# Protection and Security

Chapters 14+15

# Objectives

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

# Goals of Protection

- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

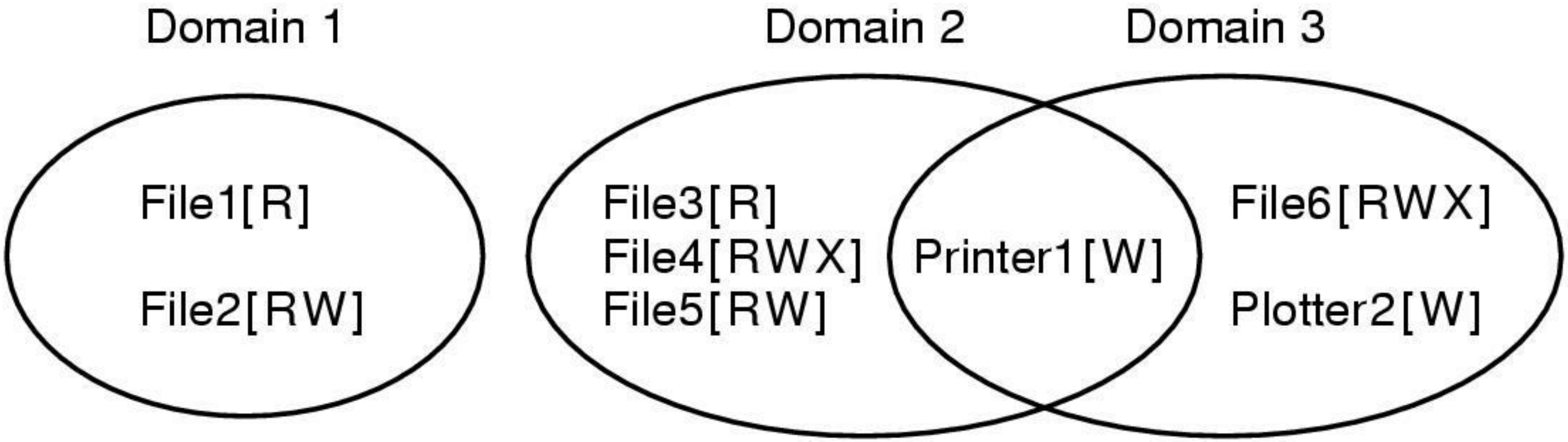
# Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks, and not more than needed.
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - "Need to know" a similar concept regarding access to data

# Principles of Protection (Cont.)

- Must consider “grain” aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists
- Domain can be user, process, procedure

# Protection Mechanisms
# Protection Domains



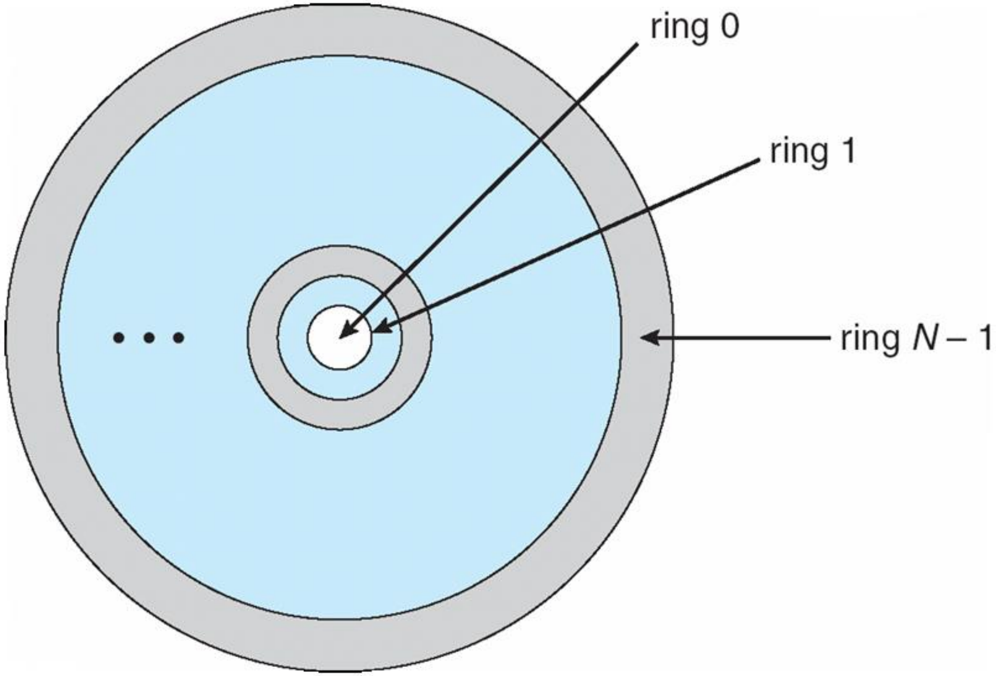
- Domains are (object, right) pairs
- Examples of three protection domains
- Printer1 is in two domains at the same time.
- Traditionally, domains correspond to users, or root (superuser).

# Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
    - Each file has associated with it a domain bit (setuid bit)
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed
    - When execution completes user-id is reset
- Domain switch accomplished via passwords
    - `su` command temporarily switches to another user’s domain when other domain’s password provided
- Domain switching via commands
    - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

# Domain Implementation (MULTICS)

- Let $D_i$ and $D_j$ be any two domain rings
- If $j < I \Rightarrow D_i \subseteq D_j$



# Multics Benefits and Limits

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex → more overhead
- But does not allow strict need-to-know
  - Object accessible in $D_j$ but not in $D_i$, then ***j*** must be < ***i***
  - But then every segment accessible in $D_i$ also accessible in $D_j$

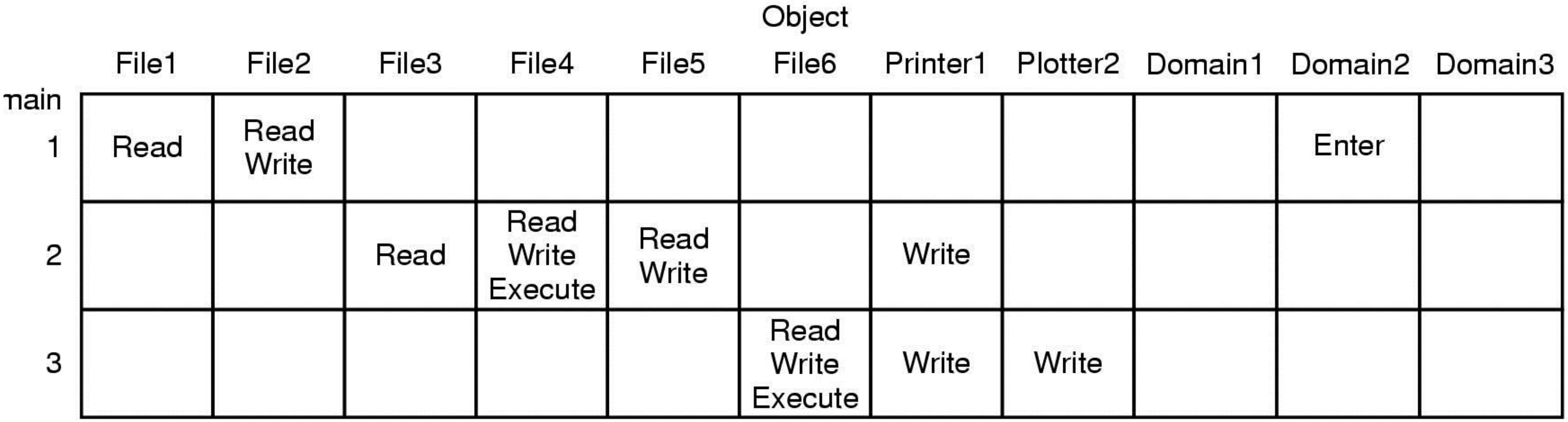
# Protection Domains

| | Object | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Domain | File1 | File2 | File3 | File4 | File5 | File6 | Printer1 | Plotter2 |
| 1 | Read | Read<br>Write | | | | | | |
| 2 | | | Read | Read<br>Write<br>Execute | Read<br>Write | | Write | |
| 3 | | | | | | Read<br>Write<br>Execute | Write | Write |

- The system keeps track of objects+domains via a protection matrix (access matrix)
- Practical implementations can differ.
- Example: domain1 has read permission to file1 and read/write permission to file2, etc.

# Protection Domains

- Domains can be objects in access matrices as well.
- Example below: a protection matrix with domains as objects
- Domain 1 can switch to (enter) domain 2, but it cannot switch back to domain 1.
  - Example of user mode → kernel mode switch (or setuid in Unix)



| | | | | | | Object | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| main | File1 | File2 | File3 | File4 | File5 | File6 | Printer1 | Plotter2 | Domain1 | Domain2 | Domain3 |
| 1 | Read | Read<br>Write | | | | | | | | Enter | |
| 2 | | | Read | Read<br>Write<br>Execute | Read<br>Write | | Write | | | | |
| 3 | | | | | | Read<br>Write<br>Execute | Write | Write | | | |

# Implementation of the Access Matrix

- Global Table.
- On invocation of a method $R_k$ on an object $O_j$ by a process $P_i$ running in a domain $D_h$,
  - the table Domain column is searched for $D_h$,
  - the Object row is searched for an entry $O_j$,
  - the entry at the intersection of the row and column is searched for the method $R_k$.

# Practical implementation of access matrices

- Access matrices are typically large and sparse.
- No point in wasting large amounts of disk space in order to store such a matrix.
- Practical implementations store this matrix in one of two ways:
  - Store row-wise and only store the non-empty entries (Access Control List — ACL)
  - Store column-wise and only store the non-empty entries (Capability lists — C-lists)

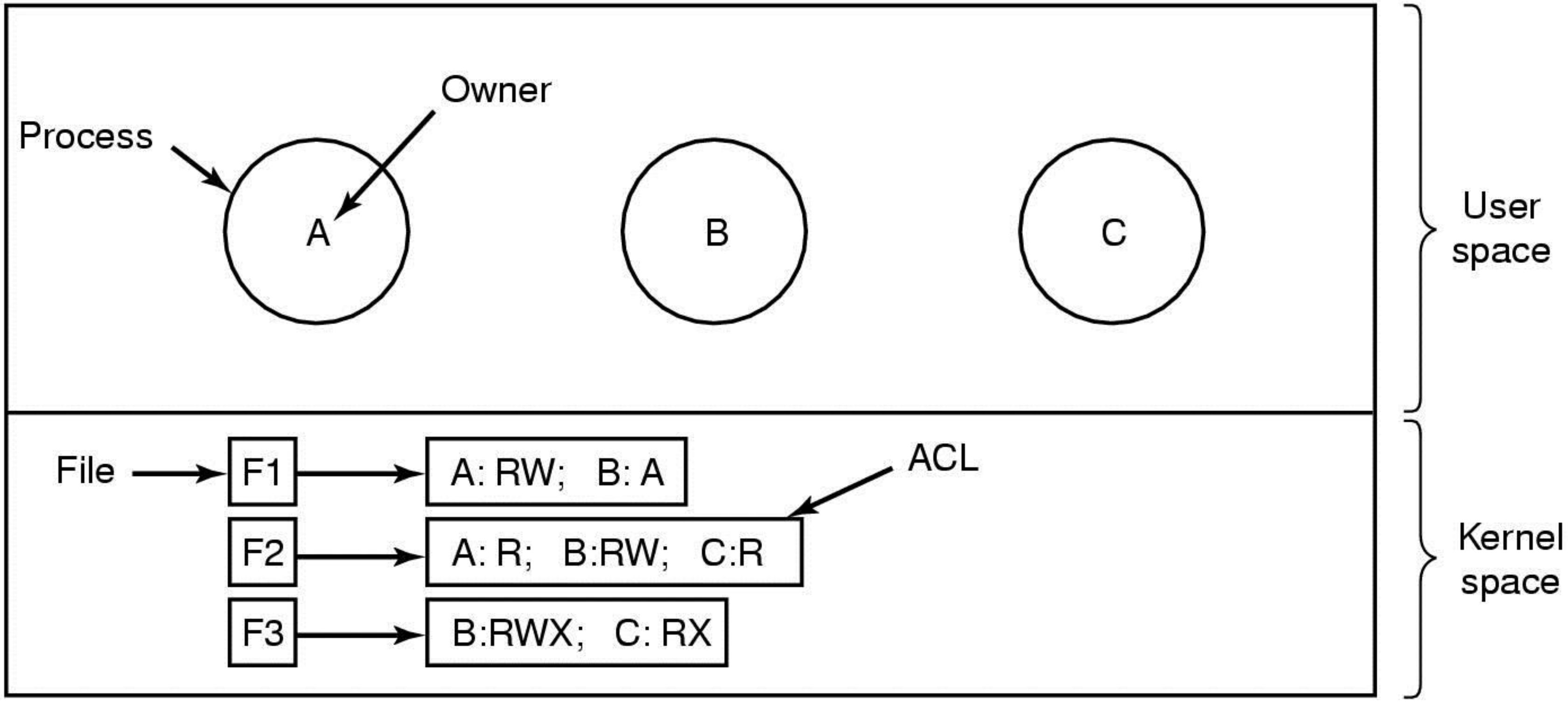
# Access Lists

- Each column in the access matrix is implemented as an access list for one Object.
- On invocation of a method $R_k$ on an object $O_j$ by a process $P_i$ running in a domain $D_h$,
  - the access is dereferenced to the object $O_j$,
  - the access list $\{< D_i, \{R_j\} >\}$ searched for $D_h$,
  - the methods are searched for an entry $R_k$.

# Access Lists

- Empty entries in Access Matrix can be discarded.
- Storage for access lists is proportional to the number of Objects
- A default can be associated with an access list so that any Domain not specified in the list can access the Objects using default methods.

# Access Control Lists



A: RW; B: A

A: R; B:RW; C:R

B:RWX; C: RX

Use of access control lists of manage file access

# Access Lists

- It is easy for the owner of the Object to grant access to another Domain or revoke access.
- It is easy to determine which processes can access which Objects.
- However, all processes can find out that the Object exists.

# Implementations of Access Lists

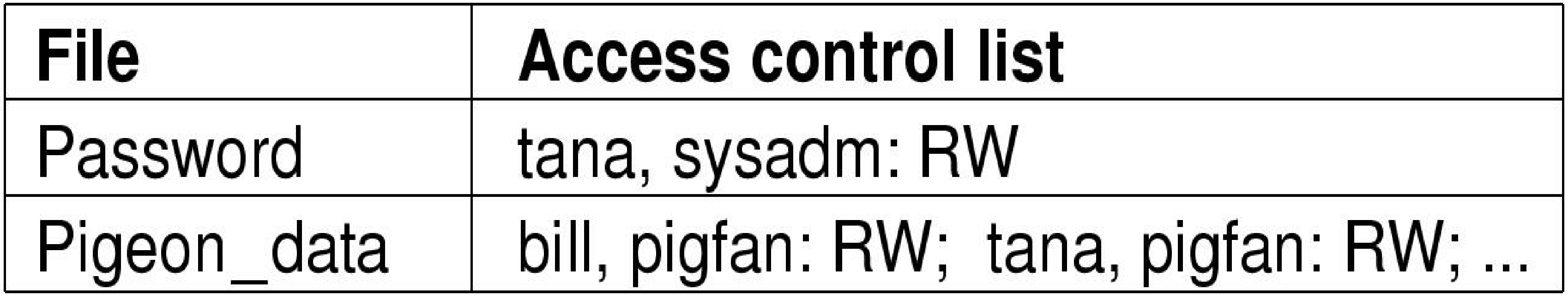
- File Systems
  - Opening a file is checked against an access list to determine if a process may open the file with a given set of access methods.
- Login Shells
  - The login to a system is checked against an access list (usually the password file owned by root).

# Implementations of Access Lists

- Rlogins are checked against an .rhost file that contains the names of machines from which a rlogin is permitted.

# Access Control Lists

- ACL’s allow different access privileges to be implemented for different roles of the same person.
- Example: Two access control lists

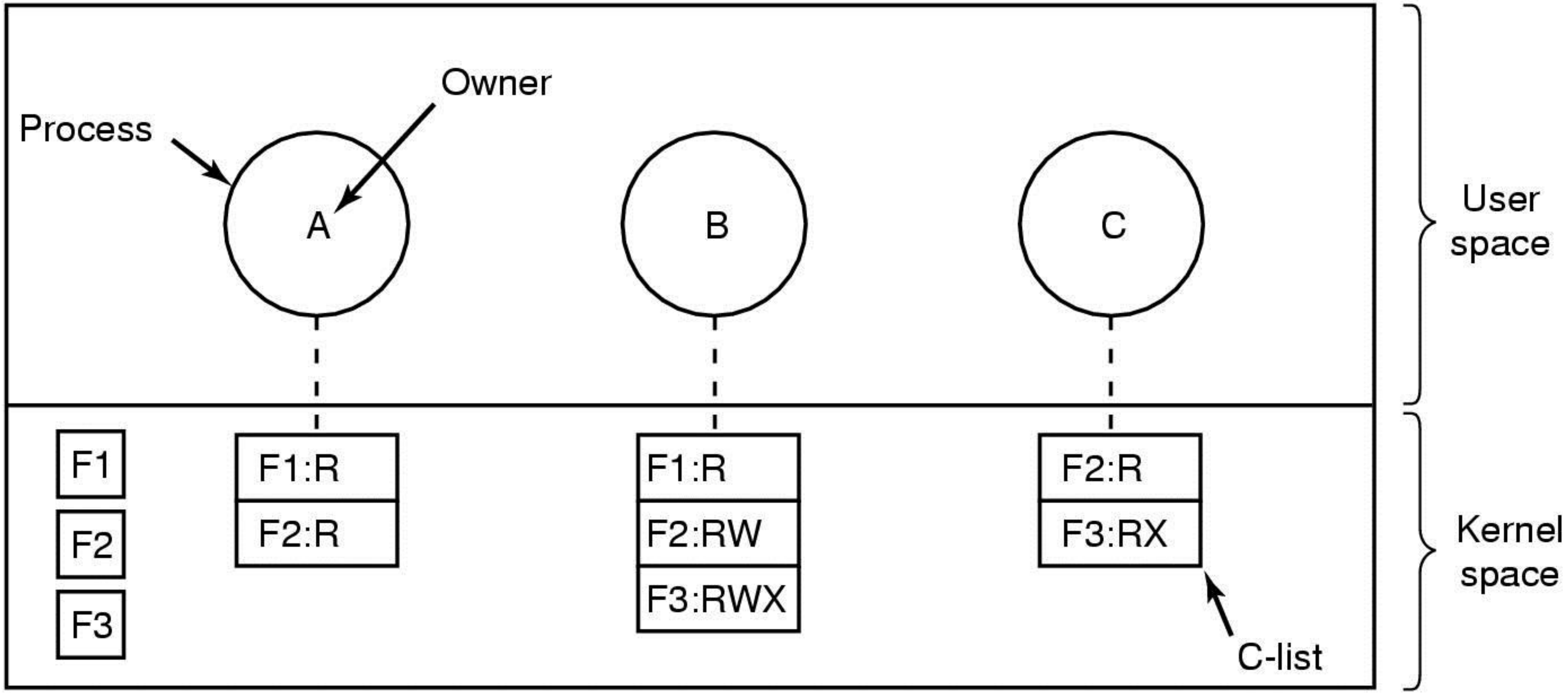


| File | Access control list |
|---|---|
| Password | tana, sysadm: RW |
| Pigeon_data | bill, pigfan: RW;  tana, pigfan: RW; ... |

# Capability Lists

- Each row in the access matrix is implemented as a capability list for each Domain.
- On invocation of a method $R_k$ on an object $O_j$ by a process $P_i$ running in a domain $D_h$,
  - the access list $\{< O_i, \{R_j\} >\}$ is searched for $O_j$,
  - the methods are searched for an entry $R_k$.

# Capabilities



Each process has a capability list

# Capability Lists

- Empty entries in Access Matrix can be discarded.
- Rather than search, a reference $< O_i, R_j >$ to an object can be treated as an index operation into the capability list.
- A capability is then just a "protected pointer".

# Capability Lists

- Having a capability to an Object is equivalent to having access permission for that object.
- A process executing in a Domain cannot modify the Domain's capability list because of security integrity.

# Capability Lists

- An application executing in a Domain can be provided just the capabilities it needs to execute its intended task - enforcing the Principle of Least Privilege.
- Processes cannot "look around" the system and see Objects they cannot access.

# How to protect capabilities

1. Tagged Architectures (IBM AS/400): Capability machines
   - Each memory word has an extra bit (tag) that indicates if the word contains a capability or not.
   - Usual computational operations (arithmetic, etc) do not use the tag.
   - Tags can be modified only by processes running in kernel mode (i.e., OS)

# How to protect capabilities

2. Keep a C-list in OS

- Refer to capability by their position in the list.
  - “read 1KB from file pointed to by capability 2”
- Example system using this method: Hydra

# How to protect capabilities

## 3. Encrypted capability

- Keep C-list in user space, but encrypt it so user cannot tamper with it.
- Suitable for distributed systems.
  – Client requests server to create object
  – Server creates object with randomly generated key and sends client capability:

| | | | |
|---|---|---|---|
| Server | Object | Rights | f(Objects,Rights,Check) |

  – Subsequent requests by client include sending  this capability to server along with request so rights can be checked.

# Capabilities

Examples of generic rights:

- Copy capability: create a new capability for the same object.
- Copy object: create a duplicate object with a new capability.
- Remove capability: delete an entry from the C-list; object unaffected.
- Destroy object: permanently remove an object and a capability.

# Capability Implementations

- Virtual Memory
  - A segment is a capability
  - It is protected from the user and can only be changed by the kernel running in supervisor state
  - It defines an object that can be accessed
  - Having the segment permits access.

# Capability Implementations

- UNIX File System
  - Each entry in the per process open file descriptor table is a capability.
  - It is protected and can only be changed by the kernel.
  - Having an open file descriptor permits access.
  - This example shows how access lists can be used to achieve simple management of protection and capabilities used to provide efficient access methods.

# Mixed Approaches: Locks and Keys

- Each Object has a list of unique bit patterns called locks
- Each Domain has a list of unique bit patterns called keys
- Processes cannot alter keys
- On access, the key used must match the lock.
- Allows implementation of concepts such as groups

# Locks and Keys

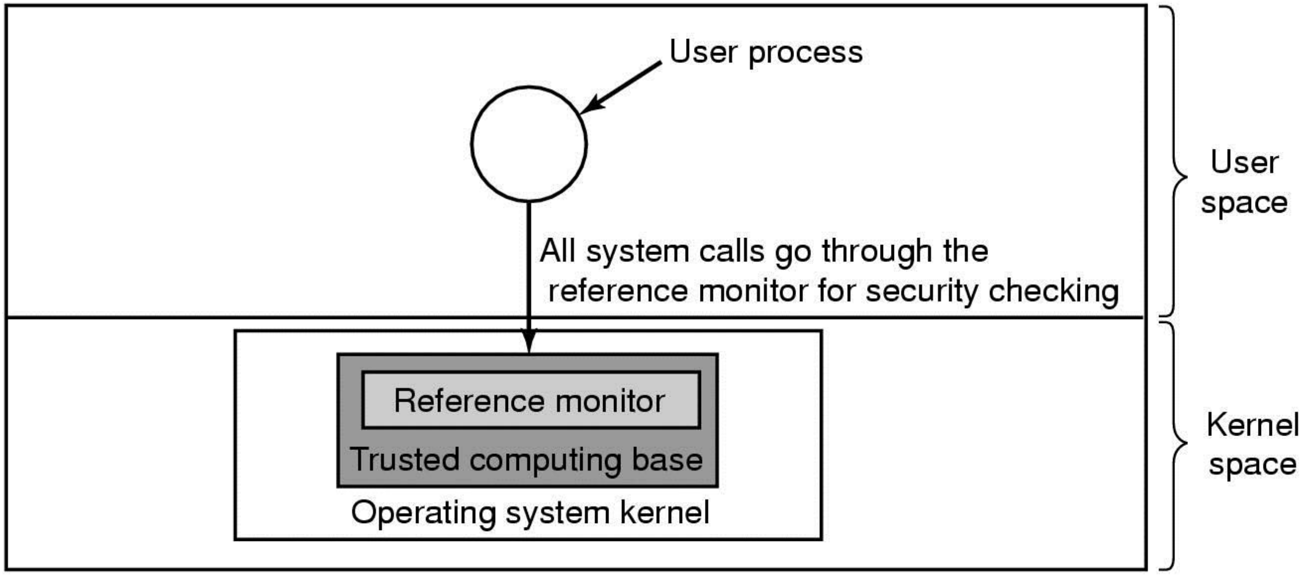
- Typically many different keys can fit a lock using a priority numbering scheme
- Key 3 fits all locks 3, 7, 15
  Key 4 fits 5,6,7,12,13,14,15
  - How is it implemented?

# Trusted Systems

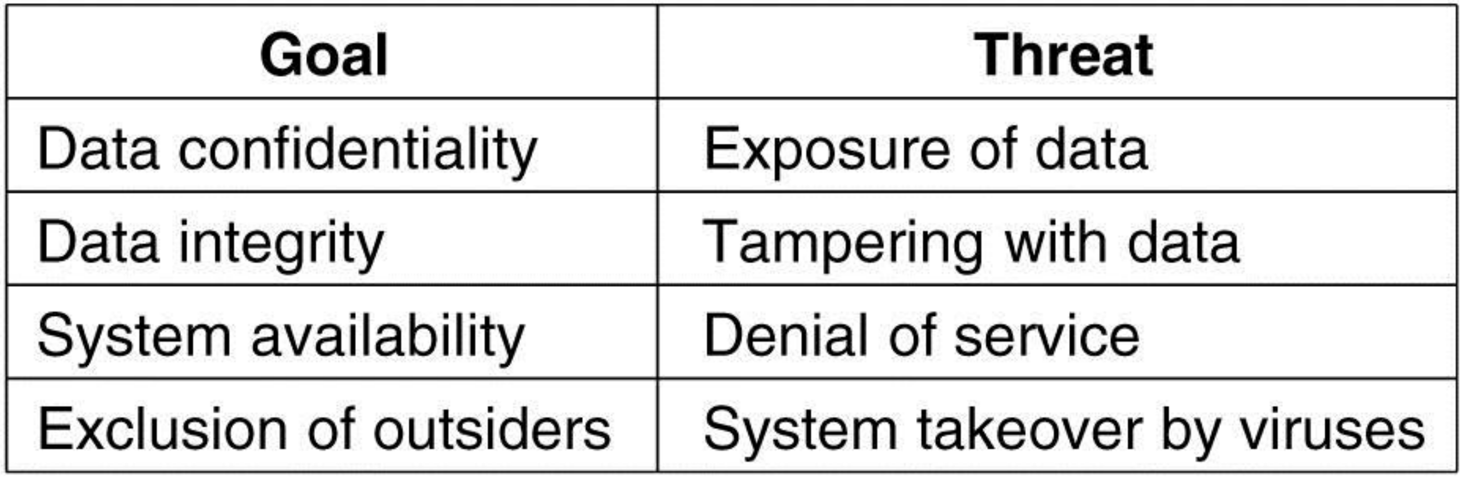
- In real world, one talks of trusted systems.
  - Trusted systems have formally stated security requirements and they meet these requirements.
  - Consist of minimal Trusted Computing Base (TCB).
    - HW and SW needed for enforcing security rules.

# Trusted Computing Base (TCB)

- At the core is “reference monitor”: accepts all syscalls related to security (open files, etc) and decide if they should be processed.
- All security decisions in one location of code/kernel



# The Security Environment Threats



| Goal | Threat |
|---|---|
| Data confidentiality | Exposure of data |
| Data integrity | Tampering with data |
| System availability | Denial of service |
| Exclusion of outsiders | System takeover by viruses |

Security goals and threats

# Accidental Data Loss

Common Causes

1. Acts of God/nature
   - fires, floods, wars
2. Hardware or software errors
   - CPU malfunction, bad disk, program bugs
3. Human errors
   - data entry, wrong tape mounted

# The Security Problem

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

# Intruders

Common Categories

1. Casual prying by nontechnical users
2. Snooping by insiders
3. Determined attempt to make money
4. Commercial or military espionage

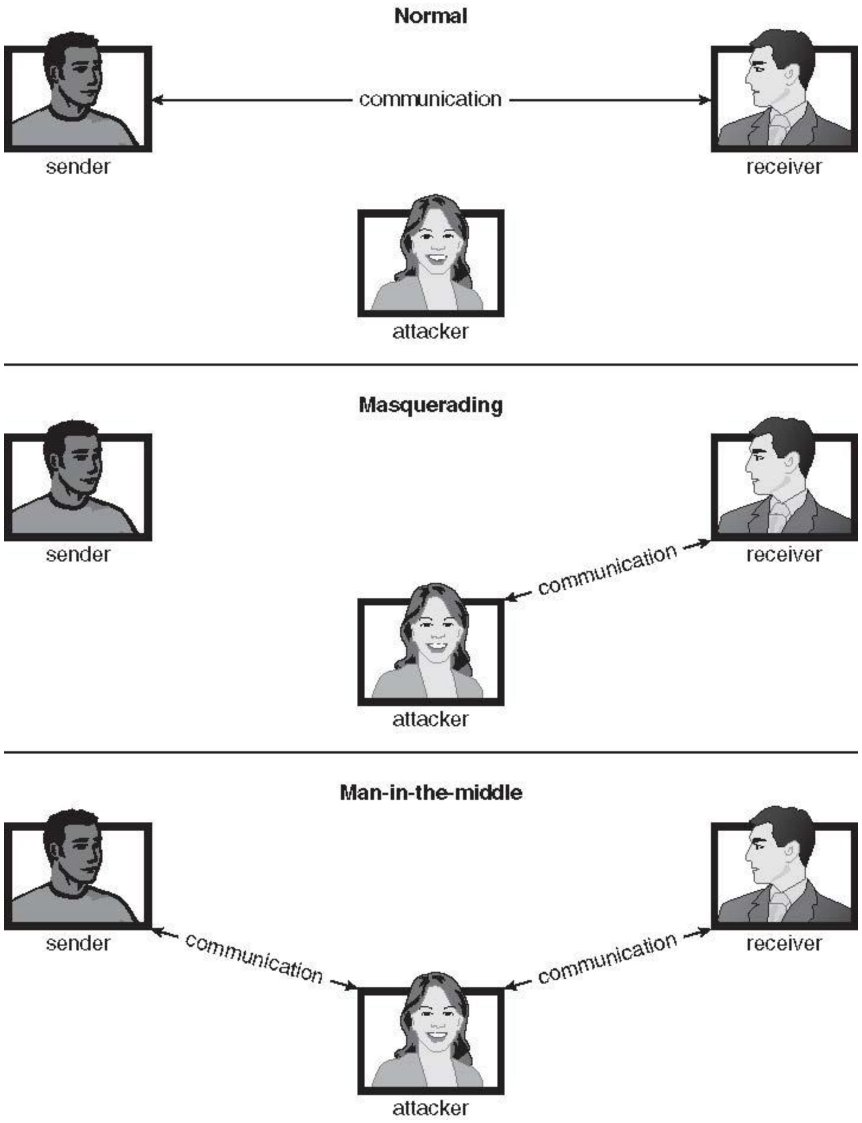
# Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use

# Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication

# Standard Security Attacks



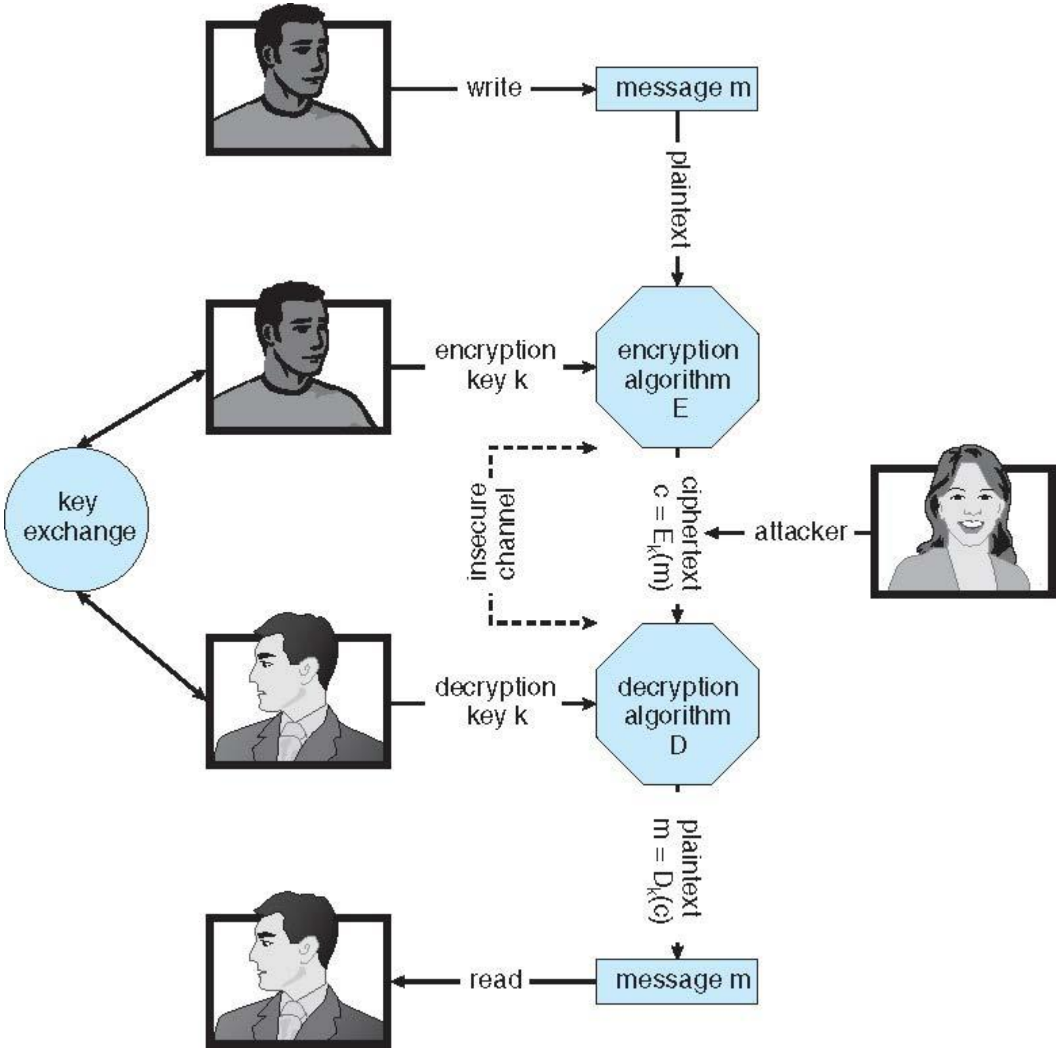
# Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

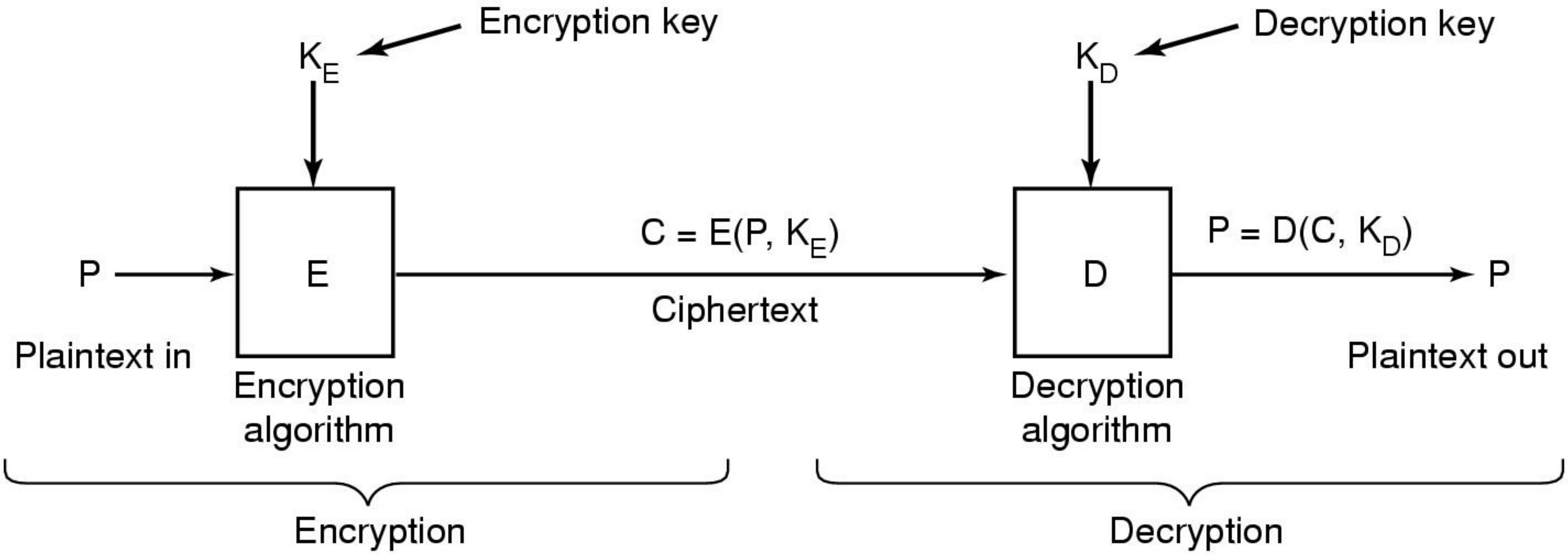
# Encryption

- A common method of protecting information transmitted over unreliable/unsecured links.
  - Clear text → Encryption → Ciphertext
  - Ciphertext → Decryption → Clear text

# Secure Communication over Insecure Medium

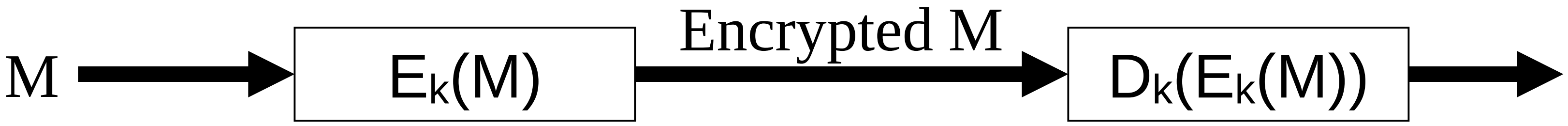


# Basics of Cryptography



Relationship between the plaintext and the ciphertext

# Encryption



M

$E_k(M)$

Encrypted M

$D_k(E_k(M))$

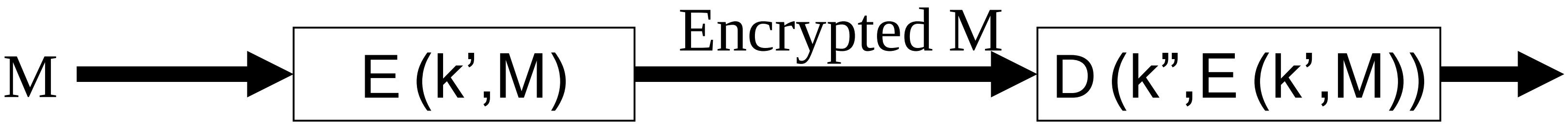
E - Encryption Algorithm

D - Decryption Algorithm

k - key(s)

M- Message (clear text)

Alternative notation (public, private keys)



M

E (k’,M)

Encrypted M

D (k”,E (k’,M))

# Secret-Key Cryptography

- Monoalphabetic substitution:
- Plaintext:
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Ciphertext:
  QWERTYUIOPASDFGHJKLZXCVBNM

# Public-Key Cryptography

- Encryption makes use of an “easy” operation, such as how much is 314159265358979 × 314159265358979?

- Decryption without the key requires you to perform a hard operation, such as what is the square root of 3912571506419387090594828508241?

# Encryption

- An encryption algorithm satisfies:
  - $D_k\left(E_k(M)\right) = M$
  - Both $E_k$ and $D_k$ can be computed efficiently. The security of the system depends only on the security of the key, and not on the security of the algorithms *E* and *D*.
  - Usually, *E* and *D* are published and publicly available.

# Encryption Systems

- Public Key Systems
- Secure Secret Key Systems

# Public Key Systems

- Use two keys: $k_{pub}$- public key which is published by the user and $k_{priv}$ - private key.
- Where $k_{priv} \neq k_{pub}$
- The holder of $k_{priv}$ can send an authenticated message to anyone because they can read the message using $k_{pub}$.
- $m = D\left(k_{priv}, E\left(k_{pub}, m\right)\right)$

  or

  $m = D(k_{pub}, E\left(k_{priv}, m\right))$

# Public Key Systems

- Example: B, C, D can all encrypt message for A using A's public key. If B encrypted message with A's public key, then C cannot decrypt it even if C knew that it was encrypted with A's public key. Need A’s private key to decrypt.

# Secure Secret Key Systems

- Use single key, called secret key which is shared between encryptor and decryptor (shared key).
- These systems are called Symmetric Systems.
- Example: Data Encryption Standard DES is an example of a shared key.

# Symmetric Systems

- Advantages:
  - Symmetric systems provide a two-way channel to their users;
  - As long as the key remains secret, the system also provides authentication.
- Problems:
  - If the key is revealed, then interceptor can immediately decrypt and encrypt information.

# Symmetric Systems

- Distribution of keys: Before communication takes place, the secret keys must be sent securely to both the sender and receiver.
  - Couriers are used to distribute keys
  - Split key in pieces and distribute pieces under separate channels.

# Symmetric Systems

- Number of keys: because it increases in square with number of people (Solution: Use Clearing house or forwarding office).

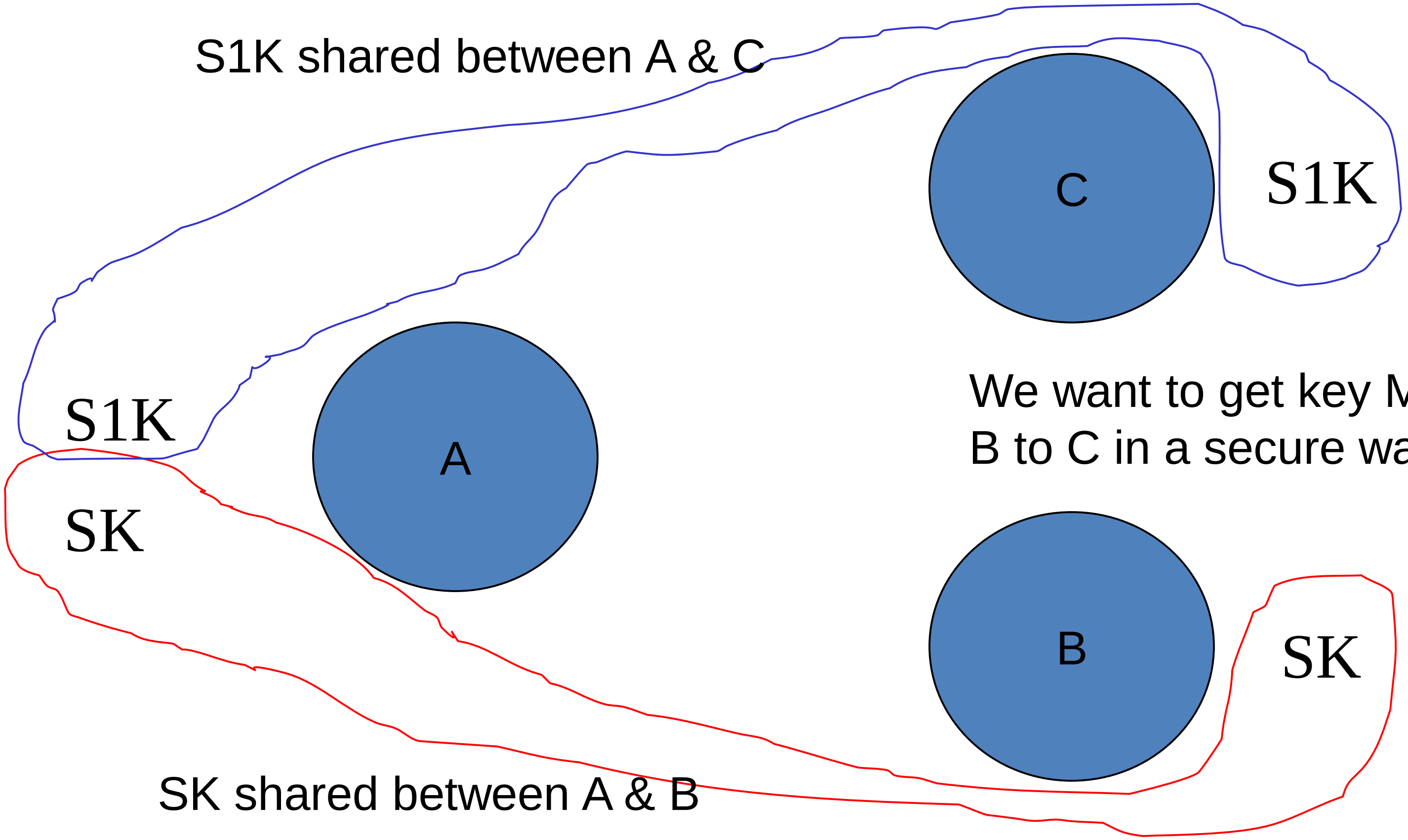
# Asymmetric Keys

- Advantages:
  - Authentication –knowledge of private key
  - Privacy – use public key of recipient
  - Key distribution easy
- Problems:
  - Slower than symmetric keys

# Distribution of Keys using Symmetric Keys: Key exchange between B & C

A: trusted server

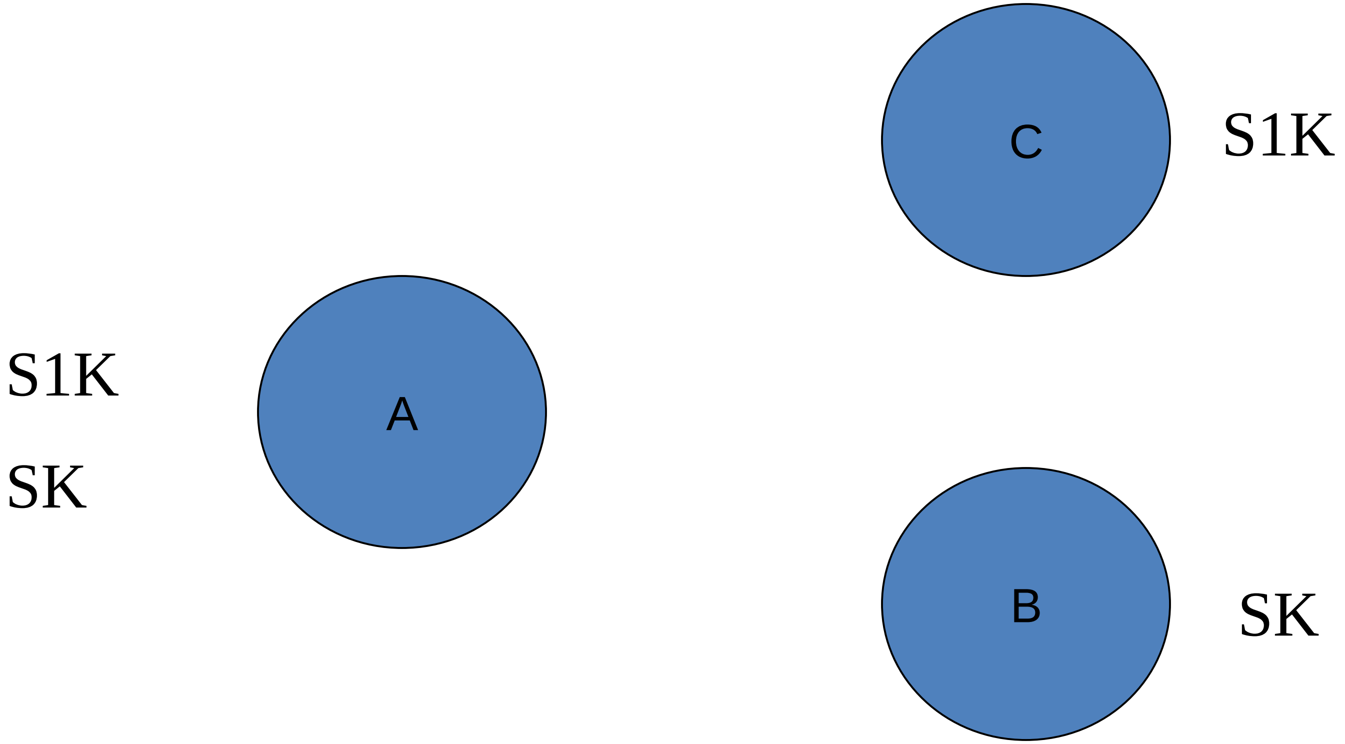
S1K shared between A & C



We want to get key MK from B to C in a secure way

SK shared between A & B

# Distribution of Keys using Symmetric Keys



C S1K

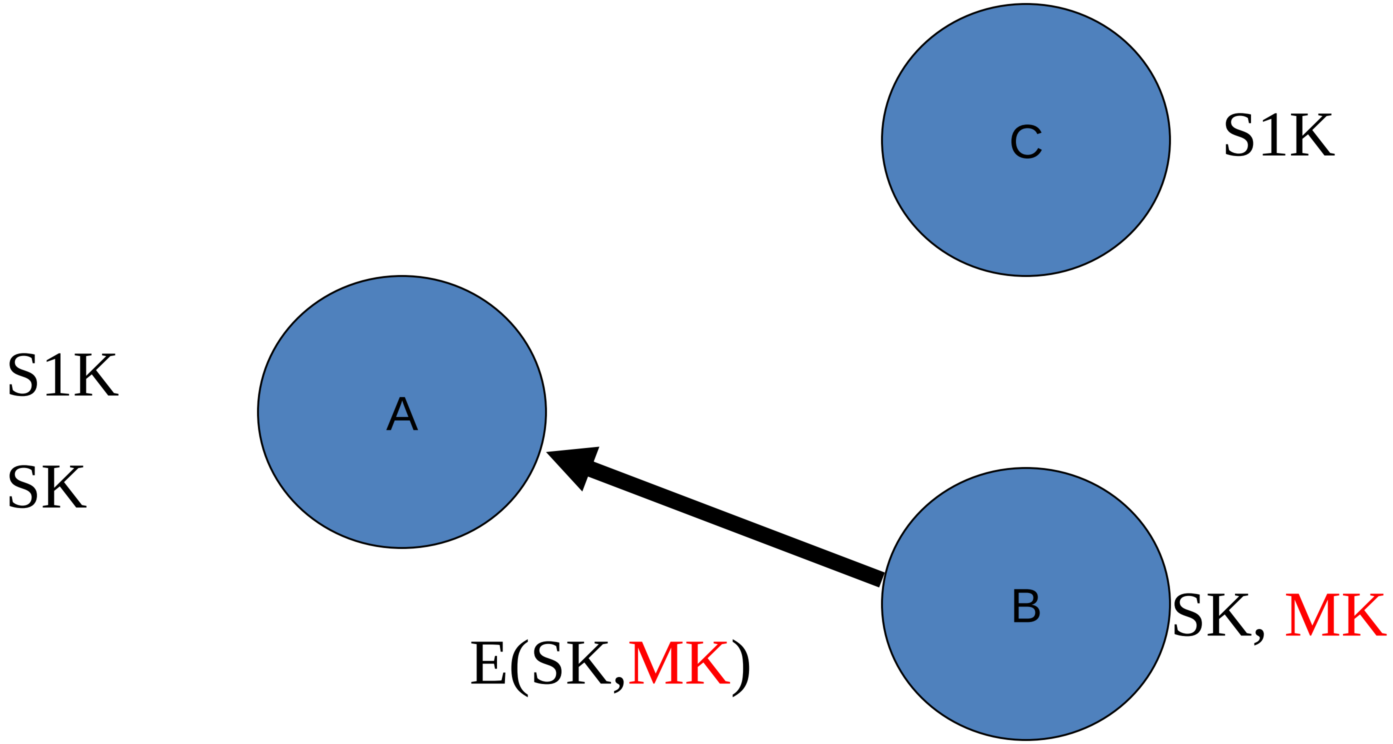
S1K

A

SK

B SK

# Distribution of Keys using Symmetric Keys



S1K

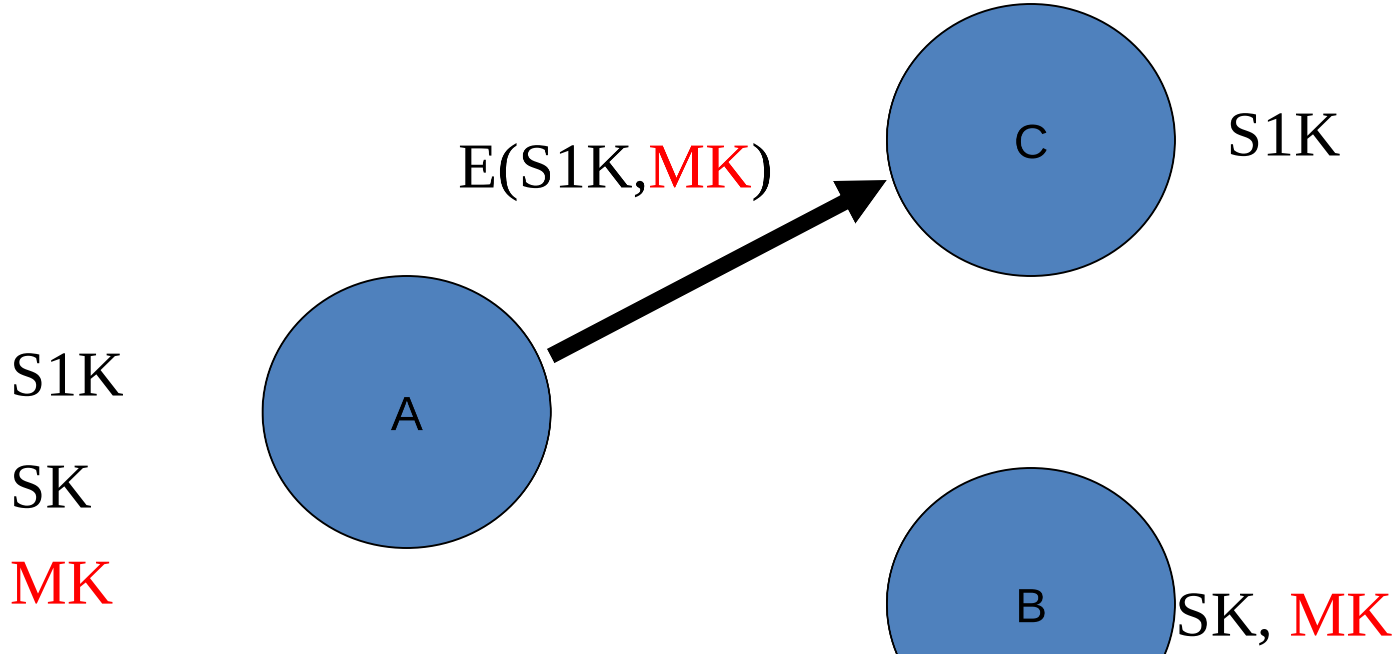
S1K

SK

SK, MK

E(SK,MK)

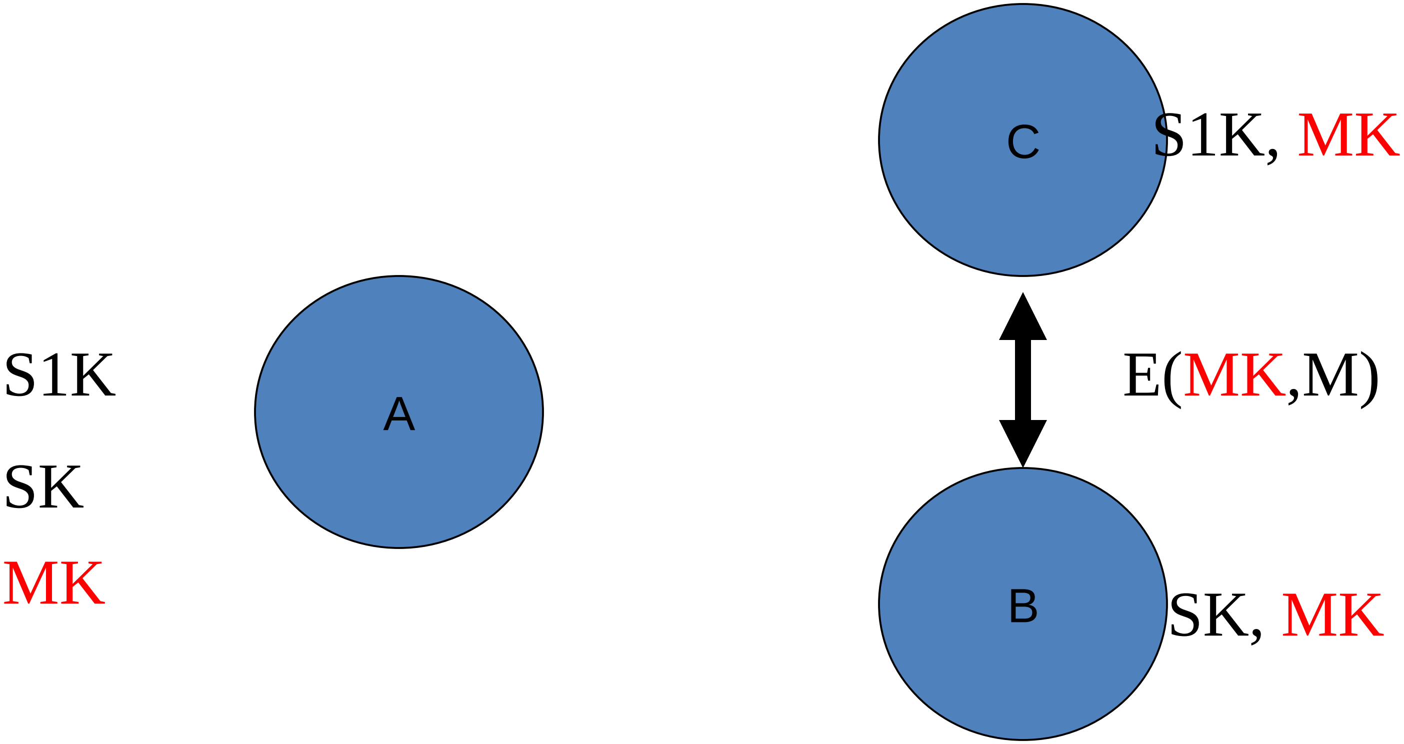
# Distribution of Keys using Symmetric Keys



A decrypts E(SK,MK) and now has MK

# Distribution of Keys using Symmetric Keys

C now has MK, can encrypt message M to send to B



# One-Way Functions

- Function such that given formula for $f(x)$
  - easy to evaluate $y = f(x)$
- But given $y$
  - computationally infeasible to compute

$$x = f^{-1}(y)$$

  - RSA (Rivest-Shamir-Adelman) method uses this principle.

# Rivest-Shamir-Adelman (RSA) Encryption

- RSA uses two keys: $d$ and $e$ with integer $n$. Hence, pair $(e, n)$ will be the public encryption key; pair $(d, n)$ will be the private key.
- Message $m$ is encrypted as follows:

$$E(m) = (m^e) \bmod n = C,$$

- Message $m$ will be decrypted as follows:

$$D(C) = \left(C^d\right) \bmod n$$

# RSA Encryption

- It is true: $\left((m^e)^d\right) mode\, n = m$
- $n$ is computed as product of two large primes $n = p \cdot q$ (100 digits or larger)
- $d$ is random integer such that: greatest common divisor $\gcd\left(d, (p-1) \cdot (q-1)\right) = 1$
- $e$ satisfies $e \cdot d \; mod \; (p-1) \cdot (q-1) = 1$
- Key $n$ is publicly known, its factors are difficult to calculate (i.e., one way function).

# RSA Encryption

- Simple Example: $e$=11, $n$=35, and let's assume $m$=3;
- $E(m) = m^{11} mod\ 35, D(C) = C^{11} mod\ 35.$
- Assume $p = 5, q = 7$. Then $\gcd(d, 24) = 1$ and $11 \cdot d\ mod\ 24 = 1$ are satisfied if $d = 11$.
- $E(3) = 12 = C, D(12) = 3.$
- Another example: $D(5) = 10, E(10) = 5.$

# Private and Public Keys

- Given the public key system, if a trusted server exists in the system, then it can be used to provide access to other services securely.
- The server can receive secure messages from clients by publishing its ``public key''.
- The clients encrypt using the public key, but no other client can decrypt the message.

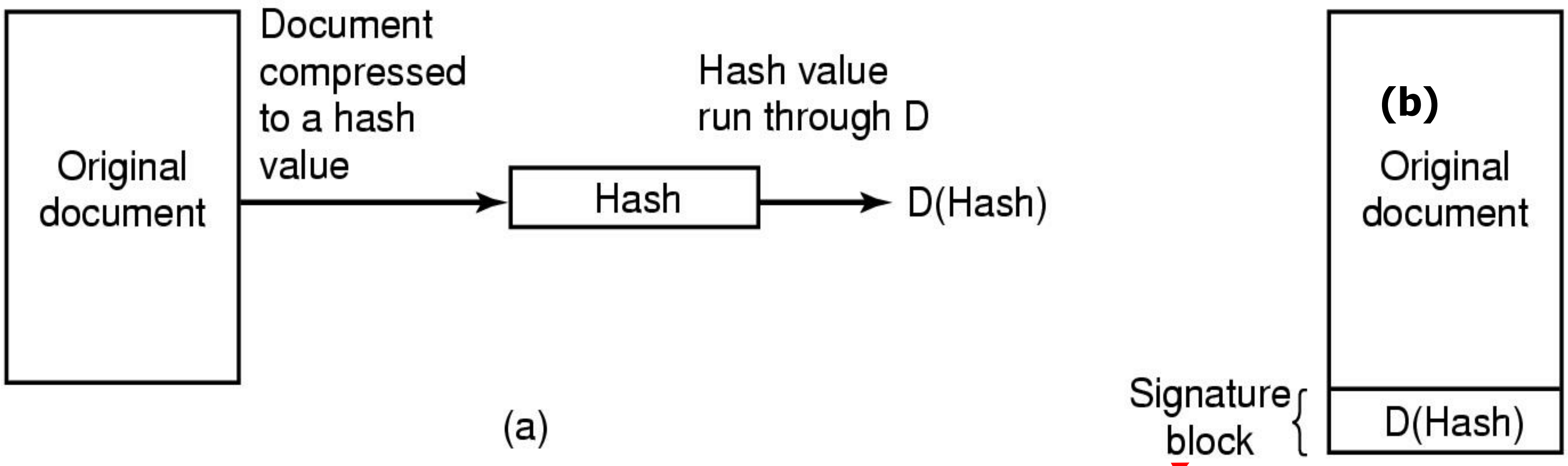
# Private and Public Keys

- The server can sign messages to the clients by encrypting the message with its private key. No other client can send such a message. All clients can decrypt the signed message and know it comes from the server.
- The server can sign a message in the clear by sending a check sum of the message encrypted with its private key. Clients can authenticate the message by computing its checksum and comparing it with the encrypted value received.

# Digital Signatures

a) Computing a signature block

b) What the receiver gets



Receiver authenticates this

# Key Exchange using Asymmetric Keys

- Given the public key system, if a trusted server exists in the system, then it can be used to provide access to other services securely.
- Send an encrypted message to the trusted server TS using its public key and include a public key for the client requesting access to service.

# Encryption and Authentication

- Similarly for servers S, send a message to the trusted TS server including an S server public key
- The TS trusted server encrypts the S server public key with the clients key and sends a message containing it back to the client

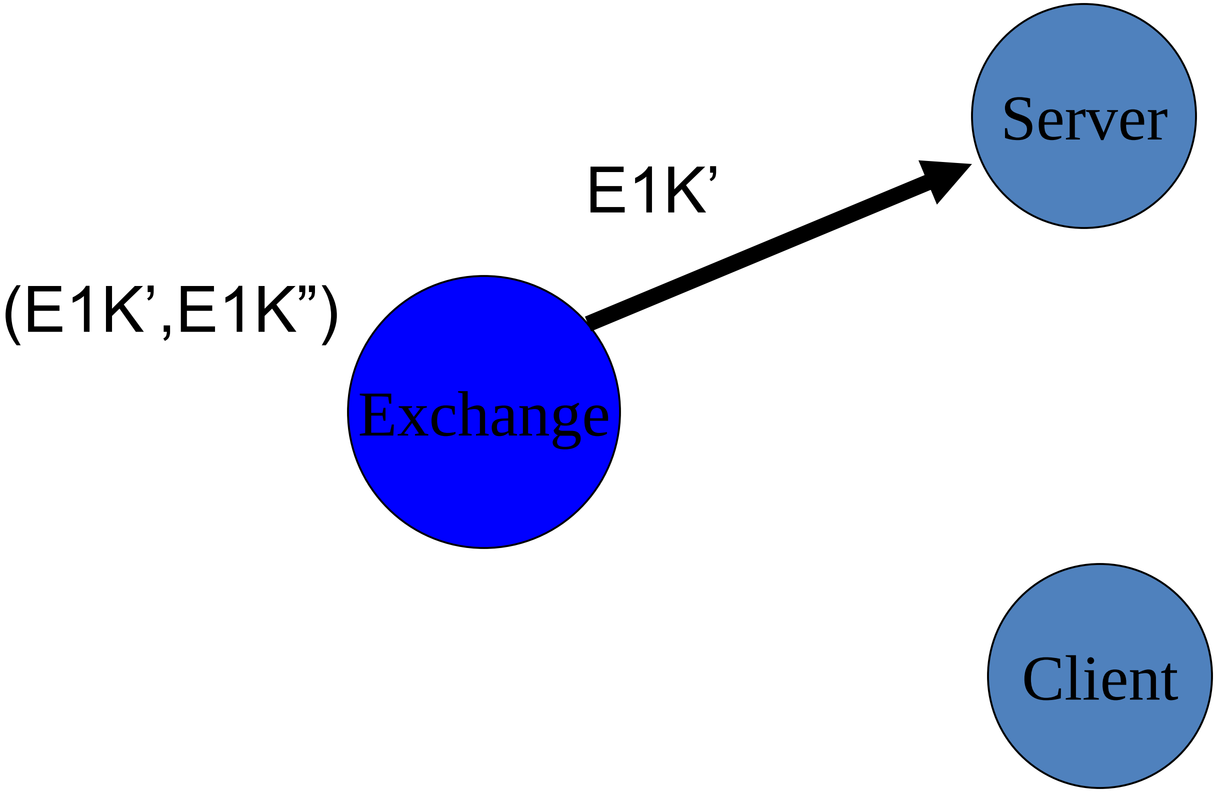
# Encryption and Authentication

- Only the client can decrypt the message to get the public key of the server S. It knows it came from the TS trusted server
- The client can now encrypt a message and send it directly to the S server with the S server's public key, only known to the client and TS trusted server
- For safety, the client sends a public key to the S server for it to use in reply.

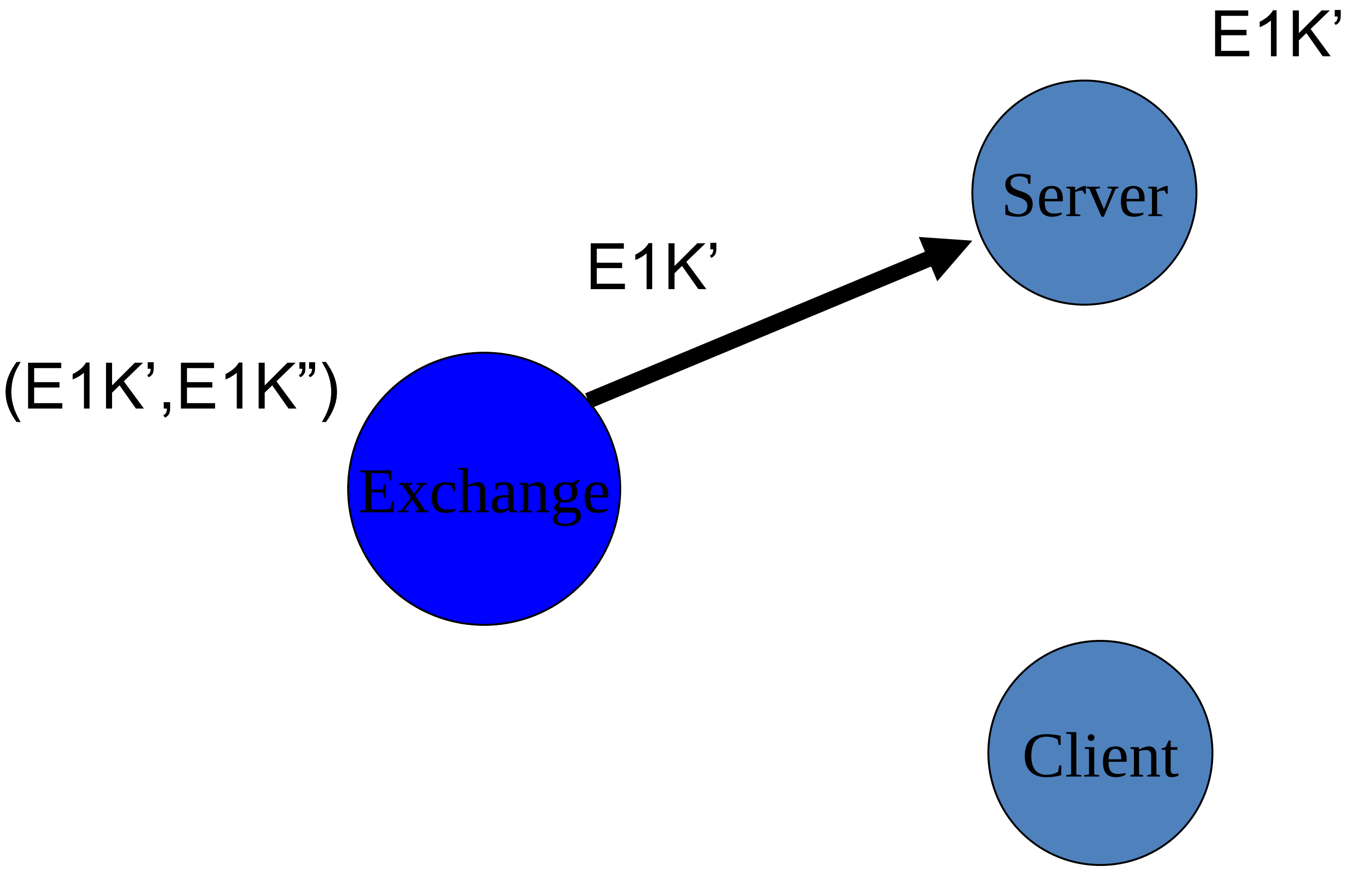
# Key exchange

- Key exchange example
  - Between server and client
  - Via trusted server exchange.

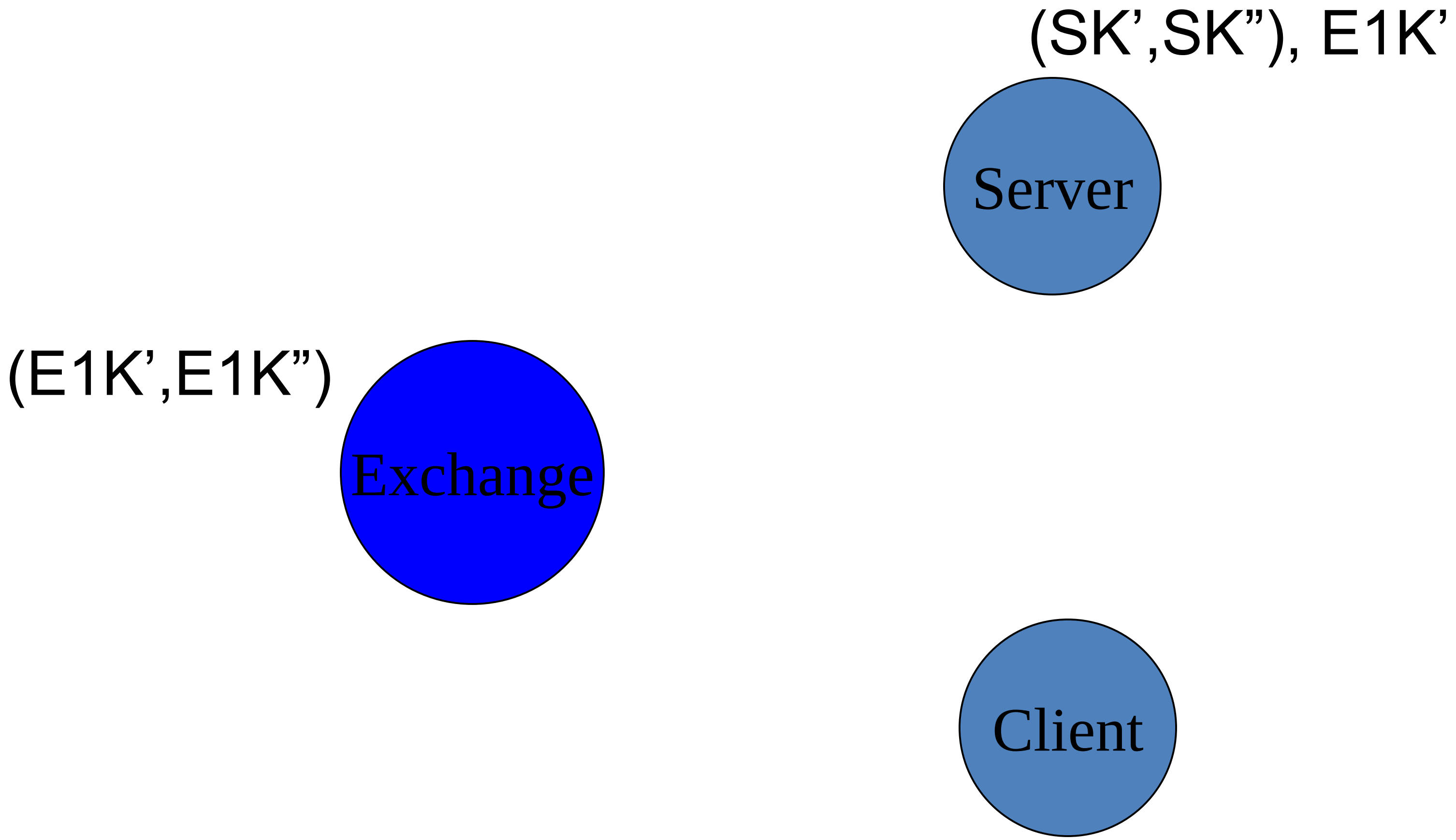
# Key Exchange



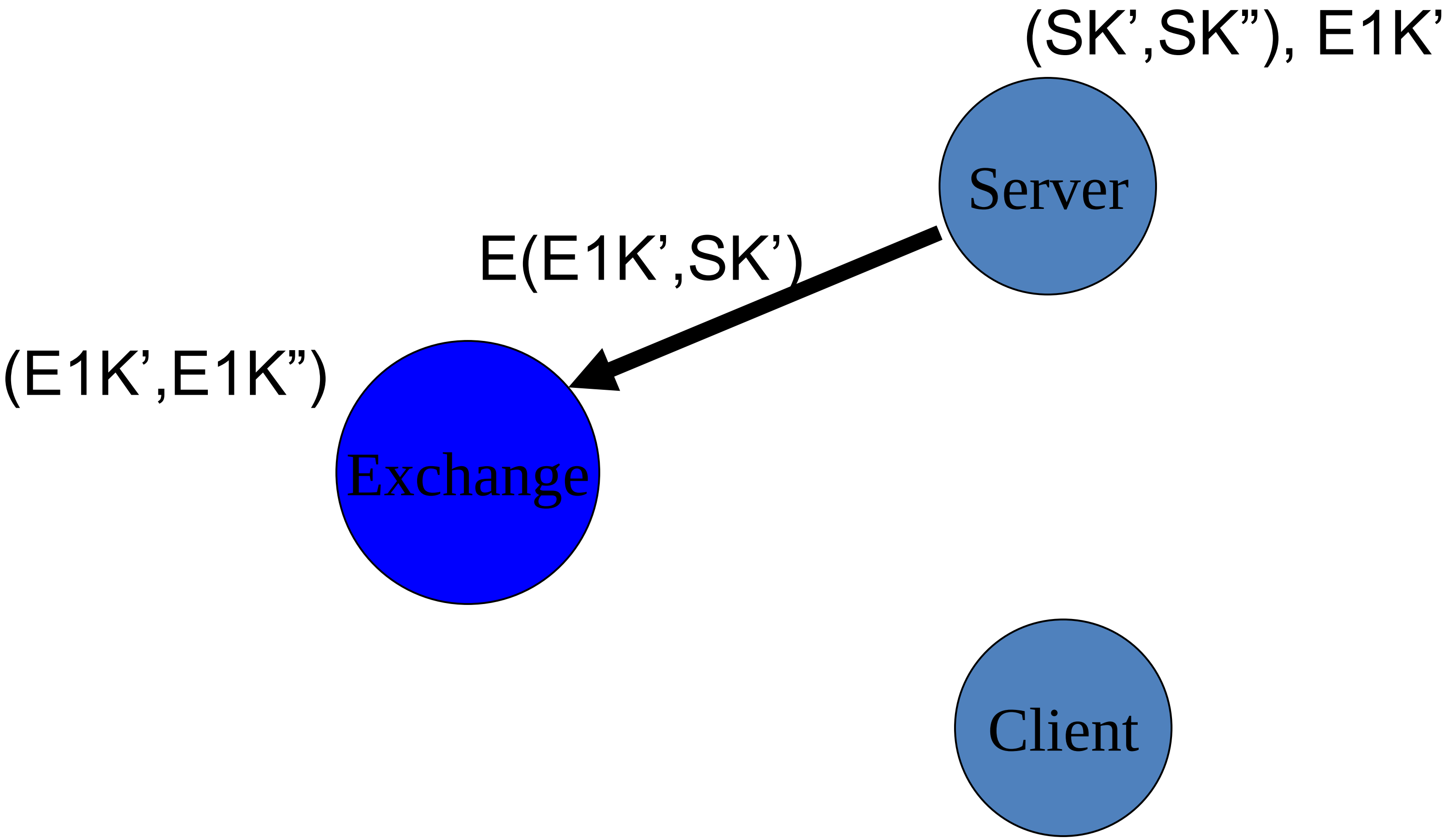
# Key Exchange



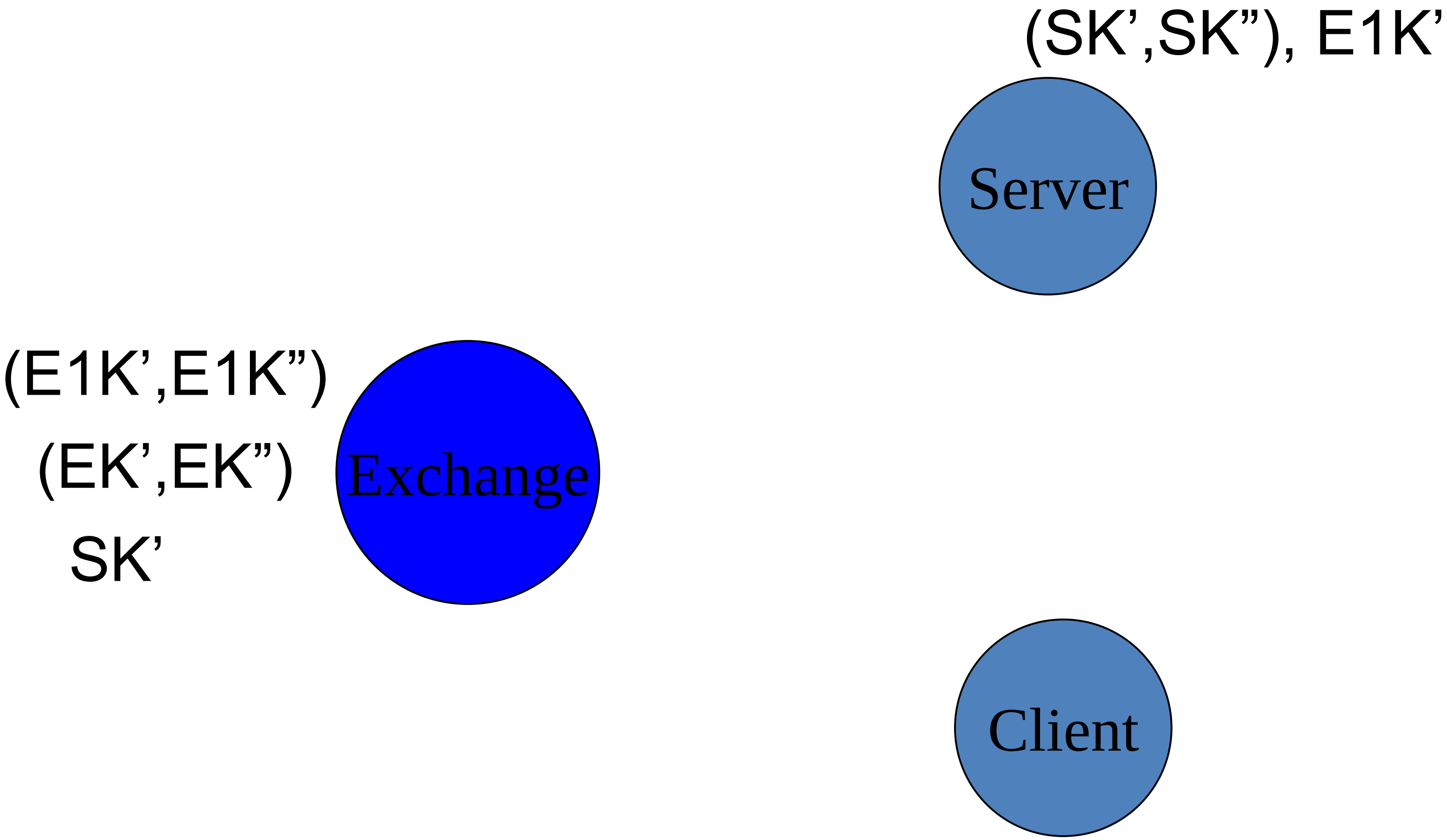
# Key Exchange



# Key Exchange



# Key Exchange



(SK’,SK”), E1K’

Server

(E1K’,E1K”)

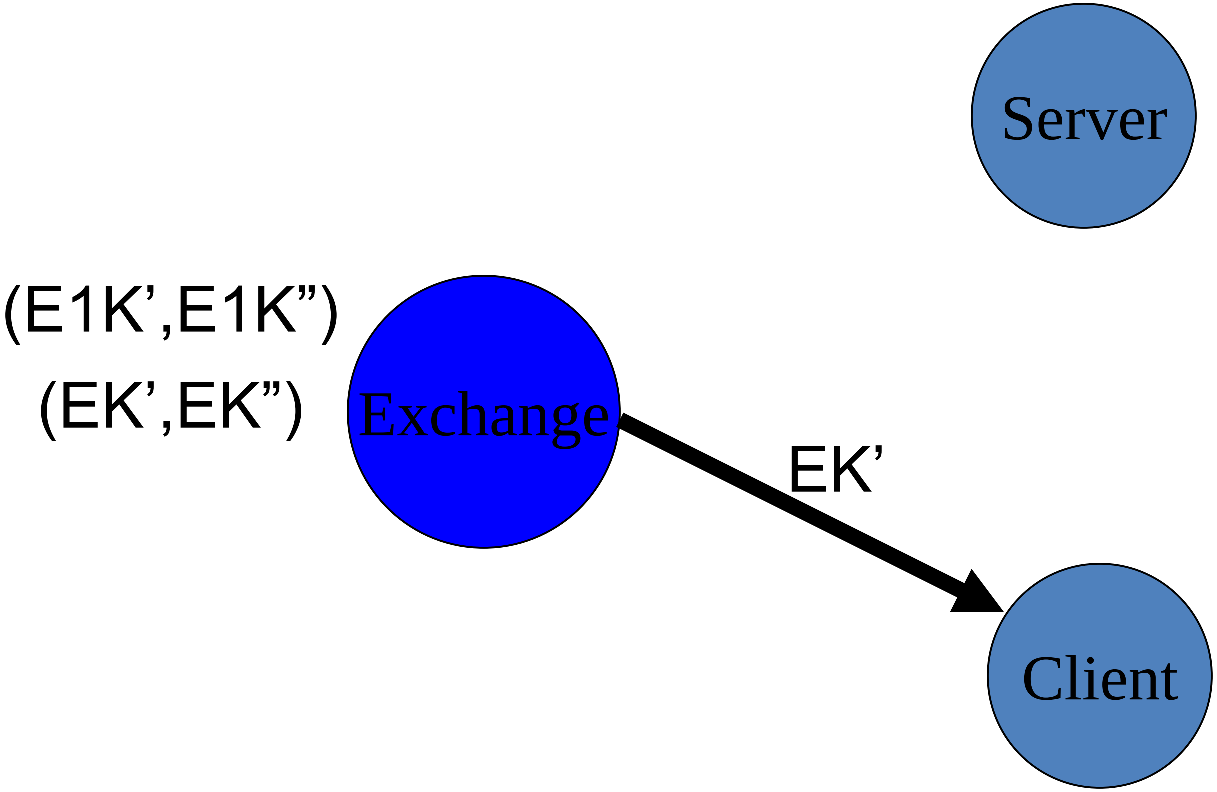
(EK’,EK”)

SK’

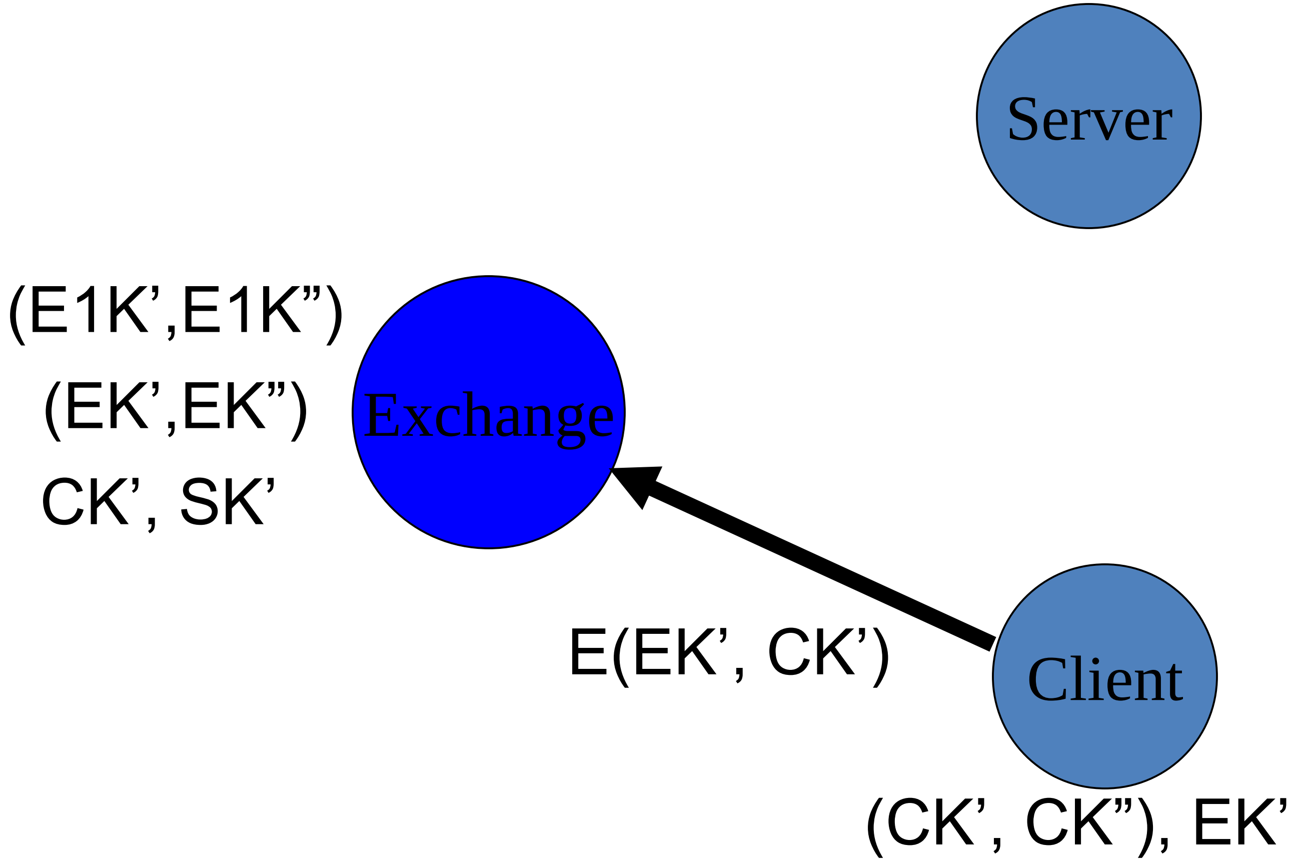
Exchange

Client

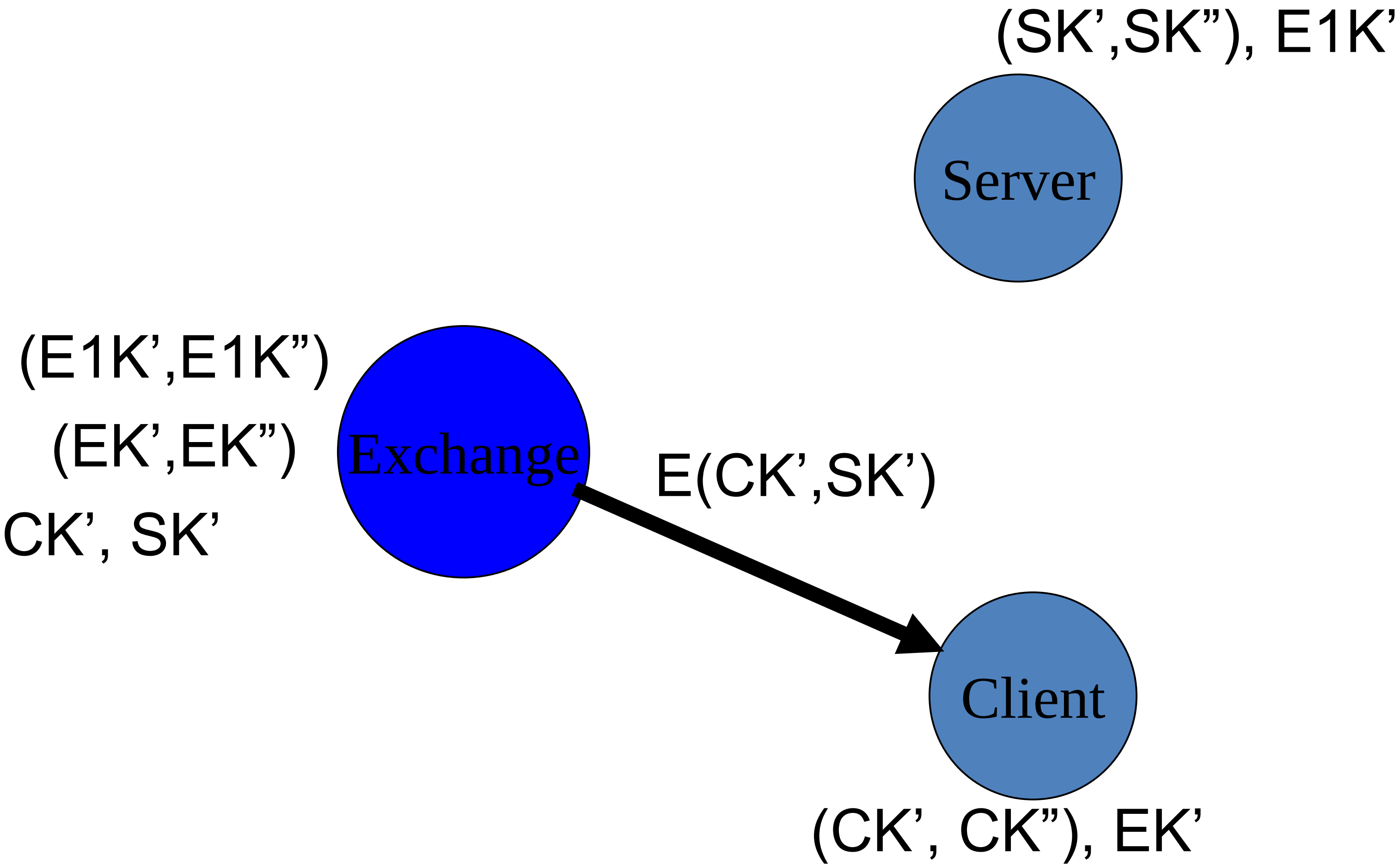
# Key Exchange



# Key Exchange



# Key Exchange



(SK’,SK”), E1K’

Server

(E1K’,E1K”)

(EK’,EK”)

CK’, SK’

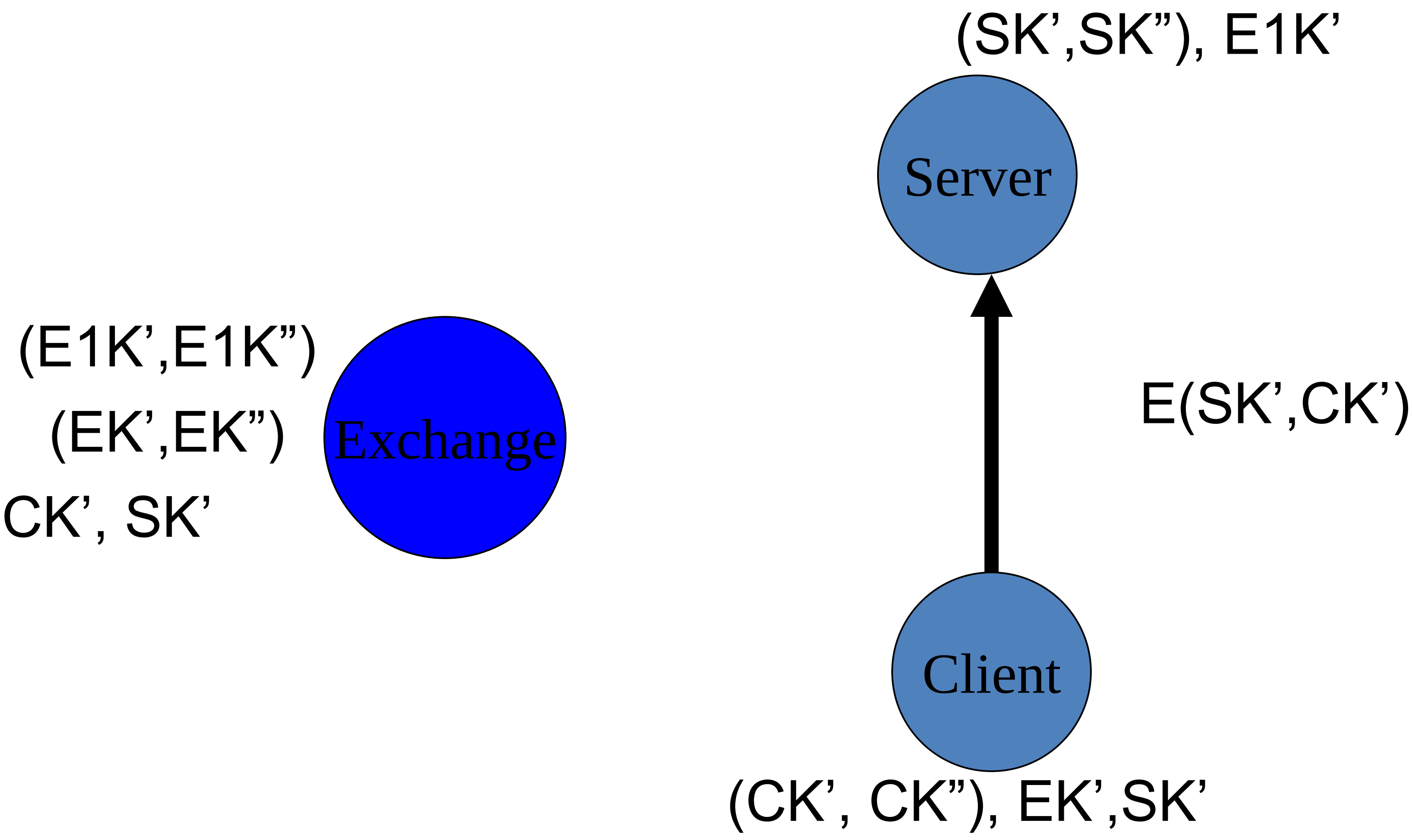
Exchange

E(CK’,SK’)

Client

(CK’, CK”), EK’

# Key Exchange



(SK’,SK”), E1K’

Server

E(SK’,CK’)

Client

(CK’, CK”), EK’,SK’

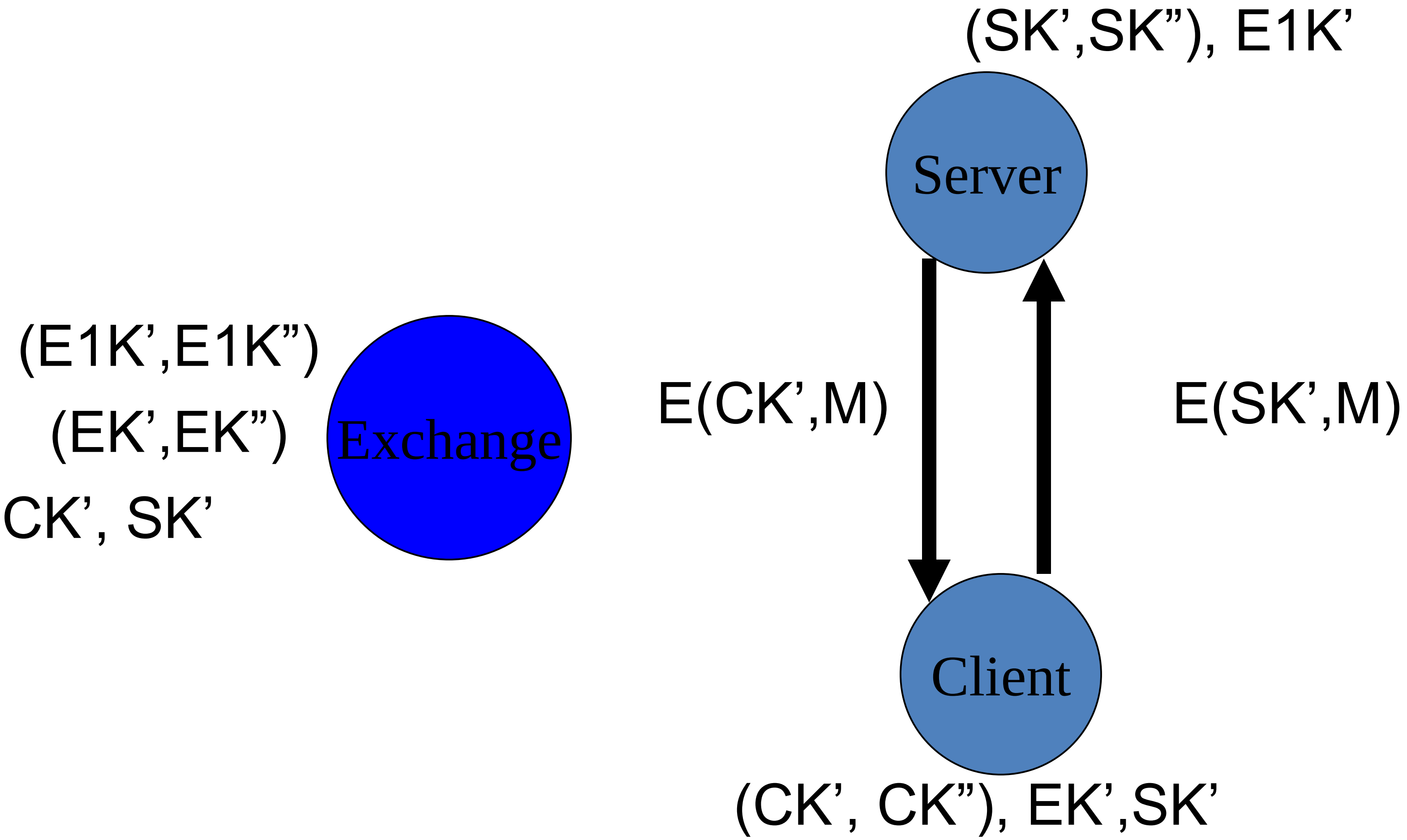
(E1K’,E1K”)

(EK’,EK”)

CK’, SK’

Exchange

# Key Exchange



# Issues

- Any client can replay an earlier communication, even if the client cannot decode the message in the communication.
- Some mechanism is needed to make sure communications are fresh! A signed timestamp or message count is often used.

# Issues

- The server has no way of knowing who a client is.
- The client needs to have some password or secret it can show to the server. A signed message indicating who the client is and using a different public key for the signature is often used.

# Distribution of Shared Keys Using Public Keys

- A client may have a two way secure communication with a trusted server if the trusted server publishes a public key.
- The client sends a shared key to the server, encrypted with the public key.
- The client and server may exchange secret messages using the shared key.

# Issues

- Once again freshness and client identification is an issue.
- Also, using the same key over and over again may allow other clients to decode the messages.

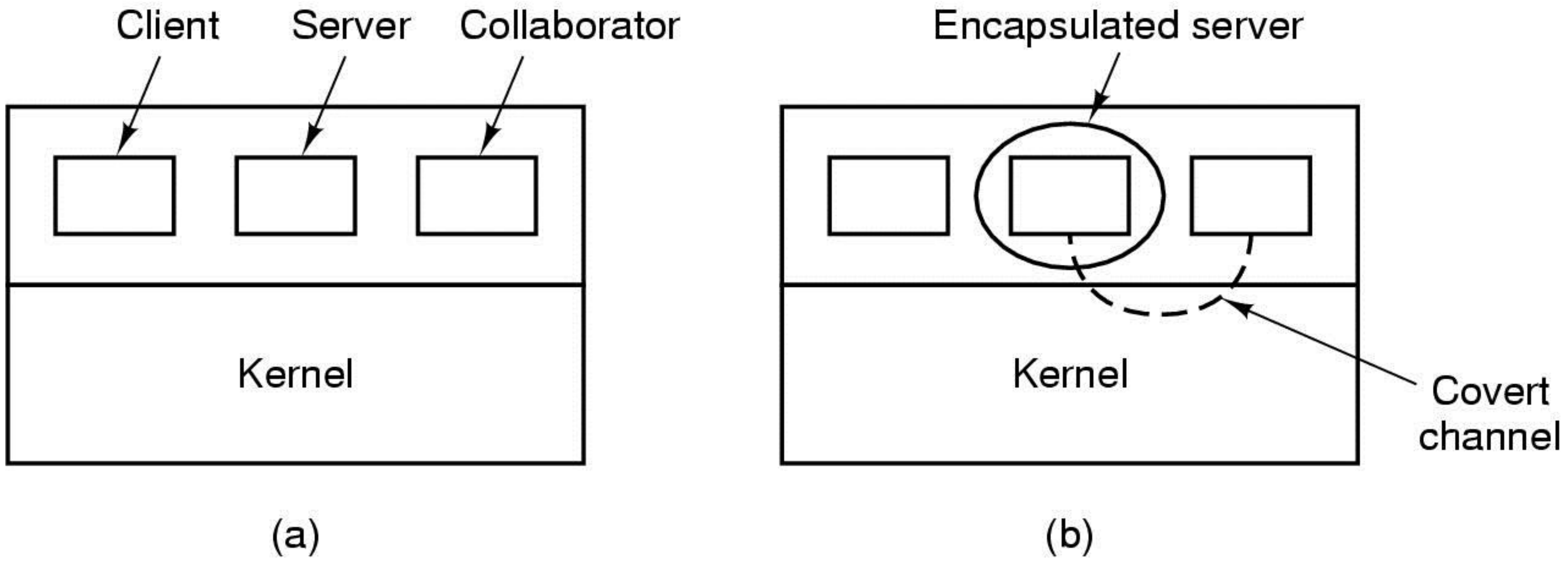
# Setting up private communications between clients

- A client may establish two way secure communication with another client through a server
- The client sends a shared key to the server using the server's public key
- The other client sends a shared key to the server using the server's public key
- If the first client wants to talk to the second client, the server sends the first client's shared key to the second client

# Issues

- Once again, freshness and client identification is an issue.
- Also, both the server and the other client know the key.
- It is better to use a third key for the communication rather than the two shared keys used to communicate with the server.

# Covert Channels



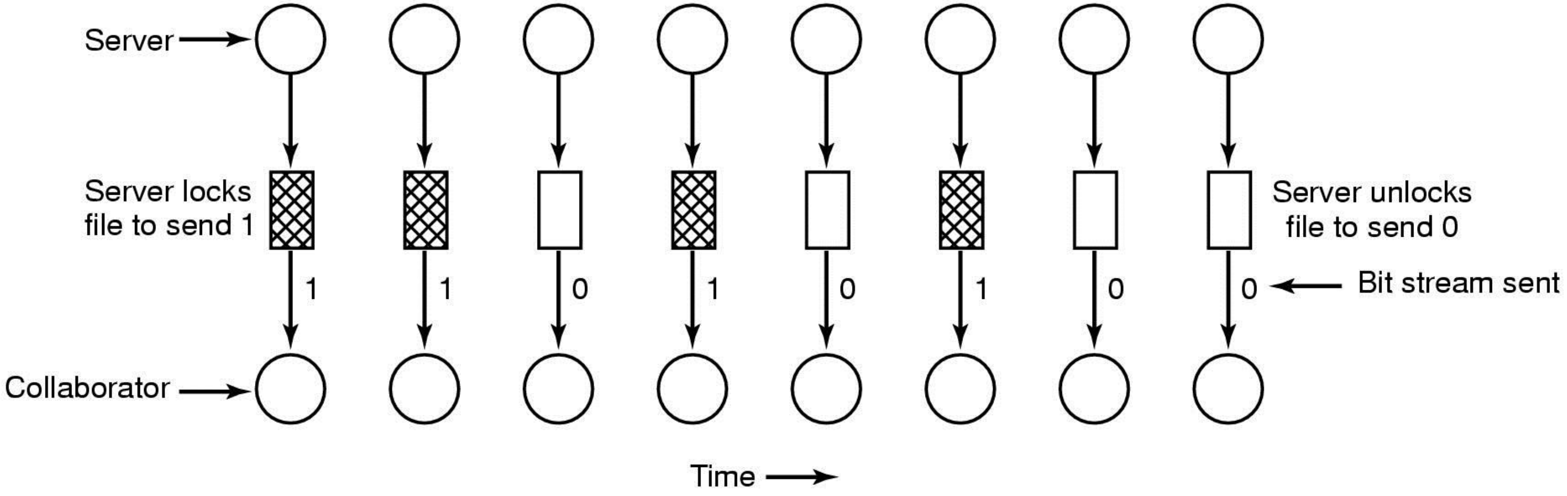
System designer encapsulates server to protect information

Covert channels: subtle means of communication

- (a) The client, server, and collaborator processes. (b) The encapsulated server can still leak to the collaborator via covert channels.

# Covert Channels

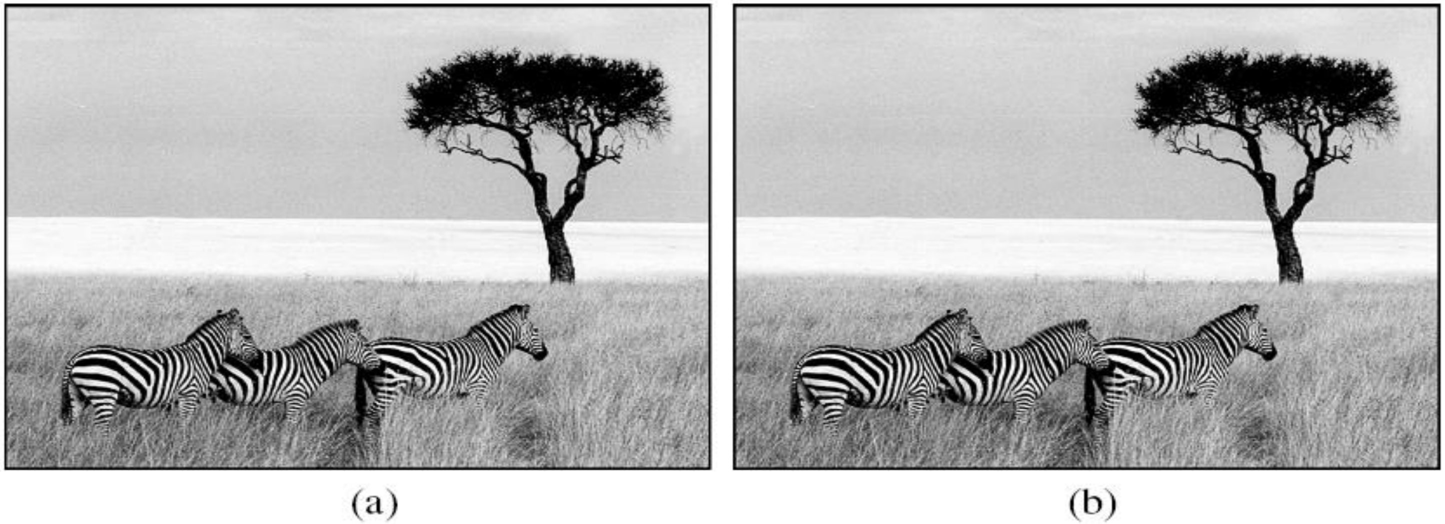
- A covert channel using file locking.



- Any timed performance degrading scheme can be used as a covert channel: many page faults: 1, no page faults: 0

# Covert Channels

- Steganography: hiding messages in images.



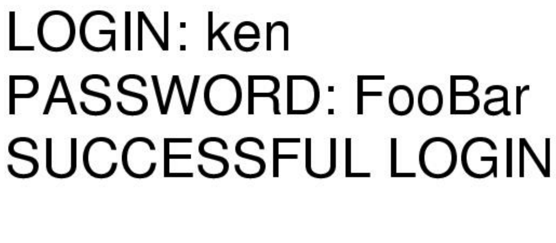
- (a) Three zebras and a tree. (b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

# User Authentication

- Basic Principles. Authentication must identify:
- Something the user knows
- Something the user has
- Something the user is

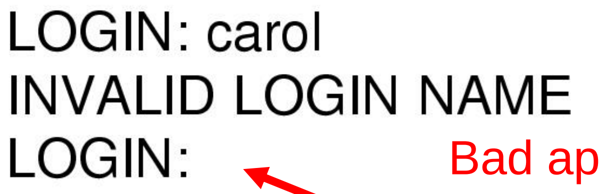
- This is done before user can use the system

# Authentication Using Passwords



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

(a)

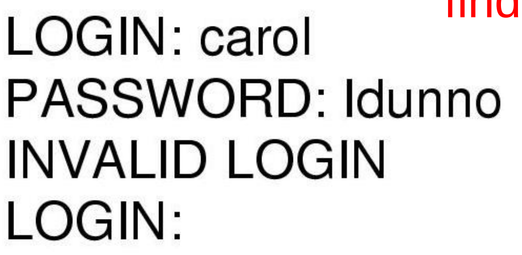


```
LOGIN: carol
INVALID LOGIN NAME
LOGIN:
```

(b)

Bad approach. Attacker can try different login names until he finds a valid one.

- (a) A successful login
- (b) Login rejected after name entered
- (c) Login rejected after name and password typed

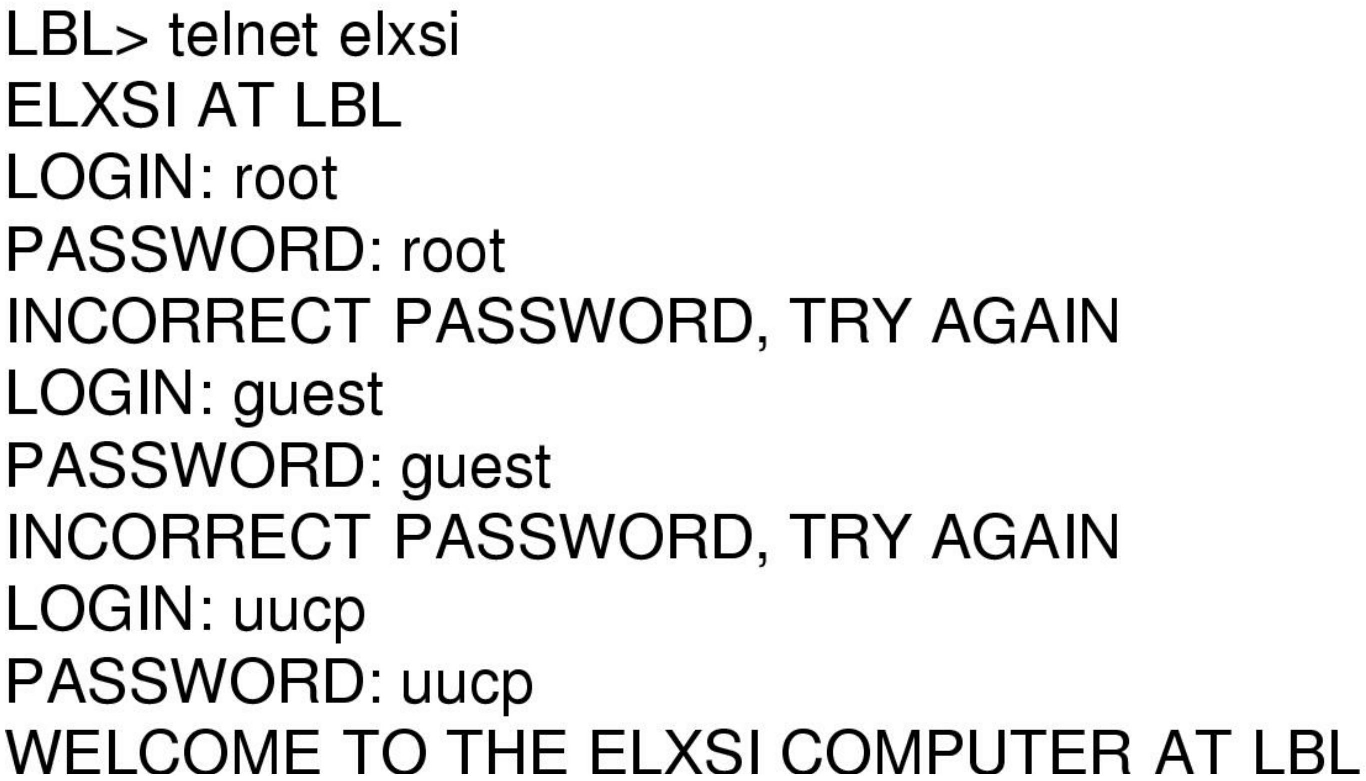


```
LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:
```

(c)

here, the attacker doesn’t know if the username or password is not valid → more work for him.

# Authentication Using Passwords



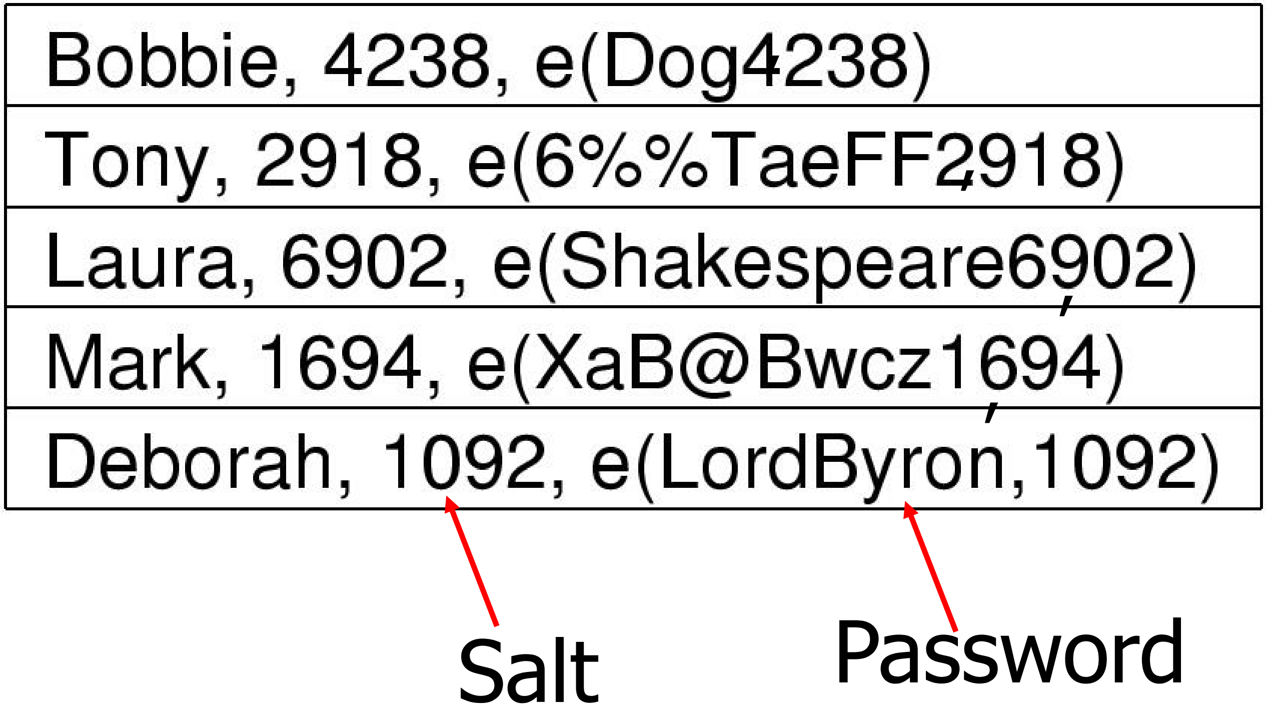
```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

- How a cracker broke into LBL
  – a U.S. Dept. of Energy research lab

# Use of salt with passwords

- Salt: an n-bit random number used in conjunction with password.
- Concatenate user password with salt and then encrypt. Store this value in password file.
- Hacker needs to do more work to break the password: now he needs not only to guess pw but also pw0000, pw0001, …

# Authentication Using Passwords



| |
|---|
| Bobbie, 4238, e(Dog4238) |
| Tony, 2918, e(6%%TaeFF,2918) |
| Laura, 6902, e(Shakespeare,6902) |
| Mark, 1694, e(XaB@Bwcz,1694) |
| Deborah, 1092, e(LordByron,1092) |

Salt
Password

The use of salt to defeat precomputation of encrypted passwords

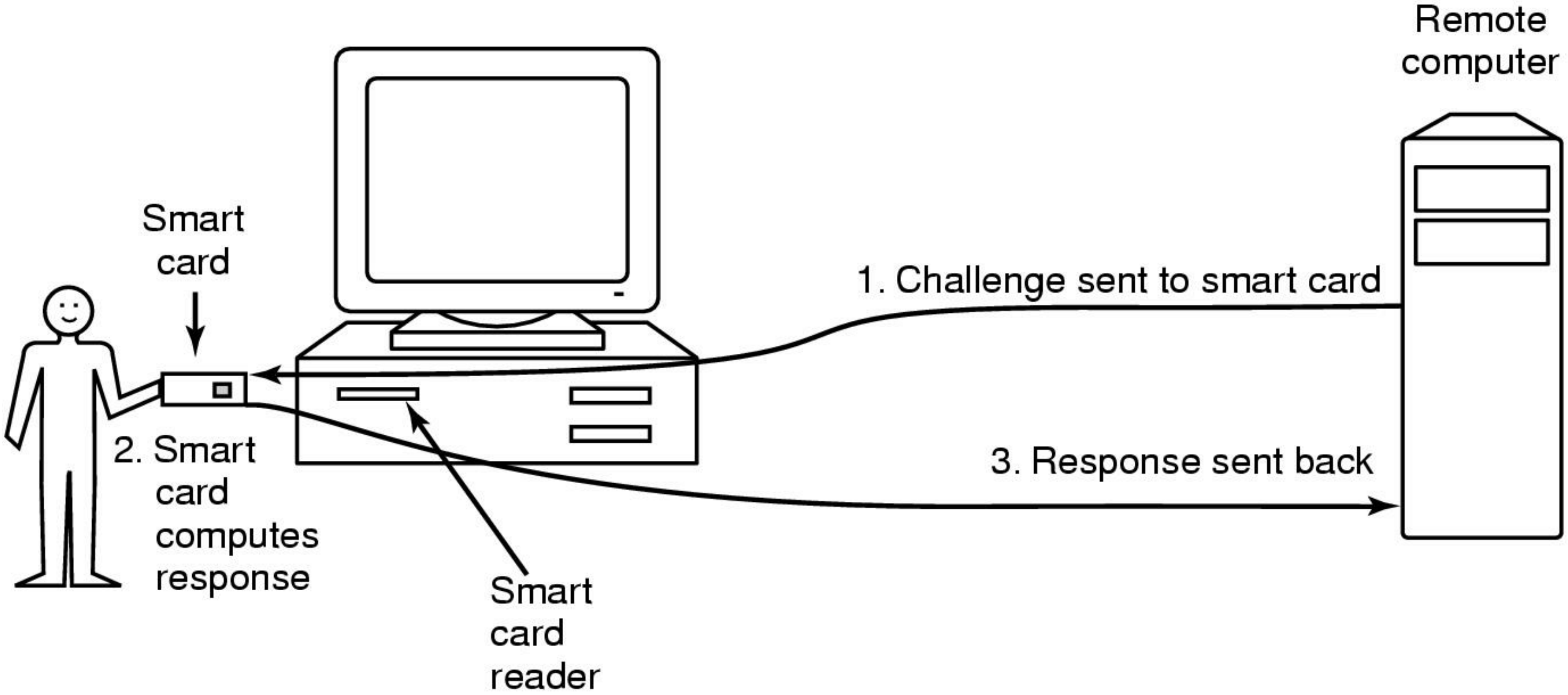
# One-time passwords

- A list of passwords generated ahead of time
- Use next password from the list and throw away.
- Eavesdroppers cannot use the password captured from wiretapping.

# Challenge-Response Authentication

- The questions should be chosen so that the user does not need to write them down.
- Examples:
- Who is Marjolein's sister?
- On what street was your elementary school?
- What did Mrs. Woroboff teach?

# Authentication Using a Physical Object

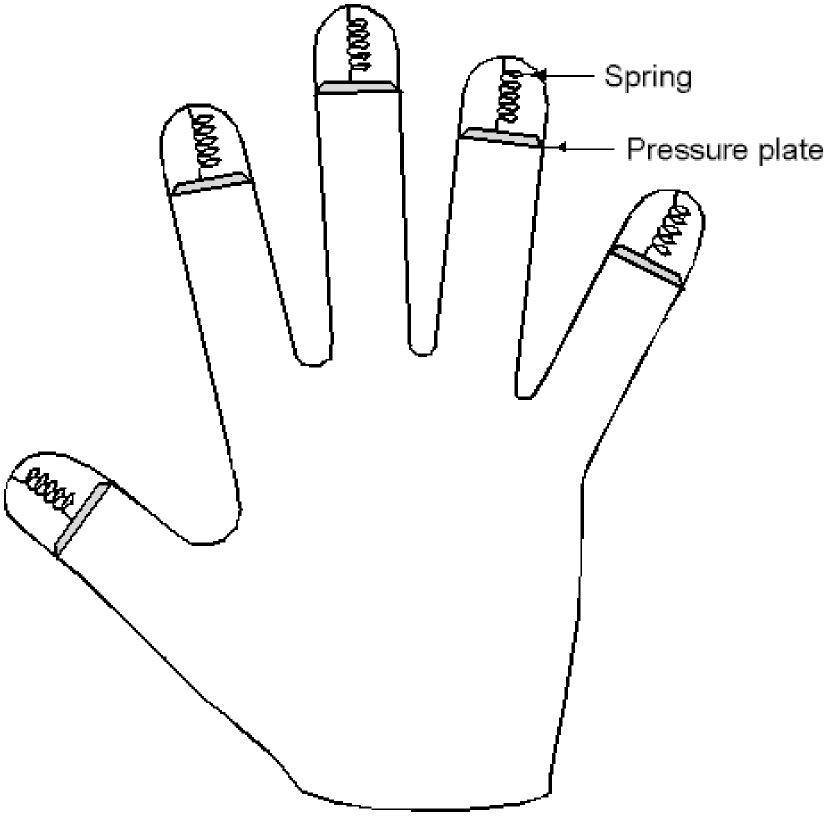


- Magnetic cards
  - magnetic stripe cards
  - chip cards: stored value cards (have EEPROM), smart cards (have processors)

# Using biometrics to authenticate

- Use of user specific information (usually some measurement from the body) along with username and password mechanism.
- Examples:
  - Retinal patterns of blood vessels
  - Fingerprints
  - Voice signatures
  - Facial matching
  - Sometimes even as simple measure as weight

# Authentication Using Biometrics



- A device for measuring finger length.

# Countermeasures

- Limiting times when someone can log in
- Automatic callback at number prespecified
- Limited number of login tries
- A database of all logins
- Simple login name/password as a trap
  - security personnel notified when attacker bites

# Operating System Security
# 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

# Login Spoofing

Login:
(a)
Login:
(b)

(a) Correct login screen

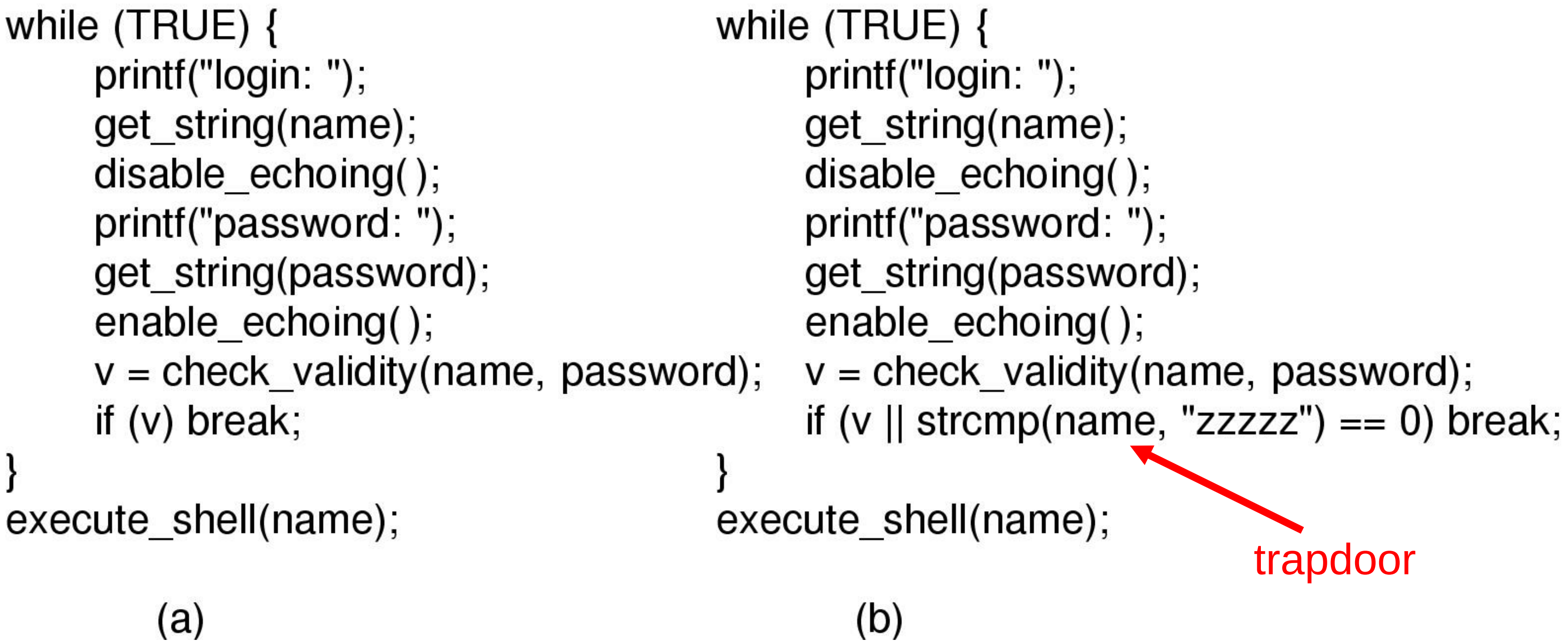
(b) Phony login screen

Remedy: get login screen by a key-combination that user programs cannot catch. (CTRL-ALT-DEL in Windows)

# Logic Bombs

- Company programmer writes program
  - potential to do harm
  - OK as long as he/she enters password daily
  - ff programmer fired, no password and bomb explodes
  - Can use for blackmail.

# 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);
```

trapdoor

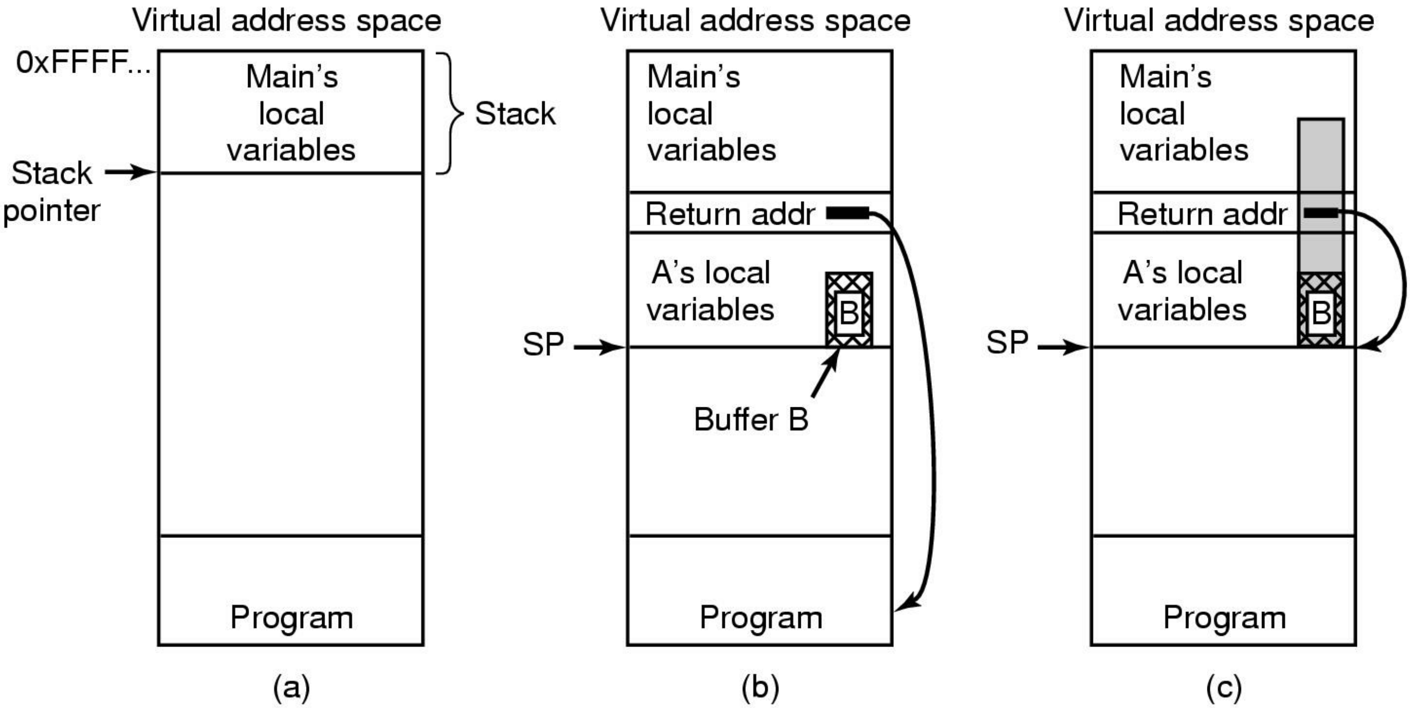
(b)

(a) Normal code.

(b) Code with a trapdoor inserted

<u>Remedy</u>: have regular code reviews.

# Buffer Overflow



Can substitute jump address of malicious function for the return address

- (a) Situation when main program is running
- (b) After program A called
- (c) Buffer overflow shown in gray

# Buffer Overflow

- Classic examples of attacks taking advantage of buffer overflow
  - Finger on BSD Unix with a very large argument that overflows the buffer.
  - When finger daemon returned, it did not return to the main() but returned to a procedure within a 536-byte string on the stack (which was given as the argument to finger that overflowed the buffer).
  - Used by the Internet Worm of 1988 by Robert Tappan Morris (of Cornell University).

# Generic Security Attacks

Typical attacks

- Request memory, disk space, tapes and just read
- Try illegal system calls
- Start a login and hit DEL, RUBOUT, or BREAK
- Try modifying complex OS structures
- Try to do specified DO NOTs
- Convince a system programmer to add a trap door
- Beg sysadmin's secretary to help a poor user who forgot password

# Famous Security Flaws

- The Unix lpr problem
- In old versions of Unix, users could print a file and remove it after it was done using lpr.
  - Symbolic link to /etc/passwd
  - Then run lpr with remove after print option
  - ➔ password file gone!

# Design Principles for Security

1. System design should be public
2. Default should be no access
3. Check for current authority
4. Give each process least privilege possible
5. Protection mechanism should be
    - simple
    - uniform
    - in lowest layers of system
6. Scheme should be psychologically acceptable

And … <u>keep it simple</u>

# Network Security

- External threat
  - code transmitted to target machine
  - code executed there, doing damage
- Goals of virus writer
  - quickly spreading virus
  - difficult to detect
  - hard to get rid of
- Virus = program can reproduce itself
  - attach its code to another program
  - additionally, do harm

# Virus Damage Scenarios

- Blackmail
- Denial of service as long as virus runs
- Permanently damage hardware
- Target a competitor's computer
  - do harm
  - espionage
- Intra-corporate dirty tricks
  - sabotage another corporate officer's files

# How Viruses Work (1): Companion viruses

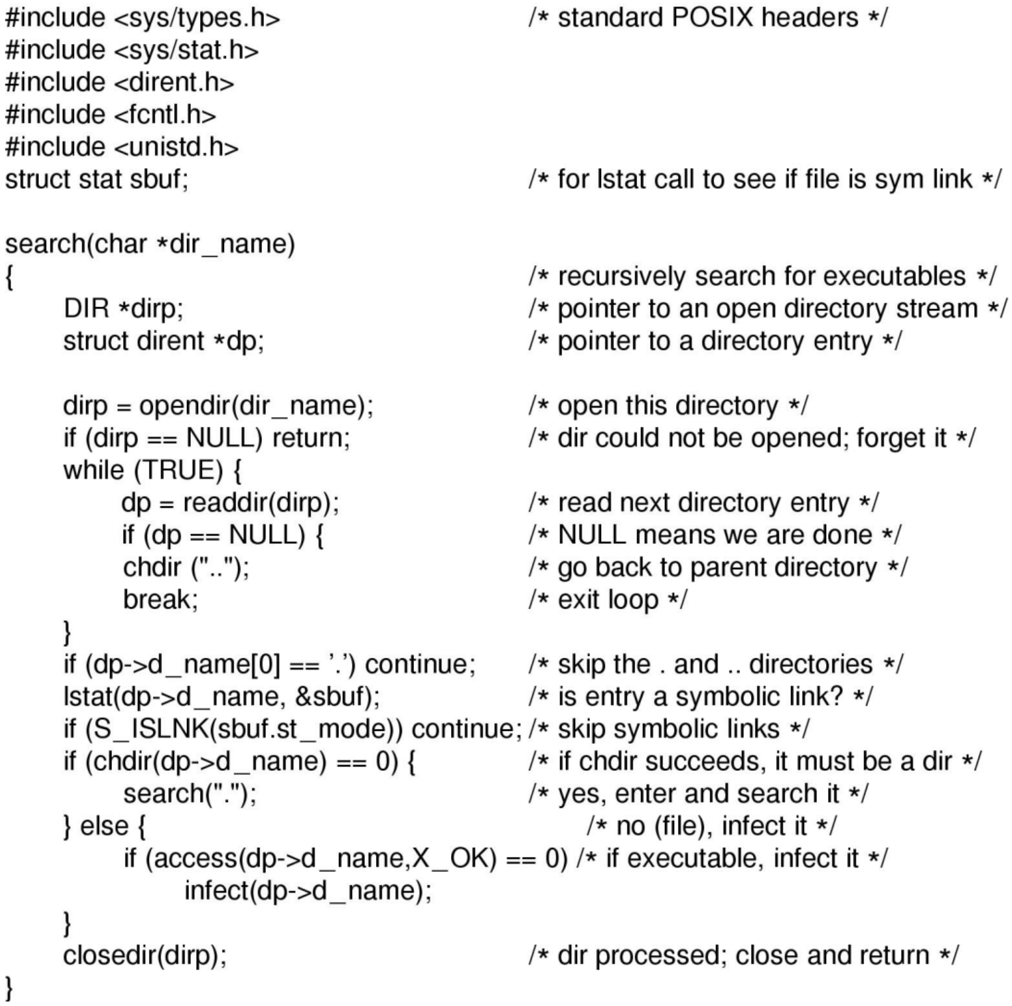
Virus written in assembly language

- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
  - eventually executes its “payload”

# How Viruses Work (2): Overwriting viruses

Recursive procedure that finds executable files on a UNIX system

Virus could infect them all

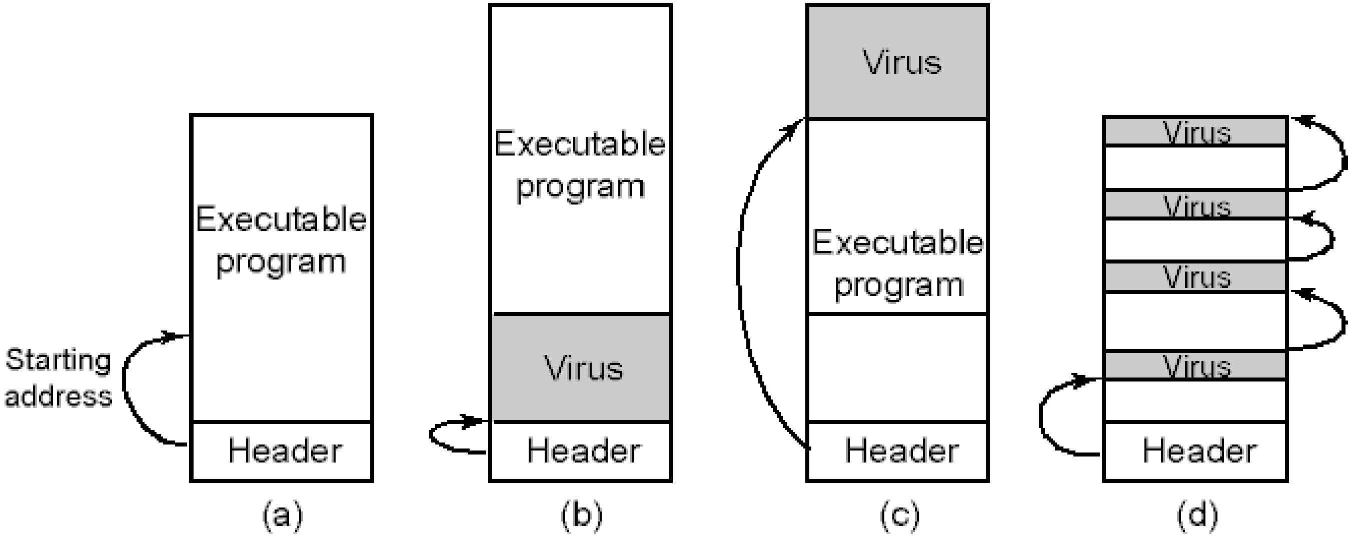


```
#include <sys/types.h>                          /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                               /* for lstat call to see if file is sym link */

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 Work (3): parasitic viruses



a) An executable program
b) With a virus at the front
c) With the virus at the end
d) With a virus spread over free space within program (cavity viruses)

# How Viruses Work (4): Boot sector viruses

- Viruses write themselves in the boot sector. At boot time, they capture the interrupt vector
- After all the drivers are loaded, they restore the interrupt vector, but keep the interrupt vector entry for system calls.
- This way, they get a chance to execute in kernel mode every time there is a legitimate system call.

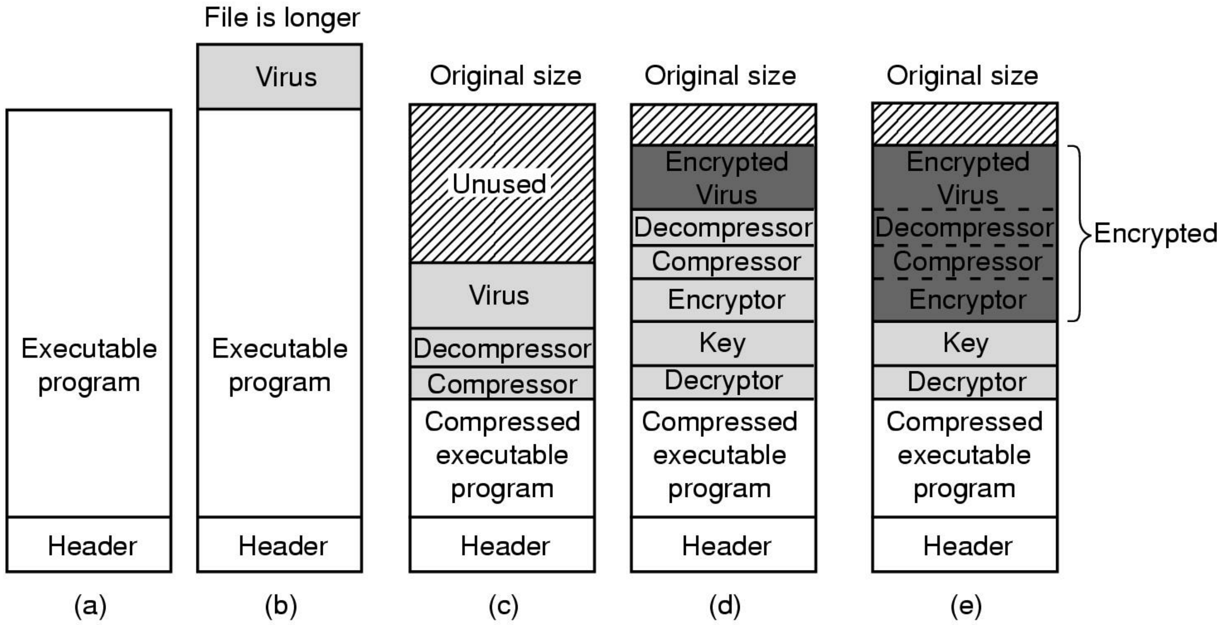
# How Viruses Spread

- Virus placed where likely to be copied
- When copied
  - infects programs on hard drive, floppy, USB drive
  - may try to spread over LAN
- Attach to innocent looking email
  - when it runs, use mailing list to replicate (example: I love you virus)

# Antivirus and Anti-Antivirus Techniques

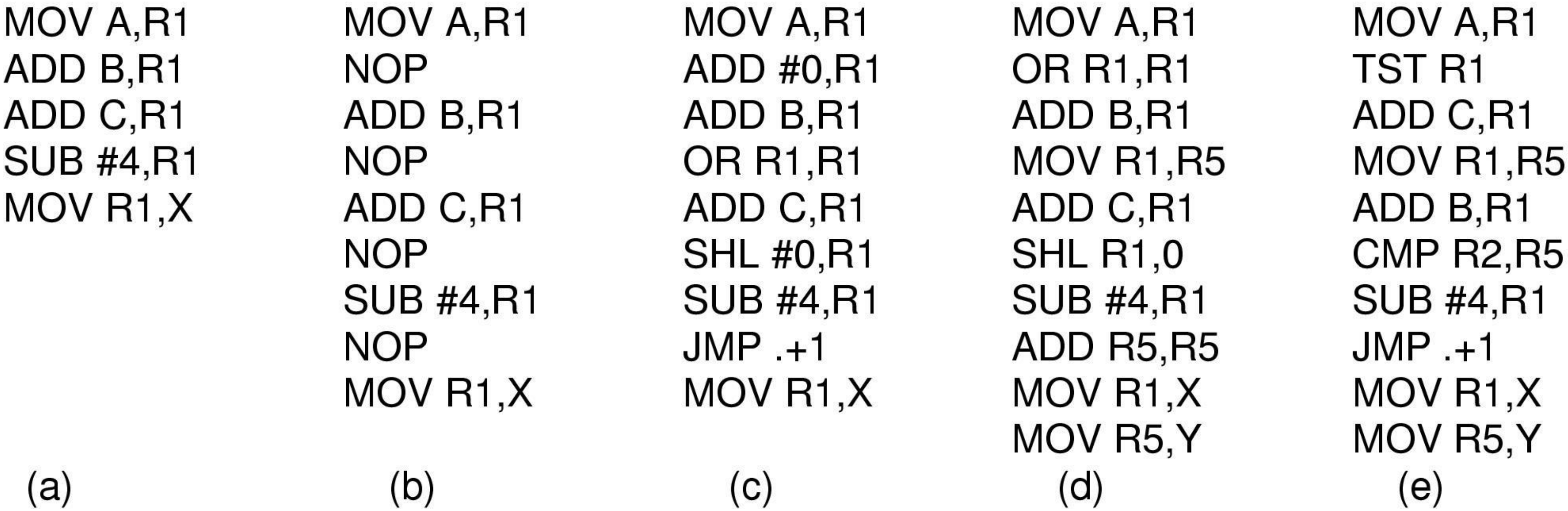
- Antivirus programs look for parts of the virus code as signatures.
- If there is a match, they flag it as possible infection.
- Or they can try to detect any changes to the file size of executable file since last virus check.
- But, virus writers can be clever. They can write viruses that can hide/change the virus.

# Antivirus and Anti-Antivirus Techniques



(a) A program
(b) Infected  program
(c) Compressed infected program (to keep file size unchanged)
(d) Encrypted virus (so virus signature cannot be detected)
(e) Compressed virus with encrypted compression code

# Antivirus and Anti-Antivirus Techniques



```
MOV A,R1      MOV A,R1      MOV A,R1      MOV A,R1      MOV A,R1
ADD B,R1      NOP           ADD #0,R1     OR R1,R1      TST R1
ADD C,R1      ADD B,R1      ADD B,R1      ADD B,R1      ADD C,R1
SUB #4,R1     NOP           OR R1,R1      MOV R1,R5     MOV R1,R5
MOV R1,X      ADD C,R1      ADD C,R1      ADD C,R1      ADD B,R1
              NOP           SHL #0,R1     SHL R1,0      CMP R2,R5
              SUB #4,R1     SUB #4,R1     SUB #4,R1     SUB #4,R1
              NOP           JMP .+1       ADD R5,R5     JMP .+1
              MOV R1,X      MOV R1,X      MOV R1,X      MOV R1,X
                                          MOV R5,Y      MOV R5,Y
  (a)            (b)           (c)           (d)           (e)
```

- Examples of a polymorphic virus
- virus can change its copy (mutate) as it propagates itself
- All of the above examples do the same thing

# Antivirus and Anti-Antivirus Techniques

- Integrity checkers (use checksums to try to detect changes in executable files)
- Behavioral checkers (anti-virus program watches for suspicious activity for all system calls)
  - Example, normal programs should not be writing to boot sector.
- Virus avoidance
  - good OS (strong kernel mode/user mode boundary)
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups
- Recovery from virus attack
  - halt computer, reboot from safe disk, run antivirus

# Mobile codes

- Any kind of foreign code that is imported from a remote site and run on local machine.
- Example mobile codes:
  - Java (or other) applets
  - Agents
  - Postscript code

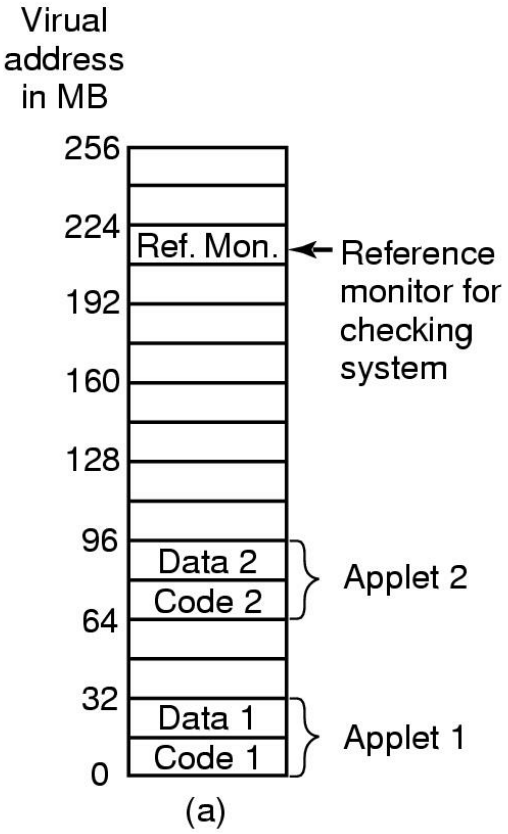
# Mobile codes

- Can mobile codes be run safely?
  - Yes, but not easily.
- Common methods to run mobile codes safely:
  - Sandboxing
  - Interpretation
  - Code signing

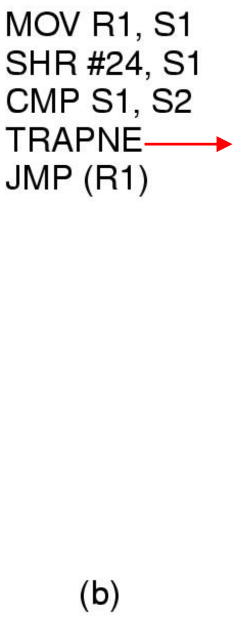
# Mobile Code: Sandboxing

- Attempts to confine each applet to a restricted range of virtual addresses.
- Each sandbox has the property that all of its addresses share the same string of higher order bits (that's for checking efficiently if the applet is remaining within its sandbox)
- For example, a 32-bit address can be divided into 256, 16-MB boundary sandboxes.
  - Higher order 8 bits will be common within each sandbox.
- The sandboxes should be large enough to fit applets, but also not waste memory space.

# Mobile Code: Sandboxing



Reference monitor traps system calls and examines if it is safe to perform them

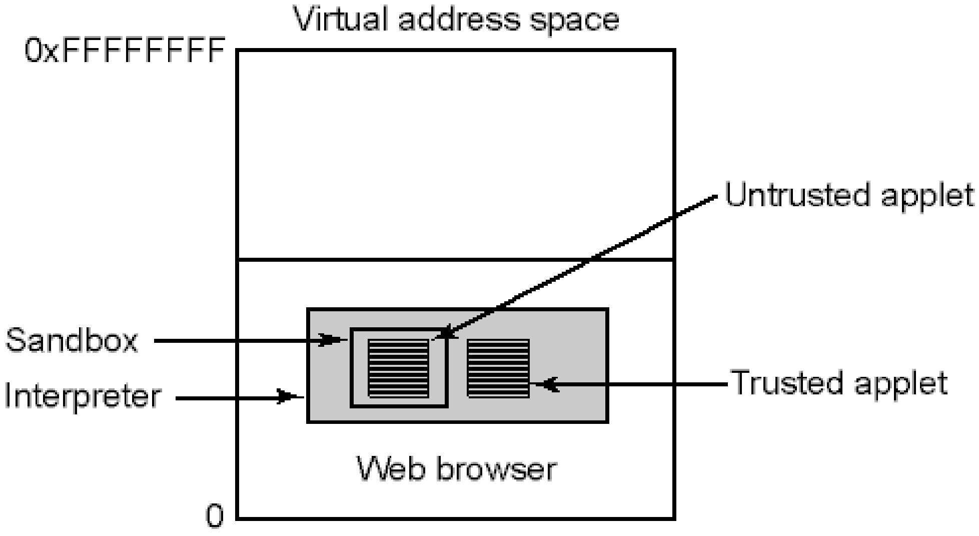


```
MOV R1, S1
SHR #24, S1
CMP S1, S2
TRAPNE
JMP (R1)
```

TRAPNE inserted before JMP instruction to determine if the contents of R1 are within sandbox during runtime.

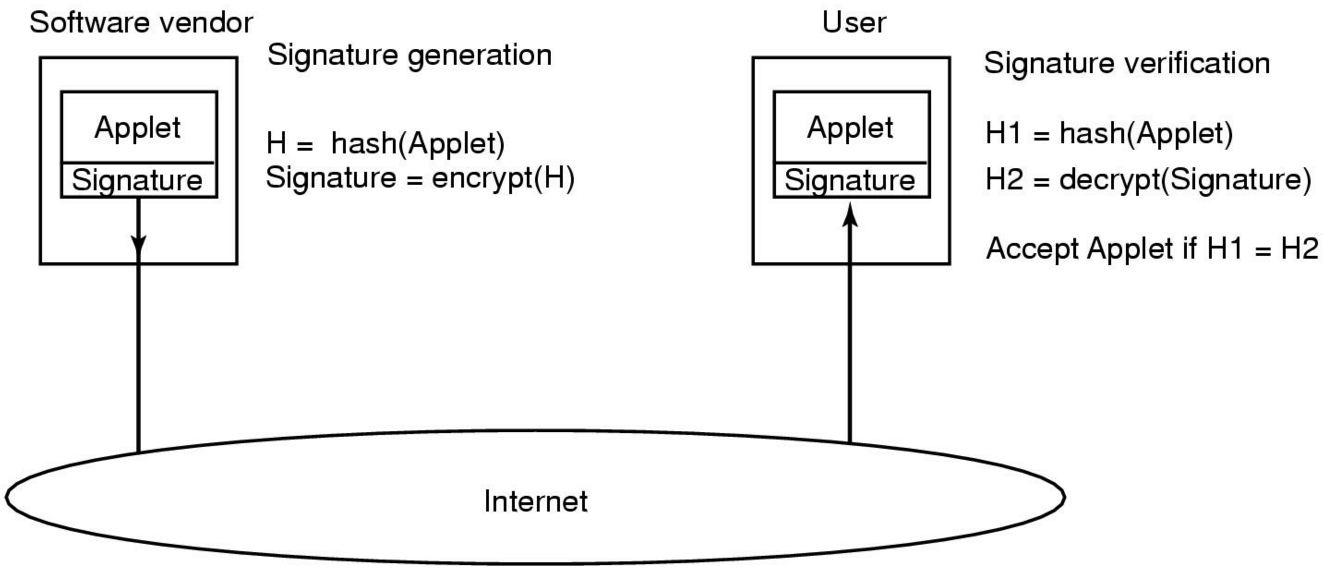
(a) Memory divided into 16-MB sandboxes
(b) One way of checking an instruction for validity

# Mobile Code: Interpretation



- Applets can be interpreted by a Web browser
- Slow!

# Mobile Code: Code signing



- Accept code from only trusted sources.
- (Public, private) key mechanism used
- How code signing works