CS 333
Introduction to Operating Systems

Class 19 - Security

Jonathan Walpole
Computer Science
Portland State University

Overview

- **Different aspects of security**
- **User authentication**
- **Protection mechanisms**
- **Attacks**
  - trojan horses, spoofing, logic bombs, trap doors, buffer overflow attacks, viruses, worms, mobile code, sand boxing
- **Brief intro to cryptography tools**
  - one-way functions, public vs private key encryption, hash functions, and digital signatures

Security overview

- **Security flavors**
  - Confidentiality - protecting secrets
  - Integrity - preventing data contents from being changed
  - Availability - ensuring continuous operation
- **Know thine enemy!**
  - User stupidity (bad default settings from companies)
  - Insider snooping
  - Outsider snooping
  - Attacks (viruses, worms, denial of service)
  - Bots

Accidental data loss

Distinguishing security from reliability:

- **Acts of God**
  - fires, floods, wars
- **Hardware or software errors**
  - CPU malfunction, bad disk, program bugs
- **Human errors**
  - data entry, wrong tape mounted
  - "you" are probably the biggest threat you'll ever face!

User Authentication

User authentication

- **Must be done before the user can use the system !**
  - Subsequent activities are associated with this user
    - **Fork process**
    - **Execute program**
    - **Read file**
    - **Write file**
    - **Send message**
- **Authentication must identify:**
  - Something the user knows
  - Something the user has
  - Something the user is

Authentication using passwords

User name: something the user knows
Password: something the user knows
How easy are they you guess (crack)?

```
LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN
```

(a)

```
LOGIN: carol
INVALID LOGIN NAME
LOGIN:
```

(b)

```
LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:
```

(c)

(a) A successful login
(b) Login rejected after name entered (easier to crack)
(c) Login rejected after name and password typed (larger search space!)

Problems with pre-set values

- **Pre-set user accounts and default passwords are easy to guess**

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

Storing passwords

- **The system must store passwords in order to perform authentication**
- **How can passwords be protected?**
  - Rely on file protection
    - **store them in protected files**
    - **compare typed password with stored password**
  - Rely on encryption
    - **store them encrypted**
      - **use one way function (cryptographic hash)**
    - **can store encrypted passwords in readable files**

Password management in Unix

- **Password file - /etc/passwd**
  - It's a world readable file!
- **/etc/passwd entries**
  - User name
  - Password (encrypted)
  - User id
  - Group id
  - Home directory
  - Shell
  - Real name
  - …

Dictionary attacks

- **If encrypted passwords are stored in world readable files and you see that another user's encrypted password is the same as yours**
  - Their password is also the same!
- **If the encryption method is well known, attackers can:**
  - Encrypt an entire dictionary
  - Compare encrypted dictionary words with encrypted passwords until they find a match

Salting passwords

- The salt is a number combined with the password prior to encryption
- The salt changes when the password changes
- The salt is stored with the password
- Different user's with the same password see different encrypted values in /etc/passwd
- Dictionary attack requires time-consuming re-encoding of entire dictionary for every salt value

Attacking password-based authentication

- **Guessing at the login prompt**
  - Time consuming
  - Only catches poorly chosen passwords
  - If the search space if large enough, manual guessing doesn't work
- **Automated guessing**
  - Requires dictionary to identify relevant portion of large search space
  - Only catches users whose password is a dictionary word, or a simple derivative of a dictionary word
  - But a random combination of characters in a long string is hard to remember!
    - **If users store it somewhere it can be seen by others**

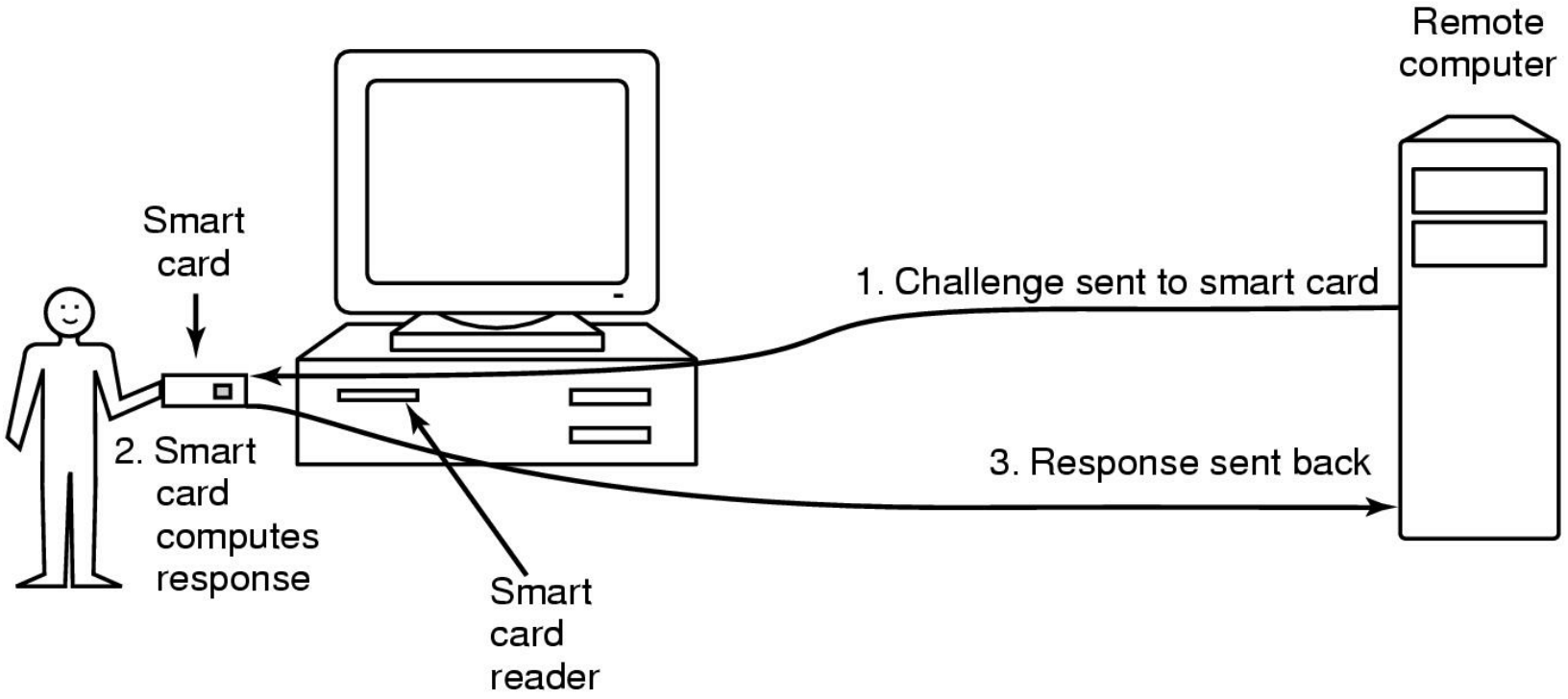
More attacks …

- **Viewing of passwords kept in the clear**
  - Written on desk, included in a network packet etc…
- **Network packet sniffers**
  - Listen to the network and record login sessions
- **Snooping**
  - observing key strokes

General counter-measures

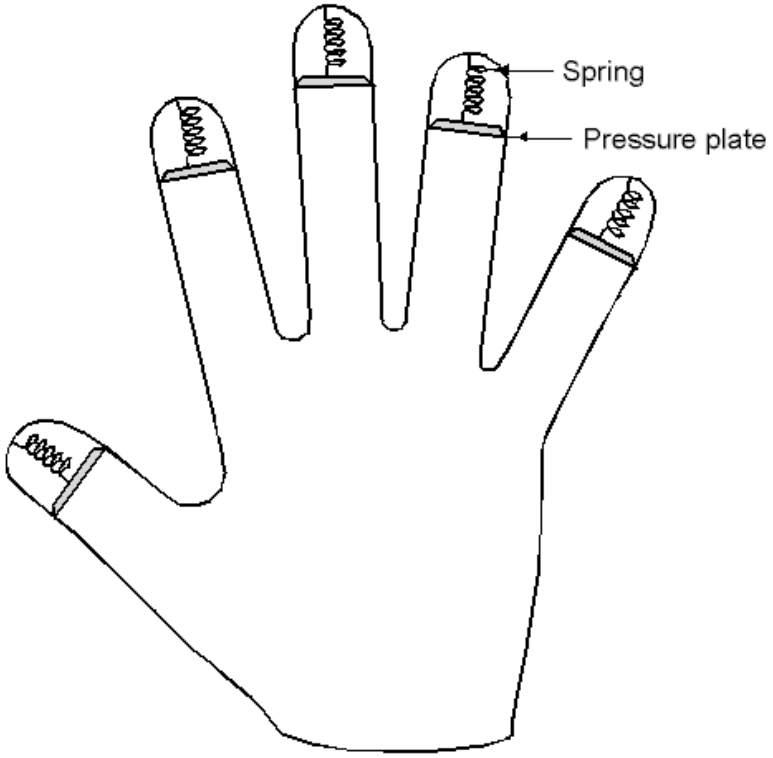
- **Better passwords**
  - No dictionary words, special characters, longer
- **Don't give up information**
  - Login prompts or any other time
- **One time passwords**
  - Satellite driven security cards
- **Limited-time passwords**
  - Annoying but effective
- **Challenge-response pairs**
  - Ask questions
- **Physical authentication combined with passwords**
  - Perhaps combined with challenge response too

Authentication using a physical object



- **Magnetic cards**
  - magnetic stripe cards
  - chip cards: stored value cards, smart cards

Authentication using biometrics

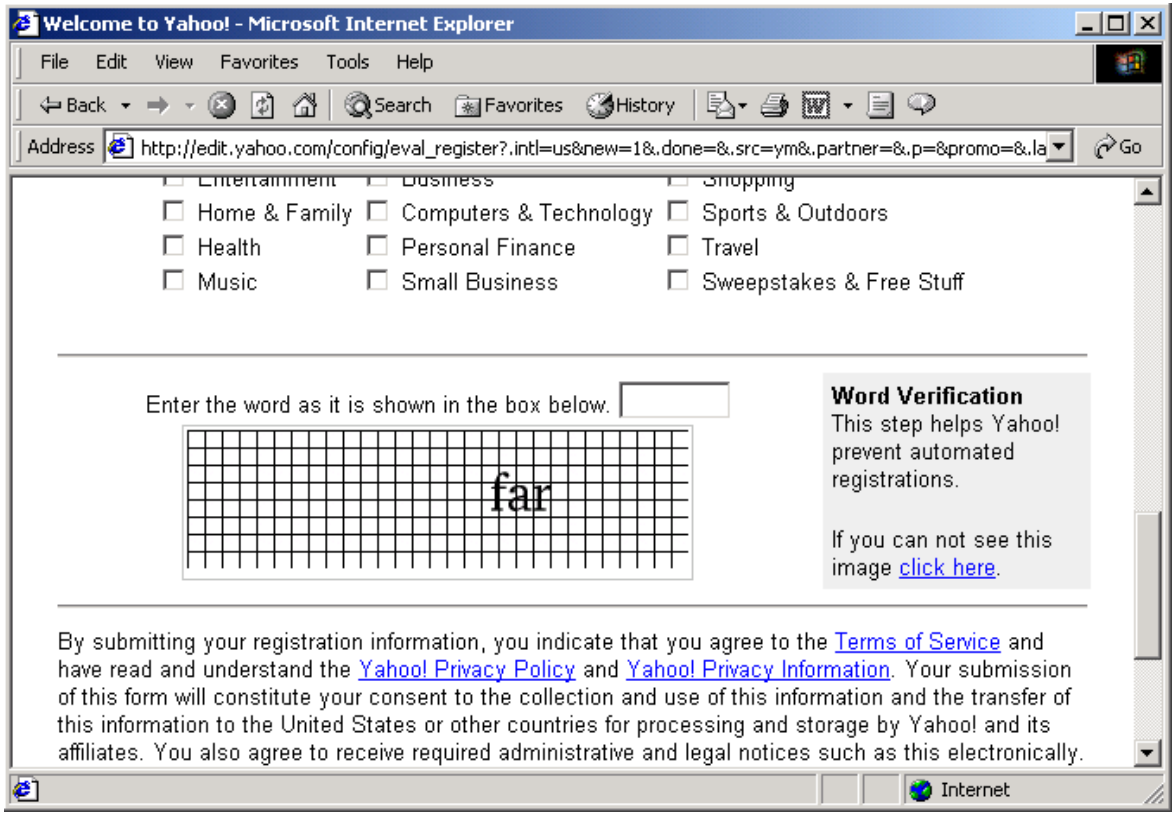


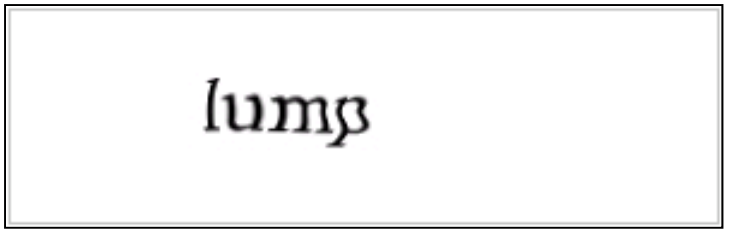
A device for measuring finger length.

More counter-measures

- **Limiting times when someone can log in**
- **Automatic callback at a pre-specified number**
- **Limited number or frequency of login tries**
- **Keep a database of all logins**
- **Honey pot**
  - leave simple login name/password as a trap
  - security personnel notified when attacker bites

Verifying the user is a human!





Protection Domains

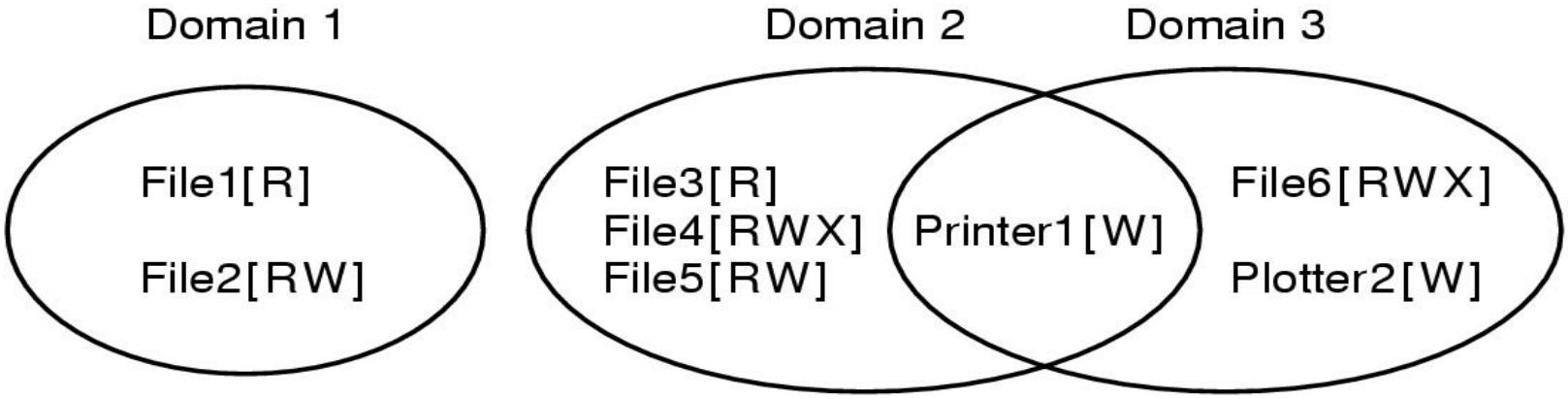
Protection domains

- **Suppose that we have successfully authenticated the user, now what?**
  - For each process created we can keep track of who it belongs to
    - **All its activities are on behalf of this user**
  - We can check all of its accesses to resources
    - **Files, memory, devices …**

Real vs effective user ids

- **We may need mechanisms for temporarily allowing access to privileged resources in a controlled way**
  - Give user a temporary "effective user id" for the execution of a specific program
  - Similar concept to system calls that allow the OS to perform privileged operations on behalf of a user
  - A program (executable file) may have setuid root privilege associated with it
    - **When executed by a user, that user's effective id is temporarily raised to root privilege**

Protection domain model



- **Every process executes in some protection domain**
  - determined by its creator, authenticated at login time
- **OS mechanisms for switching protection domains**
  - system calls
  - set UID capability on executable file
  - re-authenticating user (su)

A protection matrix

Domain	Object							
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

A protection matrix specifies the operations that are allowable on objects by a process executing in a domain.

Protection matrix with domains as objects

Domain	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Operations may include switching to other domains

Protection domains

- **A protection matrix is just an abstract representation for allowable operations**
  - We need protection "mechanisms" to enforce the rules defined by a set of protection domains

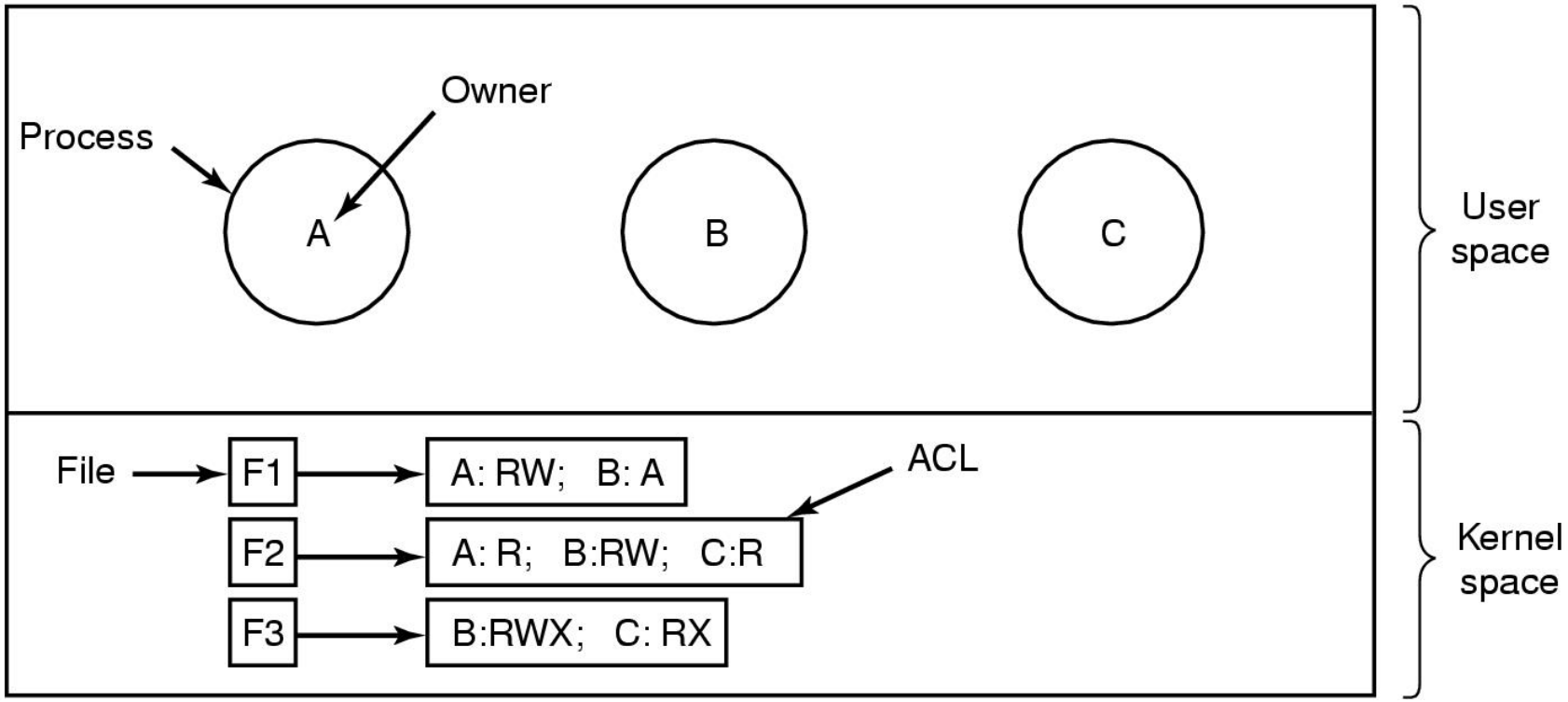
Protection Mechanisms

Access control lists (ACLs) – matrix by column

Domain	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

- **Domain matrix is typically large and sparse**
  - inefficient to store the whole thing
  - store occupied columns only, with the resource? - ACLs
  - store occupied rows only, with the domain? - Capabilities

Access control lists for file access



Example:

User's ID stored in PCB

Access permissions stored in inodes

Access Control Lists – Users vs Roles

File	Access control list
Password	tana, sysadm: RW
Pigeon_data	bill, pigfan: RW; tana, pigfan: RW; ...

- **Two access control lists with user names and roles (groups)**

Compact representation of ACLs

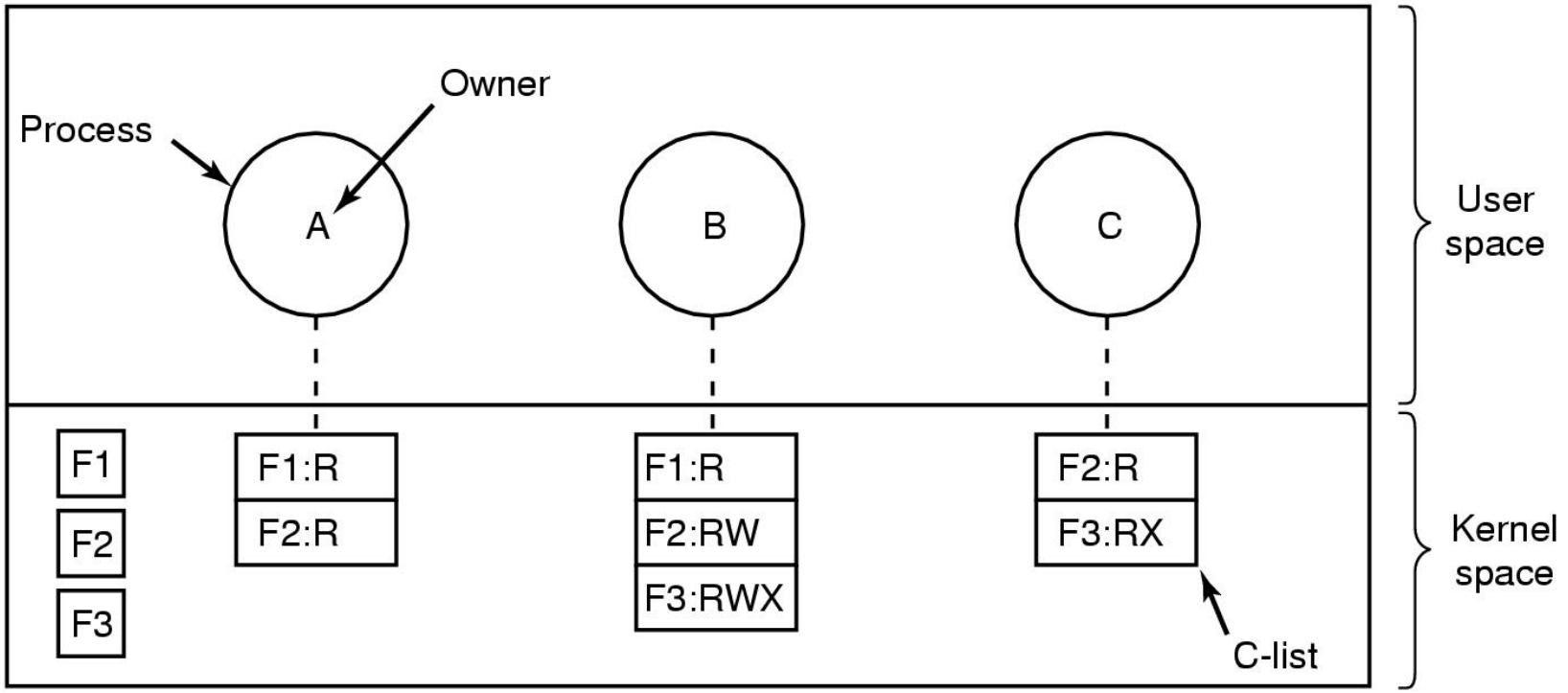
- **Problem**
  - ACLs require an entry per domain (user, role)
- **Storing on deviations from the default**
  - Default = no access
    - **high overhead for widely accessible resources**
  - Default = open access
    - **High overhead for private resources**
- **Uniform space requirements are desirable**
  - Unix Owner, Group, Others, RWX approach

Capabilities – matrix by row

Domain	Object										
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

- **Domain matrix is typically large and sparse**
  - inefficient to store the whole thing
  - store occupied columns only, with the resource? - ACLs
  - store occupied rows only, with the domain? - Capabilities

Capabilities associated with processes



- **Each process has a capability for every resource it can access**
  - Kept with other process meta data
  - Checked by the kernel on every access

Cryptographically-protected capabilities

- **Space overhead for capabilities encourages storing them in user space**
  - But what prevents a domain from manufacturing its own new capabilities?
  - Encrypted capabilities stored in user space
    - **New capabilities (encrypted) can't be guessed**

Server	Object	Rights	f(Objects, Rights, Check)

- **Generic rights include**
  - Copy capability
  - Copy object
  - Remove capability
  - Destroy object

Attacks

Login spoofing

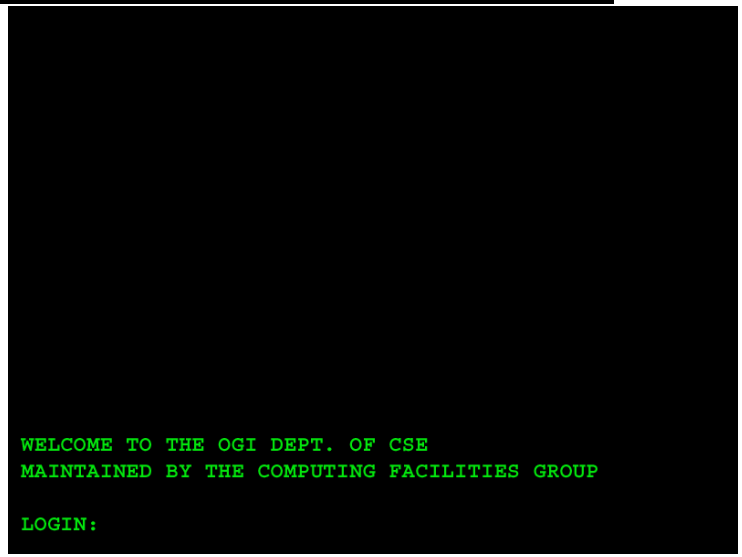
(a) (b)

(a) Correct login screen

(b) Phony login screen

Which do you prefer?

Which would you rather log into?



Trojan horses

- **Free program made available to unsuspecting user**
  - Actually contains code to do harm
- **Place altered version of utility program on victim's computer**
  - trick user into running that program
  - example, ls attack
- **Trick the user into executing something they shouldn't**

Logic bombs

- **Revenge driven attack**
- **Company programmer writes program**
  - Program includes potential to do harm
  - But its OK as long as he/she enters a password daily
  - If programmer is fired, no password and bomb "explodes"

Trap doors

```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v) break;
}
execute_shell(name);
```

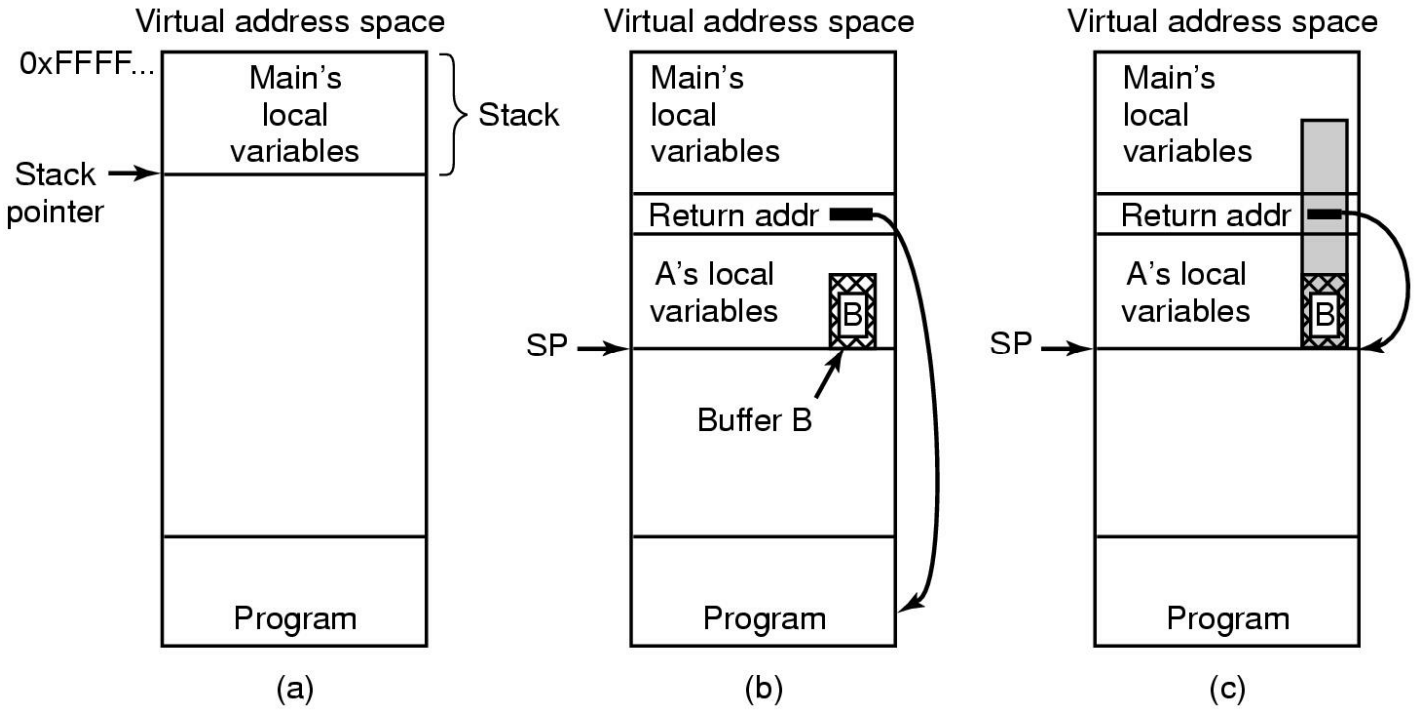
(a)

```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);
```

(b)

(a) Normal login prompt code.
(b) Login prompt code with a trapdoor inserted

Buffer overflow vulnerabilities and attacks



- (a) Situation when main program is running
- (b) After procedure *A* called
  - Buffer B waiting for input
- (c) Buffer overflow shown in gray
  - Buffer B overflowed after input of wrong type

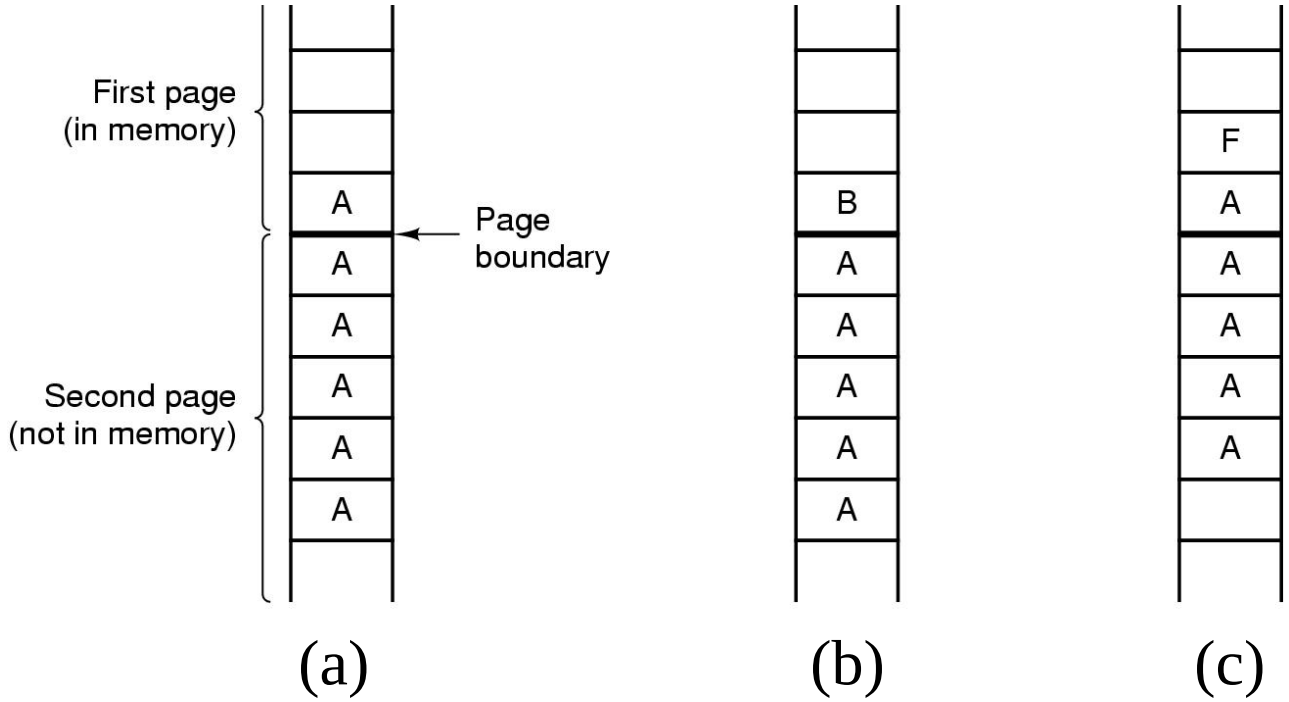
Buffer overflow attacks

- **The basic idea**
  - exploit lack of bounds checking to overwrite return address and to insert new return address and code at that address
  - exploit lack of separation between stack and code (ability to execute both)
  - allows user (attacker) code to be placed in a set UID root process and hence executed in a more privileged protection domain !
    - **If setuid root programs have this vulnerability (many do!).**

Other generic security attacks

- **Request memory, disk space, tapes and just read it**
  - Secrecy attack based on omission of zero filling on free
- **Try to do the specified DO NOTs**
  - Try illegal operations in the hope of errors in rarely executed error paths
    - **i.e, start a login and hit DEL, RUBOUT, or BREAK**
- **Convince a system programmer to add a trap door**
- **Beg someone with access to help a poor user who forgot their password**

Famous subtle security flaws



- **The TENEX password problem**
  - Place password across page boundary, ensure second page not in memory, and register user-level page fault handler
  - OS checks password one char at a time
    - **If first char incorrect, no page fault occurs**
    - **requires 128n tries instead of 128^n**

Design principles for security

- **System design should be public**
  - Security through obscurity doesn't work!
- **Default should be no access**
- **Check for "current" authority**
  - Allows access to be revoked
- **Give each process the least privilege possible**
- **Protection mechanism should be**
  - simple
  - uniform
  - in lowest layers of system
- **Scheme should be psychologically acceptable**

And … keep it simple!

External Attacks

External threats, viruses & worms

- **External threat**
  - code transmitted to target machine
  - code executed there, doing damage
  - may utilize an internal attack to gain more privilege (ie. Buffer overflow)
- **Malware = program that can reproduce itself**
  - Virus: requires human action to propagate
    - **Typically attaches its code to another program**
  - Worm: propagates by itself
    - **Typically a stand-alone program**
- **Goals of malware writer**
  - quickly spreading virus/worm
  - difficult to detect
  - hard to get rid of

Virus damage scenarios

- **Blackmail**
- **Denial of service as long as malware runs**
- **Damage data/software/hardware**
- **Target a competitor's computer**
  - do harm
  - espionage
- **Intra-corporate dirty tricks**
  - sabotage another corporate officer's files

How viruses work

- **Virus written in assembly language**
- **Inserted into another program**
  - use tool called a "dropper"
- **Virus dormant until program executed**
  - then infects other programs
  - eventually executes its "payload"

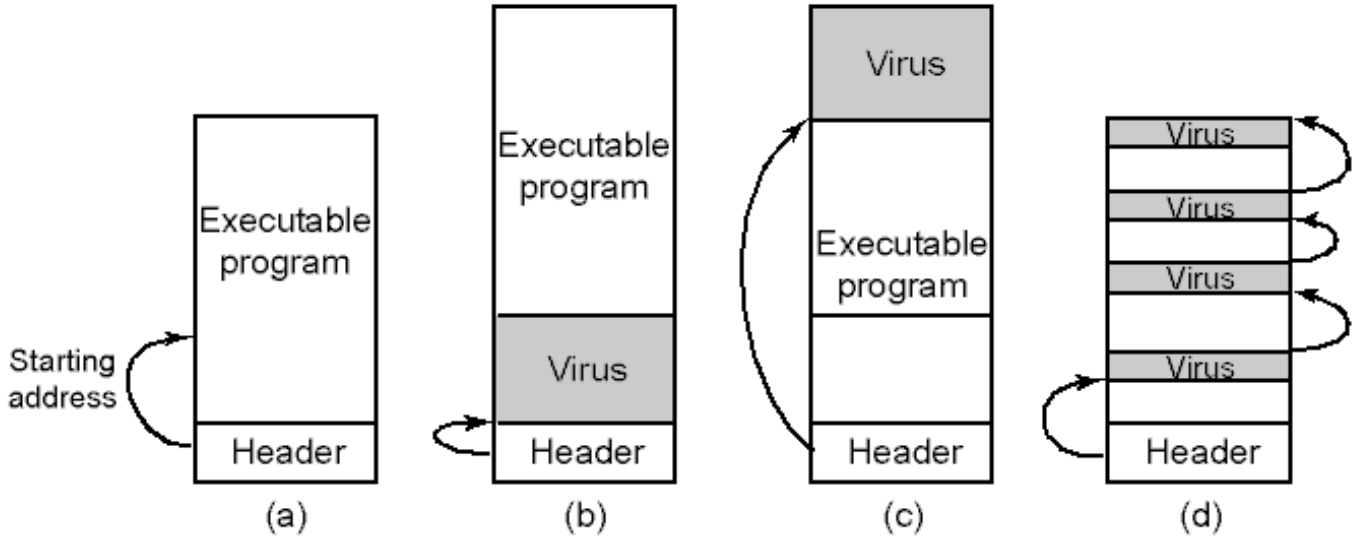
Searching for executable files to infect

Recursive procedure that finds executable files on a UNIX system

Virus could infect them all

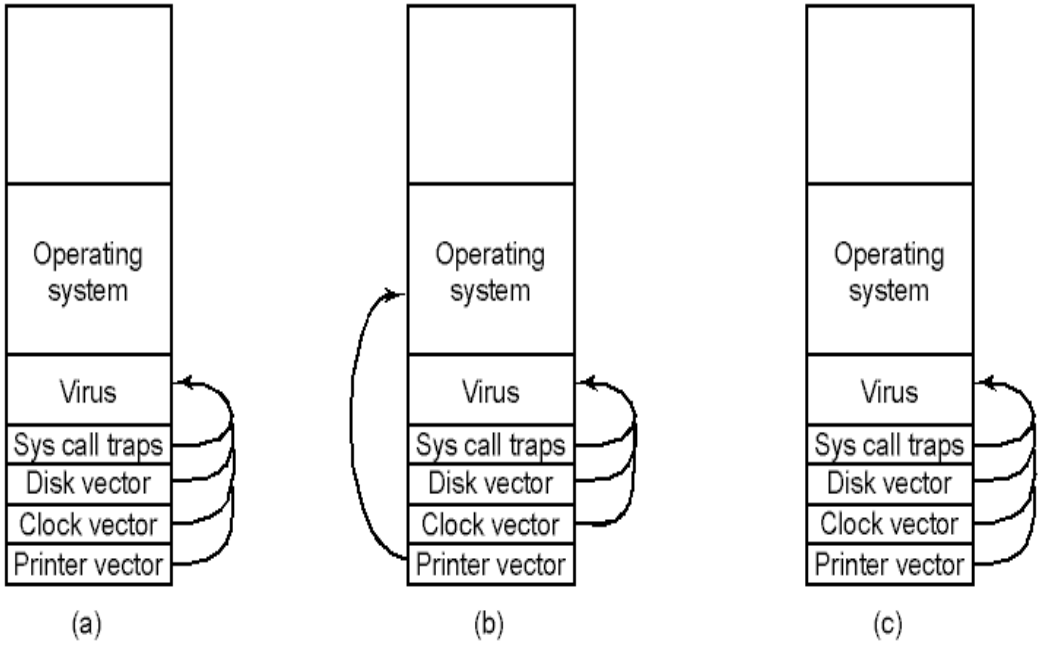
```
#include <sys/types.h>                      /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                           /* for lstat call to see if file is sym link */

search(char *dir_name)
{                                           /* recursively search for executables */
    DIR *dirp;                              /* pointer to an open directory stream */
    struct dirent *dp;                      /* pointer to a directory entry */

    dirp = opendir(dir_name);               /* open this directory */
    if (dirp == NULL) return;               /* dir could not be opened; forget it */
    while (TRUE) {
        dp = readdir(dirp);                 /* read next directory entry */
        if (dp == NULL) {                   /* NULL means we are done */
        chdir ("..");                       /* go back to parent directory */
        break;                              /* exit loop */
    }
    if (dp->d_name[0] == '.') continue;     /* skip the . and .. directories */
    lstat(dp->d_name, &sbuf);               /* is entry a symbolic link? */
    if (S_ISLNK(sbuf.st_mode)) continue;    /* skip symbolic links */
    if (chdir(dp->d_name) == 0) {           /* if chdir succeeds, it must be a dir */
        search(".");                        /* yes, enter and search it */
    } else {                                    /* no (file), infect it */
        if (access(dp->d_name,X_OK) == 0) /* if executable, infect it */
            infect(dp->d_name);
    }
    closedir(dirp);                         /* dir processed; close and return */
}
```

How viruses hide



- An executable program
- Virus at the front (program shifted, size increased)
- Virus at the end (size increased)
- With a virus spread over free space within program
  - less easy to spot, size may not increase

Difficulty extracting OS viruses

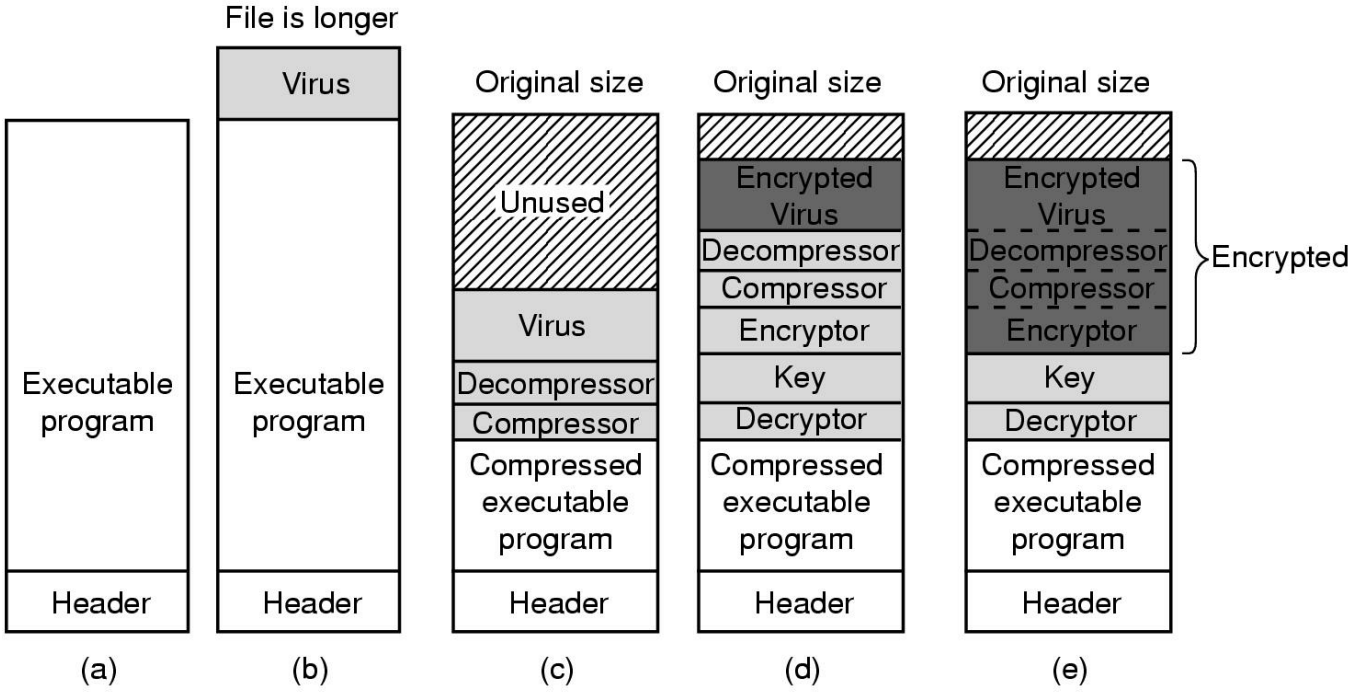


- After virus has captured interrupt, trap vectors
- After OS has retaken printer interrupt vector
- After virus has noticed loss of printer interrupt vector and recaptured it

How viruses spread

- **Virus is placed where its likely to be copied or executed**
- **When it arrives at a new machine**
  - infects programs on hard drive, floppy
  - may try to spread over LAN
- **Attach to innocent looking email**
  - when it runs, use mailing list to replicate further

Antivirus and anti-antivirus techniques



(a) A program
(b) An infected program
(c) A compressed infected program
(d) An encrypted virus
(e) A compressed virus with encrypted compression code

Anti-antivirus techniques

```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```
(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```
(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```
(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```
(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```
(e)

- **Examples of a polymorphic virus**
  - All of these examples do the same thing

Antivirus software

- **Integrity checkers**
  - use checksums on executable files
  - hide checksums to prevent tampering?
  - encrypt checksums and keep key private
- **Behavioral checkers**
  - catch system calls and check for suspicious activity
  - what does "normal" activity look like?

Virus avoidance and recovery

- **Virus avoidance**
  - good OS
  - firewall
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups
    - **Need to avoid backing up the virus!**
    - **Or having the virus infect your backup/restore software**
- **Recovery from virus attack**
  - halt computer, reboot from safe disk, run antivirus software

The Internet worm

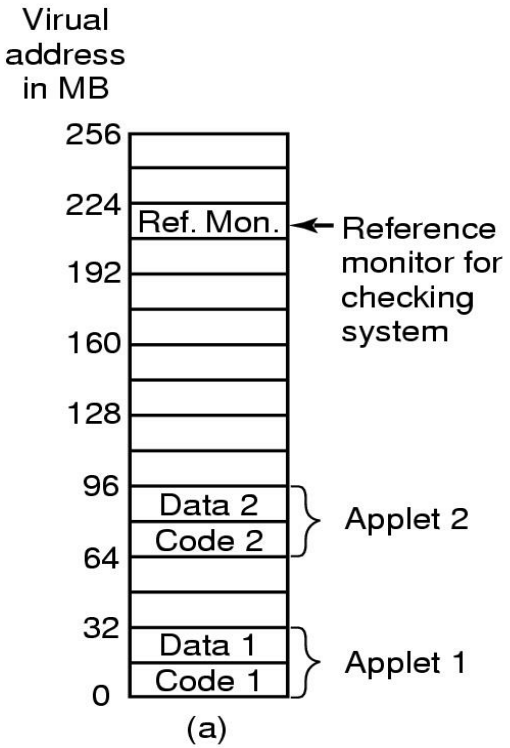
- **Robert Morris constructed the first Internet worm**
  - Consisted of two programs
    - **bootstrap to upload worm and the worm itself**
  - Worm first hid its existence then replicated itself on new machines
  - Focused on three flaws in UNIX
    - **rsh – exploit local trusted machines**
    - **fingerd – buffer overflow attack**
    - **sendmail – debug problem**
- **It was too aggressive and he was caught**

Availability and denial of service attacks

- **Denial of service (DoS) attacks**
  - May not be able to break into a system, but if you keep it busy enough you can tie up all its resources and prevent others from using it
- **Distributed denial of service (DDOS) attacks**
  - Involve large numbers of machines (botnet)
- **Examples of known attacks**
  - **Ping of death – large ping packets cause system crash**
  - **SYN floods – tie up buffer in establishment of TCP flows**
  - **UDP floods**
  - **Spoofing return address (ping etc)**
- **Some attacks are sometimes prevented by a firewall**

Security Approaches for Mobile Code

Sandboxing




```
MOV R1, S1
SHR #24, S1
CMP S1, S2
TRAPNE
JMP (R1)
```

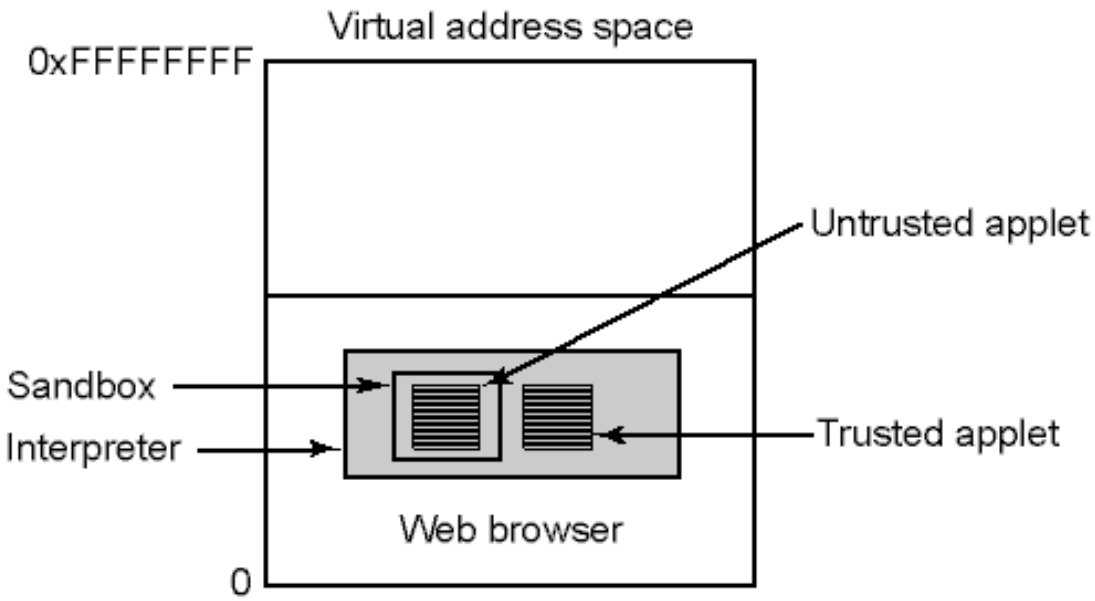
(b)

(a) Memory divided into 1-MB sandboxes

- each applet has two sandboxes, one for code and one for data
- some static checking of addresses

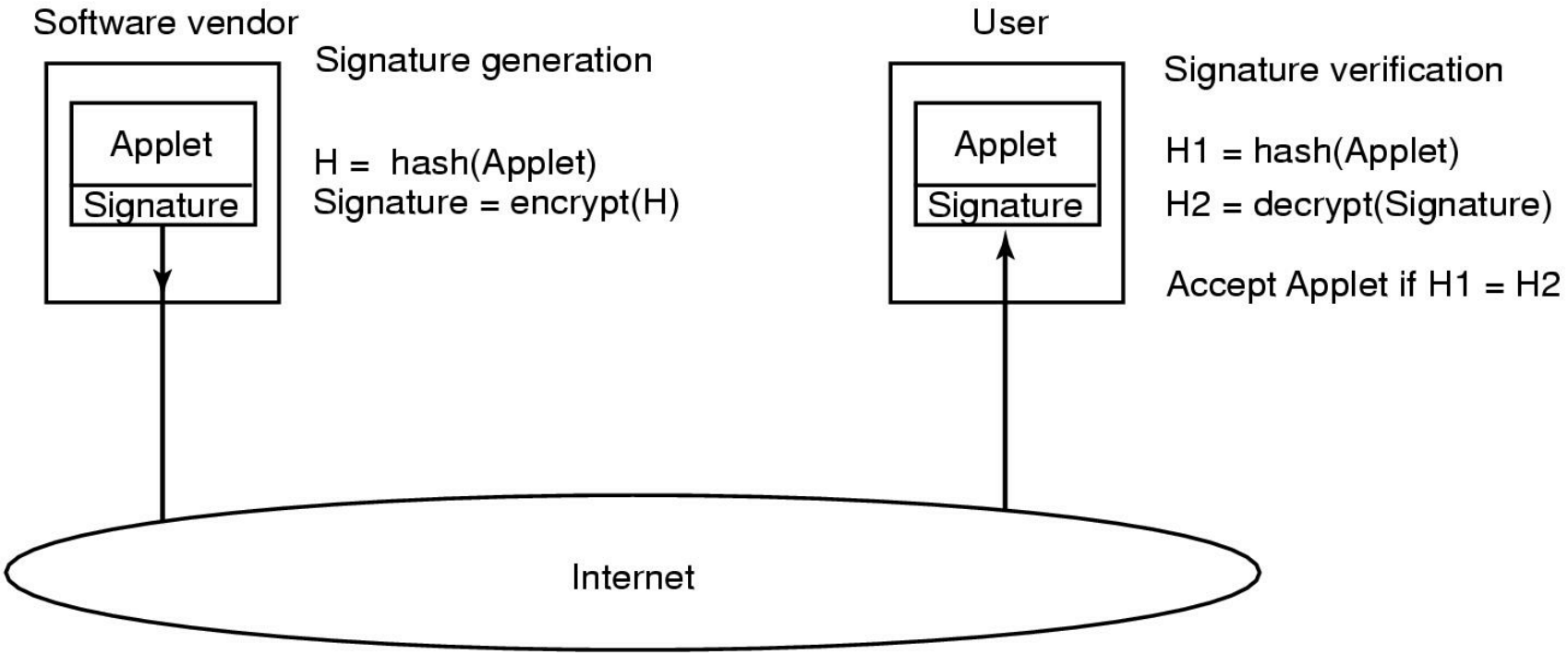
(b) Code inserted for runtime checking of dynamic target addresses

Interpretation



Applets can be interpreted by a Web browser

Code signing



How code signing works

Type safe languages

- **A type safe language**
  - compiler rejects attempts to misuse variables

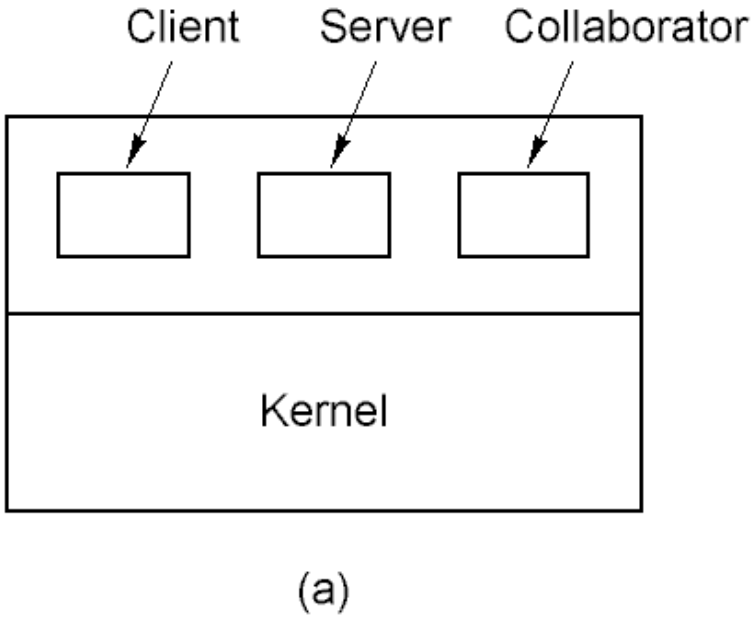
- **Checks include …**
  - Attempts to forge pointers
  - Violation of access restrictions on private class members
  - Misuse of variables by type
  - Generation of buffer/stack over/underflows
  - Illegal conversion of variables to another type

Covert Channels

Preserving secrecy

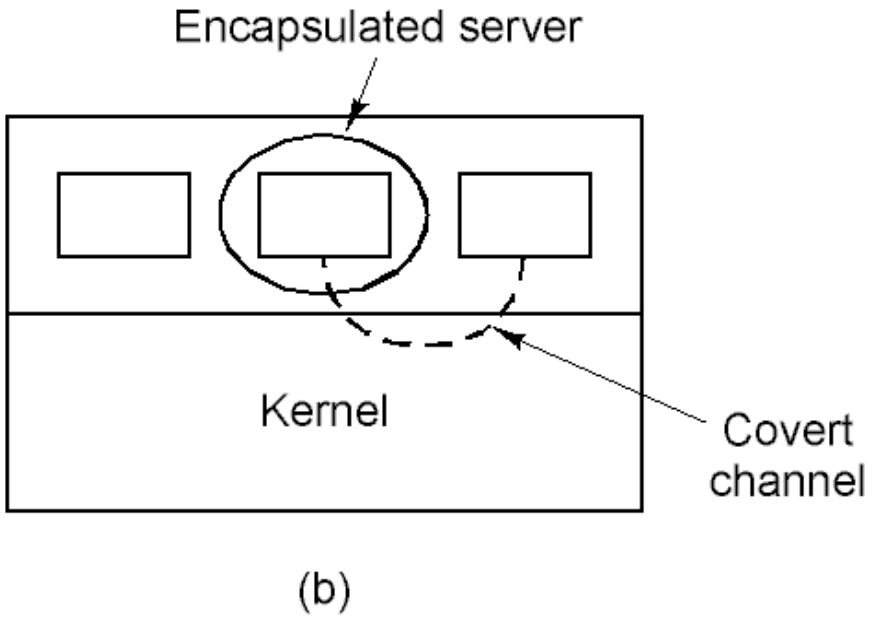
- **How can you ensure that a process in a privileged domain doesn't communicate secret domain information to a process in a non-privileged domain?**
  - Prevent/filter all interprocess communication?
- **Covert channels are ways of communicating outside of the normal ipterprocess communication mechanisms**

Covert channels



(a)

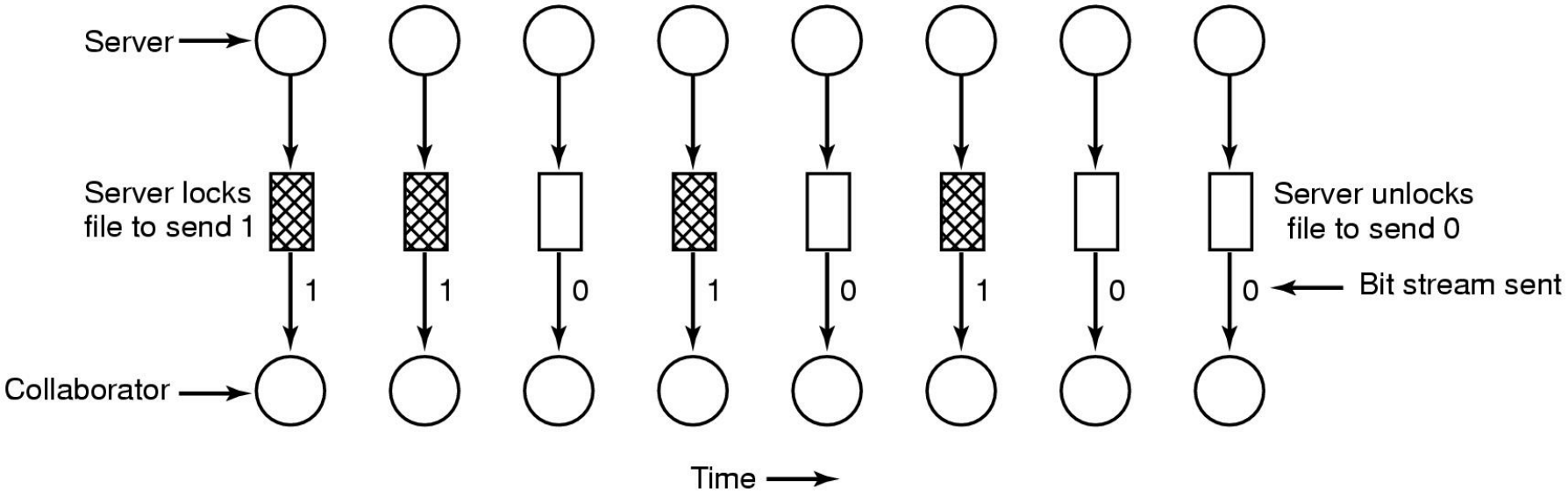
Client, server and collaborator processes



(b)

Encapsulated server can still leak to collaborator via covert channels

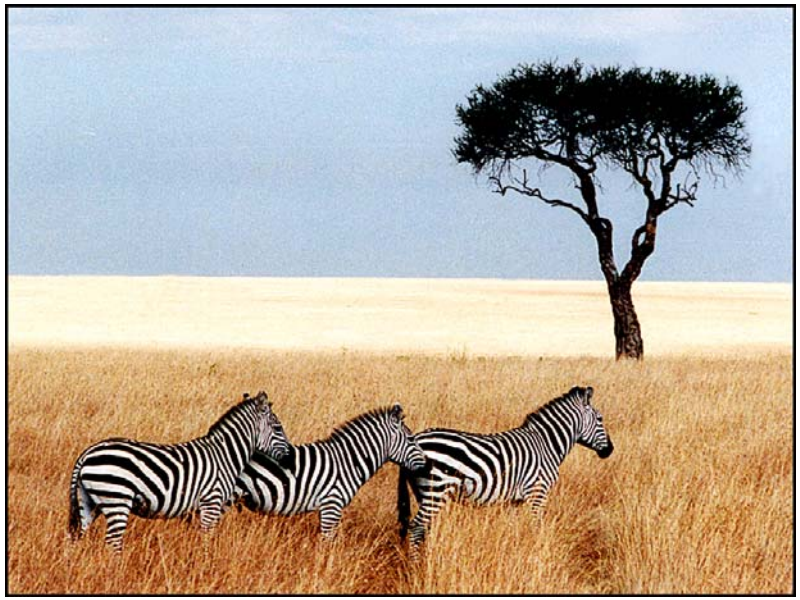
Locking as a covert channel



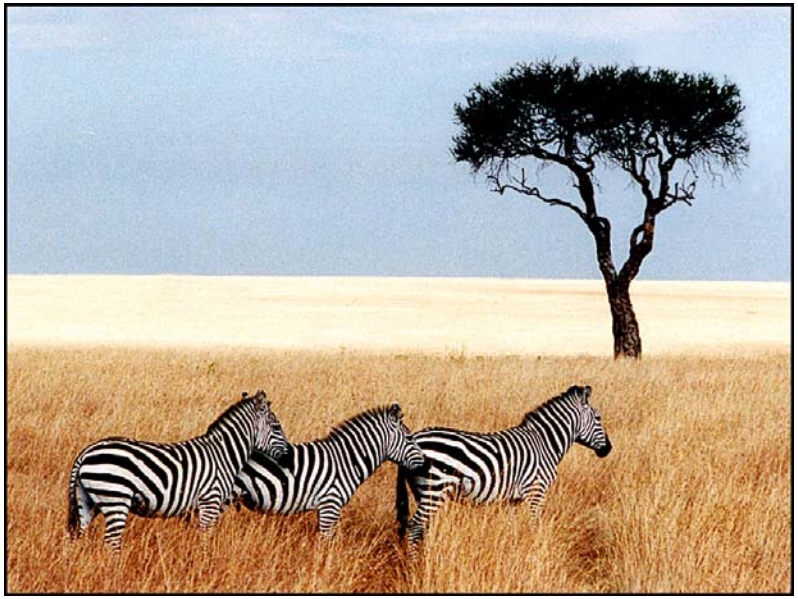
A covert channel using file locking

Covert channels

- **Pictures appear the same**
- **Picture on right has text of 5 Shakespeare plays**
  - encrypted, inserted into low order bits of color values
  - (assume high resolution images)



Zebras

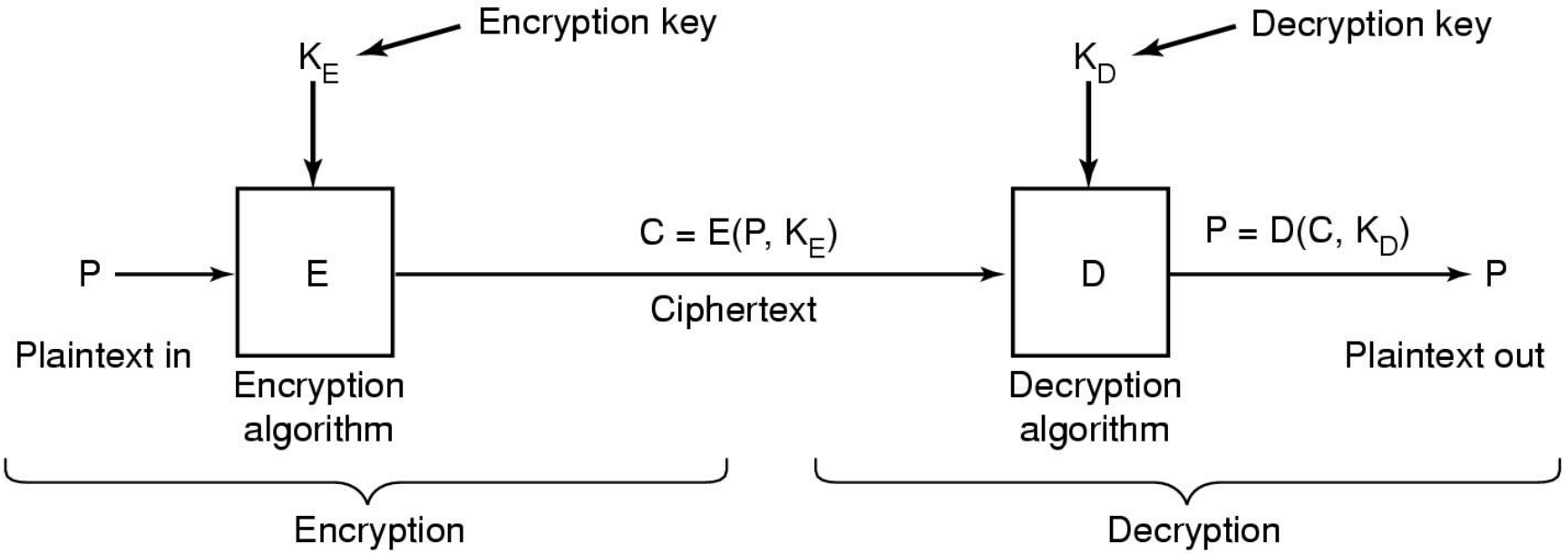


Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear

Spare Slides

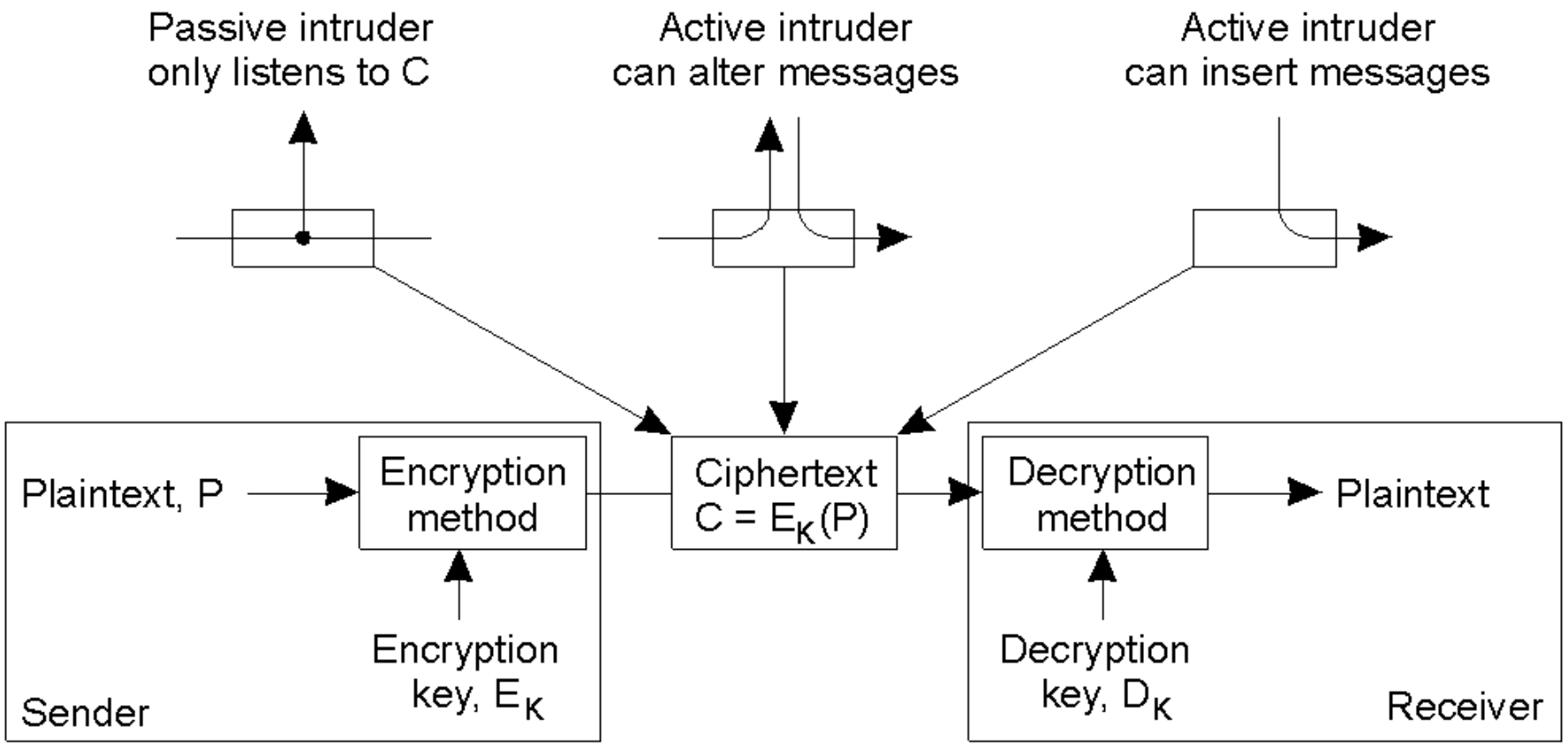
Brief Introduction to Cryptography Tools

Basics of Cryptography



Relationship between the plaintext and the ciphertext

Cryptography: confidentiality and integrity



Secret-key cryptography

- **Example: mono-alphabetic substitution**

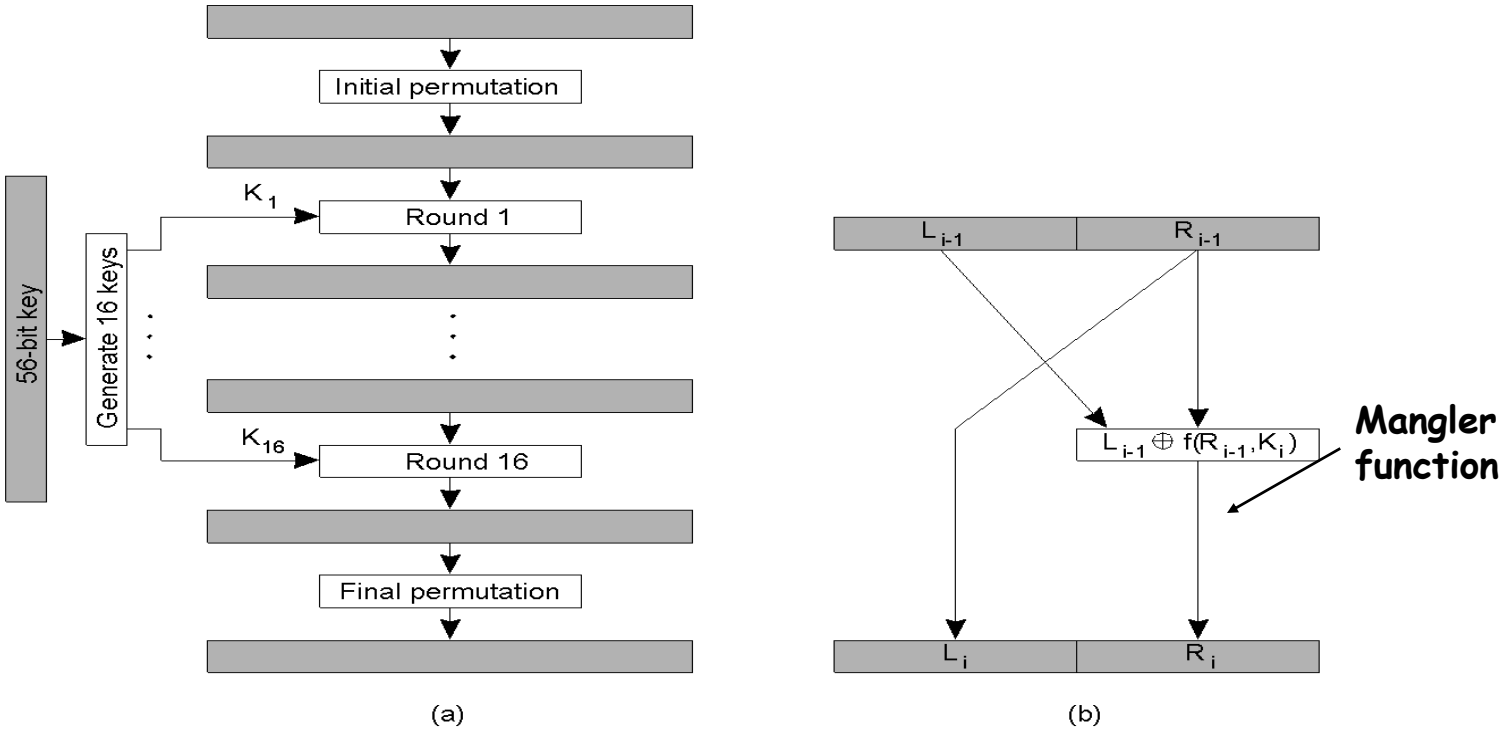
  Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ

  Cyphertext: QWERTYUIOPASDFGHJKLZXCVBNM

- **Given the encryption key (QWERTYUIOPASDFGHJKLZXCVBNM),**
  - easy to find decryption key using statistical properties of natural language (common letters and digrams)
  - … despite size of search space of 26! possible keys

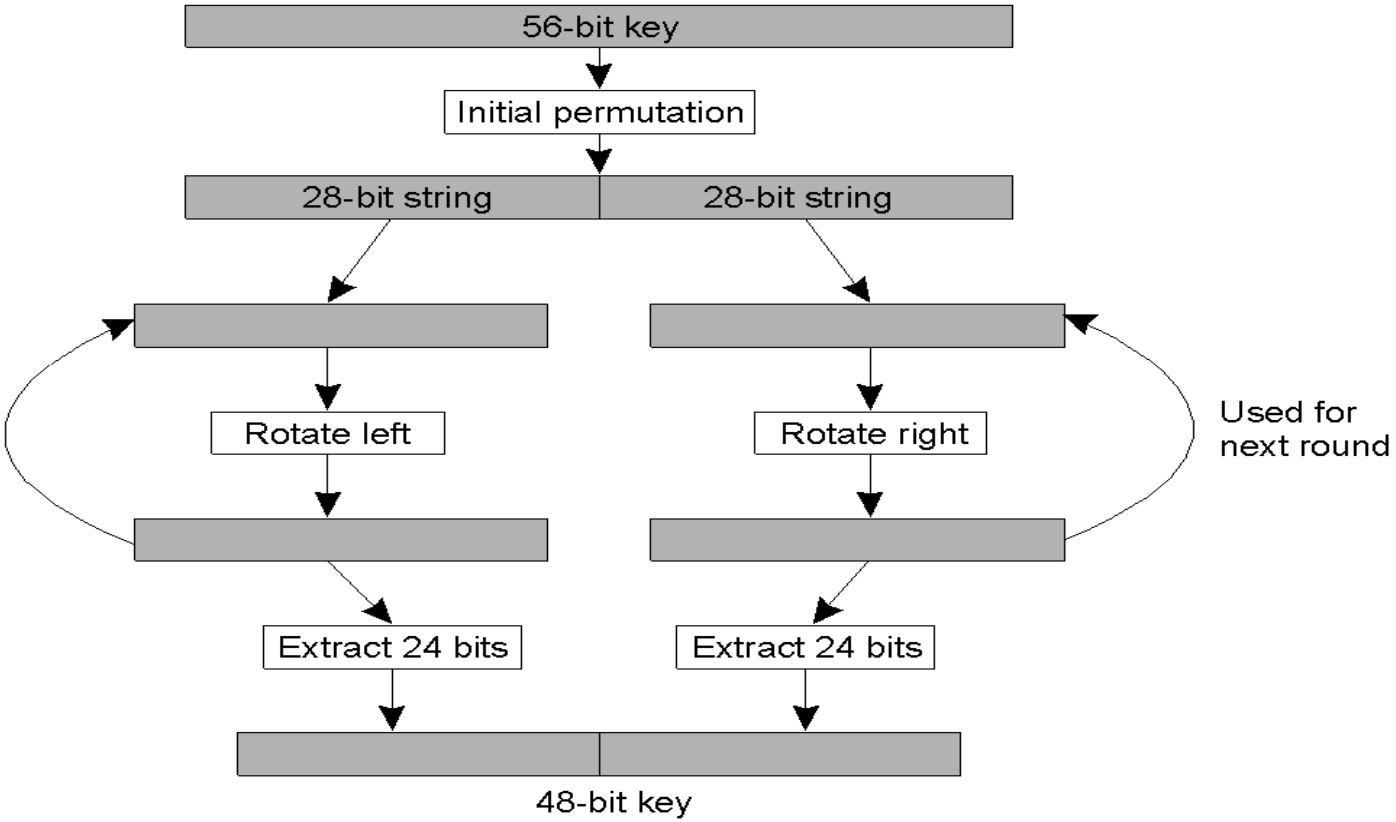
- **Function should be more complex and search space very large.**

Symmetric cryptography: DES



- **DES operates on 64-bit blocks of data**
  - initial permutation
  - 16 rounds of transformations each using a different encryption key

Per-round key generation in DES



- Each key derived from a 56-bit master by mangling function based on splitting, rotating, bit extraction and combination

Symmetric (secret) key cryptography

- **Fast for encryption and decryption**
- **Difficult to break analytically**
- **Subject to brute force attacks**
  - as computers get faster must increase the number of rounds and length of keys
- **Main problem**
  - *how to distribute the keys in the first place?*

Public-key cryptography

- **Use different keys for encryption and decryption**
- **Knowing the encryption key doesn't help you decrypt**
  - the encryption key can be made public
  - encryption key is given to sender
  - decryption key is held privately by the receiver
- **But how does it work?**

Public-key cryptography

- **Asymmetric (one-way) functions**
  - given function *f* it is easy to evaluate $y = f(x)$
  - but given *y* its computationally infeasible to find *x*

- **Trivial example of an asymmetric function**

  encryption: $y = x^2$

  decryption: $x = squareroot\ (y)$

- **Challenge**
  - finding a function with strong security properties but efficient encryption and decryption

Public-key cryptography: RSA

- **RSA (Rivest, Shamir, Adleman)**
  - encryption involves multiplying large prime numbers
  - cracking involves finding prime factors of a large number

- **Steps to generate encryption key (*e*) and decryption key (*d*)**
  - Choose two very large prime numbers, *p* and *q*
  - Compute $n = p \times q$ and $z = (p - 1) \times (q - 1)$
  - Choose a number *d* that is relatively prime to *z*
  - Compute the number *e* such that $e \times d = 1 \bmod z$

Public-key cryptography: RSA

- **Messages split into fixed length blocks of bits**
  - interpreted as numbers with value $0 <= m_i < n$
- **Encryption**

  $$c_i = m_i^e \pmod n$$

  - requires that you have n and encryption key e
- **Decryption**

  $$m_i = c_i^d \pmod n$$

  - requires that you have n and decryption key d

RSA vs DES

- RSA is more secure than DES
- RSA requires 100-1000 times more computation than DES to encrypt and decrypt
- RSA can be used to exchange private DES keys
- DES can be used for message contents

Secure hash functions

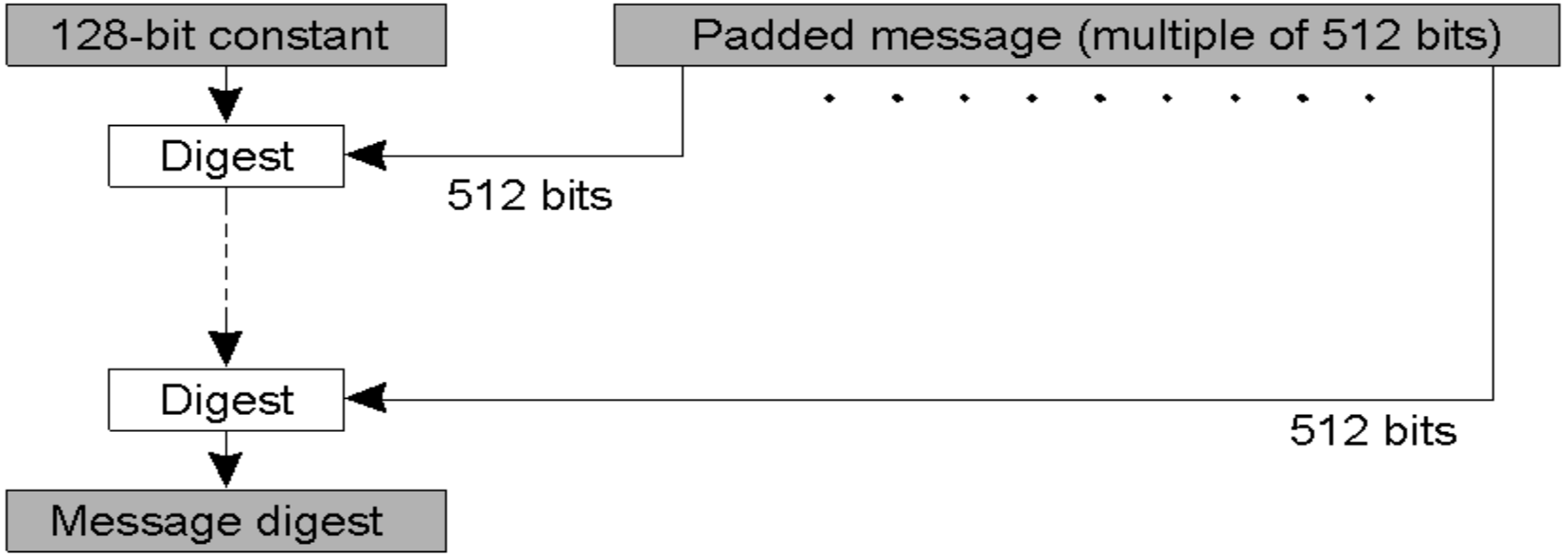
- **Hash functions $h = H(m)$ are one way functions**
  - can't find input m from output h
  - easy to compute h from m
- **Weak collision resistance**
  - given m and $h = H(m)$ difficult to find different input m' such that $H(m) = H(m')$
- **Strong collision resistance**
  - given H it is difficult to find any two different input values m and m' such that $H(m) = H(m')$
- **They typically generate a short fixed length output string from arbitrary length input string**

Example secure hash functions

- **MD5 - (Message Digest)**
  - produces a 16 byte result
- **SHA - (Secure Hash Algorithm)**
  - produces a 20 byte result

Secure hash functions : MD5

- **The structure of MD5**
  - produces a 128-bit digest from a set of 512-bit blocks
  - k block digests require k phases of processing each with four rounds of processing to produce one message digest



Per phase processing in MD5

- **Each phase involves for rounds of processing**

F (x,y,z) = (x AND y) OR ((NOT x) AND z)
G (x,y,z) = (x AND z) OR (y AND (NOT z))
H (x,y,z) = x XOR y XOR z
I (x,y,z) = y XOR (x OR (NOT z))

Per round processing in MD5

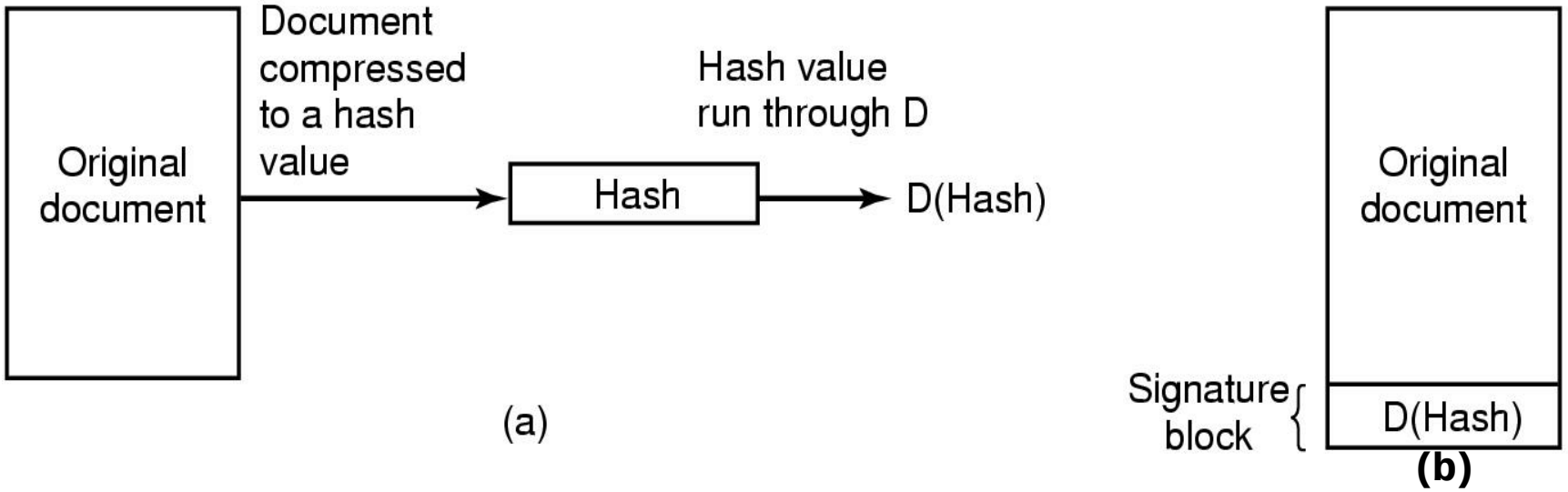
- The 16 iterations during the first round in a phase of MD5 using function F

Iterations 1-8	Iterations 9-16
$p \leftarrow (p + F(q,r,s) + b_0 + C_1) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_8 + C_9) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_1 + C_2) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_9 + C_{10}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_2 + C_3) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{10} + C_{11}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_3 + C_4) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{11} + C_{12}) \lll 22$
$p \leftarrow (p + F(q,r,s) + b_4 + C_5) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_{12} + C_{13}) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_5 + C_6) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_{13} + C_{14}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_6 + C_7) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{14} + C_{15}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_7 + C_8) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{15} + C_{16}) \lll 22$

What can you use a hash function for?

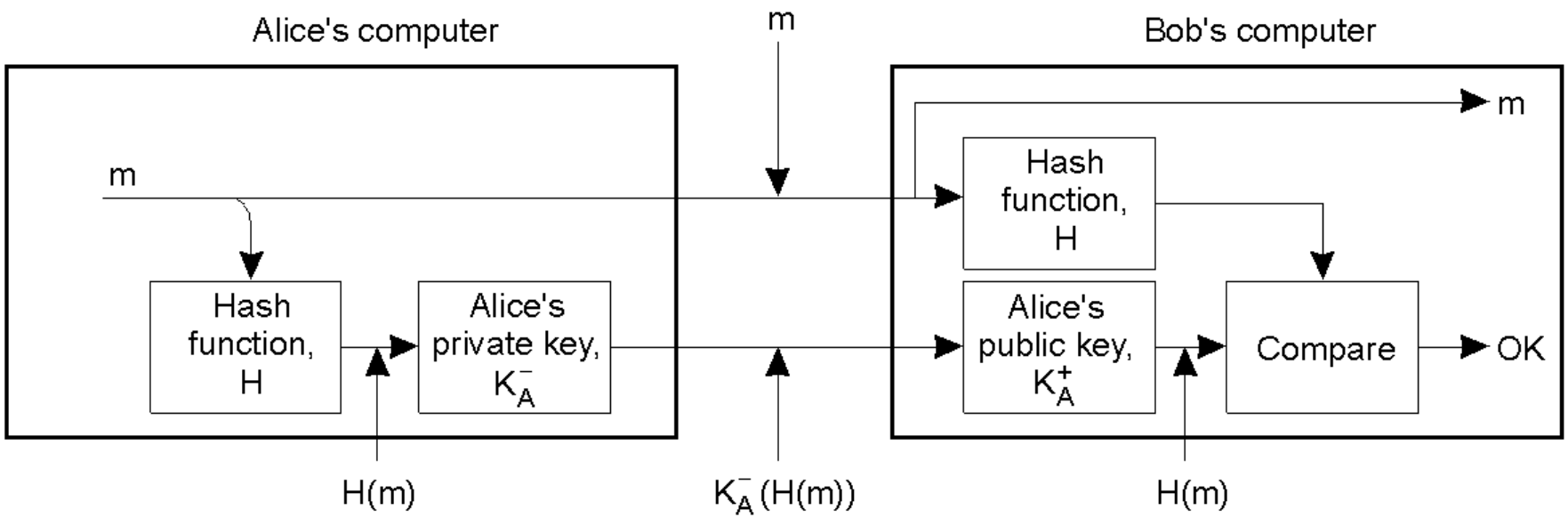
- **To verify the integrity of data**
  - if the data has changed the hash will change (weak and strong collision resistance properties)
- **To "sign" or "certify" data or software**

Digital signatures



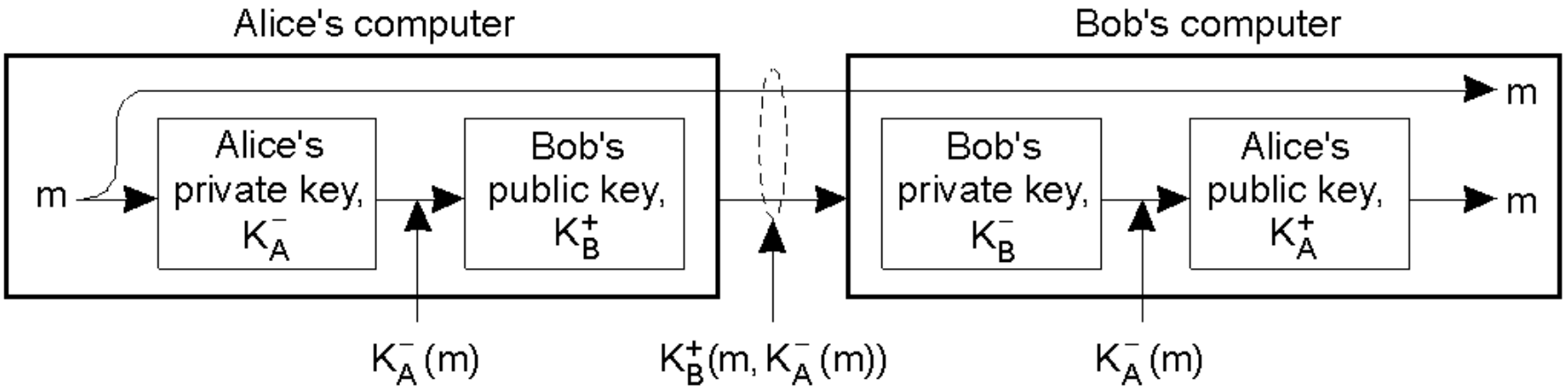
- Computing a signature block
- What the receiver gets

Digital signatures using a message digest



Notation	Description
$K_{A,B}$	**Secret key shared by A and B**
K_A^+	**Public key of A**
K_A^-	**Private key of A**

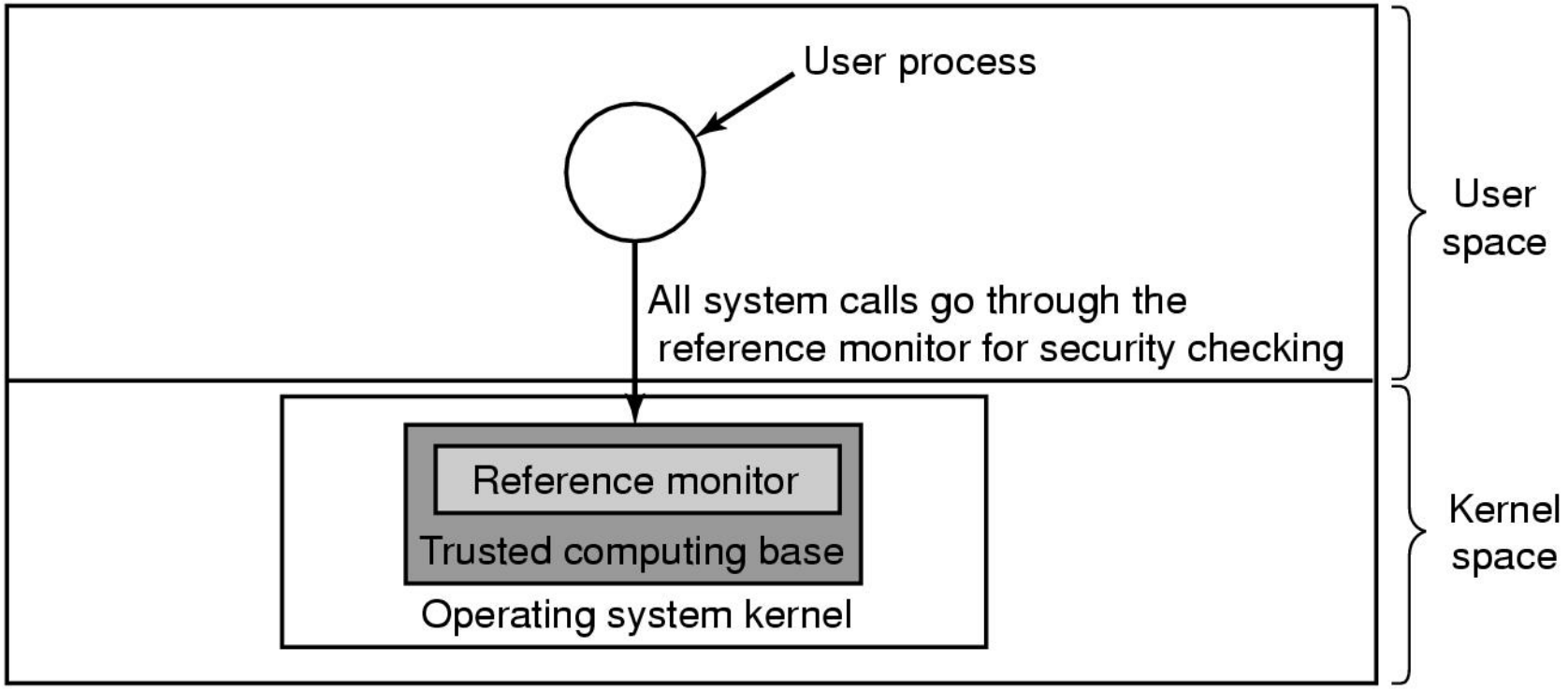
Digital signatures with public-key cryptography



Notation	Description
$K_{A,B}$	**Secret key shared by A and B**
K_A^+	**Public key of A**
K_A^-	**Private key of A**

Trusted Systems and Formal Models

Trusted Systems
Trusted Computing Base



A reference monitor

Formal Models of Secure Systems

	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute		Read Write

(a)

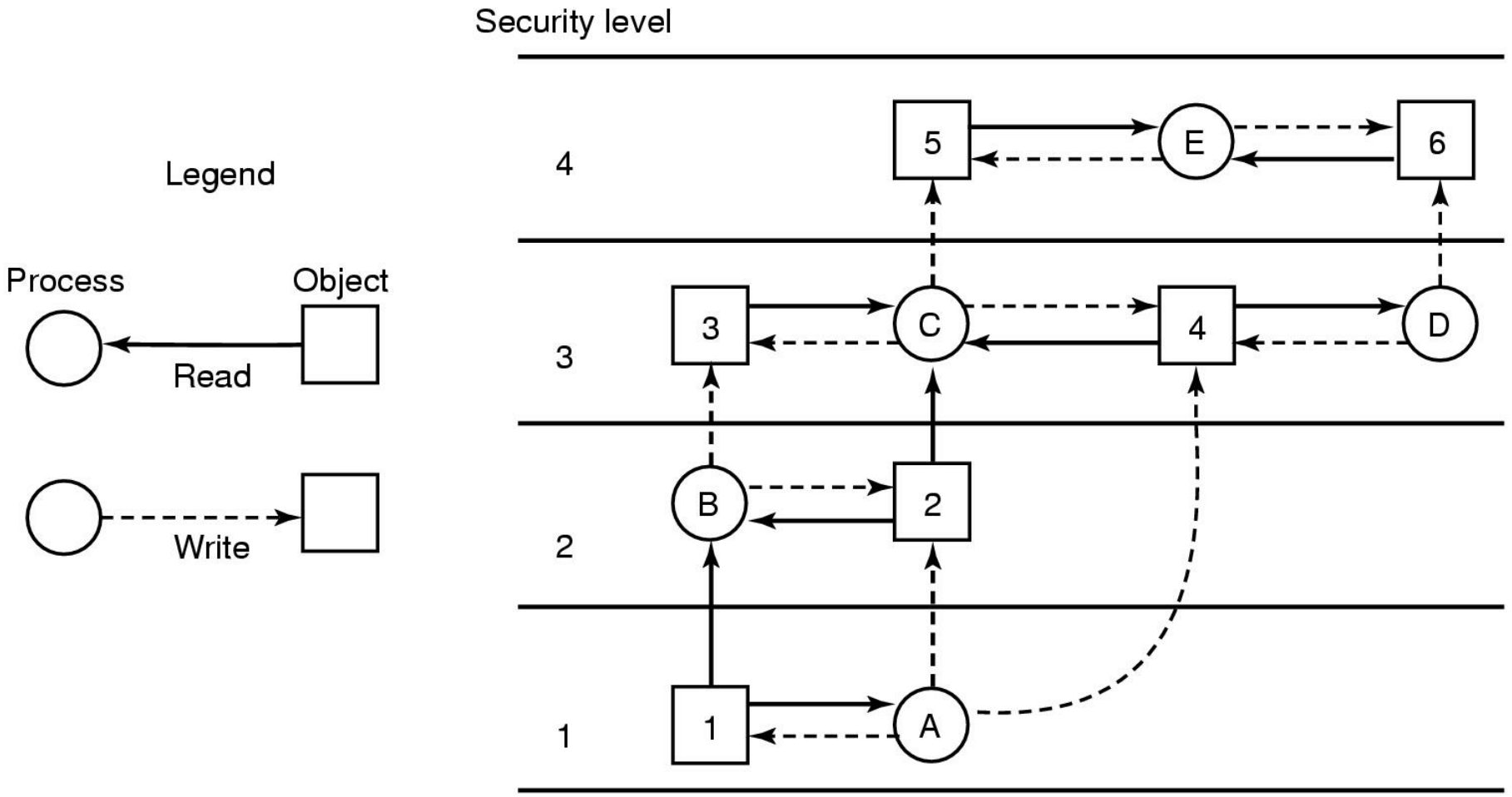
	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute	Read	Read Write

(b)

(a) An authorized state

(b) An unauthorized state

Multilevel Security (1)



The Bell-La Padula multilevel security model

Multilevel Security (2)

The Biba Model

- **Principles to guarantee integrity of data**
- **Simple integrity principle**
  - process can write only objects at its security level or lower
- **The integrity * property**
  - process can read only objects at its security level or higher

Orange Book Security (1)

Criterion	D	C1	C2	B1	B2	B3	A1
Security policy							
Discretionary access control		X	X	→	→	X	→
Object reuse			X	→	→	→	→
Labels				X	X	→	→
Label integrity				X	→	→	→
Exportation of labeled information				X	→	→	→
Labeling human readable output				X	→	→	→
Mandatory access control				X	X	→	→
Subject sensitivity labels					X	→	→
Device labels					X	→	→
Accountability							
Identification and authentication		X	X	X	→	→	→
Audit			X	X	X	X	→
Trusted path					X	X	→

- Symbol X means new requirements
- Symbol -> requirements from next lower category apply here also

Orange Book Security (2)

Assurance							
System architecture		X	X	X	X	X	→
System integrity		X	→	→	→	→	→
Security testing		X	X	X	X	X	X
Design specification and verification				X	X	X	X
Covert channel analysis					X	X	X
Trusted facility management					X	X	→
Configuration management					X	→	X
Trusted recovery						X	→
Trusted distribution							X
Documentation							
Security features user's guide		X	→	→	→	→	→
Trusted facility manual		X	X	X	X	X	→
Test documentation		X	→	→	X	→	X
Design documentation		X	→	X	X	X	X

Java security

URL	Signer	Object	Action
www.taxprep.com	TaxPrep	/usr/susan/1040.xls	Read
*		/usr/tmp/*	Read, Write
www.microsoft.com	Microsoft	/usr/susan/Office/–	Read, Write, Delete

Examples of specified protection with JDK 1.2