Security

EECE6029

Yizong Cheng

3/4/2016

Vulnerabilities and Exploits

- Valuable information is stored in computer systems.
- Guarding the information against unauthorized usage is a major concern of all operating systems.
- Operating systems are increasingly bloated.
- When a software bug is a security bug, it is a vulnerability.
- Bug-triggering input is called an exploit.
- Successful exploits allow attackers to take full control of the computer machine.

Components of Security

- Security: Files are not read or modified by unauthorized persons.
- Confidentiality: Owner of a file may specify who can see it, and the system should enforce the specifications.
  - sniffing the Wi-Fi traffic
  - privacy is another aspect of confidentiality
- Integrity: Unauthorized users should not be able to modify, remove and add any data without owner’s permission.
  - modifying records in a database
  - authenticity and nonrepudiability are other aspects of integrity
- Availability: Nobody can disturb the system to make it unusable.
  - denial-of-service attacks to Web sites

Trusted Computing Base (TCB)

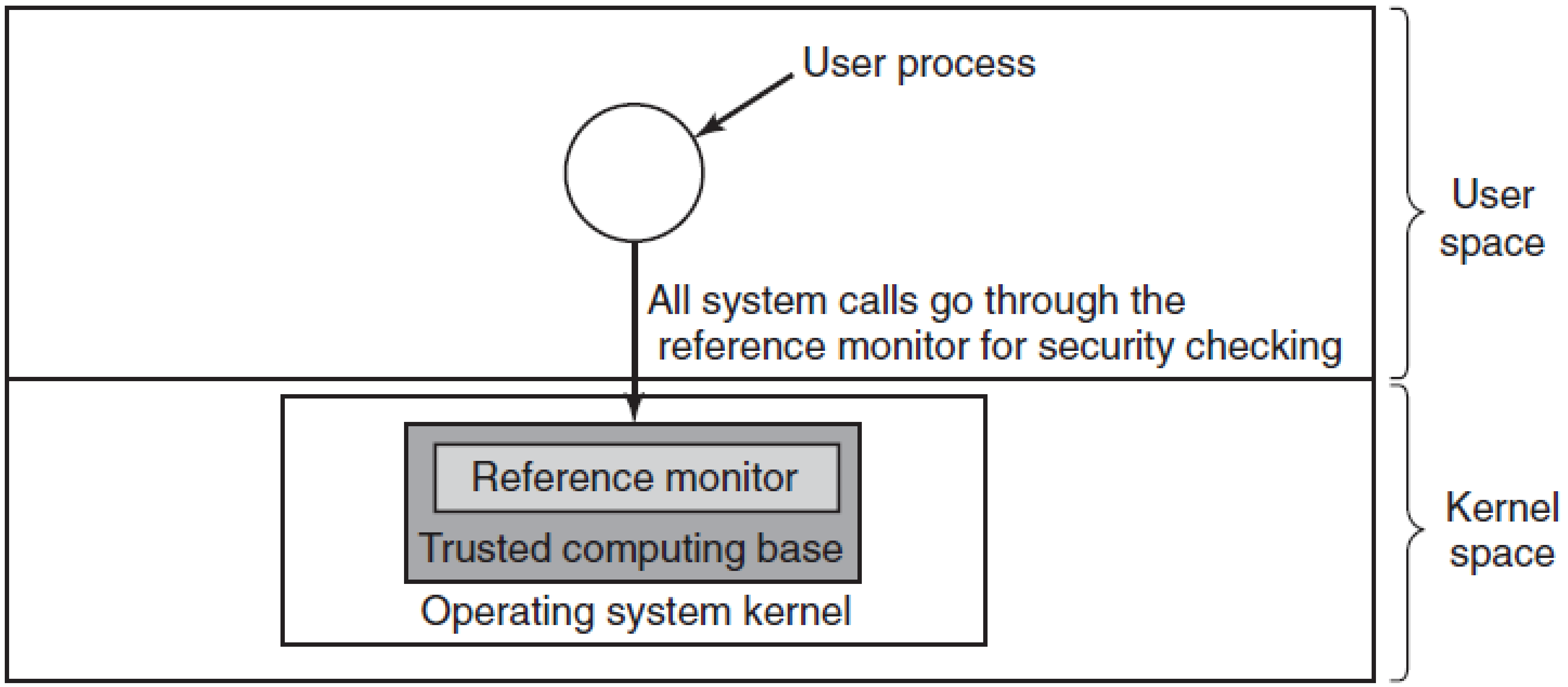


Figure 9-2. A reference monitor.

Protection Domains

- Principle of least authority (POLA): least to know to each user.
- A domain is a set of (object, rights), often corresponding to a single user.
- Objects are resources with unique names.
- Rights are permitted operations.
- UNIX defines domains by UID and GID
- Domain switch for a process: the UID and GID of a process may change if a file has the SETUID or SETGID bit on.

Protection Domains

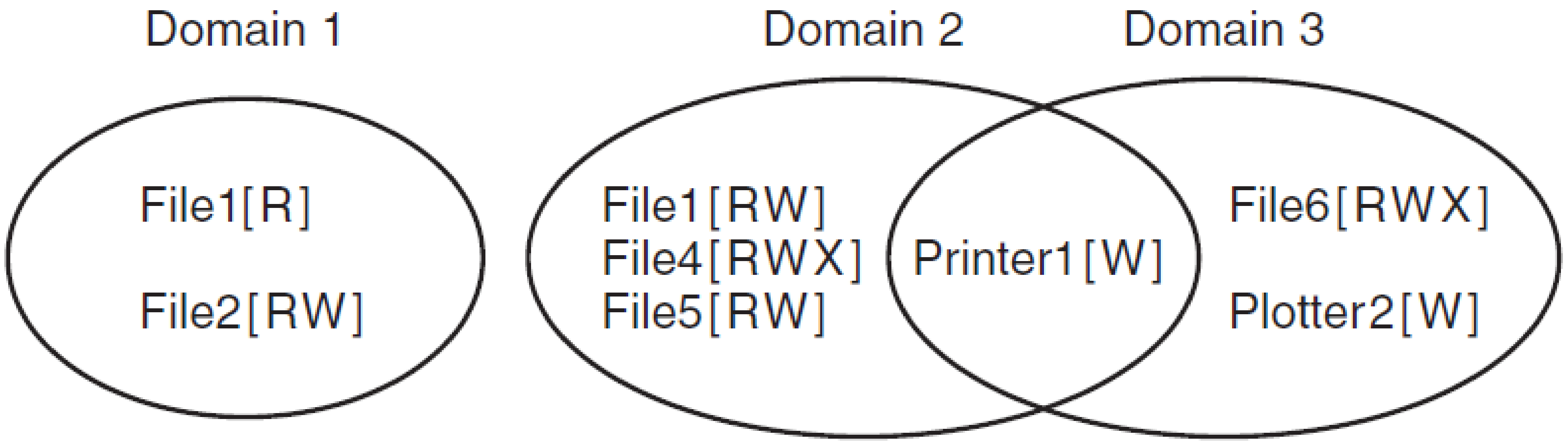
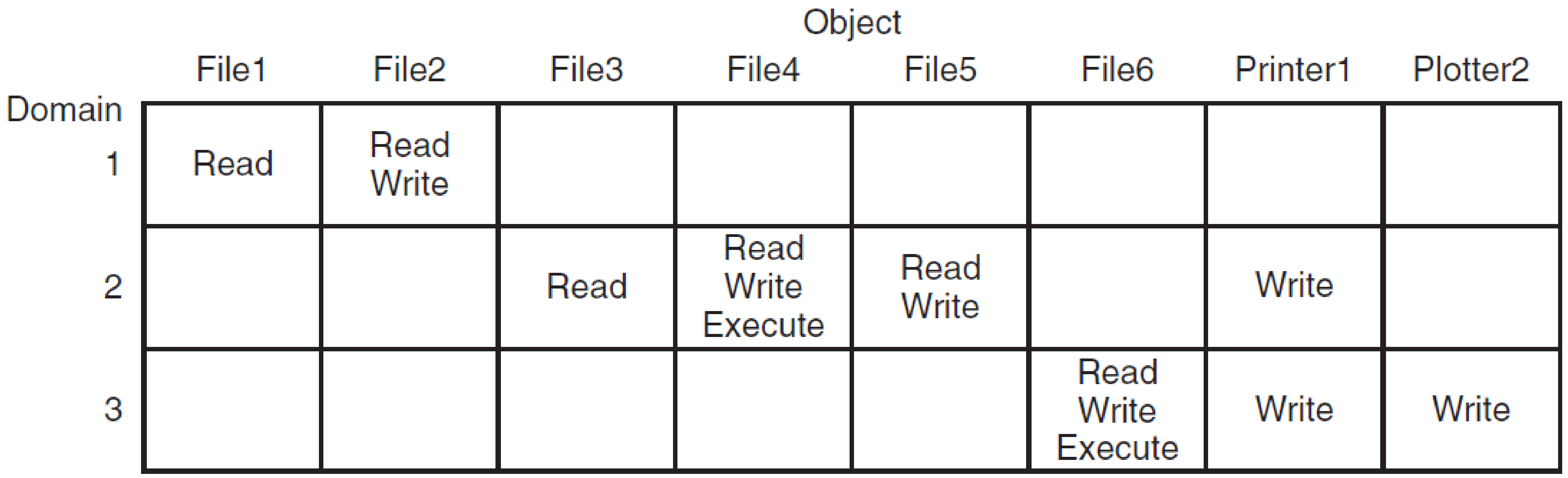


Figure 9-3. Three protection domains.

Protection Matrix

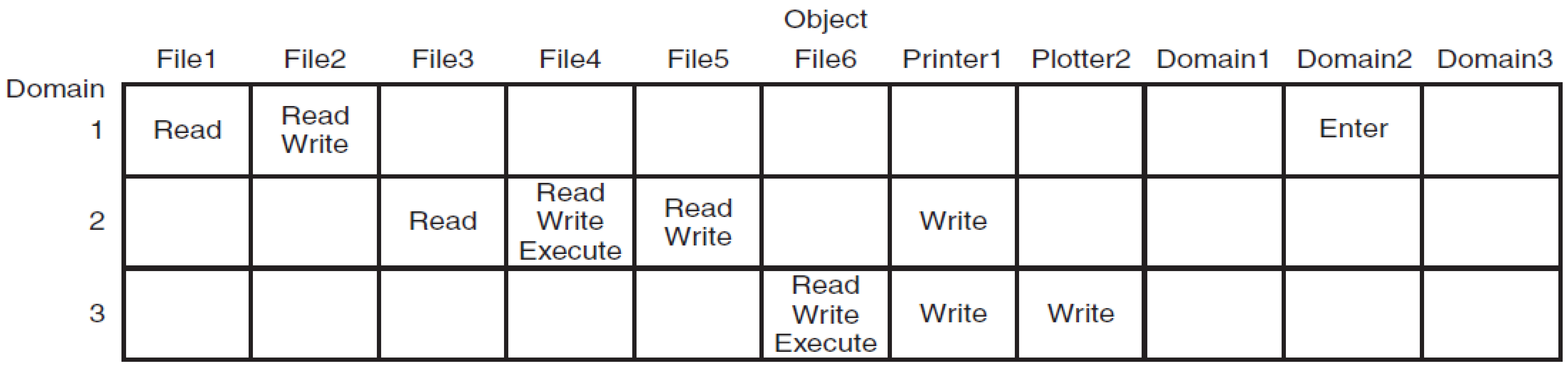


Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

Object

Figure 9-4. A protection matrix.

Domain Switch in Protection Matrix

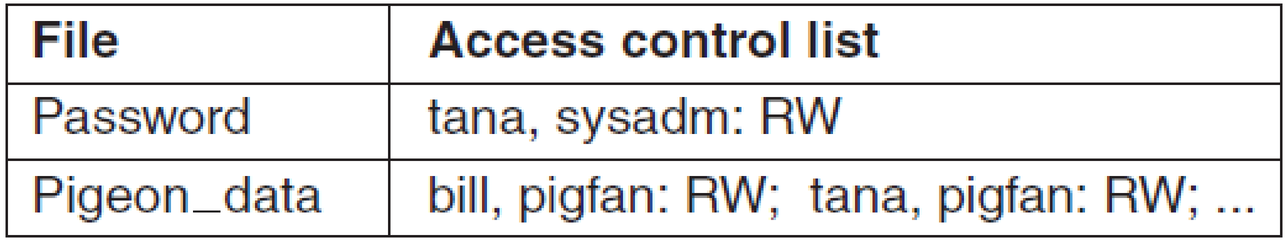


Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Figure 9-5. A protection matrix with domains as objects.

Storing the Protection Matrix

- The protection matrix (process/file) is big and sparse.
- May only store elements that are not empty.
- A linked list for each column/resource
  - access control list (ACL), used in Windows and UNIX
- A linked list for each row/process
  - capability list (C-list), L4 kernel is capability based, FreeBSD uses Capsicum



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

Figure 9-7. Two access control lists.

Access Control Lists

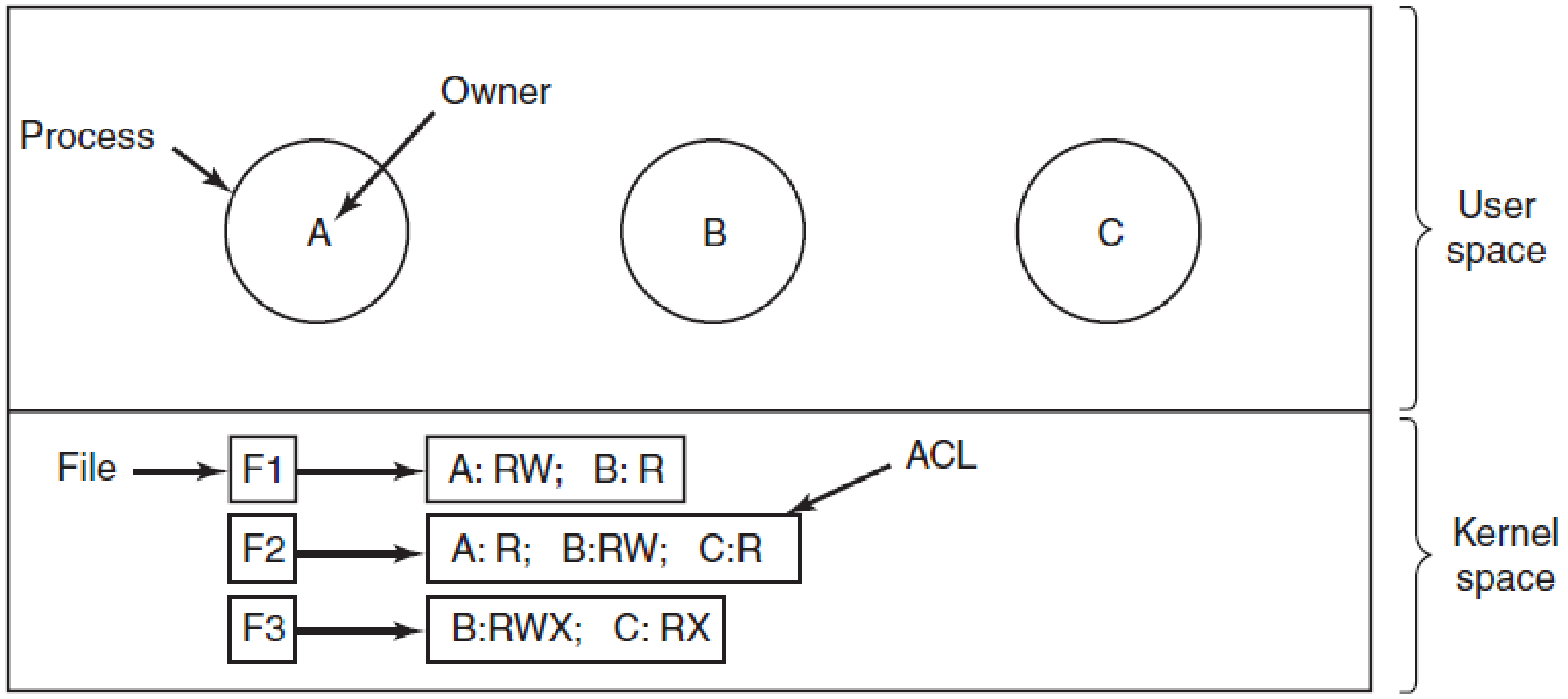


Figure 9-6. Use of access control lists to manage file access.

Capability Lists

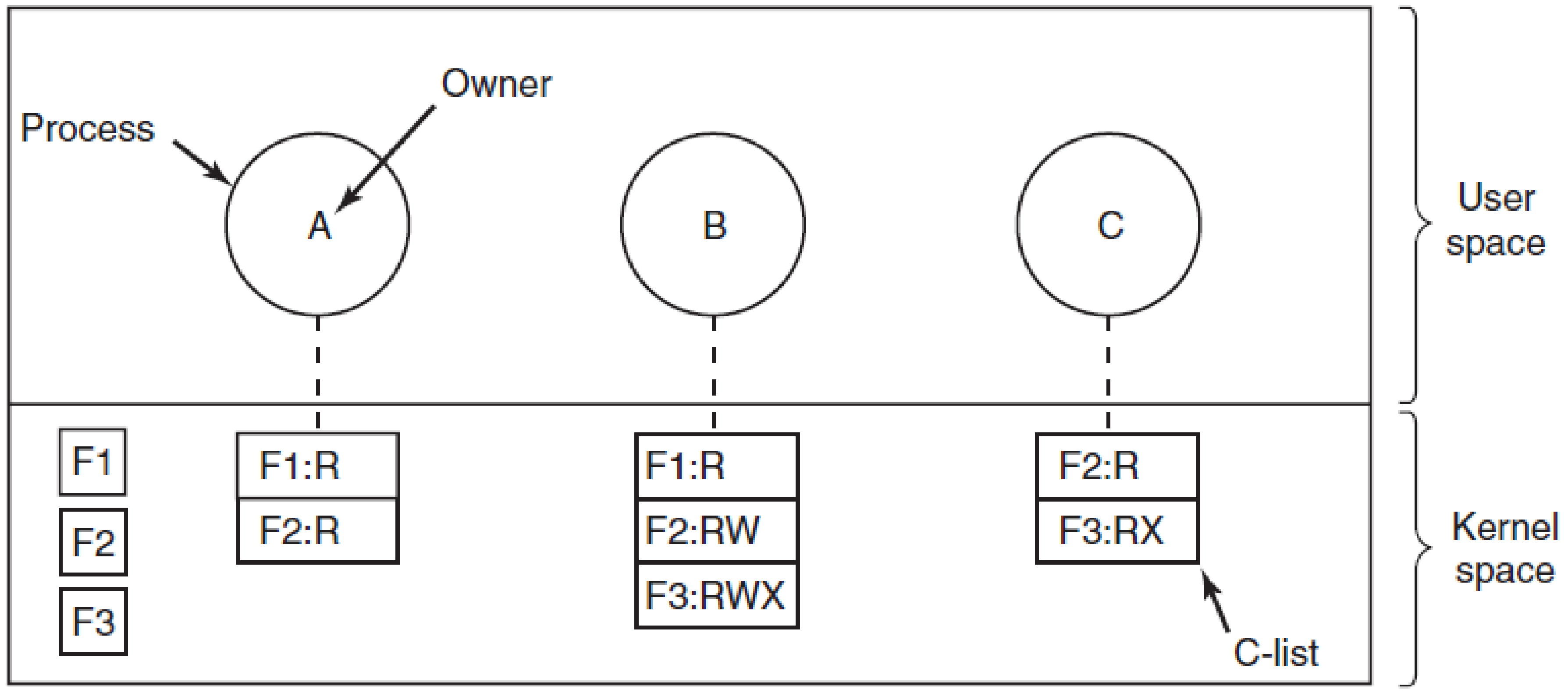


Figure 9-8. When capabilities are used, each process has a capability list.

Protecting the Capability Lists

- Tagged architecture (IBM AS/400): Each memory word has an extra bit that tells whether the word contains a capability or not.
  - can only be modified in kernel mode by the operating system
- Hydra keeps the C-list inside the operating system.
- In user space with cryptographically secure hash function f.

Server	Object	Rights	f(Objects, Rights, Check)

Figure 9-9. A cryptographically protected capability.

Multilevel Security

- The Bell-LaPadula model: read down and write up
  - A process running at security level k can read only objects at its level or lower.
  - It can only write objects at its level or higher.
- The Biba model: the reverse: no write up and no read down
  - A process running at security level k can write only objects at its level or lower.
  - It can read only objects at its level or higher.

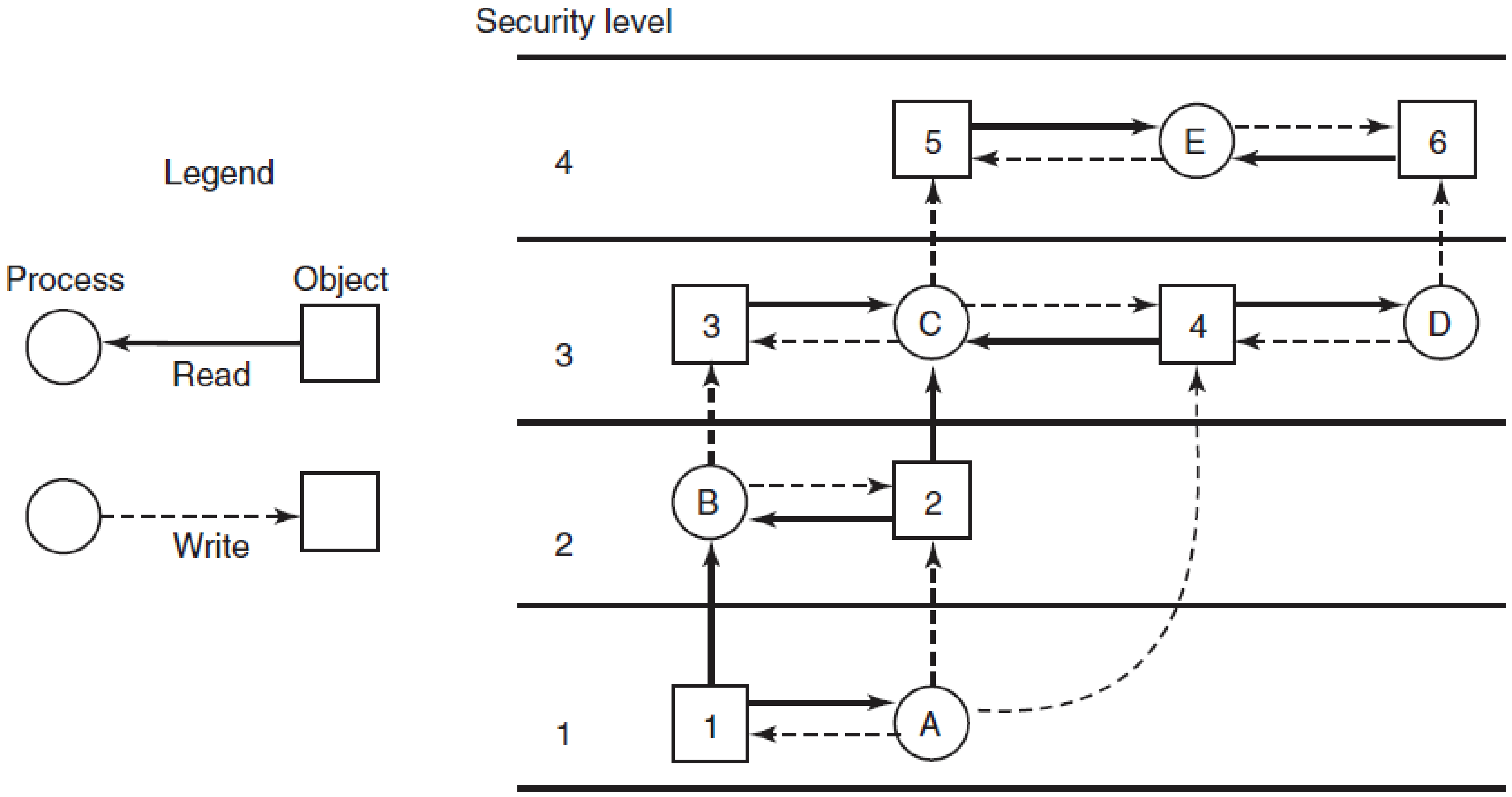


Figure 9-11. The Bell-LaPadula multilevel security model.

Covert Channels

- Security models may not prevent processes from leaking information to collaborators.
- Confinement problem: how to prevent collaborator from getting information leaks over covert channels.
- Information can be leaked as any sequence of binary events.
    - A port maintained by a server problem becomes closed or open.
    - Any degrading system performance in a clocked system is a way to send information.

Covert Channels

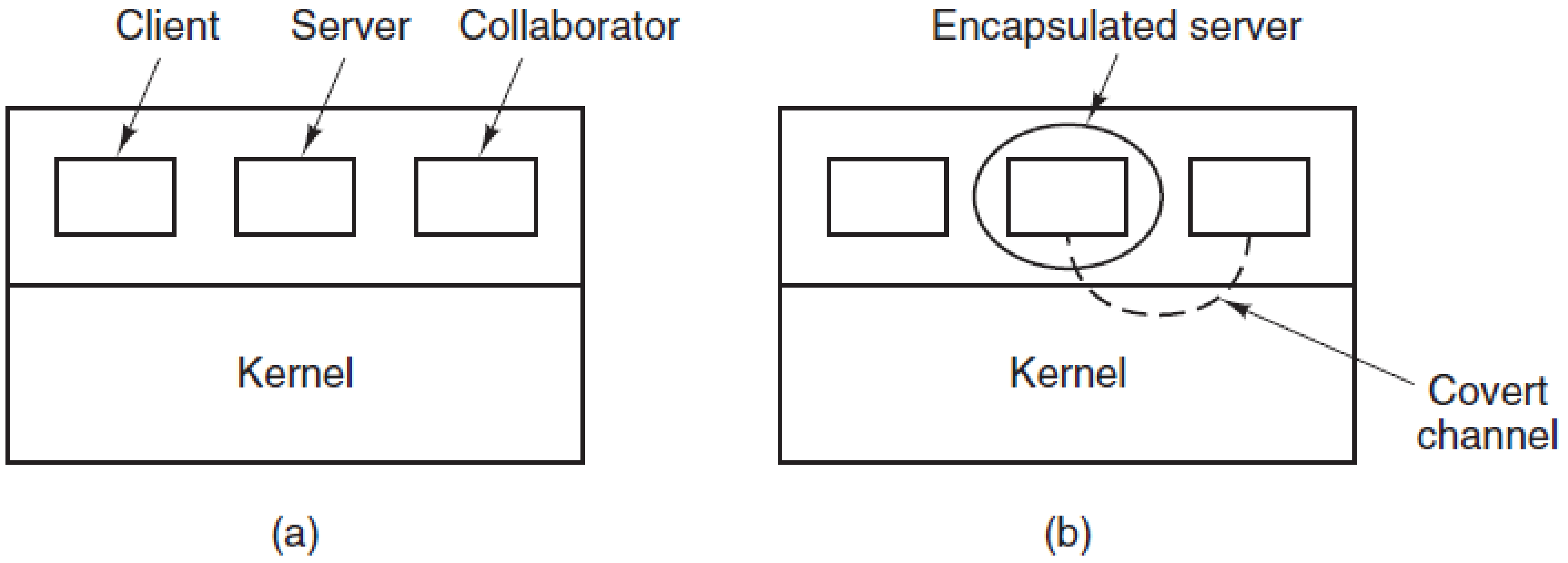


Figure 9-12. (a) The client, server, and collaborator processes. (b) The encapsulated server can still leak to the collaborator via covert channels.

Locking/Unlocking as 1/0

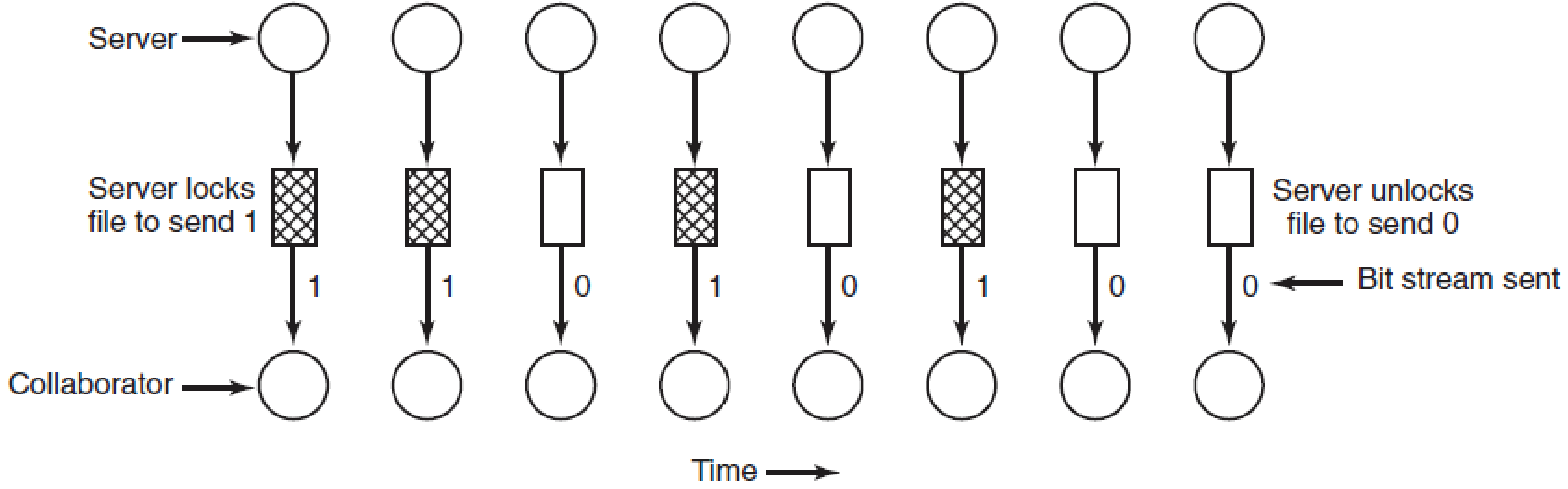


Figure 9-13. A covert channel using file locking.

Encryption

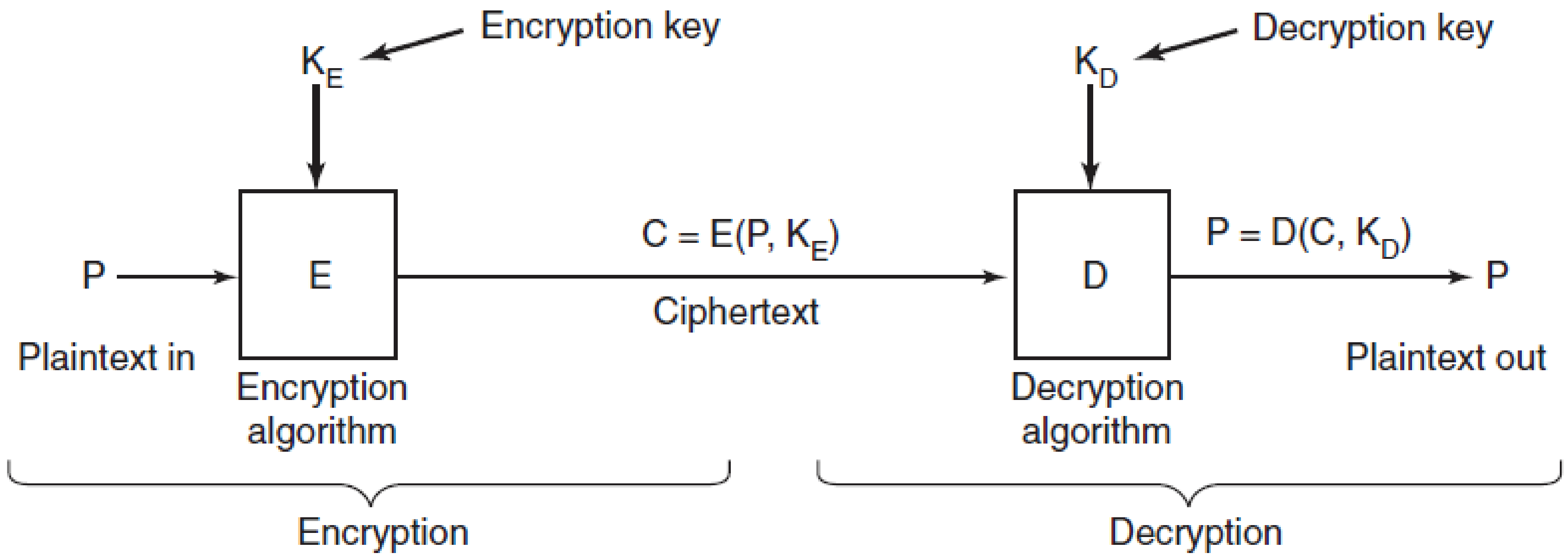


Figure 9-15. Relationship between the plaintext and the ciphertext.

Digital Signatures

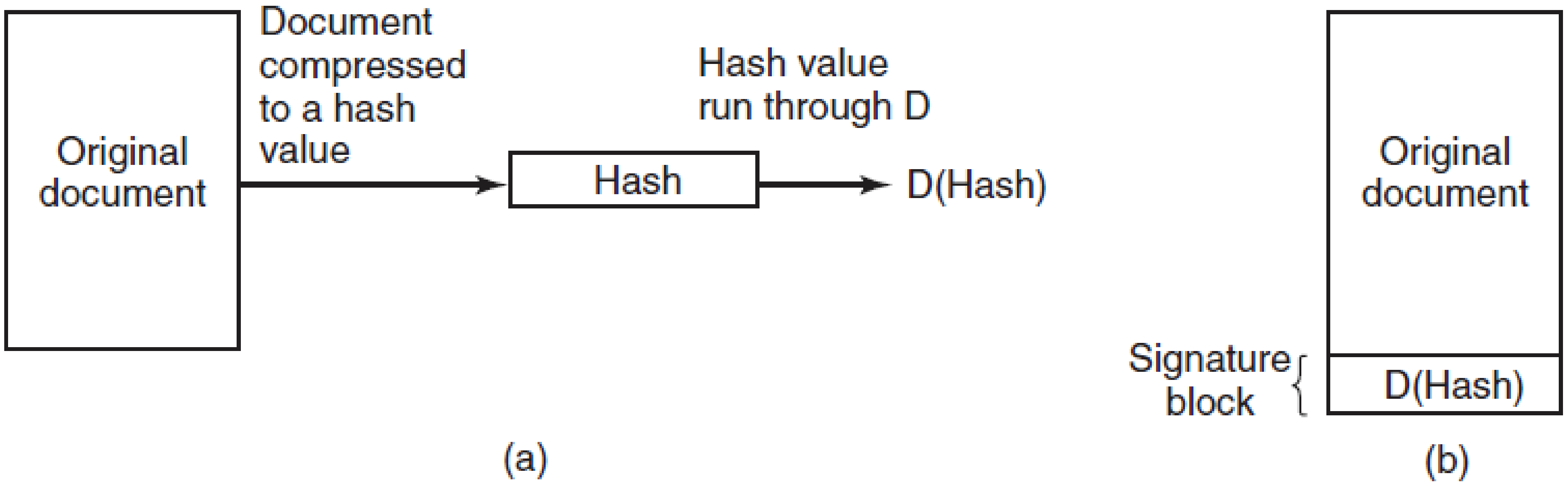


Figure 9-16. (a) Computing a signature block. (b) What the receiver gets.

Trusted Platform Modules (TPMs)

- a chip in the computer that is a cryptoprocessor with some nonvolatile storage inside for keys, required in USDOD
- remote attestation: an external party to verify that the computer is running trusted software (not illegally copied or expired)
- BIOS initializes the platform configuration register (PCR).
- PCR cannot be overwritten directly but only extended.
    - TPM takes a hash of the combination of the input value and the previous value in the PCR.
- The external party sends a nonce and TPM uses it to extend the PCR and then hashes for the bootloader, the kernel, and the applications, so that the external party can match.

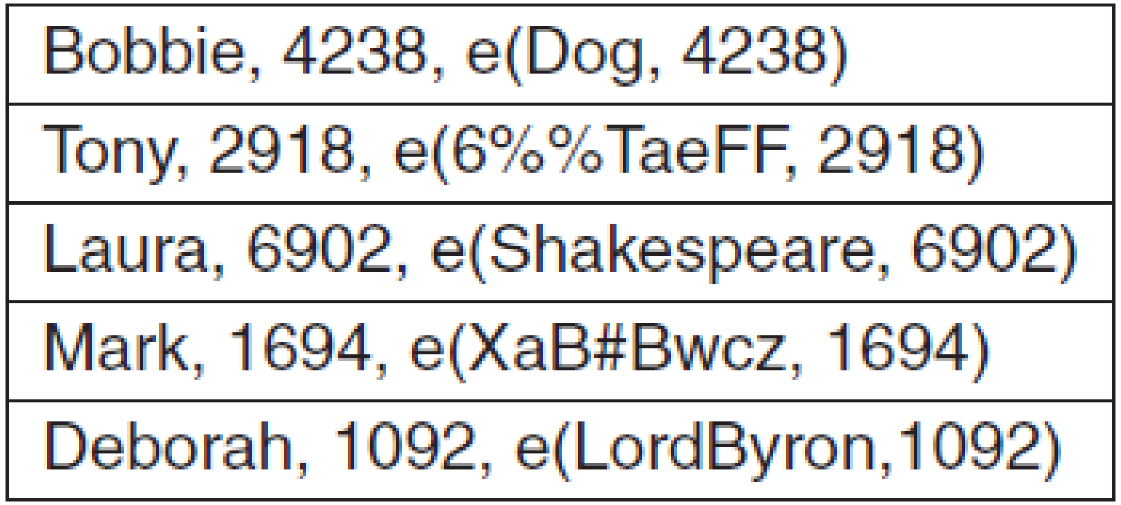
Trusted Computing

- loading the hardware with a unique encryption key inaccessible to the rest of the system.
- The hardware is not only secured for its owner, but also secured *against its owner.* (*treacherous computing*, Richard Stallman)
- The endorsement key is a 2048-bit RSA public and private key pair that is created randomly on the chip at manufacture time and cannot be changed. The private key never leaves the chip, while the public key is used for attestation.

User Authentication

- User login with username and password.
- Can be bypassed with BIOS reconfiguration.
- Original UNIX keeps a password file with all user passwords unencrypted for the superuser to see.
- UNIX then saves only the hash of the password in the password file.
- A cracker may prepare a list of likely passwords and their hash and then used the hashed version to reveal the password.
- Salt is used to defeat this.

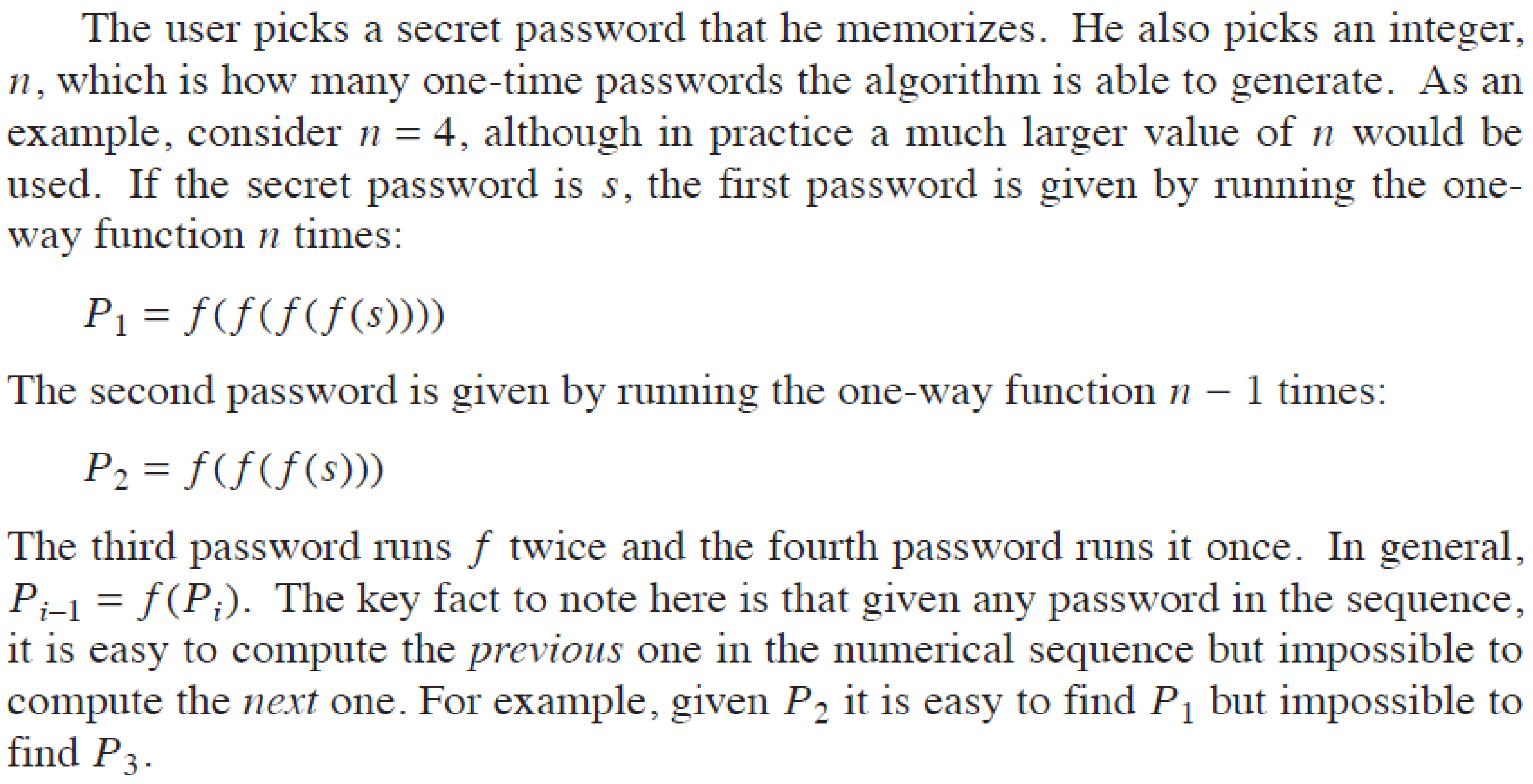
Salted Password File



Bobbie, 4238, e(Dog, 4238)
Tony, 2918, e(6%%TaeFF, 2918)
Laura, 6902, e(Shakespeare, 6902)
Mark, 1694, e(XaB#Bwcz, 1694)
Deborah, 1092, e(LordByron,1092)

Figure 9-18. The use of salt to defeat precomputation of encrypted passwords.

One-Time Passwords



The user picks a secret password that he memorizes. He also picks an integer, n, which is how many one-time passwords the algorithm is able to generate. As an example, consider $n = 4$, although in practice a much larger value of n would be used. If the secret password is s, the first password is given by running the one-way function n times:

$$P_1 = f(f(f(f(s))))$$

The second password is given by running the one-way function $n - 1$ times:

$$P_2 = f(f(f(s)))$$

The third password runs f twice and the fourth password runs it once. In general, $P_{i-1} = f(P_i)$. The key fact to note here is that given any password in the sequence, it is easy to compute the *previous* one in the numerical sequence but impossible to compute the *next* one. For example, given P_2 it is easy to find P_1 but impossible to find P_3.

The server is initialized with P_0, which is just $f(P_1)$. This value is stored in the password file entry associated with the user's login name along with the integer 1, indicating that the next password required is P_1. When the user wants to log in for the first time, he sends his login name to the server, which responds by sending the integer in the password file, 1. The user's machine responds with P_1, which can be computed locally from s, which is typed in on the spot. The server then computes $f(P_1)$ and compares this to the value stored in the password file (P_0). If the values match, the login is permitted, the integer is incremented to 2, and P_1 overwrites P_0 in the password file.

On the next login, the server sends the user a 2, and the user's machine computes P_2. The server then computes $f(P_2)$ and compares it to the entry in the password file. If the values match, the login is permitted, the integer is incremented to 3, and P_2 overwrites P_1 in the password file. The property that makes this scheme work is that even though an intruder may capture P_i, he has no way to compute P_{i+1} from it, only P_{i-1} which has already been used and is now worthless. When all n passwords have been used up, the server is reinitialized with a new secret key.