Q.1	Explain the Goals of Protection.
Ans	**Protection- Goals of protection:-** • As computer systems have become more sophisticated and pervasive in their applications, the need to protect their integrity has also grown. • Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, • such as a directory of files, or share a common physical name space, such as memory. • Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources. • Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. • Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem. • Also, an unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. • A protection-oriented system provides means to distinguish between authorized and unauthorized usage. • The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. These policies can be established in a variety of ways. • Some are fixed in the design of the system, while others are formulated by the management of a system. Still others are defined by the individual users to protect their own files and programs. • A protection system must have the flexibility to enforce a variety of policies. • Policies for resource use may vary by application, and they may change over time. • For these reasons, protection is no longer the concern solely of the designer of an operating system. The application programmer needs to use protection mechanisms as well, to guard resources created and supported by an application subsystem against misuse. In this chapter, we describe the protection mechanisms the operating system should provide, but application designers can use them as well in designing their own protection software.
Q2.	Explain Monitors?
Ans	**Domain of Protection** A computer can be viewed as a collection of processes and objects (both HW & SW). The need to know principle states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access. The modes available for a particular object may depend upon its type. **Domain Structure** A protection domain specifies the resources that a process may access.

Each domain defines a set of objects and the types of operations that may be invoked on each object.
An access right is the ability to execute an operation on an object.
A domain is defined as a set of < object, { access right set } > pairs, as shown below. Note that some domains may be disjoint while others overlap.

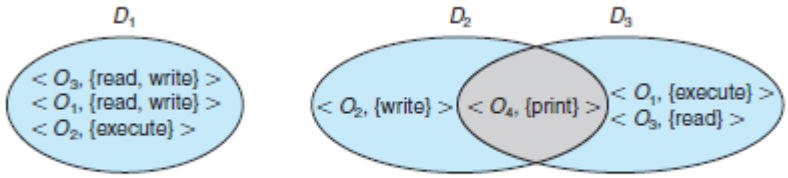


Figure 14.1 System with three protection domains.

The association between a process and a domain may be static or dynamic.
If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.

If the association is dynamic, then there needs to be a mechanism for domain switching. Domains may be realized in different fashions - as users, or as processes, or as procedures. E.g. if each user corresponds to a domain, then that domain defines the access of that user, and changing domains involves changing user ID.

An Example: UNIX

UNIX associates domains with users.
Certain programs operate with the SUID bit set, which effectively changes the user ID, and therefore the access domain, while the program is running. (and similarly for the SGID bit.) Unfortunately this has some potential for abuse.
An alternative used on some systems is to place privileged programs in special directories, so that they attain the identity of the directory owner when they run. This prevents crackers from placing SUID programs in random directories around the system. Yet another alternative is to not allow the changing of ID at all. Instead, special privileged daemons are launched at boot time, and user processes send messages to these daemons when they need special tasks performed.

An Example: MULTICS

The MULTICS system uses a complex system of rings, each corresponding to a different protection domain, as shown below:

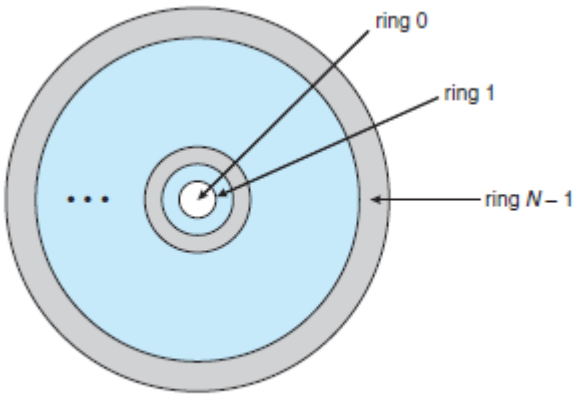


Figure 14.2 MULTICS ring structure.

Rings are numbered from 0 to 7, with outer rings having a subset of the privileges of the inner rings.

Each file is a memory segment, and each segment description includes an entry that indicates the ring number associated with that segment, as well as read, write, and execute privileges.

Each process runs in a ring, according to the current-ring-number, a counter associated with each process.

A process operating in one ring can only access segments associated with higher (farther out) rings, and then only according to the access bits. Processes cannot access segments associated with lower rings.

Domain switching is achieved by a process in one ring calling upon a process operating in a lower ring, which is controlled by several factors stored with each segment descriptor:

- An access bracket, defined by integers b1 <= b2.
- A limit b3 > b2
- A list of gates, identifying the entry points at which the segments may be called.

If a process operating in ring i calls a segment whose bracket is such that b1 <= i <= b2, then the call succeeds and the process remains in ring i.
Otherwise a trap to the OS occurs, and is handled as follows:

- If i < b1, then the call is allowed, because we are transferring to a procedure with fewer privileges. However if any of the parameters being passed are of segments below b1, then they must be copied to an area accessible by the called procedure.\
- If i > b2, then the call is allowed only if i <= b3 and the call is directed to one of the entries on the list of gates.

Overall this approach is more complex and less efficient than other protection schemes

Q.3 Explain Access Matrix

Ans.

Access matrix:-

- Our model of protection can be viewed abstractly as a matrix, called an access matrix. The rows of the access matrix represent domains, and the columns represent objects. Each entry in the matrix consists of a set of access rights.
- Because objects are defined explicitly by the column, we can omit the object name from the access right. The entry access (i, j) defines the set of operations that a process, executing in domain Di, can invoke on object Oj.
- We consider the access matrix shown in Figure. There are four domains and four objects: three files (F1, F2, F3), and one laser printer. When a process executes in domain D1, it can read files F1 and F3.. A process executing in domain D4has the same privileges as it does in domain D1, but in addition, it can also write onto files F1 and F3. Note that the laser printer can be accessed only by a process executing in domain D2.
- The access-matrix scheme provides us with the mechanism for specifying a variety of policies. More specifically, we must ensure that a process executing in domain Di can access only those objects specified in row i, and then only as allowed by the access matrix entries.

Object Domain	F1	F2	F3	printer
D1	Read		Read	
D2				Print
D3		read	Execute	
D4	Read write		Read write	

Fig Access Matrix

- Policy decisions concerning protection can be implemented by the access matrix. The users normally decide the contents of the access-matrix entries.
- Allowing controlled change to the contents of the accessmatrix entries requires three additional operations: copy, owner, and control.

Q.4 Explain Implementation of Access Matrix-
1.Global Table
2.Access list for objects

Implementation of access matrix:-

- Global Table
- Access List for Objects
- Capability List for domains
- A lock-Key Mechanism

Global Table :-

The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <domain, object, rights-set>. Whenever an operation *M* is executed on an object *Oj* within domain *Di* , the global table is searched for a triple <*Di* , *Oj* , *Rk*>, with *M* ∈ *Rk* . If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.

This implementation suffers from several drawbacks. The table is usually large and thus cannot be kept in main memory, so additional I/O is needed. Virtual memory techniques are often used for managing this table. In addition, it is difficult to take advantage of special groupings of objects or domains. For example, if everyone can read a particular object, this object must have a separate entry in every domain.

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	

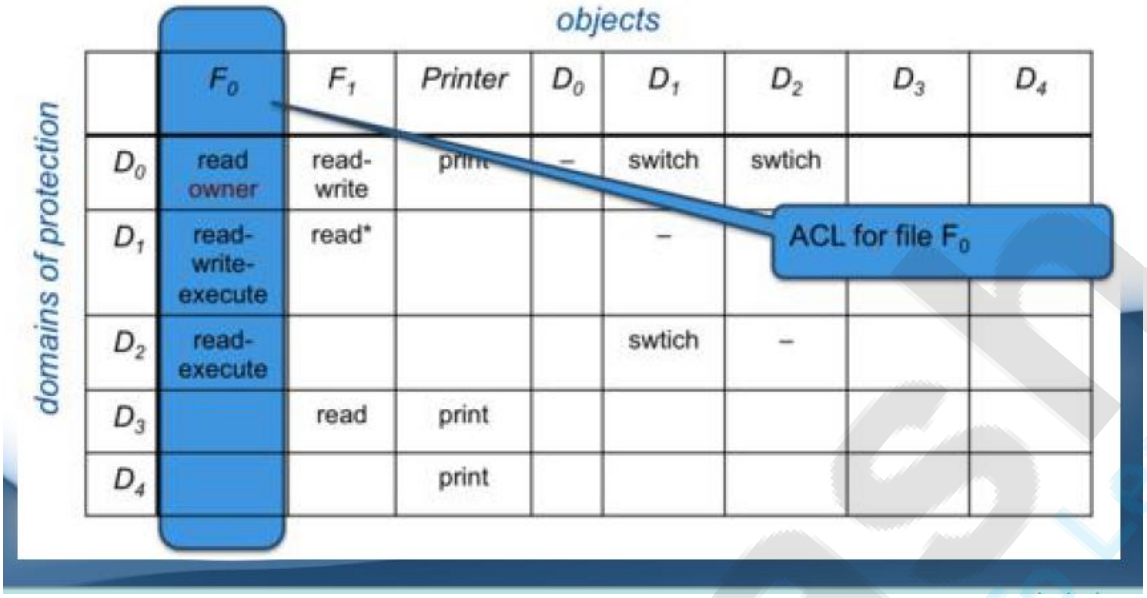
Global Table

<D1, F1, read>, <D1, F3, read>, <D2, laser printer, print>, <D3, F2, read>, <D3, F3, execute>, <D4, F1, read>, <D4, F1, write>, <D4, read, F3>, <D4, write, F3>

Access lists for objects:-

Each column in the access matrix can be implemented as an access list for one object, as described in Section 11.6.2. Obviously, the empty entries can be discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object.

This approach can be extended easily to define a list plus a ***default*** set of access rights. When an operation *M* on an object O_j is attempted in domain D_i , we search the access list for object O_j , looking for an entry <D_i , R_k> with $M \in R_k$. If the entry is found, we allow the operation; if it is not, we check the default set. If *M* is in the default set, we allow the access. Otherwise, access is denied, and an exception condition occurs. For efficiency, we may check the default set first and then search the access list.

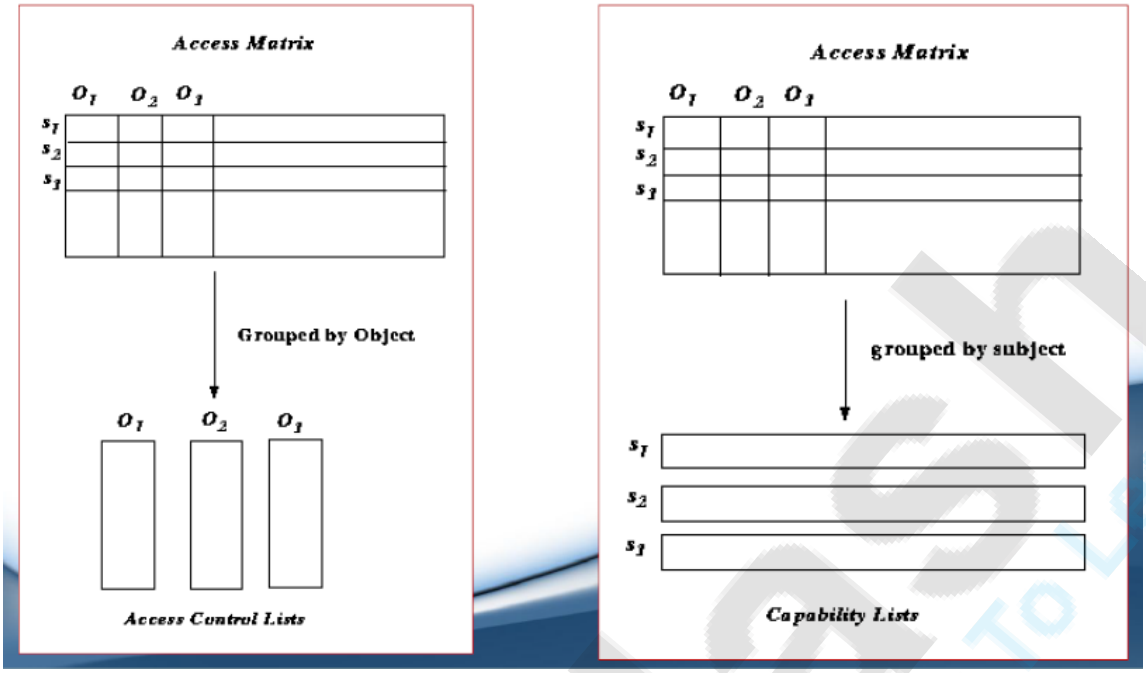


Q.5 Explain Implementation of Access Matrix-
1.Capability list for domain
2.Lock Key Scheme

Ans.

Capability Lists for Domains :-

A **capability list** for a domain is a list of objects together with the operations allowed on those objects. An object is often represented by its physical name or address, called a **capability**. To execute operation *M* on object O_j , the process executes the operation*M*, specifying the capability (or pointer) for object O_j as a parameter. Simple **possession** of the capability means that access is allowed.
The capability list is associated with a domain, but it is never directly accessible to a process executing in that domain. Rather, the capability list is itself a protected object, maintained by the operating system and accessed by the user only indirectly. Capability-based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process (where they could be modified). If all capabilities are secure, the object they protect is also secure against unauthorized access.
Capabilities were originally proposed as a kind of secure pointer, to meet the need for resource protection that was foreseen as multiprogrammed computer systems came of age. The idea of an inherently protected pointer provides a foundation for protection that can be extended up to the application level.



Lock-key scheme:-

The **lock–key scheme** is a compromise between access lists and capability lists. Each object has a list of unique bit patterns, called **locks**. Similarly, each domain has a list of unique bit patterns, called **keys**. A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.
As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain. Users are not allowed to examine or modify the list of keys (or locks) directly

Q. 6 Explain the Revocation of access rights.

Ans

Revocation of access rights:-

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users. Various questions about revocation may arise:

- **Immediate versus delayed**. Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?
- **Selective versus general**. When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?
- **Partial versus total**. Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- **Temporary versus permanent**. Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

With an access-list scheme, revocation is easy. The access list is searched for

	any access rights to be revoked, and they are deleted from the list. Revocation is immediate and can be general or selective, total or partial, and permanent or temporary. Capabilities, however, present a much more difficult revocation problem, as mentioned earlier. Since the capabilities are distributed throughout the system,wemust find them beforewe can revoke them. Schemes that implement revocation for capabilities include the following: • **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability. • **Back-pointers**. A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessary. This scheme was adopted in the MULTICS system. It is quite general, but its implementation is costly. • **Indirection**. The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object.We implement revocation by searching the global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to point to an illegal table entry. Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object. The object for a capability and its table entry must match. This scheme was adopted in the CAL system. It does not allow selective revocation. • **Keys**. A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability. A **master key** is associated with each object; it can be defined or replaced with the set-key operation. When a capability is created, the current value of the master key is associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised. Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object. This scheme does not allow selective revocation, since only one master key is associated with each object. If we associate a list of keys with each object, then selective revocation can be implemented. Finally,we can group all keys into one global table of keys. A capability is valid only if its key matches some key in the global table. We implement revocation by removing the matching key from the table.With this scheme, a key can be associated with several objects, and several keys can be associated with each object, providing maximum flexibility. In key-based schemes, the operations of defining keys, inserting them into lists, and deleting them from lists should not be available to all users. In particular, it would be reasonable to allow only the owner of an object to set the keys for that object. This choice, however, is a policy decision that the protection system can implement but should not define.
Q.7	Explain the Security Problem
	Security:- **The Security problem**

The operating system can allow users to protect their resources. We say that a system is secure if its resources art used and accessed as intended under all circumstances. Unfortunately, it is not generally possible to achieve total security. Security violations of the system can be categorized as being either intentional (malicious) or accidental. Among the forms of malicious access are the following:

- Unauthorized reading of data (theft of information).
- Unauthorized modification of data.
- Unauthorized destruction of data.

In addition, a **threat** is the potential for a security violation, such as the discovery of a vulnerability, whereas an **attack** is the attempt to break security.

Breach of confidentiality. This type of violation involves unauthorized reading of data (or theft of information). Typically, a breach of confidentiality is the goal of an intruder. Capturing secret data from a system or a data stream, such as credit-card information or identity information for identity theft, can result directly in money for the intruder.

- **Breach of integrity**. This violation involves unauthorized modification of data. Such attacks can, for example, result in passing of liability to an innocent party or modification of the source code of an important commercial application.
- **Breach of availability**. This violation involves unauthorized destruction of data. Some crackers would rather wreak havoc and gain status or bragging rights than gain financially. Website defacement is a common example of this type of security breach.
- **Theft of service**. This violation involves unauthorized use of resources. For example, an intruder (or intrusion program) may install a daemon on a system that acts as a file server.
- **Denial of service**. This violation involves preventing legitimate use of the system. **Denial-of-service (DOS)** attacks are sometimes accidental. The original Internet worm turned into a DOS attack when a bug failed to delay its rapid spread

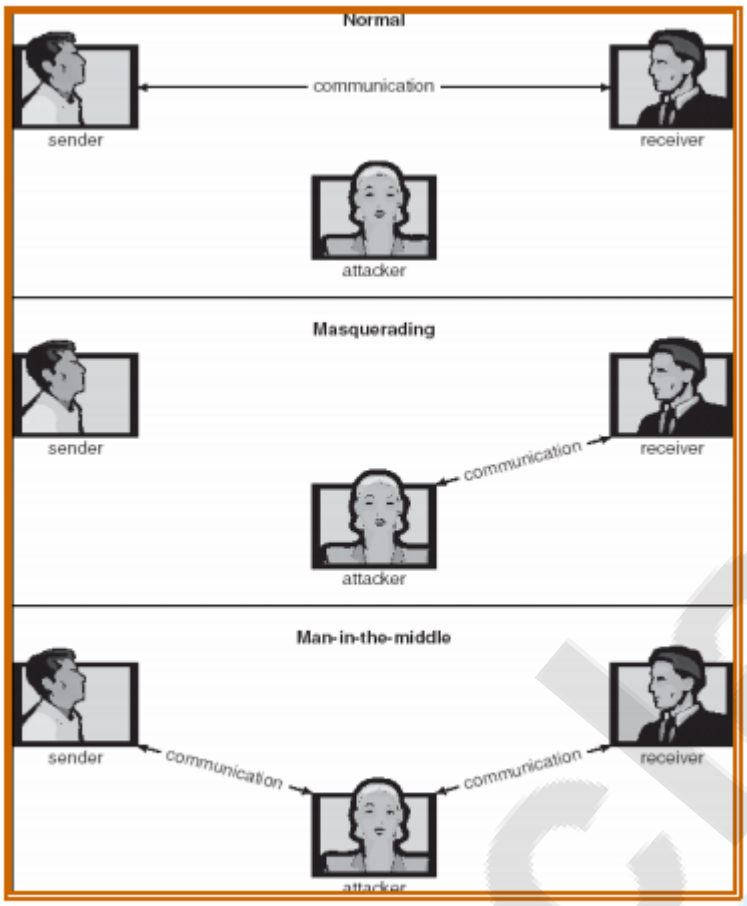


Fig 14.1 Standard Security Attack.

To protect a system, we must take security measures at four levels:

1. **Physical**. The site or sites containing the computer systems must be physically secured against armed or surreptitious entry by intruders. Both the machine rooms and the terminals or workstations that have access to the machines must be secured.

2. **Human**. Authorization must be done carefully to assure that only appropriate users have access to the system. Even authorized users, however,may be "encouraged" to let others use their access (in exchange for a bribe, for example). They may also be tricked into allowing access via **social engineering**. One type of social-engineering attack is **phishing**. Here, a legitimate-looking e-mail or web page misleads a user into entering confidential information. Another technique is **dumpster diving**, a general term for attempting to gather information in order to gain unauthorized access to the computer (by looking through trash, finding phone books, or finding notes containing passwords, for example). These security problems are management and personnel issues, not problems pertaining to operating systems.

3. **Operating system**. The system must protect itself from accidental or purposeful security breaches. A runaway process could constitute an accidental denial-of-service attack.Aquery to a service could reveal passwords. A stack overflow could allow the launching of an unauthorized

process. The list of possible breaches is almost endless.

4. **Network**. Much computer data in modern systems travels over private leased lines, shared lines like the Internet, wireless connections, or dial-up lines. Intercepting these data could be just as harmful as breaking into a computer, and interruption of communications could constitute a remote denial-of-service attack, diminishing users' use of and trust in the system

Q. 8 Explain Authentication

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc. We're going to discuss following topics in this chapter.

Authentication
One Time passwords
Program Threats
System Threats
Computer Security Classifications

Authentication:-

A major security problem for operating systems is the authentication problem. The protection system depends on an ability to identify the programs and processes that are executing. Authentication is based on one or more of three items: user possession (a key or card), user knowledge (a user identifier and password), and a user attribute (fingerprint) retina pattern, or signature).

- Constraining set of potential senders of a message
  - Complementary and sometimes redundant to encryption.
  - Also can prove message unmodified.
- An authentication algorithm consists of following components:
  - A set K of keys.
  - A set M of messages.
  - A set A of authenticators.
- A function S : K → (M→ A)
  - That is, for each k ∈ K, S(k) is a function for generating authenticators from messages.
  - Both S and S(k) for any k should be efficiently computable functions.
- A function V : K → (M× A→ {true, false}). That is, for each k ∈ K, V(k) is a function for verifying authenticators on messages.
  - Both V and V(k) for any k should be efficiently computable functions.
- For a message m, a computer can generate an authenticator a ∈ A such that V(k)(m, a) = true only if it possesses S(k).
- Thus, computer holding S(k) can generate authenticators on messages so that any other computer possessing V(k) can verify them.
- Computer not holding S(k) cannot generate authenticators on messages that can be verified using V(k).
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive S(k) from the authenticators.

Authentication – Hash Functions

- Basis of authentication.
- Creates small, fixed-size block of data (message digest, hash value) from m.
- Hash Function H must be collision resistant on m
  - Must be infeasible to find an m' ≠ m such that H(m) = H(m').
- If H(m) = H(m'), then m = m'
  - The message has not been modified.
- Common message-digest functions include MD5, which produces a 128-bit hash, and SHA-1, which outputs a 160- bit hash.

Authentication – MAC

- Symmetric encryption used in message-authentication code (MAC) authentication algorithm.
- Simple example:
  - MAC defines S(k)(m) = f (k, H(m)).
    - Where f is a function that is one-way on its first argument.
    - k cannot be derived from f (k, H(m)).
    - Because of the collision resistance in the hash function, reasonably assured no other message could create the same MAC.
    - A suitable verification algorithm is V(k)(m, a) ≡ (f (k,m) = a).
    - Note that k is needed to compute both S(k) and V(k), so anyone able to compute one can compute the other.

Authentication – Digital Signature

- Based on asymmetric keys and digital signature algorithm.
- Authenticators produced are digital signatures.
- In a digital-signature algorithm, computationally infeasible to derive S(ks) from V(kv)
  - V is a one-way function.
  - Thus, kv is the public key and ks is the private key.
- Consider the RSA digital-signature algorithm.
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Digital signature of message S(ks)(m) = H(m) ks mod N
  - The key ks again is a pair d, N, where N is the product of two large, randomly chosen prime numbers p and q.
  - Verification algorithm is V(kv)(m, a) ≡ (a kv mod N = H(m))
  - Where kv satisfies kvks mod (p − 1)(q − 1) = 1
- Why authentication if a subset of encryption?
  - Fewer computations (except for RSA digital signatures)

	• Authenticator usually shorter than message. • It Can be basis for non-repudiation.
Q. 9	Explain One time Password in Security
	One Time Password: One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways. **Random numbers** − Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen. **Secret key** − User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login. **Network password** − Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.
Q. 10	Explain Thread 1.Program Thread 2. System Thread
	Program Threats Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats. **Trojan Horse** − Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources. **Trap Door** − If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door. **Logic Bomb** − Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect. **Virus** − Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

	System Threats System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats. **Worm** − Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network. **Port Scanning** − Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system. **Denial of Service** − Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.
Q.11	
	•
Q.12	
Q.13	
Q.14	
Q.15	