nation states have developed capabilities to use computer networks for such purposes
- DDoS attacks on Estonia and Georgia
 - Probably just hackers
- Some regard Stuxnet as real cyberwarfare
 - Pretty clear it was done by US
- Attacks on Ukrainian power grid
- Continuous cyberspying by many nations
- Vulnerabilities of critical infrastructure
 - The smart grid increases the danger
- Russian election hacking in 2016

Something Else to Worry About

- Are some of the attempts to deal with cybersecurity damaging liberty?
- Does data mining for terror pose a threat to ordinary people?
 - The NSA is looking...
 - And they aren't the only ones...
- Can I trust Facebook/Google/Amazon/whoever with my private information?
- Are we in danger of losing all privacy?

Short answer:
No.

Why Aren't All Computer Systems Secure?

- Partly due to hard technical problems
- But also due to cost/benefit issues
- Security costs
- Security usually only pays off when there's trouble
- Many users perceive no personal threat to themselves
 - “I don’t have anything valuable on my computer”
 - “I don’t have any secrets and I don’t care what the government/Google/my neighbor knows about me”
- Ignorance also plays a role
 - Increasing numbers of users are unsophisticated
 - Important that computer security professionals don’t regard this ignorance as a character flaw
 - It’s a fact of life we must deal with

Legacy and Retrofitting

- We are constrained by legacy issues
 - Core Internet design
 - Popular programming languages
 - Commercial operating systems
- All developed before security was a concern
 - With little or no attention to security
- Retrofitting security works poorly
 - Consider the history of patching

Problems With Patching

- Usually done under pressure
 - So generally quick and dirty
- Tends to deal with obvious and immediate problem
 - Not with underlying cause
- Hard (sometimes impossible) to get patch to everyone
- Since it's not organic security, patches sometimes introduce new security problems
 - E.g., Microsoft 2018 patch for Meltdown “allowed any process to read the complete memory contents at gigabytes per second”

Speed Is Increasingly Killing Us

- Attacks are developed more quickly
 - Often easier to adapt attack than defense
- Malware spreads faster
 - Verizon report shows 87% of attacks took minutes or less to succeed
- More attackers generating more attacks
 - US DoD computers received 36 million malicious emails **daily** in 2018

Some Important Definitions

- Security
- Protection
- Vulnerabilities
- Exploits
- Trust

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 . . .

Problems With Trust

- 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?

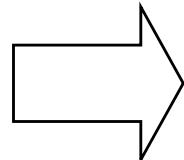
Trust Is Not a Theoretical Issue

- Most vulnerabilities that are actually exploited are based on trust problems
- Attackers exploit overly trusting elements of the computer
 - From the access control model to the actual human user
- Taking advantage of misplaced trust
- Such a ubiquitous problem that some aren't aware of its existence

Transitive Trust



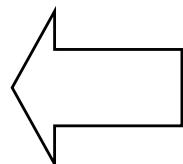
I trust Alice



Alice trusts Bob

So do I trust
Carol?
Should I?

David
trusts
Carol



Bob
trusts
David

Examples of Transitive Trust

- Trust systems in peer applications
- Chains of certificates
- But also less obvious things
 - Like a web server that calls a database
 - The database perhaps trusts the web server
 - But does the database necessarily trust the user who invoked the server?
- Programs that call programs that call programs are important cases of transitive trust
 - And this is how we build modern systems

What Are Our Security Goals?

- CIA
- Confidentiality
 - If it's supposed to be a 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

What Are the Threats?

- Theft (of data)
- Privacy
- Destruction
- Interruption or interference with computer-controlled services
- Misuse of computer controlled services

Active Threats Vs. Passive Threats

- *Passive threats* are forms of eavesdropping
 - No modification, injections of requests, etc.
- *Active threats* are more aggressive
- Passive threats are mostly to secrecy
- Active threats are to all properties

Social Engineering and Security

- The best computer security practices are easily subverted by bad human practices
 - E.g., giving passwords out over the phone to anyone who asks
 - Or responding to bogus email with your credit card number
- Social engineering attacks tend to be cheap, easy, effective
- So all our work may be for naught

Social Engineering Example

- Phishing
- Attackers send plausible email requesting you to visit a web site
- To “update” your information
- Typically a bank, popular web site, etc.
- The attacker controls the site and uses it to obtain your credit card, SSN, etc.
- Likelihood of success based on attacker’s ability to convince the victim that he’s real
 - And that the victim had better go to the site or suffer dire consequences

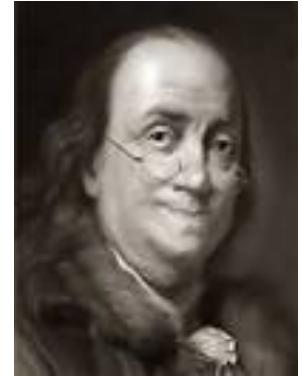
How Popular is Phishing?

- ~200,000 unique phishing web sites per month in late 2020¹
 - Targeting ~500 different brands
 - Around 120,000 unique phishing campaigns per month
- Based on gullibility of humans more than computer vulnerability
- But can computer scientists do something to help?

¹<http://www.antiphishing.org/>

Why Isn't Security Easy?

- Security is different than most other problems in CS
- The “universe” we’re working in is much more hostile
- Human opponents seek to outwit us
- Fundamentally, we want to share secrets in a controlled way
 - A classically hard problem in human relations



*Three can
keep a
secret, if
two of
them are
dead.*

What Makes Security Hard?

- You have to get everything right
 - Any mistake is an opportunity for your opponent
- When was the last time you saw a computer system that did everything right?
- So, must we wait for bug-free software to achieve security?

How Common Are Software Security Flaws?

- SANS used to publish weekly compendium of newly discovered security flaws
- About 1500 security flaws found per year
 - Only counting popular software
 - Only flaws with real security implications
 - And only those that were publicized
- SANS stopped doing this because it's not reasonable to expect anyone to keep up

Security Is Actually Even Harder

- The computer itself isn't the only point of vulnerability
- If the computer security is good enough, the foe will attack:
 - The users
 - The programmers
 - The system administrators
 - Or something you never thought of

A Further Problem With Security

- Security costs
 - Computing resources
 - People's time and attention
- If people use them badly, most security measures won't do the job
- Security must work 100% effectively
- With 0% overhead or inconvenience or learning

Another Problem

- Most computer practitioners know little or nothing about security
- Few programmers understand secure programming practices
- Few sysadmins know much about secure system configuration
- Typical users know even less

The Principle of Easiest Penetration

- *An intruder must be expected to use any available means of penetration. This is not necessarily the most obvious means, nor is it necessarily the one against which the most solid defense has been installed.*
- Put another way,
 - The smart opponent attacks you where you're weak, not where you're strong
 - And most opponents aren't stupid

But Sometimes Security Isn't That Hard

- The Principle of Adequate Protection:
 - *Computer items must be protected only until they lose their value. They must be protected to a degree consistent with their value.*
- So worthless things need little protection
- And things with timely value need only be protected for a while

Conclusion

- Security is important
- Security is hard
- A security expert's work is never done
 - At least, not for very long
- Security is full-contact computer science
 - Probably the most adversarial area in CS
- Intensely interesting, intensely difficult, and “the problem” will never be solved

Security Principles, Policies, and Tools

CS 136

Computer Security

Peter Reiher

January 7, 2021

Outline

- Security design principles
- Security policies
 - Basic concepts
 - Security policies for real systems
- Classes of security tools
 - Access control

Design Principles for Secure Systems

- Economy
- Complete mediation
- Open design
- Separation of privileges
- Least privilege
- Least common mechanism
- Acceptability
- Fail-safe defaults

Economy in Security Design

- Economical to develop
 - And to use
 - And to verify
- Should add little or no overhead
- Should do only what needs to be done
- Generally, try to keep it simple and small

Complete Mediation

- Apply security on every access to a protected object
 - E.g., each read of a file, not just the open
- Also involves checking access on everything that could be attacked

Open Design

- Don't rely on “security through obscurity”
- Assume all potential attackers know everything about the design
 - And completely understand it
- This doesn't necessarily mean publishing everything important about your security system
 - Though sometimes that's a good idea
- Obscurity can provide *some* security, but it's brittle
 - When the fog is cleared, the security disappears
 - And modern attackers have good fog blowers

Separation of Privileges

- Provide mechanisms that separate the privileges used for one purpose from those used for another
- To allow flexibility in security systems
- E.g., separate access control on each file
- Another example: different passwords for every web site you use

Least Privilege

- Give bare minimum access rights required to complete a task
- Require another request to perform another type of access
- E.g., don't give write permission to a file if the program only asked for read
- Extremely important when building complex systems

Least Common Mechanism

- Avoid sharing parts of the system's mechanism
 - Among different users
 - Among different parts of the system
- Coupling leads to possible security breaches
 - If it isn't shared, they can't see it

Acceptability

- Mechanism must be simple to use
- Simple enough that people will use it without thinking about it
- Must rarely or never prevent permissible accesses
- Sometimes expressed as principle of least astonishment
 - Any special actions required make sense

Fail-Safe Designs

- Default to lack of access
- So if something goes wrong or is forgotten or isn't done, no security lost
- If important mistakes are made, you'll find out about them
 - Without loss of security
 - But if it happens too often . . .

Security Policies

- Security policies describe how a secure system should behave
- Policy says what should happen, not how you achieve that
- Generally, if you don't have a clear policy, you don't have a secure system
 - Since you don't really know what you're trying to do

Informal Security Policies

- “Users should only be able to access their own files, in most cases.”
- “Only authorized users should be able to log in.”
- “System executables should only be altered by system administrators.”
- The general idea is pretty clear
- But it can be hard to determine if a system meets these goals

Formal Security Policies

- Typically expressed in a mathematical security policy language
- Tending towards precision
 - Allowing formal reasoning about the system and policy
- Often matched to a particular policy model
 - E.g., Bell-La Padula model
- Hard to express many sensible policies in formal ways
 - And hard to reason about them usefully

Some Important Security Policies

- Bell-La Padula
- Biba integrity policy

Bell-La Padula Model

- Probably best-known formal computer security model
- Corresponds to military classifications
- Combines mandatory and discretionary access control
- Two parts:
 - Clearances
 - Classifications

Clearances

- Subjects (people, programs, etc.) have a *clearance*
- Clearance describes how trusted the subject is
- E.g., *unclassified, confidential, secret, top secret*

Classifications

- Each object (file, database entry, etc.) has a *classification*
- The classification describes how sensitive the object is
- Using same categories as clearances
- Informally, only people with the same (or higher) clearance should be able to access objects of a particular classification

Goal of Bell-La Padula Model

- Prevent any subject from ever getting read access to data at higher classification levels than subject's clearance
 - I.e., don't let untrusted people see your secrets
- Concerned not just with objects
- Also concerned with the objects' contents
- Includes discretionary access control
 - Which we won't cover in lecture

Bell-La Padula Simple Security Condition

- *Subject S can read object O iff $l_O \leq l_S$*
- Simple enough:
 - If S isn't granted top secret clearance, S can't read top secret objects
- Are we done?

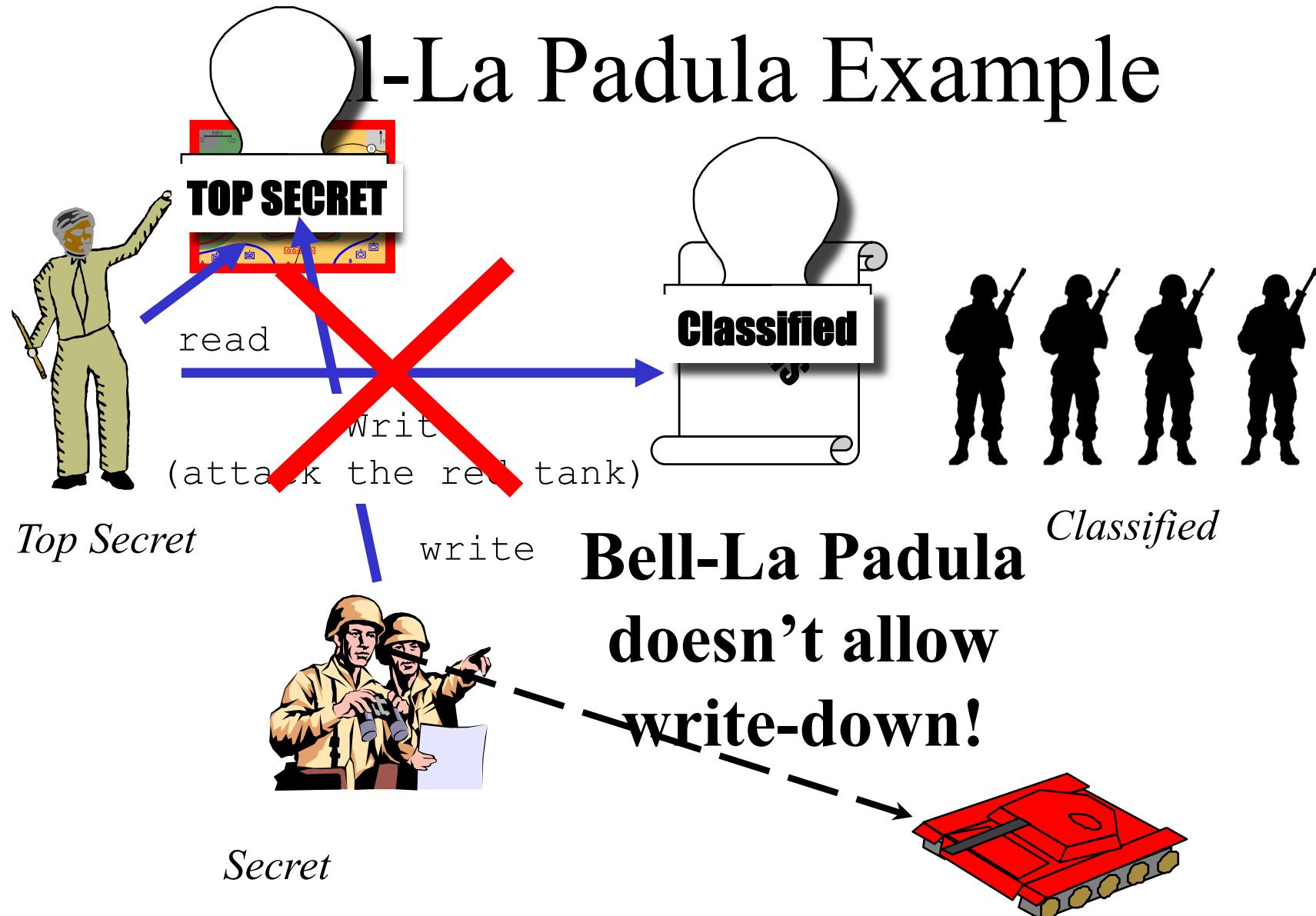
Why Aren't We Done?

- Remember, we really care about the information in an object
- A subject with top secret clearance can read a top secret object
- If careless, he could write that information to a confidential object
- Then someone with confidential clearance can read top secret information

The Bell-La Padula *-Property

- S can write O iff $l_S \leq l_O$
- Prevents *write-down*
 - Privileged subjects writing high-classification information to low-classification objects
 - E.g., a top secret user can't write to a confidential data file
- Can be proven that a system meeting these properties is “secure”

Bell-La Padula Example



So How Do You Really Use The System?

- There have to be mechanisms for reclassification
 - Usually requiring explicit operation
- Danger that reclassification process will be done incautiously
- Real systems also use classes of information
 - Remember least common mechanism?

Integrity Security Policies

- Designed to ensure that information is not improperly changed
- Often the key issue for commercial systems
- Secrecy is nice, but not losing track of your inventory is crucial

Example: Biba Integrity Policy

- Subject set S , object set O
- Set of ordered integrity levels I
- Subjects and objects have integrity levels
- Subjects at high integrity levels are less likely to screw up data
 - E.g., trusted users or carefully audited programs
- Data at a high integrity level is less likely to be screwed up
 - Probably because it badly needs not to be screwed up

Biba Integrity Policy Rules

- s can write to o iff $i(o) \leq i(s)$
- s_1 can execute s_2 iff $i(s_2) \leq i(s_1)$
- A subject s can read object o iff $i(s) \leq i(o)$
- Why do we need the read rule?

Hybrid Models

- Sometimes the issue is keeping things carefully separated
- E.g., a brokerage that handles accounts for several competing businesses
- Microsoft might not like the same analyst working on their account and IBM's
- There are issues of both confidentiality and integrity here
- Example – Chinese Wall model

The Problems With Security Policies

- Hard to define properly
 - How do you determine what to allow and disallow?
- Hard to go from policy to the mechanisms that actually implement it
- Hard to understand implications of policy
- Defining and implementing policies is a lot of work

Tools for Security

- Physical security
- Access control
- Encryption
- Authentication
- Encapsulation
- Intrusion detection
- Common sense

Physical Security

- Lock up your computer
 - Actually, sometimes a good answer
- But what about networking?
 - Networks poke a hole in the locked door
- Mobility also raises challenges
- Hard to prevent legitimate holder of a computer from using it as he wants
 - E.g., smart phone jailbreaks
- In any case, lack of physical security often makes other measures pointless

Access Controls

- Only let authorized parties access the system
- A lot trickier than it sounds
- Particularly in a network environment
- Once data is outside your system, how can you continue to control it?
 - Again, of concern in network environments

Encryption

- Algorithms to hide the content of data or communications
- Only those knowing a secret can decrypt the protection
- One of the most important tools in computer security
 - But not a panacea
- Covered in more detail later in class

Authentication

- Methods of ensuring that someone is who they say they are
- Vital for access control
- But also vital for many other purposes
- Often (but not always) based on cryptography

Encapsulation

- Methods of allowing outsiders limited access to your resources
 - Preferably making inaccessible things invisible
- Let them see or access some things
 - But not everything
 - E.g., virtual machines and sandboxes
- Simple, in concept
- Extremely challenging, in practice

Intrusion Detection

- All security methods sometimes fail
- When they do, notice that something is wrong
- And take steps to correct the problem
- Reactive, not preventative
 - But it's unrealistic to believe any prevention is certain
- Must be automatic to be really useful

Common Sense

- A lot of problems arise because people don't like to think
- The best security tools generally fail if people use them badly
- If the easiest way in is to fool people, that's what attackers will do

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
 - And at the right time and circumstances
- How do we ensure that a given resource can only be accessed when it should be?

Goals for Access Control

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

Access Control Mechanisms

- Access control lists
- Capabilities
- Access control matrices
 - Theoretical concept we won't discuss in detail
- Role based access control

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

Mandatory vs. Discretionary Access Control

- Mandatory access control is dictated by the underlying system
 - Individual users can't override it
 - Even for their own data
- Discretionary access control is under command of the user
 - System enforces what they choose
 - More common than mandatory

The Realities of Discretionary Access Control

- Most users never change the defaults on anything
 - Unless the defaults prevent them from doing something they want to do
- Most users don't think about or understand access control
- Probably not wise to rely on it to protect information you care about
 - Unless you're the one setting it
 - And you know what you're doing

Access Control Lists

- For each protected resource, maintain a single list
- Each list entry specifies a user who can access the resource
 - And the allowable modes of access
- When a user requests access to a resource, check the access control list (ACL)

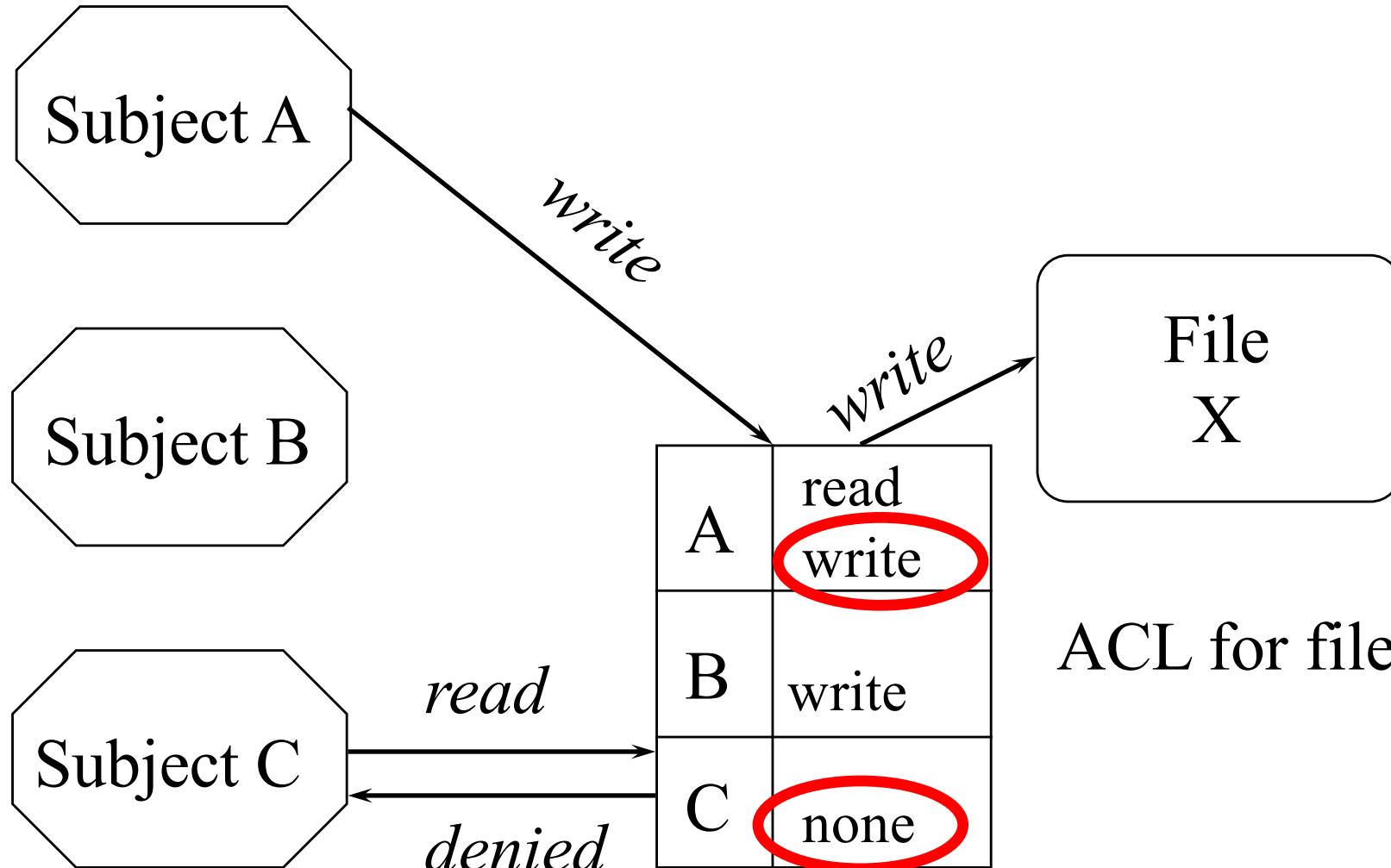
ACL Objects and Subjects

- In ACL terminology, the resources being protected are *objects*
- The entities attempting to access them are *subjects*
 - Allowing finer granularity of control than per-user

ACL Example

- An operating system example:
 - Using ACLs to protect a file
- User (Subject) A is allowed to read and write to the file
- User (Subject) B may only read from it
- User (Subject) C may not access it

An ACL Protecting a File



Issues for Access Control Lists

- How do you know that 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?
- Generally issues for OS design

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

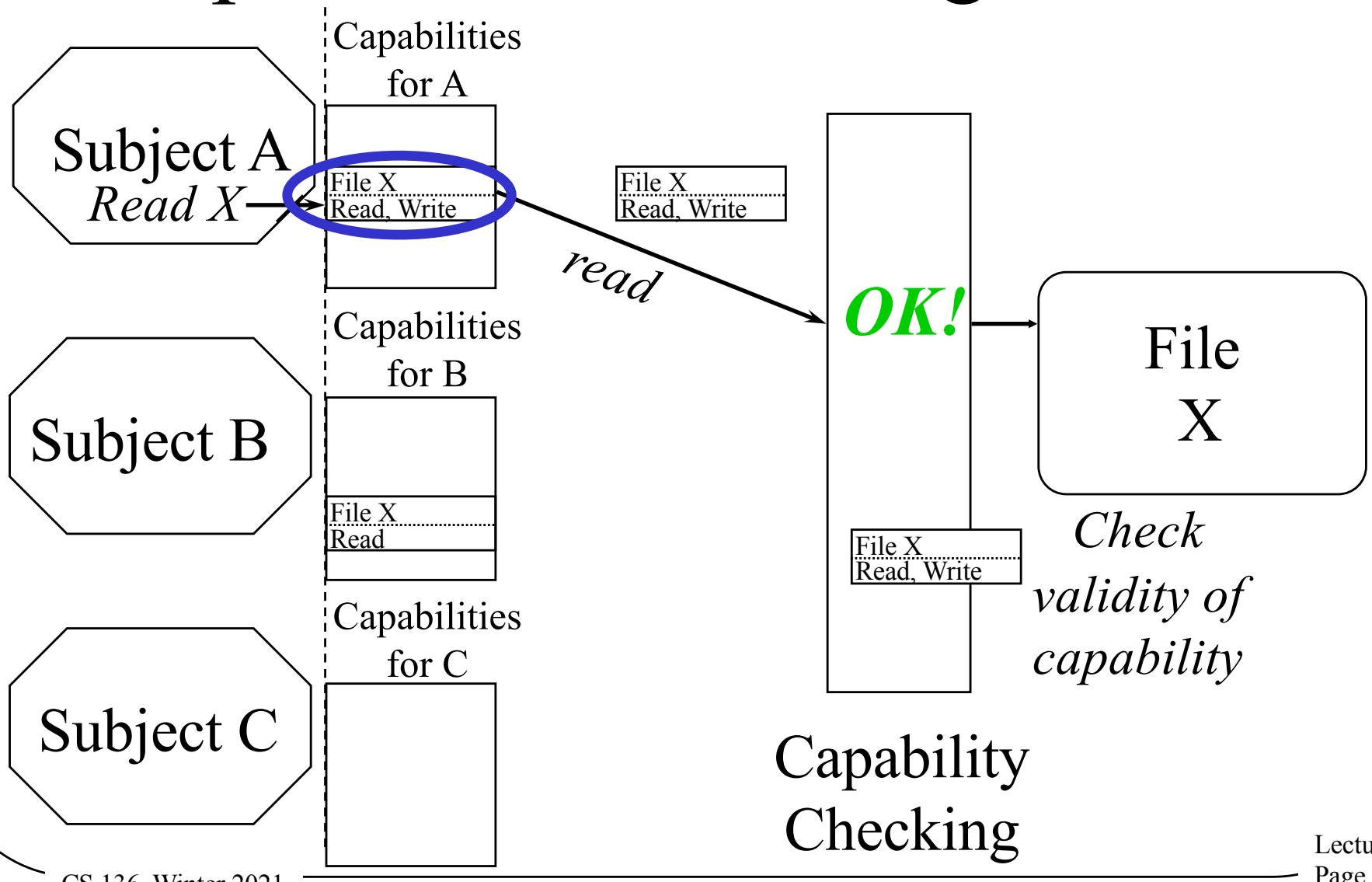
Capabilities

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

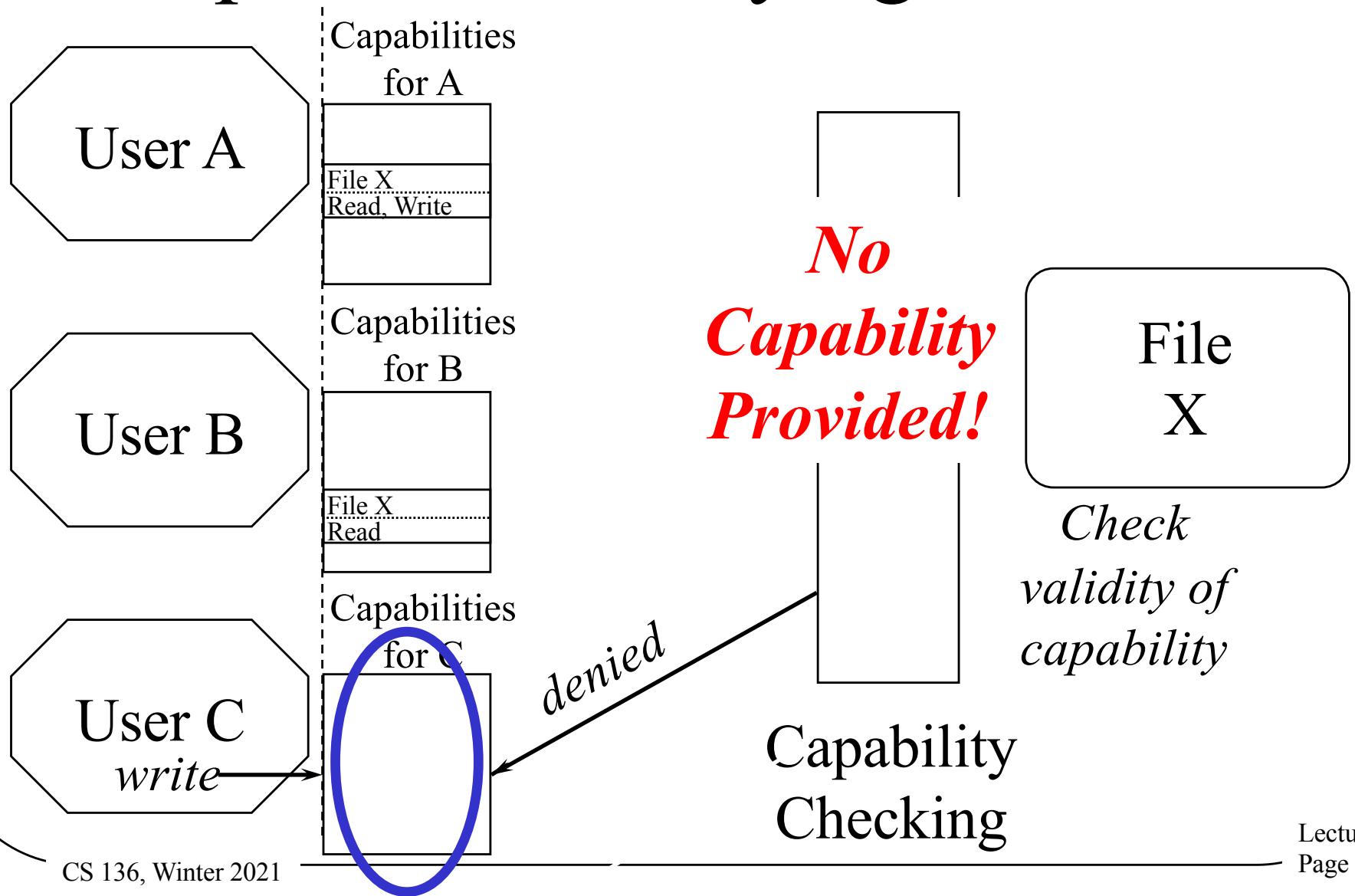
Properties of Capabilities

- Must be unforgeable
 - In single machine, keep capabilities under control of OS
 - What about in a networked system?
- In most systems, some capabilities allow creation of other capabilities
 - Process can pass a restricted set of capabilities to a subprocess

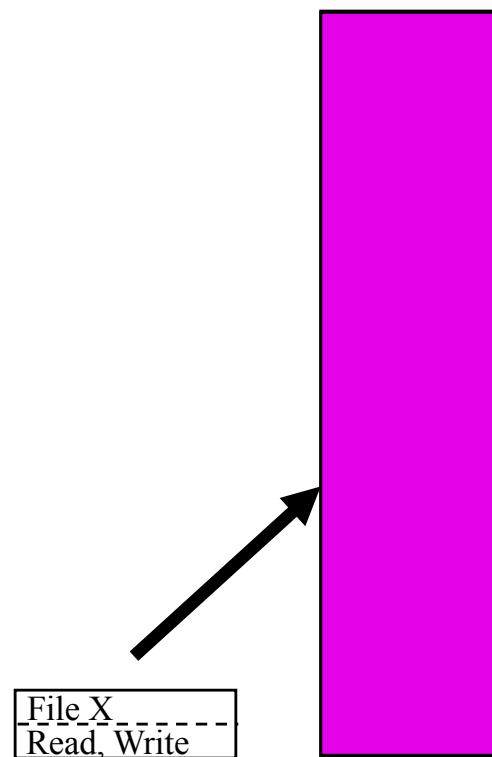
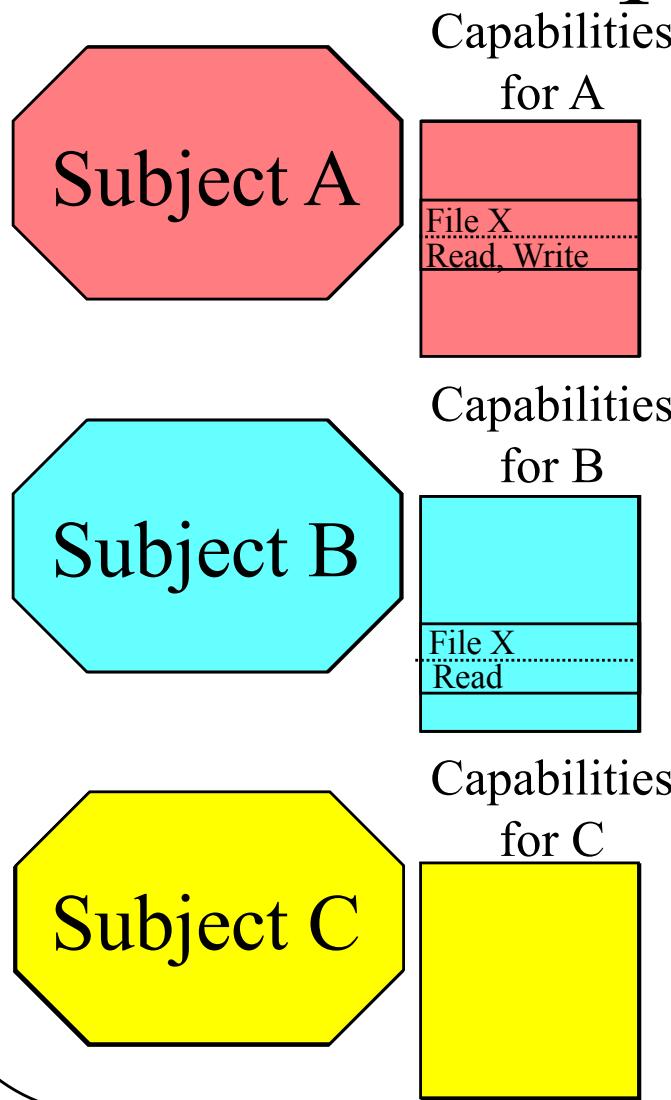
Capabilities Protecting a File



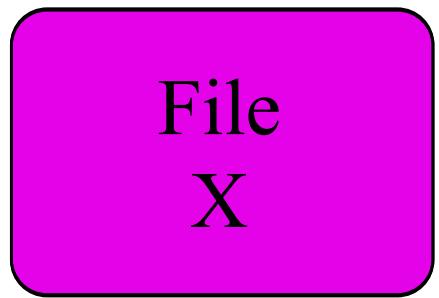
Capabilities Denying Access



How Will This Work in a Network?



How can we tell if it's a good capability?



Capability
Checking

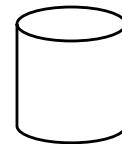
Revoking Capabilities



Fred



Nancy



Accounts
receivable

How do we take
away Fred's
capability?

Without taking
away Nancy's?

Options for Revoking Capabilities

- Destroy the capability
 - How do you find it?
- Revoke on use
 - Requires checking on use
- Generation numbers
 - Requires updating non-revoked capabilities

Pros and Cons of Capabilities

- + Easy to determine what 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

Distributed Access Control

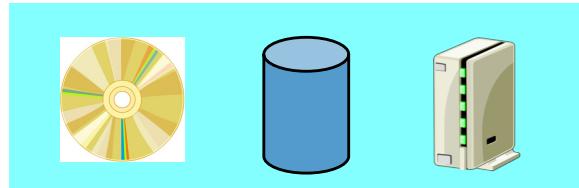
- ACLs still work OK
 - Provided you have a global namespace for subjects
 - And no one can masquerade
- Capabilities are more problematic
 - Security relies on unforgeability
 - Provided by cryptographic methods
 - Prevents forging, not copying

Role Based Access Control

- An enhancement to ACLs or capabilities
- Each user has certain roles he can take while using the system
- At any given time, the user is performing a certain role
- Give the user access to only those things that are required to fulfill that role
- Available in some form in most modern operating systems

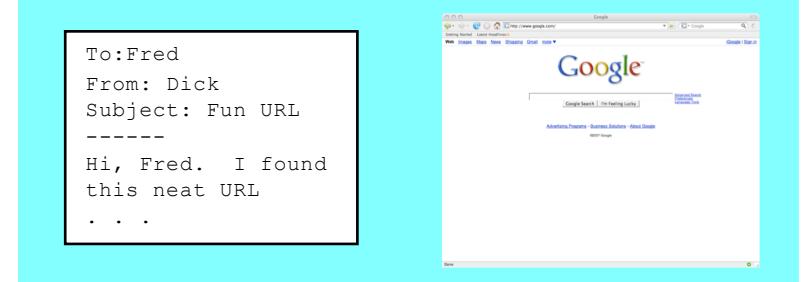
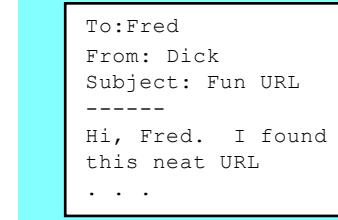
A Simple Example

Fred is a system administrator



Fred should operate under one role while doing system administration

But Fred is also a normal user



And another role while doing normal stuff

Continuing With Our Example

He decides to upgrade
the C++ compiler

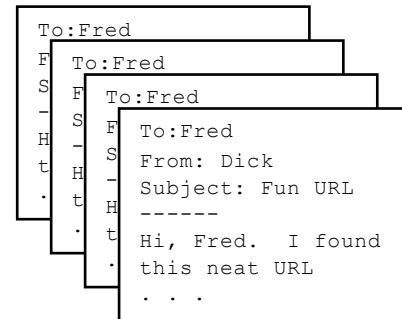


So he changes his
role to “sysadmin”

Then he has the privileges to
upgrade the compiler

But may have lost the privileges
to read “fred’s” email

Fred logs on as “fred”
He reads his email



*Result: Evil malware in
fred's email can't
“upgrade” the compiler*

Changing Roles

- Role based access control only helps if changing roles isn't trivial
 - Otherwise, the malicious code merely changes roles before doing anything else
- Typically requires providing some secure form of authentication
 - Which proves you have the right to change roles
 - Usually passwords, but other methods possible

Practical Limitations on Role Based Access Control

- Number of roles per user
- Problems of disjoint role privileges
- System administration overheads
- Generally, these cause usability and management problems

Reference Monitors

- Whatever form it takes, access control must be instantiated in actual code
 - Which checks if a given attempt to reference an object should be allowed
- That code is called a *reference monitor*
- Obviously, good reference monitors are critical for system security

Desirable Properties of Reference Monitors

- Correctness
- Proper placement
- Efficiency
- Simplicity
- Flexibility

Introduction to Cryptography

CS 136

Computer Security

Peter Reiher

January 12, 2021

Outline

- What is data encryption?
- Cryptanalysis
- Basic encryption methods
 - Substitution ciphers
 - Permutation ciphers

Introduction to Encryption

- Much of computer security is about keeping secrets
- One method is to make the secret hard for others to read
- While (usually) making it simple for authorized parties to read

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

Encryption and Data Transformations

- Encryption is all about transforming the data
- One bit or byte pattern is transformed to another bit or byte pattern
- Usually in a reversible way

Encryption Terminology

- Encryption is typically described in terms of sending a message
 - Though it's used for many other purposes
- The sender is S
- The receiver is R
- And the attacker is O

More Terminology

- *Encryption* is the process of making message unreadable/unalterable by O
- *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)
- *Ciphertext* is the encrypted form of the message (often referred to as C)

Transfer
\$100 to my
savings
account

Sqzmredq
#099 sn lx
rzuhmfr
zbbntms

Very Basics of Encryption Algorithms

- Most 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

Terminology for Encryption Algorithms

- The encryption algorithm is referred to as $E()$
- $C = E(K, P)$
- The decryption algorithm is referred to as $D()$
 - Sometimes the same algorithm as $E()$
- The decryption algorithm also has a key

Symmetric and Asymmetric Encryption Systems

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

$$P = D(K, C)$$

Expanding, $P = D(K, E(K, P))$

- Asymmetric systems use different keys for E and D:

$$C = E(K_E, P)$$

$$P = D(K_D, C)$$

Characteristics of Keyed Encryption Systems

- If you change only the key, a given plaintext encrypts to a different ciphertext
 - Same applies to decryption
- Decryption should be hard without knowing the key

Cryptanalysis

- The process of trying to break a cryptosystem
- Finding the meaning of an encrypted message without being given the key
- To build a strong cryptosystem, you must understand cryptanalysis

Forms of Cryptanalysis

- Analyze an encrypted message and deduce its contents
- Analyze one or more encrypted messages to find a common key
- Analyze a cryptosystem to find a fundamental flaw

Breaking Cryptosystems

- Most cryptosystems are breakable
- Some just cost more to break than others
- The job of the cryptosystem designer is to make the cost infeasible
 - Or incommensurate with the benefit extracted

Types of Attacks on Cryptosystems

- Ciphertext only
- Known plaintext
- Chosen plaintext
 - Differential cryptanalysis
- Algorithm and ciphertext
 - Timing attacks
- In many cases, the intent is to guess the key

Ciphertext Only

- No *a priori* knowledge of plaintext
- Or details of algorithm
- Must work with probability distributions, patterns of common characters, etc.
- Hardest type of attack

Known Plaintext

- Full or partial
- Cryptanalyst has matching sample of ciphertext and plaintext
- Or may know something about what ciphertext represents
 - E.g., an IP packet with its headers

Chosen Plaintext

- Cryptanalyst can submit chosen samples of plaintext to the cryptosystem
- And recover the resulting ciphertext
- Clever choices of plaintext may reveal many details
- Differential cryptanalysis iteratively uses varying plaintexts to break the cryptosystem
 - By observing effects of controlled changes in the offered plaintext

Algorithm and Ciphertext

- Cryptanalyst knows the algorithm and has a sample of ciphertext
- But not the key, and cannot get any more similar ciphertext
- Can use “exhaustive” runs of algorithm against guesses at plaintext
- Password guessers often work this way
- *Brute force attacks* – try every possible key to see which one works

Timing Attacks

- Usually assume knowledge of algorithm
- And ability to watch algorithm encrypting/decrypting
- Some algorithms perform different operations based on key values
- Watch timing to try to deduce keys
- Successful against some smart card crypto
- Similarly, observe power use by hardware while it is performing cryptography

Basic Encryption Methods

- Substitutions
 - Monoalphabetic
 - Polyalphabetic
- Permutations

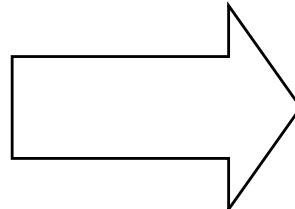
Substitution Ciphers

- Substitute one or more characters in a message with one or more different characters
- Using some set of rules
- Decryption is performed by reversing the substitutions

Example of a Simple Substitution Cipher

How did this transformation happen?

Sqzmredq
#099 sn lx
rzuhmfr
zbbntms



Sqzmredq
#099 sn lx
rzuhmfr
zbbntms

Every letter was changed to the “next lower” letter

Caesar Ciphers

- A simple substitution cipher like the previous example
 - Supposedly invented by Julius Caesar
- Translate each letter a fixed number of positions in the alphabet
- Reverse by translating in opposite direction

Is the Caesar Cipher a Good Cipher?

- Well, it worked great 2000 years ago
- It's simple, but
- It's simple
- Fails to conceal many important characteristics of the message
- Which makes cryptanalysis easier
- Limited number of useful keys

How Would Cryptanalysis Attack a Caesar Cipher?

- Letter frequencies
- In English (and other alphabetic languages), some letters occur more frequently than others
- Caesar ciphers translate all occurrences of a given plaintext letter into the same ciphertext letter
- All you need is the offset

More On Frequency Distributions

- In most languages, some letters used more than others
 - In English, “e,” “t,” and “s” are common
- True even in non-natural languages
 - Certain characters appear frequently in C code
 - Zero appears often in numeric data

Cryptanalysis and Frequency Distribution

- If you know what kind of data was encrypted, frequency distributions offer a path to breaking it
- Especially for Caesar ciphers
 - And other simple substitution-based encryption algorithms

Breaking Caesar Ciphers

- Identify (or guess) the kind of data
- Count frequency of each encrypted symbol
- Match to observed frequencies of unencrypted symbols in similar plaintext
- Provides probable mapping of cipher
- The more ciphertext available, the more reliable this technique

Example

- With ciphertext “Sqzmredq #099 sn lx
rzuhmfr zbbntms”
- Frequencies -

a	0		b	2		c	0		d	1		e	1
f	1		g	0		h	1		i	0		j	0
k	0		l	1		m	3		n	2		o	0
p	0		q	2		r	3		s	3		t	1
u	1		v	0		w	0		x	1		y	0
z	3												

Applying Frequencies To Our Example

a	0		b	2		c	0		d	1		e	1
f	1		g	0		h	1		i	0		j	0
k	0		l	1		m	3		n	2		o	0
p	0		q	2		r	3		s	3		t	1
u	1		v	0		w	0		x	1		y	0
z	3												

- The most common English letters are typically “e,” “t,” “a,” “o,” and “s”
- Four out of five of the common English letters in the plaintext map to these letters

Cracking the Caesar Cipher

- Since all substitutions are offset by the same amount, just need to figure out how much
- How about +1?
 - That would only work for $a \Rightarrow b$
- How about -1?
 - That would work for $t \Rightarrow s$, $a \Rightarrow z$, $o \Rightarrow n$, and $s \Rightarrow r$
 - Try it on the whole message and see if it looks good

More Complex Substitutions

- Monoalphabetic substitutions
 - Each plaintext letter maps to a single, unique ciphertext letter
- Any mapping is permitted
- Key can provide method of determining the mapping
 - Key could be the mapping

Are These Monoalphabetic Ciphers Better?

- Only a little
- Finding the mapping for one character doesn't give you all mappings
- But the same simple techniques can be used to find the other mappings
- Generally insufficient for anything serious

Polyalphabetic Ciphers

- Ciphers that don't always translate a given plaintext character into the same ciphertext character
- For example, use different substitutions for odd and even positions

Example of Simple Polyalphabetic Cipher

- Move one character “up” in even positions, one character “down” in odd positions
- Note that same character translates to different characters in some cases

Transfer
\$100 to my
savings
account

Sszorgds
%019 sp nx
tbujmhr
zdbptos

Are Polyalphabetic Ciphers Better?

- Depends on how easy it is to determine the pattern of substitutions
- If it's easy, then you've gained little

Cryptanalysis of Our Example

- Consider all even characters as one set
- And all odd characters as another set
- Apply basic cryptanalysis to each set
- The transformations fall out easily
- How did you know to do that?
 - You guessed
 - Might require several guesses to find the right pattern

How About For More Complex Patterns?

- Good if the attacker doesn't know the choices of which characters get transformed which way
- Attempt to hide patterns well
- But known methods still exist for breaking them

Methods of Attacking Polyalphabetic Ciphers

- Kasiski method tries to find repetitions of the encryption pattern
- Index of coincidence predicts the number of alphabets used to perform the encryption
- Both require lots of ciphertext

How Does the Cryptanalyst “Know” When He’s Succeeded?

- Every key translates a message into something
- If a cryptanalyst thinks he’s got the right key, how can he be sure?
- Usually because he doesn’t get garbage when he tries it
- He almost certainly will get garbage from any other key
- Why?

Consider A Caesar Cipher

- There are 25 useful keys (in English)
- The right one will clearly yield meaningful text
- What's the chances that any of the other 24 will?
 - Pretty poor
- So if the decrypted text makes sense, you've got the key

The More General Case

- Let's say the message is N bits long
 - So there are 2^N possible messages
 - But many of those make no sense
- Let's say the key is m bits long ($m \ll N$)
 - So there are 2^m keys
- What if only only 2^k of the possible messages make sense?
 - $2^k \ll 2^N$
 - For example, if the message was English text
- Assuming everything is random (and a good encryption algorithm tries to be)
 - The chance any wrong key decrypts to something sensible is around $1/2^N$ (k and $m \ll N$)



A Simple Example

Let's encrypt a 3 letter English message

In ASCII, that's 24 bits

0	1	1	1	1	0	0	1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Let's use a 4 bit key

1	1	0	1
---	---	---	---

1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

If we don't know the key, there are 16 possible decryptions

What are the chances one of the other 15 keys
decrypts this ciphertext to an English language word?

$$15 * 1000 / 16777216$$

Around .1%

For longer messages, lower odds

~1000 3
letter words

The Unbreakable Cipher

- There is a “perfect” substitution cipher
- One that is theoretically (and practically) unbreakable without the key
- And you can’t guess the key
 - If the key was chosen in the right way . . .

One-Time Pads

- Essentially, use a new substitution alphabet for every character
- Substitution alphabets chosen purely at random
 - These constitute the key
- Provably unbreakable without knowing this key

Example of One Time Pads

- Usually explained with bits, not characters
- We shall use a highly complex cryptographic transformation:
 - XOR
- And a three bit message
 - 010

One Time Pads at Work

0	1	0
---	---	---

Apply our sophisticated cryptographic algorithm

Flip some coins to get random numbers

0	0	1
---	---	---

0	1	1
---	---	---

We now have an unbreakable cryptographic message

What's So Secure About That?

- Any key was equally likely
- Any plaintext could have produced this message with one of those keys
- Let's look at our example more closely

Why Is the Message Secure?

Let's say there
are only two
possible
meaningful
messages

Could the
message decrypt
to either or both
of these?

0	1	1
---	---	---

0	1	0
---	---	---

0	0	0
---	---	---

There's a key that
works for each
And they're
equally likely

0	0	1
---	---	---

0	1	1
---	---	---

Security of One-Time Pads

- If the key is truly random, provable that it can't be broken without the key
- But there are problems
- Need one bit of key per bit of message
- Key distribution is painful
- Synchronization of keys is vital
- A good random number generator is hard to find

One-Time Pads and Cryptographic Snake Oil

- Companies regularly claim they have “unbreakable” cryptography
- Usually based on one-time pads
- But typically misused
 - Pads distributed with some other crypto mechanism
 - Pads generated with non-random process
 - Pads reused

Permutation Ciphers

- Instead of substituting different characters, scramble up the existing characters
- Use algorithm based on the key to control how they're scrambled
- Decryption uses key to unscramble

Characteristics of Permutation Ciphers

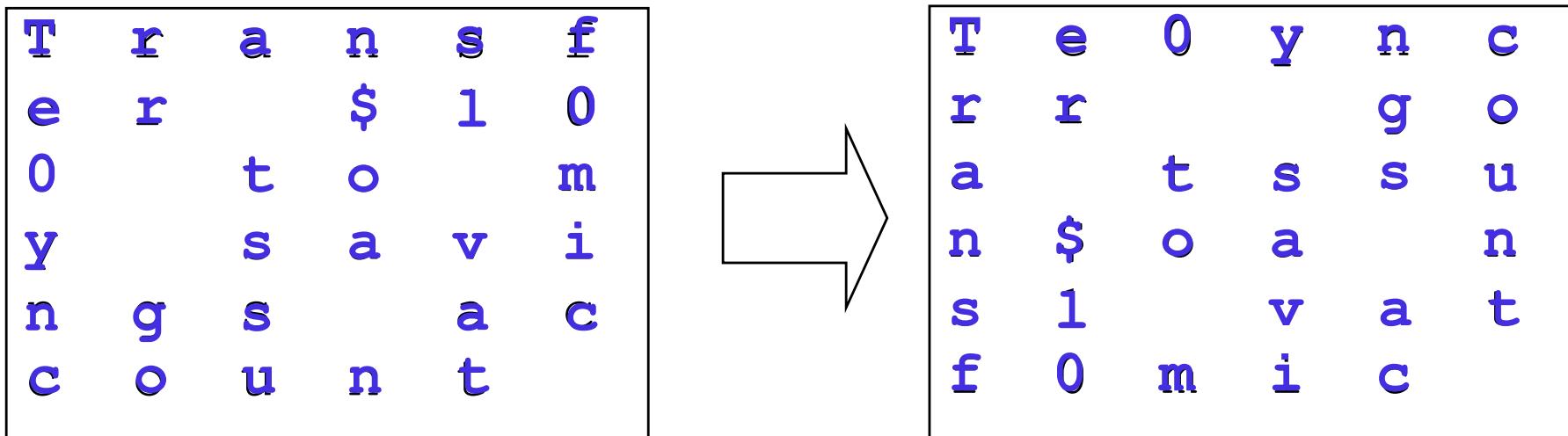
- Doesn't change the characters in the message
 - Just where they occur
- Thus, character frequency analysis doesn't help cryptanalyst

Columnar Transpositions

- Write the message characters in a series of columns
- Copy from top to bottom of first column, then second, etc.

Example of Columnar Substitution

How did this transformation happen?



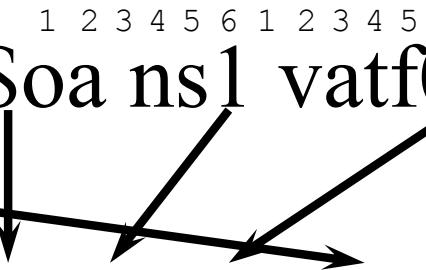
Looks a lot more cryptic written this way:

Te0yncrr goa tssun\$oa ns1 vatf0mic

Attacking Columnar Transformations

- The trick is figuring out how many columns were used
- Use information about digrams, trigrams, and other patterns
- Digrams are pairs of letters that frequently occur together (“re”, “th”, “en”, e.g.)
- For each possibility, check digram frequency

For Example,

4 5 6
Te0yncrr goa tssun\$oa ns1 vatf0mic
1 2 3 4 5 6 1 2 3 4 5 6 1 2 3

\$ 1 0 0

In our case, the presence of dollar signs and numerals in the text is suspicious

Maybe they belong together?

Umm, maybe there's 6 columns?

Double Transpositions

- Do it twice
- Using different numbers of columns
- How do you break it?
 - Find pairs of letters that probably appeared together in the plaintext
 - Figure out what transformations would put them in their positions in the ciphertext
- Can transform more than twice, if you want

Generalized Transpositions

- Any algorithm can be used to scramble the text
- Usually somehow controlled by a key
- Generality of possible transpositions makes cryptanalysis harder

Which Is Better, Transposition or Substitution?

- Well, neither, really
- Strong modern ciphers tend to use both
- Transposition scrambles text patterns
- Substitution hides underlying text characters/bits
- Combining them can achieve both effects
 - If you do it right . . .

Quantum Cryptography

- Using quantum mechanics to perform crypto
 - Mostly for key exchange
- Rely on quantum indeterminacy or quantum entanglement
- Existing implementations rely on assumptions
 - Quantum hacks have attacked those assumptions
- Not ready for real-world use, yet
- Quantum cryptanalysis even further off
 - And only helps on some ciphers

More on Cryptography

CS 136

Computer Security

Peter Reiher

January 14, 2021

Outline

- Desirable characteristics of ciphers
- Stream and block ciphers
- Cryptographic modes
- Uses of cryptography
- Symmetric and asymmetric cryptography
- Digital signatures

Desirable Characteristics of Ciphers

- Well matched to requirements of application
 - Amount of secrecy required should match labor to achieve it
- Freedom from complexity
 - The more complex algorithms or key choices are, the worse

More Characteristics

- Simplicity of implementation
 - Seemingly more important for hand ciphering
 - But relates to probability of errors in computer implementations
- Errors should not propagate

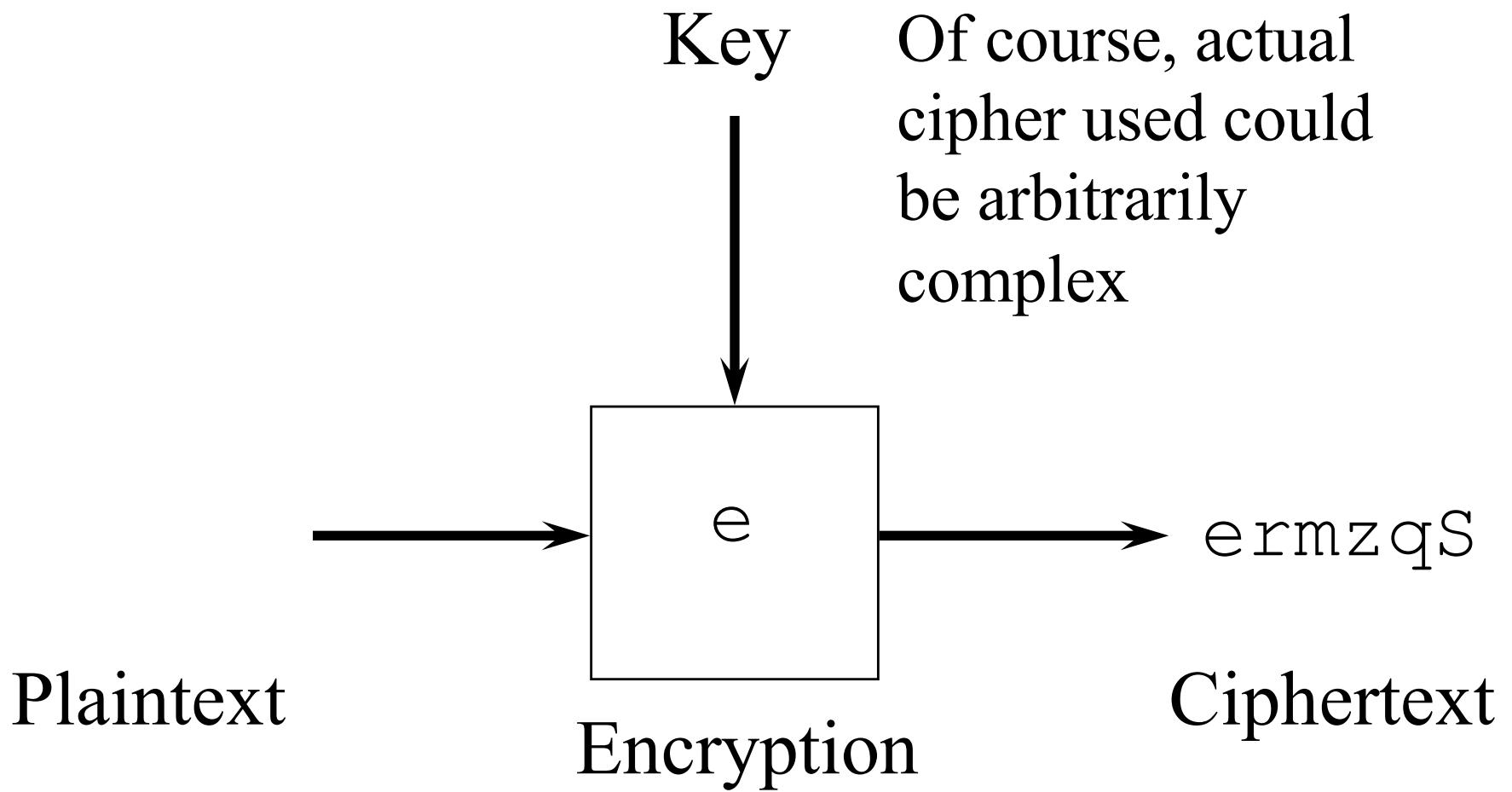
Yet More Characteristics

- Ciphertext size should be same as plaintext size
- Encryption should maximize *confusion*
 - Relation between plaintext and ciphertext should be complex
- Encryption should maximize *diffusion*
 - Plaintext information should be distributed throughout ciphertext

Stream and Block Ciphers

- Stream ciphers convert one symbol of plaintext immediately into one symbol of ciphertext
- Block ciphers work on a given sized chunk of data at a time

Stream Ciphers



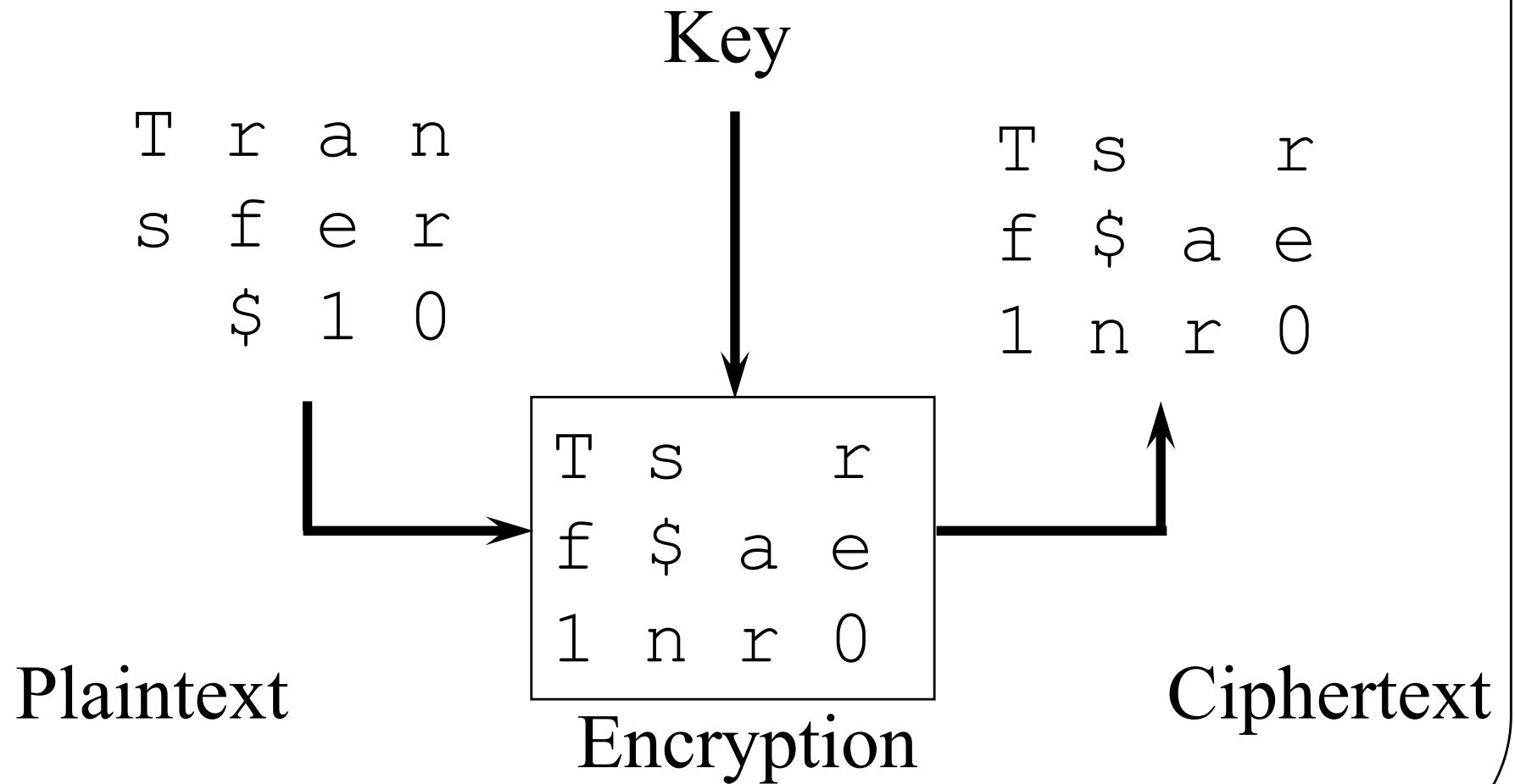
Advantages of Stream Ciphers

- + Speed of encryption and decryption
 - Each symbol encrypted as soon as it's available
- + Low error propagation
 - Errors affect only the symbol where the error occurred
 - Depending on *cryptographic mode*

Disadvantages of Stream Ciphers

- Low diffusion
 - Each symbol separately encrypted
 - Each ciphertext symbol only contains information about one plaintext symbol
- Susceptible to insertions and modifications
- Not good match for many common uses of cryptography
- Some disadvantages can be mitigated by use of proper cryptographic mode

Block Ciphers



Advantages of Block Ciphers

- + Good diffusion
 - Easier to make a set of encrypted characters depend on each other
- + Immunity to insertions
 - Encrypted text arrives in known lengths

Most common Internet crypto done with block ciphers

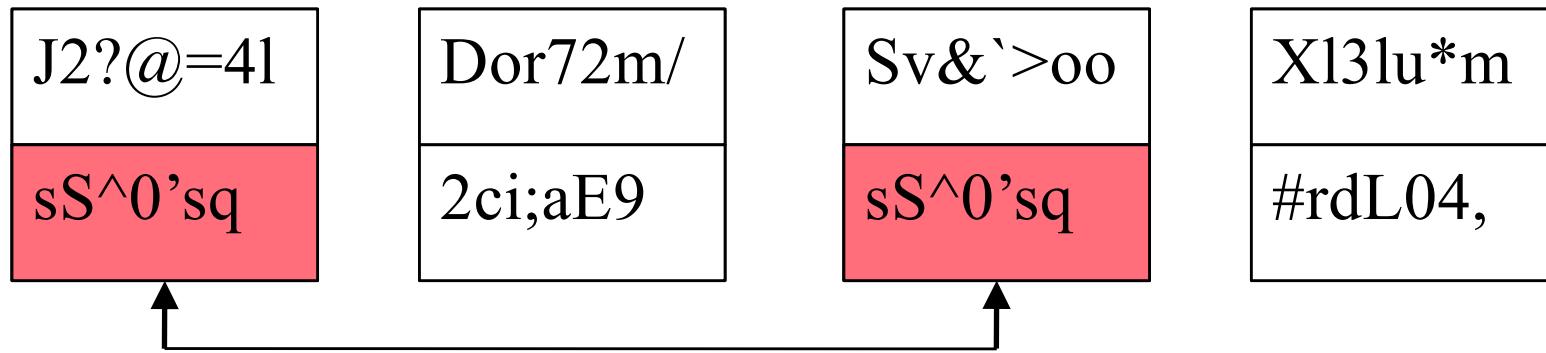
Disadvantages of Block Ciphers

- Slower
 - Need to wait for block of data before encryption/decryption starts
- Worse error propagation
 - Errors affect entire blocks

Cryptographic Modes

- Let's say you have a bunch of data to encrypt
 - Using the same cipher and key
- How do you encrypt the entire set of data?
 - Given block ciphers have limited block size
 - And stream ciphers just keep going

The Basic Situation



Let's say our block cipher has a block size of 7 characters and we use the same key for all

Now let's encrypt

There's something odd here . . .

Is this good?

Why did it happen?

Another Problem With This Approach

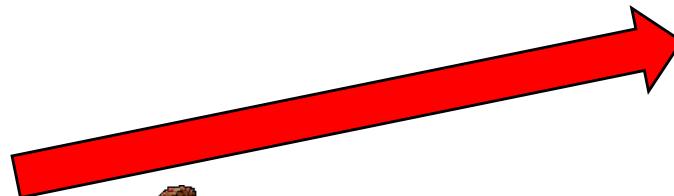
What if these are transmissions representing deposits into bank accounts?



Xl3lu*m
#rdL04,

Dor72m/
2ci;aE9

Insertion Attack!



1840326	450
2201568	5000
3370259	8900
5610993	1579
6840924	2725
8436018	10

What if account 5610993 belongs to him?

So far, so good . . .

What Caused the Problems?

- Each block of data was independently encrypted
 - With the same key
- So two blocks with identical plaintext encrypt to the same ciphertext
- Not usually a good thing
- We used the wrong *cryptographic mode*
 - Electronic Codebook (ECB) Mode

Cryptographic Modes

- A cryptographic mode is a way of applying a particular cipher
 - Block or stream
- The same cipher can be used in different modes
 - But other things are altered a bit
- A cryptographic mode is a combination of cipher, key, and feedback
 - Plus some simple operations

So What Mode Should We Have Used?

- Cipher Block Chaining (CBC) mode might be better
- Ties together a group of related encrypted blocks
- Hides that two blocks are identical
- Foils insertion attacks

Cipher Block Chaining Mode

- Adds feedback into encryption process
- The encrypted version of the previous block is used to encrypt this block
- For block $X+1$, XOR the plaintext with the ciphertext of block X
 - Then encrypt the result
- Each block's encryption depends on all previous blocks' contents
- Decryption is similar

What About the First Block?

- If we send the same first block in two messages with the same key,
 - Won't it be encrypted the same way?
- Might easily happen with message headers or standardized file formats
- CBC as described would encrypt the first block of the same message sent twice the same way both times

Initialization Vectors

- A technique used with CBC
 - And other crypto modes
 - Abbreviated IV
- Ensures that encryption results are always unique
 - Even for duplicate message using the same key
- XOR a random string with the first block
 - $\text{plaintext} \oplus IV$
 - Then do CBC for subsequent blocks

Encrypting With An IV

First block of message

1	1	0	1	0	0	0	1
---	---	---	---	---	---	---	---

Initialization vector

0	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

XOR IV and message

0	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---

Encrypt msg and send
IV plus message

Second block of message

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

Use previous msg for CBC

Apply CBC

1	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---

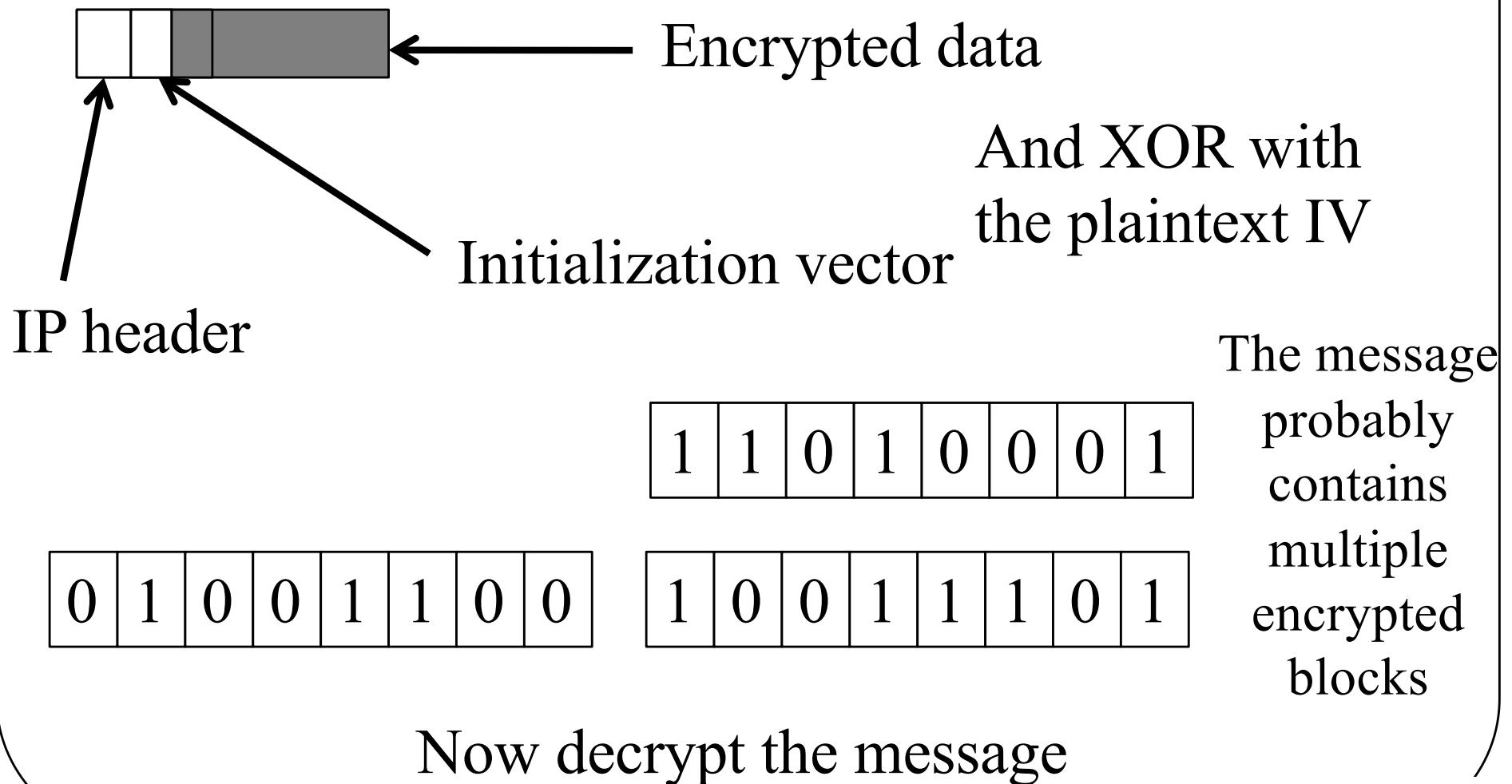
Encrypt and send second
block of msg

No need to also send 1st block again

How To Decrypt With Initialization Vectors?

- First block received decrypts to
$$P = \text{plaintext} \oplus IV$$
- $\text{plaintext} = P \oplus IV$
- No problem if receiver knows IV
 - Typically, IV is sent in the message
- Subsequent blocks use standard CBC
 - So can be decrypted that way

An Example of IV Decryption



For Subsequent Blocks



Use previous ciphertext
block instead of IV

And XOR with the
previous ciphertext block

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

0	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Now decrypt the message

Some Important Crypto Modes

- Electronic codebook mode (ECB)
- Cipher block chaining mode (CBC)
- Cipher-feedback mode (CFB) and
Output-feedback mode (OFB)
Both convert block to stream cipher

Uses of Cryptography

- What can we use cryptography for?
- Lots of things
 - Secrecy
 - Authentication
 - Prevention of alteration

Cryptography and Secrecy

- Pretty obvious
- Only those knowing the proper keys can decrypt the message
 - Thus preserving secrecy
- Used cleverly, it can provide other forms of secrecy

Cryptography and Authentication

- How can I prove to you that I created a piece of data?
- What if I give you the data in encrypted form?
 - Using a key only you and I know
- Then only you or I could have created it
 - Unless one of us told someone else the key . . .

Using Cryptography for Authentication

- If both parties cooperative, standard cryptography can authenticate
 - Problems with *non-repudiation*, though
- What if three parties want to share a key?
 - No longer certain who created anything
 - Public key cryptography can solve this problem
- What if I want to prove authenticity without secrecy?

Cryptography and Non-Alterability

- Changing one bit of an encrypted message completely garbles it
 - For many forms of cryptography
- If a checksum is part of encrypted data, that's detectable
- If you don't need secrecy, can get the same effect
 - By encrypting only the checksum

Symmetric and Asymmetric Cryptosystems

- Symmetric - the encrypter and decrypter share a secret key
 - Used for both encrypting and decrypting
- Asymmetric – encrypter has different key than decrypter

Description of Symmetric Systems

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

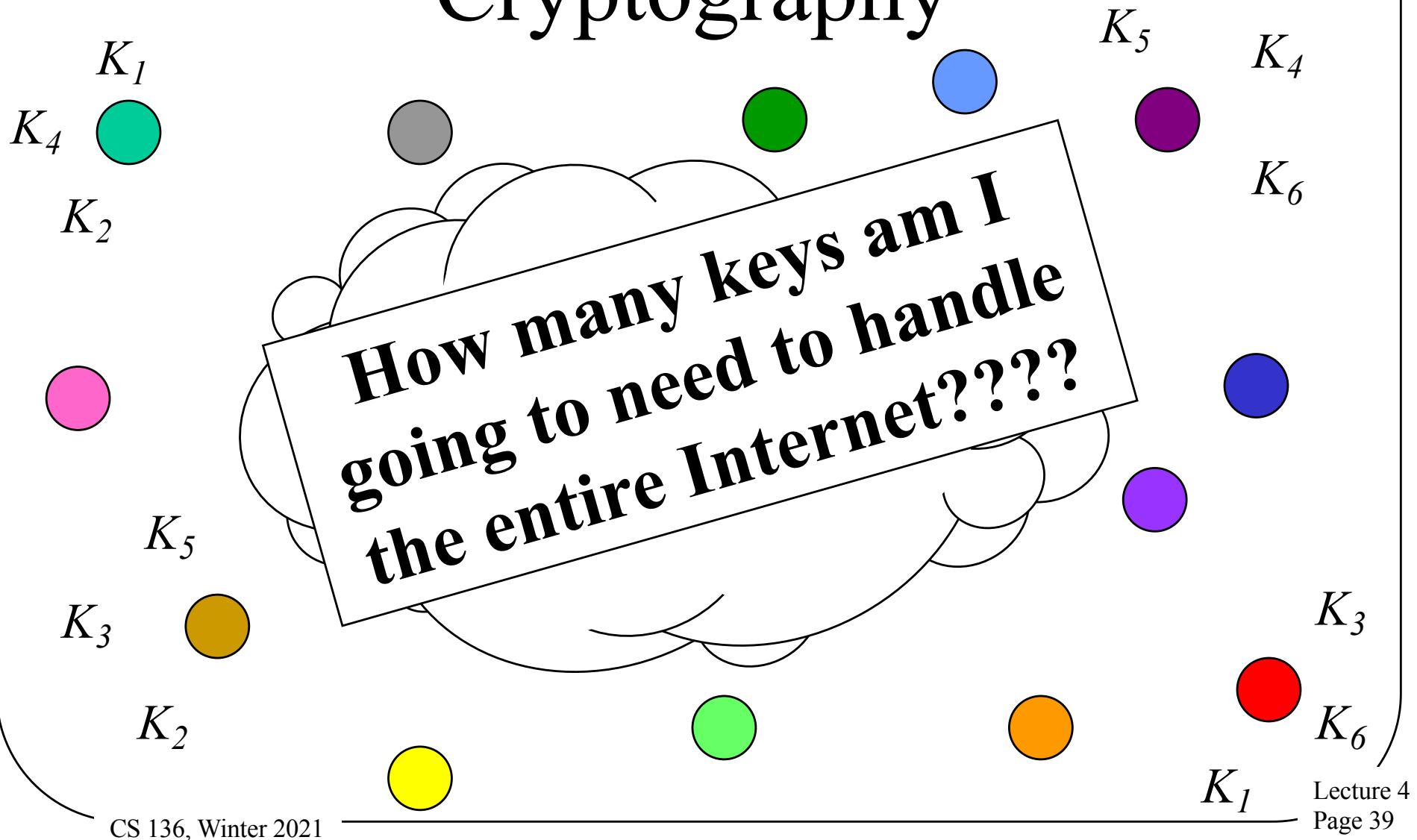
Advantages of Symmetric Key Systems

- + Encryption and authentication performed in a single operation
- + Well-known (and trusted) ones perform faster than asymmetric key systems
- + Doesn't require any centralized authority
 - Though key servers help a lot

Disadvantage of Symmetric Key Systems

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

Scaling Problems of Symmetric Cryptography



Sample Symmetric Key Ciphers

- The Data Encryption Standard
- The Advanced Encryption Standard
- There are many others

The Data Encryption Standard

- Well known symmetric cipher
- Developed in 1977, still much used
 - Shouldn't be, for anything serious
- Block encryption, using substitutions, permutations, table lookups
 - With multiple *rounds*
 - Each round is repeated application of operations
- Only serious problem based on short key

The Advanced Encryption Standard

- A relatively new cryptographic algorithm
- Intended to be the replacement for DES
- Chosen by NIST
 - Through an open competition
- Chosen cipher originally called Rijndael
 - Developed by Dutch researchers
 - Uses combination of permutation and substitution
- 128 or 256 bit keys

Increased Popularity of AES

- Gradually replacing DES
 - As was intended
- Various RFCs describe using AES in IPsec
- FreeS/WAN IPsec (for Linux) includes AES
- Most commercial VPNs use AES
- Used in modern Windows and Mac OS systems

Is AES Secure?

- No complete breaks discovered so far
- But some disturbing problems
 - Attacks that work on versions of AES using fewer rounds
 - Attacks that get keys quicker than brute force
 - But not practical time (e.g., in 2^{126} operations)
- But unusable crypto flaws often lead to usable ones
- Attacks on crypto only get better over time, never worse

Public Key Encryption Systems

- The encrypter and decrypter have different keys

$$C = E(K_E, P)$$

$$P = D(K_D, C)$$

- Often, works the other way, too

$$C' = E(K_D, P)$$

$$P = D(K_E, C')$$

Practical Use of Public Key Cryptography

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

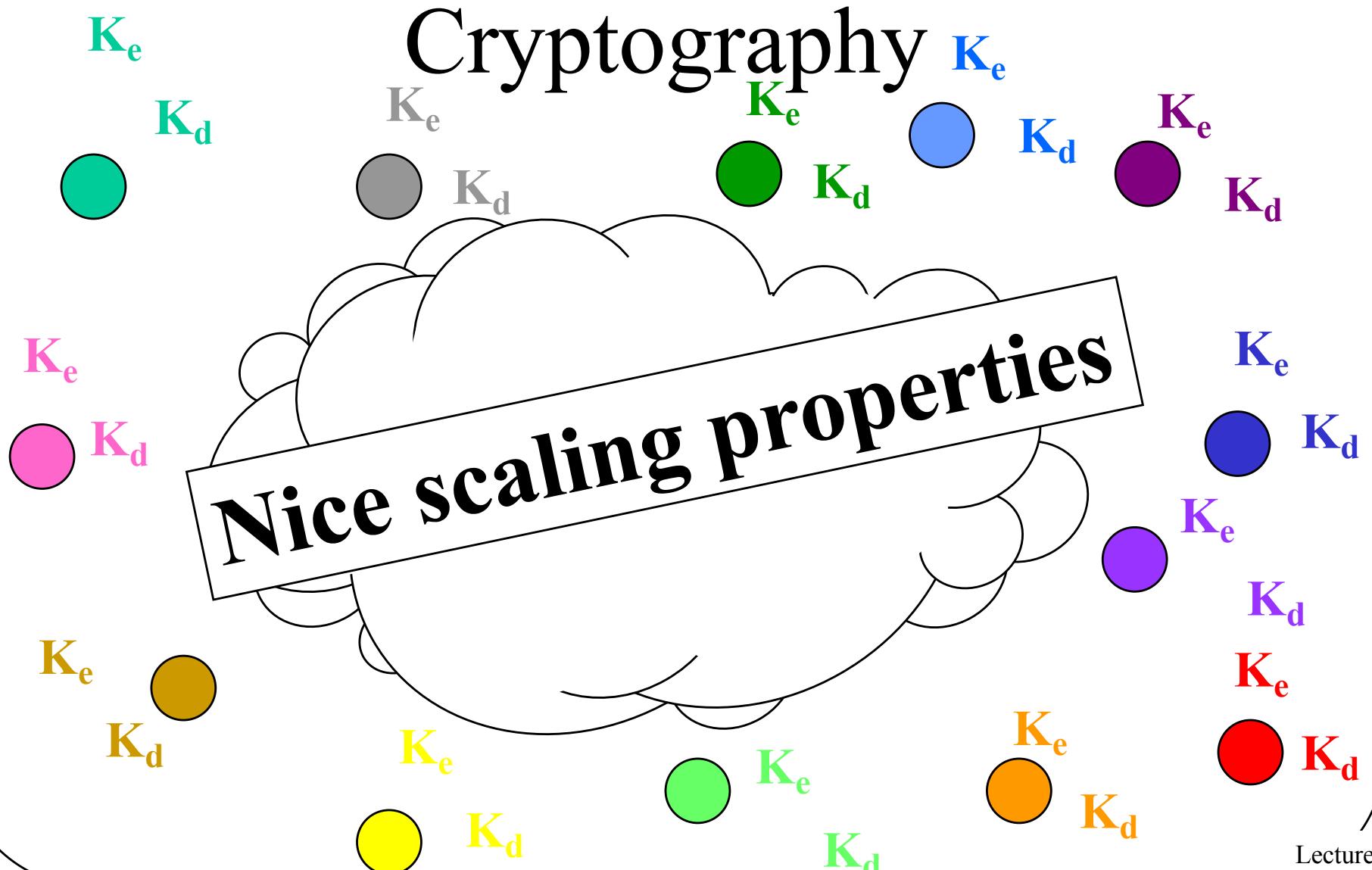
Authentication With Shared Keys

- If only two people know the key, and I didn't create a properly encrypted message -
 - The other guy must have
- But what if he claims he didn't?
- Or what if there are more than two?
- Requires authentication servers

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

Scaling of Public Key



Key Management Issues

- 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”

Issues of Key Publication

- Security of public key cryptography depends on using the right public key
- If I am fooled into using the wrong one, that key's owner reads my message
- Need high assurance that a given key belongs to a particular person
- Which requires a *key distribution infrastructure*

RSA Algorithm

- Most popular public key cryptographic algorithm
- In wide use
- Has withstood much cryptanalysis
- Based on hard problem of factoring large numbers

RSA Keys

- Keys are functions of a pair of 100-200 digit prime numbers
- Relationship between public and private key is complex
- Recovering plaintext without private key (even knowing public key) is supposedly equivalent to factoring product of the prime numbers

Comparison of AES and RSA

- AES is much more complex
- However, AES uses only simple arithmetic, logic, and table lookup
- RSA uses exponentiation to large powers
 - Computationally 1000 times more expensive in hardware, 100 times in software
- RSA key selection also much more expensive

Is RSA Secure?

- Conjectured that security depends on factoring large numbers
 - But never proven
 - Some variants proven equivalent to factoring problem
- Probably the conjecture is correct
- Key size for RSA doesn't have same meaning as DES and AES

PK and Brute Force Attacks

- Nobody performs brute force attacks on PK systems
 - Because there's an easier way
 - Remember the principle of easiest penetration?
- Attack the mathematical relationship between public and private key
- Instead of checking all 2^{2048} keys

Attacks on Factoring RSA Keys

- In 2005, a 663 bit RSA key was successfully factored
- A 768 bit key factored in 2009
- Research on integer factorization suggests keys up to 2048 bits may be insecure
- Insecure key length will only increase
- The longer the key, the more expensive the encryption and decryption

Elliptic Curve Cryptography

- RSA and similar algorithms related to factoring products of large primes
- Other math can be used for PK, instead
 - Properties of elliptic curves, e.g.
- Can give same security as other public key schemes, with much smaller keys
- Widely studied, regarded as safe
 - But the NSA is pushing it . . .
 - Often used for small devices

Combined Use of Symmetric and Asymmetric Cryptography

- 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 AES with that session key for the rest of the transmission

Combining Symmetric and Asymmetric Crypto

Alice wants to share
the key only with Bob



But there are problems we'll discuss later



Alice

K_{EA}

K_{DA}

K_{EB}

K_S

Only Bob

can decrypt it

Bob

Only Alice could

have created it

K_{EB}

K_{DB}

K_{EA}

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

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

$$K_S = D(C, K_{DB}, K_{EA})$$

Digital Signature Algorithms

- In some cases, secrecy isn't required
- But authentication is
- The data must be guaranteed to be that which was originally sent
- Especially important for data that is long-lived

Desirable Properties of Digital Signatures

- Unforgeable
- Verifiable
- Non-repudiable
- Cheap to compute and verify
- Non-reusable
- No reliance on trusted authority
- Signed document is unchangeable

Encryption and Digital Signatures

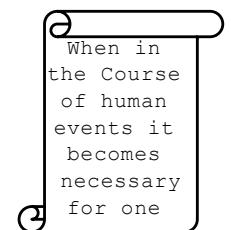
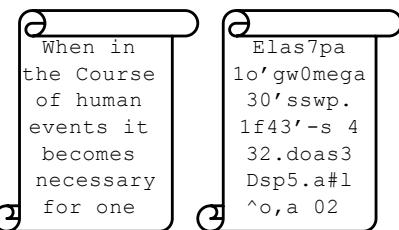
- Digital signature methods are based on encryption
- The basic act of having performed encryption can be used as a signature
 - If only I know K , then $C=E(P,K)$ is a signature by me
 - But how to check it?

Signatures With Shared Key Encryption

- Requires a trusted third party
- Signer encrypts document with secret key shared with third party
- Receiver checks validity of signature by consulting with trusted third party
- Third party required so receiver can't forge the signature

For Example,

K_S



K_S

Signatures With Public Key Cryptography

- Signer encrypts document with his private key
- Receiver checks validity by decrypting with signer's public key
- Only signer has the private key
 - So no trusted third party required
- But receiver must be certain that he has the right public key

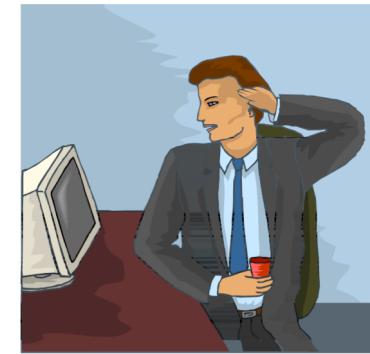
For Example,

K_d



When in
the Course
of human
events it
becomes
necessary
for one

Elas7pa
1o'gw0mega
30'sswp.
1f43'-s 4
32.doas3
Dsp5.a#1
^o,a 02



K_e

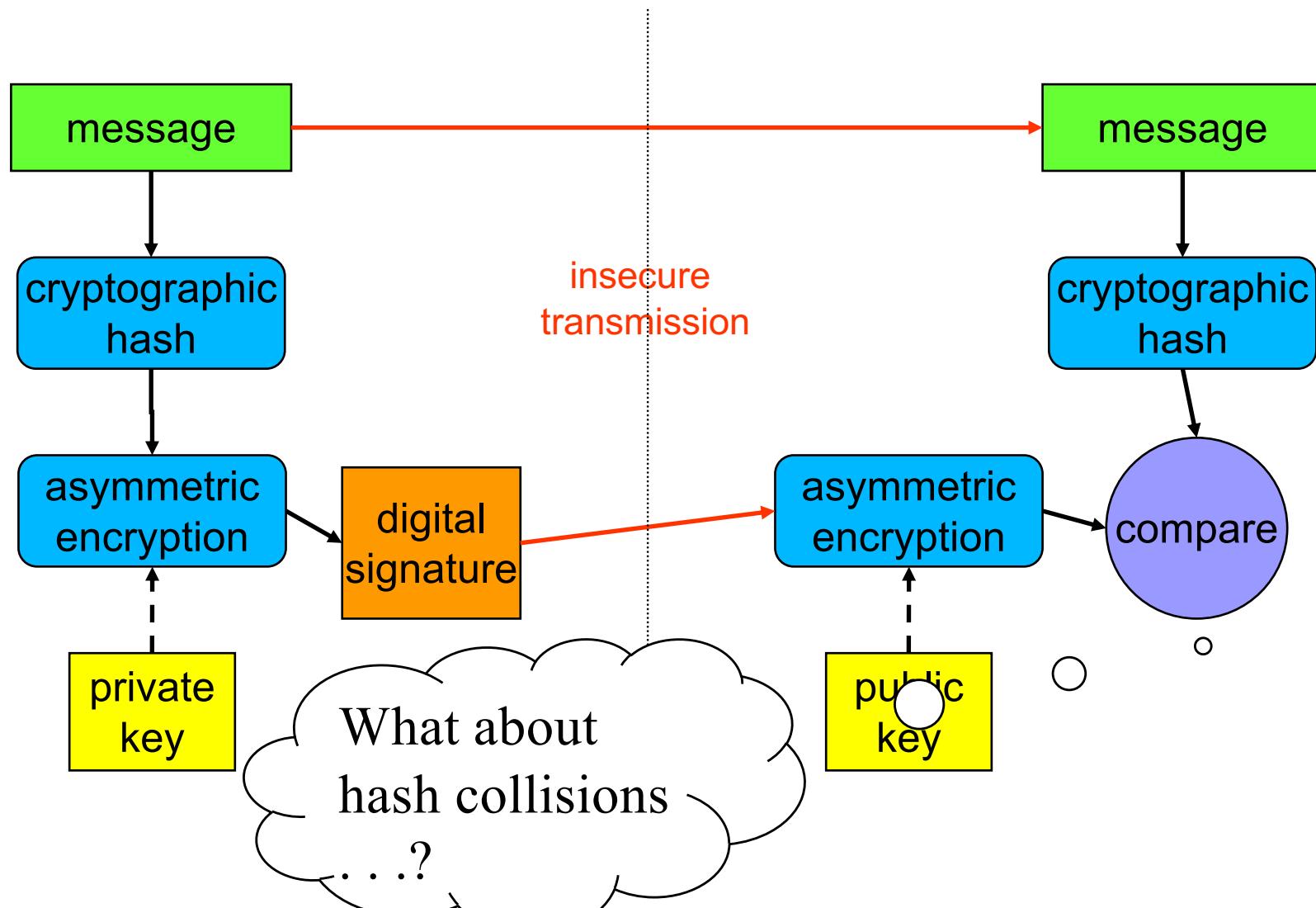
Alice's
public
key

When in
the Course
of human
events it
becomes
necessary
for one

Problems With Simple Encryption Approach

- Computationally expensive
 - Especially with public key approach
- Document is encrypted
 - Must be decrypted for use
 - If in regular use, must store encrypted and decrypted versions

Signing a Hash Only



More on Cryptography

CS 136

Computer Security

Peter Reiher

January 19, 2021

Outline

- Properties of keys
- Certificates
- Key management

Introduction

- It doesn't matter how strong your encryption algorithm is
- If the opponents can get hold of your keys, your security is gone
- Proper use of keys is crucial to security in computing systems
- Ciphers don't get cracked often, but keys get leaked all the time

Properties of Keys

- Length
- Randomness
- Lifetime
- Secrecy
- Generation

Key Length

- If your cryptographic algorithm is otherwise perfect, its strength depends on key length
- Since the only attack is a brute force attempt to discover the key
- The longer the key, the more brute force required

Are There Real Costs for Key Length?

- Generally, more bits is more secure
- Why not a whole lot of key bits, then?
- Some encryption done in hardware
 - More bits in hardware costs more
- Some software encryption slows down as you add more bits, too
 - Public key cryptography especially
- Some algorithms have defined key lengths only
- If the attack isn't brute force, key length might not help

Key Randomness

- Brute force attacks assume you chose your key at random
- If attacker learns how you chose your key
 - He can reduce brute force costs
- How good is your random number generator?

Generating Random Keys

- Well, don't use `rand()`¹
- The closer the method chosen approaches true randomness, the better
- But, generally, don't want to rely on exotic hardware
- True randomness is not essential
 - Need same statistical properties
 - And non-reproducibility

¹See <http://eprint.iacr.org/2013/338.pdf> for details

Cryptographic Methods

- Start with a random number
5924
- Use a cryptographic hash on that number

1a401eca764d2588e5e02067518a1460e333f91d27089903f3658cde0e0bee79

- If the cryptographic hash is a good one, the new number looks pretty random
- Extract what you need from it

1a401eca764d2588e5e02067518a1460e333f91d27089903f3658cde0e0bee79

- Produce new keys by hashing old ones

6028fe87c4df480b74976ed9207142d4d20c4a271c851812d5771d38d54b2488

Is This Method Secure?

- It depends (in part) on the strength of hash algorithm
 - Does it produces “random” numbers?
 - Is the part you extract itself random?
- Another problem: It falls apart if any key is ever broken or divulged
 - Doesn’t have *perfect forward secrecy*

Perfect Forward Secrecy

- A highly desirable property in a cryptosystem
- It means that the session key will not compromise any other
 - E.g., don't derive one key from another using a repeatable algorithm
- Keys do get divulged, so minimize the resulting damage

SHA256 hash of 5924 is always
1a401eca764d2588e5e02067518a1460e333f91d27089903f3658cde0e0bee79

Random Noise

- Observe an event that is likely to be random
 - Physical processes (cosmic rays, etc.)
 - Real world processes (variations in disk drive delay, keystroke delays, etc.)
- Assign bit values to possible outcomes
- Record or generate them as needed
- More formally described as *gathering entropy*
- Keys derived with proper use of randomness have good perfect forward secrecy

On Users and Randomness

- Some crypto packages require users to provide entropy
 - To bootstrap a pool of random numbers
 - Users do this:
 - They usually try to make things simple
 - And not really care about randomness
 - Better to have the crypto package get its own entropy
- But they can screw up, too: recent case of SecureRandom() bug causing skipping of entropy gathering.
Result: some cryptocurrencies used insecure keys.

Don't Go Crazy on Randomness

- Make sure you actually get a new key every time
 - A surprisingly common flaw
- Make sure it's non-reproducible
 - So attackers can't play it back
- Make sure there aren't obvious patterns
- Attacking truly unknown patterns in fairly random numbers is extremely challenging
 - They'll probably mug you, instead

Key Lifetime

- If a good key's so hard to find,
 - Why every change it?
- How long should one keep using a given key?

Why Change Keys?

- Long-lived keys are more likely to be compromised
- The longer a key lives, the more data is exposed if it's compromised
- The longer a key lives, the more resources opponents can (and will) devote to breaking it
- The more a key is used, the easier the cryptanalysis on it

**A secret that cannot be readily changed should
be regarded as a vulnerability**

Practicalities of Key Lifetimes

- In some cases, changing keys is inconvenient
 - E.g., encryption of data files
- Keys used for specific communications sessions should be changed often
 - E.g., new key for each VoIP call
- Keys used for key distribution can't be changed too often
- Some keys must be stored permanently or at least for a long time

Key Storage

- Symmetric session keys
 - Avoid storing them permanently
 - Get rid of them when session ends
- Long term symmetric keys
 - E.g., for disk encryption
 - Safe storage is critical
- Private asymmetric keys
 - Usually require long-term storage
 - Safe storage is critical

Storing a User's Keys

- Where are a user's keys kept?
 - Given they must be used often
- Permanently on the user's machine?
 - What happens if the machine is cracked?
- People can't remember random(ish) keys
 - Hash keys from passwords/passphrases?
- Keep keys on smart cards or hardware security enclaves?
- Get them from key servers?
 - Not a popular solution

Destroying Old Keys

- Never keep a key around longer than necessary
 - Gives opponents more opportunities
- Destroy keys securely
 - For computers, remember that information may be in multiple places
 - Caches, virtual memory pages, freed file blocks, stack frames, etc.
 - Real modern attacks based on finding old keys in unlikely places

Key Secrecy

- Seems obvious
- Of course you keep your keys secret
- However, not always handled well in the real world
- Particularly with public key cryptography

Some Problems With Key Sharing

- Private keys are often shared
 - Same private key used on multiple machines
 - For multiple users
 - Stored in “convenient” places
 - Perhaps backed up on tapes in plaintext form

Why Do People Do This?

- For convenience
- To share expensive certificates
- Because they aren't thinking clearly
- Because they don't know any better
- A recent example:
 - RuggedCom's Rugged Operating System for power plant control systems
 - Private key embedded in executable

To Make It Clear,

- **PRIVATE KEYS ARE PRIVATE!**
- They are for use by a single user
- They should never be shared or given away
- They must never be left lying around in insecure places
 - Widely distributed executables are insecure
 - Just because it's tedious to decipher executables doesn't mean can't be done
- The entire security of PK systems depends on the secrecy of the private key!

Key Generation

- How do you get a key in the first place?
- For PK ciphers, predefined key generation algorithms
- For local use symmetric keys, random/pseudorandom methods
- How about for shared session keys?

The Shared Session Key Problem

- Two parties wish to communicate securely across a network
- They don't have a symmetric key
- What do they do to get one?
 - Not just to generate one
 - But to make sure both have it
 - And no one else does

Key Exchange Protocols

- Network protocols that solve this problem
- Some are based on trusted servers
- Others use PK methods
- There is a way to do it without either trusted servers or PK . . .

Diffie/Hellman Key Exchange

- Securely exchange a key
 - Without previously sharing any secrets
 - No PK available
 - No other symmetric key either
- Using an insecure channel
 - I.e., the bad guys can hear everything the good guys tell each other

Exchanging a Key in Diffie/Hellman

- How can Alice and Bob possibly get a shared secret session key in this case?
- First, they set things up
- Alice and Bob agree on a large prime n and a number g
 - g should be primitive root mod n
- n and g don't need to be secrets
 - Typically predefined in their software

Exchanging the Key, Con't

- Alice secretly chooses a large random integer x and sends Bob $X = g^x \text{mod } n$
 - Bob secretly chooses a random large integer y and sends Alice $Y = g^y \text{mod } n$
 - Alice computes $k = \overline{Y^x \text{mod } n}$
 - Bob computes $k' = \overline{X^y \text{mod } n}$
 - k and k' are both equal to $g^{xy} \text{mod } n$
 - But nobody else can compute k or k'
-
- Alice knows x
- Bob knows y

Why Can't Others Get the Secret?

- What do they know?
 - n , g , X , and Y
 - Not x or y
- Knowing X and y gets you k
- Knowing Y and x gets you k'
 - $k = k'$
- Knowing X and Y gets you nothing
 - Unless you compute the discrete logarithm to obtain x or y
 - Which is believed to be hard

A Snake in Diffie/Hellman’s Grass

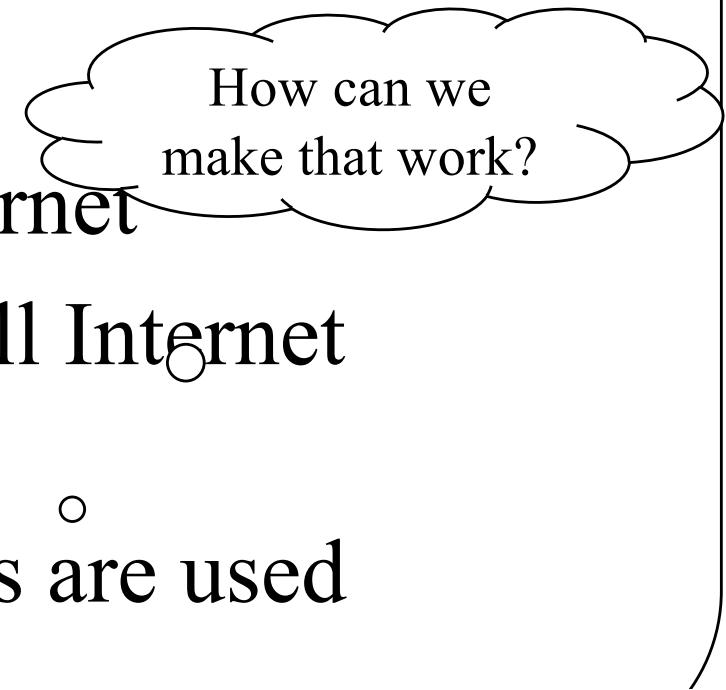
- Diffie/Hellman “guarantees” that two parties share a secret
- But it doesn’t guarantee who those two parties are
- How does Alice know whether the Y she heard actually was sent by Bob?
 - What if it was sent by an attacker?

The Core of the Problem

- Diffie/Hellman does not authenticate the parties
 - It does not provide evidence of who is talking
- To get the real effect you want, you need to authenticate the parties
 - Either during the key exchange
 - Or after you've got the encrypted session

The Ubiquity of the Problem

- You actually always require authentication in any key distribution
- Otherwise, you don't know who else has the key
- A core problem for the Internet
 - At the base of securing all Internet commerce
- But Diffie-Hellman variants are used everywhere in the Internet^o



Authentication for Key Distribution

- You have a key on one machine
- You need to get it to another machine
- Since it's over a network, you need assurance that it's the right key
 - From the intended party
- How to get that assurance?

Basic Approaches

- Key servers
- Certificates

Key Servers

- Machines whose job it is to distribute keys to other machines
- Clients can authenticate themselves to the key server
- Key server can authenticate itself to the clients
- So clients need not authenticate other clients
- Note the transitive trust issue here
- Not the popular solution

Certificates

- Server-less authentication
 - Used throughout the Internet
- What is a certificate?
- A signed electronic document proving you are who you claim to be
- Most often used to solve key distribution problem

Public Key Certificates

- The most common kind of certificate
- Addresses the biggest challenge in widespread use of public keys
 - How do I know whose key it is?
- Essentially, a copy of your public key signed by a trusted authority
- Presentation of the certificate alone serves as authentication of your public key

Implementation of Public Key Certificates



- Set up a universally trusted authority
- Every user presents his public key to the authority
- The authority returns a certificate
 - Containing the user's public key signed by the authority's private key
- In essence, a special type of key server

Checking a Certificate

- Every user keeps a copy of the authority's public key
- When a new user wants to talk to you, he gives you his certificate
- Decrypt the certificate using the authority's public key
- You now have an authenticated public key for the new user
- Authority need not be checked on-line

Certificates and Diffie-Hellman

- Certificates are used to solve the Diffie-Hellman authentication problem
- Each party includes a digital signature of its number (X or Y)
- The other party extracts the sender's PK from its certificate
- And then checks the digital signature
 - Giving assurance of who you're exchanging a key with

Well, Sort Of

- Not what's usually done in the Internet
- Common case is ordinary user wants to talk to a big server
- Ordinary users don't have certificates
- Big servers do
- So the server signs and the user doesn't

Consider the implications of that . . .

Scaling Issues of Certificates

- If there are 1-2 billion Internet users needing certificates, can one authority serve them all?
- Probably not
- So you need multiple authorities
- Does that mean everyone needs to store the public keys of all authorities?

Certification Hierarchies

- Arrange certification authorities hierarchically
- Single authority at the top produces certificates for the next layer down
- And so on, recursively

Using Certificates From Hierarchies

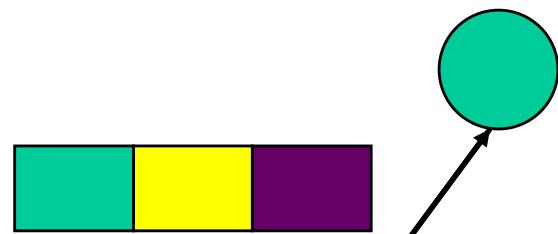
- I get a new certificate
- I don't know the signing authority
- But the certificate also contains that authority's certificate
- Perhaps I know the authority who signed this authority's certificate

Extracting the Authentication

- Using the public key of the higher level authority,
 - Extract the public key of the signing authority from the certificate
- Now I know his public key, and it's authenticated
- I can now extract the user's key and authenticate it

A Example

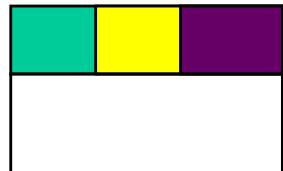
Alice gets a message with a certificate



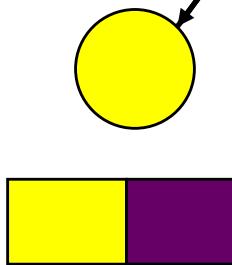
Then she uses [yellow] to check [teal]

Should Alice believe that he's really [teal]?

So she uses [purple] to check [yellow]



Alice has never heard of [yellow]. But she has heard of [purple].



Give me a certificate saying that I'm [teal]

How can [teal] prove who he is?

Certification Hierarchies Reality

- We don't use a single certification hierarchy
- Instead, we rely on large numbers of independent certifying authorities
 - Each of which may have its own internal hierarchy
- Essentially, a big list
- Is this really better?

Certificates and Trust

- Ultimately, the point of a certificate is to determine if something is trusted
 - Do I trust the request enough to perform some financial transaction?
- So, Trustysign.com signed this certificate
- How much confidence should I have in the certificate?

Potential Problems in the Certification Process

- What measures did Trustysign.com use before issuing the certificate?
- Is the certificate itself still valid?
- Is Trustysign.com's signature/certificate still valid?
- Who is trustworthy enough to be at the top of the hierarchy?

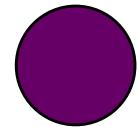
Trustworthiness of Certificate Authority

- How did Trustysign.com issue the certificate?
- Did it get an in-person sworn affidavit from the certificate's owner?
- Did it phone up the owner to verify it was him?
- Did it just accept the word of the requestor that he was who he claimed to be?
- Has authority been compromised?

What Does a Certificate Really Tell Me?

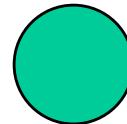
- That the certificate authority (CA) tied a public/private key pair to identification information
- Generally doesn't tell me why the CA thought the binding was proper
- I may have different standards than that CA

Showing a Problem Using the Example

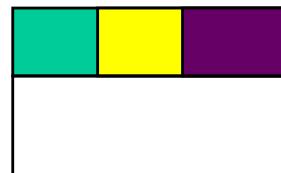


Alice likes how █ verifies identity

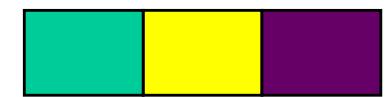
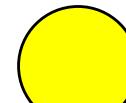
What if █ uses █'s lax policies to pretend to be █ ?



But is she equally happy with how █ verifies identity?



Does she even know how █ verifies identity?



Another Big Problem

- Things change
 - E.g., compromise of Adobe private keys
- One result of change is that what used to be safe or trusted isn't any more
- If there is trust-related information out in the network, what will happen when things change?

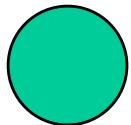
Revocation

- A general problem for keys, certificates, etc.
- How does the system revoke something related to t
 - In a network environment
 - Safely, efficiently, etc.
 - Related to revocation problem for capabilities

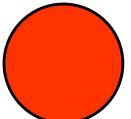
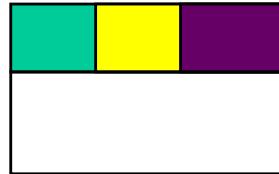
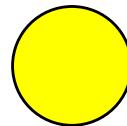
Revocation in a distributed system is always hard

Revisiting Our Example

Someone discovers
that  has obtained
a false certificate for

How does Alice make sure
that she's not accepting 's
false certificate?



Realities of Certificates

- Most OSes (or browsers) come with set of “pre-trusted” certificate authorities
- System automatically processes (i.e., trusts) certificates they sign
- Usually no hierarchy
- If not signed by one of these, present it to the user
 - Who always accepts it . . .

An Example

- Firefox web browser
- Makes extensive use of certificates to validate entities
 - As do all web browsers
- Comes preconfigured with several certificate authorities
 - Nearly 100 of them
 - Many are at the top of a hierarchy
 - So there are several hundred certificates

Firefox Preconfigured Certificate Authorities

- Some you'd expect:
 - Microsoft, RSA Security, Verisign, etc.
- Some you've probably never heard of:
 - Unizeto Sp. z.o.o., Netlock Kft., Krajowa Izba Rozliczeniowa S.A.

The Upshot

- If Netlock Kft. says someone's OK, I trust them
 - I've never heard of Netlock Kft.
 - I have no reason to trust Netlock Kft.
 - But my system's security depends on them

The Problem in the Real World

- In 2011, a Dutch authority (DigiNotar) was compromised
- Attackers generated lots of bogus certificates signed by DigiNotar
 - “Properly” signed by that authority
 - For popular web sites
- Until compromise discovered, everyone trusted them

Effects of DigiNotar Compromise

- Attackers could transparently redirect users to fake sites
 - What looked like Twitter was actually attacker's copycat site
- Allowed attackers to eavesdrop without any hint to users
- Apparently used by authorities in Iran to eavesdrop on dissidents

How Did the Compromise Occur?

- DigiNotar had crappy security
 - Out-of date antivirus software
 - Poor software patching
 - Weak passwords
 - No auditing of logs
 - Poorly designed local network
 - A company providing security services paid little attention to security
- But how
were you
supposed to
know that?*

Another Practicality

- Certificates have expiration dates
 - Important for security
 - Otherwise, long-gone entities would still be trusted
- But perfectly good certificates also expire
 - Then what?

The Reality of Expired Certificates

- When I hear my server's certificate has expired, what do I do?
 - I trust it anyway
 - After all, it's my server
- But pretty much everyone does that
 - For pretty much every certificate
- Not so secure

The Core Problem With Certificates

- Anyone can create some certificate
- Typical users have no good basis for determining whose certificates to trust
 - They don't even really understand what they mean
- Therefore, they trust almost any certificate

Should We Worry About Certificate Validity?

- Starting to be a problem
 - Stuxnet is one example
 - Compromise of DigiNotar and Adobe also
 - Recent report of attackers deceptively obtaining certificates for corporate executives
- Not the way most attackers break in today
- With all their problems, still not the weakest link
 - But now being exploited, mostly by most sophisticated adversaries

One Approach

- OCSP
- An online system that indicates if certificates have been revoked
- Used in different ways by different OSes and browsers
- Obviously requires network access
 - And ability to reach OCSP site

Security Protocols
CS 136
Computer Security
Peter Reiher
January 21, 2021

Outline

- Designing secure protocols
- Key exchange protocols
- Common security problems in protocols

Basics of Security Protocols

- Assume (usually) that your encryption is sufficiently strong
- Given that, how do you design a message exchange to achieve a given result securely?
- Not nearly as easy as you probably think
- Many of the concepts are important in many areas of computer/network security

Security Protocols

- A series of steps involving two or more parties designed to accomplish a task with suitable security
- Sequence is important
- Cryptographic protocols use cryptography
 - Most useful protocols are cryptographic
- Different protocols assume different levels of trust between participants

Types of Security Protocols

- Arbitrated protocols
 - Involving a trusted third party
- Adjudicated protocols
 - Trusted third party, after the fact
- Self-enforcing protocols
 - No trusted third party

Participants in Security Protocols



Alice



Bob

And the Bad Guys



Eve

Who only listens
passively



And sometimes
Alice or Bob
might cheat



Mallory

Who is actively
malicious

Trusted Arbitrator



Trent

A disinterested third party trusted by all legitimate participants

Arbitrators often simplify protocols, but add overhead and may limit applicability

Goals of Security Protocols

- Each protocol is intended to achieve some very particular goal
 - Like setting up a key between two parties
- Protocols may only be suitable for that particular purpose
- Important secondary goal is minimalism
 - Fewest possible messages
 - Least possible data
 - Least possible encryption

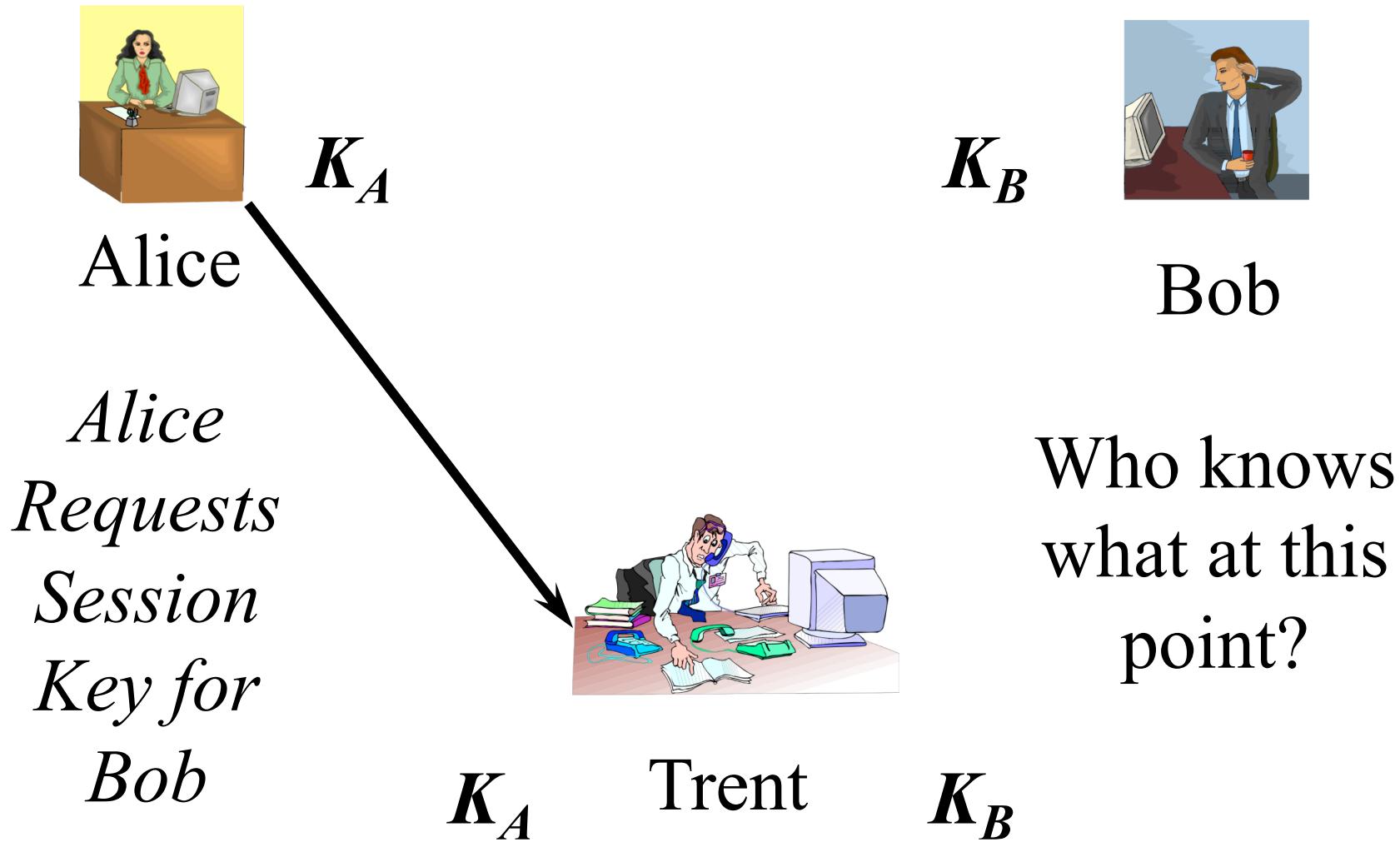
Key Exchange Protocols

- Often we want a different encryption key for each communication session
- How do we get those keys to the participants?
 - Securely
 - Quickly
 - Even if they've never communicated before

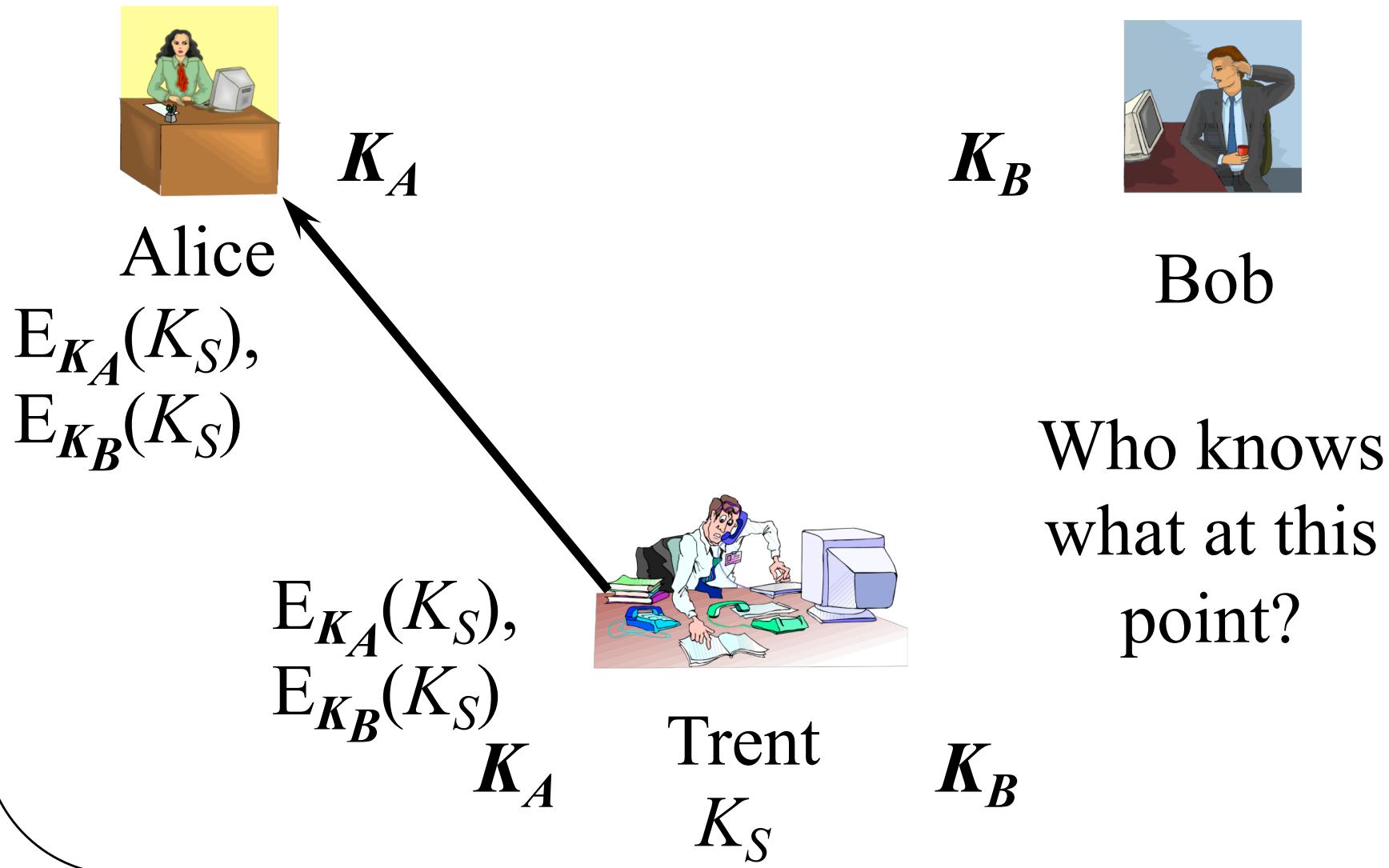
Key Exchange With Symmetric Encryption and an Arbitrator

- Alice and Bob want to talk securely with a new key
- They both trust Trent
 - Assume Alice & Bob each share a key with Trent
- How do Alice and Bob get a shared key?

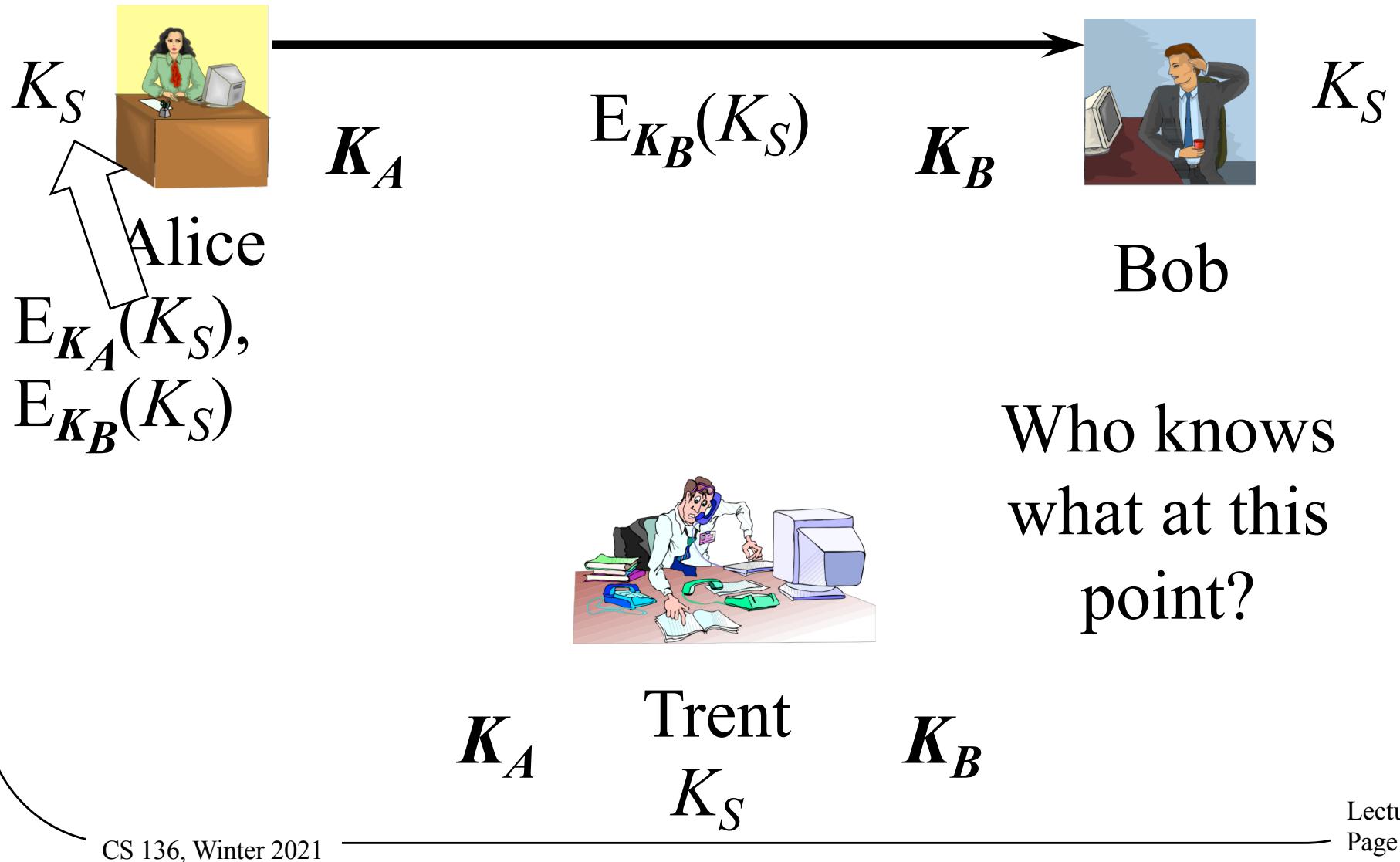
Step One



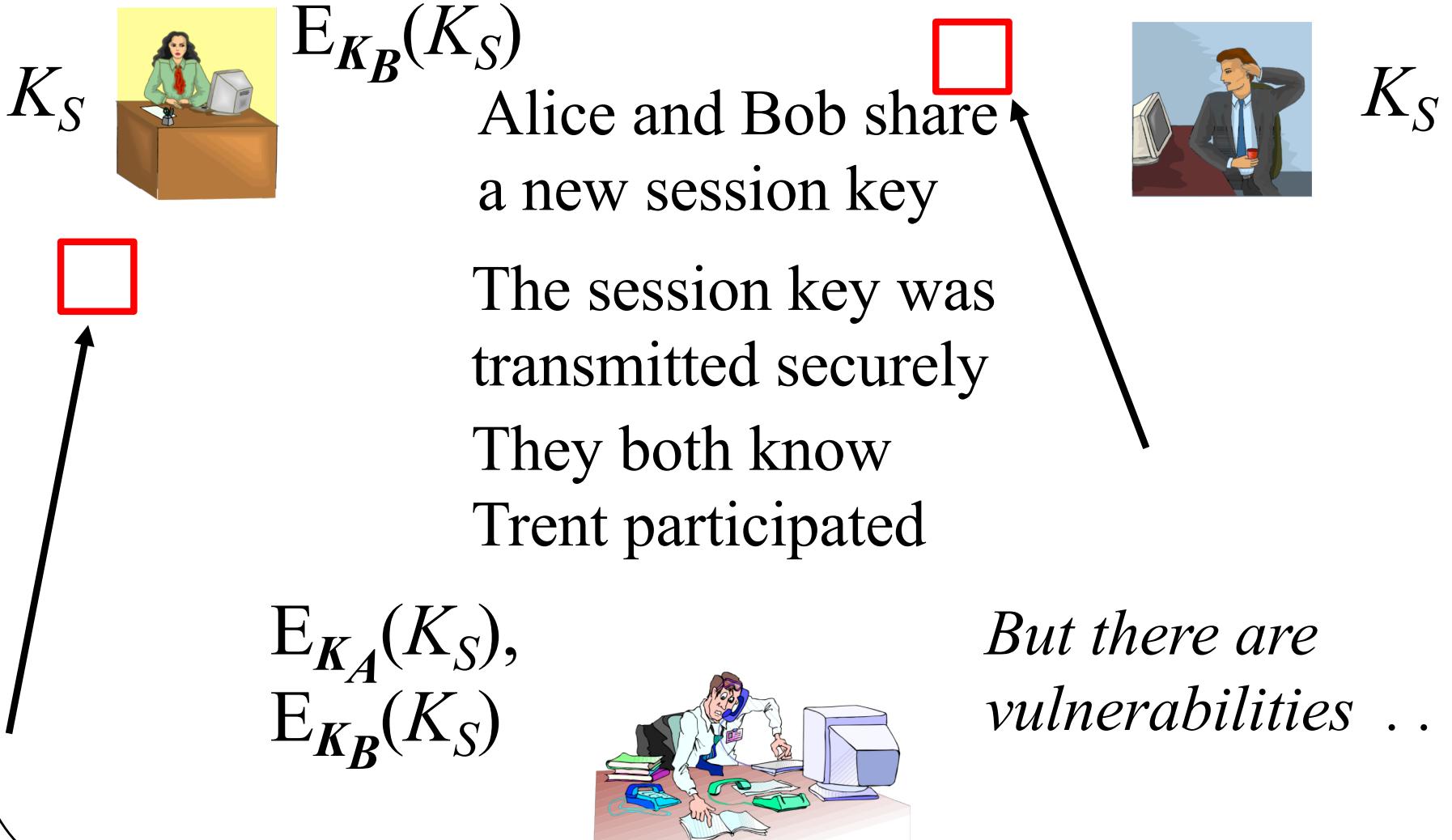
Step Two



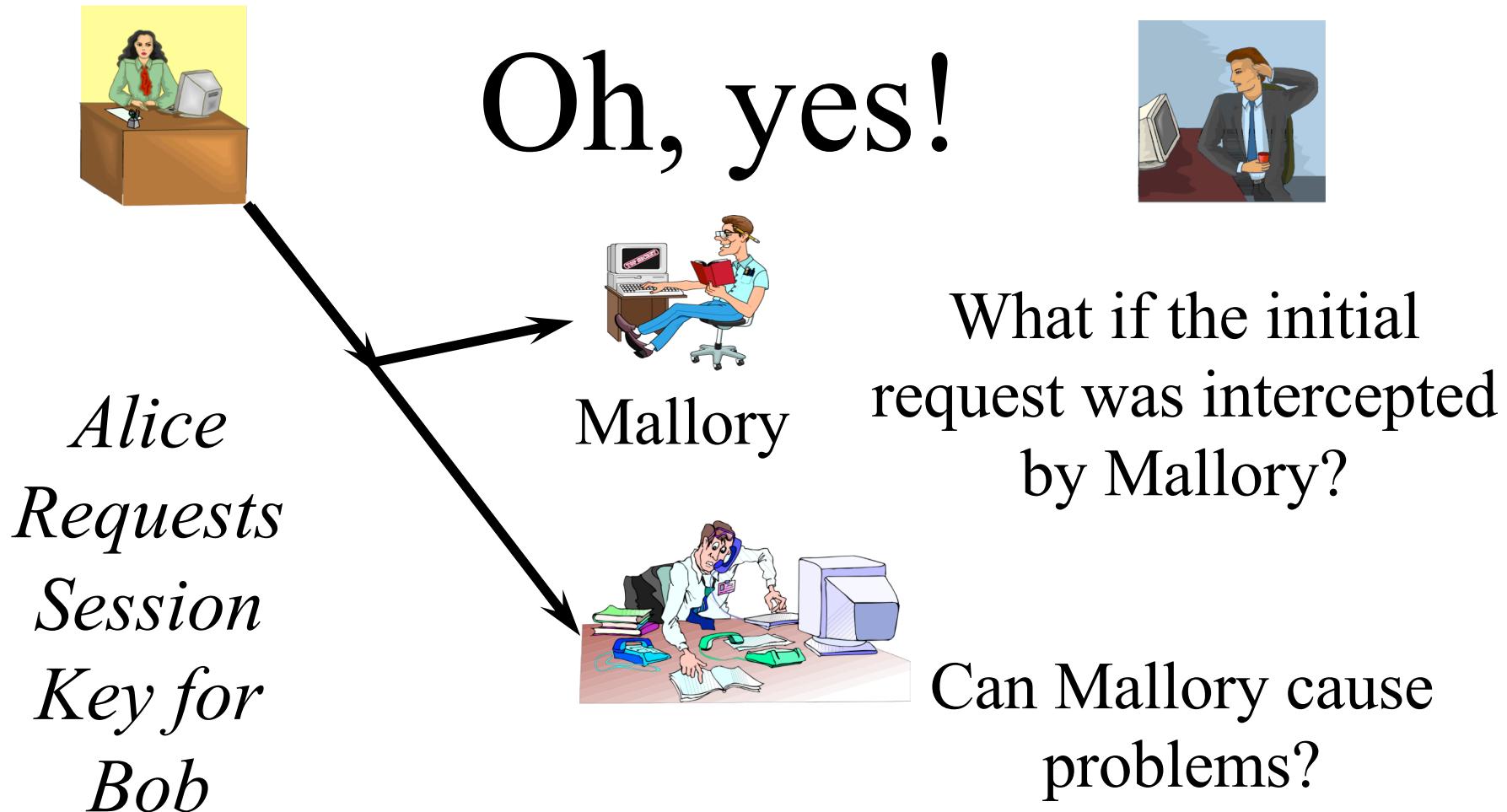
Step Three



What Has the Protocol Achieved?



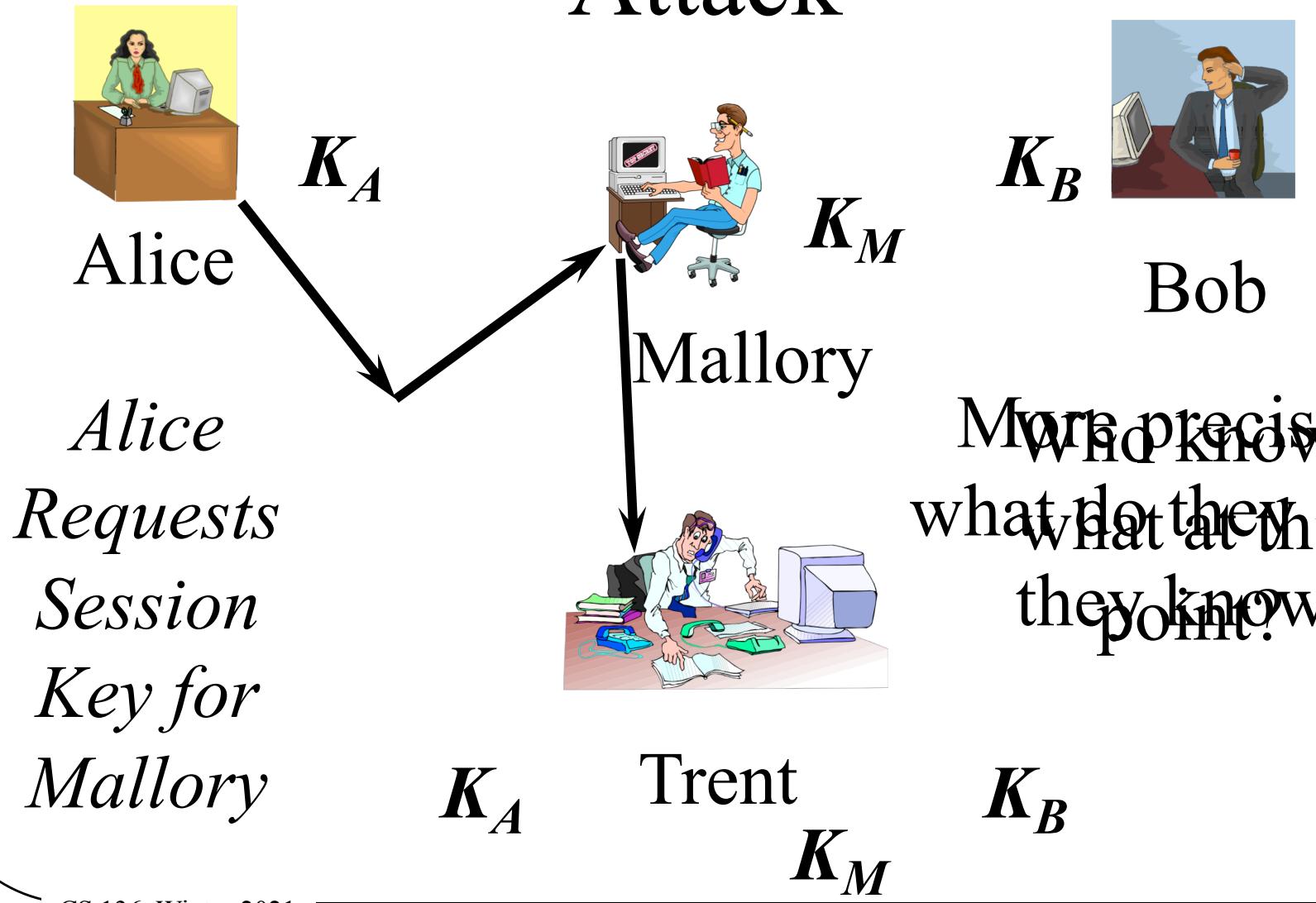
Problems With the Protocol



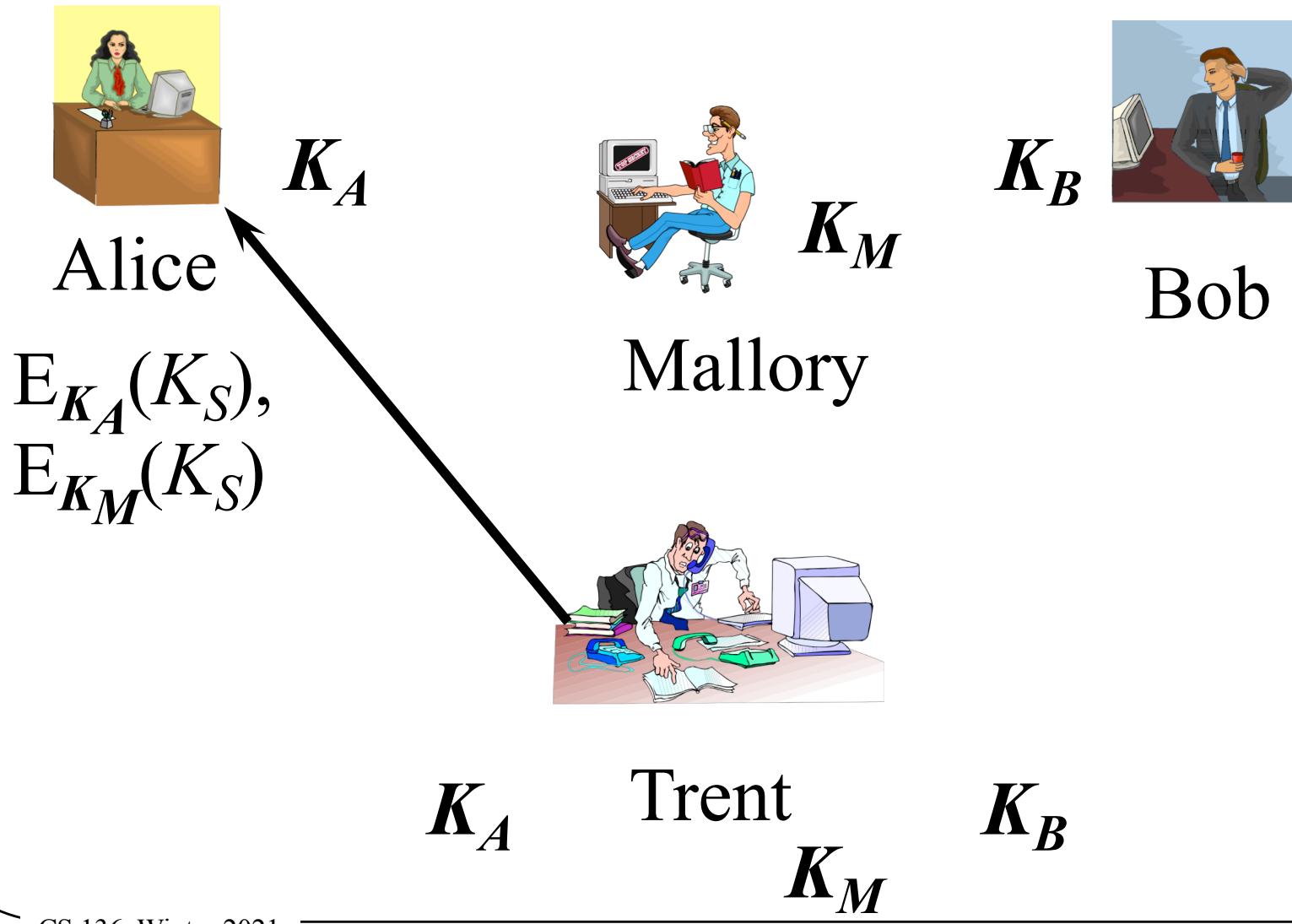
The Man-in-the-Middle Attack

- A class of attacks where an active attacker interposes himself secretly in a protocol
- Allowing alteration of the effects of the protocol
- Without necessarily attacking the encryption

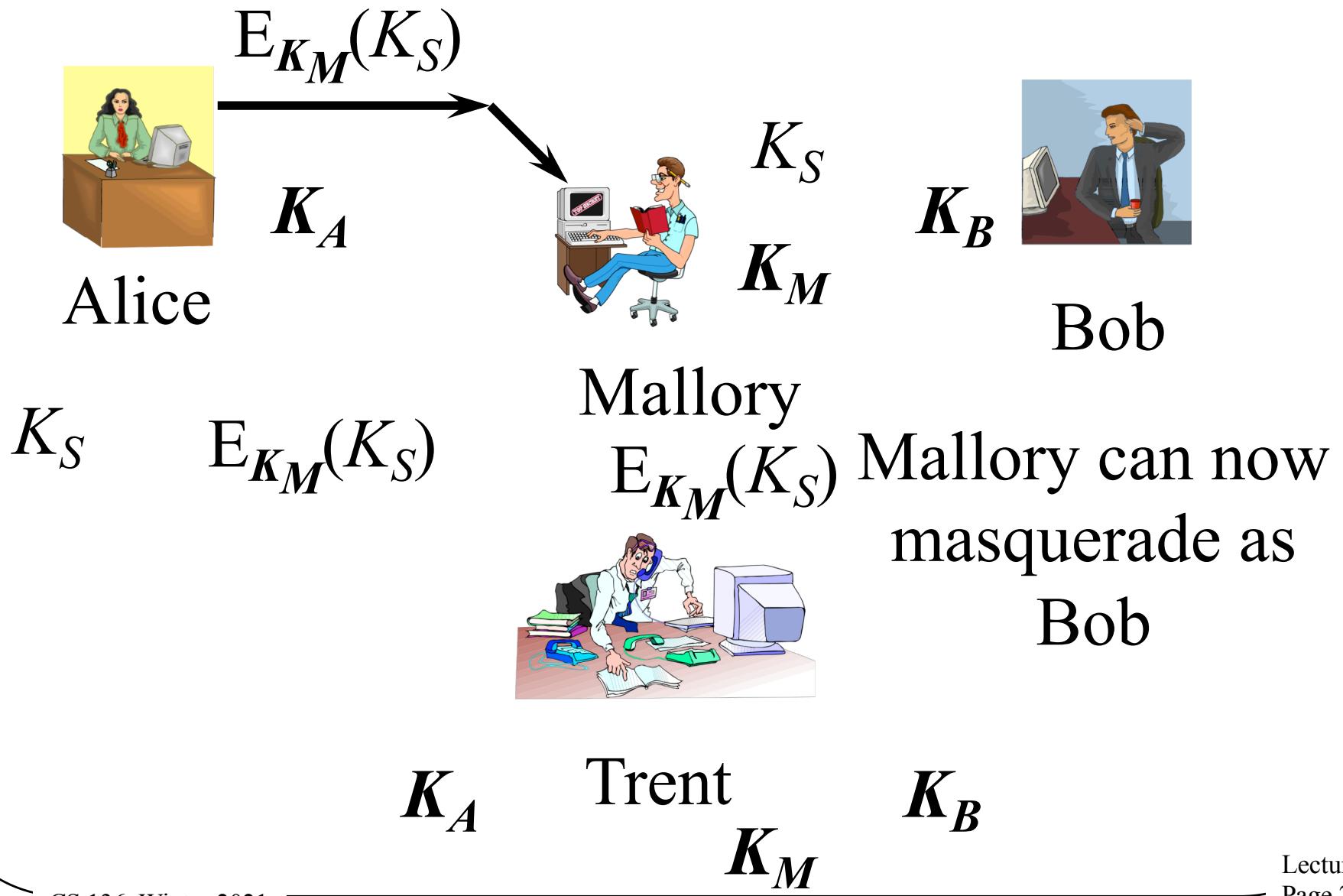
Applying the Man-in-the-Middle Attack



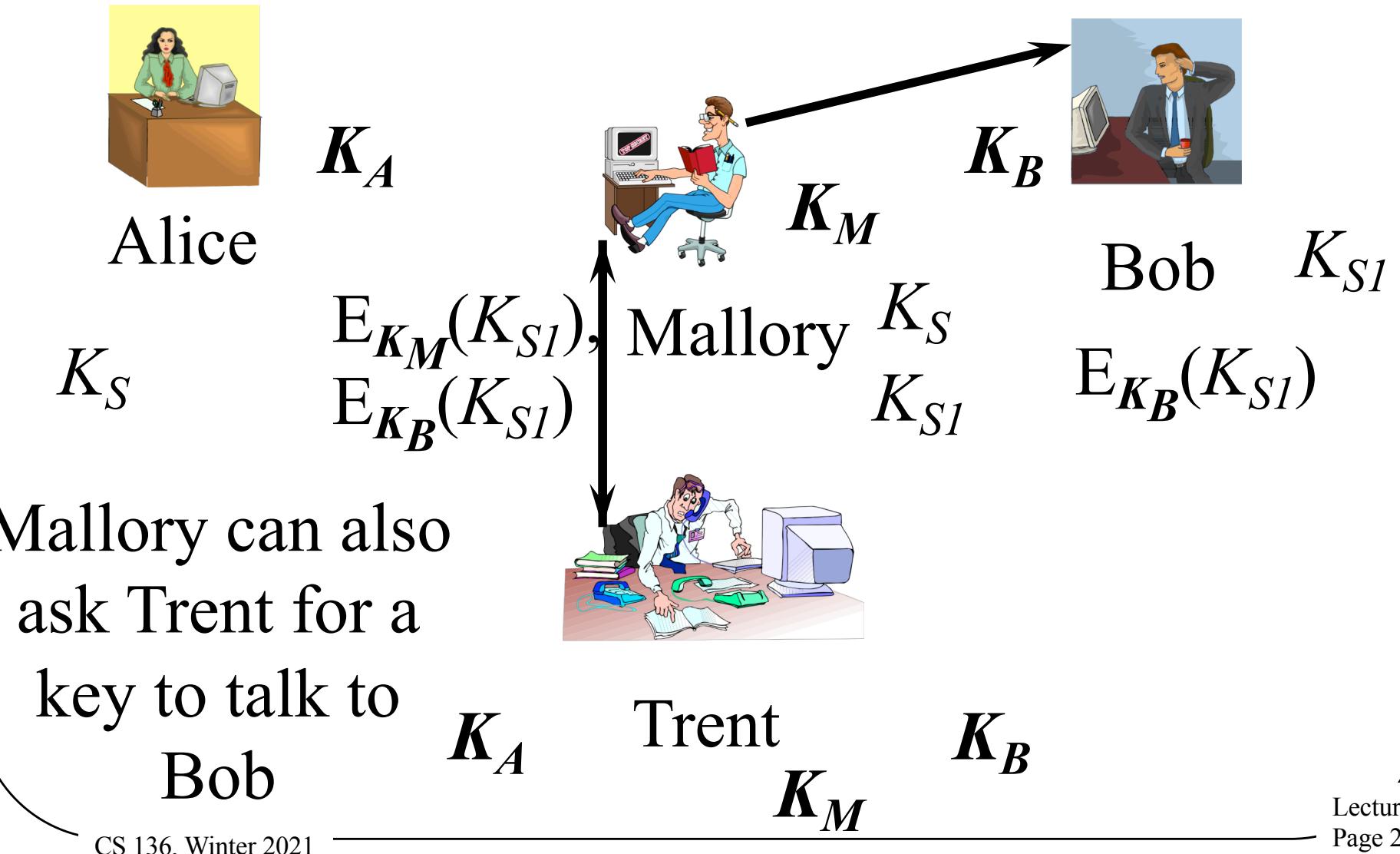
Trent Does His Job



Alice Gets Ready to Talk to Bob



Really Getting in the Middle



Mallory Plays Man-in-the-Middle



Alice

K_S

Alice's big secret

$E_{K_S}(\text{Alice's big secret})$

$E_{K_S}(\text{Bob's big secret})$

Bob's big secret

Mallory K_S

K_{S1}

Bob K_{S1}

Alice's big secret

$E_{K_S}(\text{Alice's big secret})$ **Bob's big secret**

$E_{K_{S1}}(\text{Alice's big secret})$

$E_{K_{S1}}(\text{Bob's big secret})$

$E_{K_S}(\text{Bob's big secret})$ **Bob's big secret**

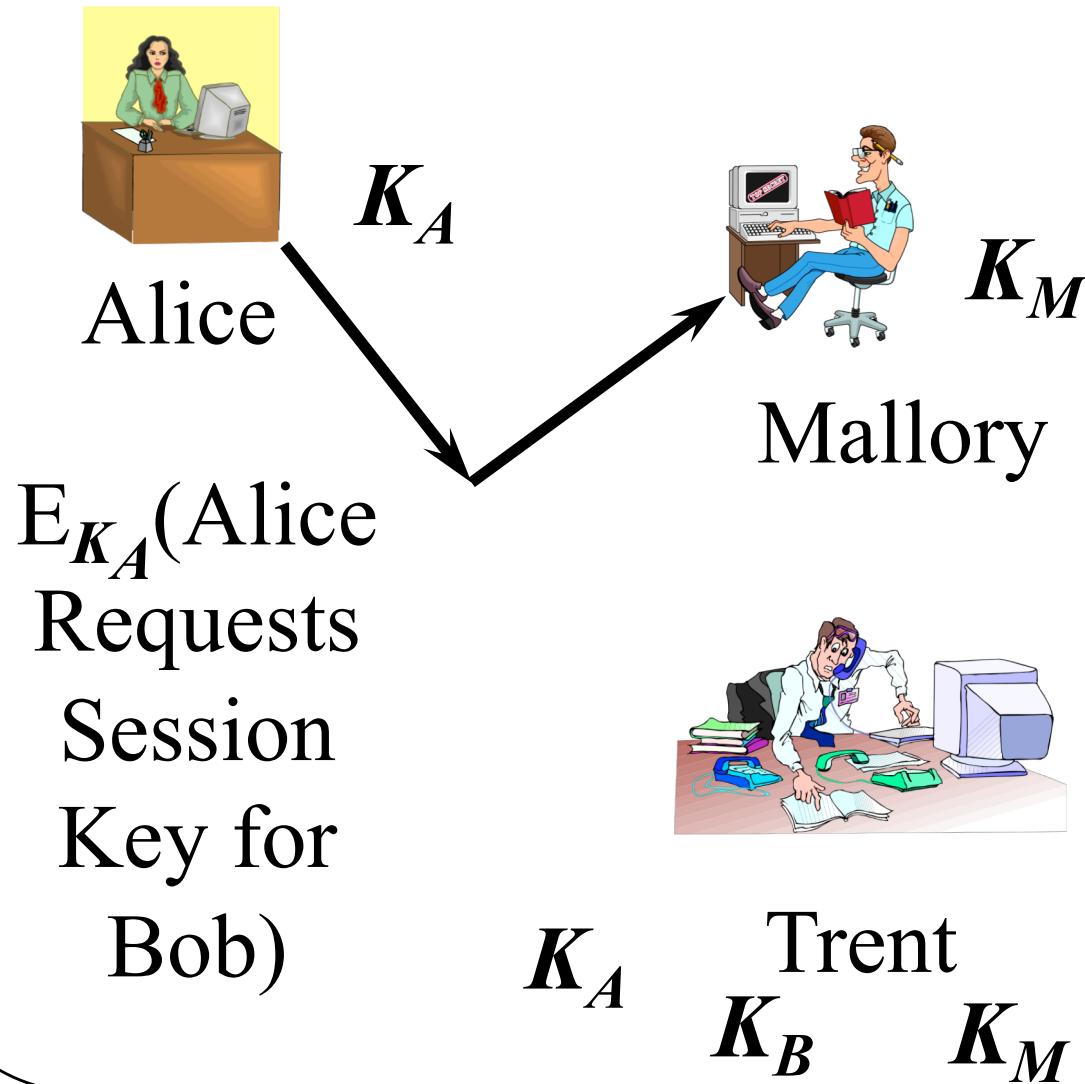
Alice's big secret

Bob's big secret

Defeating the Man In the Middle

- Problems:
 - 1). Trent doesn't really know what he's supposed to do
 - 2). Alice doesn't verify he did the right thing
- Minor changes can fix that
 - 1). Encrypt request with K_A
 - 2). Include identity of other participant in response - $E_{K_A}(K_S, \text{Bob})$

Applying the First Fix



Bob

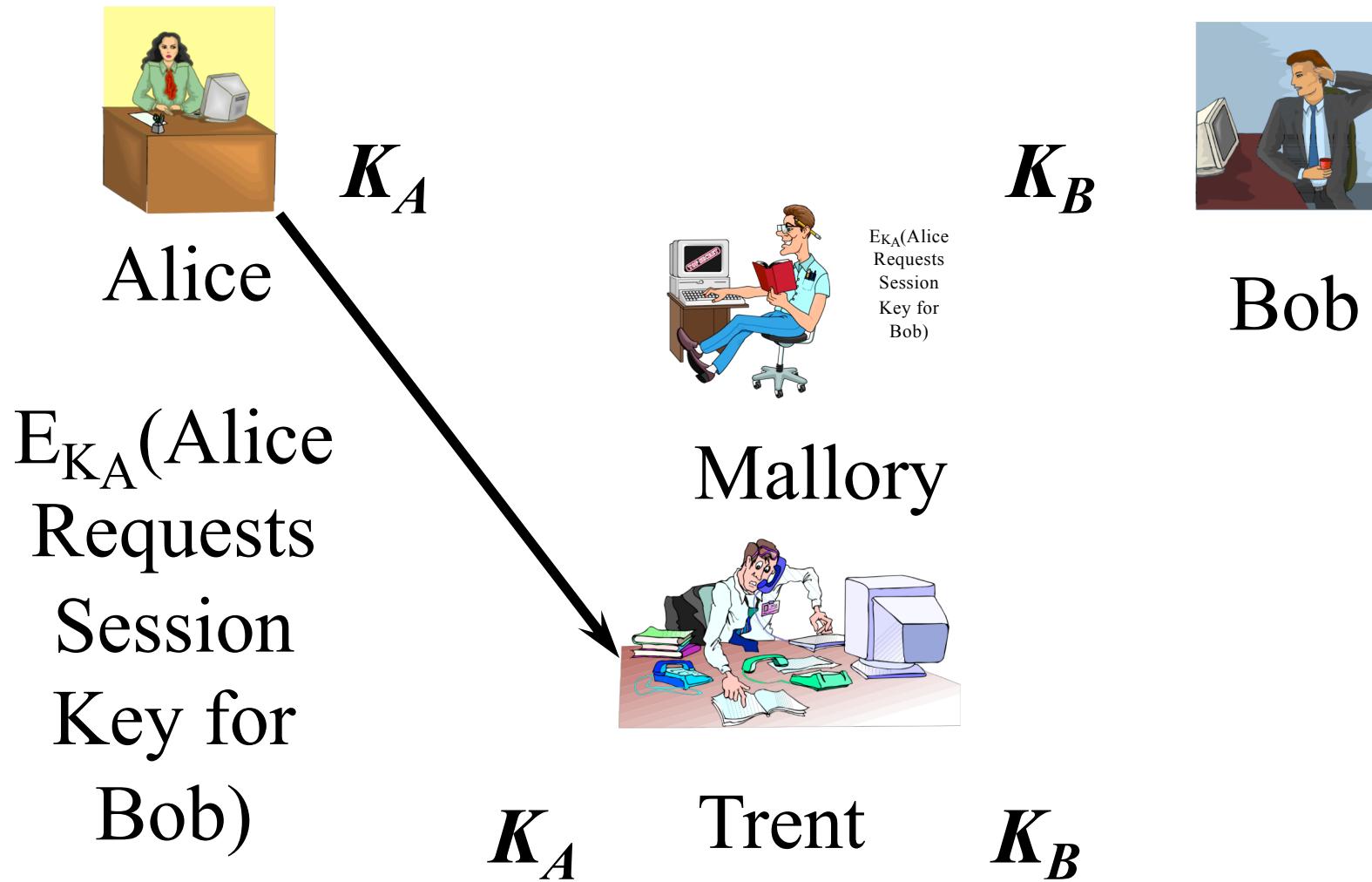
Mallory can't
read the request

And Mallory
can't forge or
alter Alice's
request

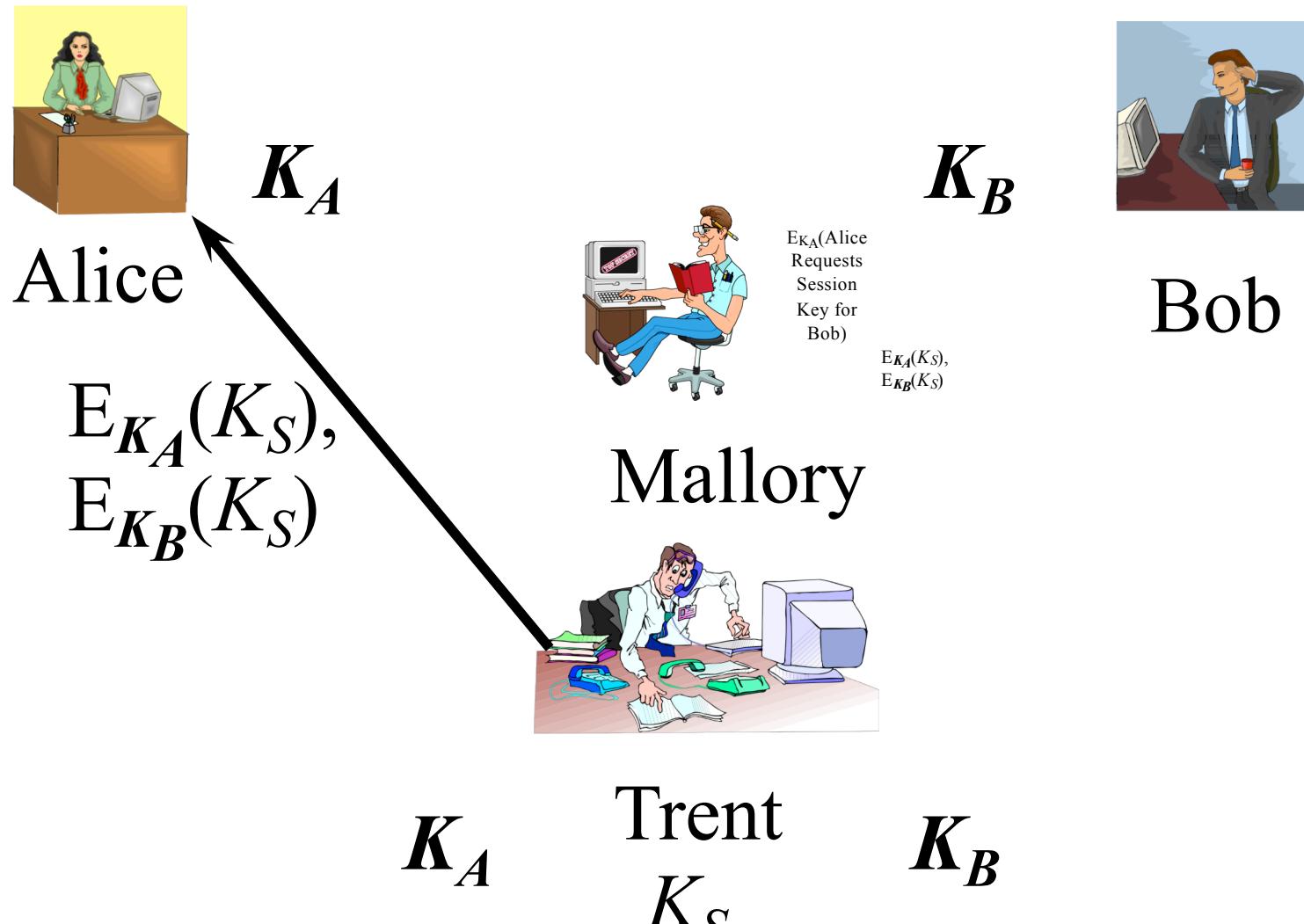
But There's Another Problem

- A replay attack
- Replay attacks occur when Mallory copies down a bunch of protocol messages
- And then plays them again
- In some cases, this can wreak havoc
- Why does it here?

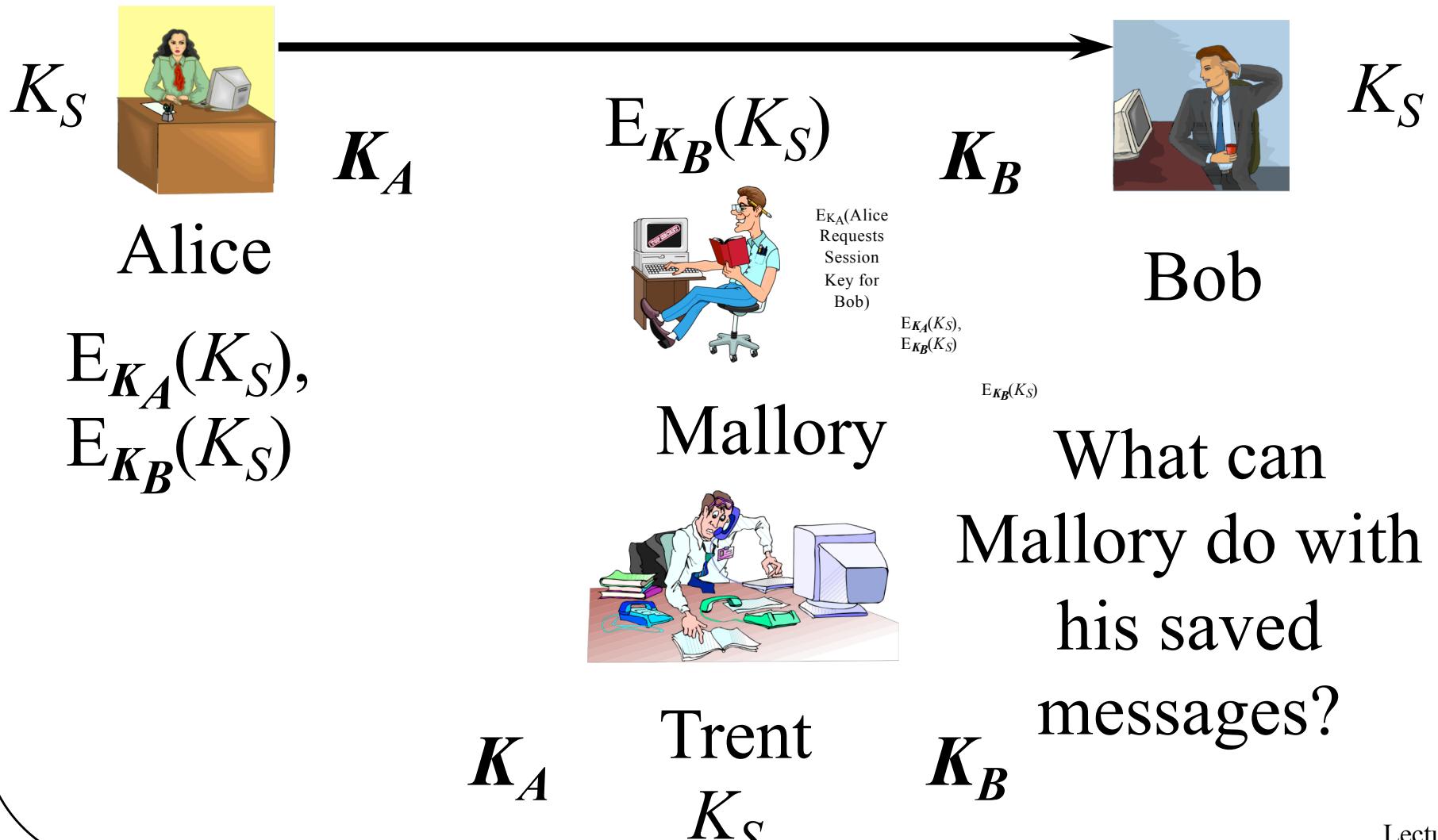
Step One



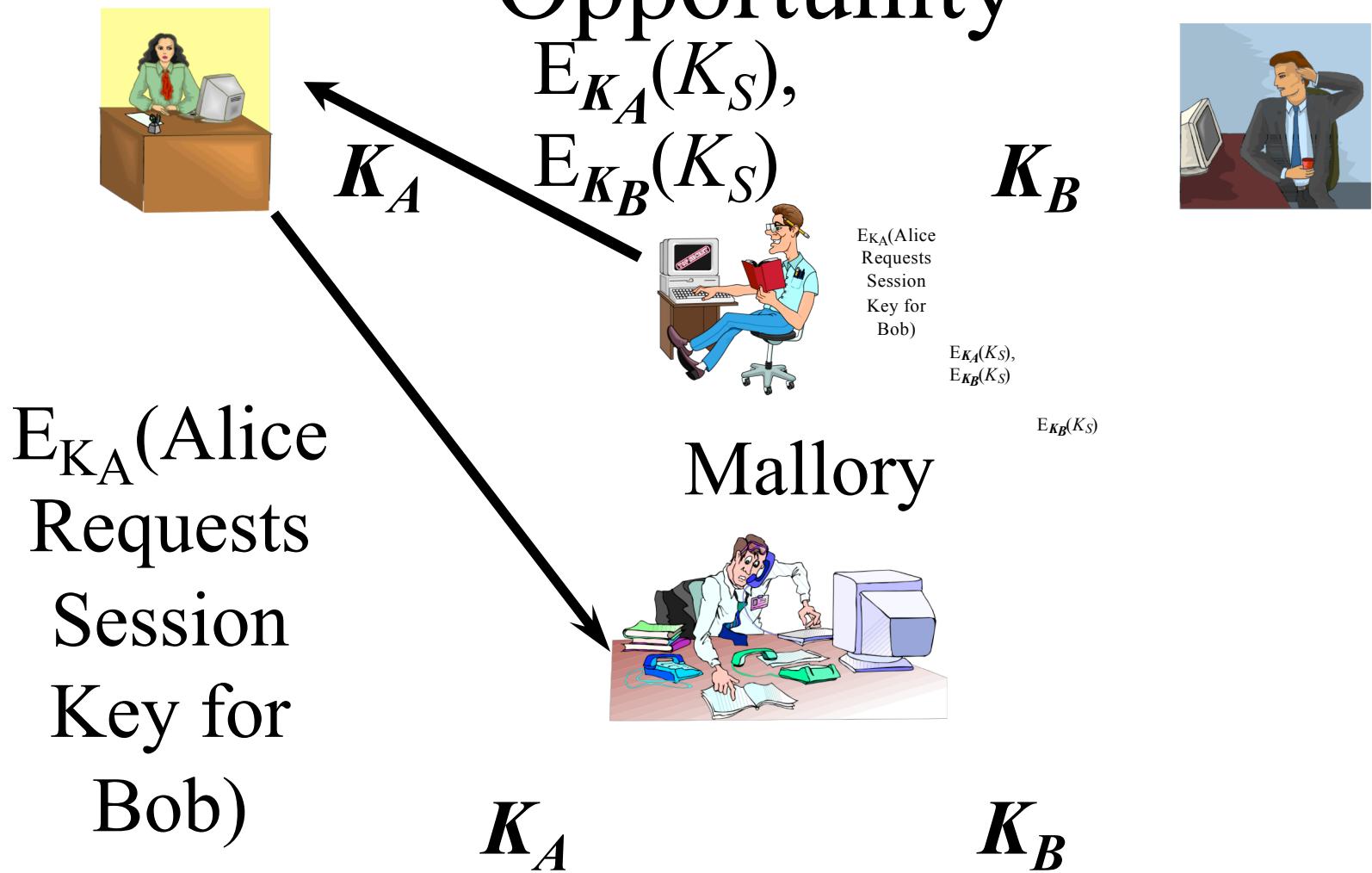
Step Two



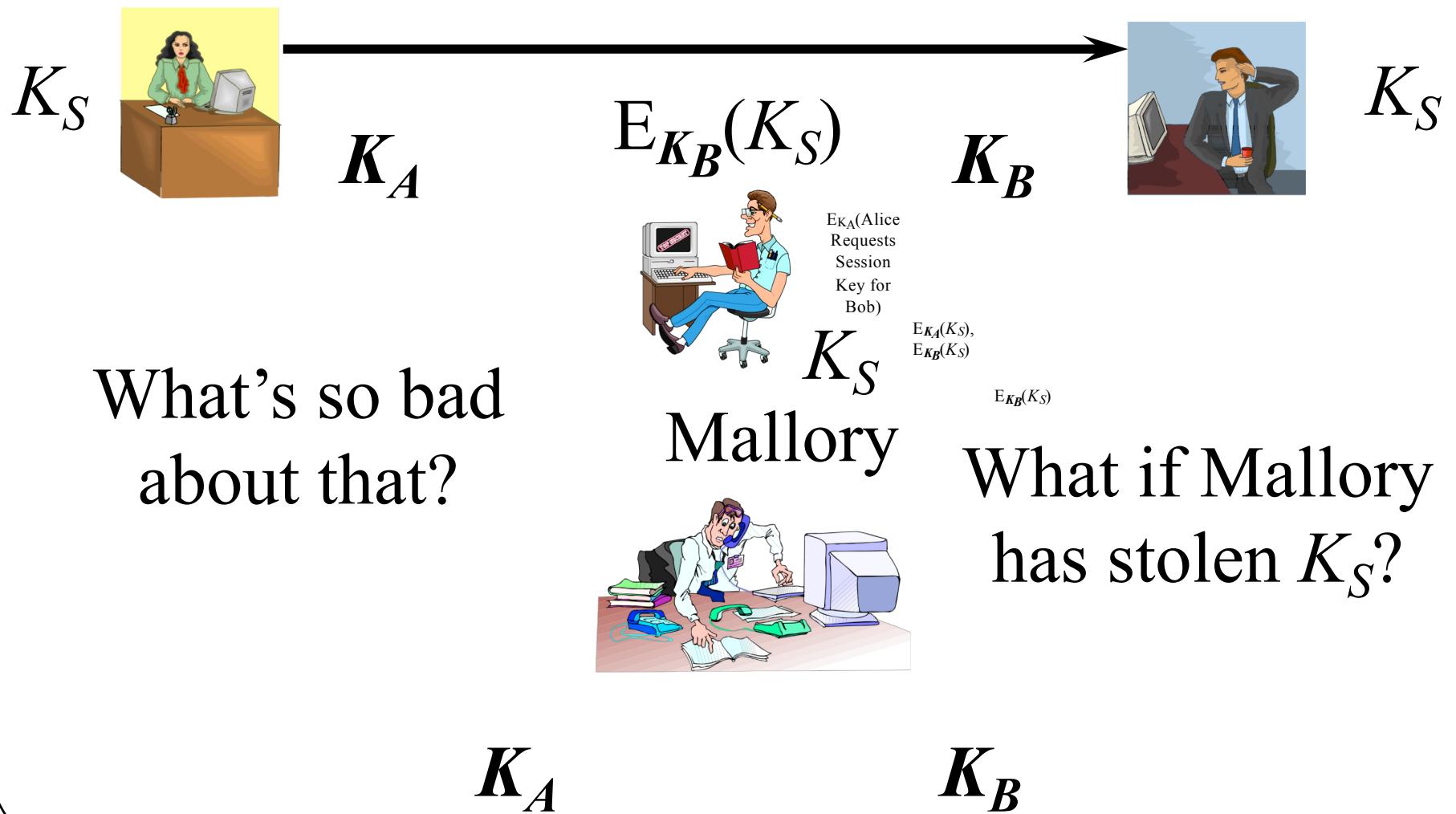
Step Three



Mallory Waits for His Opportunity



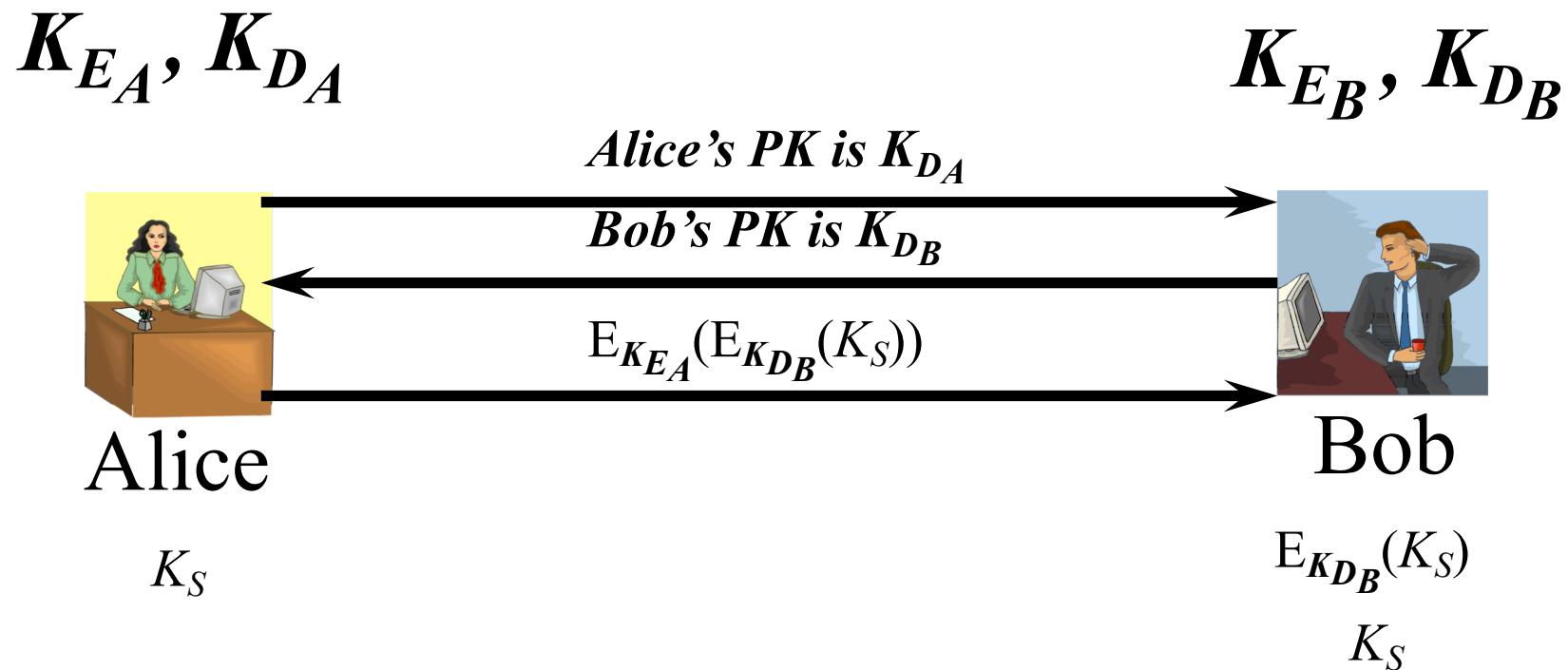
What Will Happen Next?



Key Exchange With Public Key Cryptography

- With no trusted arbitrator
- Alice sends Bob her public key
- Bob sends Alice his public key
- Alice generates a session key and sends it to Bob encrypted with his public key, signed with her private key
- Bob decrypts Alice's message with his private key
- Encrypt session with shared session key

Basic Key Exchange Using PK



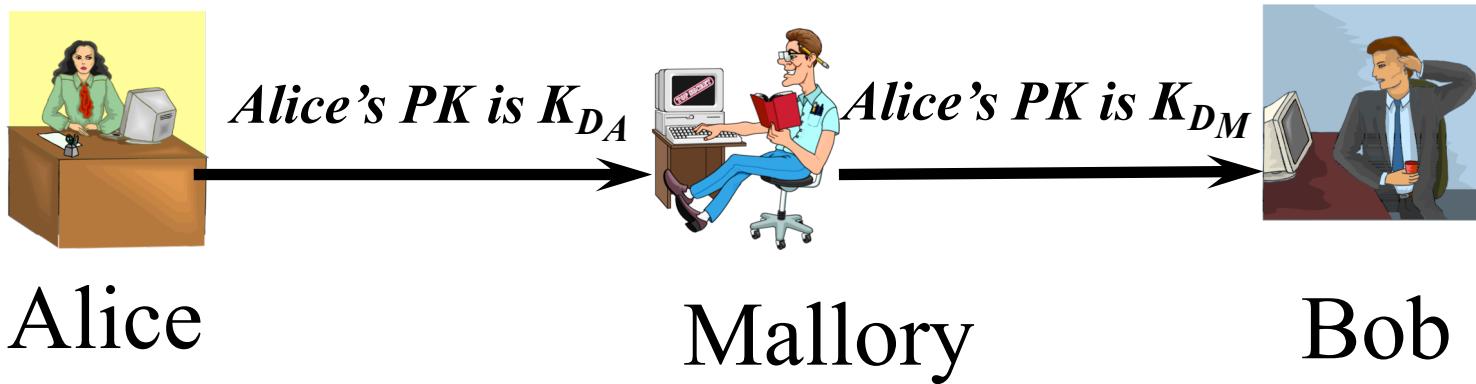
Bob verifies the message came from Alice
Bob extracts the key from the message

Man-in-the-Middle With Public Keys

K_{EA}, K_{DA}

K_{EM}, K_{DM}

K_{EB}, K_{DB}



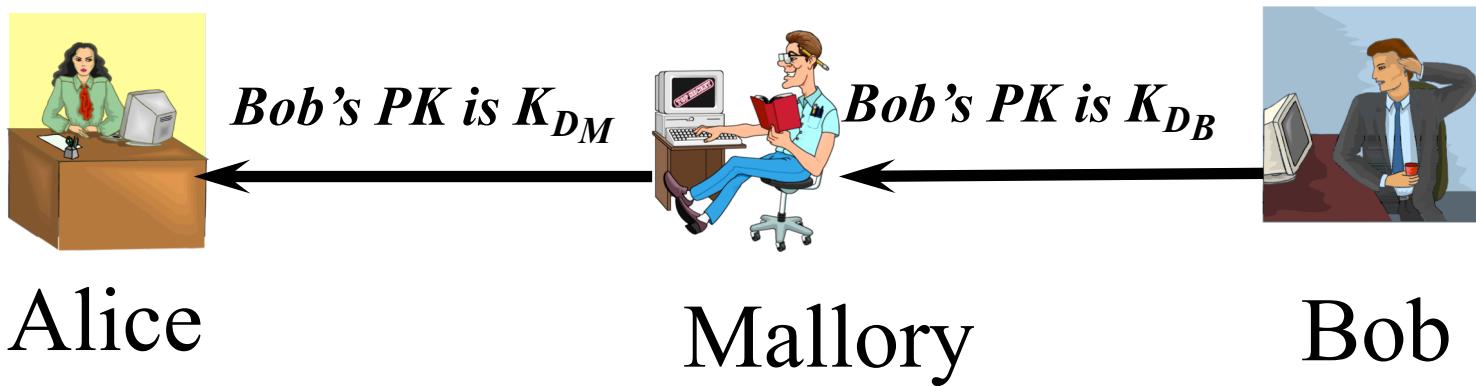
Now Mallory can pose as Alice to Bob

And Bob Sends His Public Key

$$K_{E_A}, K_{D_A}$$

$$K_{EM}, K_{DM}$$

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



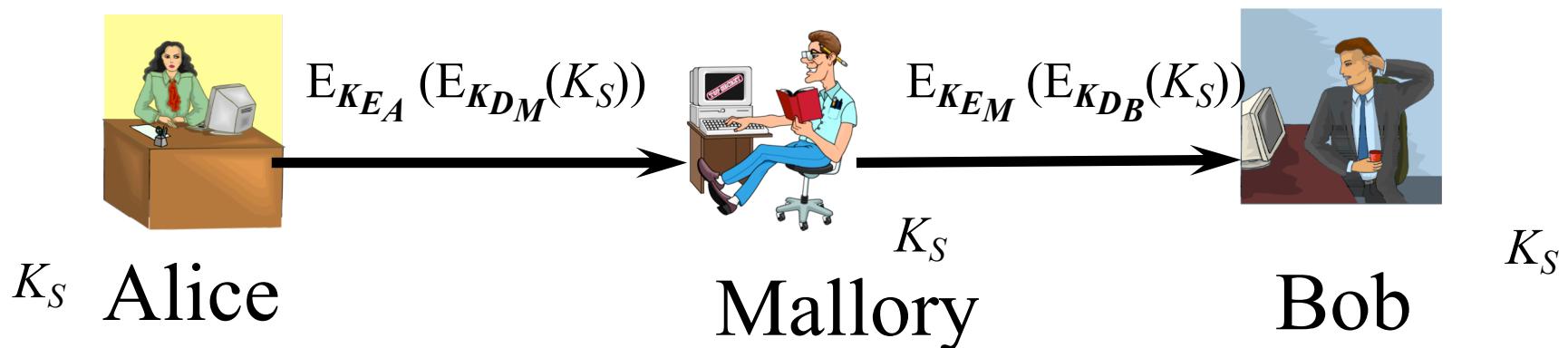
Now Mallory can pose as Bob to Alice

Alice Chooses a Session Key

K_{EA}, K_{DA}

K_{EM}, K_{DM}

K_{EB}, K_{DB}



Bob and Alice are sharing a session key

Unfortunately, they're also sharing it
with Mallory

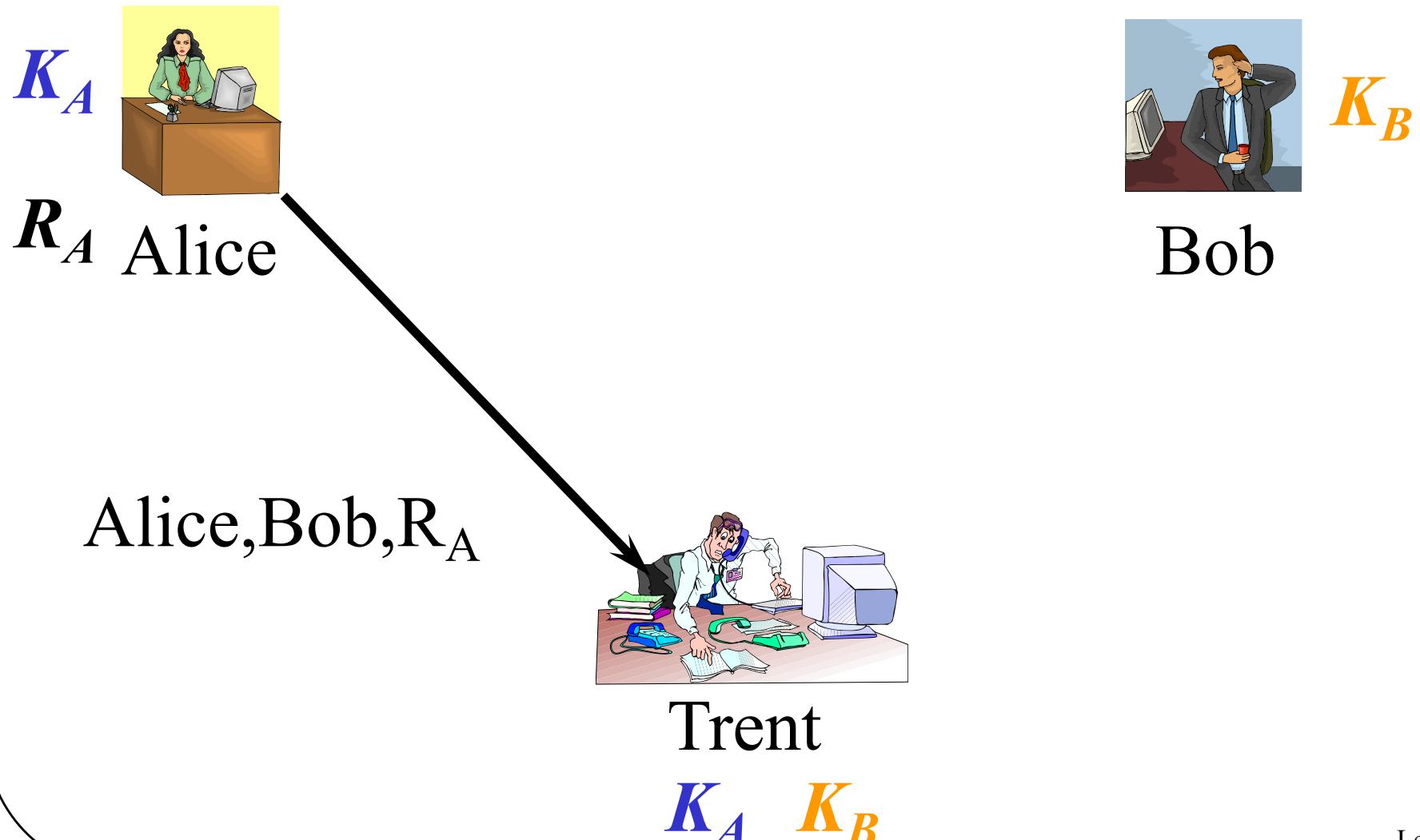
Combined Key Distribution and Authentication

- Usually the first requires the second
 - Not much good to be sure the key is a secret if you don't know who you're sharing it with
- How can we achieve both goals?
 - In a single protocol
 - With relatively few messages

Needham-Schroeder Key Exchange

- Uses symmetric cryptography
- Requires a trusted authority
 - Who takes care of generating the new key
- More complicated than some protocols we've seen

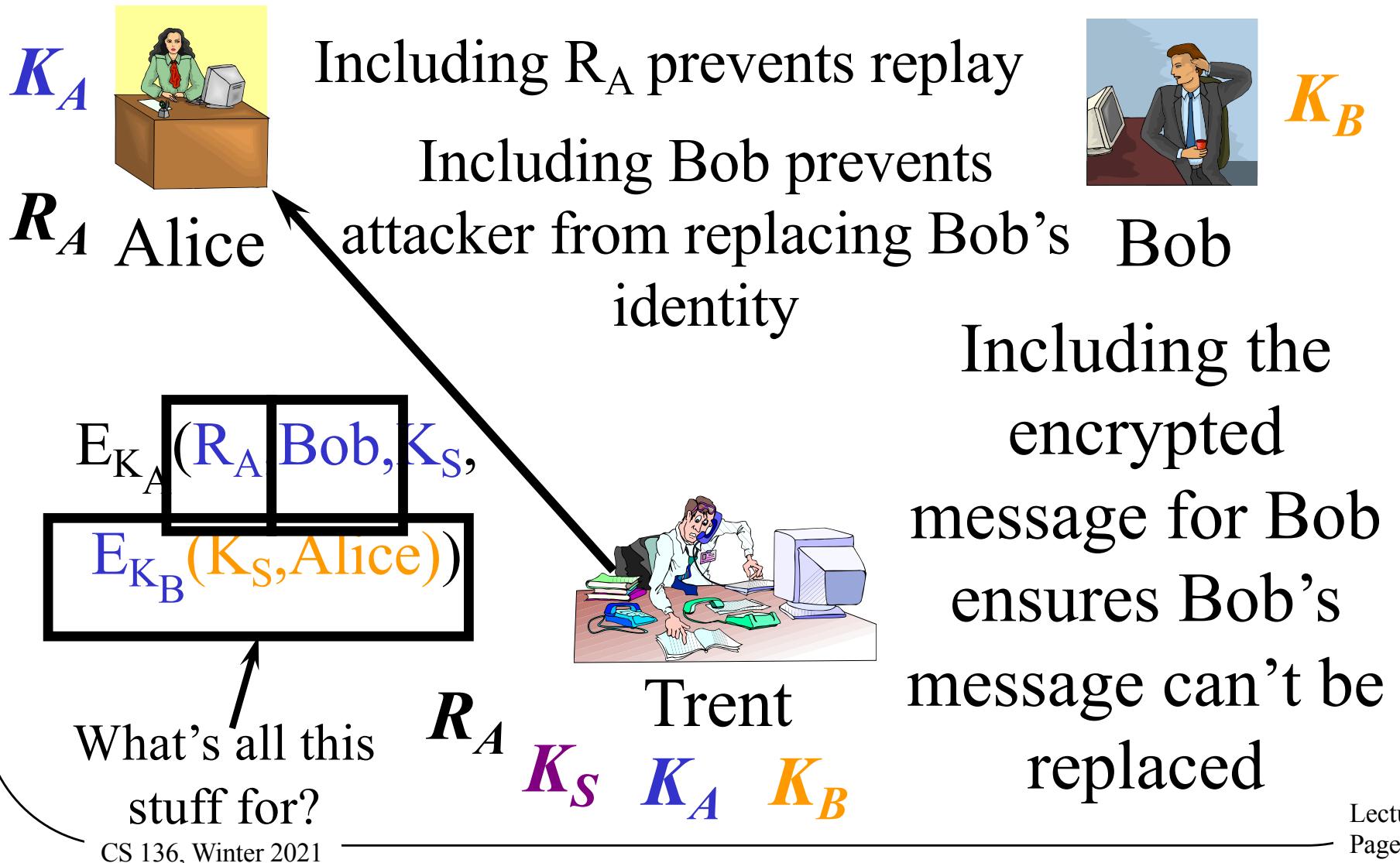
Needham-Schroeder, Step 1



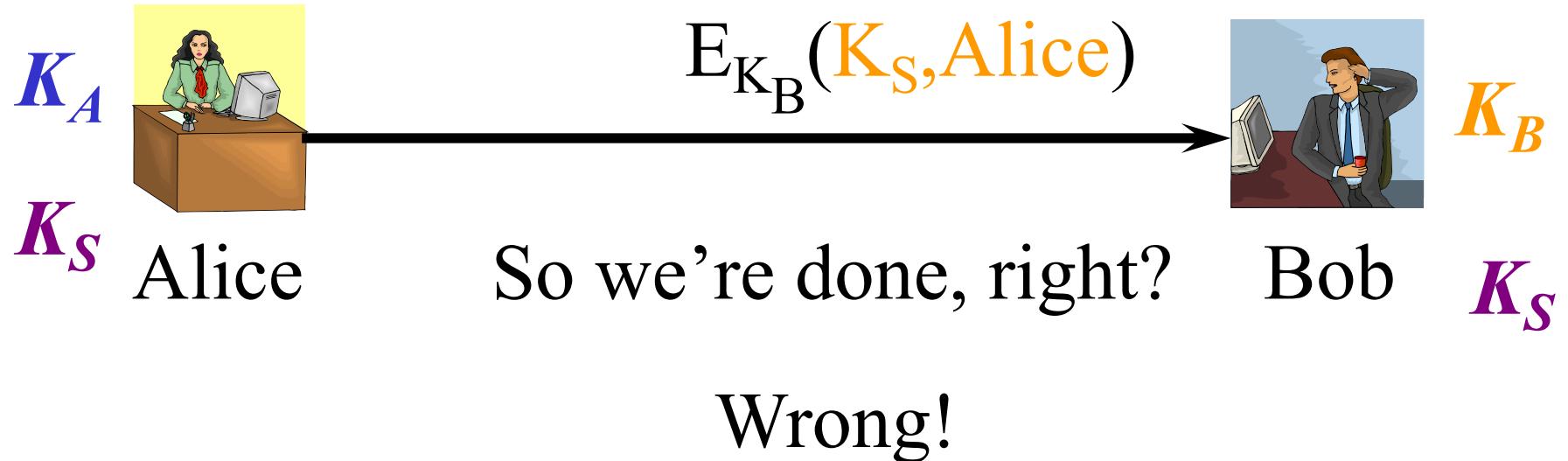
What's the Point of R_A ?

- R_A is random number chosen by Alice for this invocation of the protocol
 - Not used as a key, so quality of Alice's random number generator not too important
- Helps defend against replay attacks
- This kind of random number is sometimes called a *nonce*

Needham-Schroeder, Step 2



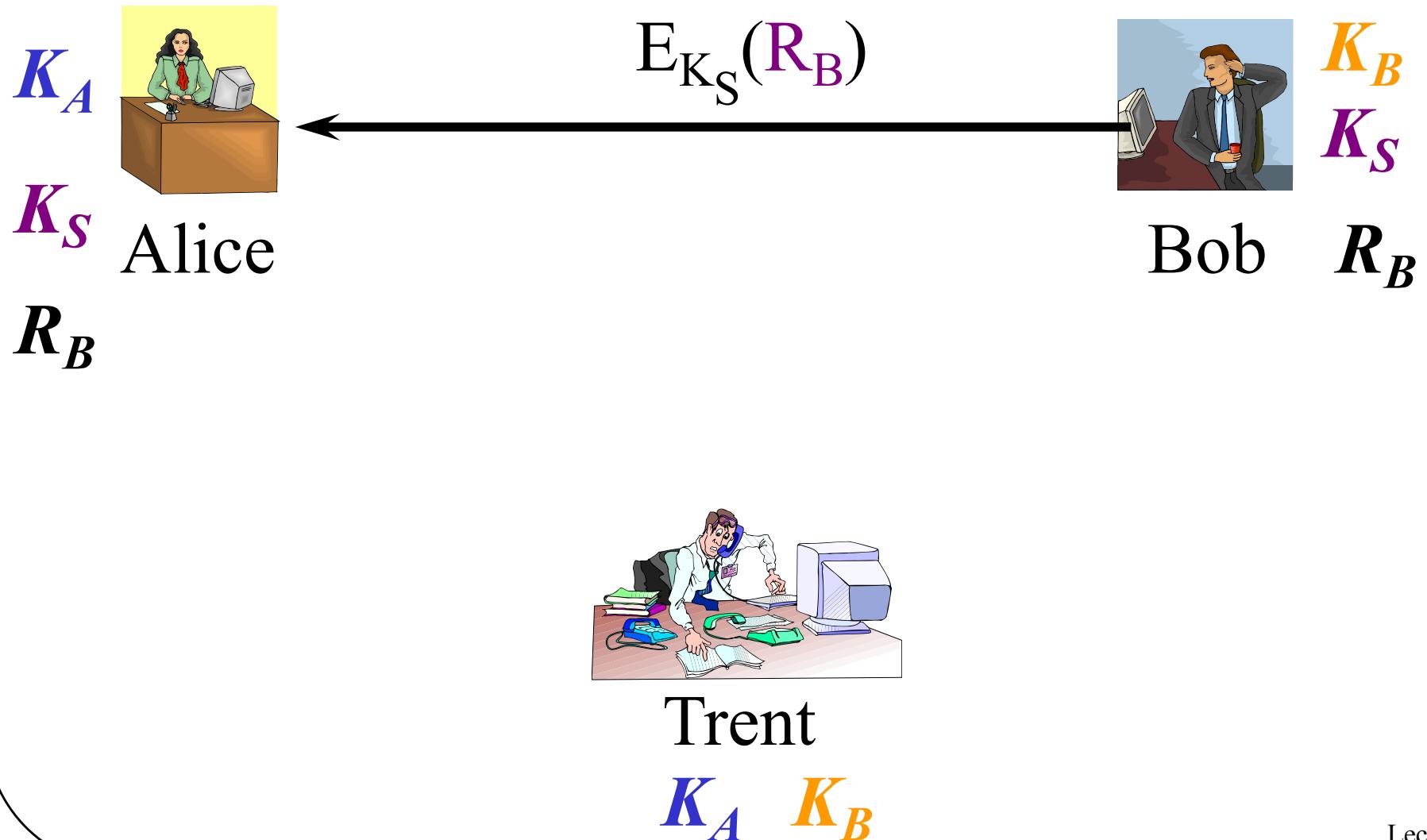
Needham-Schroeder, Step 3



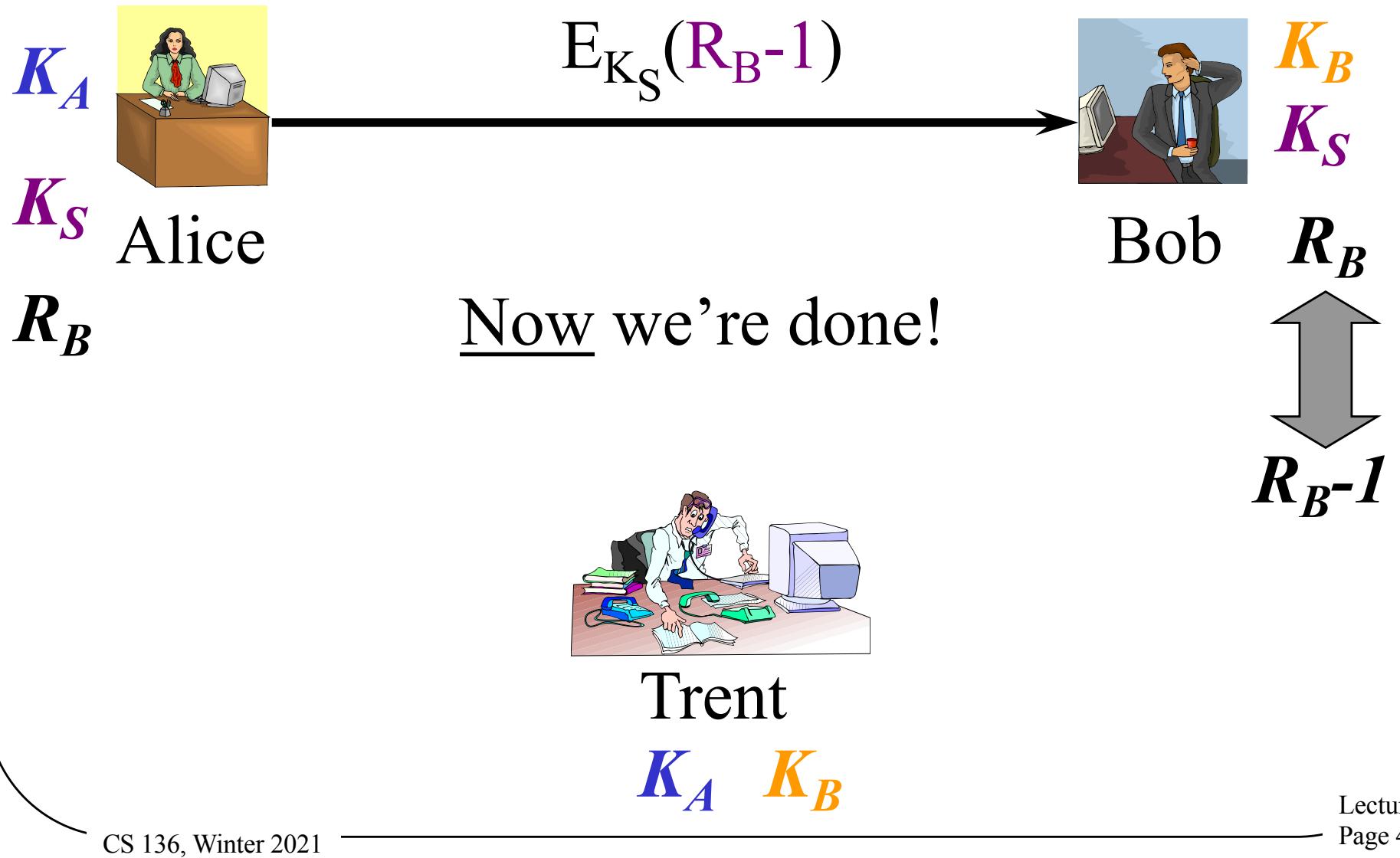
Trent

K_A K_B

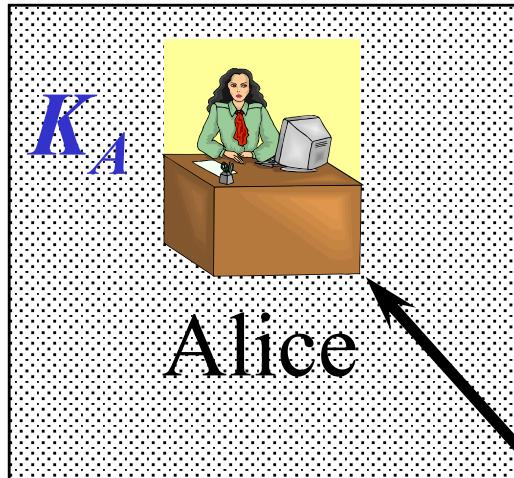
Needham-Schroeder, Step 4



Needham-Schroeder, Step 5



What's All This Extra Stuff For?



Alice knows she's talking to Bob



K_B

Trent said she was

Can Mallory jump in later?



Trent

$E_{K_A}(R_A, Bob, K_S)$

$E_{K_B}(K_S, Alice)$

$K_S \quad K_A \quad K_B$

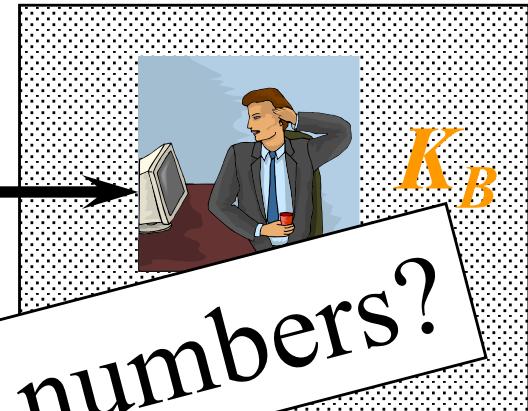
No, only Bob could read the key package Trent created

What's All This Extra Stuff For?



K_A
 K_S Alice

$E_{K_B}(K_S, \text{Alice})$



K_B

Can Mallory jump in?
Trent later
messages will use
 K_S , which Mallory
doesn't know



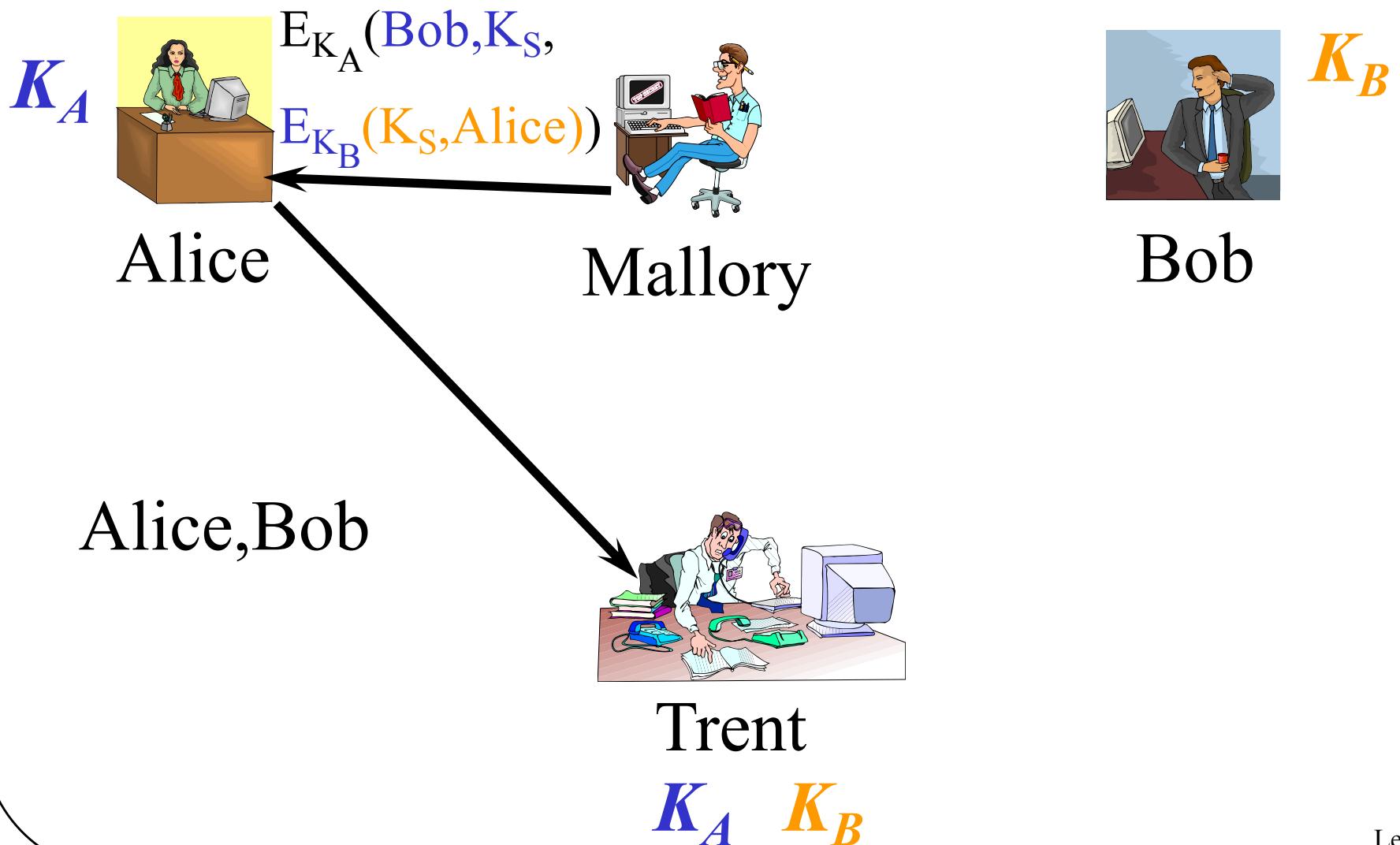
Trent
 K_A K_B

Bob knows
he's talking
to Alice

Mallory Causes Problems

- Alice and Bob do something Mallory likes
- Mallory watches the messages they send to do so
- Mallory wants to make them do it again
- Can Mallory replay the conversation?
 - Let's try it without the random numbers

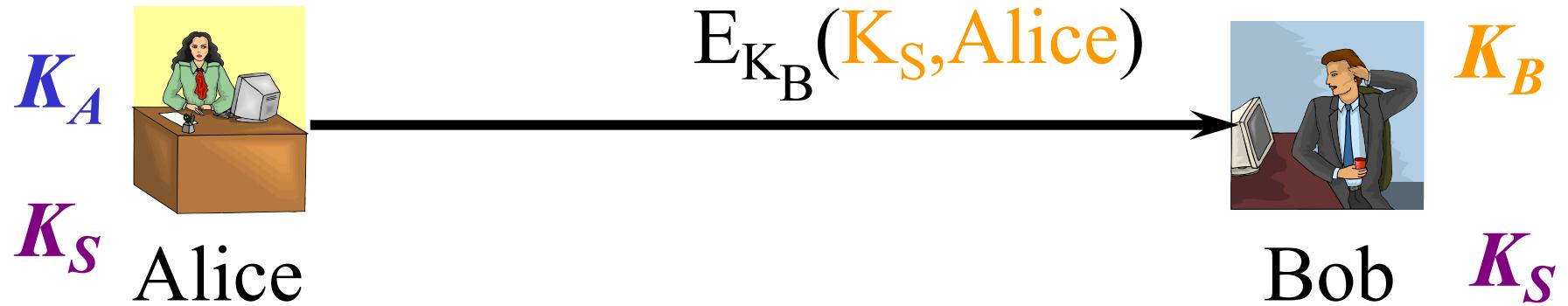
Mallory Waits For His Chance



What Will Alice Do Now?

- The message could only have been created by Trent
- It properly indicates she wants to talk to Bob
- It contains a perfectly plausible key
- Alice will probably go ahead with the protocol

The Protocol Continues



Mallory steps aside for a bit



Trent

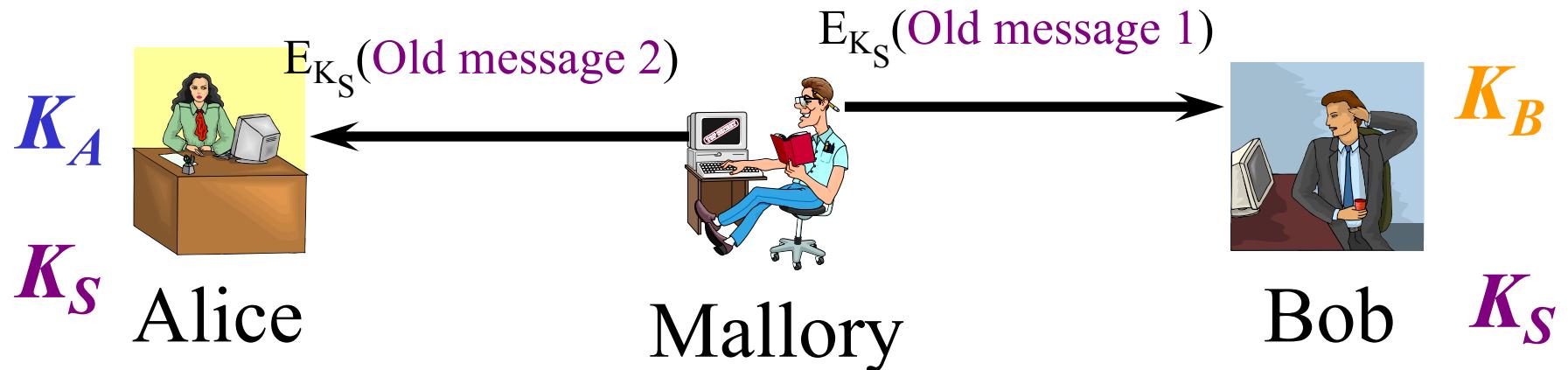
K_A K_B

With no
nonces, we're
done

So What's the Problem?

- Alice and Bob agree K_S is their key
 - They both know the key
 - Trent definitely created the key for them
 - Nobody else has the key
- But . . .

Mallory Steps Back Into the Picture



Mallory can
replay Alice and
Bob's old
conversation



Trent

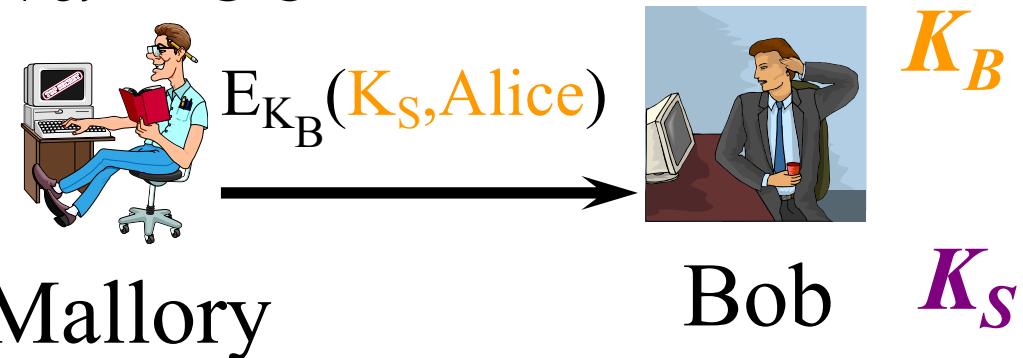
K_A K_B

It's using the
current key, so
Alice and Bob
will accept it

How Do the Random Numbers Help?

- Alice's random number assures her that the reply from Trent is fresh
- But why does Bob need another random number?

Why Bob Also Needs a Random Number



Let's say Alice
doesn't want to
talk to Bob



But Mallory
wants Bob to
think Alice wants
to talk

Trent
 K_A K_B

So What?



Mallory can now play back an old message from Alice to Bob

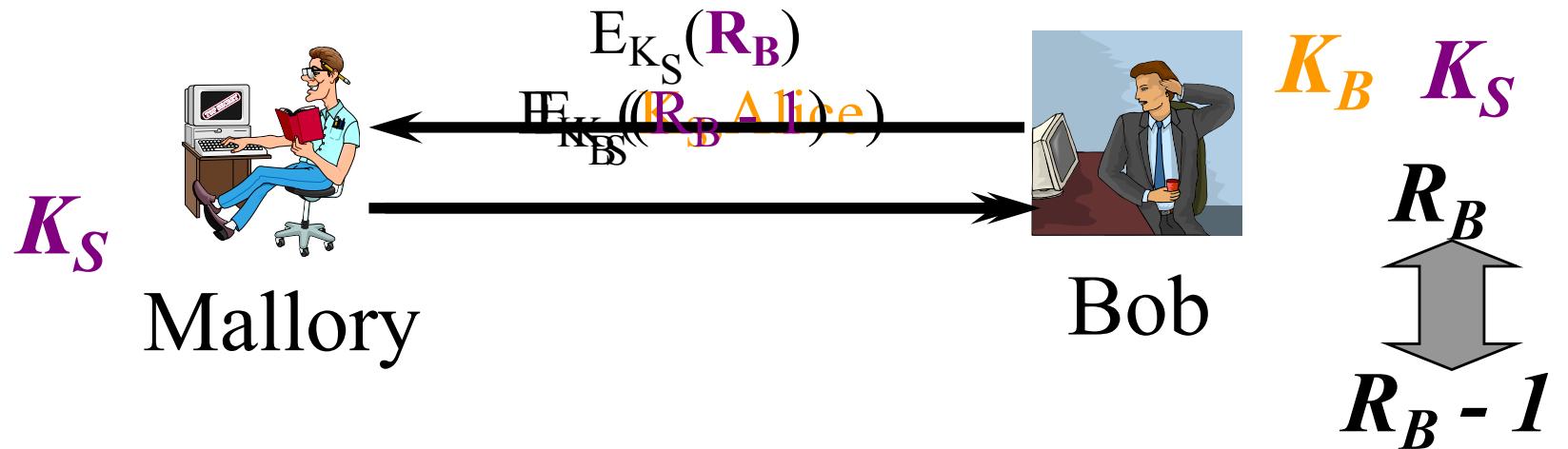
And Bob will have no reason to be suspicious

Bob's random number exchange assures him that Alice really wanted to talk

So, Everything's Fine, Right?

- Not if any key K_S ever gets divulged
- Once K_S is divulged, Mallory can forge Alice's response to Bob's challenge
- And convince Bob that he's talking to Alice when he's really talking to Mallory

Mallory Cracks an Old Key



Mallory compromises 10,000 computers belonging to 10,000 grandmothers to crack K_S

Unfortunately, Mallory knows K_S

So Mallory can answer Bob's challenge

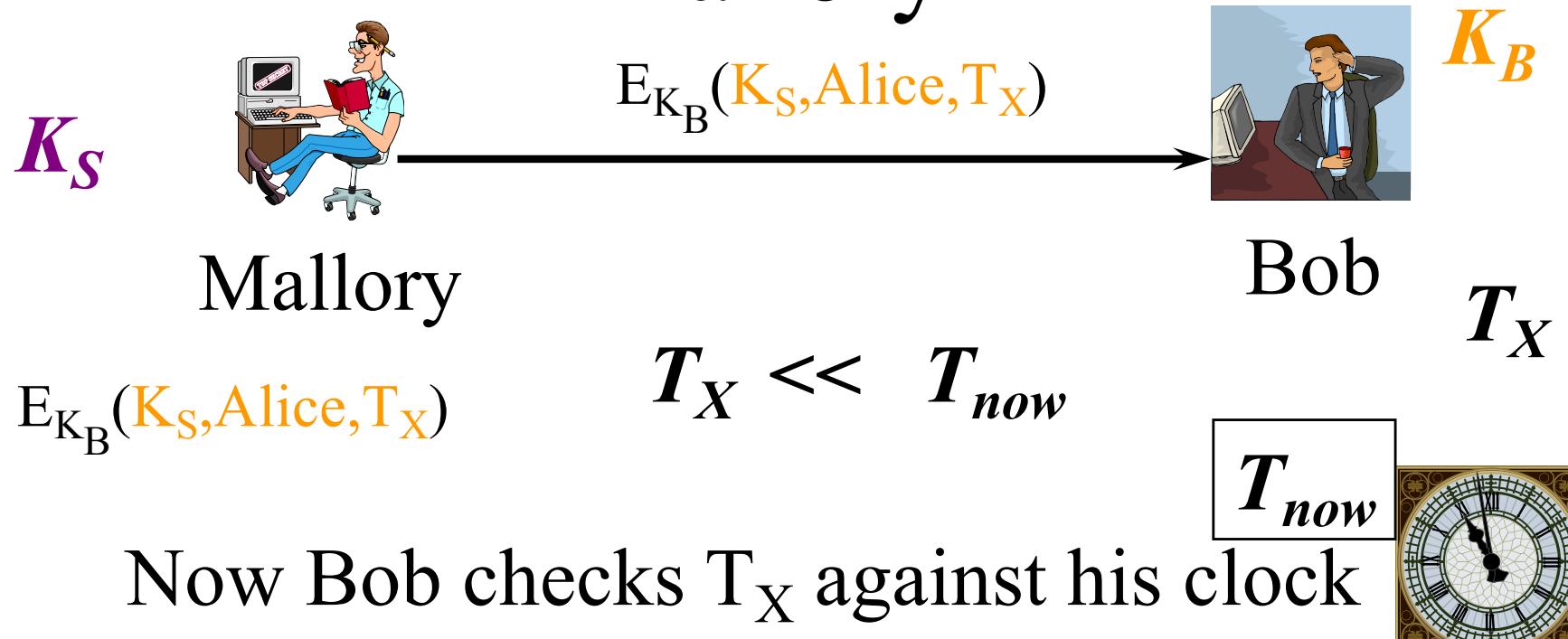
Timestamps in Security Protocols

- One method of handling this kind of problem is timestamps
- Proper use of timestamps can limit the time during which an exposed key is dangerous
- But timestamps have their own problems

Using Timestamps in the Needham-Schroeder Protocol

- The trusted authority includes timestamps in his encrypted messages to Alice and Bob
- Based on a global clock
- When Alice or Bob decrypts, if the timestamp is too old, abort the protocol

Using Timestamps to Defeat Mallory



So Bob, fearing replay, discards K_S

And Mallory's attack is foiled

Problems With Using Timestamps

- They require a globally synchronized set of clocks
 - Hard to obtain, often
 - Attacks on clocks become important
- They leave a window of vulnerability

The Suppress-Replay Attack

- Assume two participants in a security protocol
 - Using timestamps to avoid replay problems
- If the sender's clock is ahead of the receiver's, attacker can intercept message
 - And replay later, when receiver's clock still allows it

Handling Clock Problems

- 1). Rely on clocks that are fairly synchronized and hard to tamper with
 - Perhaps GPS signals
- 2). Make all comparisons against the same clock
 - So no two clocks need to be synchronized

Is This Overkill?

- Some of these attacks are pretty specialized
 - Requiring special access or information
- Some can only achieve certain limited effects
- Do we really care?

Why Should We Care?

- Bad guys are very clever
- Apparently irrelevant vulnerabilities give them room to show that
- Changes in how you use protocols can make vulnerabilities more relevant
- A protocol without a vulnerability is always better
 - Even if you currently don't care

Something to Bear in Mind

- These vulnerabilities aren’t specific to just these protocols
- They are common and pop up all over
 - Even in cases where you aren’t thinking about a “protocol”
- Important to understand them at a high conceptual level

Authentication

Computer Security

Peter Reiher

January 26, 2021

Outline

- Introduction
- Basic authentication mechanisms

Introduction

- Much of security is based on good access control
- Access control only works if you have good authentication
- What is authentication?

Authentication

- Determining the identity of some entity
 - Process
 - Machine
 - Human user
- Requires notion of identity
- And some degree of proof of identity

Authentication Vs. Authorization

- *Authentication* is determining who you are
- *Authorization* is determining what someone is allowed to do
- Can't authorize properly without authentication
- Purpose of authentication is usually to make authorization decisions

Proving Identity in the Physical World

- Most frequently done by physical recognition
 - I recognize your face, your voice, your body
- What about identifying those we don't already know?

Other Physical Identification Methods

- Identification by credentials
 - You show me your driver's license
- Identification by recommendation
 - You introduce me to someone
- 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

Differences in Cyber Identification

- Usually the identifying entity isn't human
- Often the identified entity isn't human, either
- Often no physical presence required
- Often no later rechecks of identity

Identifying 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

Identifying Computers and Programs

- No physical characteristics
 - Faces, fingerprints, voices, etc.
- Generally easy to duplicate programs
- Not smart enough to be flexible
 - Must use methods they will understand
- Again, good at computations

Physical Presence Optional

- Often authentication required over a network or cable
- Even if the party to be identified is human
- So authentication mechanism must work in face of network characteristics
 - Active wiretapping
 - Everything is converted to digital signal

Identity Might Not Be Rechecked

- Human beings can make identification mistakes
- But they often recover from them
 - Often quite easily
- Based on observing behavior that suggests identification was wrong
- Computers and programs rarely have that capability
 - If they identify something, they believe it

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 networks connect remote devices to computers, susceptible to password sniffers
- Unless quite long, brute force attacks often work on them

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

Passwords and Single Sign-On

- Many systems ask for password once
 - Resulting authentication lasts for an entire “session”
- Used on its own, complete mediation definitely not achieved
- Trading security for convenience
- Especially if others can use the authenticated machine

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*
- And compare it to the stored version

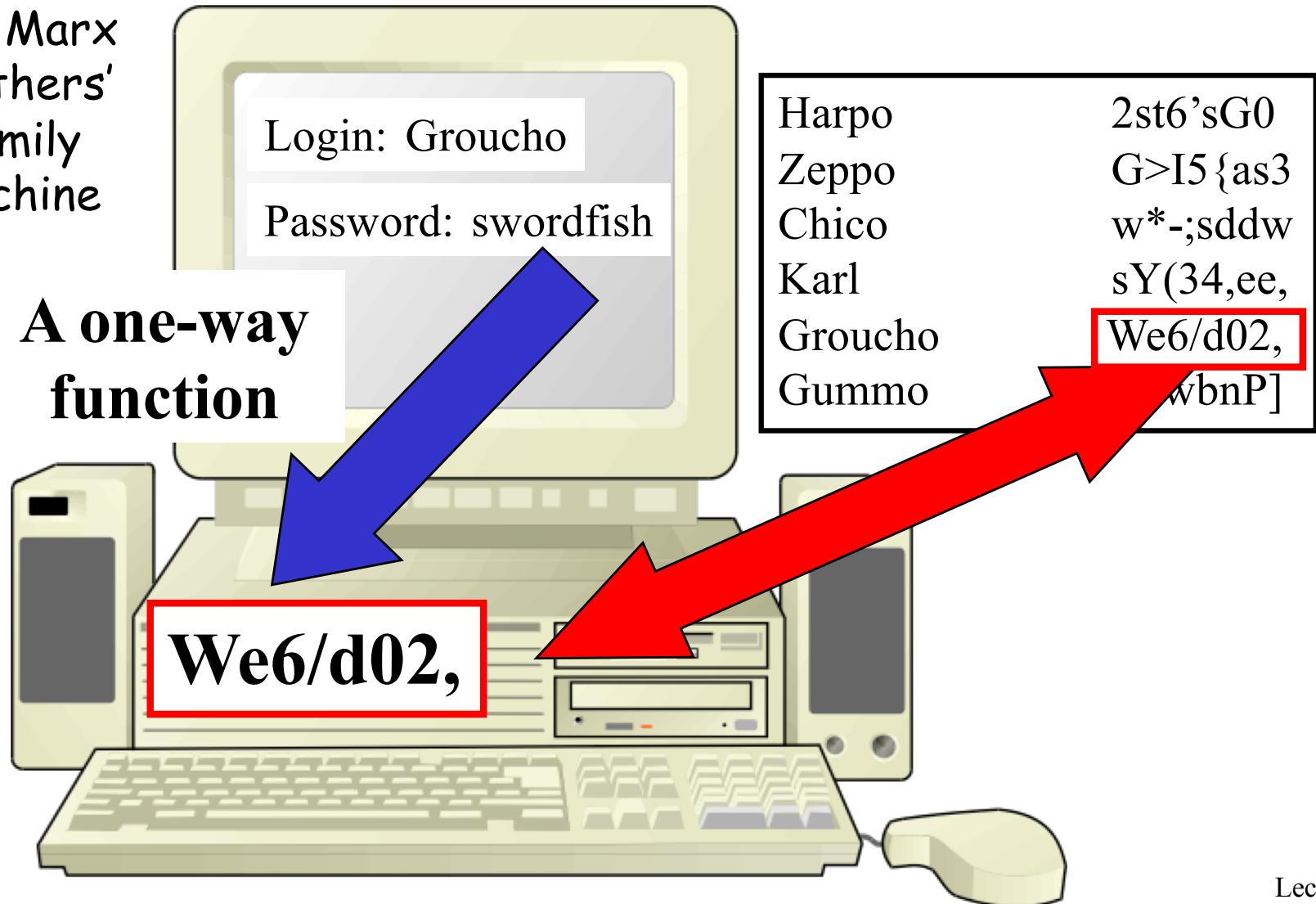
One Way Functions

- Functions that convert data A into data B
- But it's hard to convert data B back into data A
- Often done as a particular type of cryptographic operation
 - E.g., cryptographic hashing
- Depending on particular use, simple hashing might be enough

Standard Password Handling

The Marx
Brothers'
Family
Machine

A one-way
function



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 on an Encrypted Password File

Harpo
Zeppo
Chico
Karl
Groucho
Gummo

2st6'sG0
G>I5{as3
sY(34,ee
3(;wbnP]



Now you can hack
the Communist
Manifesto!

sY(34,ee

Rats!!!!

Dictionaries

- Real dictionary attacks don't use Webster's
- Dictionary based on probability of words being used as passwords
- Partly set up as procedures
 - E.g., try user name backwards
- Checks common names, proper nouns, etc. early in the process
- Tend to evolve to match user trends

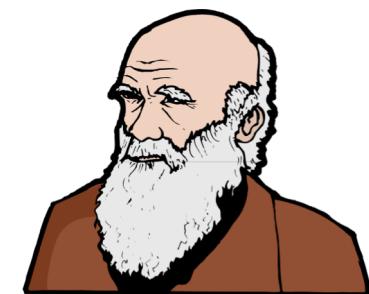
A Serious Issue

- All Linux machines use the same one-way function to encrypt passwords
- If someone runs the entire dictionary through that function,
 - Will they have a complete list of all encrypted dictionary passwords?
 - For all Linux systems?

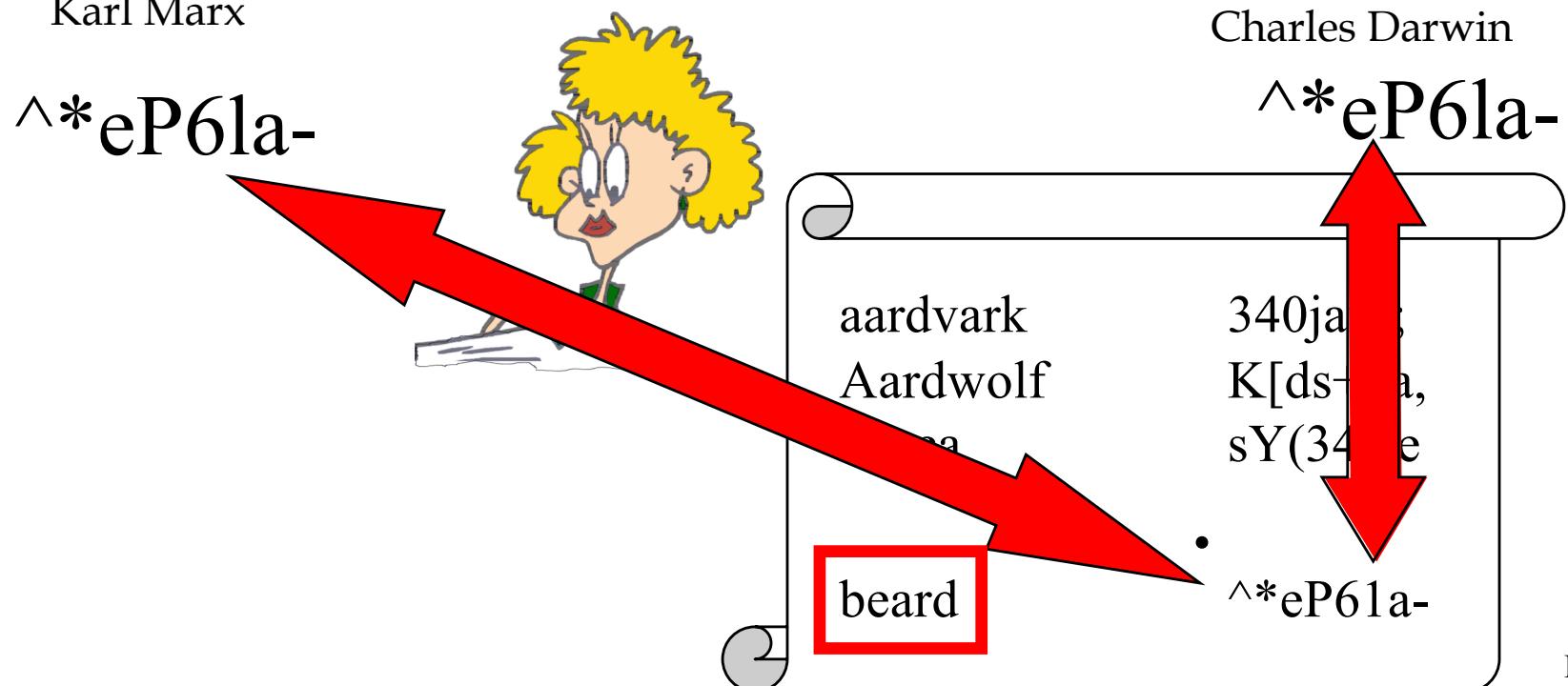
Illustrating the Problem



Karl Marx



Charles Darwin



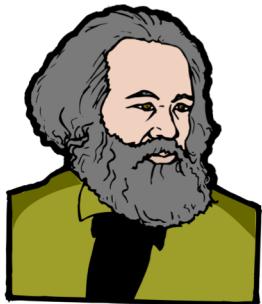
The Real Problem

- Not just that Darwin and Marx chose the same password
- But that anyone who chose that password got the same encrypted result
- So the attacker need only encrypt every possible password once
- And then she has a complete dictionary usable against anyone

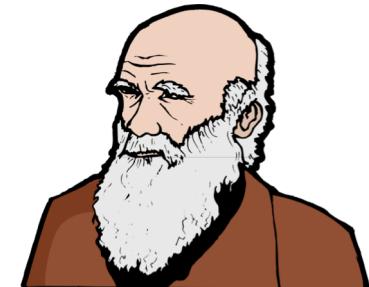
Salted Passwords

- 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

Did It Fix Our Problem?



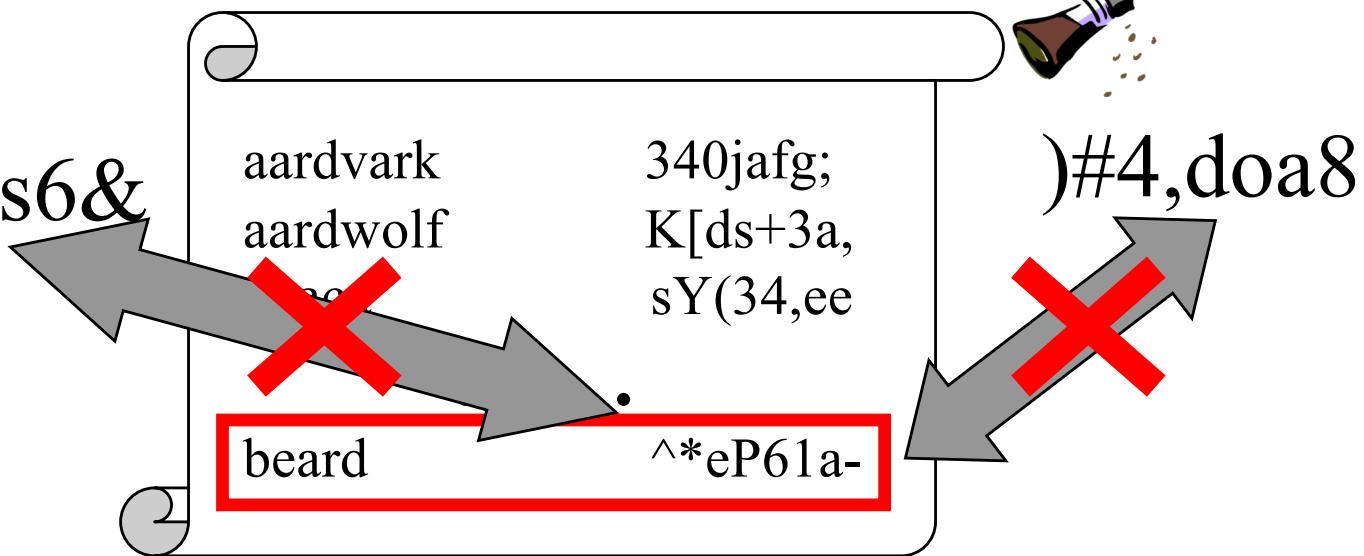
Karl Marx



Charles Darwin



D0Cls6&



What Is This Salt, Really?

- An integer that is combined with the password before hashing
- How will you be able to check passwords by hashing them, then?
- By storing the salt integer with the password
 - Generally in plaintext
- Note the resemblance to nonces and IVs
- Why is it OK (or OK-ish) to leave this important information in plaintext?

Modern Dictionary Attacks

- Modern machines are very fast
- Even with salting, huge dictionaries can be checked against encrypted passwords quickly
- In 2012, Ars Technica challenged 3 hackers to crack 16,000 hashed, salted passwords
 - Using dictionary attacks, they got 90% of them in 20 hours

GPU Password Cracking

- GPUs Even salted, hashed cracking
 - High passwords are in peril.
 - 200x faster than CPU
- Possible to build cheap GPU unit for this purpose (less than \$2000)
 - The moral?
- Prototypes crack 20-50% of passwords in example files in a few minutes

Password Management

- Limit login attempts
- Encrypt your passwords
- Protecting the password file
- Forgotten passwords
- Generating new passwords
- Password transport

Limit Login Attempts

- Don't allow dictionary attacks "over the wire"
- After some reasonable number of failed login attempts, do something
 - Lock account
 - Slow down
 - iPhone does this, Android doesn't
 - Watch more closely

Encrypt Your Passwords

- Using the techniques we just covered
- One would think this advice isn't necessary, but . . .
 - Yahoo lost half a million unencrypted passwords in 2012
- Encryption is more expensive and less convenient
 - But a lot more secure

Protecting the Password File

- So it's OK to leave the encrypted version of the password file around?
- No, it isn't
- Why make it easy for attackers?
- Dictionary attacks on single accounts still work
- And there are “popular” passwords, leading to easy dictionary attacks even with encryption
- Generally, don't give access to the encrypted file, either

Other Issues for Proper Handling of Users' Passwords

- Sites should store unencrypted passwords as briefly as possible
 - Partly issue of how they store the file
 - Partly issue of good programming
- Don't leave passwords in temp files or elsewhere
- Should not be possible to print or save someone's unencrypted password
- Use encrypted network transport for passwords
- If your server is compromised, all of this might not help

Wireless Networks and Passwords

- Wireless networks are often unencrypted
- Web sites used to request and transport passwords in the clear
- So eavesdroppers could hear passwords being transported
- Important to encrypt these messages
 - Use HTTPS if your web site requires passwords

Handling Forgotten Passwords

- Users frequently forget passwords
- How should your site deal with it?
- Bad idea:
 - Store plaintext passwords and send them on request
- Better idea:
 - Generate new passwords when old ones forgotten
- Example of common security theme:
 - Security often at odds with usability

Generating New Passwords

- Easy enough to generate a random one
- But you need to get it to the user
- If attacker intercepts it, authentication security compromised
- How do you get it to the user?

Transporting New Passwords

- Engineering solution is usually to send it in email
 - To an address the user registered with you earlier
- Often fine for practical purposes
- But there are very serious vulnerabilities
 - E.g., unencrypted wireless networks
- If you really care, use something else
 - E.g., surface mail

User Issues With Passwords

- Password proliferation
- Choosing passwords

Password Proliferation

- Practically every web site you visit wants you to enter a password
- Should you use the same password for all of them?
- Or a different password for each?

Using the Same Password

- + Easier to remember
- Much less secure

One password guesser gets all your authentication info

Do you trust all the sites you visit equally?

Compromise in one place compromises you everywhere

Real attacks are based on this vulnerability

Using Different Passwords

- + Much more secure
- But how many passwords can you actually remember?
- And you might “solve” this problem by choosing crummy passwords
- Or by altering one good password in a predictable way

Other Options

- Use a few passwords
 - Maybe classified by type of site or degree of trust
- Write down your passwords
 - Several disadvantages
 - Could write down hints, instead
- Use algorithm customized to sites
- Password vaults

Password Vaults

- Also known as key chains and password managers
- Store a lot of passwords in encrypted form
- Require another password to decrypt them
- Typically integrated with web browsers
 - Most users log into sites via browsers, anyway
- Single sign-on issue

Just say no to vaults that don't encrypt the passwords!

Choosing Passwords

- Typically a compromise between:
 - Sufficient security
 - Remembering it
- Major issues:
 - Length
 - Complexity

How Long Should Passwords Be?

- Generally a function of how easy it is for attackers to attack them
- Changes as speed of processors increase
- Nowadays, 15 character password are pretty safe
 - If they aren't guessable . . .
- Old sites may demand shorter ones

Some Password Strategies

Some password vaults will choose good passwords for you

- Use first letters from a phrase you remember (or entire phrase, if allowed)
- Use several randomly chosen words
- Replace letters with numbers and symbols
 - Helps, but less if you use common replacements (e.g., “0” for “o”)
 - Also less useful if you limit it to 1st and last character of password

Challenge/Response Authentication

- 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

Differences From Passwords

- Challenge/response systems ask for different information every time
- Or at least the questions come from a large set
- Best security achieved by requiring what amounts to encryption of the challenge
 - But that requires special hardware
 - Essentially, a smart card

Challenge/Response Problems

- Either the question is too hard to answer without special hardware
- Or the question is too easy for intruders to spoof the answer
- Still, commonly used in real-world situations
 - E.g., authenticating you by asking your childhood pet's name
 - “Security questions” used as an alternative to passwords

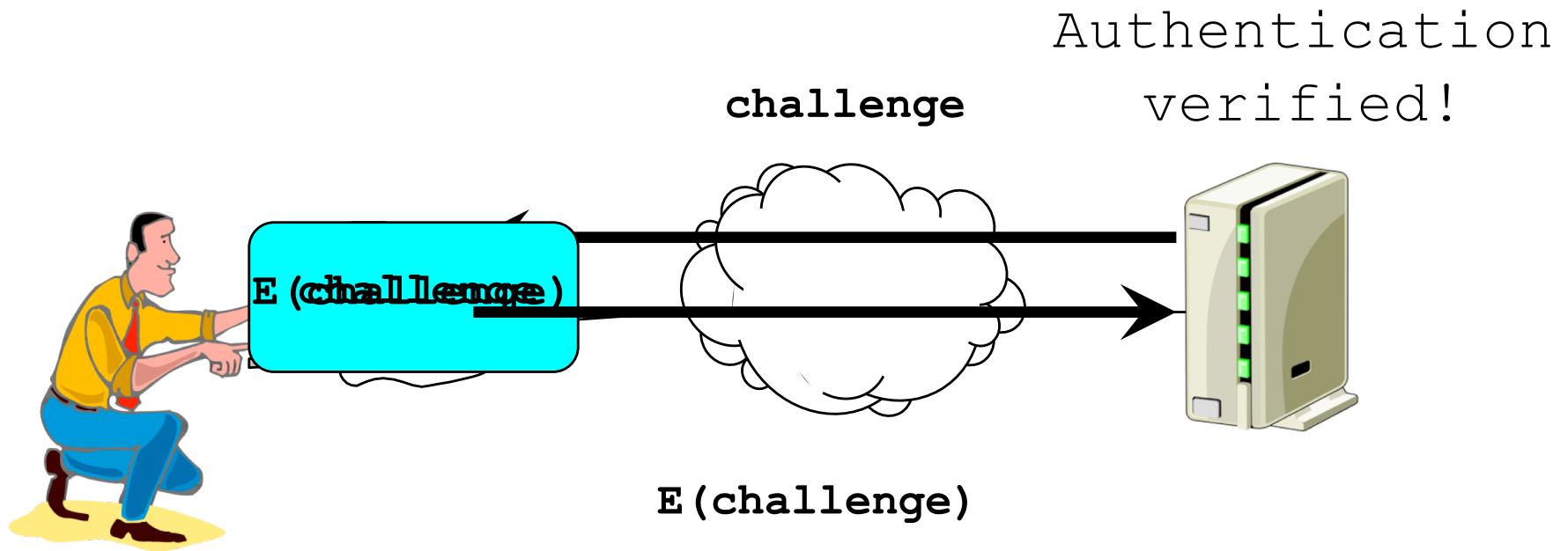
Identification Devices

- Authentication by what you have
- A smart card or other hardware device that is readable by the computer
- Authenticate by providing the device to the computer

Simple Use of Authentication Tokens

- If you have the token, you are identified
- Generally requires connecting the authentication device to computer
 - Unless done via wireless
- Weak, because it's subject to theft and spoofing
- How can we do better?

Authentication With Smart Cards



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

Some Details on Smart Cards

- Cryptography performed only on smart card
 - So compromised client machine can't steal keys
- Often user must enter password to activate card
 - Should it be entered to the card or the computer?

Problems With Identification Devices

- If lost or stolen, you can't authenticate yourself
 - And maybe someone else can
 - Often combined with passwords to avoid this problem (two factor authentication)
- Unless cleverly done, susceptible to sniffing attacks
- Requires special hardware

Authentication Through Biometrics

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

Problems With Biometric Authentication

- Requires very special hardware
 - Except systems that use typing patterns
- May not be as foolproof as you think
- Many physical characteristics vary too much for practical use
- Generally not helpful for authenticating programs or roles
- What happens when it's cracked?
 - You only have two retinas, after all

When Do Biometrics (Maybe) Work Well?

- When you use them for authentication
 - Carefully obtain clean readings from legitimate users
 - Compare those to attempts to authenticate
- When biometric readers are themselves secure
- When attacks are rare or difficult
- In conjunction with other authentication

When Do Biometrics (Definitely) Work Poorly?

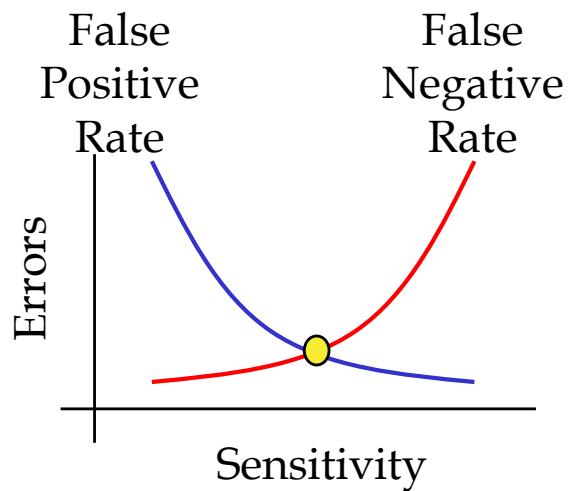
- Finding “needles in haystacks”
 - Face recognition of terrorists in airports
- When working off low-quality readings
- When the biometric reader is easy to bypass or spoof
 - Anything across a network is suspect
- When the biometric is “noisy”
 - Too many false negatives

Characterizing Biometric Accuracy

How many false positives?

Match made when it shouldn't have been
Versus how many false negatives?

Match not made when it should have been



The Crossover Error
Rate (CER)

Generally, the lower the
CER is, the better the system
But sometimes one rate more
important than the other

Some Typical Crossover Error Rates

Technology	Rate
Retinal Scan	1:10,000,000+
Iris Scan	1:131,000
Fingerprints	1:500
Facial Recognition	1:500
Hand Geometry	1:500
Signature Dynamics	1:50
Voice Dynamics	1:50

Data as of 2002

Things can improve a lot in this area over time

Also depends on how you use them

And on what's important to your use

Biometrics and Usability

- Always a tradeoff in false positives vs. false negatives
- For consumer devices, false negatives are very, very bad
 - People discard devices that won't let the legitimate user in
- Can you make the false positive rate non-trivial with almost no false negs?

Didn't Carnegie Mellon Perfect Facial Recognition?

- Not really
- Quick and dirty version got 1 in 3 right
- With more photos and time, did better
- Facebook claims to have improved on that (97+% on matching photos, 2014)
- But think about how accurate your use of biometrics needs to be
- In many cases, you need 5 nines or so

Authentication by Where You Are

- Sometimes useful in ubiquitous computing
- The issue is whether the message in question is coming from the machine that's nearby
- Less important who owns that machine
- Requires sufficient proof of physical location
- And ability to tie a device at that location to its messages
- Sometimes used in conjunction with other authentication methods
 - E.g., the door opens only if an authorized user is right outside it

Multi-Factor Authentication

- Our options for authentication have a problem:

They all suck!

- People choose lousy passwords
- Smart cards get stolen
- Biometrics have the Goldilocks problem

So What To Do?

- Maybe reduce the problems of each approach by using the strengths of the others
- Require more than one type of authentication
- Unless all work, don't authenticate

Practical Multifactor Authentication

- Most commonly:
 - Something you know + something you have
 - E.g., password and smart phone
 - Provide the password and respond to a probe of your phone
 - ATMs have used this idea for years
 - Your PIN and ATM card
- Other combos, are possible, though

Is This Better?

- Now two things need to go wrong for false authentication
 - But now either thing can go wrong for a false negative . . .
- Are the factors really orthogonal?
- Are both factors non-trivial?
- Is one factor likely to suffer a catastrophic break?

Multifactor Authentication and Users

- Users generally don't like it
 - At least at first
- Only works when they're motivated
 - E.g., they can't avoid it and can't access vital things without it
- They might get over it . . .

Regardless, It's State-of-the-Art

- All expert security advice says to use multi-factor authentication
- Especially for anything with any serious implications
- Not quite everywhere yet
- But becoming more common

Operating System Security

Computer Security

Peter Reiher

January 28, 2021

Outline

- What does the OS protect?
- Authentication for operating systems
- Memory protection
 - Buffer overflows
- IPC protection
 - Covert channels
- Stored data protection
 - Full disk encryption

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

Single User Vs. Multiple User Machines

- The majority of today's computers usually support a single user
- Some computers are still multi-user
 - Often specialized servers
- Single user machines often run multiple processes, though
 - Often through downloaded code
- Increasing numbers of embedded machines
 - Effectively no (human) user

Trusted Computing

- Since OS security is vital, how can we be sure our OS is secure?
- Partly a question of building in good security mechanisms
- But also a question of making sure you're running the right OS
 - And it's unaltered
- That's called *trusted computing*

How Do We Achieve Trusted Computing?

- From the bottom up
- We need hardware we can count on
- It can ensure the boot program behaves
- The boot can make sure we run the right OS
- The OS will protect at the application level

TPM and Bootstrap Security

- Trusted Platform Module (TPM)
 - Special hardware designed to improve OS security
- Proves OS was booted with a particular bootstrap loader
 - Using tamperproof HW and cryptographic techniques
- Also provides secure key storage and crypto support

TPM and the OS Itself

- Once the bootstrap loader is operating, it uses TPM to check the OS
- Essentially, ensures that expected OS was what got booted
- OS can request TPM to verify applications it runs
- Remote users can request such verifications, too

Transitive Trust in TPM

- You trust the app, because the OS says to trust it
- You trust the OS, because the bootstrap says to trust it
- You trust the bootstrap, because somebody claims it's OK
- You trust the whole chain, because you trust the TPM hardware's attestations

Trust vs. Security

- TPM doesn't guarantee security
 - It (to some extent) verifies trust
- It doesn't mean the OS and apps are secure, or even non-malicious
- It just verifies that they are versions you have said you trust
- Offers some protection against tampering with software
- But doesn't prevent other bad behavior

Status of TPM

- Hardware widely installed
 - Not widely used
- Microsoft Bitlocker uses it
 - When available
- A secure Linux boot loader and OS work with it
- Some specialized software uses TPM

SecureBoot

- A somewhat different approach to ensuring you boot the right thing
- Built into the boot hardware and SW
- Designed by Microsoft
- Essentially, only allows booting of particular OS versions

Some Details of SecureBoot

- Part of the Unified Extensible Firmware Interface (UEFI)
 - Replacement for BIOS
- Microsoft insists on HW supporting these features
- Only boots systems with pre-arranged digital signatures
- Some issues of who can set those

Security Enclaves

- Many modern processors have trusted hardware components on the chip
- Typically intended to handle security-sensitive operations
 - Often by hiding crypto keys
- The approach has proven challenging
- Most such hardware has known flaws

Authentication and Authorization in Operating Systems

- The OS must authenticate all user requests
 - Otherwise, can't control access to critical resources
- Human users log in
 - Locally or remotely
- Processes run on their behalf
 - And request resources
- Once authenticated, requests must be authorized

In-Person User Authentication

- Authenticating the physically present user
- Most frequently using password techniques
- Sometimes biometrics
- To verify that a particular person is sitting in front of keyboard and screen

Remote User Authentication

- Many users access machines remotely
- How are they authenticated?
- Most typically by password
- Sometimes via public key crypto
- Sometimes at OS level, sometimes by a particular process
 - In latter case, what is their OS identity?
 - What does that imply for security?

Process Authentication

- Successful login creates a primal process
 - Under ID of user who logged in
- The OS securely ties a process control block to the process
 - Not under user control
 - Contains owner's ID
- Processes can fork off more processes
 - Usually child process gets same ID as parent
- Usually, special system calls can change a process' ID

For Example,

- Process X wants to open file Y for read
- File Y has read permissions set for user Bill
- If process X belongs to user Bill, system ties the open call to that user
- And file system checks ID in open system call to file system permissions
- Other syscalls (e.g., RPC) similar

Authorization in Operating Systems

- Operating systems allow user processes to perform system calls
 - Which generally do things that not all users/processes should do
- When operation requires permissions, we need to check those
- When is that?
- When should the OS perform authorization?

Authorization and Reference Monitors

- If an operation requires authorization, it should pass through a reference monitor
- Reference monitors add overhead
 - So we don't want to use them unnecessarily
- But when will it be necessary?
- A question for OS design and implementation

Protecting Memory

- What is there to protect in memory?
- Page tables and virtual memory protection
- Special security issues for memory
- Buffer overflows

What Is In Memory?

- Executable code
 - Integrity required to ensure secure operations
- Copies of permanently stored data
 - Secrecy and integrity issues
- Temporary process data
 - Mostly integrity issues

Mechanisms for Memory Protection

- Most general purpose systems provide some memory protection
 - Logical separation of processes that run concurrently
- Usually through virtual memory methods
- Originally arose mostly for error containment, not security

Paging and Security

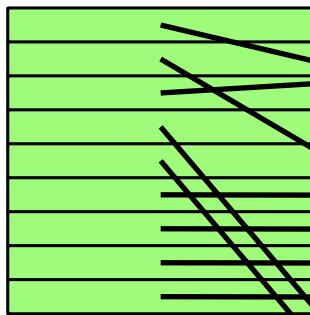
- Main memory is divided into page frames
- Every process has an address space divided into logical pages
- For a process to use a page, it must reside in a page frame
- If multiple processes are running, how do we protect their frames?

Protection of Pages

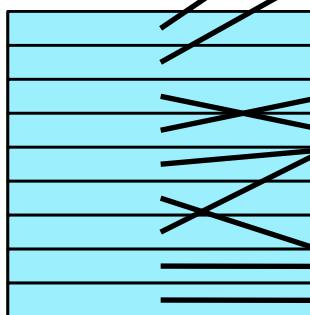
- Each process is given a page table
 - Translation of logical addresses into physical locations
- All addressing goes through page table
 - At unavoidable hardware level
- If the OS is careful about filling in the page tables, a process can't even name other processes' pages

Page Tables and Physical Pages

Process Page Tables



Process A



Process B

Physical Page Frames



Any address
Process A
names goes
through the
green table

Any address
Process B
names goes
through the
blue table

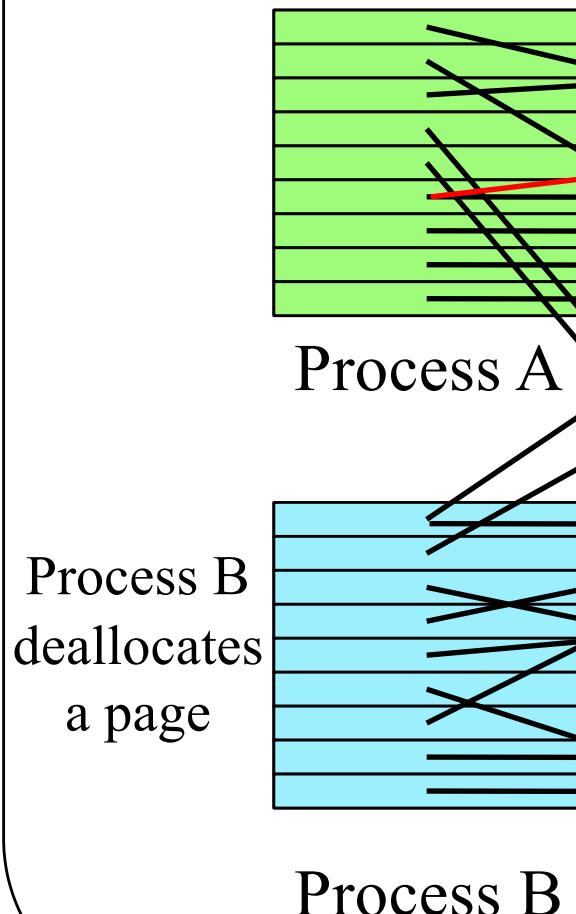
They can't
even name
each other's
pages

Security Issues of Page Frame Reuse

- A common set of page frames is shared by all processes
- The OS switches ownership of page frames as necessary
- When a process acquires a new page frame, it used to belong to another process
 - Can the new process read the old data?

Reusing Pages

Process Page Tables



Physical Page Frames

What happens now if Process A requests a page?

Can Process A now read Process B's deallocated data?

Strategies for Cleaning Pages

- Don't bother
 - Basic Linux strategy
- Zero on deallocation
- Zero on reallocation
- Zero on use
- Clean pages in the background
 - Windows strategy

Special Interfaces to Memory

- Some systems provide a special interface to memory
- If the interface accesses physical memory,
 - And doesn't go through page table protections,
 - Then attackers can read the physical memory
 - Letting them figure out what's there and find what they're looking for

Buffer Overflows

- One of the most common causes for compromises of operating systems
- Due to a flaw in how operating systems handle process inputs
 - Or a flaw in programming languages
 - Or a flaw in programmer training
 - Depending on how you look at it

What Is a Buffer Overflow?

- A program requests input from a user
- It allocates a temporary buffer to hold the input data
- It then reads all the data the user provides into the buffer, but . . .
- It doesn't check how much data was provided

For Example,

```
int main() {  
    char name[32];  
    printf("Please type your name: ");  
    gets(name);  
    printf("Hello, %s", name);  
    return (0);  
}
```

- What if the user enters more than 32 characters?

Well, What If the User Does?

- Code continues reading data into memory
- The first 32 bytes go into name buffer
 - Allocated on the stack
 - Close to record of current function
- The remaining bytes go onto the stack
 - Right after name buffer
 - Overwriting current function record
 - Including the instruction pointer

Why Is This a Security Problem?

- The attacker can cause the function to “return” to an arbitrary address
- But all attacker can do is run different code than was expected
- He hasn’t gotten into anyone else’s processes
 - Or data
- So he can only fiddle around with his own stuff, right?

Is That So Bad?

- Well, yes
- That's why a media player can write configuration and data files
- Unless roles and access permissions set up very carefully, a typical program can write all its user's files

The Core Buffer Overflow Security Issue

- Programs often run on behalf of others
 - But using your identity
- Maybe OK for you to access some data
- But is it OK for someone who you're running a program for to access it?
 - Downloaded programs
 - Users of web servers
 - Many other cases

Using Buffer Overflows to Compromise Security

- Carefully choose what gets written into the instruction pointer
- So that the program jumps to something you want to do
 - Under the identity of the program that's running
- Such as, execute a command shell
- Usually attacker provides this code

Effects of Buffer Overflows

- A remote or unprivileged local user runs a program with greater privileges
- If buffer overflow is in a root program, it gets all privileges, essentially
- Can also overwrite other stuff
 - Such as heap variables
- Common mechanism to allow attackers to break into machines

Stack Overflows

- The most common kind of buffer overflow
- Intended to alter the contents of the stack
- Usually by overflowing a dynamic variable
- Usually with intention of jumping to exploit code
 - Though it could instead alter parameters or variables in other frames
 - Or even variables in current frame

Heap Overflows

- Heap is used to store dynamically allocated memory
- Buffers kept there can also overflow
- Generally doesn't offer direct ability to jump to arbitrary code
- But potentially quite dangerous

What Can You Do With Heap Overflows?

- Alter variable values
- “Edit” linked lists or other data structures
- If heap contains list of function pointers, can execute arbitrary code
- Generally, heap overflows are harder to exploit than stack overflows
- But they exist
 - E.g., one discovered in Google Chrome in 2020 (not Chrome’s first)

Some Recent Buffer Overflows

- Adobe Reader
- Cisco Integrated Management Controller
- Grub2 bootloader
- Instagram for Android
- sudo for Linux
- S3 CODESYS automation software

Fixing Buffer Overflows

- Write better code (check input lengths, etc.)
- Use programming languages that prevent them
- Add OS controls that prevent overwriting the stack
- Put things in different places on the stack, making it hard to find the return pointer (e.g., Microsoft ASLR)
- Don't allow execution from places in memory where buffer overflows occur (e.g., Windows DEP)
 - Or don't allow execution of writable pages
- Why aren't these things commonly done?
 - Sometimes they are, but not always effective
- When not, presumably because programmers and designers neither know nor care about security

Protecting Interprocess Communications

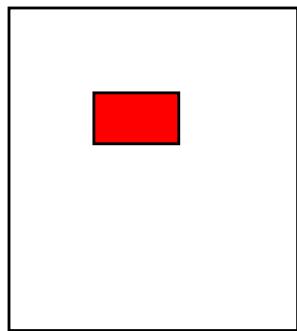
- Operating systems provide various kinds of interprocess communications
 - Messages
 - Semaphores
 - Shared memory
 - Sockets
- How can we be sure they're used properly?

IPC Protection Issues

- How hard it is depends on what you're worried about
- For the moment, let's say we're worried about one process improperly using IPC to get info from another
 - Process A wants to steal information from process B
- How would process A do that?

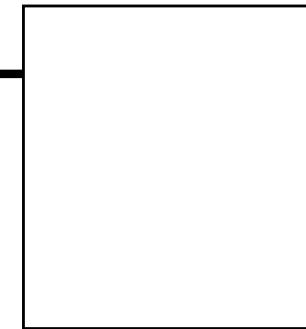
Message Security

Process A



Gimme your
secret

Process B



That's probably
not going to work

Can process B use message-based IPC to steal the secret?

How Can B Get the Secret?

- He can convince the system he's A
 - A problem for authentication
- He can break into A's memory
 - That doesn't use message IPC
 - And is handled by page tables
- He can forge a message from someone else to get the secret
 - But OS tags IPC messages with identities
- He can “eavesdrop” on someone else who gets the secret

Can an Attacker Really Eavesdrop on IPC Message?

- On a single machine, what is a message send, really?
- A copy from a process buffer to an OS buffer
 - Then from OS buffer to another process' buffer
 - Sometimes optimizations skip some copies
- If attacker can't get at processes' internal buffers and can't get at OS buffers, he can't "eavesdrop"
- Need to handle page reuse (discussed earlier)
- Also an issue for properly checking authorization (discussed earlier)

Other Forms of IPC

- Semaphores, sockets, shared memory, RPC
- Pretty much all the same
 - Use system calls for access
 - Which belong to some process
 - Which belongs to some principal
 - OS can check principal against access control permissions at syscall time
 - Ultimately, data is held in some type of memory
 - Which shouldn't be improperly accessible

So When Is It Hard?

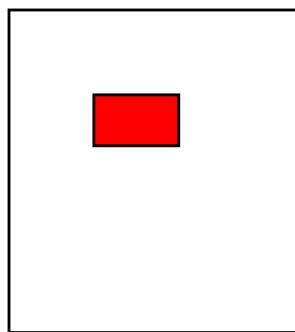
1. When there's a bug in the OS
 - E.g., not always checking authorization
 - Allowing masquerading, eavesdropping, etc.
 - Or, if the OS itself is compromised, all bets are off
2. What if it's not a single machine?
3. What if the OS has to prevent cooperating processes from sharing information?

Distributed System Issues

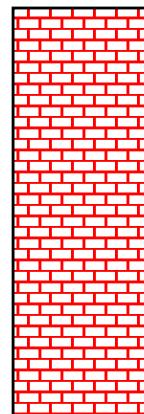
- What if your RPC is really remote?
- RPC tries to make remote access look “just like” local access
- The hard part is authentication
 - The call didn’t come from your OS
 - How do you authenticate its origin?
- With usual remote authentication and authorization mechanisms

The Other Hard Case

Process A



Process B



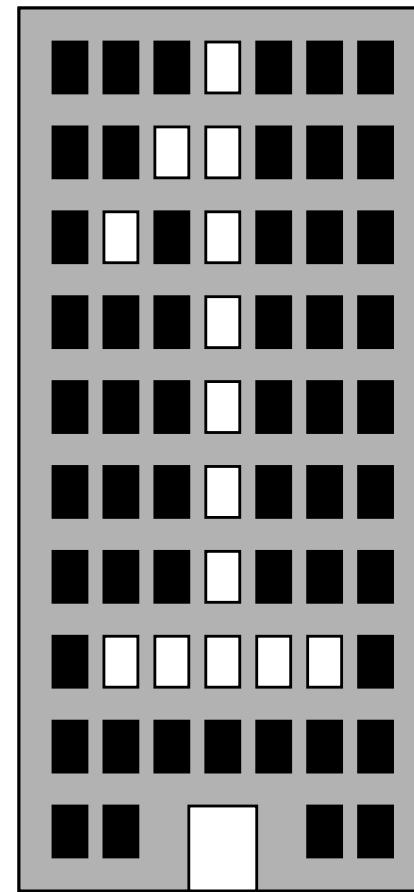
Process A wants to tell the secret to process B
But the OS has been instructed to prevent that
A necessary part of Bell-La Padula, e.g.
Can the OS prevent A and B from colluding
to get the secret to B?

OS Control of Interactions

- OS can “understand” the security policy
- Can maintain labels on files, process, data pages, etc.
- Can regard any IPC or I/O as a possible leak of information
 - To be prohibited if labels don’t allow it

Covert Channels

- Tricky ways to pass information
- Requires cooperation of sender and receiver
 - Generally in active attempt to deceive system
- Use something not ordinarily regarded as a communications mechanism



Covert Channels in Computers

- Generally, one process “sends” a covert message to another
 - But could be computer to computer
- How?
 - Disk activity
 - Page swapping
 - Time slice behavior
 - Use of a peripheral device
 - Limited only by imagination

Handling Covert Channels

- Relatively easy if you know details of how the channel is used
 - Put randomness/noise into channel to wash out message
- Hard to impossible if you don't know what the channel is
- Not most people's problem

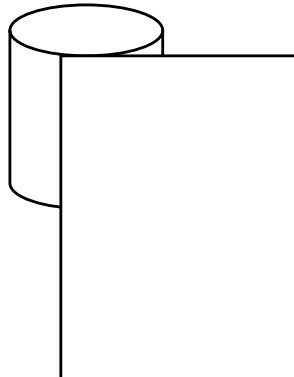
Stored Data Protection

- Files are a common example of a typically shared resource
- If an OS supports multiple users, it needs to address the question of file protection
- Simple read/write access control
- What else do we need to do?
- Protect the raw disk or SSD

Encrypted File Systems

- Data stored on disk is subject to many risks
 - Improper access through OS flaws
 - But also somehow directly accessing the disk
- If the OS protections are bypassed, how can we protect data?
- How about if we store it in encrypted form?

An Example of an Encrypted File System



SECRET
\$100 to
fix
software
problems

K_s

Issues for
encrypted file
systems:

When does the
cryptography occur?

Where does the
key come from?

What is the
granularity of
cryptography?

When Does Cryptography Occur?

- Transparently when a user opens a file?
 - In disk drive?
 - In OS?
 - In file system?
- By explicit user command?
 - Or always, implicitly?
- How long is the data decrypted?
- Where does it exist in decrypted form?

Where Does the Key Come From?

- Provided by human user?
- Stored somewhere in file system?
- Stored on a smart card?
- Stored in the disk hardware?
- Stored on another computer?
- Where and for how long do we store the key?

What Is the Granularity of Cryptography?

- An entire disk?
- An entire file system?
- Per file?
- Per block?
- Consider both in terms of:
 - How many keys?
 - When is a crypto operation applied?

What Are You Trying to Protect Against With Crypto File Systems?

- Unauthorized access by improper users?
 - Why not just access control?
- The operating system itself?
 - What protection are you really getting?
 - Unless you're just storing data on the machine
- Data transfers across a network?
 - Why not just encrypt while in transit?
- Someone who accesses the device not using the OS?
 - A realistic threat in your environment?

Full Disk Encryption

- All data on the disk is encrypted
- Data is encrypted/decrypted as it enters/leaves disk
- Primary purpose is to prevent improper access to stolen disks
 - Designed mostly for portable machines (laptops, tablets, etc.)

HW Vs. SW Full Disk Encryption

- HW advantages:
 - Faster
 - Totally transparent, works for any OS
 - Setup probably easier
- HW disadvantages:
 - Not ubiquitously available today
 - More expensive (not that much, though)
 - Might not fit into a particular machine
 - Backward compatibility

Example of Hardware Full Disk Encryption

- Seagate's Momentus 7200 FDE line
- Hardware encryption for entire disk
 - Using AES
- Key accessed via user password, smart card, or biometric authentication
 - Authentication information stored internally on disk
 - Check performed by disk, pre-boot
- 3 Gbytes/sec maximum transfer rate (2021)
- Primarily for laptops

Example of Software Full Disk Encryption

- Microsoft BitLocker
- Doesn't encrypt quite the whole drive
 - Unencrypted partition holds bootstrap
- Uses AES for cryptography
- Key stored either in special hardware or USB drive
- Microsoft claims “single digit percentage” overhead
 - One independent study claims 12%

Network Security Computer Security

Peter Reiher

February 2, 2021

Some Important Network Characteristics for Security

- Degree of locality
- Media used
- Protocols used

Degree of Locality

- Some networks are very local
 - E.g., an Ethernet
 - Benefits from:
 - Physical locality
 - Small number of users and machines
 - Common goals and interests
- Other networks are very non-local
 - E.g., the Internet backbone
 - Many users/sites share bandwidth

Network Media

- Some networks are wires, cables, or over telephone lines
 - Can be physically protected
- Other networks are satellite links or other radio links
 - Physical protection possibilities more limited

Protocol Types

- TCP/IP is the most used
 - But it only specifies some common intermediate levels
 - Other protocols exist above and below it
- In places, other protocols replace TCP/IP
- And there are lots of supporting protocols
 - Routing protocols, naming and directory protocols, network management protocols
 - And security protocols (IPSec, ssh, tls)

Implications of Protocol Type

- The protocol defines a set of rules that will always be followed
 - But usually not quite complete
 - And they assume everyone is at least trying to play by the rules
 - What if they don't?
- Specific attacks exist against specific protocols

Threats To Networks

- Wiretapping
- Impersonation
- Attacks on message
 - Confidentiality
 - Integrity
- Denial of service attacks

Wiretapping

- **Passive wiretapping** is listening in illicitly on conversations
- **Active wiretapping** is injecting traffic illicitly
- **Packet sniffers** can listen to all traffic on a broadcast medium
 - Ethernet or 802.11, e.g.
- Wiretapping on wireless often just a matter of putting up an antenna

Impersonation

- A packet comes in over the network
 - With some source indicated in its header
- Often, the action to be taken with the packet depends on the source
- But attackers may be able to create packets with false sources

Violations of Message Confidentiality

- Other problems can cause messages to be inappropriately divulged
- Misdelivery can send a message to the wrong place
 - Clever attackers can make it happen
- Message can be read at an intermediate gateway or a router
- Sometimes an intruder can get useful information just by traffic analysis

Message Integrity

- Even if the attacker can't create the packets he wants, sometimes he can alter proper packets
- To change the effect of what they will do
- Typically requires access to part of the path message takes

Denial of Service

- Attacks that prevent legitimate users from doing their work
- By flooding the network
- Or corrupting routing tables
- Or flooding routers
- Or destroying key packets

How Do Denial of Service Attacks Occur?

- Basically, the attacker injects some form of traffic
- Most current networks aren't built to throttle uncooperative parties very well
- All-inclusive nature of the Internet makes basic access trivial
- Universality of IP makes reaching most of the network easy

An Example: SYN Flood

- Based on vulnerability in TCP
- Attacker uses initial request/response to start TCP session to fill a table at the server
- Preventing new real TCP sessions
- SYN cookies and firewalls with massive tables are possible defenses

Normal SYN Behavior

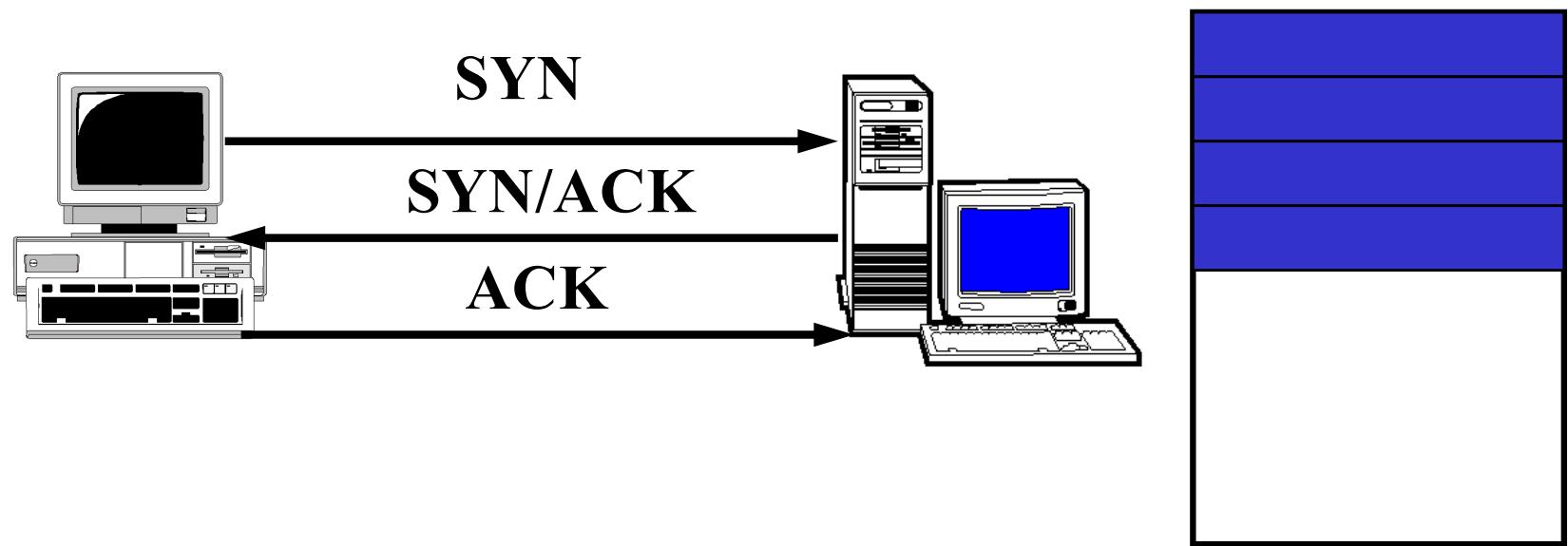
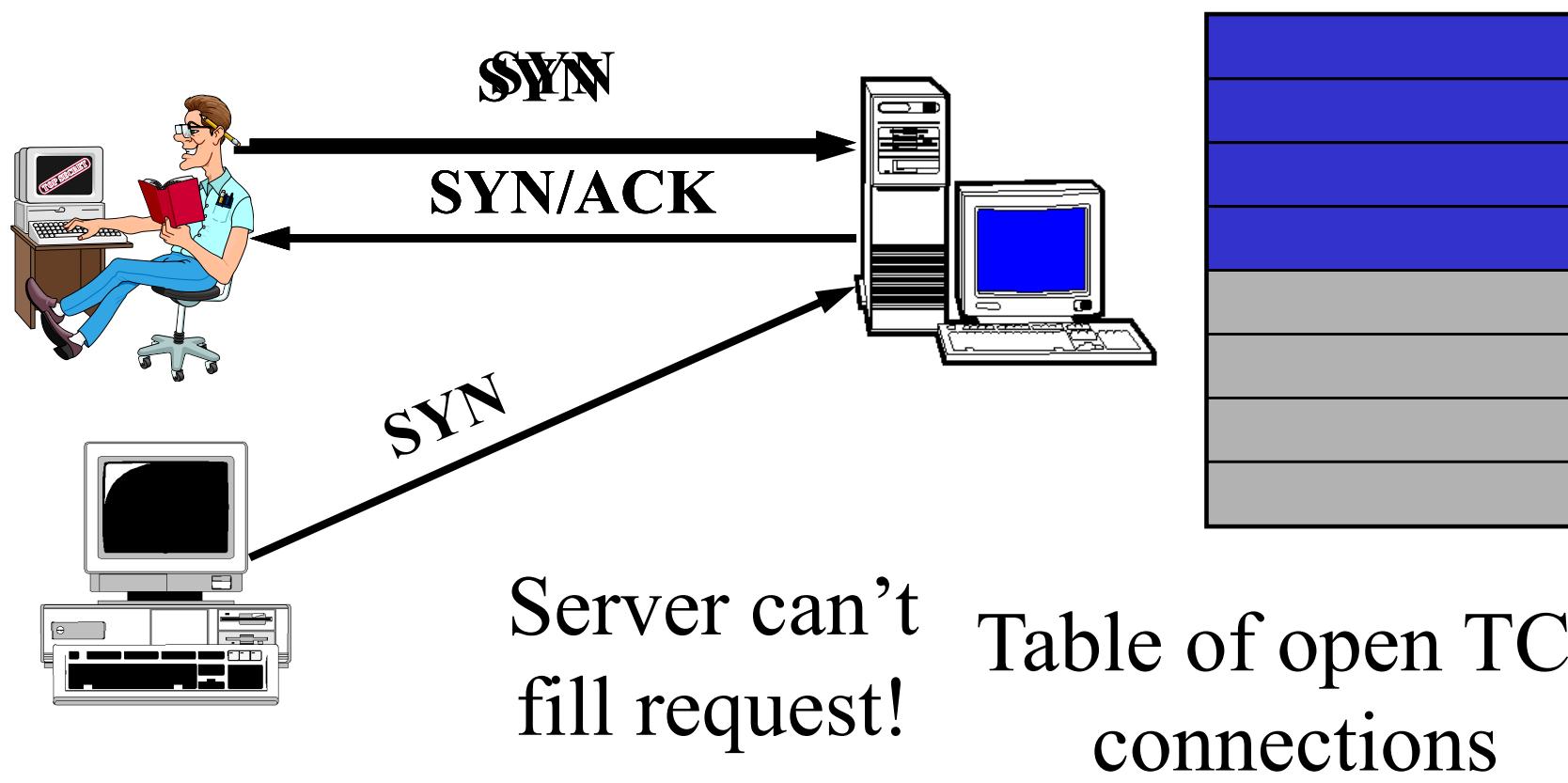


Table of open TCP
connections

A SYN Flood



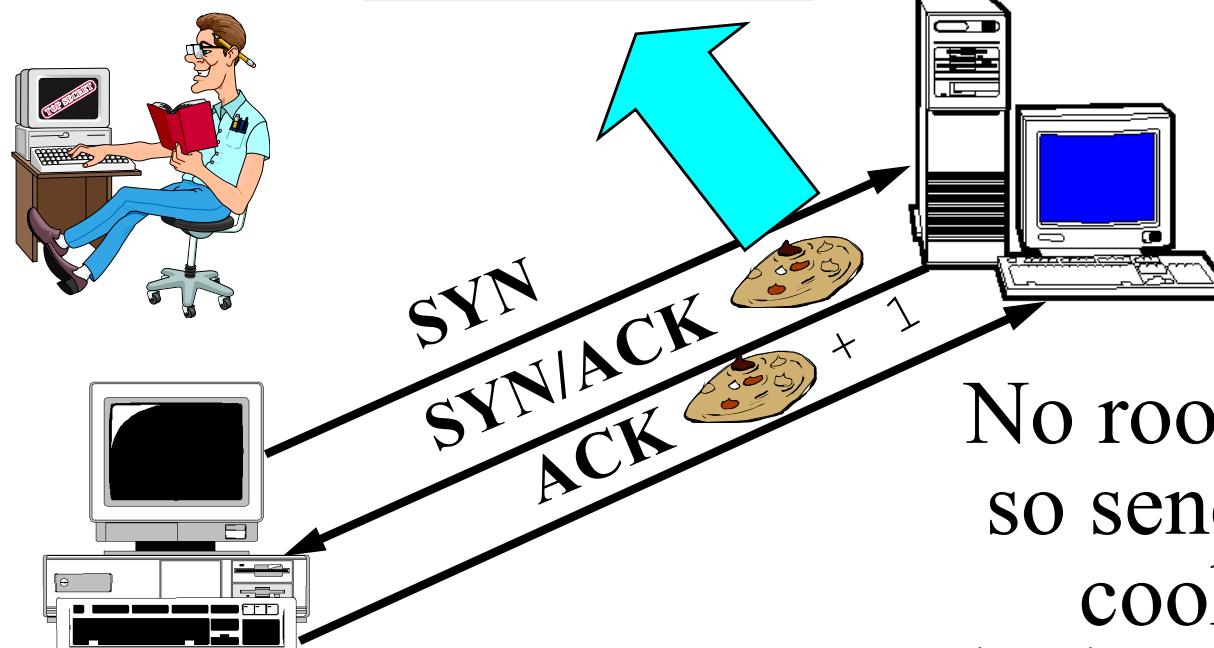
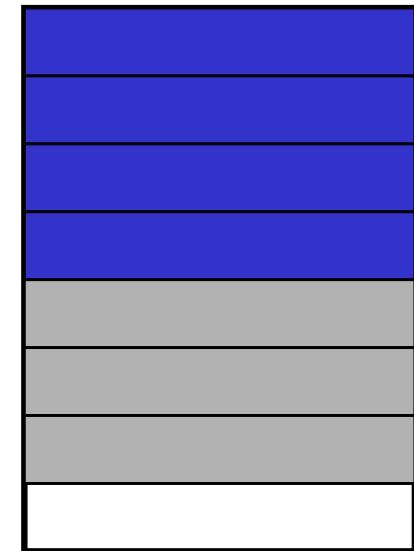
And no changes
to TCP protocol
itself

SYN Cookies

SYN/ACK number is
secret function of
various information

Client IP address
& port, server's
IP address and
port, and a timer

KEY POINT:
Server doesn't
need to save
cookie value!



No room in the table,
so send back a SYN
cookie, instead
Server recalculates cookie to
determine if proper response

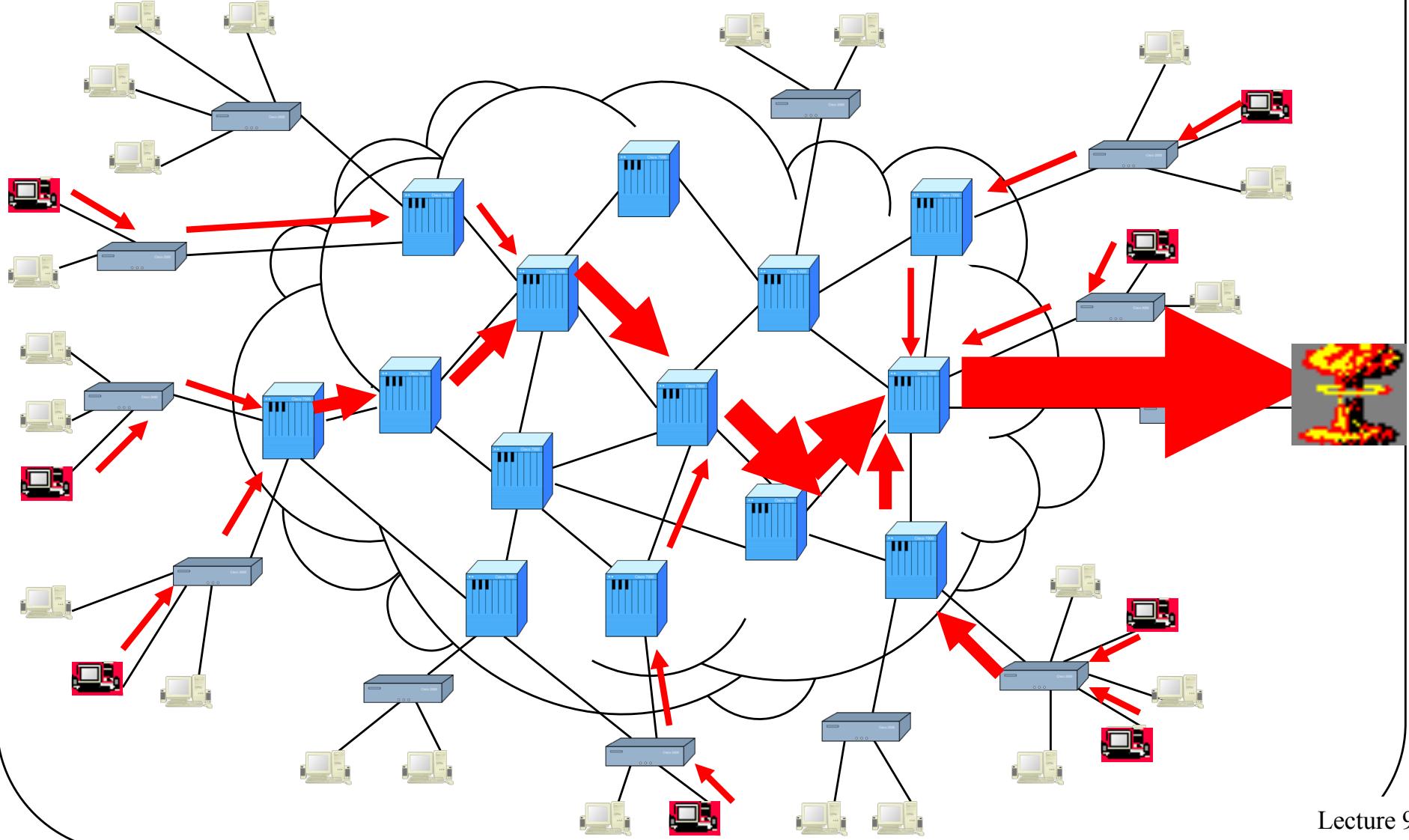
General Network Denial of Service Attacks

- Need not tickle any particular vulnerability
- Can achieve success by mere volume of packets
- If more packets sent than can be handled by target, service is denied
- A hard problem to solve

Distributed Denial of Service Attacks

- Goal: Prevent a network site from doing its normal business
- Method: overwhelm the site with attack traffic
- Response: ?

The Problem



Why Are These Attacks Made?

- Generally to annoy
- Sometimes for extortion
- Sometimes to prevent adversary from doing something important
- If directed at infrastructure, might cripple parts of Internet

Attack Methods

- Pure flooding
 - Of network connection
 - Or of upstream network
- Overwhelm some other resource
 - SYN flood
 - CPU resources
 - Memory resources
 - Application level resource
- Direct or reflection

Why “Distributed”?

- Targets are often highly provisioned servers
- A single machine usually cannot overwhelm such a server
- So harness multiple machines to do so
- Also makes defenses harder

How to Defend?

- A vital characteristic:
 - Don't just stop a flood
 - ENSURE SERVICE TO LEGITIMATE CLIENTS!!!
- If you deliver a manageable amount of garbage, you haven't solved the problem
- Nor have you if you prevent a flood by dropping all packets

Complicating Factors

- High availability of compromised machines
 - Millions of zombie machines out there
- Internet is designed to deliver traffic
 - Regardless of its value
- IP spoofing allows easy hiding
- Distributed nature makes legal approaches hard
- Attacker can choose all aspects of his attack packets
 - Can be a lot like good ones

Basic Defense Approaches

- Overprovisioning
- Dynamic increases in provisioning
- Hiding
- Tracking attackers
- Legal approaches
- Reducing volume of attack
- None of these are totally effective

Traffic Control Mechanisms

- Filtering
 - Source address filtering
 - Other forms of filtering
- Rate limits
- Protection against traffic analysis
 - Padding
 - Routing control

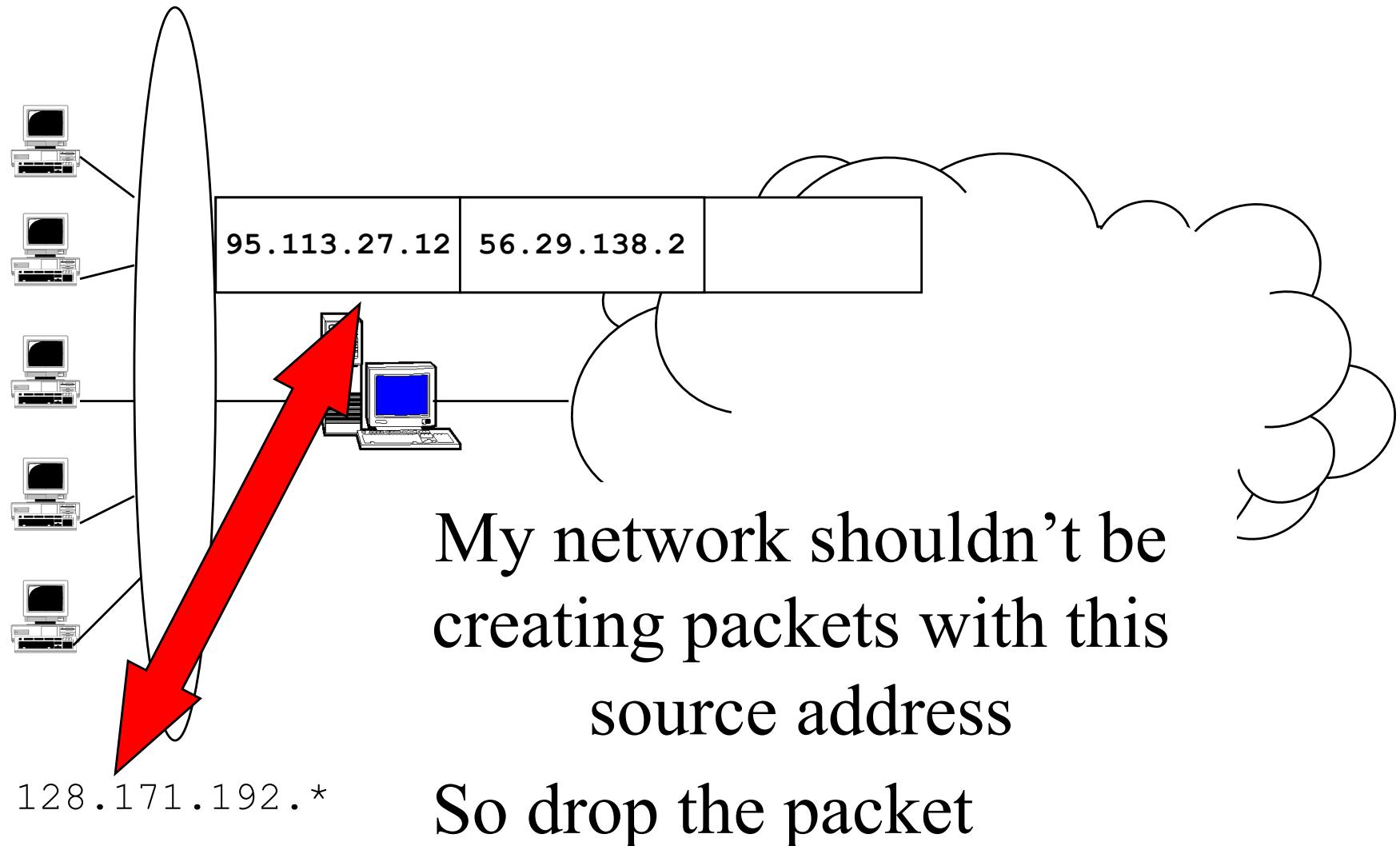
Source Address Filtering

- Filtering out some packets because of their source address value
 - Usually because you believe their source address is spoofed
- Often called ingress filtering
 - Or egress filtering . . .

Source Address Filtering for Address Assurance

- Router “knows” what network it sits in front of
 - In particular, knows IP addresses of machines there
- Filter outgoing packets with source addresses not in that range
- Prevents your users from spoofing other nodes’ addresses
 - But not from spoofing each other’s

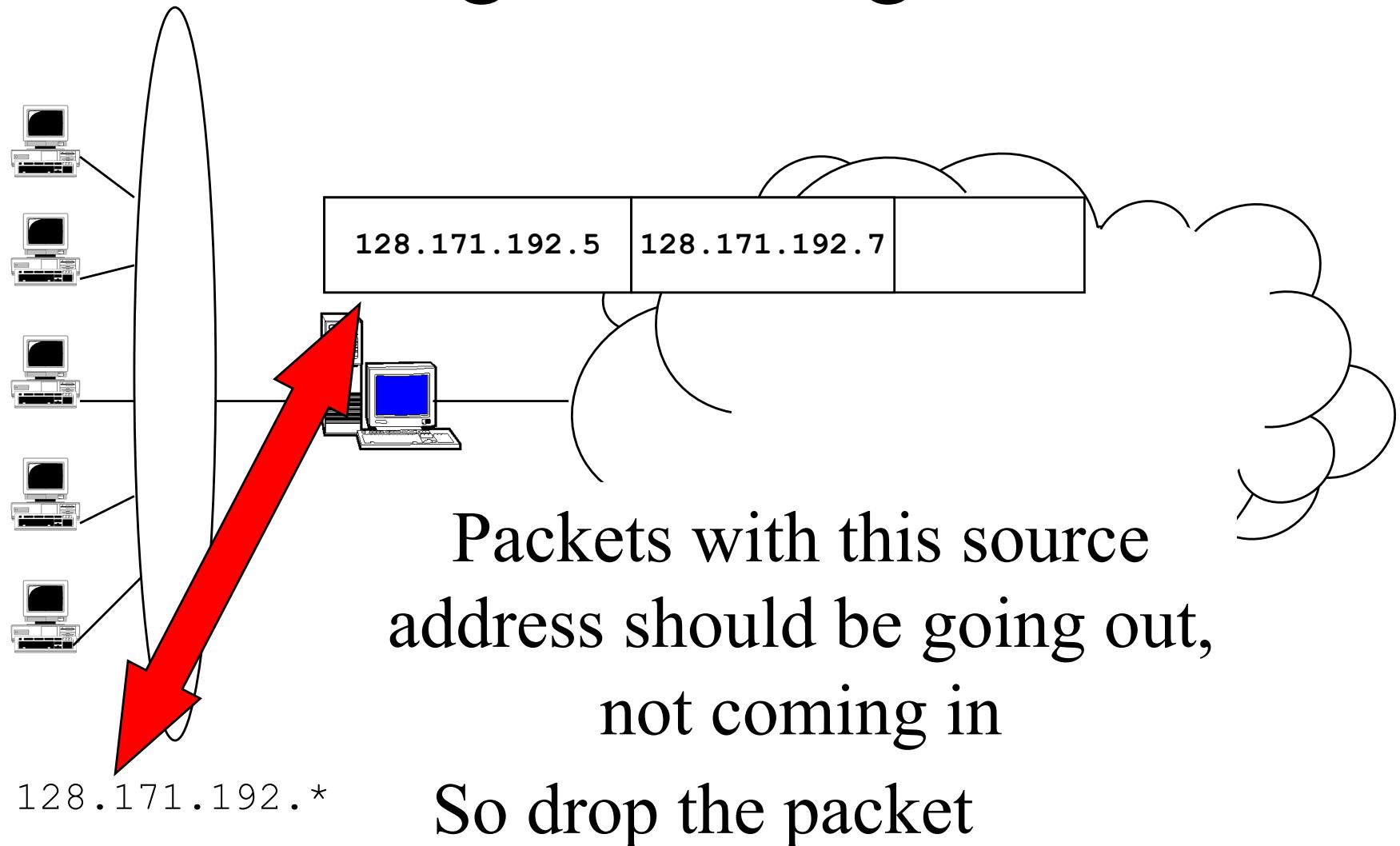
Source Address Filtering Example



Source Address Filtering in the Other Direction

- Often called egress filtering
 - Or ingress filtering . . .
- Occurs as packets leave the Internet and enter a border router
 - On way to that router's network
- What addresses shouldn't be coming into your local network?

Filtering Incoming Packets



Other Forms of Filtering

- One can filter on things other than source address
 - Such as worm signatures, unknown protocol identifiers, etc.
- Also, there are unallocated IP addresses in IPv4 space
 - Can filter for packets going to or coming from those addresses
- Some source addresses for local use only
 - Internet routers can drop packets to/from them

Realistic Limits on Filtering

- Little filtering possible in Internet core
 - Packets being handled too fast
 - Backbone providers don't want to filter
 - Great damage if you screw it up
- Filtering near edges has its own limits
 - In what's possible
 - In what's affordable
 - In what the router owners will do

Another Filtering Possibility

- Redirect packets to a special filtering site on the edge of the network
- They filter on any basis they want
 - Including packet contents
- What they don't drop they send to you
- Many DDoS defense services work this way
- Incurs serious delay penalties

Rate Limits

- Many routers can place limits on the traffic they send to a destination
- Ensuring that the destination isn't overloaded
 - Popular for denial of service defenses
- Limits can be defined somewhat flexibly
- But often not enough flexibility to let the good traffic through and stop the bad

Padding

- Sometimes you don't want intruders to know what your traffic characteristics are
- Padding adds extra traffic to hide the real stuff
- Fake traffic must look like real traffic
 - Usually means encrypt it all
- Must be done carefully, or clever attackers can tell the good stuff from the noise

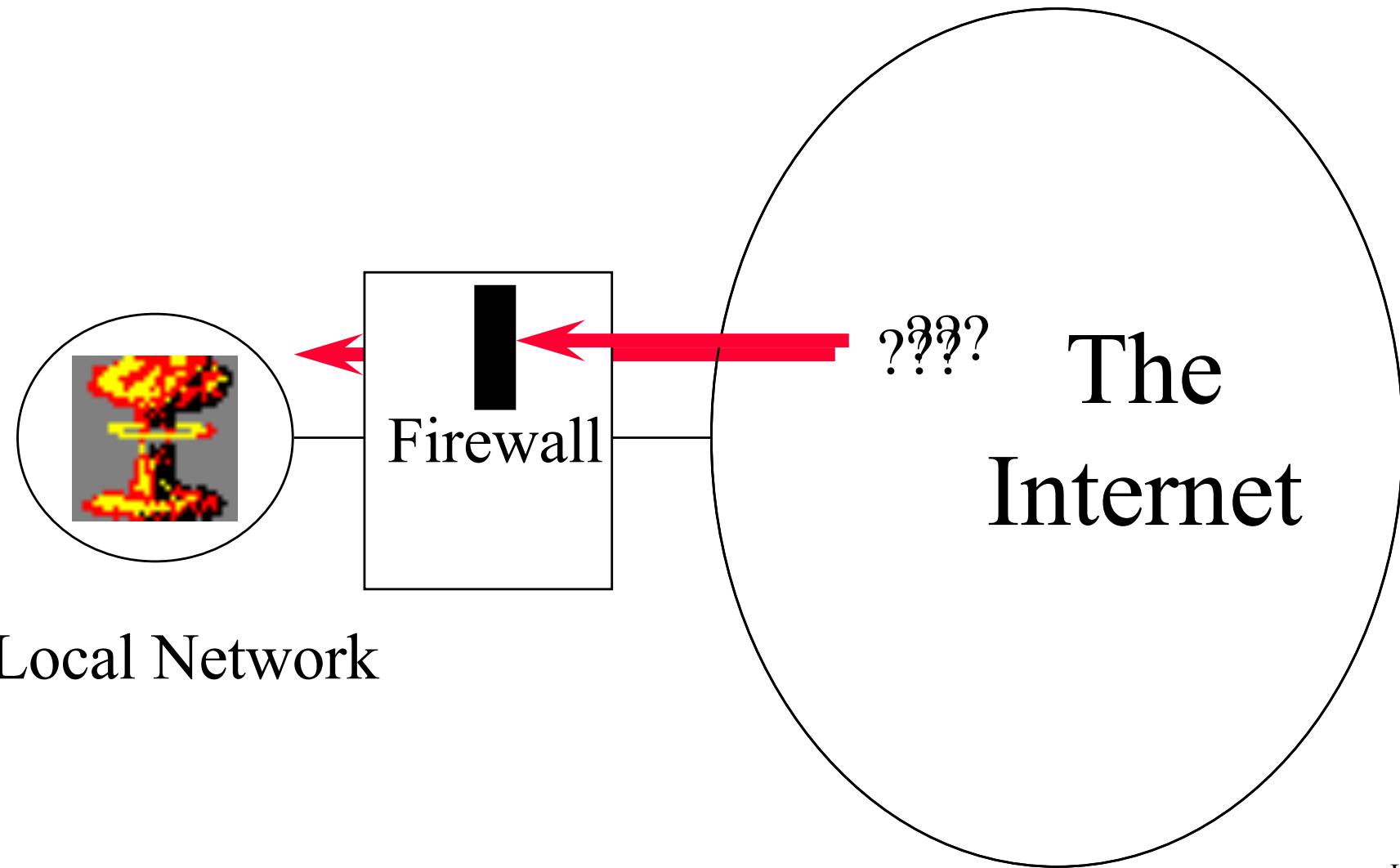
Routing Control

- Use ability to control message routing to conceal the traffic in the network
- Used in *onion routing* to hide who is sending traffic to whom
 - For anonymization purposes
- Routing control also used in some network defense
 - To hide real location of a machine
 - E.g., SOS DDoS defense system

Firewalls

- What is a firewall?
- A machine to protect a network from malicious external attacks
- Typically a machine that sits between a LAN/WAN and the Internet
- Running special software to regulate network traffic

Typical Use of a Firewall



Local Network

The
Internet

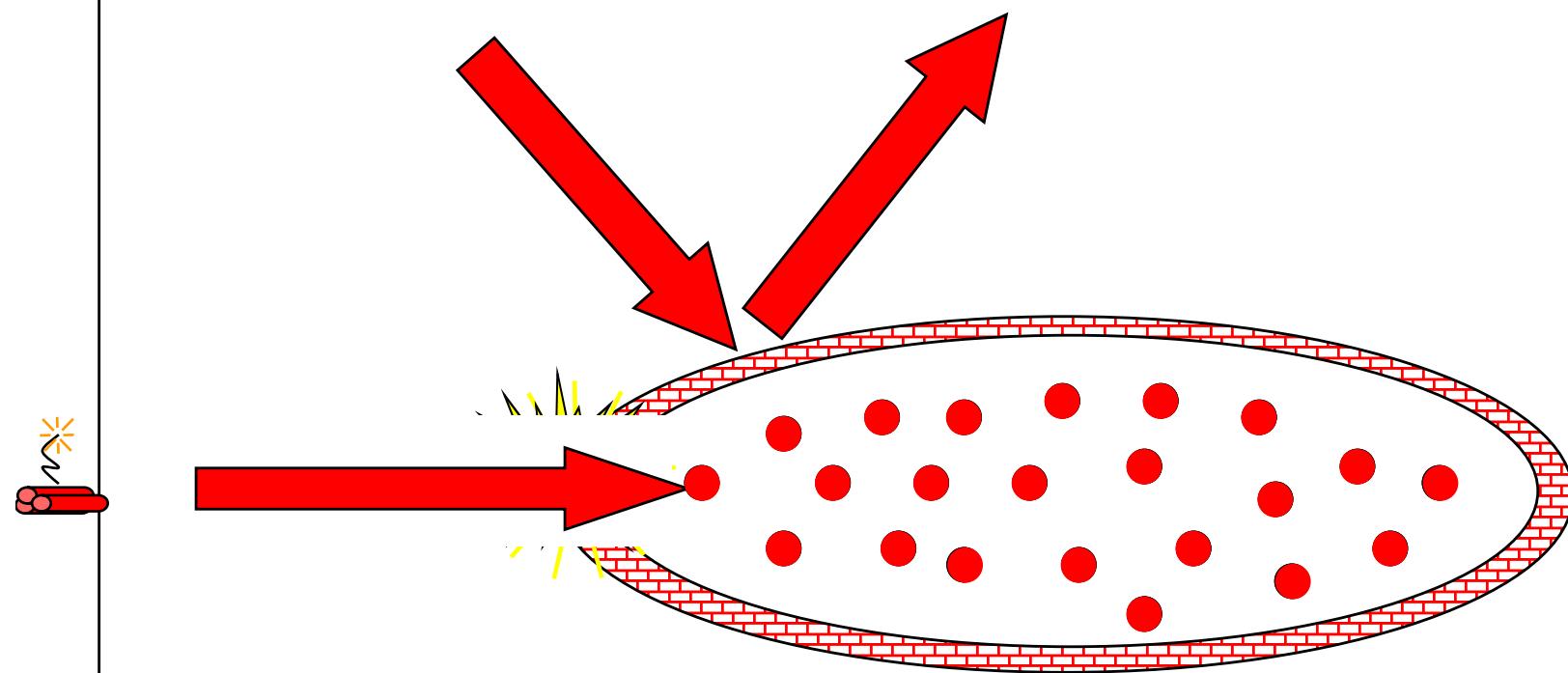
Firewalls and Perimeter Defense

- Firewalls implement a form of security called *perimeter defense*
- Protect the inside of something by defending the outside strongly
 - The firewall machine is often called a *bastion host*
- Control the entry and exit points
- If nothing bad can get in, I'm safe, right?

Weaknesses of Perimeter Defense Models

- Breaching the perimeter compromises all security
- Windows passwords are a form of perimeter defense
 - If you get past the password, you can do anything
- Perimeter defense is part of the solution, not the entire solution

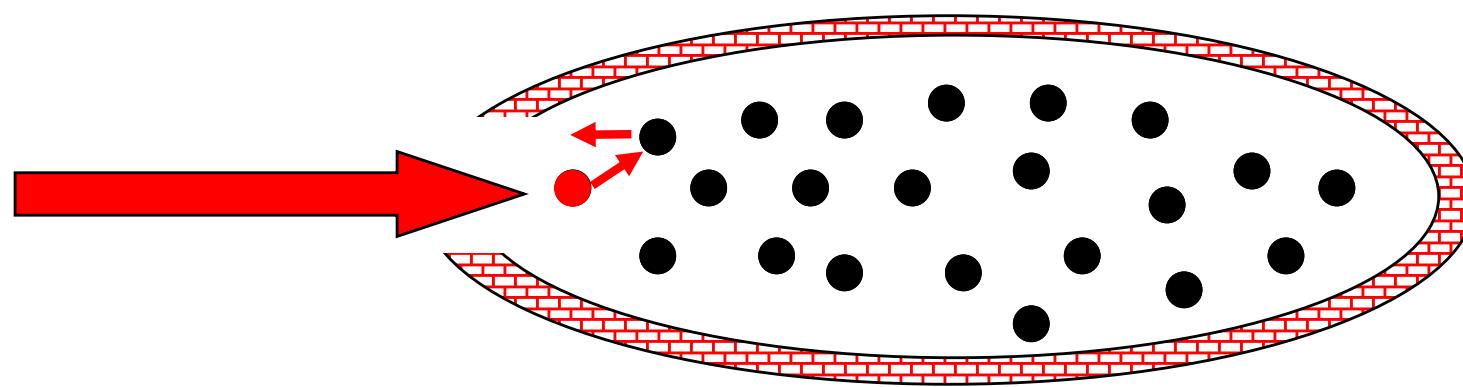
Weaknesses of Perimeter Defense



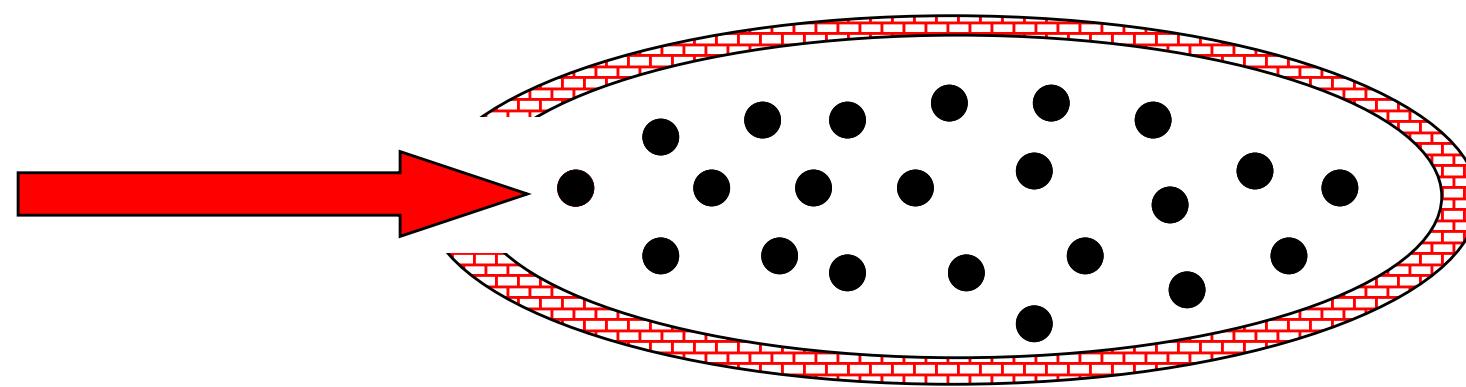
Defense in Depth

- An old principle in warfare
- Don't rely on a single defensive mechanism or defense at a single point
- Combine different defenses
- Defeating one defense doesn't defeat your entire plan

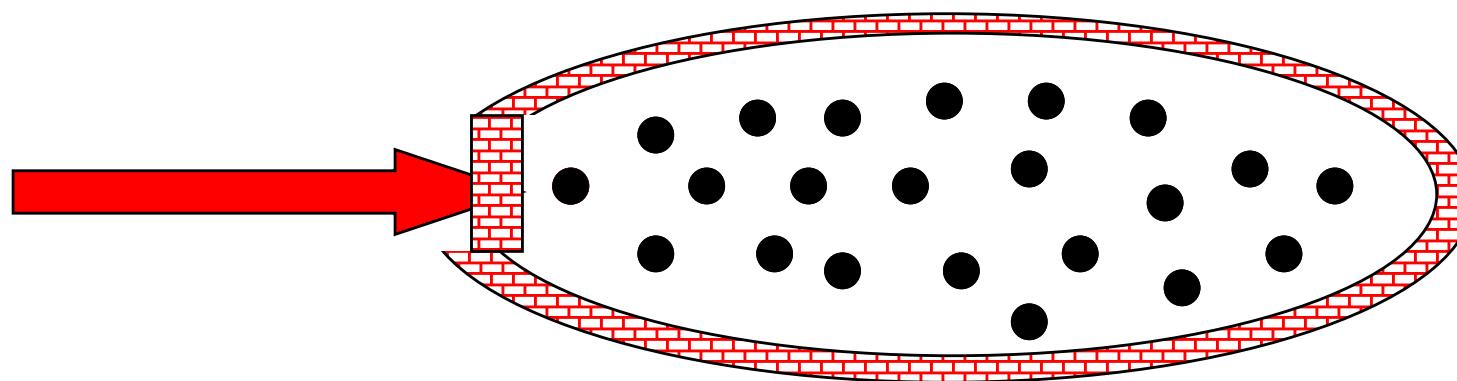
So What Should Happen?



Or, Better



Or, Even Better



So Are Firewalls Any Use?

- Definitely!
- They aren't the full solution, but they are absolutely part of it
- Anyone who cares about security needs to run a decent firewall
- They just have to do other stuff, too

The Brass Tacks of Firewalls

- What do they really do?
- Examine each incoming packet
- Decide to let the packet through or drop it
 - Criteria could be simple or complex
- Perhaps log the decision
- Maybe send rejected packets elsewhere
- Pretty much all there is to it

Types of Firewalls

- Filtering gateways
 - AKA screening routers
- Application level gateways
 - AKA proxy gateways
- Reverse firewalls

Filtering Gateways

- Based on packet header information
 - Primarily, IP addresses, port numbers, and protocol numbers
- Based on that information, either let the packet through or reject it
- *Stateless* firewalls

Example Use of Filtering Gateways

- Allow particular external machines to telnet into specific internal machines
 - Denying telnet to other machines
- Or allow full access to some external machines
- And none to others

A Fundamental Problem

- IP addresses can be spoofed
- If your filtering firewall trusts packet headers, it offers little protection
- Situation may be improved by IPsec
 - But hasn't been yet
- Firewalls can perform the ingress/egress filtering discussed earlier

Filtering Based on Ports

- Most incoming traffic is destined for a particular machine and port
 - Which can be derived from the IP and TCP headers
- Only let through packets to select machines at specific ports
- Makes it impossible to externally exploit flaws in little-used ports
 - If you configure the firewall right . . .

Pros and Cons of Filtering Gateways

- + Fast
- + Cheap
- + Flexible
- + Transparent
- Limited capabilities
- Dependent on header authentication
- Generally poor logging
- May rely on router security

Application Level Gateways

- Also known as proxy gateways
- Firewalls that understand the application-level details of network traffic
 - To some degree
- Traffic is accepted or rejected based on the probable results of accepting it
- *Stateful* firewalls

How Application Level Gateways Work

- The firewall serves as a general framework
- Various proxies are plugged into the framework
- Incoming packets are examined
 - Handed to the appropriate proxy
- Proxy typically accepts or rejects

Deep Packet Inspection

- Another name for typical activity of application level firewalls
- Looking into packets beyond their headers
 - Especially the IP header
- “Deep” sometimes also means deeper understanding of what’s going on
 - Though not always
- It almost always means “expensive”
 - In terms of performance

Firewall Proxies

- Programs capable of understanding particular kinds of traffic
 - E.g., FTP, HTTP, videoconferencing
- Proxies are specialized
- A good proxy has deep understanding of the network application
- Typically limited by complexity and performance issues

Pros and Cons of Application Level Gateways

- + Highly flexible
- + Good logging
- + Content-based filtering
- + Potentially transparent
- Slower
- More complex and expensive
- Highly dependent on proxy quality

Reverse Firewalls

- Normal firewalls keep stuff from the outside from getting inside
- Reverse firewalls keep stuff from the insider from getting outside
- Often colocated with regular firewalls
- Why do we need them?

Possible Uses of Reverse Firewalls

- Concealing details of your network from attackers
- Preventing compromised machines from sending things out
 - E.g., intercepting bot communications or stopping DDoS
 - Preventing data exfiltration

Firewall Characteristics

- Statefulness
- Transparency
- Handling authentication
- Handling encryption

Stateful Firewalls

- Much network traffic is connection-oriented
 - E.g., telnet and videoconferencing
- Proper handling of that traffic requires the firewall to maintain state
- But handling information about connections is more complex

Firewalls and Transparency

- Ideally, the firewall should be invisible
 - Except when it vetoes access
- Users inside should be able to communicate outside without knowing about the firewall
- External users should be able to invoke internal services transparently

Firewalls and Authentication

- Many systems want to give special privileges to specific sites or users
- Firewalls can only support that to the extent that strong authentication is available
 - At the granularity required
- For general use, may not be possible
 - In current systems

Firewalls and Encryption

- Firewalls provide no confidentiality
- Unless the data is encrypted
- But if the data is encrypted, the firewall can't examine it
- So typically the firewall must be able to decrypt
 - Or only work on unencrypted parts of packets
- Can decrypt, analyze, and re-encrypt

Network Security, Continued

Computer Security

Peter Reiher

February 4, 2021

Firewall Configuration and Administration

- Again, the firewall is the point of attack for intruders
- Thus, it must be extraordinarily secure
- How do you achieve that level of security?

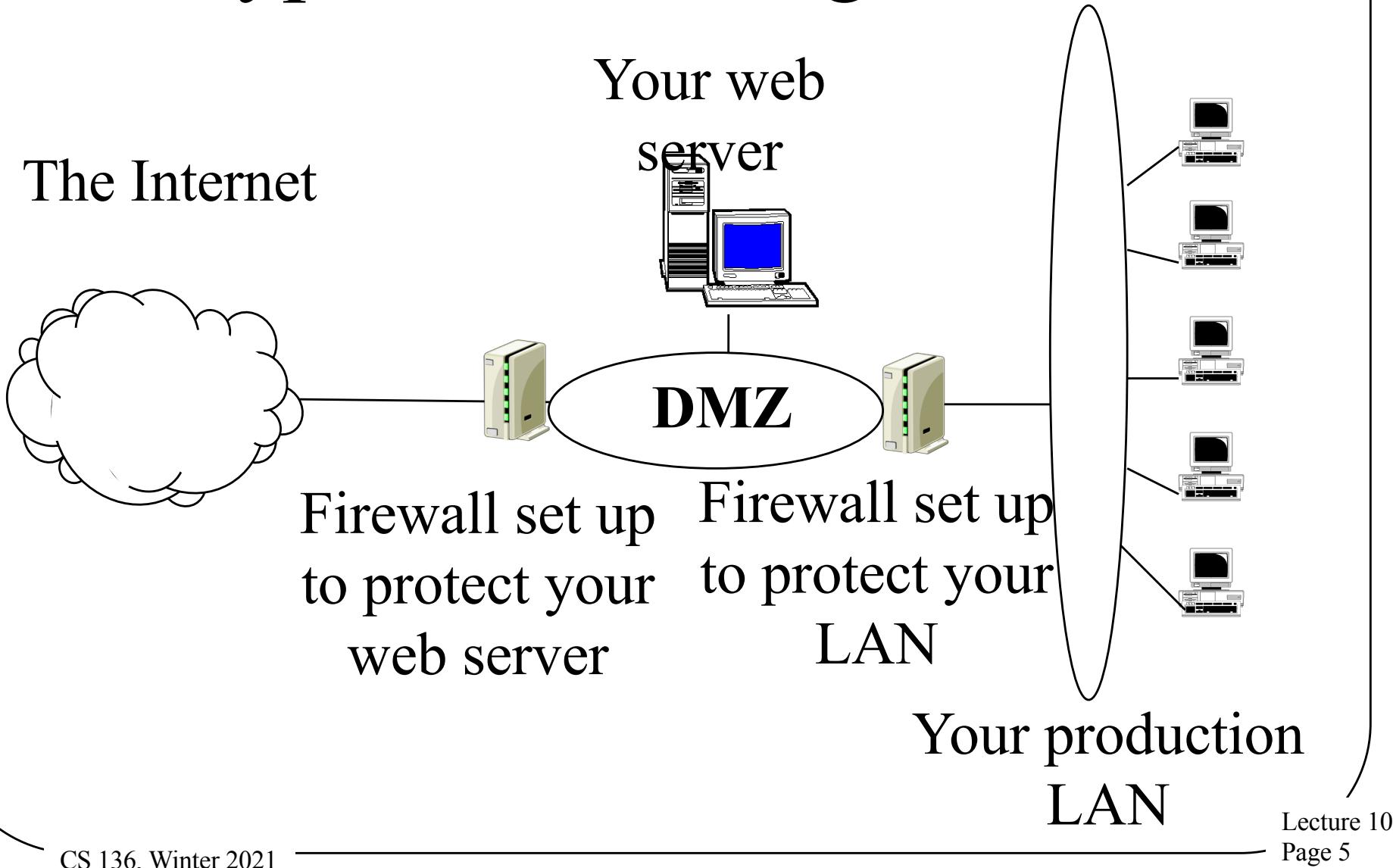
Firewall Location

- Clearly, between you and the bad guys
- But you may have some different types of machines/functionalities
- Sometimes makes sense to divide your network into segments
 - Typically, less secure public network and more secure internal network
 - Using separate firewalls

Firewalls and DMZs

- A standard way to configure multiple firewalls for a single organization
- Used when organization runs machines with different openness needs
 - And security requirements
- Basically, use firewalls to divide your network into segments

A Typical DMZ Organization



Advantages of DMZ Approach

- Can customize firewalls for different purposes
- Can customize traffic analysis in different areas of network
- Keeps inherently less safe traffic away from critical resources

Dangers of a DMZ

- Things in the DMZ aren't well protected
 - If they're compromised, provide a foothold into your network
- One problem in DMZ might compromise all machines there
- Vital that main network doesn't treat machines in DMZ as trusted
- Must avoid back doors from DMZ to network

Firewall Hardening

- Devote a special machine only to firewall duties
- Alter OS operations on that machine
 - To allow only firewall activities
 - And to close known vulnerabilities
- Strictly limit access to the machine
 - Both login and remote execution

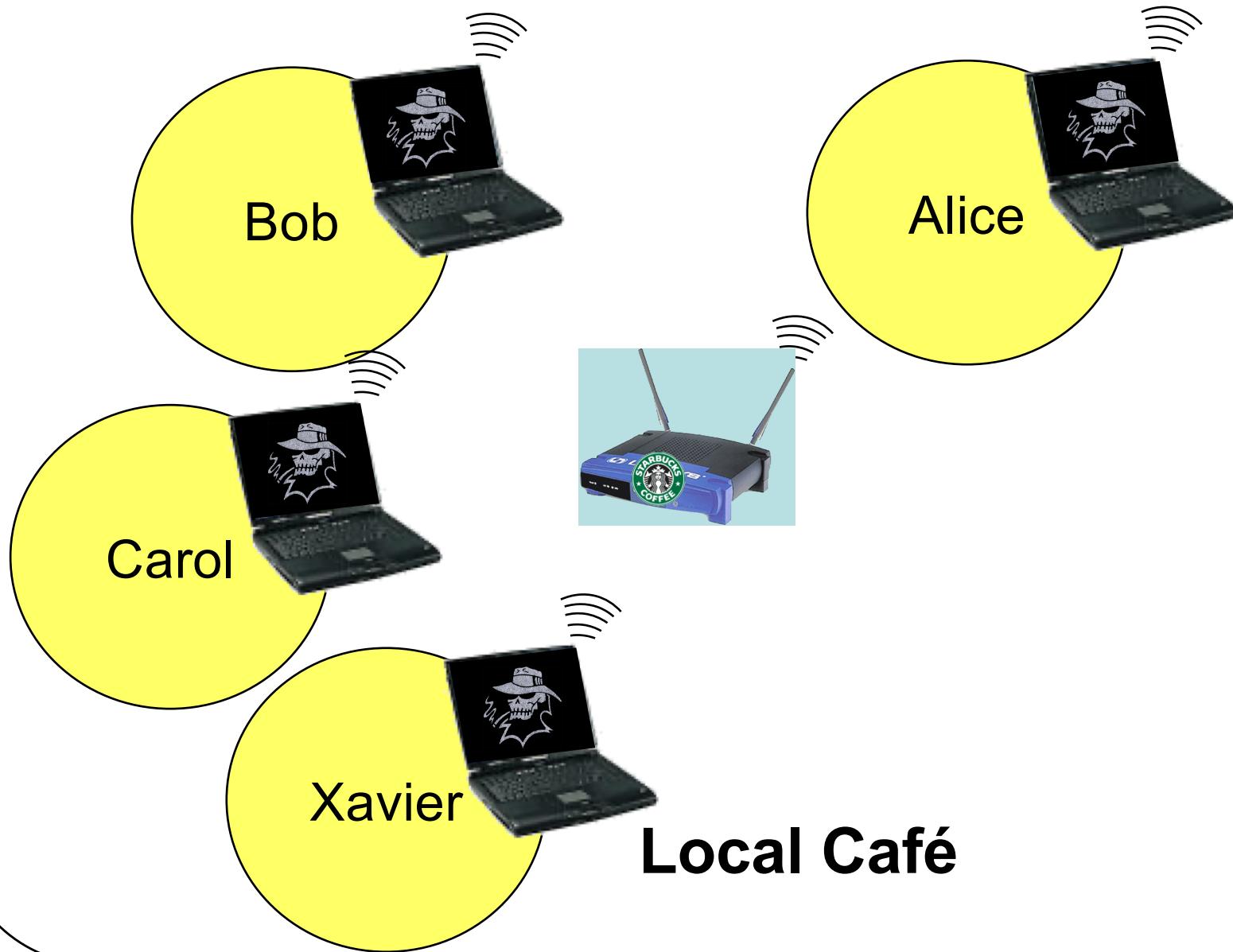
Keep Your Firewall Current

- New vulnerabilities are discovered all the time
- Must update your firewall to fix them
- Even more important, sometimes you have to open doors temporarily
 - Make sure you shut them again later
- Can automate some updates to firewalls
- How about getting rid of old stuff?

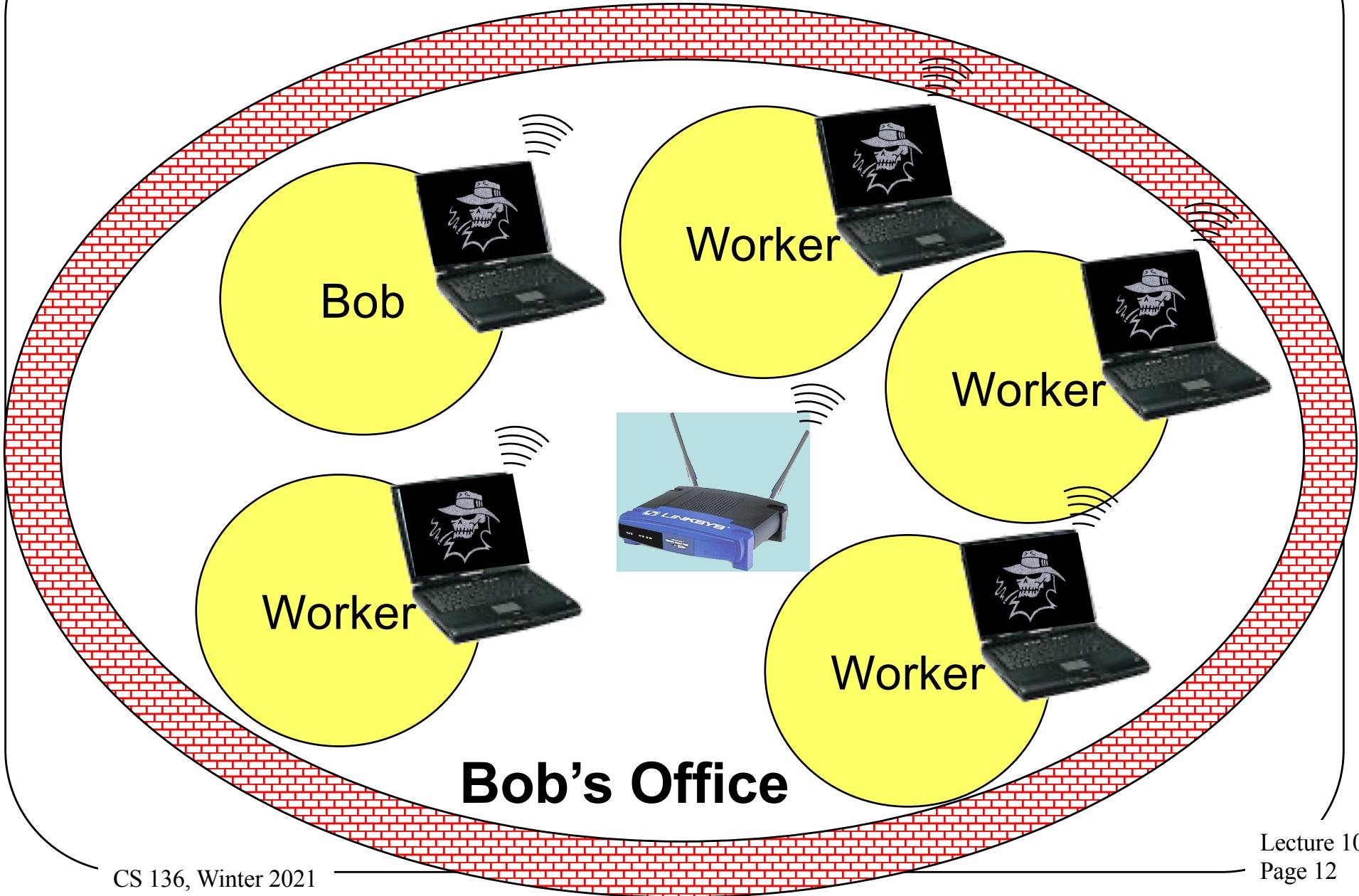
Closing the Back Doors

- Firewall security is based on assumption that all traffic goes through the firewall
- So be careful with:
 - Wireless connections
 - Portable computers
 - Sneakernet mechanisms and other entry points
- Put a firewall at every entry point to your network
- And make sure all your firewalls are up to date

What About Portable Computers?



Now Bob Goes To Work . . .



How To Handle This Problem?

- Essentially *quarantine* the portable computer until it's safe
- Don't permit connection to wireless access point until you're satisfied that the portable is safe
 - Or put them in constrained network
- Common in Cisco, Microsoft, and other companies' products
 - *Network access control*

Single Machine Firewalls

- Instead of separate machine protecting network,
- A machine puts software between the outside world and the rest of machine
- Under its own control
- To protect itself
- Available on most modern systems

Pros of Personal Firewalls

- + Customized to particular machine
 - Specific to local software and usage
- + Under machine owner's control
- + Can use in-machine knowledge for its decisions
- + May be able to do deeper inspection
- + Provides defense in depth

Cons of Personal Firewalls

- Only protects that machine
- Less likely to be properly configured
 - Since most users don't understand security well
 - And/or don't view it as their job
 - Probably set to the default
- On the whole, generally viewed as valuable

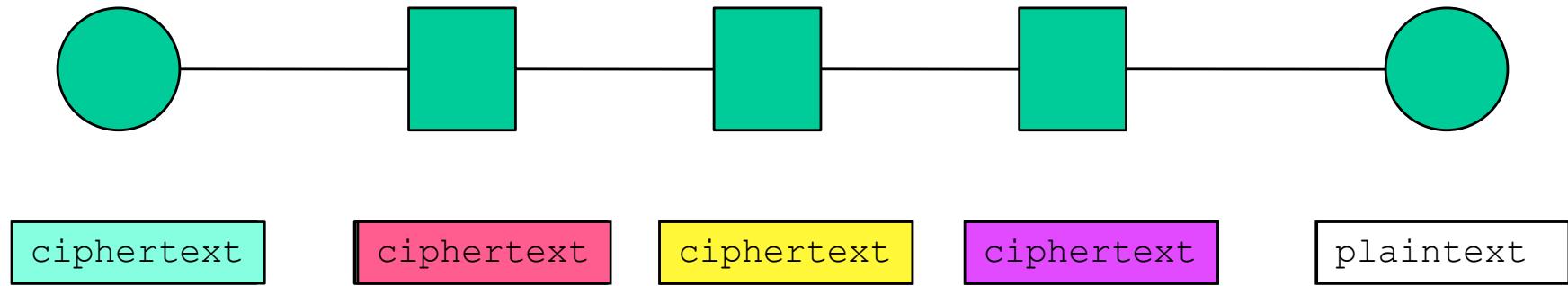
Encryption and Network Security

- Cryptography is widely used to protect networks
- Relies on encryption algorithms and protocols discussed previously
- Can be applied at different places in the network stack
- With different effects and costs

Link Level Encryption

Source

Destination

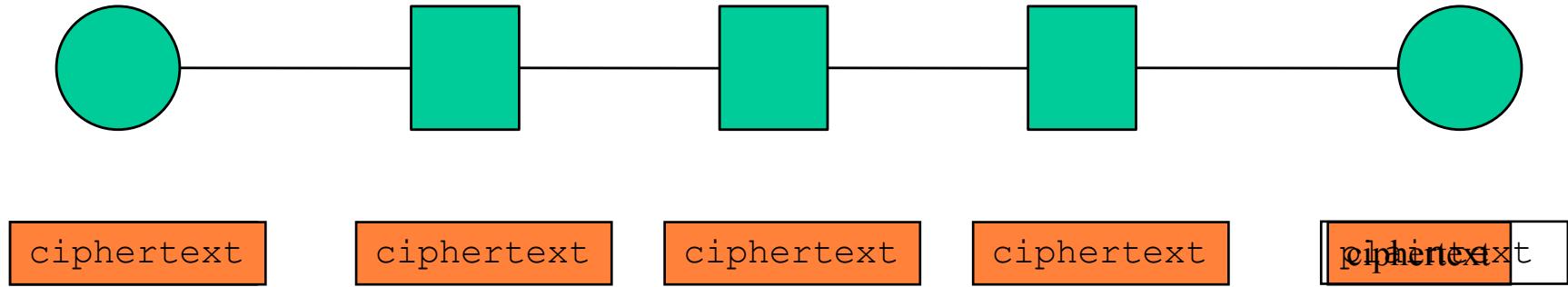


Let's say we want to send a message using encryption
Different keys (maybe even different ciphers) used at
each hop

End-to-End Encryption

Source

Destination



Cryptography only at the end points

Only the end points see the plaintext

Normal way network cryptography done

When
would link
encryption
be better?

Where Are the Endpoints, Anyway?

- If you do end-to-end encryption, where are the endpoints?
- The network layer end points?
- The transport layer end points?
- The application layer end points?
- Maybe not even end machine to end machine (e.g., VPNs)?
- Has serious implications for where you do cryptography
 - And keying and trust issues

IPsec

- A standard for applying cryptography at the network layer of IP stack
- Provides various options for encrypting and authenticating packets
 - On end-to-end basis
 - Without concern for transport layer (or higher)

What IPsec Covers

- Message integrity
- Message authentication
- Message confidentiality

What Isn't Covered

- Non-repudiation
- Digital signatures
- Key distribution
- Traffic analysis
- Handling of security associations
- Some of these covered in related standards

Some Important Terms for IPsec

- Security Association - “A Security Association (SA) is a simplex ‘connection’ that affords security services to the traffic carried by it.”
 - Basically, a secure one-way channel
- SPI (Security Parameters Index) – Combined with destination IP address and IPsec protocol type, uniquely identifies an SA

General Structure of IPsec

- Really designed for end-to-end encryption
 - Though could do link level
- Designed to operate with either IPv4 or IPv6
- Meant to operate with a variety of different ciphers
- And to be neutral to key distribution methods
- Has sub-protocols
 - E.g., Encapsulating Security Payload

Encapsulating Security Payload (ESP) Protocol

- Encrypt the data and place it within the ESP
- The ESP has normal IP headers
- Can be used to encrypt just the payload of the packet
- Or the entire IP packet

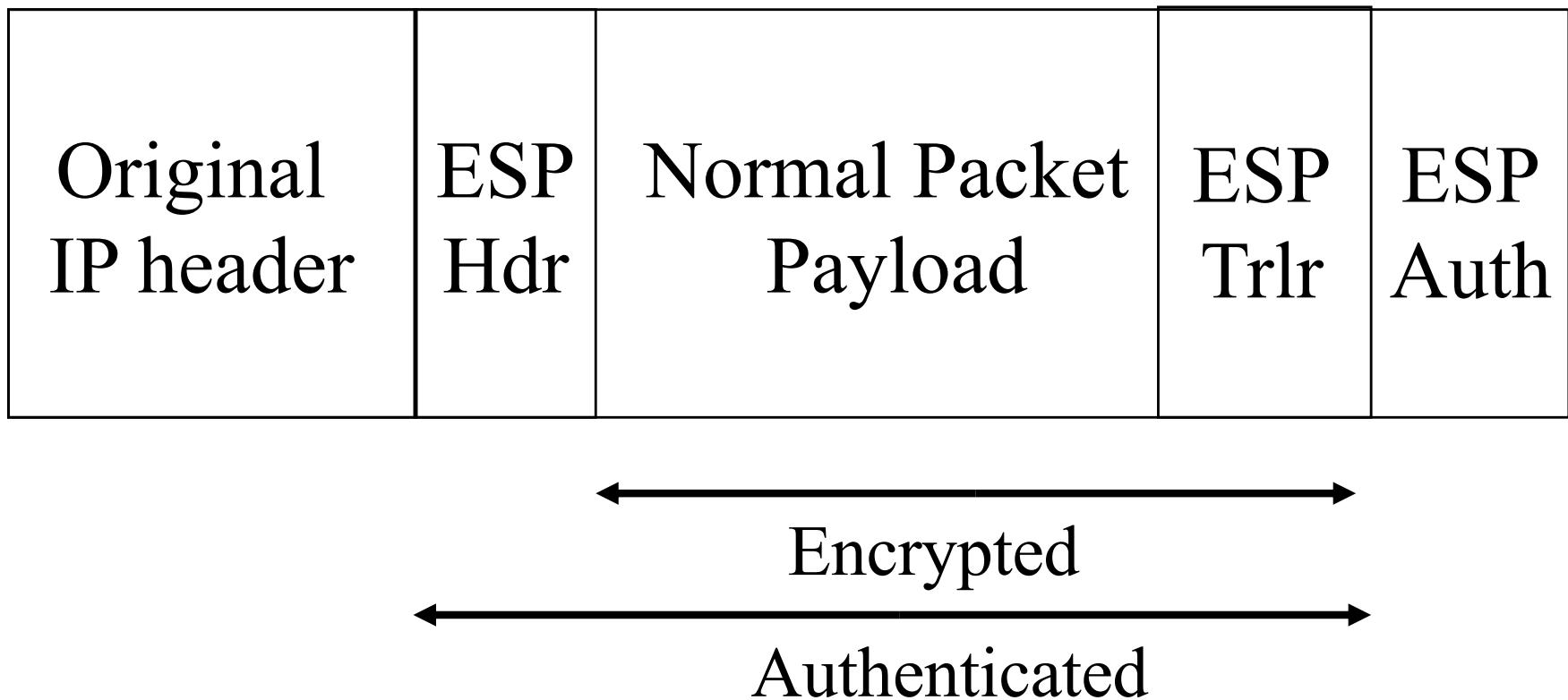
ESP Modes

- Transport mode
 - Encrypt just the transport-level data in the original packet
 - No IP headers encrypted
- Tunnel mode
 - Original IP datagram is encrypted and placed in ESP
 - Unencrypted headers wrapped around ESP

ESP in Transport Mode

- Extract the transport-layer frame
 - E.g., TCP, UDP, etc.
- Encapsulate it in an ESP
- Encrypt it
- The encrypted data is now the last payload of a cleartext IP datagram

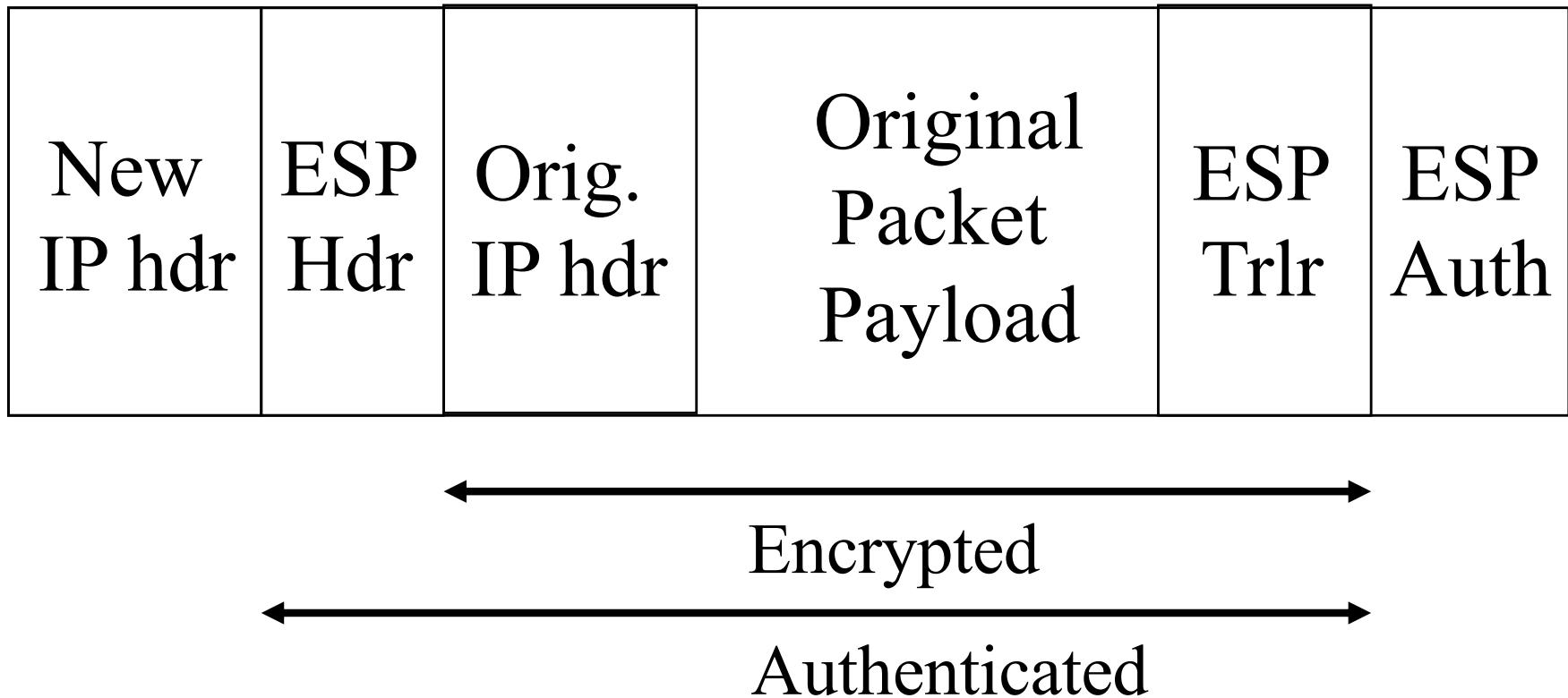
ESP Transport Mode



Using ESP in Tunnel Mode

- Encrypt the IP datagram
 - The entire datagram
- Encapsulate it in a cleartext IP datagram
- Routers not understanding IPsec can still handle it
- Receiver reverses the process

ESP Tunnel Mode



Uses and Implications of Tunnel Mode

- Typically used when there are security gateways between sender and receiver
 - And/or sender and receiver don't speak IPsec
- Outer header shows security gateway identities
 - Not identities of real parties
- Can thus be used to hide some traffic patterns

What IPsec Requires

- Protocol standards
 - To allow messages to move securely between nodes
- Supporting mechanisms at hosts running IPsec
 - E.g., a Security Association Database
- Lots of plug-in stuff to do the cryptographic heavy lifting

The Protocol Components

- Pretty simple
- Necessary to interoperate with non-IPsec equipment
- So everything important is inside an individual IP packet's payload
- No inter-message components to protocol
 - Though some security modes enforce inter-message invariants at endpoints

The Supporting Mechanisms

- Methods of defining security associations
- Databases for keeping track of what's going on with other IPsec nodes
 - To know what processing to apply to outgoing packets
 - To know what processing to apply to incoming packets

Plug-In Mechanisms

- Designed for high degree of generality
- So easy to plug in:
 - Different crypto algorithms
 - Different hashing/signature schemes
 - Different key management mechanisms

Status of IPsec

- Accepted Internet standard
- Widely implemented and used
 - Supported in Windows 2000, XP, Vista, Windows 7 and later
 - In Linux 2.6 (and later) kernel
- The architecture doesn't require everyone to use it
- RFC 3602 on using AES in IPsec still listed as “proposed”
- AES will become default for ESP in IPsec

SSL and TLS

- SSL – Secure Socket Layer
- TLS – Transport Layer Security
 - A replacement for SSL
- The common standards for securing network applications in Internet
 - E.g., web browsing
- Essentially, standards to negotiate, set up, and apply crypto

The Basics of SSL

- Usually a client/server operation
- Client contacts server
- A negotiation over authentication, key exchange, and cipher takes place
- Authentication is performed and key agreed upon
- Then all packets are encrypted with that key and cipher at application level

Common Use

- Server authenticates to client using an X.509 certificate
 - Typically, client not authenticated
 - Though option allows it
- Client provides material to server to derive session key
- Client and server derive same session key, start sending encrypted packets

Crypto in TLS/SSL

- Several options supported
- RSA or elliptic curve for PK part
- AES, DES, 3DES, or others for session cryptography
- Not all are regarded as still secure
- Chosen by negotiation between client and server

Use of SSL/TLS

- The core crypto for web traffic
- Commonly used for many other encrypted communications
- Used in all major browsers
- Usually not part of OS per se
 - But all major OSes include libraries or packages that implement it

Security Status of SSL/TLS

- Kind of complex
- SSL is not very secure
- Early versions of TLS not so secure
- Later versions of TLS fairly secure
 - Depending on cipher choice
- Recent chosen-plaintext attacks shown to work on all versions before TLS 1.3
 - In special circumstances

SSL/TLS and the Problem of Generality

- SSL and later TLS were designed to allow use of many options
- Including many different ciphers
- Some of which became insecure
 - Subject to brute force attacks
 - Or a cipher vulnerability
- In some situations, attackers could force use of these weak ciphers

TLS 1.3

- Latest version of TLS
- Standard released in August 2018
 - Implementations somewhat later
 - Available in Chrome, Firefox
- Removed a bunch of less secure ciphers from the suite
 - So only secure crypto would be used

IPSec vs. TLS

- IPSec works “between” the network and transport layers
 - Secures packets, not connections
 - Usable with any transport
- TLS is above the transport layer
 - Or inside it, depending on point of view
 - Secures connections, not just packets
 - Inherently based on TCP, not available for UDP

Virtual Private Networks

- VPNs
- What if your company has more than one office?
- And they're far apart?
 - Like on opposite coasts of the US
- How can you have secure cooperation between them?
- Could use leased lines, but . . .

Encryption and Virtual Private Networks

- Use encryption to convert a shared line to a “private line”
- Set up a firewall at each installation’s network
- Set up shared encryption keys between the firewalls
- Encrypt all traffic using those keys

Actual Use of Encryption in VPNs

- VPNs run over the Internet
- Internet routers can't handle fully encrypted packets
- Obviously, VPN packets aren't entirely encrypted
- They are encrypted in a tunnel mode
 - Often using IPSec or TLS
- Gives owners flexibility and control

Key Management and VPNs

- All security of the VPN relies on key secrecy
- How do you communicate the key?
 - In early implementations, manually
 - Modern VPNs use IKE or proprietary key servers
- How often do you change the key?
 - IKE allows frequent changes

VPNs and Firewalls

- VPN encryption is typically done between firewall machines
 - VPN often integrated into firewall product
- Do I need the firewall for anything else?
- Probably, since I still need to allow non-VPN traffic in and out
- Need firewall “inside” VPN
 - Since VPN traffic encrypted
 - Including stuff like IP addresses and ports
 - “Inside” can mean “later in same box”

VPNs and Portable Computing

- Increasingly, workers connect to offices remotely
 - While on travel
 - Or when working from home
- VPNs offer a secure solution
 - Typically as software in the portable computer
- Usually needs to be pre-configured

VPN Deployment Issues

- Desirable not to have to pre-deploy VPN software
 - Clients get access from any machine
- Possible by using downloaded code
 - Connect to server, download VPN applet, away you go
 - Often done via web browser
 - Leveraging existing SSL code
 - Authentication via user ID/password
 - Implies you trust the applet . . .
- Issue of compromised user machine

Wireless Network Security

- Wireless networks are “just like” other networks
- Except . . .
 - Almost always broadcast
 - Generally short range
 - Usually supporting mobility
 - Often very open

Types of Wireless Networks

- 802.11 networks
 - Variants on local area network technologies
- Bluetooth networks
 - Very short range
- Cellular telephone networks
- Line-of-sight networks
 - Dedicated, for relatively long hauls
- Satellite networks

The General Solution For Wireless Security

- Wireless networks inherently less secure than wired ones
 - So we need to add extra security
 - How to do it?
 - Link encryption
 - Encrypt traffic just as it crosses the wireless network
- Decrypt it before sending it along

Why Not End-to-End Encryption?

- Some non-wireless destinations might not be prepared to perform crypto
 - What if wireless user wants protection anyway?
- Doesn't help wireless access point provide exclusive access
 - Any eavesdropper can use network

802.11 Security

- Originally, 802.11 protocols didn't include security
- Once the need became clear, it was sort of too late
 - Huge number of units in the field
 - Couldn't change the protocols
- So, what to do?

WEP

- First solution to the 802.11 security problem
- Wired Equivalency Protocol
- Intended to provide encryption in 802.11 networks
 - Without changing the protocol
 - So all existing hardware just worked
- The backward compatibility worked
- The security didn't

What Did WEP Do?

- Used stream cipher (RC4) for confidentiality
 - With 104 bit keys
 - Usually stored on the computer using the wireless network
 - 24 bit IV also used
- Used checksum for integrity

What Was the Problem With WEP?

- Access point generates session key from its own permanent key plus IV
 - Making replays and key deduction attacks a problem
- IV was intended to prevent that
- But it was too short and used improperly
- In 2001, WEP cracking method shown
 - Took less than 1 minute to get key

WPA, WPA2, and WPA3

- Generates new key for each session
- Serious security flaws in WPA required creation of WPA2
- Serious security flaw in WPA2 required creation of WPA3
- And there's a serious flaw in WPA3
- Bottom line: use WPA3 if available

Honeypots and Honeynets

- A *honeypot* is a machine set up to attract attackers
- Classic use is to learn more about attackers
- Ongoing research on using honeypots as part of a system's defenses

Setting Up A Honeypot

- Usually a machine dedicated to this purpose
- Probably easier to find and compromise than your real machines
- But has lots of software watching what's happening on it
- Providing early warning of attacks

What Have Honeypots Been Used For?

- To study attackers' common practices
- There are lengthy traces of what attackers do when they compromise a honeypot machine
- Not clear these traces actually provided much we didn't already know

Honeynets

- A collection of honeypots on a single network
 - Maybe on a single machine with multiple addresses
 - More often using virtualization
- Typically, no other machines are on the network
- Since whole network is phony, all incoming traffic is probably attack traffic

What Can You Do With Honeynets?

- Similar things to honeypots
 - But at the network level
- Also good for tracking the spread of worms
 - Worm code typically visits them repeatedly
- Main tool for detecting and analyzing botnets
- Gives evidence of DDoS attacks
 - Through *backscatter*
 - Based on attacker using IP spoofing

Honeynets and Botnets

- Honeynets widely used by security researchers to “capture” bots
- Honeynet is reachable from Internet
- Intentionally weakly defended
- Bots tend to compromise them
- Researcher gets a copy of the bot
- Which they analyze for various purposes

Do You Need A Honeypot?

- Not in the same way you need a firewall
- Only useful if your security administrator spending a lot of time watching things
 - E.g., very large enterprises
- Or if your job is observing hacker activity
- Something that someone needs to be doing
 - Particularly, security experts watching the overall state of the network world
 - But not necessarily you