Operating Systems: Protection and Security

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 16 and 17)

Protection - Goal of OS

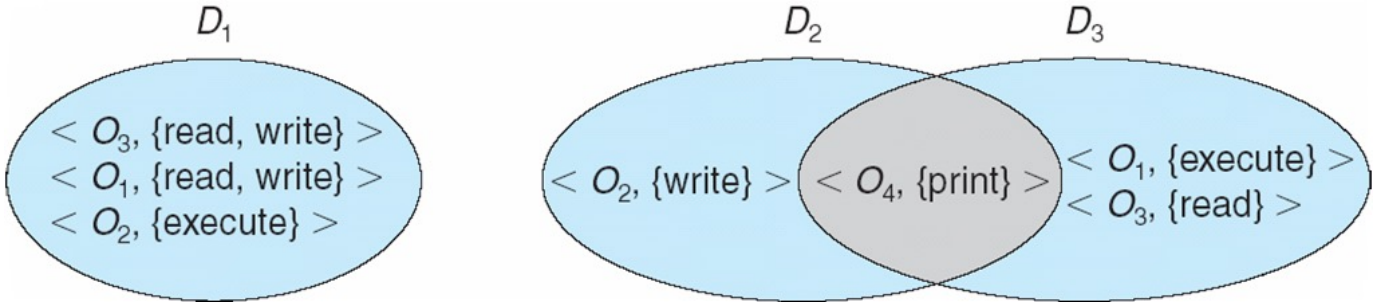
- Prevent malicious misuse of the system.
- Ensure that each shared resource is used only in accordance with system **policies**
- Ensure that errant programs cause the minimal amount of damage possible

Protection is achieved by having a mechanism in place for **controlling the access** of programs, processes, or users **to the resources** defined by a computer system.

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks

Domain of Protection

- A computer system is a collection of processes and object: Hardware objects (e.g.: CPU, memory, printers, disks) and Software objects (such as files).
- Processes operates within a **(protection) domain**, which specifies the resources (objects) a process may access.
- Each domain is defined by a **set of objects and the types of operations** that may be invoked on each object, and is represented as a set of pairs of ***<object-name*, access *rights-set*>**, where *access rights-set* is a subset of all valid operations that can be performed on the object
  - E.g.: <file *F*, {read, write}>



Domain of Protection cont…

- The association between a process and a domain may be static or dynamic.
  - **Association is static** – The set of resources (domain) available to the process is fixed throughout the process's lifetime
    - A mechanism must be available to change the content of a domain.
  - **Association is dynamic** – The set of resources (domain) available to the process changes throughout the process's lifetime
    - **Domain switching** (switch from one domain to another).
- Domains may be realized in different ways
  - As users (**UNIX associates domains with users**), or
  - As processes/procedures

Access Matrix

- The model of protection previously discussed can be viewed as a matrix called **access matrix**, where
  - Rows represent Domains
  - Columns represent Objects
  - **`access(i, j)-`** is the set of operations that a process executing in domain D_i can invoke on Object O_j

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix – Domain Switching

- Domain switching can be supported under this model by:
  - including *domains* among the *objects* of the access matrix, and
  - providing **`switch`** access right to other domains.
- Switching from D_i to D_j ⇔ **`switch ∈ access(i, j)`**
- A process executing in domain D_2 can switch to domains D_3 or D_4.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix Cont…

- To **allow controlled changes** to the access matrix, three additional operations are needed:
  - **Copy rights**
  - **Owner rights**
  - **Control rights**
- Copy and owner rights only allow the modification of rights within a column.
  - We shall assume that copy and owner rights are applicable to non–domain objects.
- Control rights changes the entries in a row.
  - Control right is applicable to only domain objects.

Access Matrix with *Copy* Rights

- **Copy Rights:** Denoted by *. Processes in that domain have the right to copy that access right denoted by * within the same column (same object).

- For example: process executing in domain D_2 can copy the read operation into any entry associated with file F_2

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With Owner Rights

- **Owner rights:** Denoted by **owner,** enables adding new rights or removing existing ones in a column or to that object.
- For example, domain $\mathbf{D_1}$ is the owner of $\mathbf{F_1}$ and thus can add and delete any valid right in column $\mathbf{F_1}$

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix with Control rights

- **Control rights:** Denoted by control, only apply to domain objects.
- Allows a process operating in one domain to affect the rights available in other domains.
  - If `access(i,j)` includes the `control` right, then a process executing in domain D_i can **remove** any access right from row j
- For example in the table below, a process operating in domain $\mathbf{D_2}$ has the right to **remove** any of the rights in domain $\mathbf{D_4}$.

Access Matrix with Control rights

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

- The access matrix is generally, a **sparse matrix** (less entries)
- It can be implemented in many ways:
  - Global Table
  - Access Lists for Objects
  - Capability Lists for Domains
  - Lock–Key Mechanism
- Each method has its own advantages and disadvantages.
  - Many systems employ a combination of the above listed methods.

Implementation of Access Matrix (Cont.)

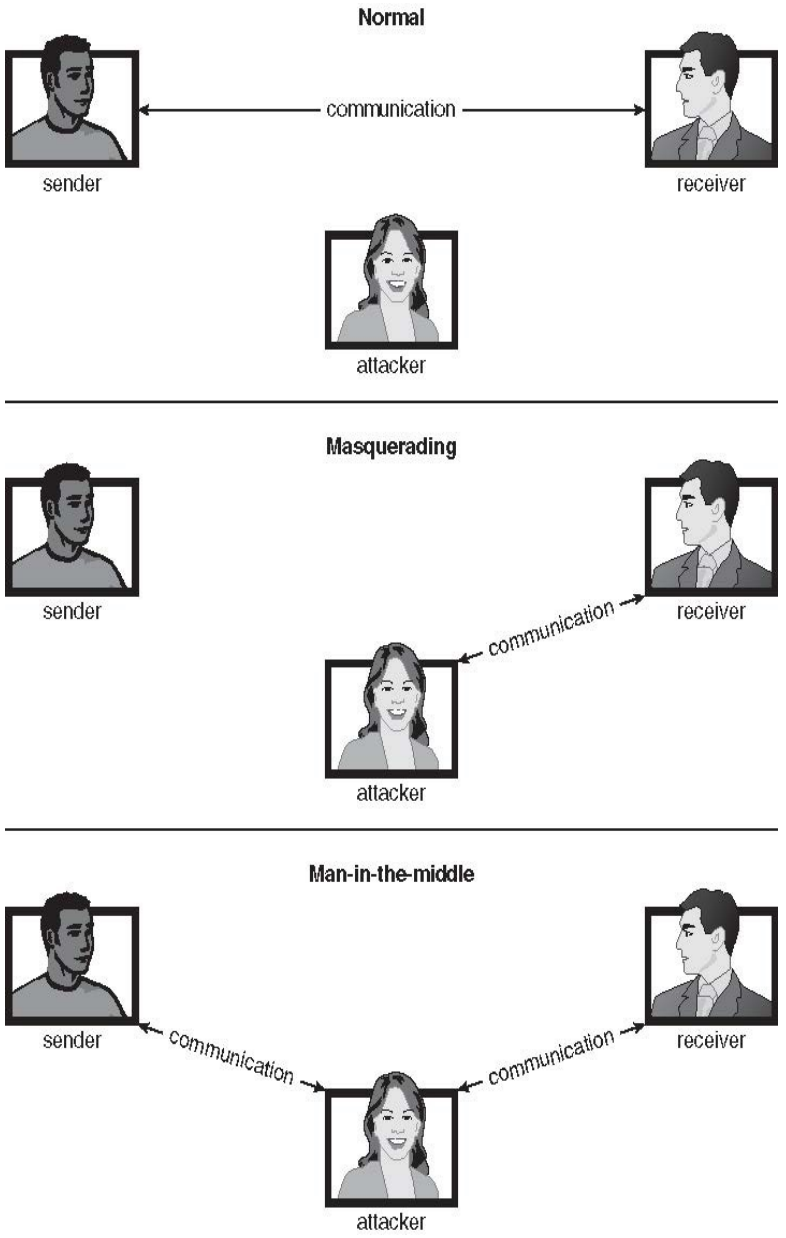
- **Global Table** consists of ordered triples <domain, object, rights-set>.
- **Access lists for objects**
  - Each column implemented as an access list for one object
  - Resulting in per-object list consisting of ordered pairs **`<domain, rights-set>`** defining all domains with non-empty set of access rights for the object
- **Capability list for domains**
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects (called a **capability**) together with operations allows on them

Security

- Protection is providing controlled access to programs and data stored in a computer system.
  - Protection deals with internal threats
- **Security**, on the other hand, requires not only an adequate protection system but also consideration of the external environment within which the system operates.
  - Security deals with external threats/intruders
- **Intruders** attempt to breach security
- **Threat** is potential security violation
- **Attack** is an attempt to breach security
  - Attack can be accidental (easier to protect from) or malicious.

Standard Security Attacks

- **Masquerading**
  - Pretending to be an authorized user to escalate privileges
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - In a network communication, a man-in-the-middle attack may be preceded by a session hijacking, in which an active communication session is intercepted.



Threats

Many variations, many names. Broadly classified as program, system and network threats.

- Trojan Horse
- Spyware
- Trap Door
- Logic Bomb
- Stack and Buffer Overflow
- Viruses
- Worm
- Port scanning
- Denial of Service

Cryptography

- Cryptography means to constrain potential senders (*sources*) and/or receivers (*destinations*) of *messages*
  - Based on secrets called **keys** used to process messages.
- Cryptography helps with the following two major scenarios:
  - **Encryption** - Enables a sender to send a message to the intended receiver.
    - This is achieved by encoding the message, such that it can only be understood (decrypted) by the receiver.
  - **Authentication** - Enables a recipient of a message to verify sender.
    - It is also used to check if a message has been modified

Encryption

- *Encryption constrains the set of possible receivers of a message*
- **Encryption** is the process of encoding messages (called **ciphertexts**) using keys.
- **Decryption** is the process of decoding messages using keys.
- An algorithm used for encryption must provide the following essential property:
  - Given a ciphertext, a computer can compute the message only if possesses the key
  - Given a ciphertext, it is impossible to derive the key from it.

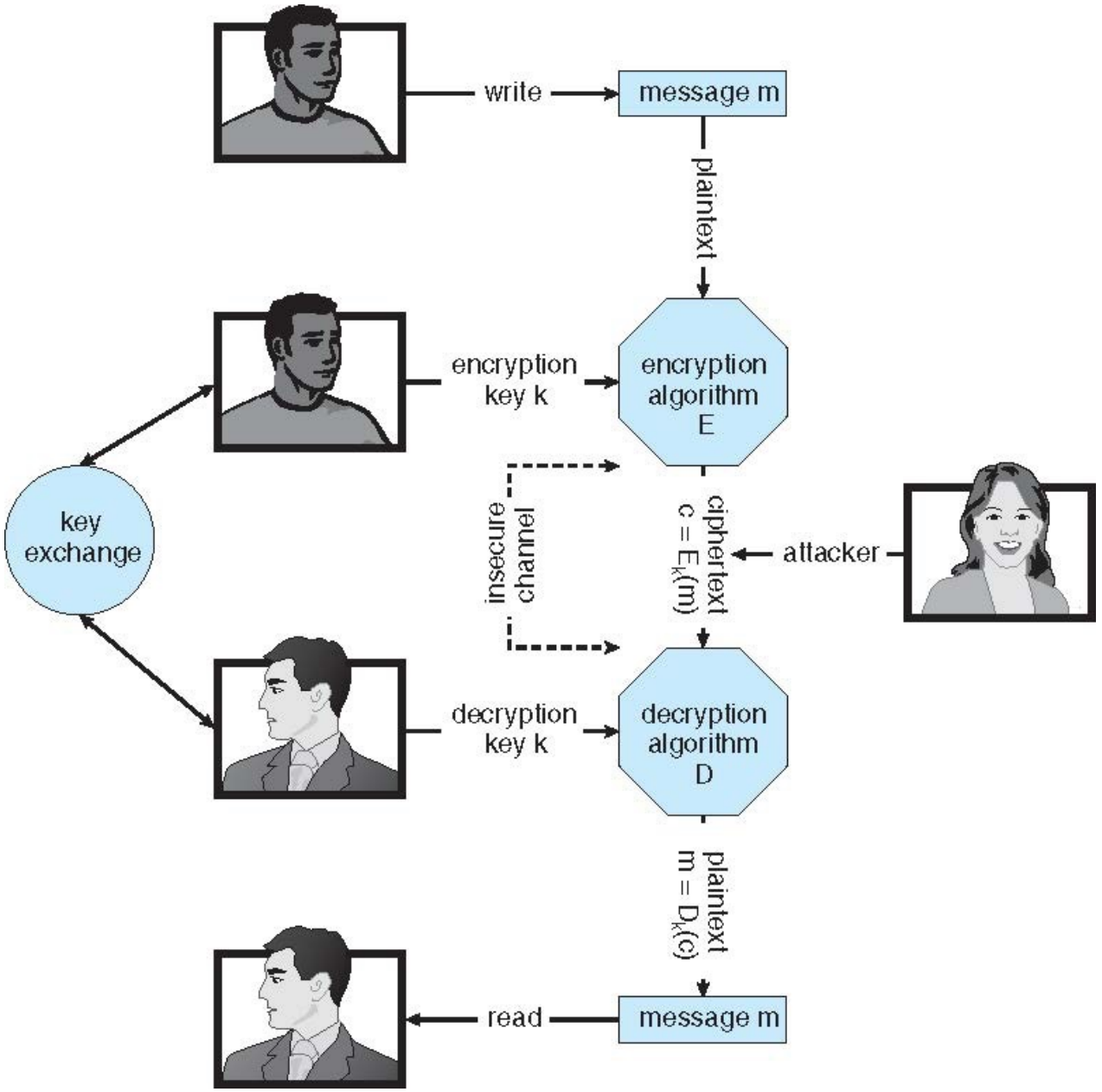
Encryption Algorithms

- There are two main types of encryption algorithms:
  - Symmetric
  - Asymmetric

Symmetric Encryption

- **Same key** used to encrypt and decrypt messages
  - Therefore, *key* must be kept secret and safely guarded.
- Examples of symmetric encryption algorithms are:
  - Data-Encryption Standard (DES)
  - Triple-DES
  - Advanced Encryption Standard (AES)

Secure Communication over Insecure Medium



Asymmetric Encryption

- **Asymmetric encryption** is based on having two different keys to encrypt and decrypt messages.
  - **public key** – is used to **encrypt data** and is published.
  - **private key** – is used to **decrypt data** and is private; that is, key known only to individual decrypting message
- **RSA Algorithm** is one of the most widely used asymmetric encryption algorithms.
- However RSA is computationally intensive.
  - Therefore used primarily to encrypt and decrypt small sized data. For example, keys.

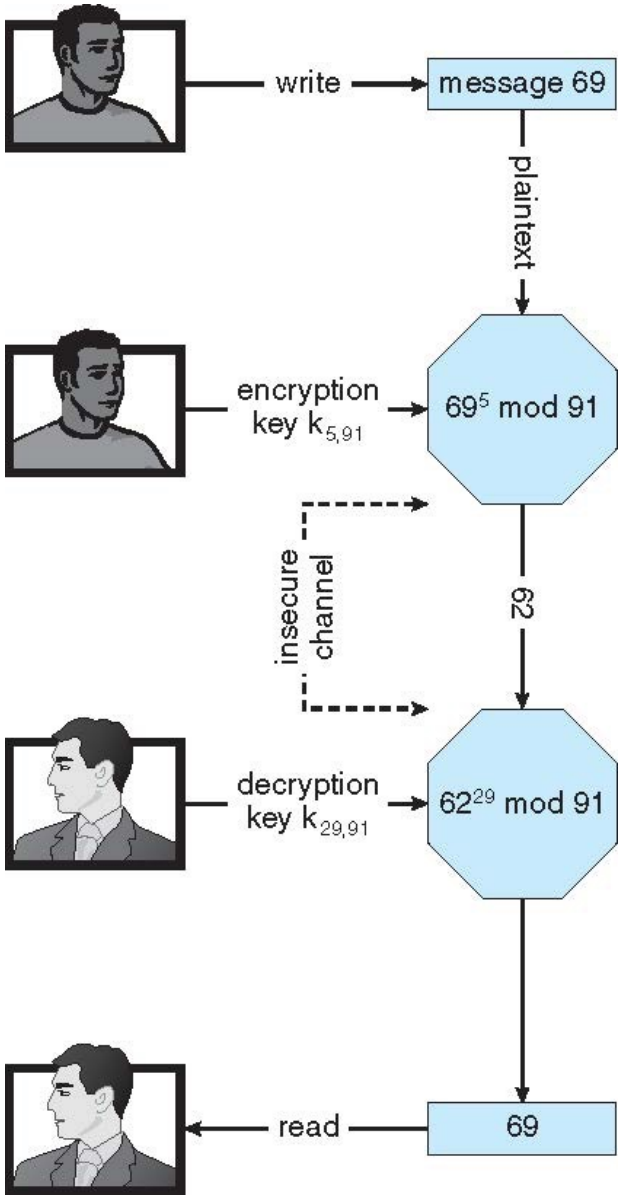
RSA Algorithm

- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely distributed
  - $K_{e,N} = (k_e, N)$ is the **public key**
  - $K_{d,N} = (k_d, N)$ is the **private key**
  - $N = p * q$, *where p, q are* two large, randomly chosen prime (for example 512 bits long)
  - K_e satisfies the condition that it is relatively prime to $(p–1)(q–1)$ and< $(p–1)(q–1)$
  - k_d satisfies $k_e k_d \bmod (p–1)(q–1) = 1$
  - **Encryption algorithm** is $E_{ke,N}(m) = m^{ke} \bmod N$, where
  - **Decryption algorithm** is then $D_{kd,N}(c) = c^{kd} \bmod N$

RSA Algorithm Example

- For example, let $p = 7$ and $q = 13$
- We then calculate $N = 7*13 = 91$ and $(p–1)(q–1) = 72$
- We next select k_e relatively prime to 72 and < 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29
- We how have our keys
  - Public key, $k_{e,N} = (5, 91)$
  - Private key, $k_{d,N} = (29, 91)$
- Encrypting the message (m) 69 with the public key results in the ciphertext (C) = $E_{ke,N}(m) = 69^5 \bmod 91 = 62$
- The ciphertext C is decrypted using the decryption algorithm = $D_{kd,N}(C) = 62^{29} \bmod 91 = 69 = m$

Encryption using RSA Asymmetric Cryptography



Key Distribution

- Delivery of symmetric key is huge challenge
  - Sometimes done **out-of-band** (e.g.: on paper or in a conversation)
  - Even asymmetric key distribution needs care – man-in-the-middle attack (next slide)
    - To solve this, use a **digital certificate** which is a public key digitally signed by a trusted party.

Man-in-the-middle Attack on Asymmetric Cryptography

- The person who wants to receive an encrypted message sends out his public key,
- However, the attacker also sends her "bad" public key (which matches her private key).
- The person who wants to send the encrypted message knows no better and so uses the bad key to encrypt the message.
- The attacker then happily decrypts it!