Sheets for MIEIC's SOPE

based on teaching material supplied by
A. Tanenbaum for book:
Modern Operating Systems, ed...

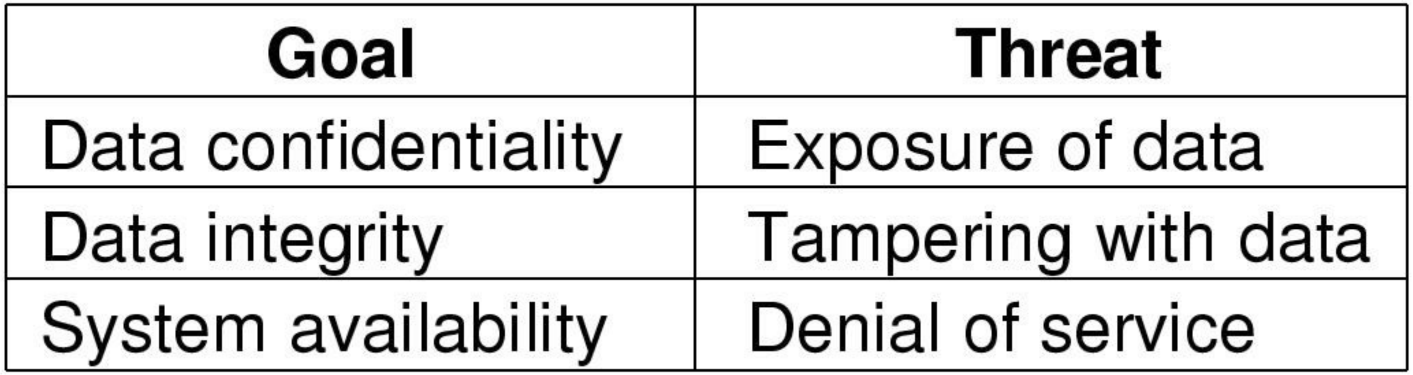
Basics of Security

Basics of Security

The security environment
Basics of cryptography
User authentication
Attacks from inside & outside
Protection mechanisms
Trusted systems

The Security Environment

Threats



Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service

Security goals and threats

Intruders

Common Categories

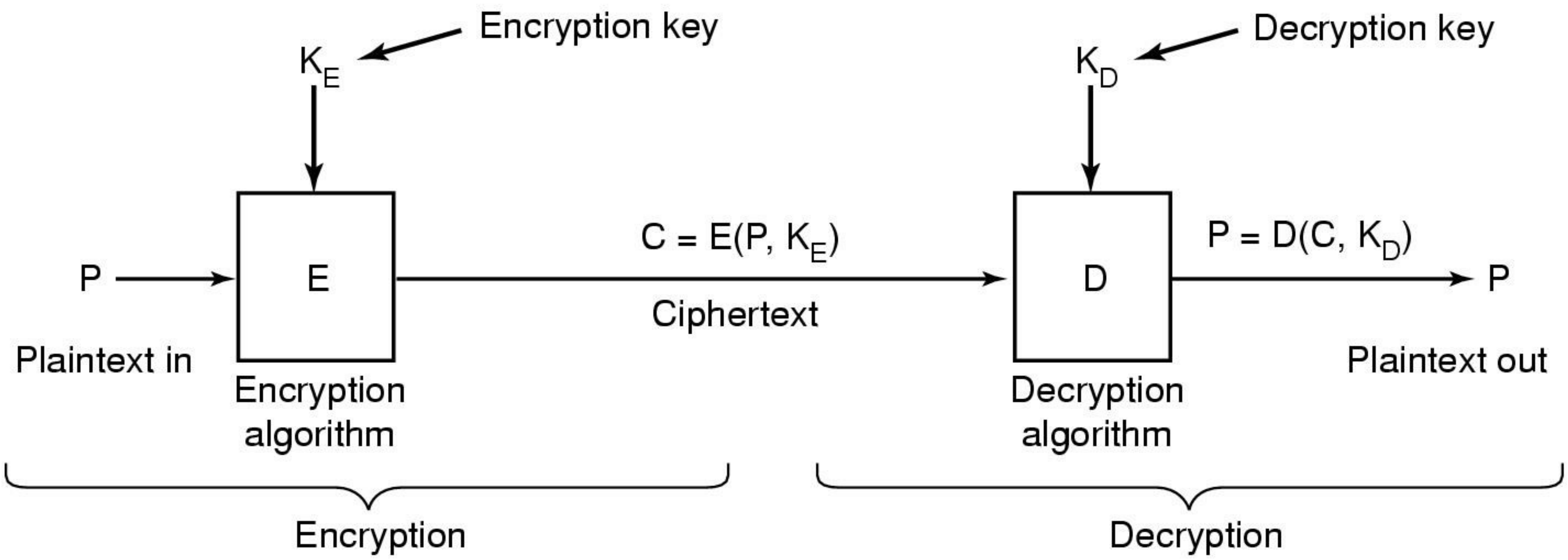
1. Casual prying by nontechnical users
2. Snooping by insiders
3. Determined attempt to make money
4. Commercial or military espionage

Accidental Data Loss

Common Causes

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

Basics of Cryptography



Relationship between the *plaintext* and the *ciphertext*

Secret-Key Cryptography

- Toy example: mono-alphabetic substitution
  - each letter replaced by different letter
- Given the encryption key,
  - easy to find decryption key: usually, the same!
- Secret-key crypto called ***symmetric-key crypto***

Public-Key Cryptography

- All users pick a public key/private key pair
  - publish the public key
  - private key not published

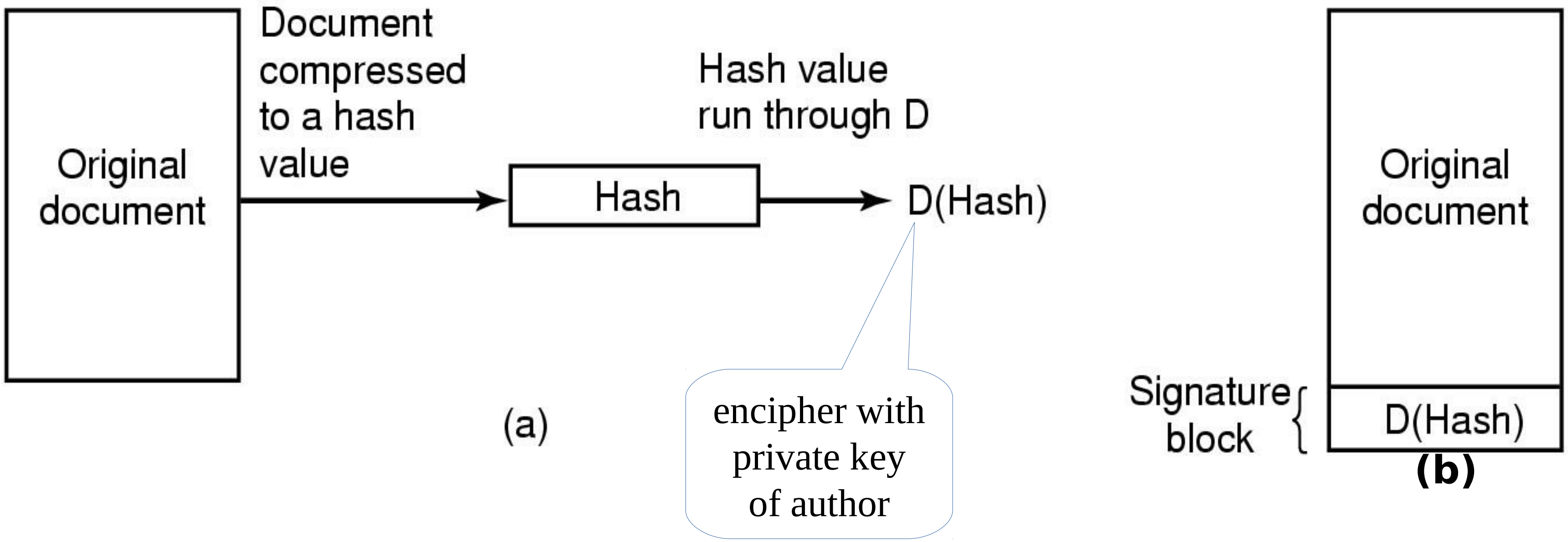
- Public key is the encryption key
  - private key is the decryption key

- Public-key crypto called ***asymmetric-key crypto***

Hash Function

- Function such that given formula for f(x)
  - easy to evaluate y = f(x)
- But given y
  - computationally infeasible to find x
- Also, two different x should give two different y
  - if (x_1 != x_2) then f(x_1) != f(x_2)

Digital Signatures



a) **Usual** computation of signature block

- the Hash is not really necessary, it is a performance trick

b) What the receiver gets

- verification needs public key of author

User Authentication

Basic Principles. Authentication must identify:

1. Something the user knows
   - e.g. password
2. Something the user has
   - e.g. debit card
3. Something the user is
   - e.g. correct fingerprint

This is done before user can use the system

Countermeasures

- Limiting times when someone can log in
- Automatic callback at number pre-specified
- Limited number of login tries
- A database of past 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

A questão do `./prog` - simulação de um ataque:

- Berto colocou "`.`" no início da variável de ambiente `PATH`, para evitar escrever sempre `./prog` (ou, por exemplo, `` `pwd`/prog ``) qdo queria executar "`prog`" mal acabasse de compilar `prog.c`. Agora bastava escrever "`prog`" e o programa corria;
- Berto vai ao WC deixando o computador ligado;
- Carloso aproveita e coloca um cavalo de Tróia "`rm`" no dir de trabalho de Berto;
- Berto volta e escreve "`rm *.old`" no terminal para remover os seus fichs antigos;
- como o caminho de pesquisa do executável "`rm`" (`PATH`) começa pelo dir corrente, o cavalo de Tróia é executado em vez do verdadeiro utilitário e, para além de remover os fichs pedidos, elimina também todos os fichs da pasta `Documentos`!

Defesa1:

- numa típica instalação inicial a variável `PATH` não inclui o "`.`". O utilizadores estão protegidos contra este ataque mas têm de andar sempre a escrever (pelo menos) "`./prog`" para executar o seu programa `prog`.

Defesa 2:

- https://web.fe.up.pt/~jmcruz/so/so.2021-i/doc/unix-freq-probs.html (1ª linha)

Login Spoofing

Login:
(a)
Login:
(b)

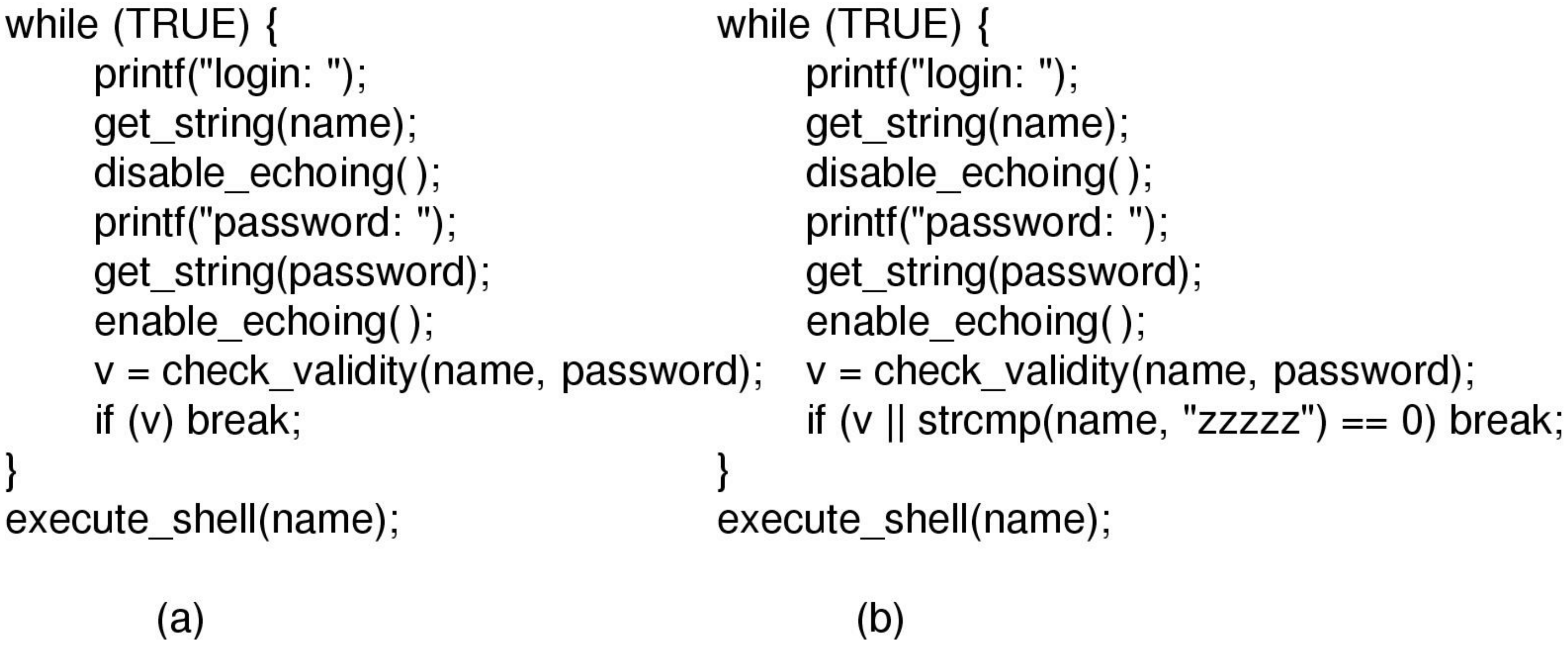
(a) Correct login screen

(b) Phony login screen

Logic Bombs

- Company programmer writes program
  - potential to do harm
  - Ok as long as he/she enters password daily
  - if programmer fired, no password is entered, the 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)

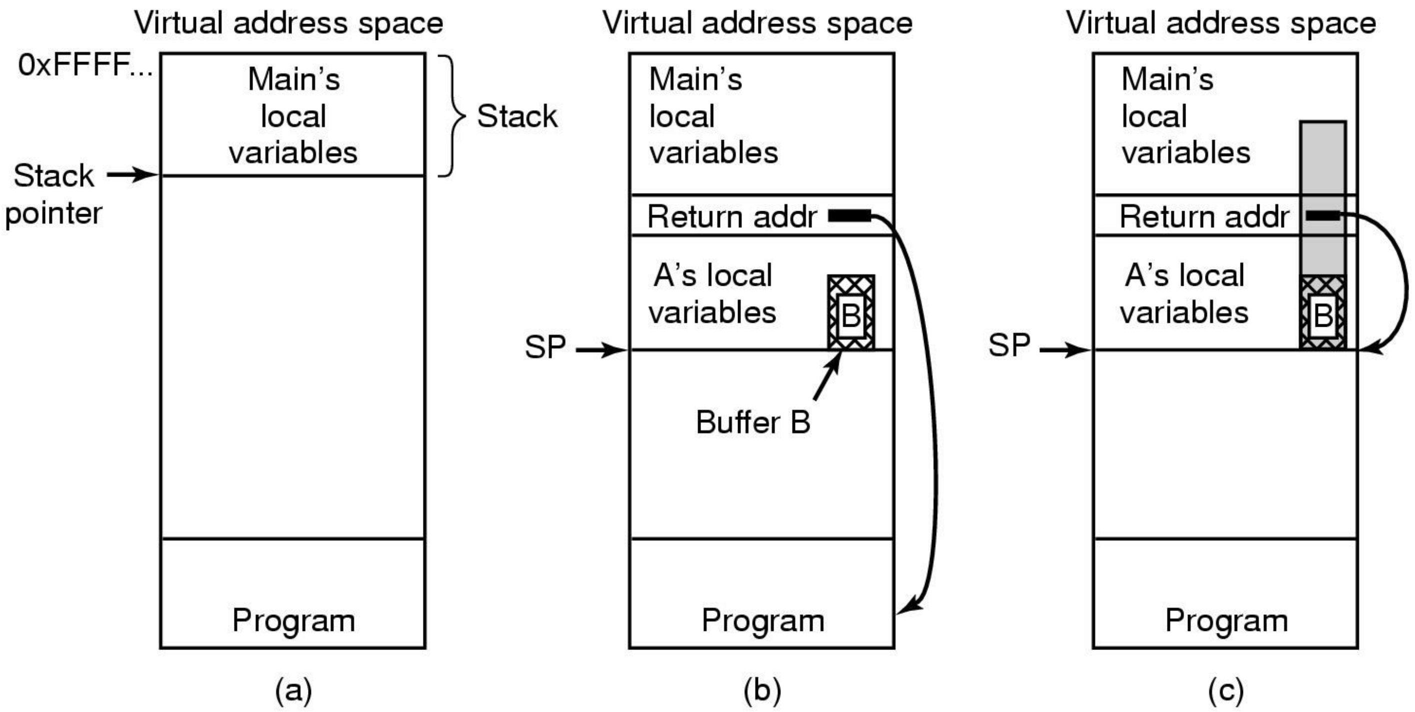
```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);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

(b)

(a) Normal code.

(b) Code with a trapdoor inserted

Buffer Overflow



(a) Situation when main function is running

(b) After call of function *A*

- return from A points to main function just after call of A

(c) Buffer overflow shown in gray

- carefully crafted malicious input overflows Buffer B, and sets return from A to malicious code

Generic Security Attacks

Typical attacks

- Request memory or disk space and just read them on
- Try illegal system calls
- Start a login and hit CTRL-C
- Try modifying complex OS structures
- Try to do specified DO NOTs
- Convince a system programmer to add a trap door
- Beg front-office administrative to help a poor user who forgot password

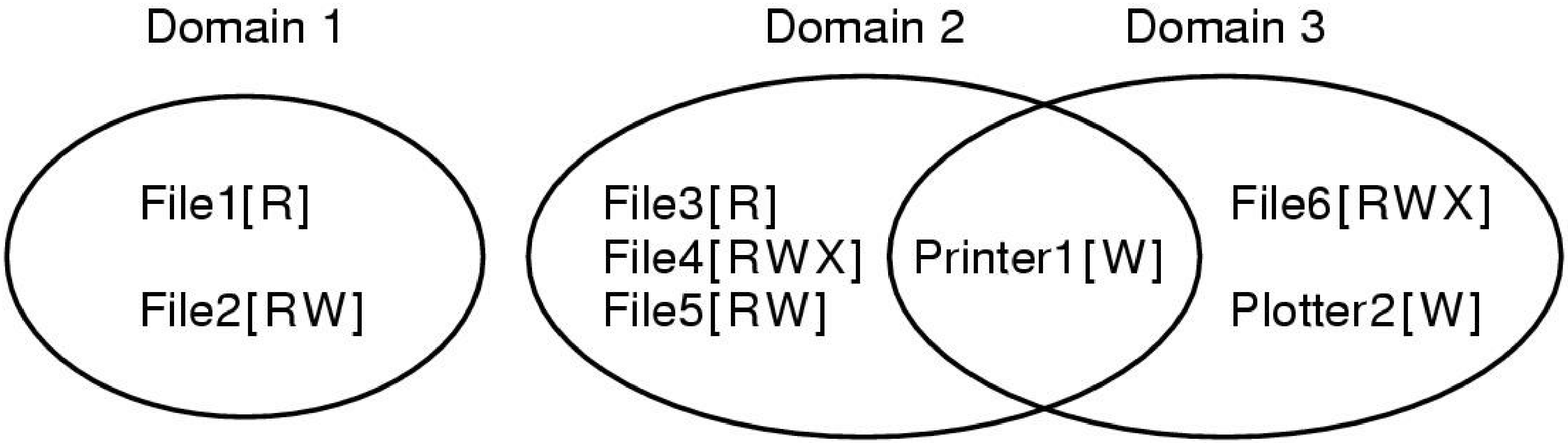
Design Principles for Security

1. System design should be public
2. Default should be NO access
3. Check for current user's clearance
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 … keep it simple

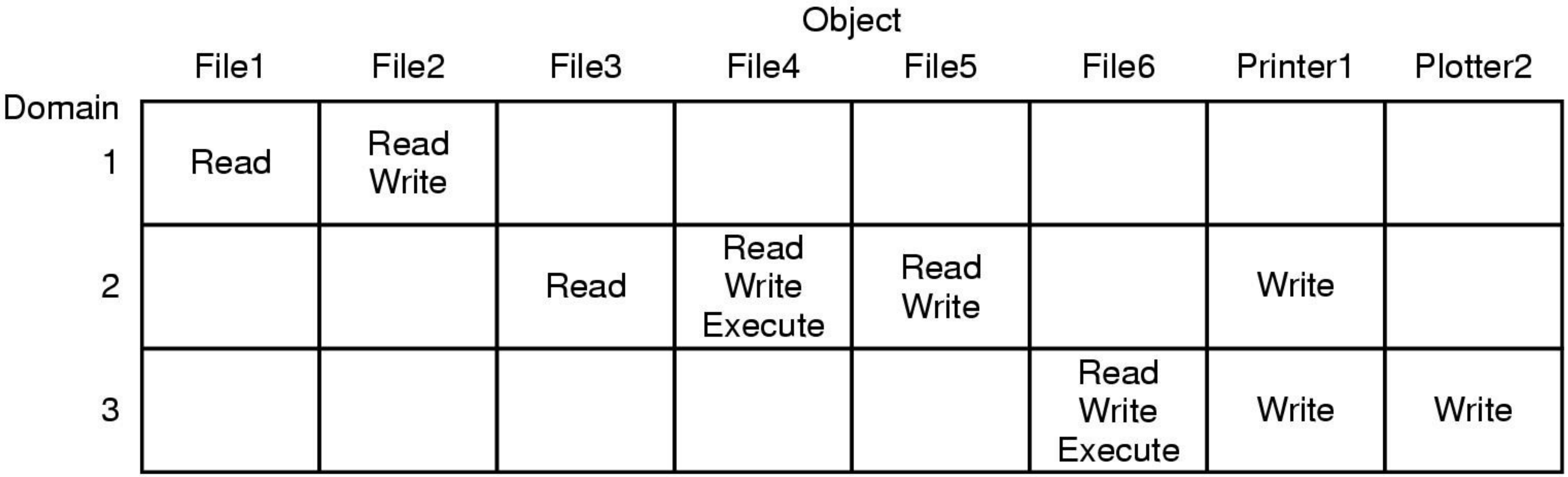
Protection Mechanisms

Protection Domains (1)



Examples of three protection domains

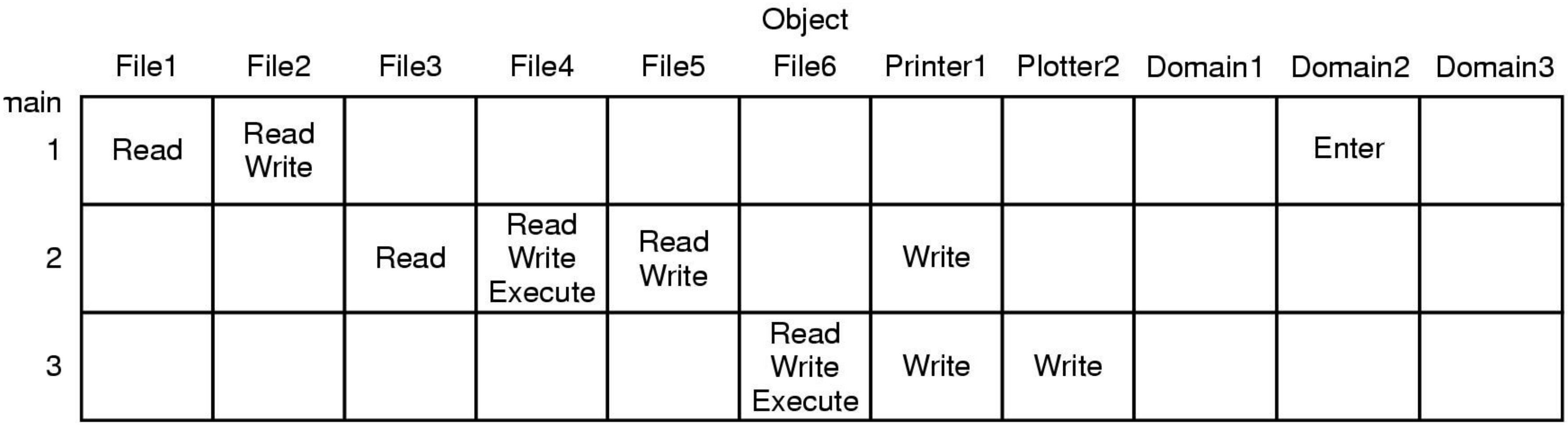
Protection Domains (2)



	Object							
Domain	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

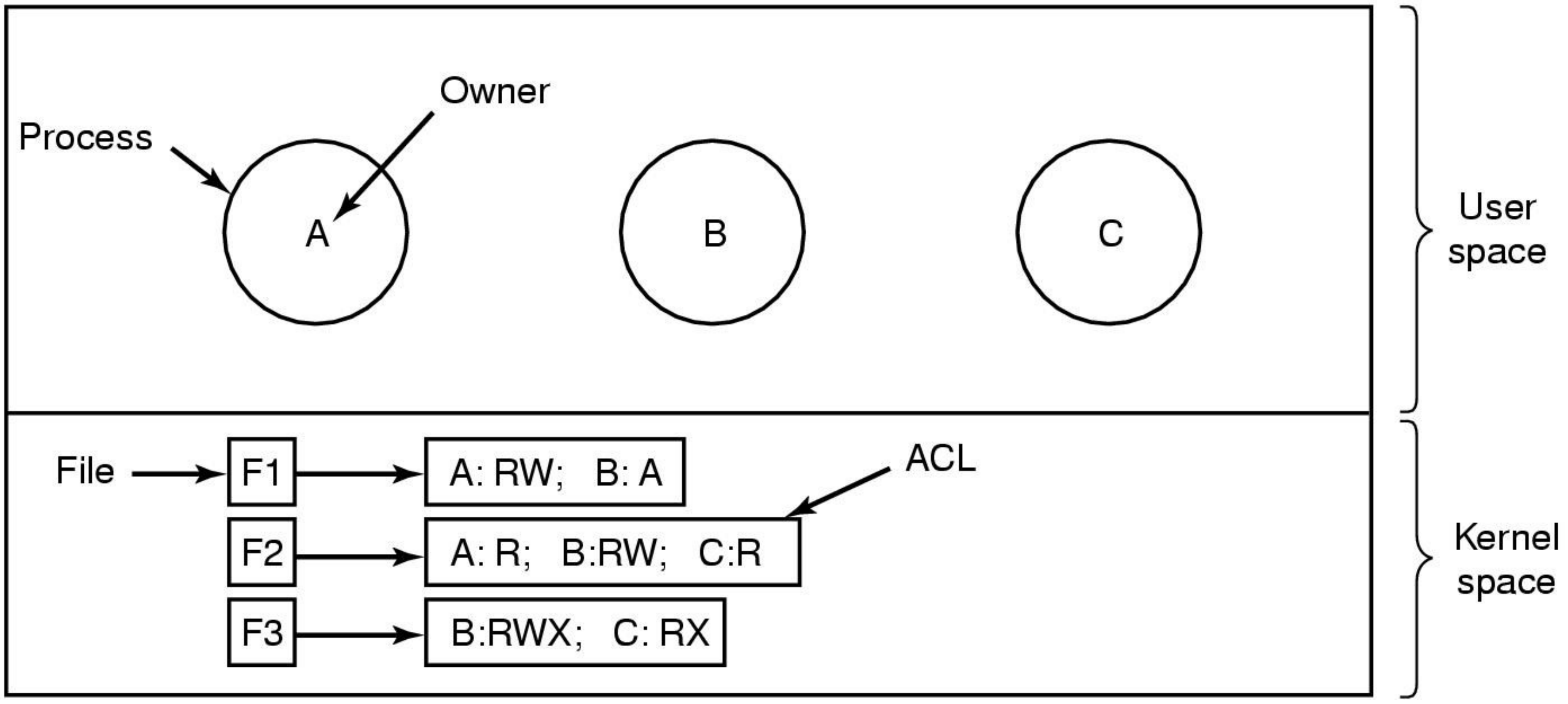
Protection Domains (3)



	Object										
main	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			

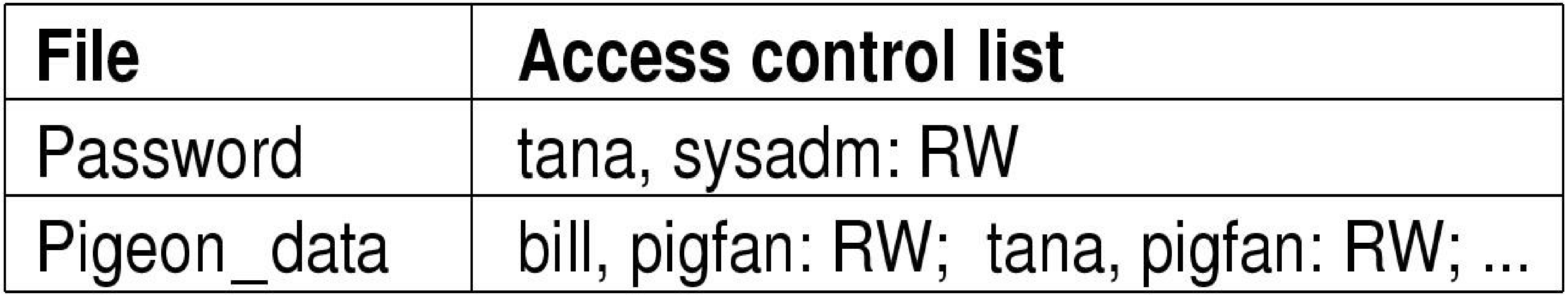
A protection matrix with domains as objects

Access Control Lists (1)



Use of access control lists of manage file access

Access Control Lists (2)

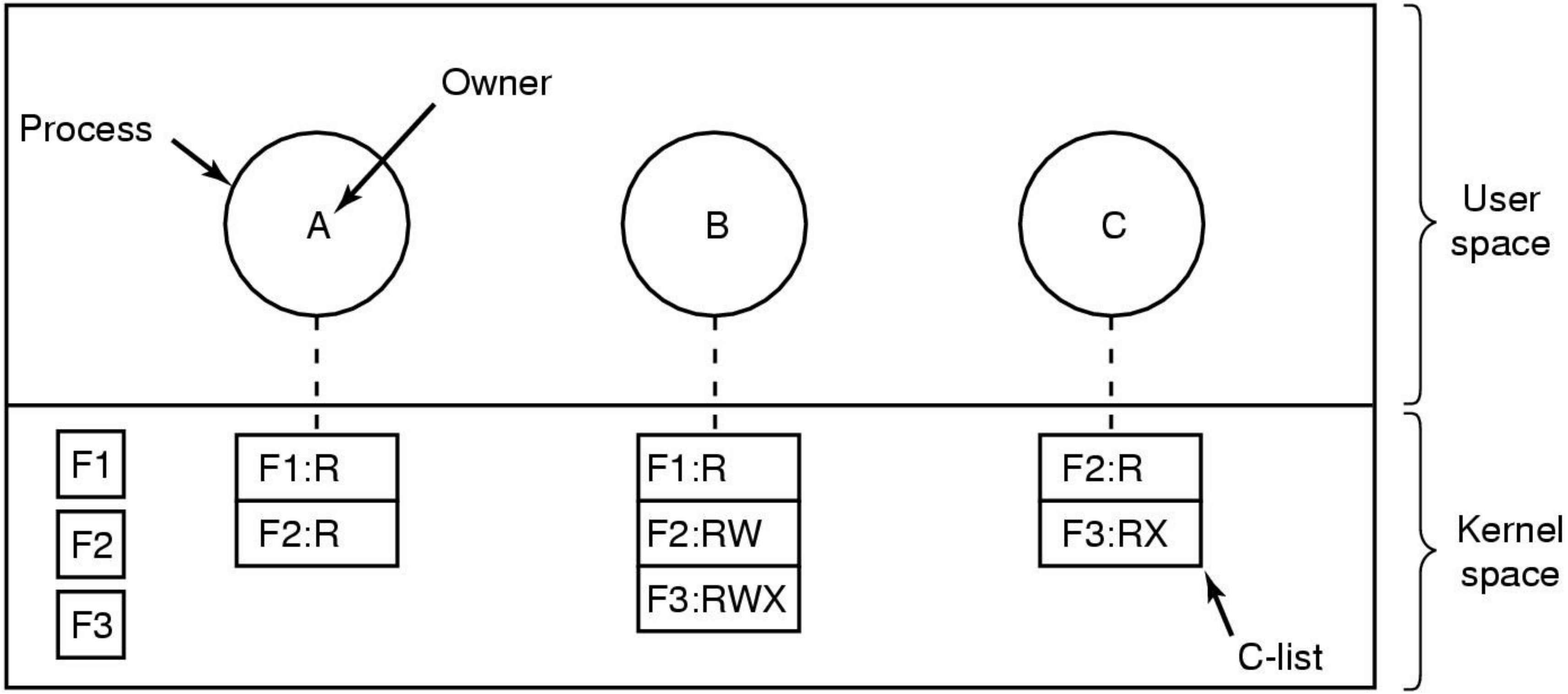


File	Access control list
Password	tana, sysadm: RW
Pigeon_data	bill, pigfan: RW; tana, pigfan: RW; ...

user, group

Two access control lists

Capabilities (1)



Each process has a capability list

Capabilities (2)

- Cryptographically-protected capability

Server	Object	Rights	hash (Objects, Rights, Check)

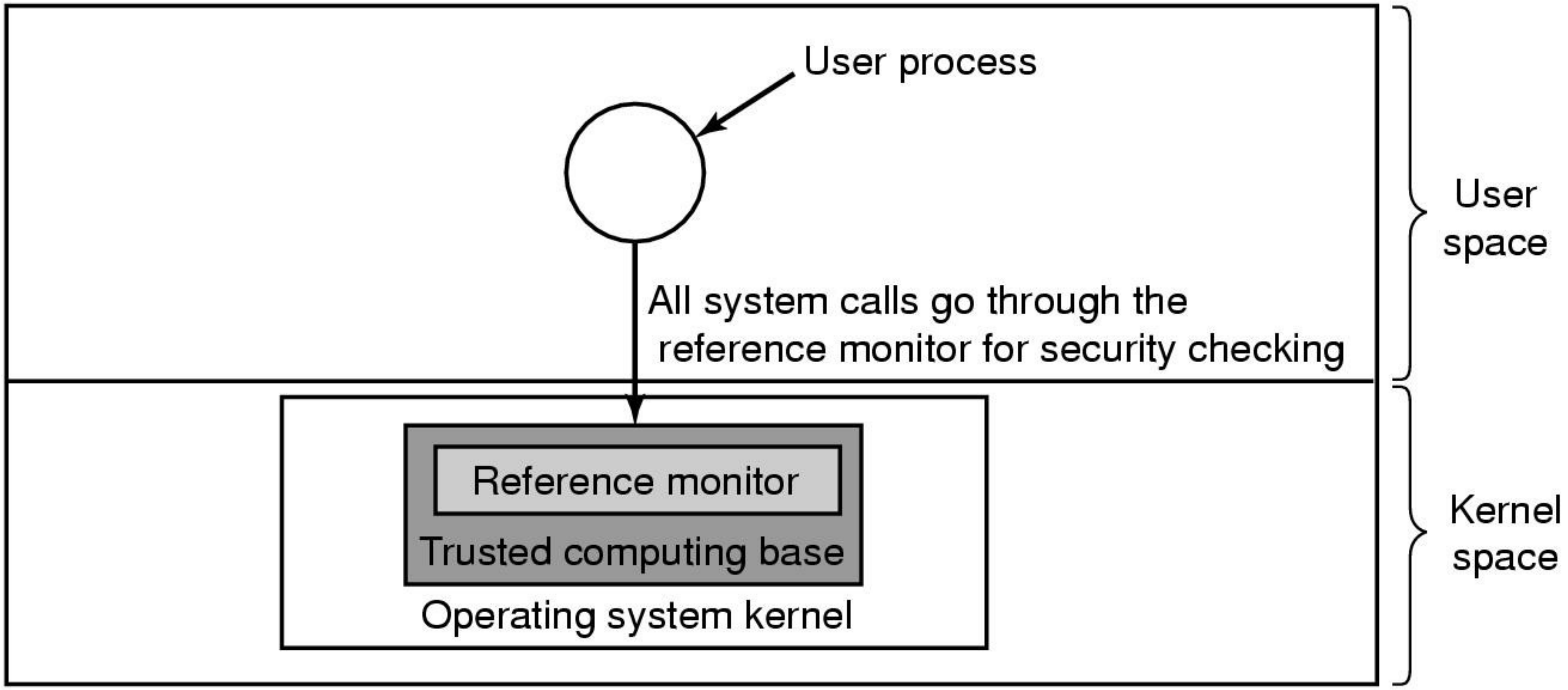
secret

- Generic Rights

1. Copy capability
2. Copy object
3. Remove capability
4. Destroy object

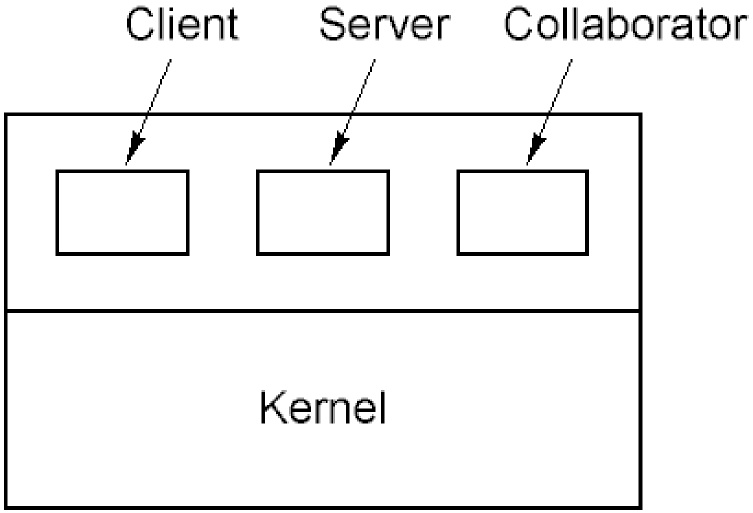
Trusted Systems

Trusted Computing Base



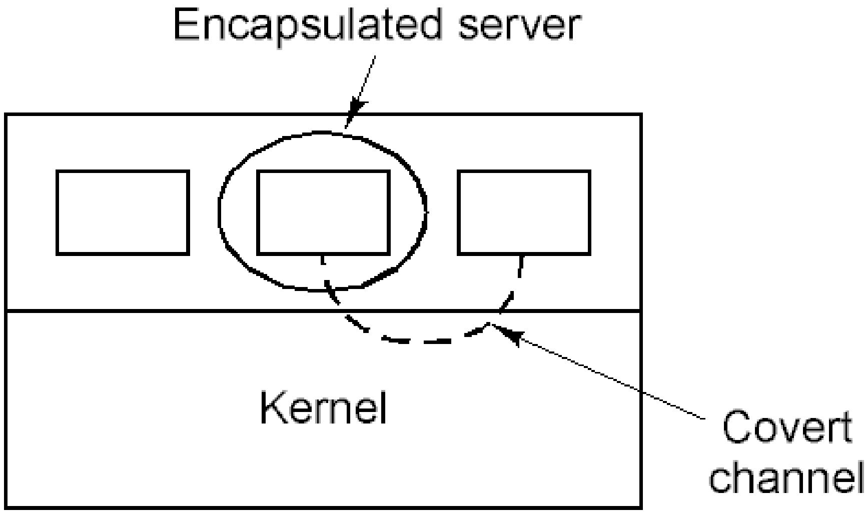
A reference monitor

Covert Channels (1)



(a)

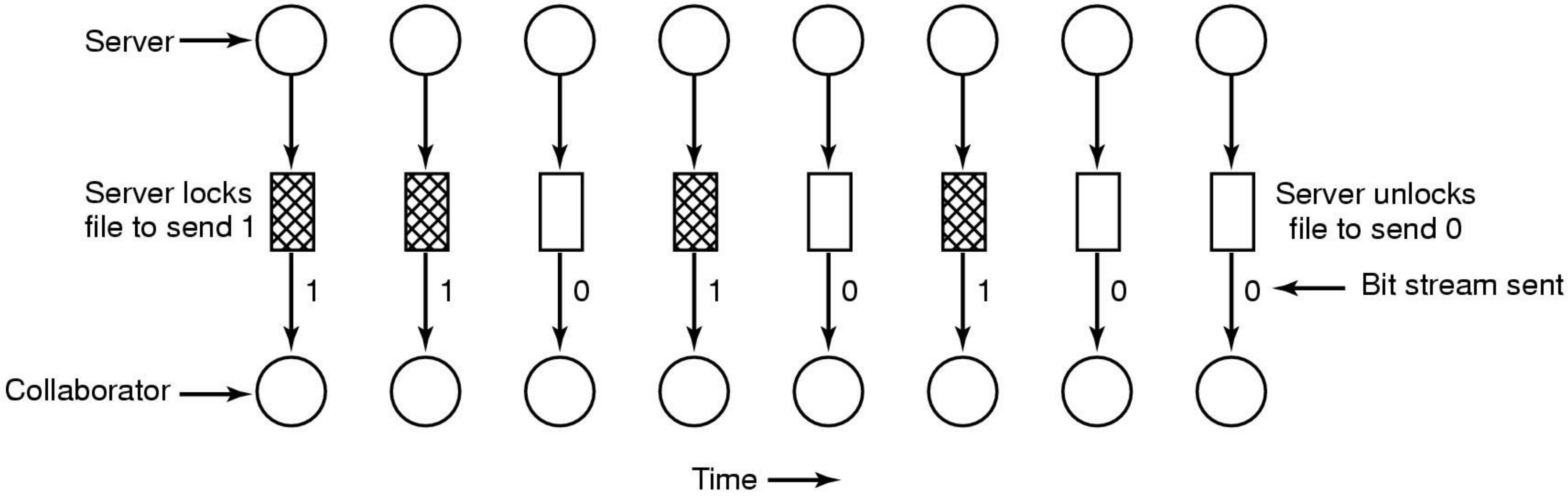
Client, server and collaborator processes



(b)

Encapsulated server can still leak to collaborator via covert channels

Covert Channels (2)



A covert channel using file locking

Covert Channels (3): steganography

- Pictures appear the same
- Picture on right has text of 5 Shakespeare plays
  - encrypted, inserted into low order bits of color values

Zebras

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