9

SECURITY

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

Fig. 9-1. Security goals and threats.

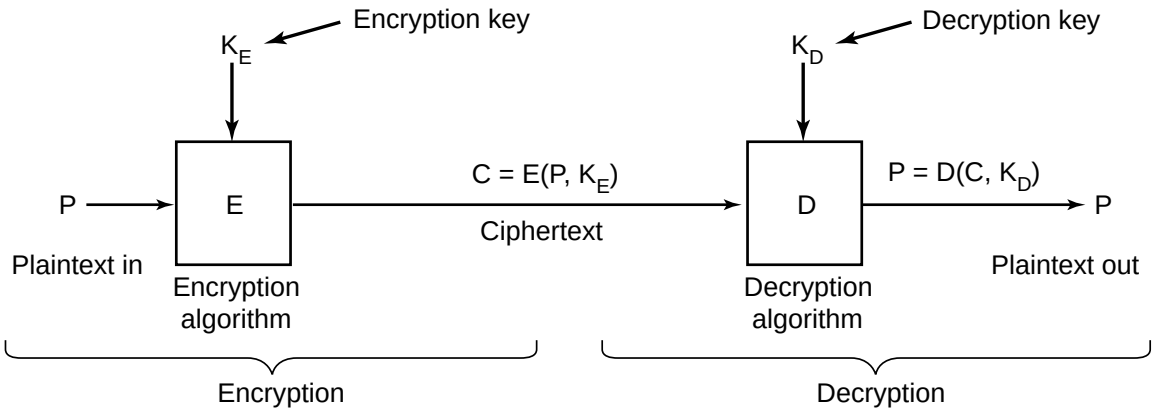

Fig. 9-2. Relationship between the plaintext and the ciphertext.

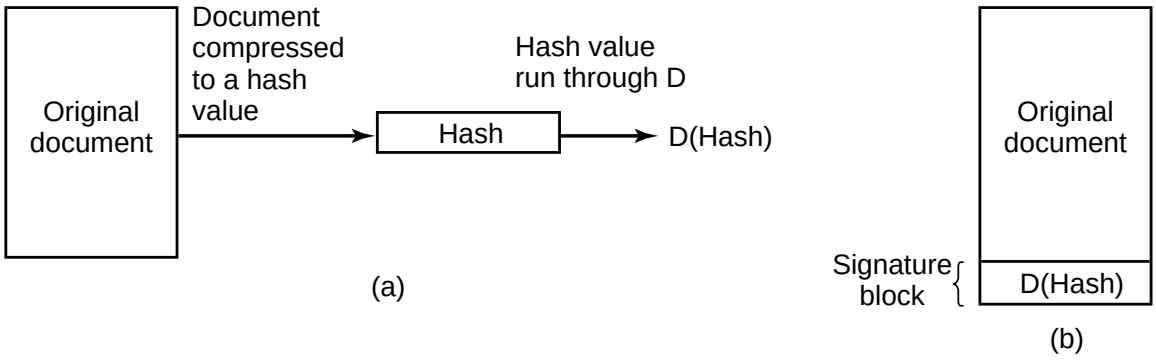

Fig. 9-3. (a) Computing a signature block. (b) What the receiver gets.

LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN

(a)

LOGIN: carol
INVALID LOGIN NAME
LOGIN:

(b)

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

(c)

Fig. 9-4. (a) A successful login. (b) Login rejected after name is entered. (c) Login rejected after name and password are typed.

```
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
```

Fig. 9-5. How a cracker broke into a U.S. Dept. of Energy computer at LBL.

Bobbie, 4238, e(Dog4238)
Tony, 2918, e(6%%TaeFF2918)
Laura, 6902, e(Shakespeare6902)
Mark, 1694, e(XaB@Bwcz1694)
Deborah, 1092, e(LordByron,1092)

Fig. 9-6. The use of salt to defeat precomputation of encrypted passwords.

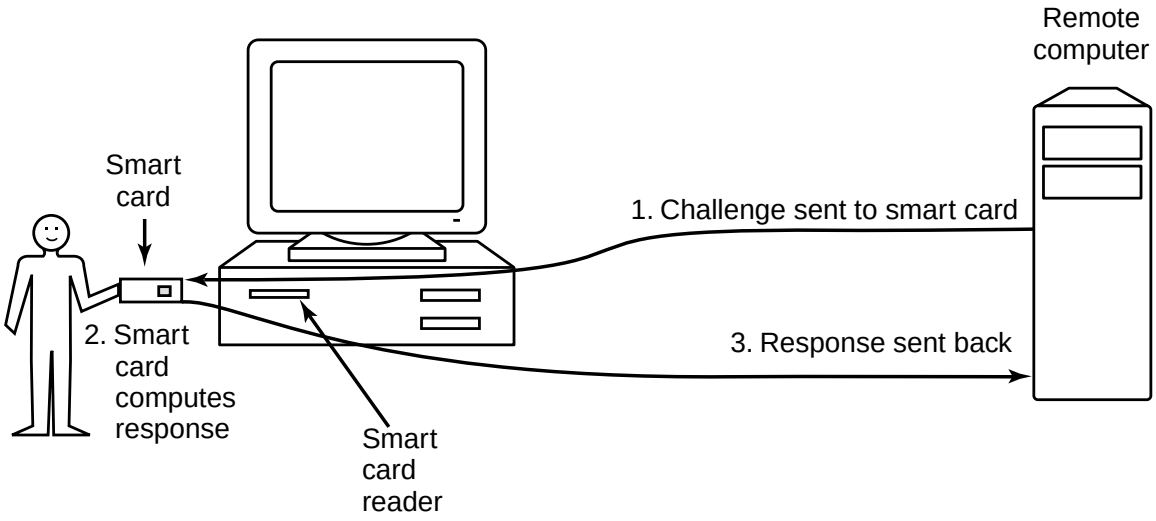

Fig. 9-7. Use of a smart card for authentication.

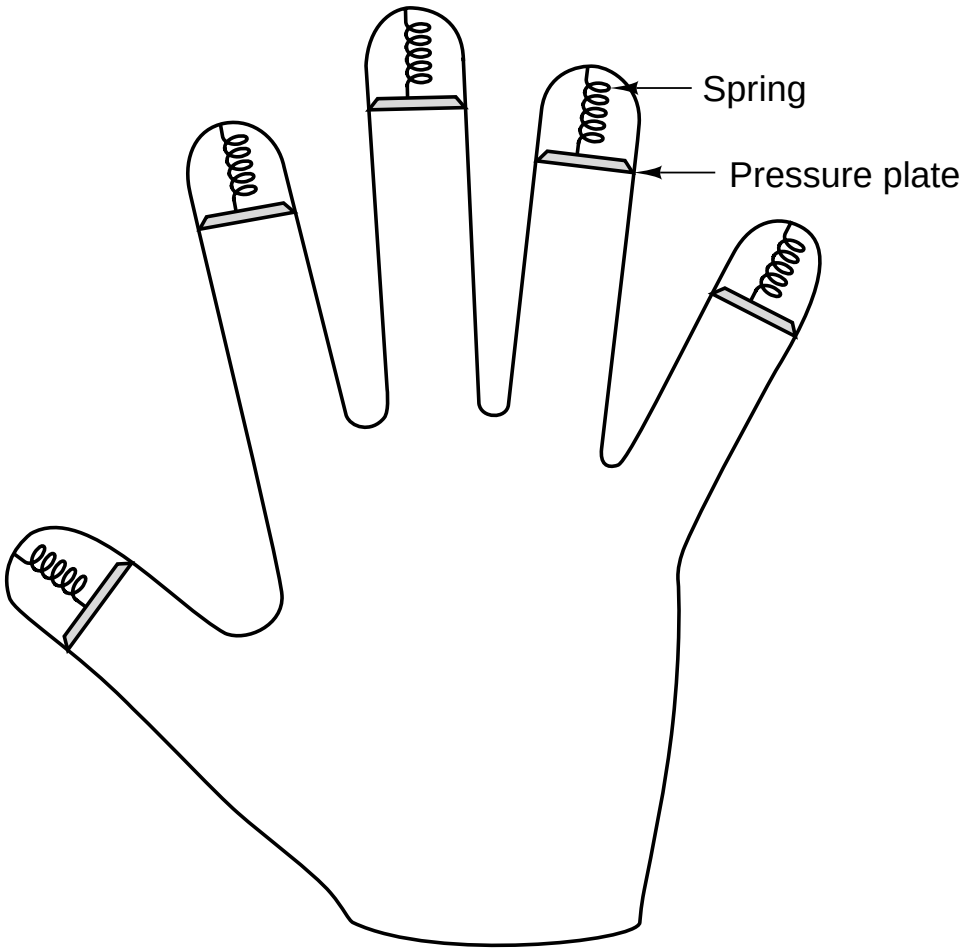

Fig. 9-8. A device for measuring finger length.

Fig. 9-9. (a) Correct login screen. (b) Phony login screen.

```
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)

Fig. 9-10. (a) Normal code. (b) Code with a trap door inserted.

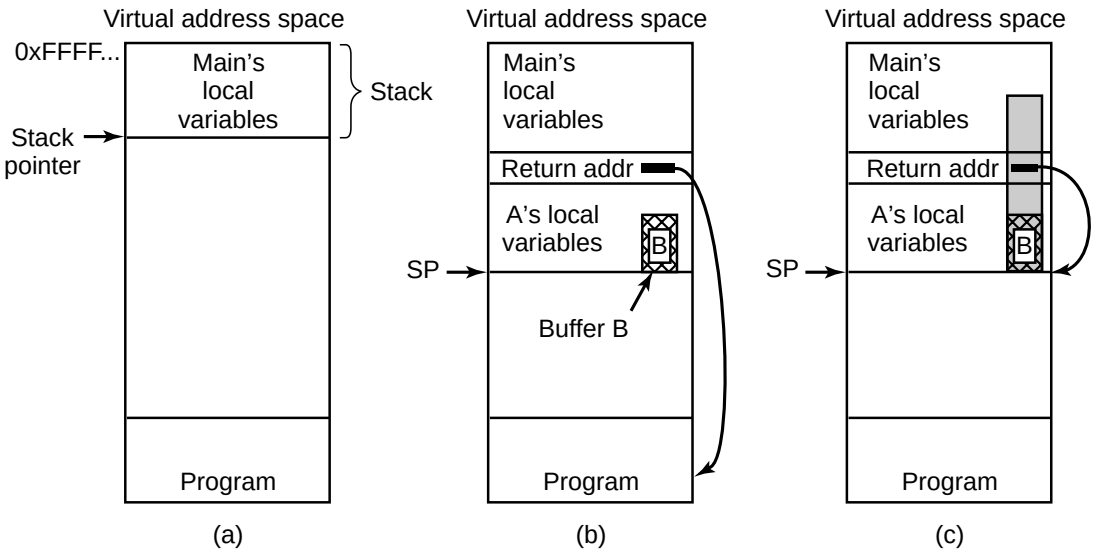

Fig. 9-11. (a) Situation when the main program is running. (b) After the procedure *A* has been called. (c) Buffer overflow shown in gray.

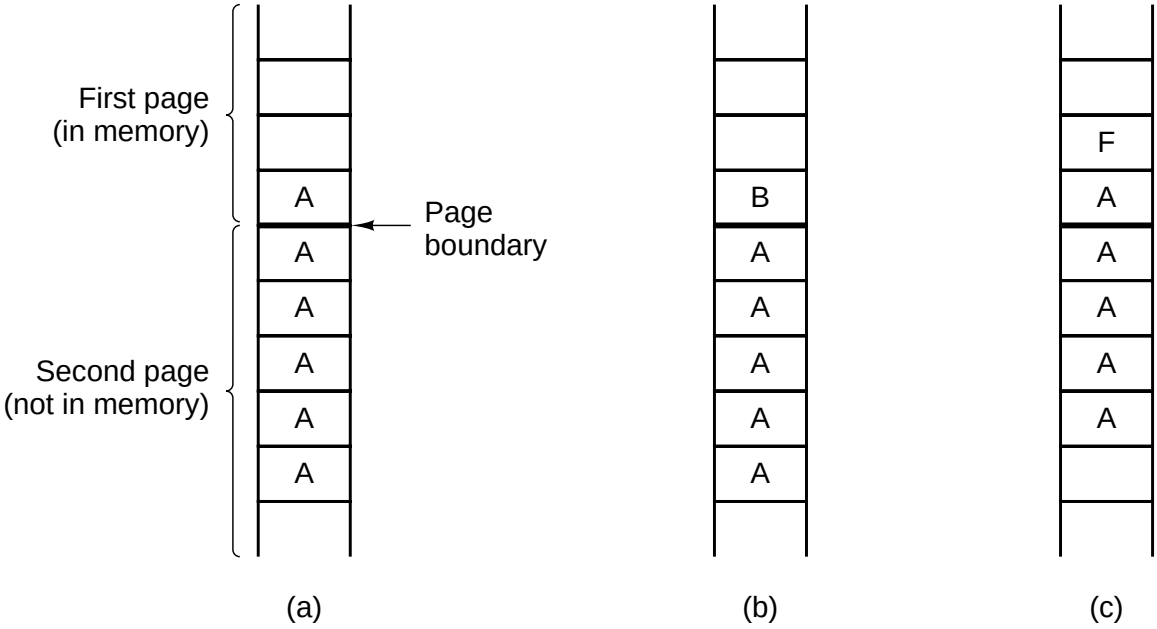

Fig. 9-12. The TENEX password problem.

```
#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 */
}
```

Fig. 9-13. A recursive procedure that finds executable files on a UNIX system.

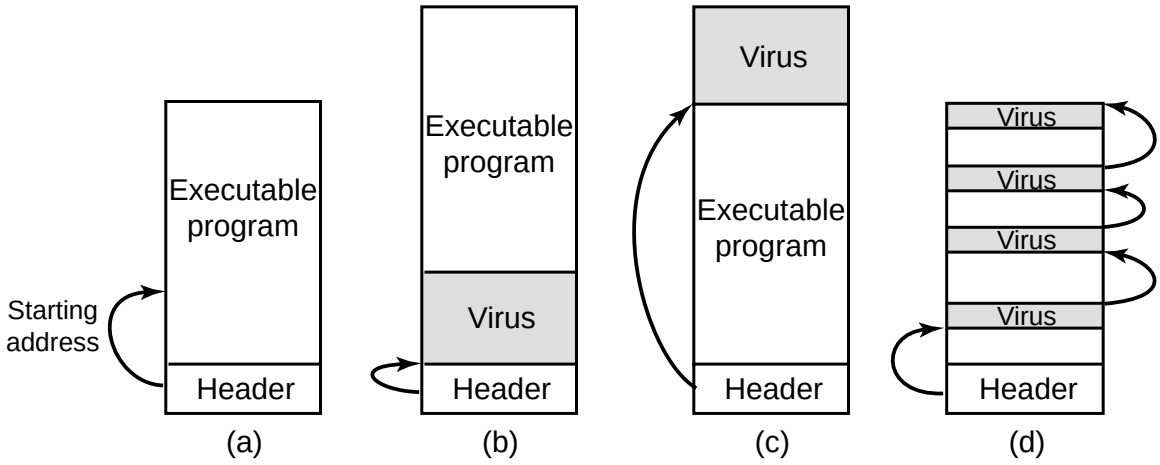

Fig. 9-14. (a) An executable program. (b) With a virus at the front. (c) With a virus at the end. (d) With a virus spread over free space within the program.

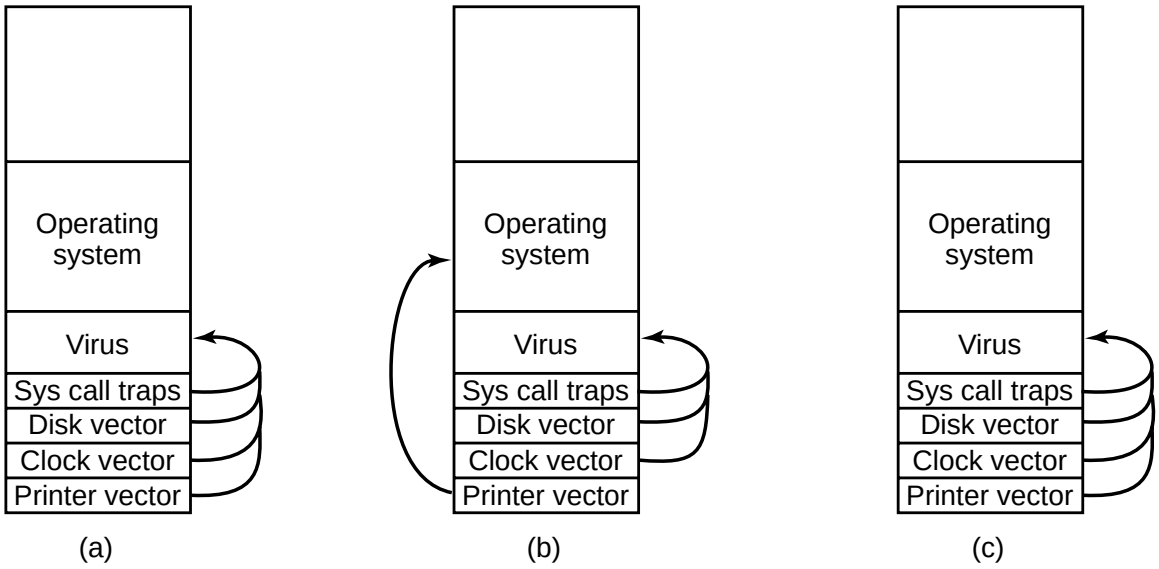

Fig. 9-15. (a) After the virus has captured all the interrupt and trap vectors. (b) After the operating system has retaken the printer interrupt vector. (c) After the virus has noticed the loss of the printer interrupt vector and recaptured it.

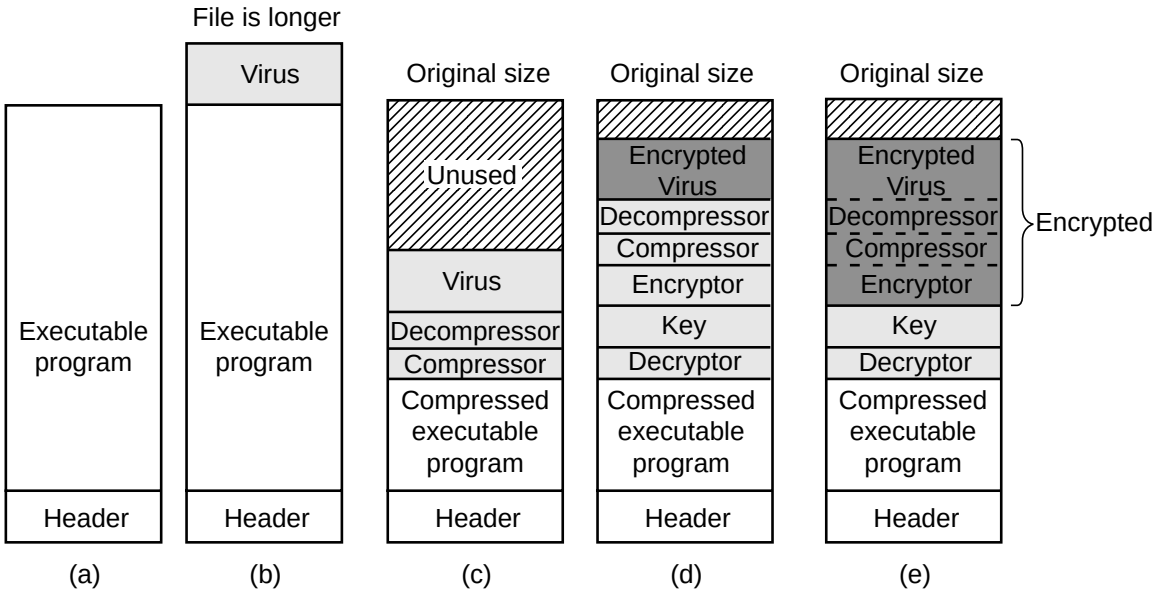

Fig. 9-16. (a) A program. (b) An infected program.
(c) A compressed infected program. (d) An encrypted virus. (e) A compressed virus with encrypted compression code.

```
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)

Fig. 9-17. Examples of a polymorphic virus.

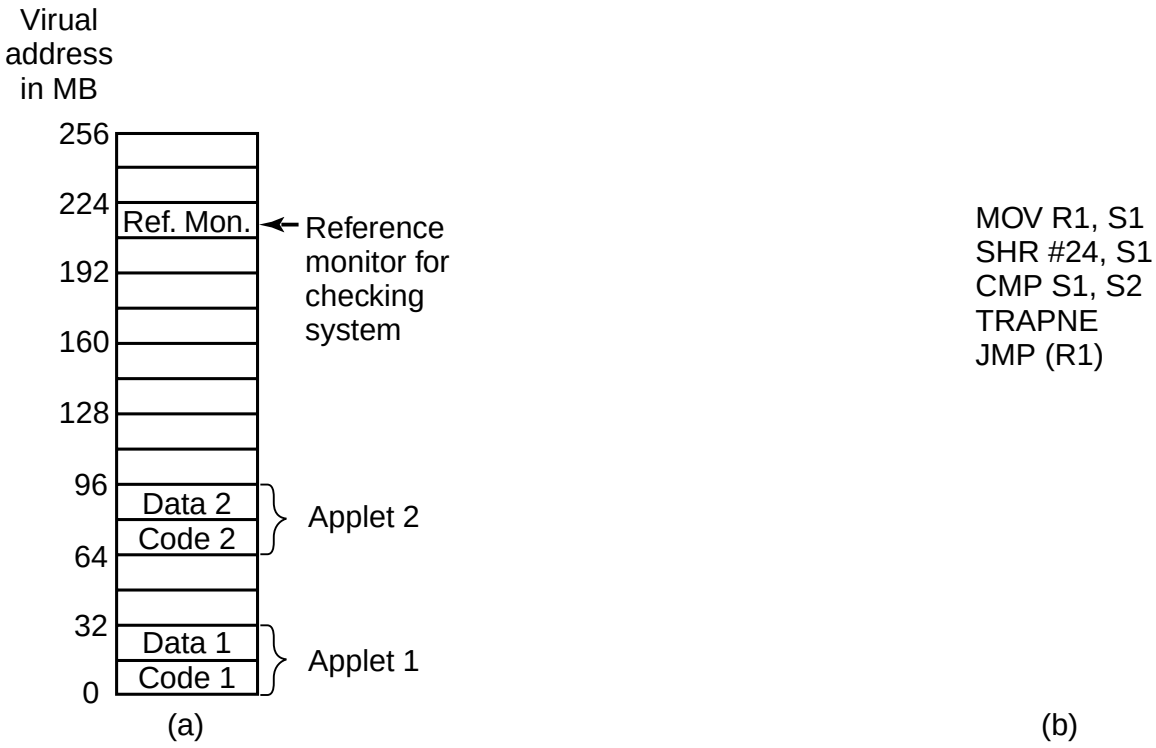

Fig. 9-18. (a) Memory divided into 16-MB sandboxes. (b) One way of checking an instruction for validity.

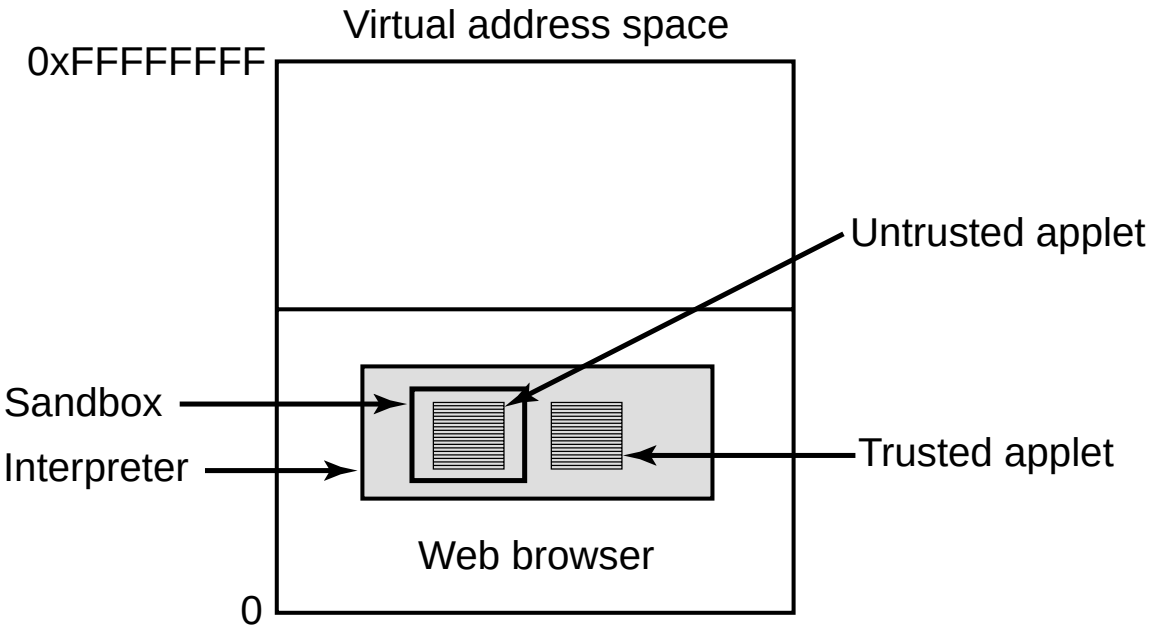

Fig. 9-19. Applets can be interpreted by a Web browser.

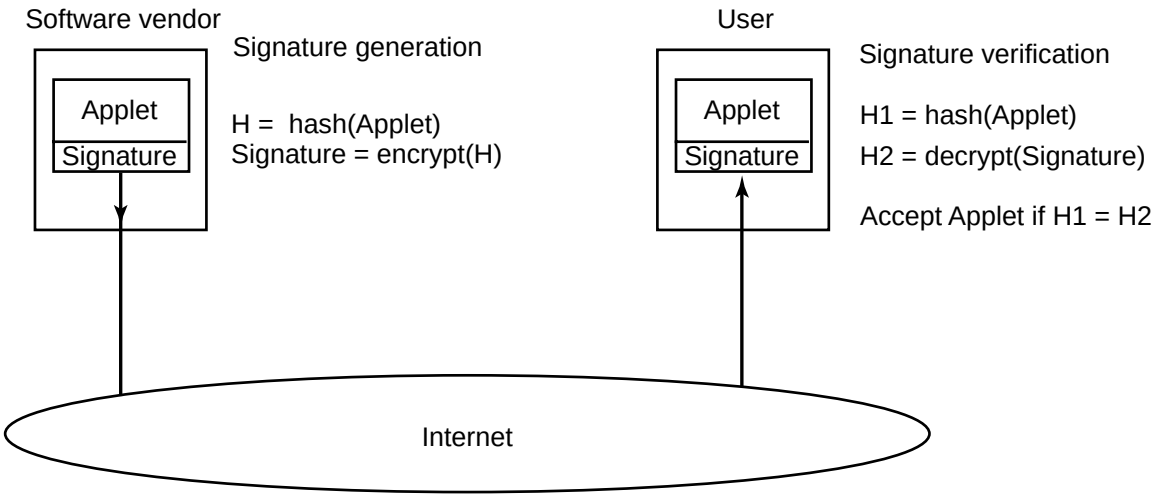

Fig. 9-20. How code signing works.

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

Fig. 9-21. Some examples of protection that can be specified with JDK 1.2.

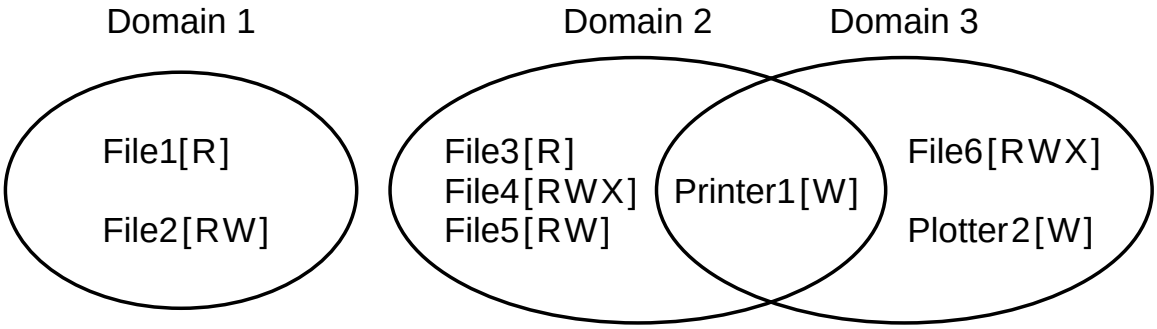

Fig. 9-22. Three protection domains.

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

Fig. 9-23. A protection matrix.

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

Fig. 9-24. A protection matrix with domains as objects.

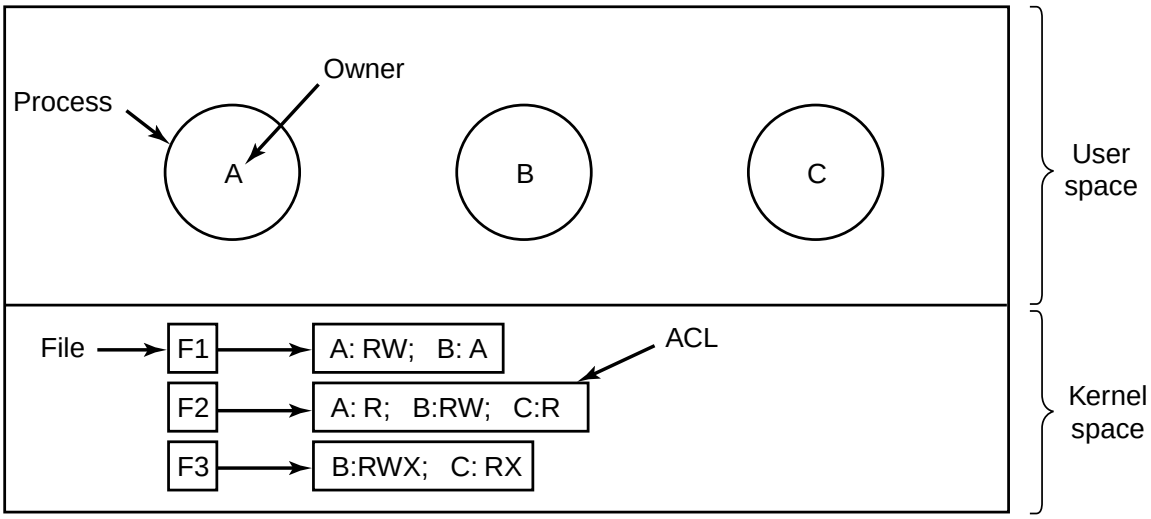

Fig. 9-25. Use of access control lists to manage file access.

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

Fig. 9-26. Two access control lists.

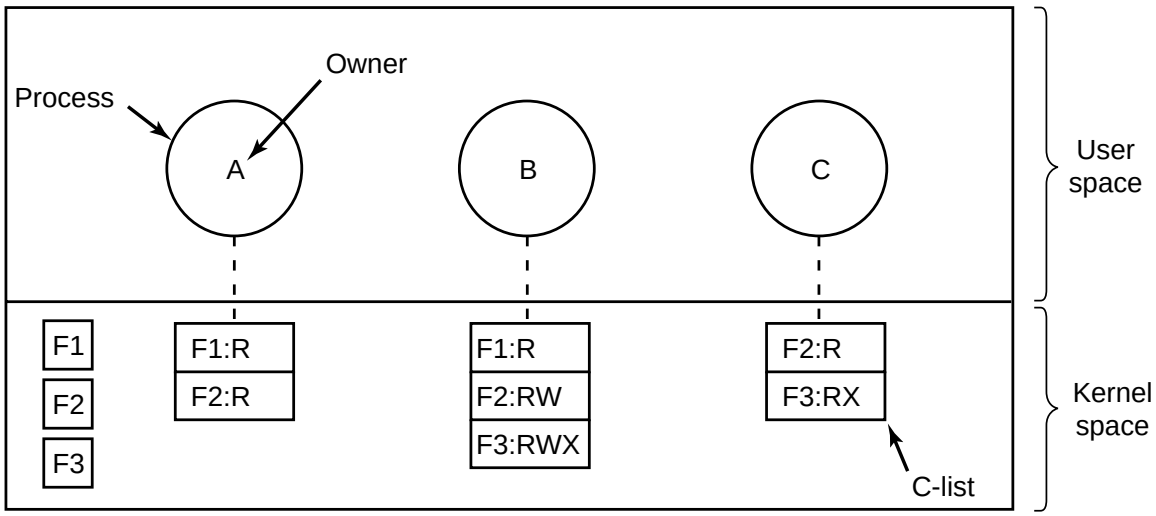

Fig. 9-27. When capabilities are used, each process has a capability list.

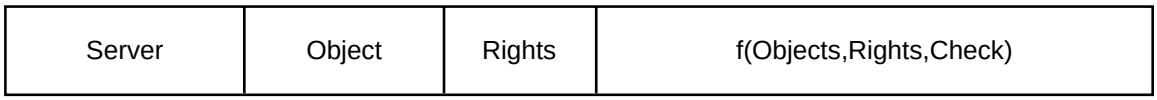

Fig. 9-28. A cryptographically-protected capability.

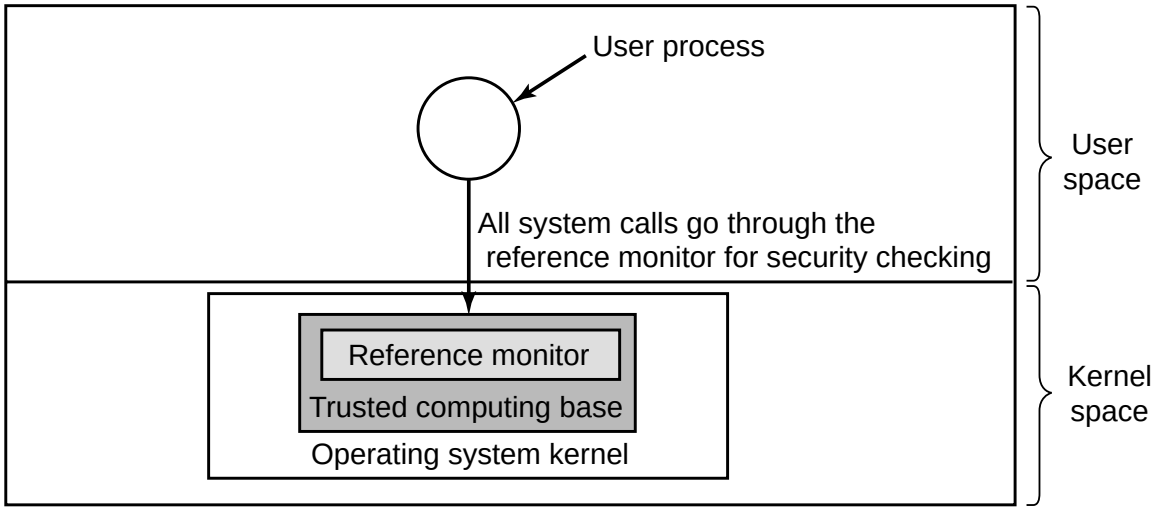

Fig. 9-29. A reference monitor.

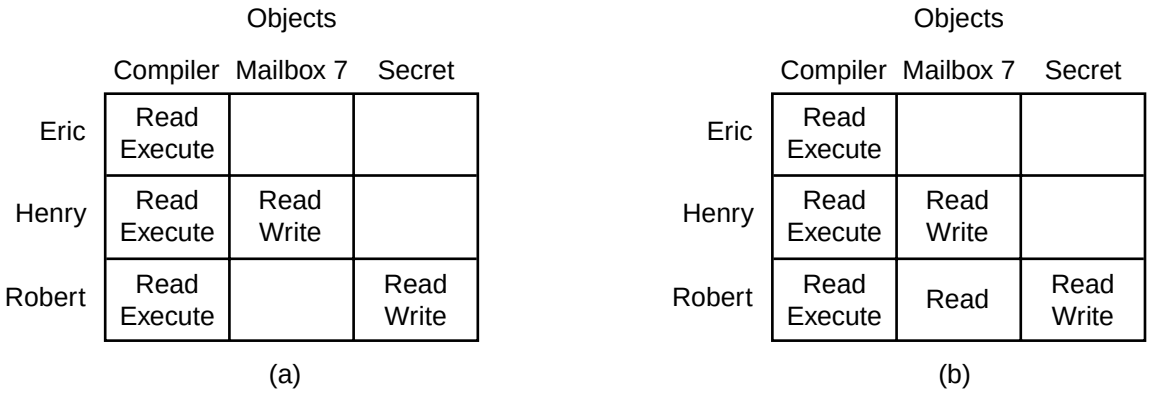

Fig. 9-30. (a) An authorized state. (b) An unauthorized state.

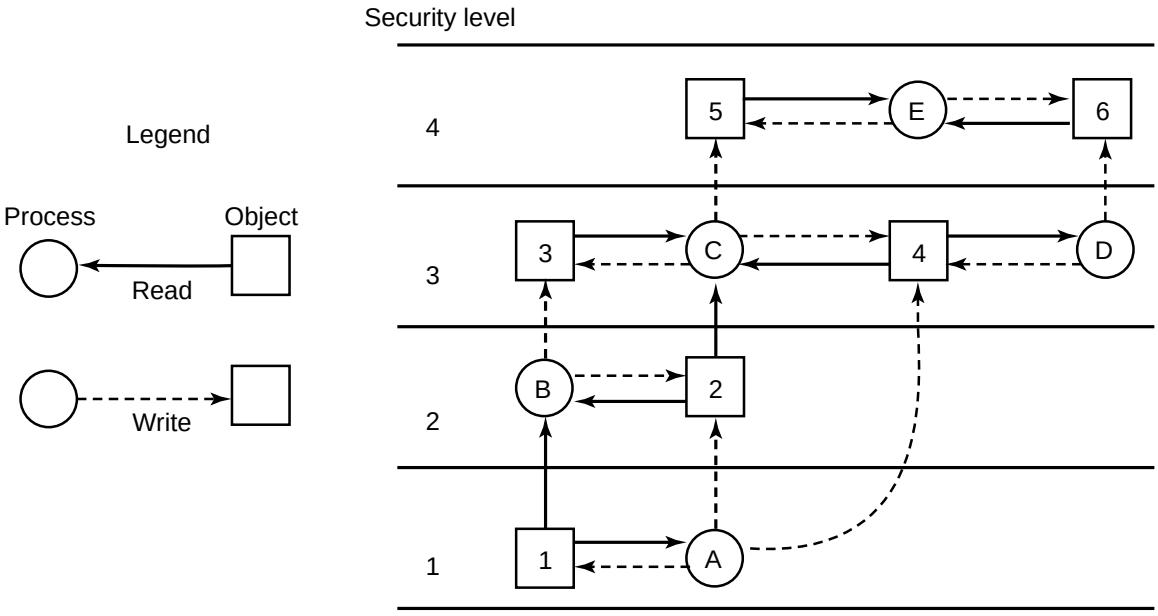

Fig. 9-31. The Bell-La Padula multilevel security model.

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	→
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

Fig. 9-32. Orange Book security criteria. The symbol X means that there are new requirements here. The symbol → means that the requirements from the next lower category also apply here.

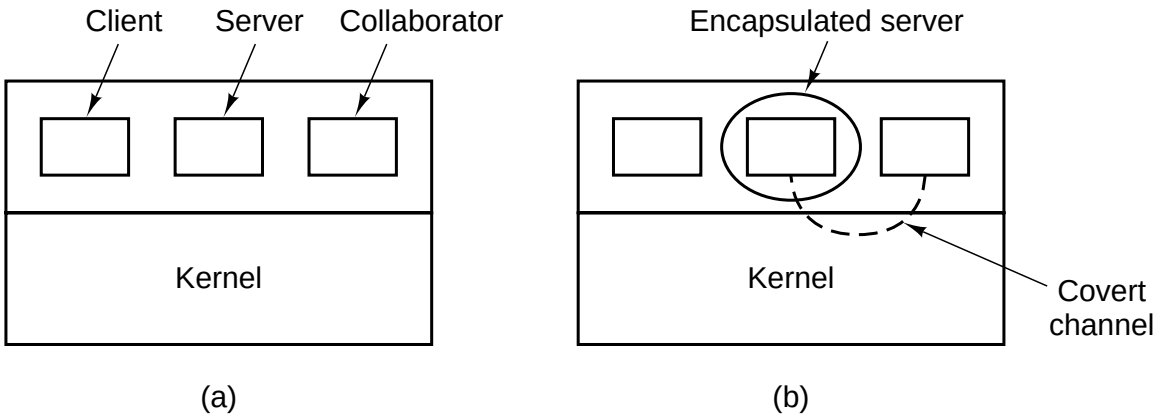

Fig. 9-33. (a) The client, server, and collaborator processes. (b) The encapsulated server can still leak to the collaborator via covert channels.

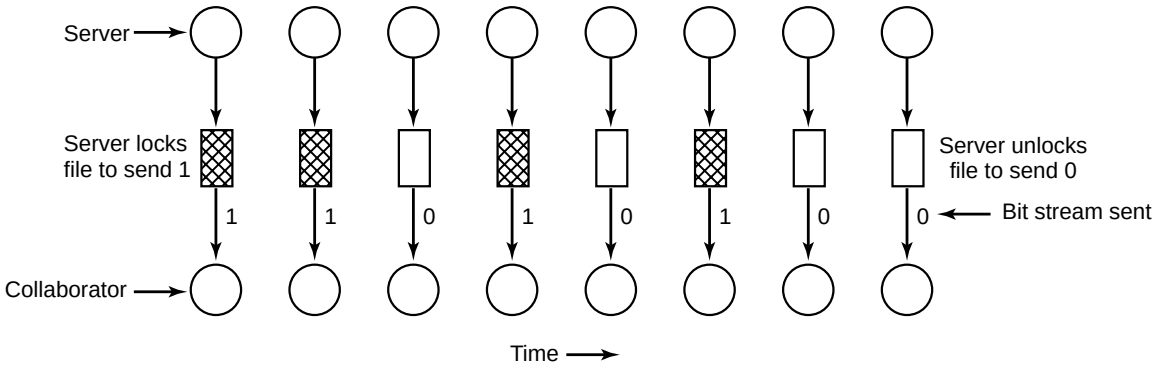

Fig. 9-34. A covert channel using file locking.

(a) (b)

Fig. 9-35. (a) Three zebras and a tree. (b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.