Security Overview

W4118 Operating Systems I

columbia-os.github.com

The Security Environment Threats

Goal	Threat
Confidentiality	Exposure of data
Integrity	Tampering with data
Availability	Denial of service

Security goals and threats.

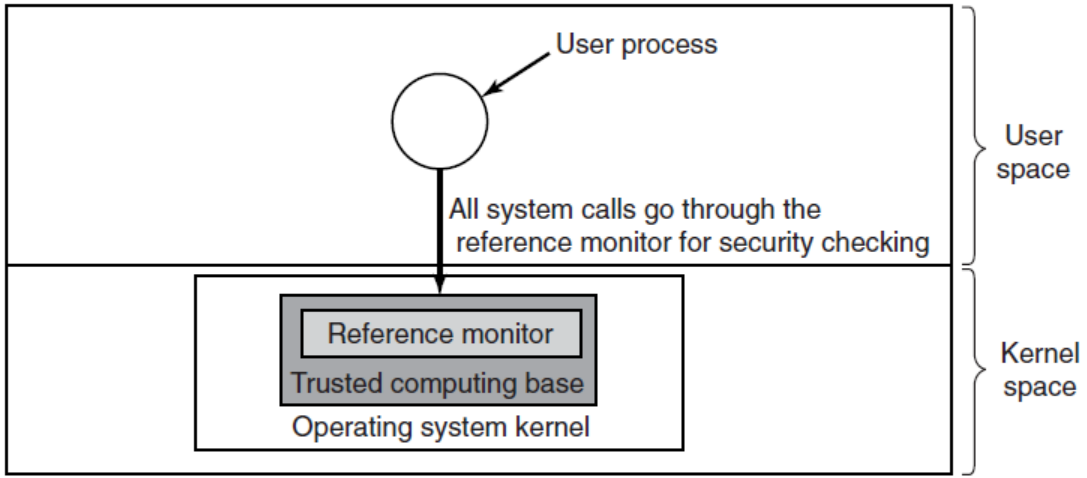
Can We Build Secure Systems?

Two questions concerning security:

1. Is it possible to build a secure computer system?

2. If so, why is it not done?

Trusted Computing Base



A reference monitor.

Protection Domains (1)

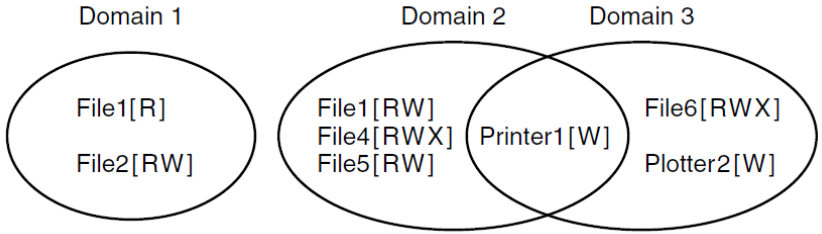
- A **protection domain** is a conceptual framework that defines the set of objects (such as files, memory segments, or devices) a process can access and the operations (read, write, execute) it can perform on them.

- **Association:** In Unix, a protection domain is typically associated with the **User ID (UID)** and **Group ID (GID)** of a process.

- **Switching:** A process can change its protection domain through mechanisms like the setuid bit, which allows a program to run with the privileges of the file's owner (often root).

- **Scope:** It defines the "sandbox" or boundaries within which a subject operates

Protection Domains (1)



Three protection domains.

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.

Access Control (1)

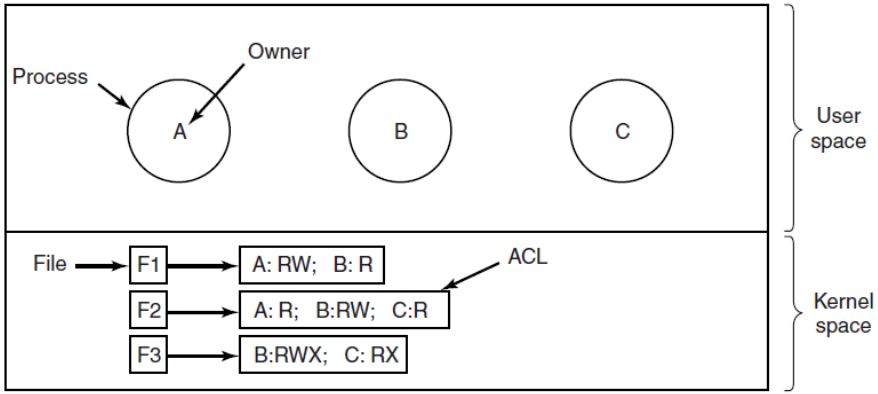
- **Access control** refers to the specific mechanisms and policies used to enforce the rules defined by a protection domain.

- **Unix implementation:** Traditionally, Unix uses **Discretionary Access Control (DAC)**, where the owner of an object decides who has access.

- **Methods:**
  - **File Permissions:** The classic "rwx" (read, write, execute) bits for User, Group, and Others.
  - **Access Control Lists (ACLs):** More modern Unix systems support Full ACLs, allowing specific permissions for multiple individual users or groups beyond the standard "ugo" model.

- **Concept:** It is often visualized as a **column** in an Access Control Matrix, where permissions are attached to the **object** (e.g., a file).

Access Control Lists (2)



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

Two access control lists.

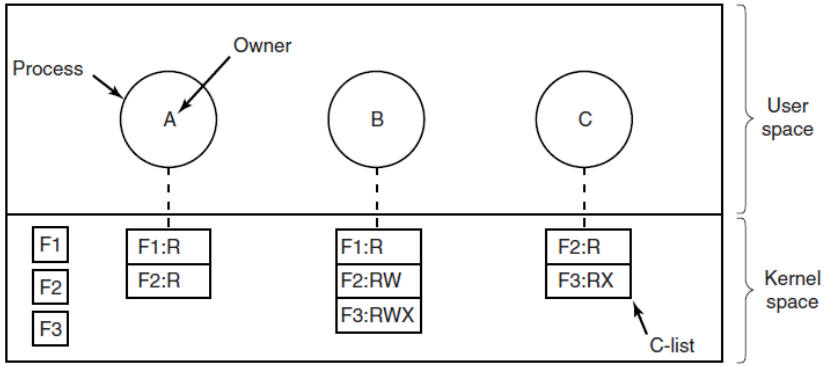
Use of access control lists to manage file access.

```
int setxattr(size_t size;
             const char *path, const char *name,
             const void value[size], size_t size, int flags);
int lsetxattr(size_t size;
              const char *path, const char *name,
              const void value[size], size_t size, int flags);
int fsetxattr(size_t size;
              int fd, const char *name,
              const void value[size], size_t size, int flags);
```

Capabilities (1)

- **Capabilities** are a more granular way to assign specific privileges to processes without giving them full root access.
- **Fine-grained control:** Instead of a binary "root or not-root" model, capabilities split traditional root privileges into smaller, distinct units. For example:
  - CAP_NET_BIND_SERVICE: Allows a process to bind to ports below 1024.
  - CAP_SYS_TIME: Allows a process to set the system clock.
- **Tokens of Authority:** A capability is like an unforgeable "token" or "key" that a subject holds.
- **Concept:** In an Access Control Matrix, capabilities represent a **row**, showing all the rights a particular **subject** (process) holds.

Capabilities (2)



Server	Object	Rights	f(Objects,Rights,Check)

A cryptographically protected capability.

```
getcap /usr/bin/my_tcpdump

sudo setcap 'cap_net_raw,cap_net_admin=eip' /usr/bin/my_tcpdump
```

Summary Security Mechanism

Summary Comparison Table

Feature	Protection Domain	Access Control (Standard Unix)	Capabilities (Modern Unix/ Linux)
Focus	The boundaries of a subject's power.	Who can access a specific object.	What specific privileged actions a process can do.
Identity	Tied to UID/GID.	Tied to the Object (File/Directory).	Tied to the Subject (Process/ Thread).
Granularity	Coarse (User level).	Moderate (User/ Group/Other).	Fine-grained (Specific system privileges).
Matrix View	The entire environment.	**Column** (Object-centric).	**Row** (Subject-centric).

SELinux

- Security-Enhanced Linux (SELinux) is a security mechanism for the Linux kernel that provides **Mandatory Access Control (MAC)**
- It goes beyond traditional file permissions (i.e. –rwx).
- It relies only on user ownership and uses **security labels** to determine if a process can access a file, directory, or port

- **Example : httpd**
  - **Context Labels:** The Apache process is labeled with a specific "type" called httpd_t.
  - **Strict Access:** SELinux policy rules only allow httpd_t to access files labeled with web content types, like httpd_sys_content_t (found in /var/www/html/).
  - **Security Result:** Even if an attacker gains full control of the Apache process, they cannot read files in /tmp (labeled tmp_t) or /home because there is no rule allowing httpd_t to interact with those labels.

Linux Seccomp

- Seccomp is a Linux facility that limits access to a subset of system call in order to limit vulnerabilities or exploits.
- E.g. older most restricted mode only allows 4 calls
  `read()`: Allowed only from already-open filedescriptors.
  `write()`: Allowed only to already-open file descriptors.
  `_exit()`: To terminate the process.
  `sigreturn()`: To return from a signal handler

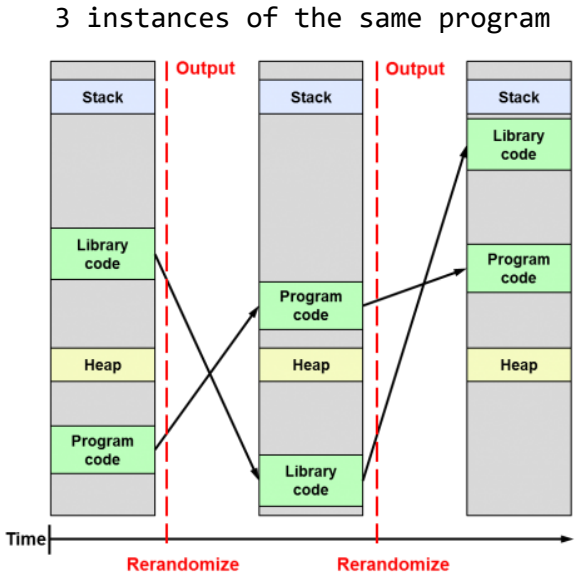
- It is a one way transition into “secure mode”
- Process calls prctl(PR_SET_SECCOMP, ...) or seccomp()
- Nowadays it is implemented using BPF/eBPF filters.

- Example usage: webbrowsers, containers -> policy driven

Address Space Layout Randomization

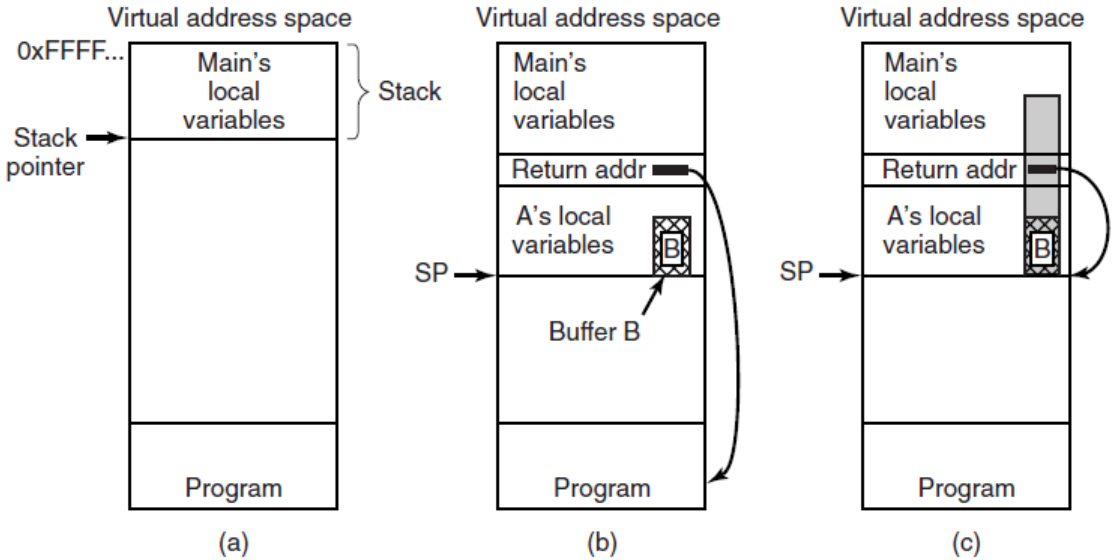
- ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.
- Available for user (ASLR) or the kernel (KASLR)
- Requires position independent code

- Prevent certain attacks
  like buffer overflow attacks.
- Makes predicting target addresses difficult



Buffer Overflow Attacks

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

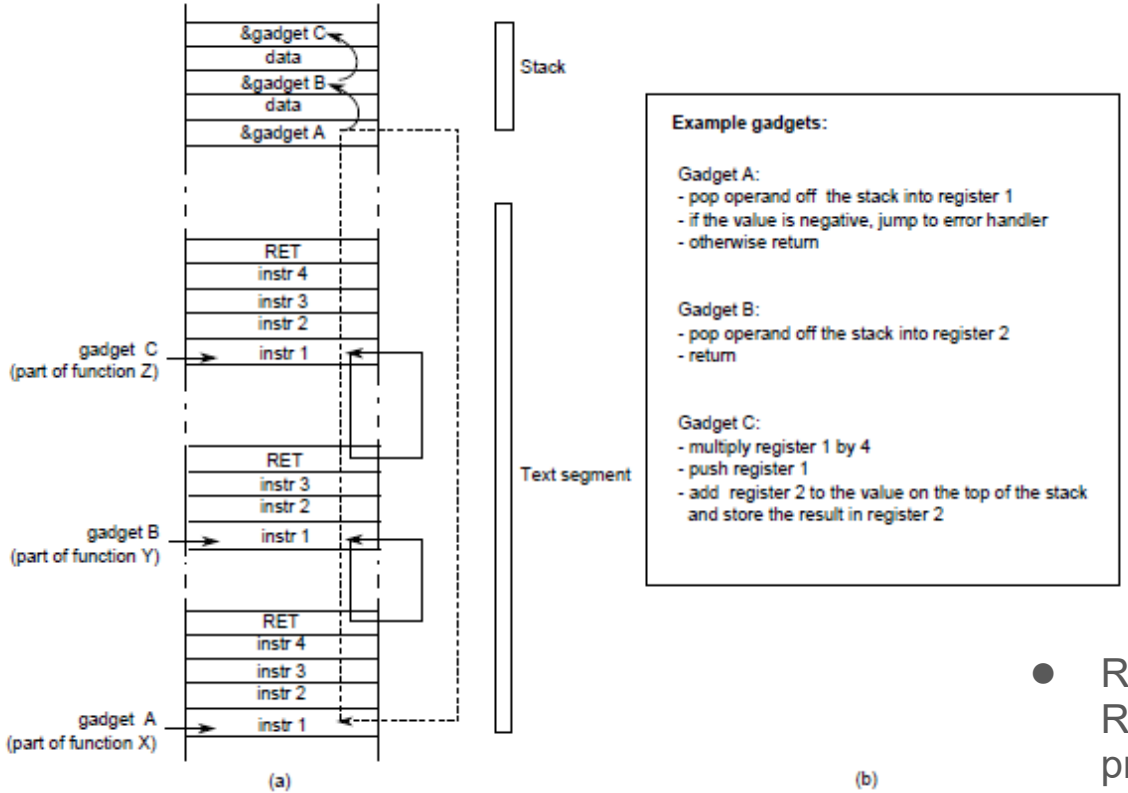


Avoiding Stack Canaries

- Skipping the stack canary: by modifying *len* first, the attack is able to bypass the canary and modify the return address directly.

```
01. void A (char *date) {
02.     int len;
03.     char B [128];
04.     char logMsg [256];
05.
06.     strcpy (logMsg, date);   /* first copy the string with the date in the log message */
07.     len = strlen (date);      /* determine how many characters are in the date string */
08.     gets (B);                 /* now get the actual message */
09.     strcpy (logMsg+len, B);/* and copy it after the date into logMessage */
10.     writeLog (logMsg);        /* finally, write the log message to disk */
11. }
```

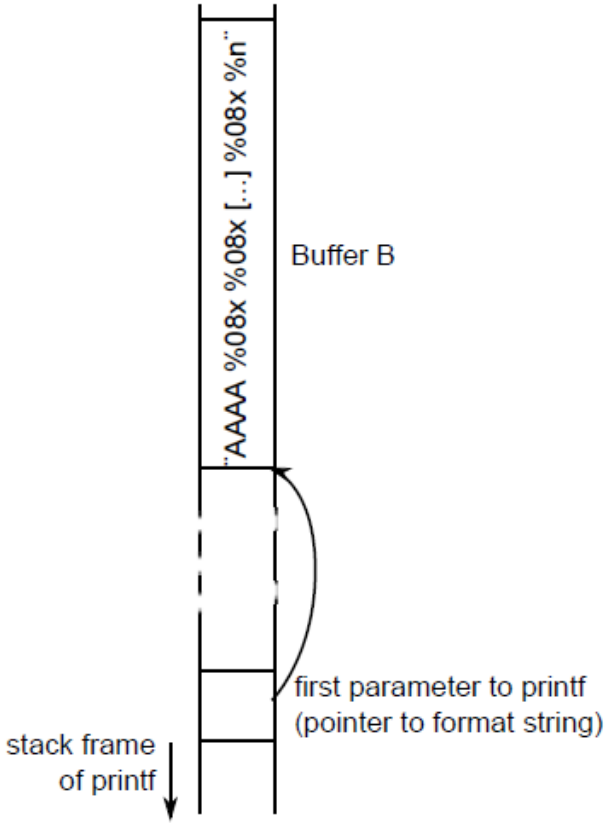
Code Reuse Attacks



- ROP
  Return-oriented programming
  linking gadgets

Format String Attacks

- A format string attack. By using exactly the right number of %08x, the attacker can use the first four characters of the format string as an address.

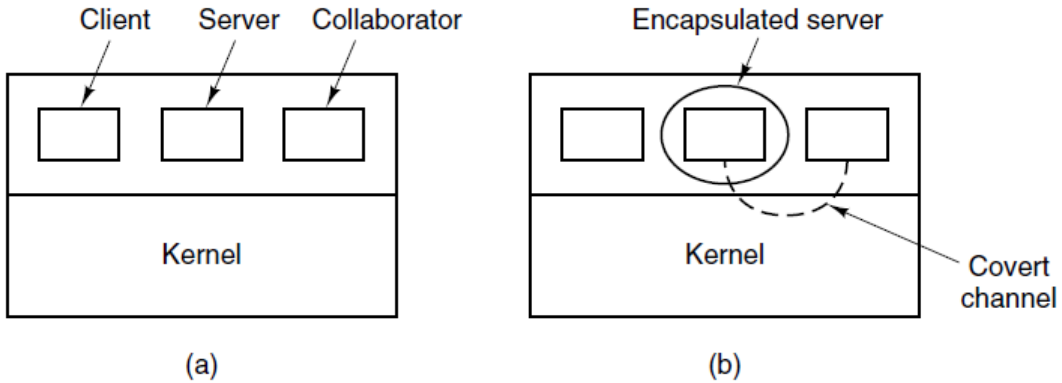


Command Injection Attacks

- Code that might lead to
  a command injection attack.

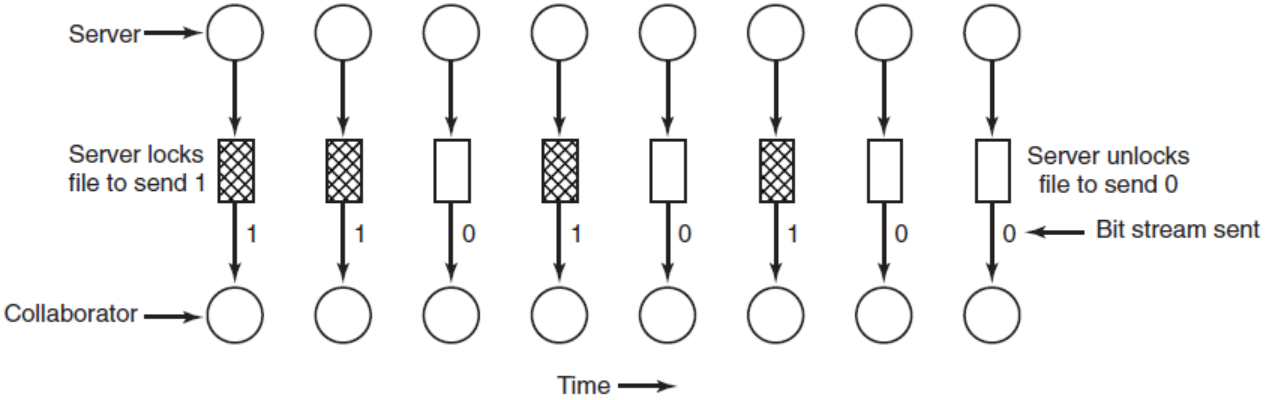
```
int main(int argc, char *argv[])
{
  char src[100], dst[100], cmd[205] = "cp ";          /* declare 3 strings */
  printf("Please enter name of source file: ");       /* ask for source file */
  gets(src);                                          /* get input from the keyboard */
  strcat(cmd, src);                                   /* concatenate src after cp */
  strcat(cmd, " ");                                   /* add a space to the end of cmd */
  printf("Please enter name of destination file: ");  /* ask for output file name */
  gets(dst);                                          /* get input from the keyboard */
  strcat(cmd, dst);                                   /* complete the commands string */
  system(cmd);                                        /* execute the cp command */
}
```

Covert Channels (1)



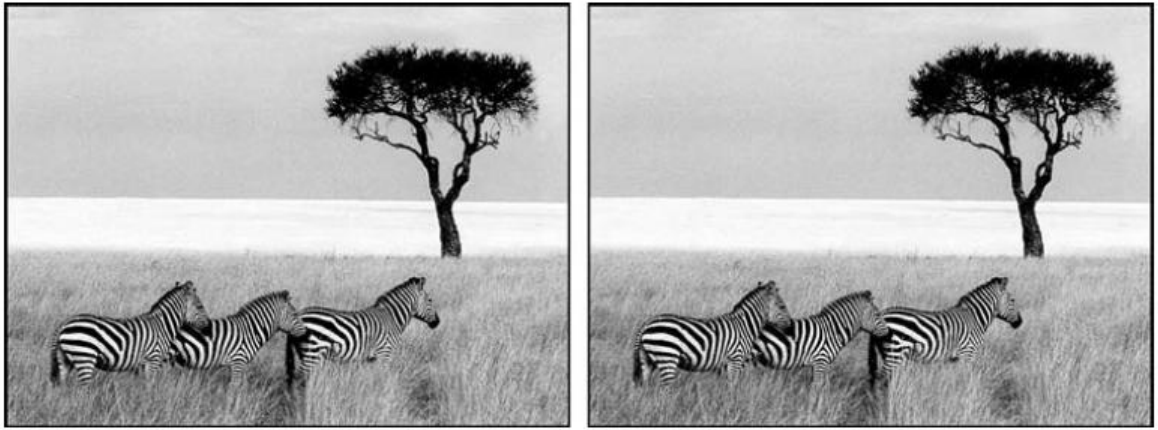
(a) The client, server, and collaborator processes.
(b) The encapsulated server can still leak to the collaborator via covert channels.

Covert Channels (2)



Example: A covert channel using file locking.

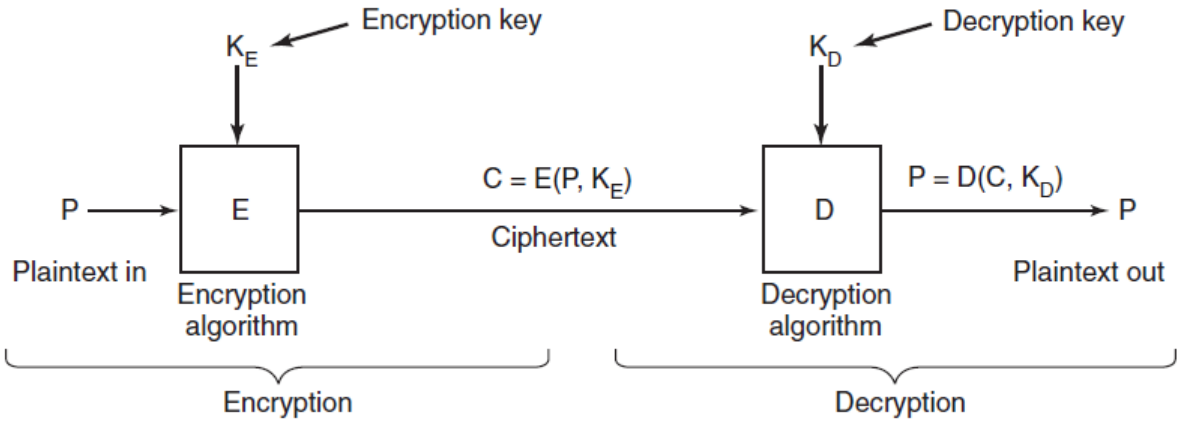
Steganography



(a) (b)

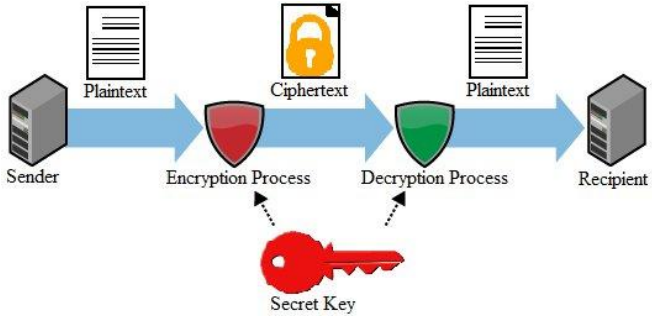
(a) Three zebras and a tree.
(b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

Basics of Cryptography

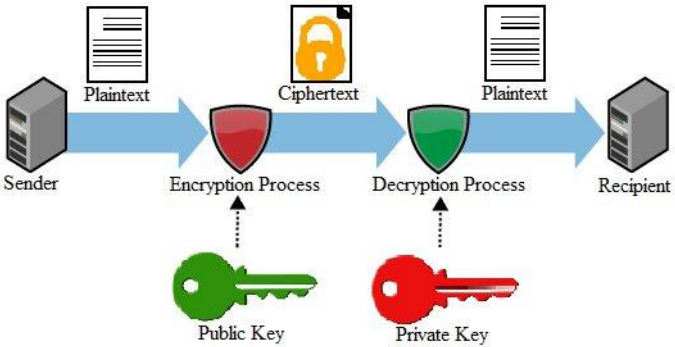


Relationship between the plaintext and the ciphertext.

Types of Encryption Keys



Symmetric Key
(e.g. DES [Data Encryption Standard])



Asymmetric Key
(e.g. RSA [Rivest, Shamir, Adleman])

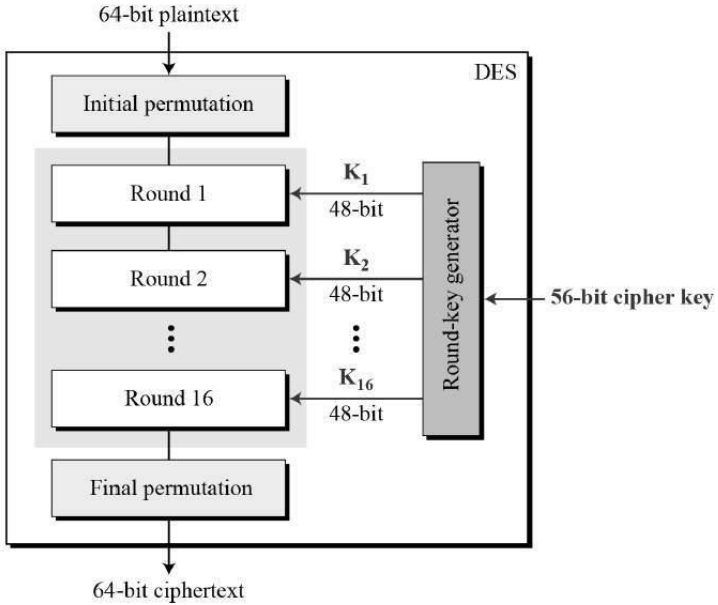
Secret-Key Cryptography

plaintext:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ciphertext:	Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

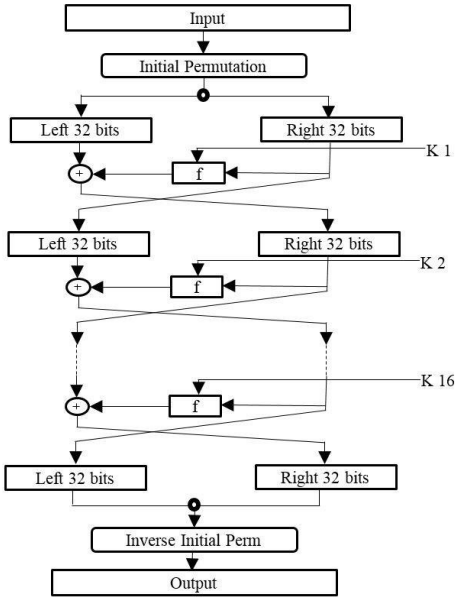
An encryption algorithm in which each letter is replaced by a different letter.

- Above is a symmetric key as, the function is reversible
- This particular function is "trivial"

DES (Data Encryption Standard)



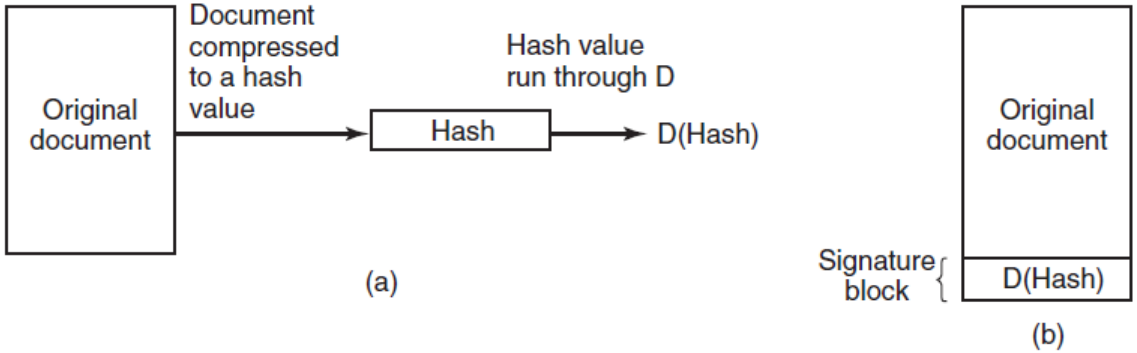
General idea



Detailed Execution

Decryption is the inverse operation (bottom up)

Digital Signatures



(a) Computing a signature block.
(b) What the receiver gets.

Authentication (1)

Methods of authenticating users when they attempt to log in based on one of three general principles:

1. Something the user knows.

2. Something the user has.

3. Something the user is.

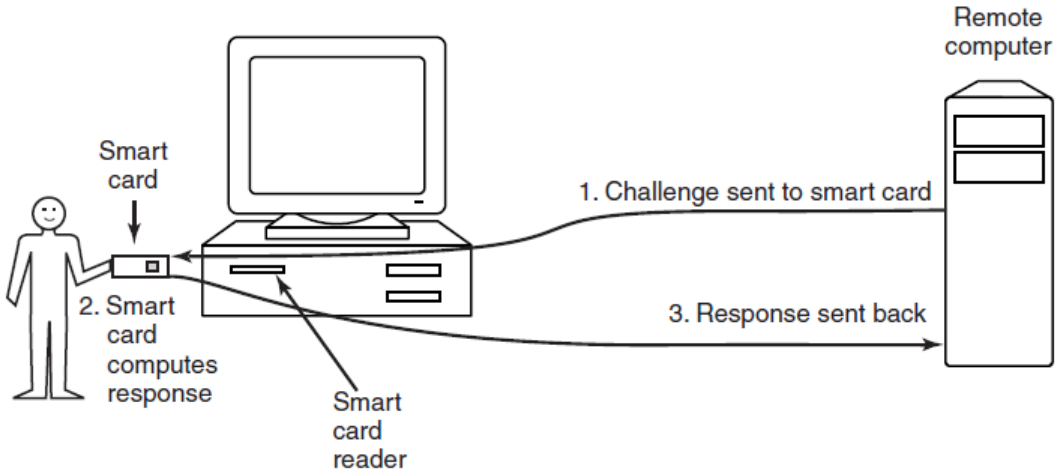
Challenge-Response Authentication

Questions should be chosen so that the user does not need to write them down.

Examples:

1. Who is Madalina’s sister?
2. On what street was your elementary school?
3. What did Hubertus Franke teach?
4. Two factor authentication

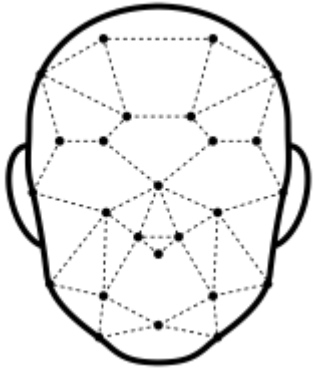
Authentication Using a Physical Object



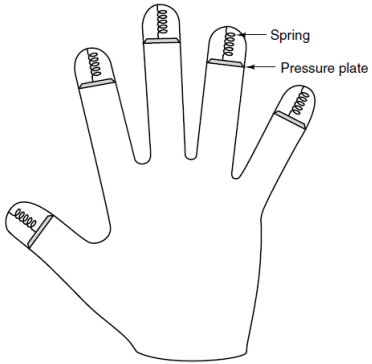
Use of a smart card for authentication.

Authentication Using Biometrics

Fingerprint Reader



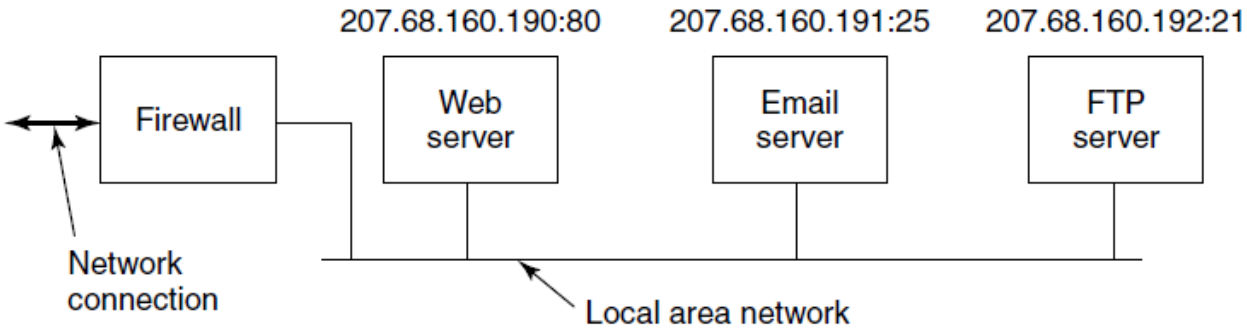
Face Recognition



A device for measuring finger length.

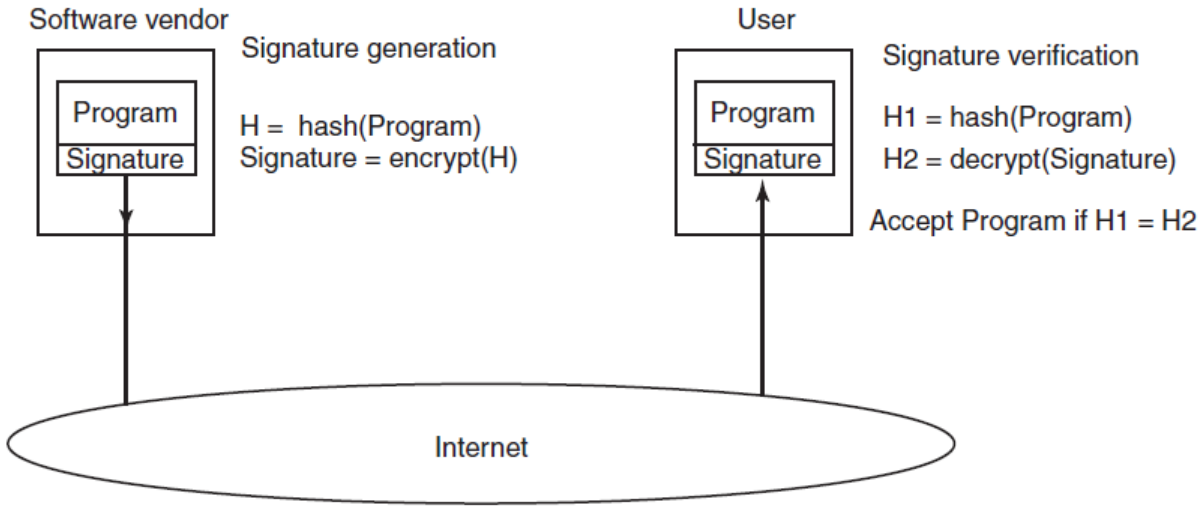
Iris Scan

Firewalls



A simplified view of a hardware firewall protecting a LAN with three computers

Code Signing



How code signing works.