LECTURE NOTES

ON

INFORMATION SECURITY

Dr. P L Srinivasa Murthy
Professor

Information Technology

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)

Dundigal – 500 043, Hyderabad

UNIT – I

Security attacks (interruption, interception, modification and fabrication), security services (confidentiality, authentication, integrity, non-repudiation, access control and availability) and mechanisms, a model for internetwork security, internet standards and rfcs, buffer overflow & format string vulnerabilities, tcp session hijacking, attacks, route table modification, udp hijacking, and man-in-the-middle attacks.

Introduction:

Network Security: It can be defined as "measures adopted to prevent the unauthorized use, misuse, modification or denial of use of knowledge, facts, data or capabilities". Three aspects of IS are:

- **Security Attack:**

  Any action that comprises the security of information
- **Security Mechanism:**

  A mechanism that is designed to detect, prevent, or recover from a security.
- **Security Service:**

  It is a processing or communication service that enhances the security of the data processing systems and information transfer. The services are intended to counter

Security Attacks

Security attacks can be classified in terms of Passive attacks and Active attacks as per X.800 and RFC 2828
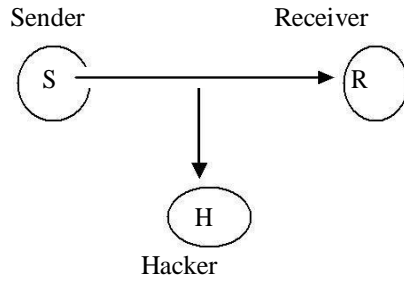
Different kinds of attacks are:

Interruption

An asset of the system is destroyed or becomes unavailable or unusable. It is an attack on availability

Examples:

i. Destruction of some hardware
ii. Jamming wireless signals
iii. Disabling file management systems

Interception

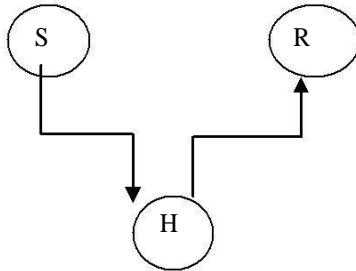An unauthorized party gains access to an asset. Attack on confidentiality.

Examples:
i. Wire tapping to capture data in a network.
ii. Illicitly copying data or programs
iii. Eavesdropping

Modification:
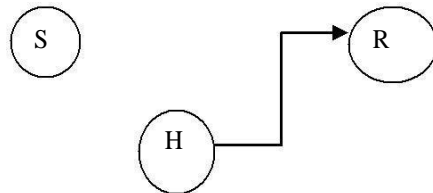

When an unauthorized party gains access and tampers an asset. Attack is on Integrity.

Examples:
i. Changing data file
ii. Altering a program and the contents of a message

Fabrication

An unauthorized party inserts a counterfeit object into the system. Attack on Authenticity. Also called impersonation

Examples:

i. Hackers gaining access to a personal email and sending message
ii. Insertion of records in data files
iii. Insertion of spurious messages in a network

Passive Attacks

A Passive attack attempts to learn or make use of information from the system, but does not affect system resources.

Two types:

Release of message content

It may be desirable to prevent the opponent from learning the contents (i.e sensitive or confidential info) of the transmission

Traffic analysis

A more subtle technique where the opponent could determine the location and identity of communicating hosts and could observe the frequency & length of encrypted messages being
exchanged there by guessing the nature of communication taking place.

Passive attacks are very difficult to detect because they do not involve any alternation of the data. As the communications take place in a very normal fashion, neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. So, the emphasis in dealing with passive attacks is on prevention rather than detection.

Active Attacks

Active attacks involve some modification of the data stream or creation of a false stream. An active attack attempts to alter system resources or affect their operation.

Four types:

- **Masquerade:** Here, an entity pretends to be some other entity. It usually includes one of the other forms of active attack.
- **Replay:** It involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.
- **Modification of messages:** It means that some portion of a legitimate message is altered, or that messages are delayed to produce an unauthorized effect.
  Ex: "John's acc no is 2346" is modified as "John's acc no is 7892"
- **Denial of service:** This attack prevents or inhibits the normal use or management of communication facilities.
  Ex: a: Disruption of entire network by disabling it
  b: Suppression of all messages to a particular destination by a third party.

Security Services:

It is a processing or communication service that is provided by a system to give a specific kind of production to system resources. Security services implement security policies and are implemented by security mechanisms.

Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. It is used to prevent the disclosure of information to unauthorized individuals or systems.".

The other aspect of confidentiality is the protection of traffic flow from analysis. **Ex:** A credit card number has to be secured during online transaction.

Authentication

This service assures that a communication is authentic. For a single message transmission, its function is to assure the recipient that the message is from intended source. Two specific authentication services defines in X.800 are

- **Peer entity authentication:** Verifies the identities of the peer entities involved in communication. Provides use at time of connection establishment and during data transmission. Provides confidence against a masquerade or a replay attack

- **Data origin authentication:** Assumes the authenticity of source of data unit, but does not provide protection against duplication or modification of data units. Supports applications like electronic mail, where no prior interactions take place between communicating entities.

Integrity

Integrity means that data cannot be modified without authorization two types of integrity services are available. They are

- **Connection-Oriented Integrity Service:** This service deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion, modification, reordering or replays.
- **Connectionless-Oriented Integrity Service:** It deals with individual messages regardless of larger context, providing protection against message modification

  only.

An integrity service can be applied with or without recovery. Because it is related to active attacks, major concern will be detection rather than prevention. If a violation is detected and the service reports it, either human intervention or automated recovery machines are required to recover.

Access Control

This refers to the ability to control the level of access that individuals or entities have to a network or system and how much information they can receive..

Availability

It is defined to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity.

According to X.800, the security mechanisms are divided into those implemented in a specific protocol layer and those that are not specific to any particular protocol layer or security service. X.800 also differentiates reversible & irreversible encipherment mechanisms. A reversible encipherment mechanism is simply an encryption algorithm that allows data to be encrypted and subsequently decrypted, where as irreversible encipherment include hash algorithms and message authentication codes used in digital signature and message authentication applications

Specific Security Mechanisms:

Incorporated into the appropriate protocol layer in order to provide some of the OSI security services,

- **Encipherment:** It refers to the process of applying mathematical algorithms for converting data into a form that is not intelligible. This depends on algorithm used and encryption keys.
- **Digital Signature:** The appended data or a cryptographic transformation applied to any data unit allowing to prove the source and integrity of the data unit and protect against forgery.
- **Access Control:** A variety of techniques used for enforcing access permissions to the system resources.

- **Data Integrity:** A variety of mechanisms used to assure the integrity of a data unit or stream of data units.
- **Authentication Exchange:** A mechanism intended to ensure the identity of an entity by means of information exchange
- **Traffic Padding:** The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.
- **Routing Control:** Enables selection of particular physically secure routes for certain data and allows routing changes once a breach of security is suspected.
- **Notarization:** The use of a trusted third party to assure certain properties of a data exchange

Pervasive Security Mechanisms:

These are not specific to any particular OSI security service or protocol layer.

- **Trusted Functionality:** That which is perceived to b correct with respect to some criteria
- **Security Level:** The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.
- **Event Detection:** It is the process of detecting all the events related to network security.
- **Security Audit Trail:** Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
- **Security Recovery:** It deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

A Model Of Inter Network Security
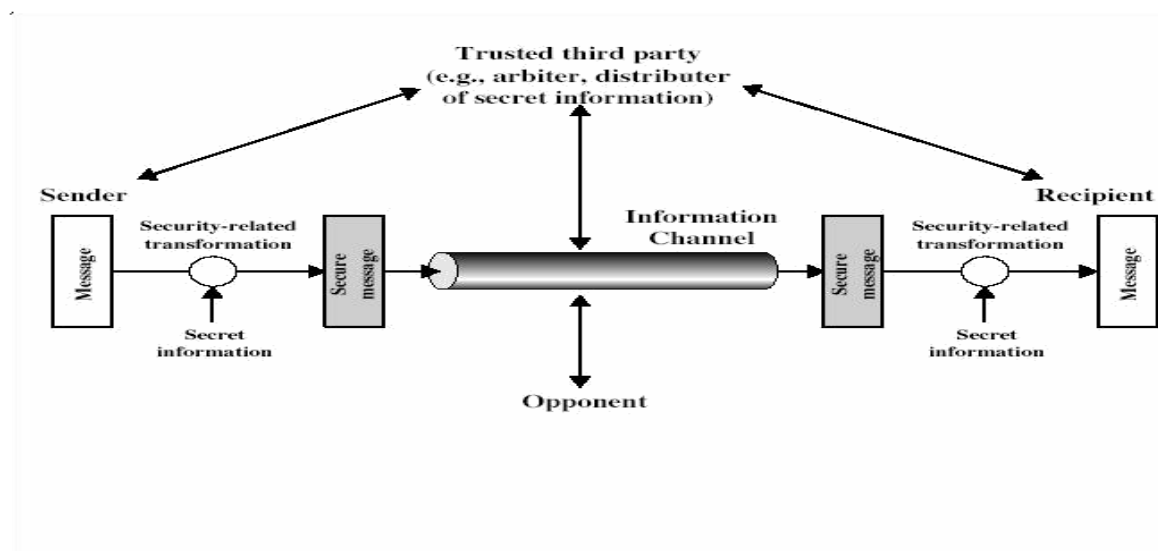
Data is transmitted over network between two communicating parties, who must cooperate for the exchange to take place. A logical information channel is established by defining a route through the internet from source to destination by use of communication protocols by the two parties. Whenever an opponent presents a threat to confidentiality, authenticity of information, security aspects come into play. Two components are present in providing security

- A security-related transformation on the information to be sent making it unreadable by the opponent, and the addition of a code based on the contents of the message, used to verify the identity of sender.
- Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception

A trusted third party may be needed to achieve secure transmission. It is responsible for distributing the secret information to the two parties, while keeping it away from any opponent. It also may be needed to settle disputes between the two parties regarding authenticity of a message transmission. The general model shows that there are four basic tasks in designing a particular security service:

1. Design an algorithm for performing the security-related transformation. The algorithm should be such that an opponent cannot defeat its purpose
2. Generate the secret information to be used with the algorithm
3. Develop methods for the distribution and sharing of the secret information
4. Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service

Threat is placement of some logic in computer system affecting various applications and utility programs. This inserted code presents two kinds of threats.

- **Information access threats** intercept or modify data on behalf of users who should not have access to that data
- **Service threats** exploit service flaws in computers to inhibit use by legitimate users

Viruses and worms are two examples of software attacks inserted into the system by means of a disk or also across the network. The security mechanisms needed to cope with unwanted access fall into two broad categories
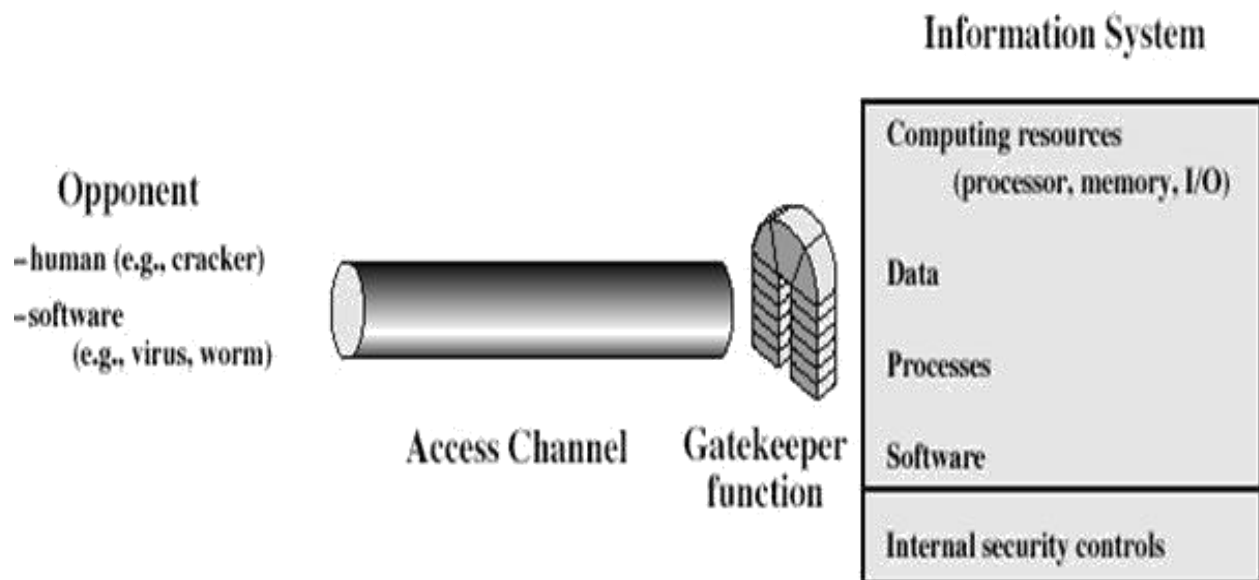
- Placing a gatekeeper function, which includes a password-based login methods that provide access to only authorized users and screening logic to detect and reject worms, viruses etc
- An internal control, monitoring the internal system activities analyzes the stored information and detects the presence of unauthorized users or intruders.

Internet Standards and RFC'S

Most of the protocols related to TCP/IP protocol suite are already standardized or under the process of standardization. An organization known as internet society is responsible for development and publication of these standards. It is the actually a professional membership organization that supervises a large in internet development and standardization

An internet society refers to the organization responsible for monitoring and coordinating internet design, engineering and management. Three organizations under the internet society are responsible for actual work of standards development & publication

1. INTERNET ARICHITECTURE BOARD (IAB): Responsible for defining the overall architecture of the internet, providing guidance and broad direction to IETF

2. INETRNET ENGINEERING TASK FORCE (IETF): The protocol engineering and development arm of the internet

3. INTERNET ENGINEERING STEERING GROUP (IESG): Responsible for technical management of IETF activities and the internet standards process

Working groups chartered by IETF carry out actual development of new standards and protocols for the internet as membership is voluntary; any party can enter into working group will make a draft version made available as an internet draft placed in IETF's "internet drafts" online directory. This will remain up to six months, where interested parties may review & comment on it. During this time, IESG may approve the draft as an RFC or else it is withdrawn from directory, and a revised edition is published.

The IETF is responsible for publishing the RFC'S with approval of IESG. The RFC'S are working notes of the internet research and development community. The entire activities of the IETF are categorized into eight areas each having a categorized into eight areas each having it & numerous working groups

| IETF Area	Theme	Example Working Groups
General	IETF process and procedures	Policy framework Process for organization of Internet standards
Applications	Internet applications	Web-related protocols (HTTP) EDI-Internet integration LDAP
Internet	Internet infrastructure	IPv6 PPP extensions
Operations and management	Standards and definitions for network operations	SNMPv3 Remote network monitoring
Routing	Protocols and management for routing information	Multicast routing OSPF QoS routing
Security	Security protocols and technologies	Kerberos IPSec X.509 S/MIME TLS
Transport	Transport-layer protocols	Differentiated services IP telephony NFS RSVP
User services	Methods to improve the quality of information available to users of the Internet	Responsible use of the Internet User services FYI documents

The Standardization Process:

IESG decides which RFC's become internet standard based on IETF recommendations. To become a standard, a specification must meet the following criteria.
O BE stable and easily understandable o Be
technically competent

- o Have multiple, independent and interoperable implementations with substantial operations experience.
- o Enjoy significant public support.
- o Be recognizably useful in some or all parts of internet

The RFC publication process is shown below, in which a specification passes through a sequence of steps called standards track, in order to qualify as a standard. It involves excessive scrutinizing and testing. The actual process starts after the approval of internet draft documentation as an RFC by IESG.
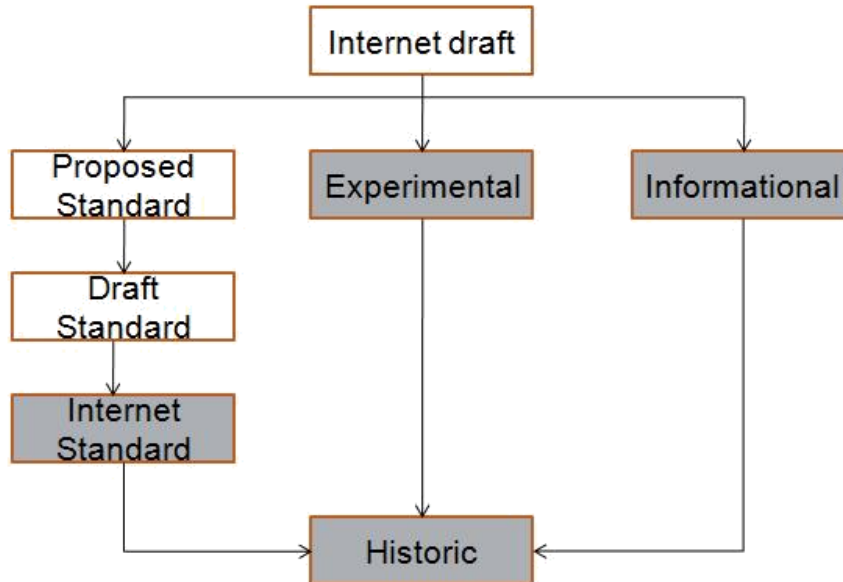
For a specification to act as a draft standard it must pass through at least two non- dependent interoperable implementations for achieving proper operational experience once, necessary implementations and operational experience is achieved, it can be regarded as internet standard. Now, this specification is equipped with two numbers, an STD number and an RFC number .Finally, when a protocol becomes outdated, it is assigned to the historic state.

Internet Standard Categories
All the internet standards fall into two categories

- ☐ <u>TECHINICAL SPECIFICATION (TS):</u> TS defines a protocol, service, procedure, convention or format. Most internet standards are TS's.
- ☐ <u>APPLICABILITY STATEMENT (AS):</u> AS specifies how, and under what circumstances, one or more TS may be applied to support a particular internet capability. It identifies one or more

  TS's that are relevant to the capability and may specify values or ranges for particular parameters associated with a TS or functional subsets of a TS that are relevant for capability.

11

Vulnerability: Vulnerability is an inherent weakness in design, configuration, implementation or management of a network or system that renders it susceptible to a threat. Vulnerabilities are what make networks susceptible to information loss and downtime. Every network and system has some kind of vulnerability.

Buffer Overflow: A buffer overflow occurs when a program or process tries to store more data in a buffer than it was intended to hold..

It happens when the attacker intentionally enters more data than a program was written to handle. The data runs over and overflows the section of valid data like part of programming instructions, user files, confidential information etc there by enabling the attacker's data to overwrite it. This allows an attacker to overwrite data that controls the program and can take over control of the program to execute the attacker's code instead of programmer's code.

Exploiting the over flowable buffer involves the following tasks

- Finding a way of injecting into the buffer

- Specify a return address where malicious code resides for the program to execute the code

- Determining the payload/code to be executed

Buffer Injection Techniques

For creating an exploit, it is important to determine a way of getting a large buffer into the overflowable buffer. A simple process of filling a buffer over the network

- **Injection vector:** It refers to the customized operational code needed to monitor and control an instruction pointer on the remote system. It depends on host and targeted machine and is used to execute the payload.
- **Payload:** Something like a virus that can run at anytime, anywhere irrespective of its injection into a remote machine.

Determining the location of payload

Both injection vector and payload are commonly located in the stack, but the problem with this approach is that one has to keep track of the payload size and how the payload interacts with injection vector. For example, collision occurs when payload starts before injection vector and a jump instruction is included to overcome this which makes the payload jump over injection code. But, if these problems become too complex, the payload has to be placed somewhere else.

Any location in the program, where you can store a buffer becomes a candidate for storing a payload. The main step is to get the processor to start executing that buffer. Some common places to store payloads include

- Files on disk, which are then loaded into memory
- Environment variables controlled by a local user
- Environment variables passed within a web request
- User-controlled fields within a network protocol

Once the payload is injected, the task is simply to get the instruction pointer to load the address of payload. This technique of storing the payload somewhere other than stack has made tight and difficult to exploit buffer overflows very much possible. A single off-by-one error can still be used to take control of a computer.

Methods to execute payload

There are several techniques that are used to execute payload. These are the ways to decide what to put into the saved EIP on the stack to make it finally point to our code.

☐ **Direct Jump (Guessing offsets)**

Here, an overflow code is instructed to jump directly to a specific location in memory. No effort to determine the true location of the stack in memory is made. Though it is simple to use, it has two major drawbacks.

  ☐ If the address of stack contains a null character, the entire payload has to be placed before the injection i.e. reducing the available space for payload.

  ☐ As the address of a payload is not always constant, it requires initial guessing of the address to be jumped.

☐ **Blind Return**

The ESP register points to the current stack location. Any 'ret' instruction will cause the EIP register to be loaded with whatever is pointed to by ESP. this is called 'popping'. Any ret instruction leads to popping of the EIP with top most value on a stack allowing the EIP to point for a new address. If the attacker is able to inject an initial EIP value that points to a ret instruction, the value stored at ESP will be loaded into the ESI.

Nothing can be injected into the instruction pointer that will cause a register to be used for execution. The instruction pointer is made point to a real instruction.

☐ **Pop Return**

If the value on the top of the stack does not point to an address within the attacker's buffer, the injected EIP can be set to point to a series of pop instructions followed by a 'ret'. This causes the stack to be popped a number of times, before a value is used for EIP register.

This technique is useful when there is an address near the top of stack that points to within the attacker's buffer and the attacker just pops down the stack until the useful address is reached.

☐ **Call Register**

If a register is already loaded with an address that points to the payload, the attacker simply needs to load the EIP to an instruction that performs a "call EDX" or "call EDI" or equivalent.

Many useful pairs are found by a search of process memory, and can be used from almost any normal process. As, these are part of kernel interface DLL, they will normally be at fixed address which can be hand coded. These vary for different versions of windows depending on the type of service pack applied.

☐ **Push Return**

It slightly varies from call register method and it also makes use of the value stored in a register. If the register is loaded, but the attacker cannot find a call instruction, another option is to find a "push" followed by a "return".

Stack Frame:

The term 'stack frame' refers to the collection of the entire information related to a stack of any function. The information includes the arguments that are passed to any function, the stored EIP along with any other stored registers and local variables. It can be effectively explained by the 'call' and 'ret' instructions.

☐ **Call Instruction**

This instruction is used to change the processor control in such a way that the control now points to a different piece of code somewhere inside a program, there by notifying the point where to return after executing the function call. The operations are

  ☐ The immediate next instruction after a call is pushed onto the stack to be executed after returning from function.
  ☐ Jump to the address available at the top of a stack.

- **Ret Instruction:** The return instruction takes the control back to the location immediately after a call function in the caller. The operations are
  - The return address at the top of the stack is popped off
  - The address popped off the stack is then jumped

Hence, a combination of 'push' and 'return' statements allow jumping to specific portion of code and returning from it after executing it. As the location of the stored EIP is available onto a stack, writing a popped value at that location is possible.

Format String Vulnerability

. A format string vulnerability occurs when programmers pass externally supplied data to a *printf* function as or as part of the format string argument.

Format string attacks can be used to crash a program or to execute harmful code. The problem stems from the use of unfiltered user input as the format string parameter in certain C functions that perform formatting, such as printf(). The first is where a *printf* function is called with no separate format string argument, simply a single stringargument. A malicious user may use the %s and %x format tokens, among others, to print data from the stack or possibly other locations in memory. One may also write arbitrary data to arbitrary locations using the %n format token, which commands printf() and similar functions to write the number of bytes formatted to an address stored on the stack.

Format string vulnerability attacks fall into three categories: denial of service, reading and writing.

- Format string vulnerability denial of service attacks are characterized by utilizing multiple instances of the %s format specifier to read data off of the stack until the program attempts to read data from an illegal address, which will cause the program to crash.
- Format string vulnerability reading attacks typically utilize the %x format specifier to print sections of memory that we do not normally have access to. This is a serious problem and can lead to disclosure of sensitive information. For example, if a program accepts authentication information from clients and does not clear it immediately after use, these vulnerabilities can be used to read it.
- Format string vulnerability writing attacks utilize the %d, %u or %x format specifiers to overwrite the Instruction Pointer and force execution of user-supplied shell code. This is exploited using single write method or multiple writes method.

Session Hijacking:

Session Hijacking is a common-cum valiant security threat to which most systems are prone to. It refers to the exploitation of a valid computer session to gain unauthorized access toinformation or services in a computer system. Session hijack is a process whereby the attacker inserts themselves into an existing communication session between two computers. The three main protocols that manage the data flow on which session hijacking occurs are TCP, UDP, and HTTP.

Session hijacking can be done at two levels: Network Level and Application Level. Network level hijacking involves TCP and UDP sessions, whereas Application level session hijack occurs with HTTP sessions. The network level refers to the interception and tampering of packets transmitted between client and server during a TCP or UDP session. The application level refers to obtaining session IDs to gain control of the HTTP user session as defined by the web application.

TCP Session Hijacking

TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent. In order to guarantee that packets are delivered in the right order, TCP uses acknowledgement (ACK) packets and sequence numbers to create a "full duplex reliable stream connection between two end points," with the end points referring to the communicating hosts. The connection between the client and the server begins with a three-way handshake.

Fig: The three way handshake method for session establishment and sending Data over TCP

Client sends a synchronization (SYN) packet to the server with initial sequence number X.

- Server responds by sending a SYN/ACK packet that contains the server's own sequence number p and an ACK number for the client's original SYN packet. This ACK number indicates the next sequence number the server expects from the client

- Client acknowledges receipt of the SYN/ACK packet by sending back to the server an ACK packet with the next sequence number it expects from the server, which in this case is P+1.

After the handshake, it's just a matter of sending packets and incrementing the sequence number to verify that the packets are getting sent and received.

The goal of the TCP session hijacker is to create a state where the client and server are unable to exchange data, so that he can forge acceptable packets for both ends, which mimic the real packets. Thus, attacker is able to gain control of the session. At this point, the reason why the client and server will drop packets sent between them is because the server's sequence number no longer matches the client's ACK number and likewise, the client's sequence number no longer matches the server's ACK number. To hijack the session in the TCP network the hijacker should employ following techniques:

- **IP Spoofing:** IP spoofing is "a technique used to gainunauthorized access to computers,whereby the intruder sends messages to a computer with an IP address indicating that the message is coming from a trusted host." Once the hijacker has successfully spoofed an IP address, he determines the next sequence number that the server expects and uses it to inject the forged packet into the TCP session before the client can respond. By doing so, he creates the "desynchronized state."

- **Blind Hijacking:** If source routing is disabled, the session hijacker can also employ blindhijacking where he injects his malicious data into intercepted communications in the TCP session. It is called "blind" because the hijacker can send the data or commands, but cannot see the response. The hijacker is basically guessing the responses of the client and server.
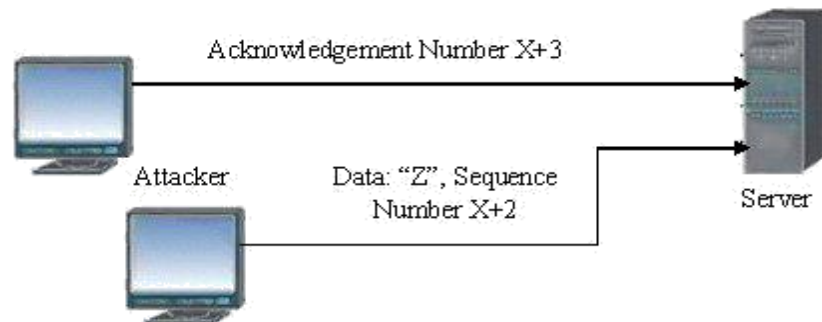

Fig: Blind Injection technique

15

☐**Man in the Middle attack**(packet sniffing): This technique involves using a packet sniffer that intercepts the communication between the client and server. With all the data between the hosts flowing through the hijacker's sniffer, he is free to modify the content of the packets. The trick to this technique is to get the packets to be routed through the hijacker's host.

UDP Session Hijacking

UDP which stands for User Datagram Protocol is defined as "a connectionless protocol that, like TCP, runs on top of IP networks. Unlike TCP/IP, UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagram's over an IP network." Therefore, the delivery, integrity, non-duplication and ordering are not guaranteed i.e. it does not use packet sequencing and synchronizing. UDP doesn't use sequence numbers like TCP. It is mainly used for broadcasting messages across the network or for doing DNS queries.
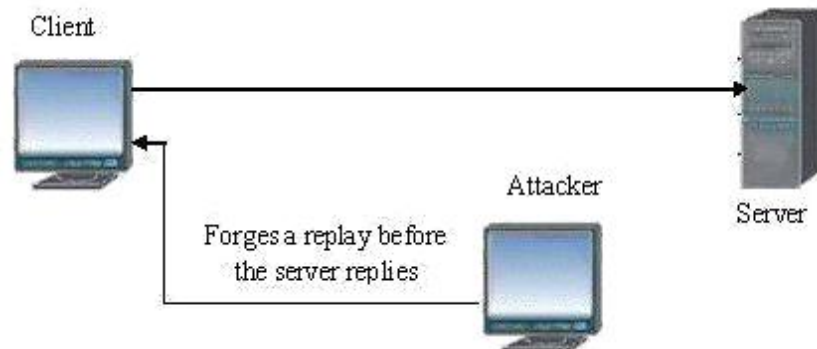
Fig: Session Hijacking over UDP

Hijacking a session over User Datagram Protocol (UDP) is exactly the same as over TCP, except that UDP attackers do not have to worry about the overhead of managing sequence number and other TCP mechanisms. Since UDP is connectionless, injecting data into session without being detected is extremely easy. If the "man in the middle" situation exists, this can be very easy for the attacker, since he can also stop the server's reply from getting to the client in the first place

To defend a network against these attacks, a defender has to implement both security measures at Application level and Network level. Network level hijacks can be prevented by **ciphering the packets** so that the hijacker cannot decipher the packet headers, to obtain anyinformation which will aid in spoofing. This encryption can be provided by using protocols such as *IPSEC, SSL, SSH* etc. To prevent your Application session to be hijacked it is recommended to use **Strong Session ID's** so that they cannot be hijacked or deciphered at any cost.

Route Table Modification:

An attacker would be able to put himself in such a position to block packets by modifying routing tables, so that packets flow through a system he has control of (Layer 3 redirection), by changing bridge tables by playing games with spanning-tree frames (Layer 2 redirection), or by rerouting physical cables so that the frames must flow through the attacker's system (Layer 1 redirection). Most of the time, an attacker will try to change route tables remotely. There has been some research in the area of changing route tables on a mass scale by playing games with the Border Gateway Protocol (BGP) that most Internet service providers (ISPs) use to exchange routes with each other.

A more locally workable attack might be to spoof Internet Control Message Protocol (ICMP) and redirect packets to fool some hosts into thinking that there is a better route via the attacker's IP address. Many OS's accept ICMP redirects in their default configuration. Unless, the connection is to be broken entirely (or proxy it in some way), the packets have to be forwarded back to the real router, so they can reach their ultimate destination. When that happens, the real router is likely to send ICMP redirect packets to the original host, too, informing it that there is a better route. To attempt that sort of attack, it is necessary to keep up the flow of ICMP redirect messages.

If the attacker has managed to change route tables to get packets to flow through his system, some of the intermediate routers will be aware of the route change, either because of route tables changing or possibly because of an Address Resolution Protocol (ARP) table change

.The end nodes would not normally be knowledgeable to this information, if there are at least a few routers between the two nodes. Possibly the nodes could discover the change via a traceroute-style utility, unless the attacker has planned for that and programmed his "router" to account for it (by not sending the ICMP unreachables and not decrementing the Time-to-Live [TTL] counter on the IP packets).

ARP Attacks

Another way to make sure that your attacking machine gets all the packets going through it is to modify the ARP tables on the victim machine(s). An ARP table controls the Media Access Control (MAC)-address-to-IP-address mapping on each machine. ARP is designed to be a dynamic protocol, so as new machines are added to a network or existing machines get new MAC addresses for whatever reason, the rest update automatically in a relatively short period of time. There is absolutely no authentication in this protocol.

Address Resolution Protocol (ARP) spoofing, also known as **ARP poisoning** or **ARP Poison Routing** (APR), is a technique used to attack an Ethernet wired or wireless network. ARP Spoofing allows an attacker to sniff data frames on a local area network (LAN), modify the traffic, or stop the traffic altogether. The attack can only be used on networks that actually make use of ARP and not another method of address resolution.

The principle of ARP spoofing is to send fake, or "spoofed", ARP messages to an Ethernet LAN. Generally, the aim is to associate the attacker's MAC address with the IP address of another node (such as the default gateway). Any traffic meant for that IP address would be mistakenly sent to the attacker instead. The attacker could then choose to forward the traffic to the actual default gateway (passive sniffing) or modify the data before forwarding it (man-in-the-middle attack). The attacker could also launch a denial-of-service attack against a victim by associating a nonexistent MAC address to the IP address of the victim's default gateway. ARP spoofing attacks can be run from a compromised host or from an attacker's machine that is connected directly to the target Ethernet segment. Also spoofed ARP replies are sent at an extremely rapid rate to the switch making its MAC table to overflow and sometimes resulting in switches being reverted to broadcast mode, allowing the sniffing to be done. The best defense against ARP attacks are having a static ARP, DHCP Snooping (access control based on IP, MAC, and port) and detection. Some detection techniques are ARPWatch (Free UNIX Program), Reverse ARP (RARP- used to detect MAC cloning) and Promiscuous Mode Sniffing.

Man in the Middle Attacks

In cryptography, the **man-in-the-middle attack** (often abbreviated **MITM**), is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances (ex: unencrypted Wi-Fi access point).
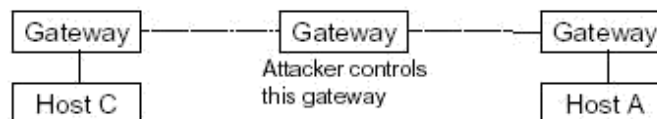
This is not easy in the Internet because of hop-by-hop routing, unless you control one of the backbone hosts or source routing is used. This can also be done combined with IP source routing option. IP source routing is used to specify the route in the delivery of a packet, which is independent of the normal delivery mechanisms. If the traffic can be forced through specific routes (=specific hosts), and if the reverse route is used to reply traffic, a host on the route can easily impersonate another host. Once in the middle, the attacker can perform injection, key manipulation, downgrade attack and filtering.

Injection implies possibility of adding packets to an already established connection or modifying sequence numbers, maintaining connection synchronization while injecting packets. Key manipulation is possible in protocols like SSHv1(modification of public key exchanged by client and server), IPSEC, HTTPS (issuing fake certificates to clients relying on browser mis configuration). Downgrade attacks involve forcing a client to initialize a SSH1 connection rather than SSH2 or sometimes blocking the key material exchanged in IPSEC. In filtering attacks, the attacker can modify the payload of packets by recalculating the checksum or can create filters in the path and in some cases like full-duplex can change the length of payload.

UNIT-2

Conventional encryption principles, conventional encryption algorithms, cipher block modes of operation, location of encryption devices, key distribution approaches of message authentication, secure hash functions and hmac

Conventional Encryption principles

A Symmetric encryption scheme has five ingredients

1. _**Plain Text**_: This is the original message or data which is fed into the algorithm as input.

2. _**Encryption Algorithm**_: This encryption algorithm performs various substitutions and transformations on the plain text.

3. _**Secret Key**_: The key is another input to the algorithm. The substitutions and transformations performed by algorithm depend on the key.
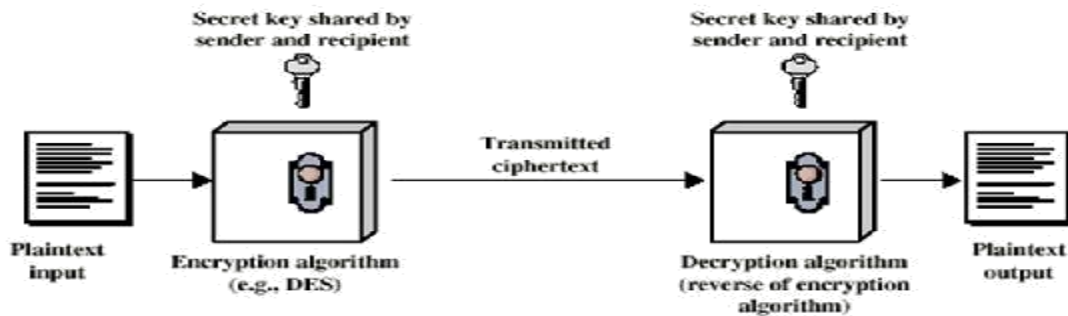
Simplified Model of Conventional Encryption

4. _**Cipher Text**_**:** This is the scrambled (unreadable) message which is output of the encryption algorithm. This cipher text is dependent on plaintext and secret key. For a given plaintext, two different keys produce two different cipher texts.

5. _**Decryption Algorithm**_: This is the reverse of encryption algorithm. It takes the cipher text and secret key as inputs and outputs the plain text.

Two main requirements are needed for secure use of conventional encryption:

(i). A strong encryption algorithm is needed. It is desirable that the algorithm should be in such a way that, even the attacker who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or figure out the key.

(ii). The secret key must be distributed among the sender and receiver in a very secured way. If in any way the key is discovered and with the knowledge of algorithm, all communication using this key is readable.

Cryptography

A cipher is a secret method of writing, as by code. **Cryptography**, in a very broad sense, is the study of techniques related to aspects of information security. Hence cryptography is concerned with the writing (ciphering or encoding) and deciphering (decoding) of messages in secret code. Cryptographic systems are classified along three independent dimensions:

☐ *The type of operations used for performing plaintext to ciphertext*

All the encryption algorithms make use of two general principles; substitution and transposition through which plaintext elements are rearranged. Important thing is that no information should be lost.

☐ *The number of keys used*

If single key is used by both sender and receiver, it is called symmetric, single-key, secret-key or conventional encryption. If sender and receiver each use a different key, then it is called asymmetric, two-key or public-key encryption.

☐ *The way in which plaintext is processed*

A block cipher process the input as blocks of elements and generated an output block for each input block. Stream cipher processes the input elements continuously, producing output one element at a time as it goes along.

Cryptanalysis

The process of attempting to discover the plaintext or key is known as cryptanalysis. It is very difficult when only the cipher text is available to the attacker as in some cases even the encryption algorithm is not known. The most common attack under these circumstances is brute-force approach of trying all the possible keys. This attack is made impractical when the key size is considerably large. The table below gives an idea on types of attacks on encrypted messages.

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext to be decoded
Known plaintext	• Encryption algorithm • Ciphertext to be decoded • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Cryptology covers both cryptography and cryptanalysis. *Cryptology* is a constantly evolving science; ciphers are invented and, given time, are almost certainly breakable. *Cryptanalysis* is the best way to understand the subject of cryptology. *Cryptographers* are constantly searching for the perfect security system, a system that is both fast and hard and a system that encrypts quickly but is hard or impossible to break. *Cryptanalysts* are always looking for ways to break the security provided by a cryptographic system, mostly though mathematical understanding of the cipher structure.

Cryptography can be defined as the conversion of data into a scrambled code that can be deciphered and sent across a public or a private network.

- A *Ciphertext-only attack* is an attack with an attempt to decrypt ciphertext when only the ciphertext itself is available.

- A *Known-plaintext attack* is an attack in which an individual has the plaintext samples and its encrypted version(ciphertext) thereby allowing him to use both to reveal further secret information like the key

- A *Chosen- plaintext attack* involves the cryptanalyst be able to define his own plaintext, feed it into the cipher and analyze the resulting ciphertext.

- A *Chosen-ciphertext attack* is one, where attacker has several pairs of plaintext-ciphertext and ciphertext chosen by the attacker.

An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext and time is available to the opponent. Example for this type is One-time Pad.

An encryption scheme is **computationally secure** if the ciphertext generated by the scheme meets the following criteria:

- Cost of breaking cipher exceeds the value of the encrypted information.

- Time required to break the cipher exceeds the useful lifetime of the information. The average

  time required for exhaustive key search is given below:

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/µs	Time required at 10^6 decryptions/µs
32	$2^{32} = 4.3 \times 10^9$	2^{31} µs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} µs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} µs = 5.4×10^{24} years	5.4×10^{18} years

Feistel Cipher Structure

Most symmetric block ciphers are based on a *Feistel Cipher Structure.* It was first described by Horst Feistel of IBM in 1973 and is still forms the basis for almost all conventional encryption schemes. It makes use of two properties namely *diffusion* and *confusion*; identified by Claude Shannon for frustrating statistical cryptanalysis. Confusion is basically defined as the concealment of the relation between the secret key and the cipher text. On the other hand, diffusion is regarded as the complexity of the relationship between the plain text and the cipher text.
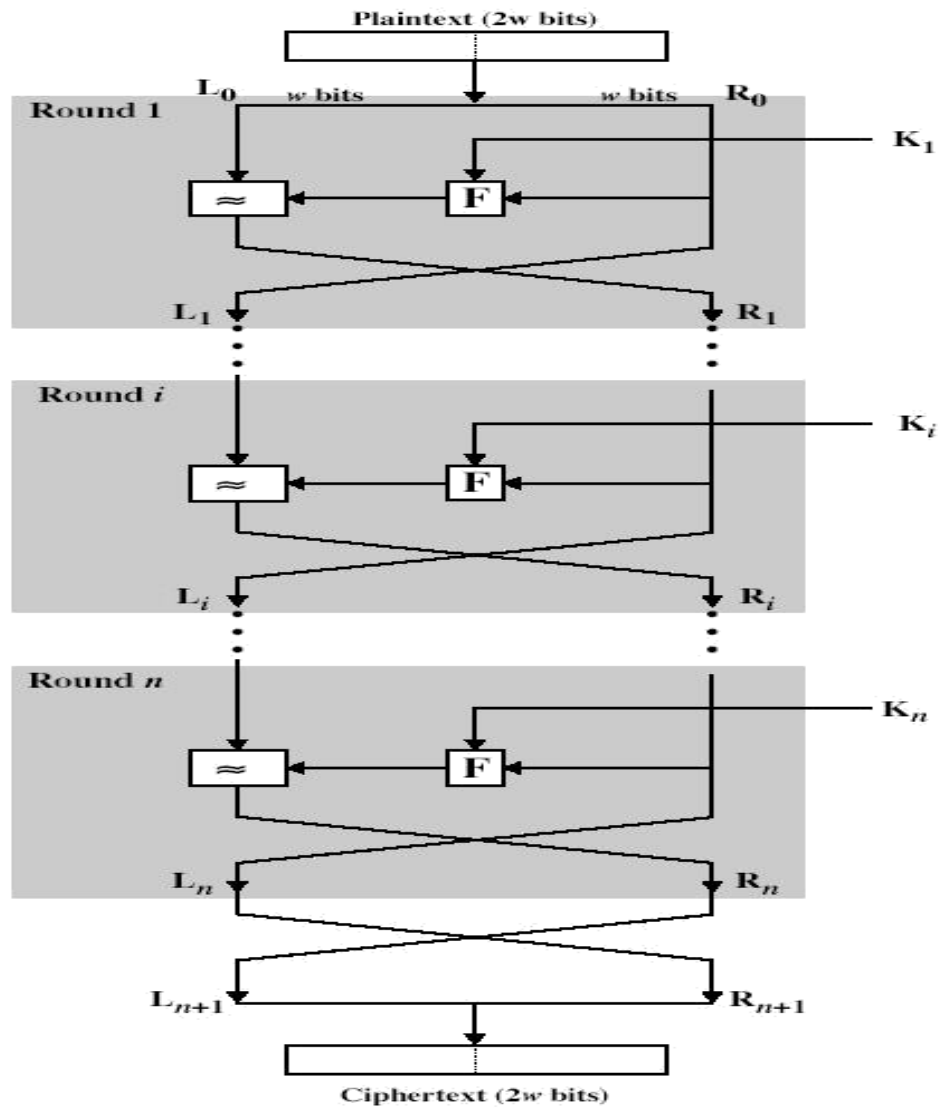
The function of Feistel Cipher is shown in the above figure and can be explained by following steps:
- The input to the encryption algorithm is a plaintext block of length 2w bits and a key K.
- The plaintext block is divided into two halves: L_i and R_i.
- The two halves pass through n rounds of processing and then combine to produce the cipher text block
- Each Round i has inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as a unique subkey K_i generated by a sub-key generation algorithm.
- All rounds have the same structure which involves substitution (mapping) on left half of data, which is done by applying a round function F to right half of data and then taking

  XOR of the output of that function and left half of data. The round function F is common to every round but parameterized by round subkey K_i.
- Then a permutation is performed that consists of interchange of the two halves of data.

For each round $i = 0, 1, \ldots, n$, compute
$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$. Then the ciphertext is ($Rn + 1, Ln + 1$).

Decryption of a ciphertext (R_{n+1}, L_{n+1}) is accomplished by computing for
$$i = n, n-1, \ldots, 0$$

$$R_i = L_{i+1}$$
$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$. Then (L_0, R_0) is the plaintext again.

The structure is a particular form of substitution-permutation network (SPN) proposed by Shannon. The realization or development of a Feistel encryption scheme depends on the choice of the following parameters and design features:

- **Block size:** larger block sizes mean greater security but slower processing. Block size of 64 bits has been nearly universal in block cipher design.

- **Key Size:** larger key size means greater security but slower processing. Most common key length in modern algorithms is 128 bits.

- **Number of rounds:** multiple rounds offer increasing security but slows cipher. Typical size is 16 rounds.

- **Subkey generation algorithm:** greater complexity will lead to greater difficulty of cryptanalysis.

- **Round Function:** greater complexity will make cryptanalysis harder.

- **Fast software en/decryption & ease of analysis:** are more recent concerns for practical use and testing.
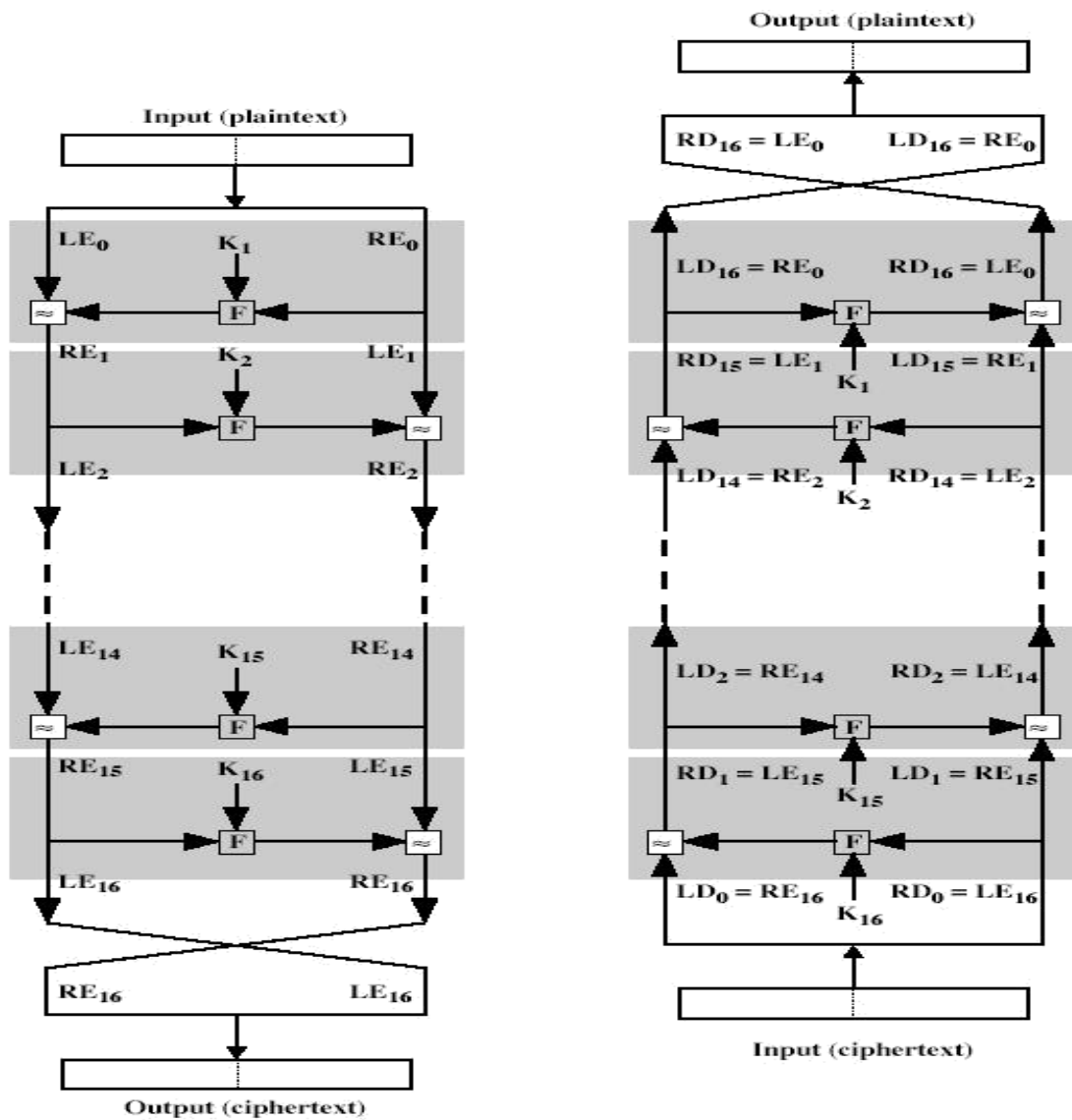
Feistel Cipher Decryption

The process of decryption with a Fiestel cipher is same as the encyption process. Use the ciphertext as input to the algorithm, but use the subkeys K_i in the reverse order. Use K_n in the first round and K_{n-1} in the second round and so on until k_1 is used in the last round. Main advantage is we need not implement two different algorithms for encryption and decryption.

The Fiestel cipher has the advantage that encryption and decryption operations are very similar, even identical in some cases requiring only a reversal in the key schedule. Therefore, the size of the code or circuitry required to implement such a cipher is nearly halved.

24

Simplified DES

S-DES is a reduced version of the DES algorithm. It has similar properties to DES but deals with a much smaller block and key size (operates on 8-bit message blocks with a 10-bit key). The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext. S-DES scheme is shown below:
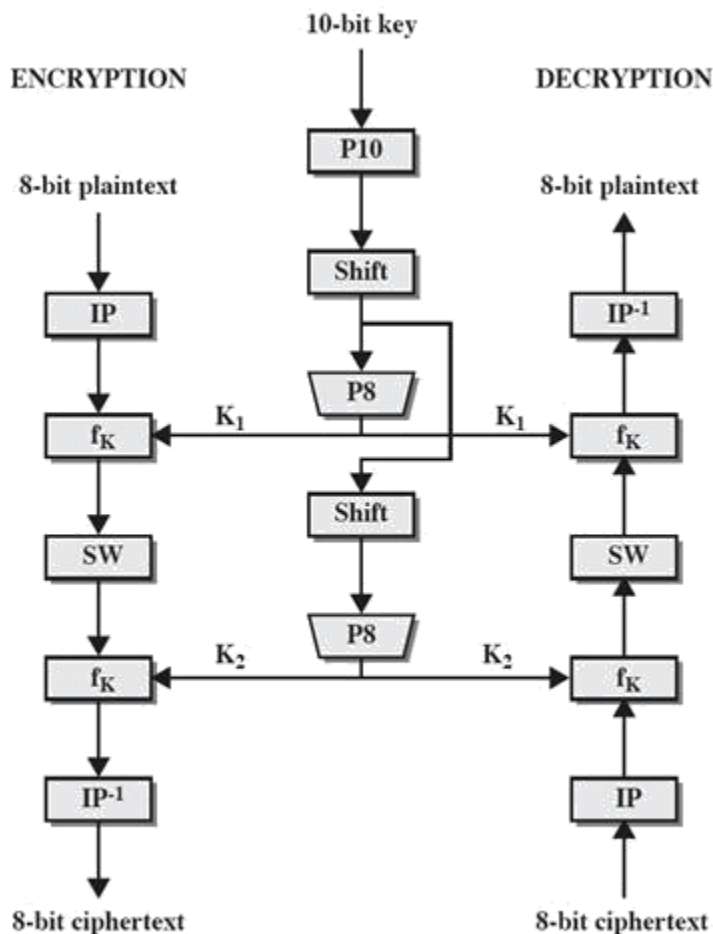
Simplified DES Scheme

The encryption algorithm involves five functions: and initial permuatation(IP), a complex function labeled f_k, which involves both permutations and substitution operations and depends on a key input, a single permutation function (SW) that switches the two halves of the data, the function f_k again and finally a permutation function that is inverse of the IP i.e. IP^{-1}.

As shown in figure, the function f_k takes the data from encryption function along with 8-bit key. The key is choosen to be 10-bit length from which two 8-bit subkeys are generated. The initial 10-bit key is subjected to a permutation (P10) followed by a shift operation. The output of this shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first key (k_1) and also feeds into another shift and another instance of P8 to produce the second subkey (k_2). The encryption algorithm can be written as:

$$\textbf{Ciphertext} = \textbf{IP}^{-1} (\textbf{f}_{k2}(\textbf{SW}(\textbf{f}_{k1}(\textbf{IP(plaintext)}))))$$

Where $K_1 = P8(shift(p10(key)))$
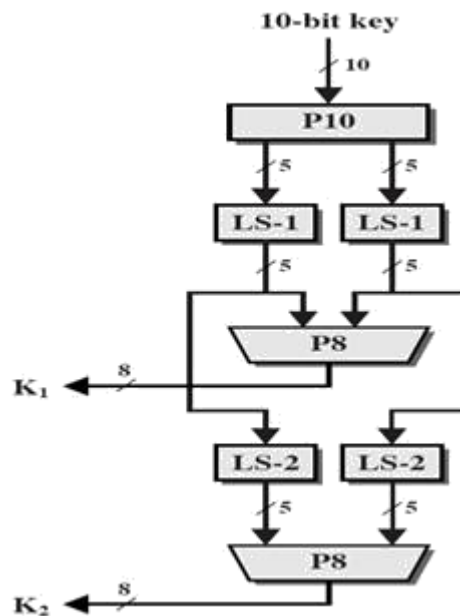$K_2 = P8(shift(shift(p10(key))))$

Decryption is also shown in the above figure and can be given as:

$$\textbf{Plaintext} = \textbf{IP}^{-1} (\textbf{f}_{k1}(\textbf{SW}(\textbf{f}_{k2}(\textbf{IP(ciphertext)}))))$$

Key Generation:

The key generation process is shown below:

As shown above, a 10-bit key shared between sender and receiver is used and fist passed through a permutation P10.

The Switch Function:

This function interchanges the left and right 4 bits so that the second instance of $\textbf{f}_K$ operates on a different 4 bits. For second instance all other parameters remain same, but the key is K_2. The S-boxes operates as follows:-
The first and fourth input bits are treated as 2-bit numbers that specify a row of the S-box, and the second and third input bits specify a column of S-box. The entry in that row and column in base2 is the 2-bit output.

Data Encryption Standard

In 1974, IBM proposed "Lucifer", an encryption algorithm using 64-bit keys. Two years later (1977), NBS (now NIST) in consultation with NSA made a modified version of that algorithm into a standard. DES uses the two basic techniques of cryptography - confusion and diffusion. At the simplest level, diffusion is achieved through numerous permutations and confusion is achieved through the XOR operation and the S-Boxes. This is also called an S-P network The DES encryption scheme can be explained by the following figure
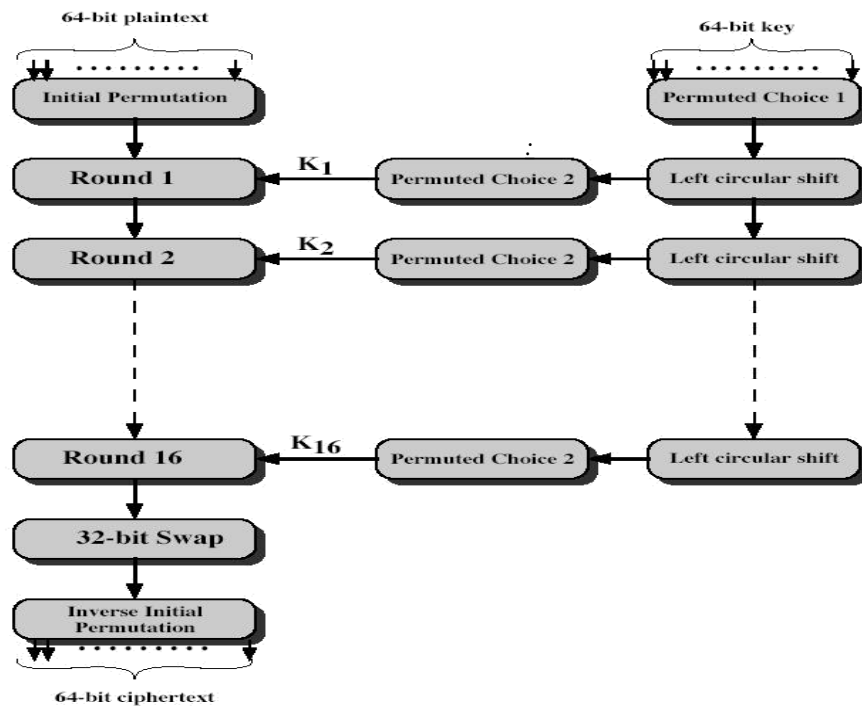
The plain text is 64 bits in length and the key in 56 bits in length. Longer plain text amounts are processed in 64-bit blocks. The main phases in the left hand side of the above figure i.e. processing of the plain text are,

☐ **Initial Permutation (IP):** The plaintext block undergoes an initial permutation. 64 bits of the block are permuted.

☐ **A Complex Transformation:** 64 bit permuted block undergoes 16 rounds of complex transformation. Subkeys are used in each of the 16 iterations.

☐ **32-bit swap:** The output of 16th round consists of 64bits that are a function of input plain text and key.32 bit left and right halves of this output is swapped.

☐ **Inverse Initial Permutation (IP^{-1}):** The 64 bit output undergoes a permutation that is inverse of the initial permutation.

☐ **A Complex Transformation:** 64 bit permuted block undergoes 16 rounds of complex transformation. Subkeys are used in each of the 16 iterations.

☐ **32-bit swap:** The output of 16th round consists of 64bits that are a function of input plain text and key.32 bit left and right halves of this output is swapped.

☐ **Inverse Initial Permutation (IP^{-1}):** The 64 bit output undergoes a permutation that is inverse of the initial permutation.

The following figure shows a closer view of algorithms for a single iteration. The 64bit permuted input passes through 16 iterations, producing an intermediate 64-bit value at the conclusion of each iteration.
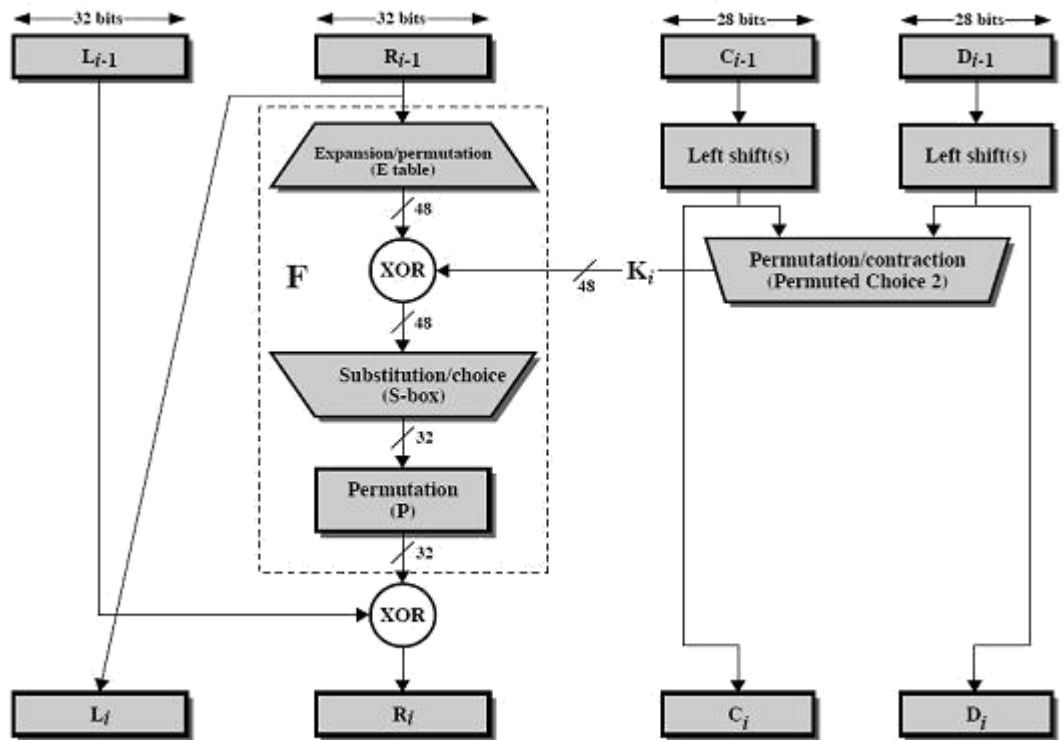
Single Round of DES Algorithm

The left and right halves of each 64 bit intermediate value are treated as separated 32-bit quantities labeled L (left) and R (Right). The overall processing at each iteration is given by following steps, which form one round in an S-P network.

Li = Ri-1.

Ri = L i-1 $\oplus$ F(R i-1, Ki)

Where Function F can be described as **P(S(E(R(i-1)) $\square$ K(i)))**

The left hand output of an iteration (Li) is equal to the right hand input to that iteration Ri-1. The right hand output Ri is exclusive OR of Li-1 and a complex function F of Ri-1 and Ki. The fucntion F can be depicted by the following figure. S1, S2-----S8 represent the "S-boxes" , which maps each combination of 48 input bits into a particular 32 bit pattern. For the generation of subkey of length 48 bits, a 56bit key is used which is first passed through a permutation funciton and then halved to get two 28 bit quantities labeled C0 and D0. At each iteration, these two C and D are subjected to a circular left shift or rotation of 1 or 2 bits. These shifted values serve as input to the next iteration and also to another permutation function which produces a 48-bit output. This output is fed as input to function

F(R $_{i-1}$, Ki).
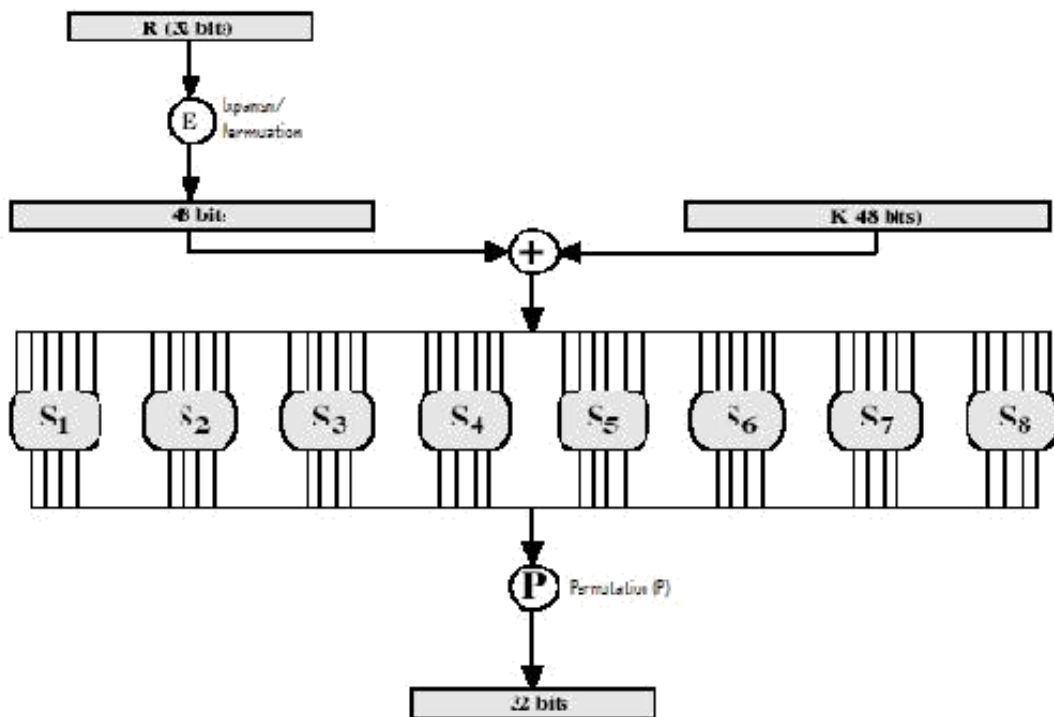
The first and last bits of the input to the box Si form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for Si. The middle 4-bits select a particular column. The decimal value in the cell selected by the row and column is converted to its 4-bit representation to produce the output.

The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. The process of decryption with DES is essentially the same as the encryption process: no different algorithm is used. The ciphertext is used as input to the DES algorithm and the keys are used in the reverse order i.e. K16 in the first iteration, K15 on the second iteration and so on until k1 is used on the sixteenth and last iteration.

Strength of DES:

Avalanche Effect: An effect in DES and other secret key ciphers where each small change in plaintext implies that somewhere around half the ciphertext changes. The avalanche effect makes it harder to successfully cryptanalyze the ciphertext. DES exhibits a strong Avalanche effect.

Concern about the strength of DES falls into two categories i.e. strength of algorithm itself and use of 56-bit key. Though many attempts were made over the years to find and exploit weaknesses in the algorithm, none of them were successful in discovering any fatal weakness in DES. A serious concern is with the key size as the time passed the security in DES became getting compromised by the advent of supercomputers which succeeded in breaking the DES quickly using a brute-force attack. If the only form of attack that could be made on an encryption algorithm is brute force, the way of countering it is obviously using long keys. If a key of size 128 bits is used, it takes approximately 10^{18} years to break the code making the algorithm unbreakable by brute-force approach.

The two analytical attacks on DES are Differential cryptanalysis and Linear cryptanalysis. Both make use of Known plaintext-ciphertext pairs and try to attack the round structure and the S-Boxes. Recent advancements showed that using Differential cryptanalysis, DES can be broken using 2^{47} plaintext-ciphertext pairs and for linear cryptanalysis, the number is even reduced to 2^{41}.

Triple DES

The first answer to problems of DES is an algorithm called Double DES which includes double encryption with two keys. It increases the key size to 112 bits, which seems to be secure. But, there are some problems associated with this approach.

issue of reduction to single stage:

In other words, could there be a key K3 such that $E_{K2}(E_{K21}(P)) = E_{K3}(P)$?

"meet-in-the-middle" attack:

- Works when given a known (P,C) pair
- since $X = E_{K1}(P) = D_{K2}(C)$
- attack by encrypting P with all 2^{56} keys K_1 and store
- then decrypt C with all possible 2^{56} keys K_2 and match X value

Triple DES was the answer to many of the shortcomings of DES. Since it is based on the DES algorithm, it is very easy to modify existing software to use Triple DES. 3DES was developed in 1999 by IBM – by a team led by Walter Tuchman. 3DES prevents a meet-in-the-middle attack. 3DES has a 168-bit key and enciphers blocks of 64 bits. It also has the advantage of proven reliability and a longer key length that eliminates many of the shortcut attacks that can be used to reduce the amount of time it takes to break DES. 3DES uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence.
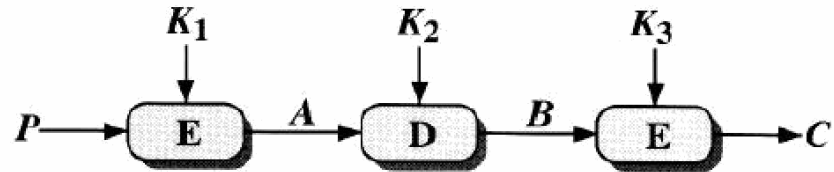
(a) Encryption
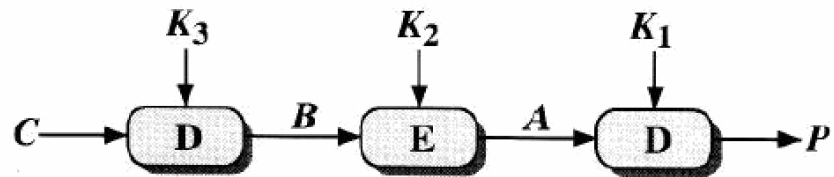
(b) Decryption

$$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$

Where C= ciphertext, P= plaintext and
$E_K[X]$ = encryption of X using key K
$D_K[Y]$ = decryption of Y using key K
Decryption is simply the same operation with the keys reversed

$$P = D_{K1}[E_{K2}[D_{K3}[C]]]$$

Triple DES runs three times slower than standard DES, but is much more secure if used properly. With three distinct keys, TDEA has an effective key length of 168 bits making it a formidable algorithm. As the underlying algorithm is DEA, it offers the same resistance to cryptanalysis as is DEA.
Triple DES can be done using 2 keys or 3 keys.

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. The drawbacks of 3DES being it is very slow and also it uses 64-bit block size same as DES. For reasons of both efficiency and security, a larger key size is desirable. So, NIST (National Institute of Standards and Technology) has called for proposals for a new AES, which should have security strength equal to or better than 3DES and significantly, improved efficiency. NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.

Out of all the algorithms that were submitted, five were shortlisted and upon final evaluation, NIST selected **Rijndael** as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: **Dr. Joan Daemen and Dr. Vincent Rijmen.**

AES Evaluation:

There are three main categories of criteria used by NIST to evaluate potential candidates.
- *Security:* Resistance to cryptanalysis, soundness of math, randomness of output, etc
- *Cost:* Computational efficiency (speed), Memory requirements
- *Algorithm/Implementation Characteristics:* Flexibility, hardware and software suitability, algorithm simplicity

Simplified AES

The encryption algorithm takes a 16-bit block of plaintext as input and a 16-bit key and produces a 16-bit block of ciphertext as output. The S-AES decryption algorithm takes a 16-bit block of ciphertext and the same 16-bit key used to produce that ciphertext as input and produces the original 16-bit block of plaintext as output. The encryption algorithm involves the use of four different functions, or transformations: add key (A_K) nibble substitution (NS), shift row (SR), and mix column (MC).

The encryption algorithm can be expressed as:

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$$, so that A_{K0} is applied first.

The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of four functions; and round 2 contains only 3 functions. Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key. S- AES encryption and decryption scheme is shown below.

Each function operates on a 16-bit state, treated as a 2 x 2 matrix of nibbles, where one nibble equals 4 bits. The initial value of the state matrix is the 16-bit plaintext; the state matrix is modified by each subsequent function in the encryption process, producing after the last function the 16-bit ciphertext. The following figure shows the ordering of nibbles within the matrix is by column. So, for example, the first eight bits of a 16-bit plaintext input to the encryption cipher occupy the first column of the matrix, and the second eight bits occupy the second column. The 16-bit key is similarly organized, but it is somewhat more convenient to view the key as two bytes rather than four nibbles The expanded key of 48 bits is treated as three round keys, whose bits are labelled as follows: $K_0 = k_0...k_{15}$; $K_1 = k_{16}...k_{31}$; $K_2 = k_{32}...k_{47}$.
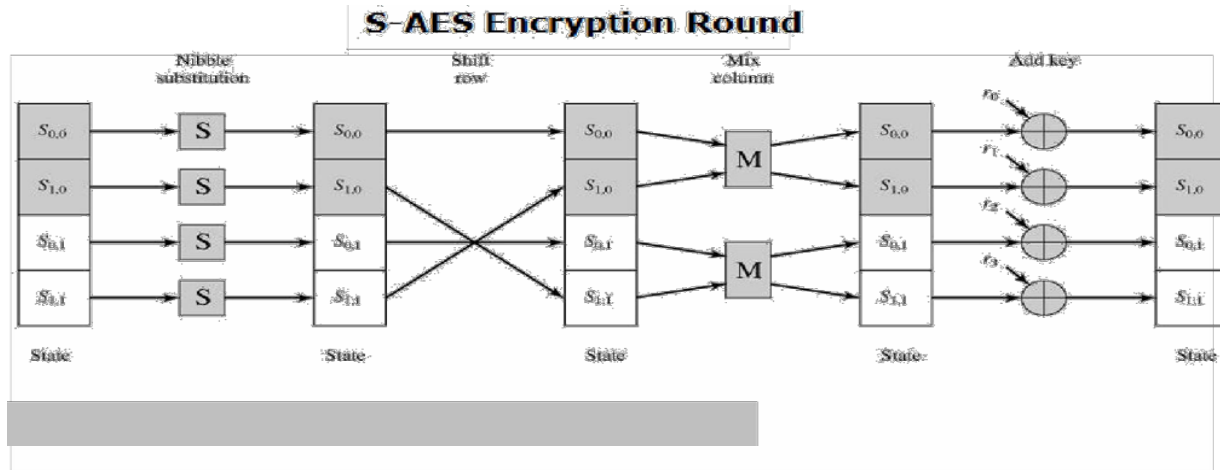
S-AES Data Structures

(a) State matrix

(b) Key

33
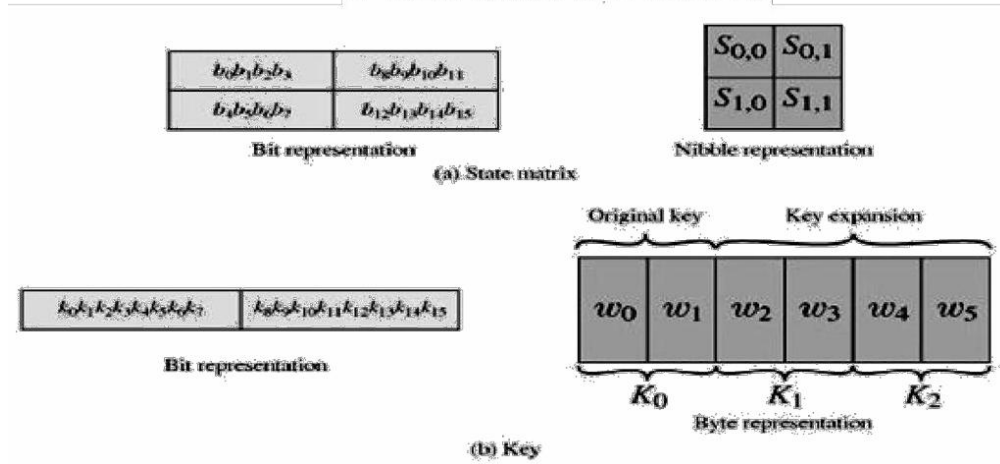
The following figure shows the essential elements of a full round of S-AES. The decryption as shown above can be given as:

$$A_{K_0} \circ INS \circ ISR \circ IMC \circ A_{K_1} \circ INS \circ ISR \circ A_{K_2}$$

in which three of the functions have a corresponding inverse function: inverse nibble substitution (INS), inverse shift row (ISR), and inverse mix column (IMC).

S-AES Encryption and Decryption

The individual functions that are part of the encryption algorithm are given below.

Add Key

The add key function consists of the bitwise XOR of the 16-bit state matrix and the 16-bit round key. As shown in the above example, it can also be viewed as a nibble-wise or bitwise operation. The inverse of the add key function is identical to the add key function, because the XOR operation is its own inverse.

Nibble Substitution

The nibble substitution function is a simple table lookup. AES defines a 4 x 4 matrix of nibble values, called an S-box that contains a permutation of all possible 4-bit values. Each individual nibble of the state matrix is mapped into a new nibble in the following way: The leftmost 2 bits of the nibble are used as a row value and the rightmost 2 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 4-bit output value. For example, the hexadecimal value A references row 2, column 2 of the S-box, which contains the value 0. Accordingly, the value A is mapped into the value 0.
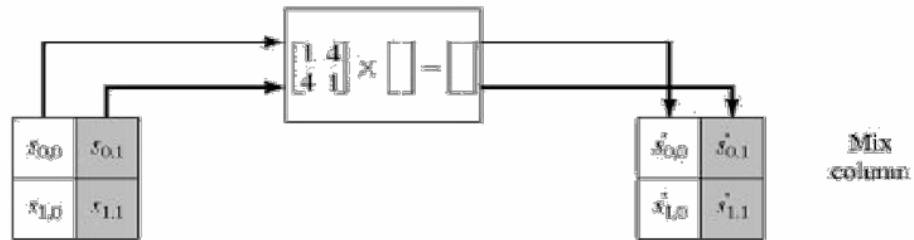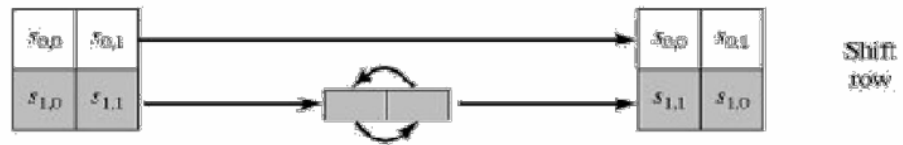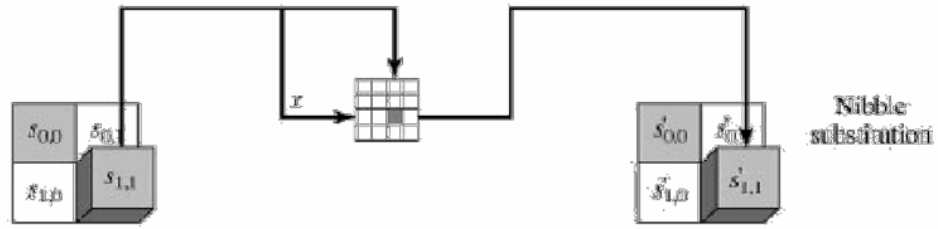
34

For the example, after nibble substitution, the output is

8	1
A	C

$\rightarrow$

6	4
0	C

S-AES Transformations

Nibble substitution

Shift row

Mix column

Add key

35

Shift Row

The shift row function performs a one-nibble circular shift of the second row of the state matrix; the first row is not altered. Our example is shown below:

6	4
0	C

$\rightarrow$

6	4
C	0

The inverse shift row function is identical to the shift row function, because it shifts the second row back to its original position.

Mix Column

The mix column function operates on each column individually. Each nibble of a column is mapped into a new value that is a function of both nibbles in that column. The transformation can be defined by the following matrix multiplication on the state matrix.

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

Where arithmetic is performed in $GF(2^4)$, and the symbol $\cdot$ refers to multiplication in $GF(2^4)$. The example is shown below:

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as follows:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

Key Expansion

For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. The following figure shows the expansion into 6 words, by the calculation of 4 new words from the initial 2 words. The algorithm is as follows:

$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus RCON(1) \oplus SubNib(RotNib(w_1))$$

$$w_3 = w_2 \oplus w_1$$

$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus RCON(2) \oplus SubNib(RotNib(w_3))$$

$$w_5 = w_4 \oplus w_3$$

RCON is a round constant, defined as follows: $\mathbf{RC[i] = x^{i+2}}$, **so that $\mathbf{RC[1]=x^3=1000}$** and $\mathbf{RC[2]=x^4}$ **mod $(\mathbf{x^4 + x + 1})$** $\mathbf{= x + 1 = 0011}$. RC[i] forms the leftmost nibble of a byte, with the rightmost nibble being all zeros. Thus, RCON(1) = 10000000 and RCON(2) = 00110000.

For example, suppose the key is 2D55 = 0010 1101 0101 0101 = $w_0 w_1$. Then,

$$w_2 = 00101101 \oplus 10000000 \oplus \text{SubNib}(01010101)$$

$$= 00101101 \oplus 10000000 \oplus 00010001 = 10111100$$

$$w_3 = 10111100 \oplus 01010101 = 11101001$$

$$w_4 = 10111110 \oplus 00110000 \oplus \text{SubNib}(10011110)$$

$$= 10111100 \oplus 00110000 \oplus 00101111 = 10100011$$

$$w_5 = 10100011 \oplus 11101001 = 01001010$$

The S-Box

The S-box is constructed as follows:

- Initialize the S-box with the nibble values in ascending sequence row by row. The first row contains the hexadecimal values 0, 1, 2, 3; the second row contains 4, 5, 6, 7; and so on. Thus, the value of the nibble at row i, column j is 4i + j.
- Treat each nibble as an element of the finite field $GF(2^4)$ modulo $x^4 + x + 1$. Each nibble $a_0 a_1 a_2 a_3$ represents a polynomial of degree 3.
- Map each byte in the S-box to its multiplicative inverse in the finite field $GF(2^4)$ modulo $x^4 + x + 1$; the value 0 is mapped to itself.
- Consider that each byte in the S-box consists of 4 bits labeled (b_0, b_1, b_2, b_3). Apply the following transformation to each bit of each byte in the S-box: The AES standard depicts this transformation in matrix form as follows:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The prime (') indicates that the variable is to be updated by the value on the right. Remember that addition and multiplication are being calculated modulo 2.

To apply a block cipher in a variety of applications, four "modes of operation" have been defined by NIST (FIPS 81). The four modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used. As new applications and requirements have appeared, NIST has expanded the list of recommended modes to five in Special Publication 800-38A. These modes are intended for use with any symmetric block cipher, including triple DES and AES.

Electronic Codebook Book (ECB)

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key. *ECB is the simplest of the modes, and is used when only a single block of info needs to be sent.*

(a) Encryption

(b) Decryption
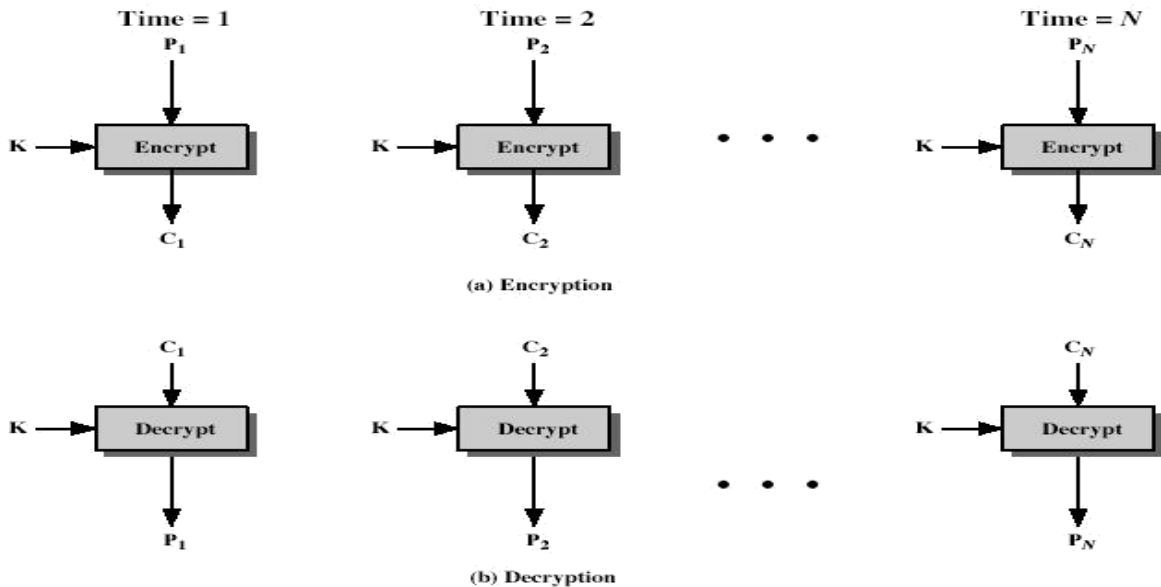
Break the plaintext into 64-bit blocks and encrypt each of them with the same key. The last block should be padded to 64-bit if it is shorter. Same block and same key always yields same cipher block. Each block is a value which is substituted, like a codebook, hence the name Electronic Code Book. Each block is encoded independently of the other blocks.

$$C_i = DES_{K1}(P_i)$$

ECB is not appropriate for any quantity of data, since repetitions can be seen, esp. with graphics, and because the blocks can be shuffled/inserted without affecting the en/decryption of each block. Its main use is to send one or a very few blocks, eg a session encryption key.

Cipher Block Chaining Mode (CBC)

To overcome the problems of repetitions and order independence in ECB, want some way of making the ciphertext dependent on **all** blocks before it. This is what CBC gives us, by combining the previous ciphertext block with the current message block before encrypting. To start the process, use an Initial Value (IV), which is usually well known (often all 0's), or otherwise is sent, ECB encrypted, just before starting CBC use.

(a) Encryption
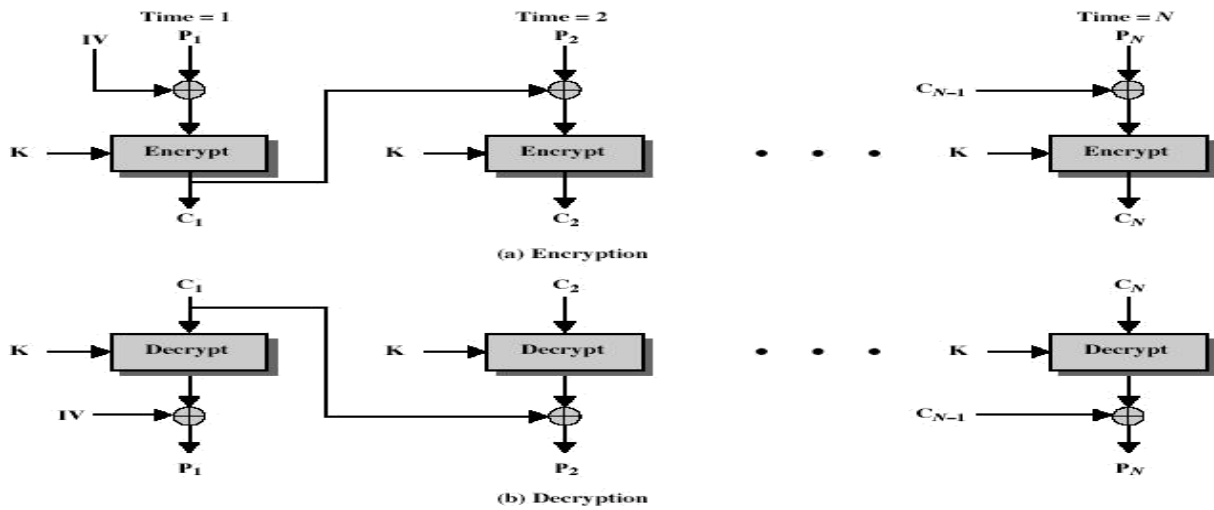
(b) Decryption

All cipher blocks will be chained so that if one is modified, the ciphertext cannot be decrypted correctly. Each plaintext block is XORed with the previous cipher block before encryption, hence the name CBC. The first plaintext block is XORed with an initialization vector IV, which is to be protected securely, (e.g., send it encrypted in ECB mode).

$$C_i = DES_{K1}(P_i \text{ XOR } C_{i-1})$$

CBC is the block mode generally used. The chaining provides an avalanche effect, which means the encrypted message cannot be changed or rearranged without totally destroying the subsequent data. However there is the issue of ensuring that the IV is either fixed or sent encrypted in ECB mode to stop attacks on 1st block.\

r Feed Back Mode (CFB)

If the data is only available a bit/byte at a time (eg. terminal session, sensor value etc), then must use some other approach to encrypting it, so as not to delay the info. it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as plaintext. Thus, if 8-bit characters are being transmitted, each character should beencrypted to produce a cipher text output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

(a) Encryption
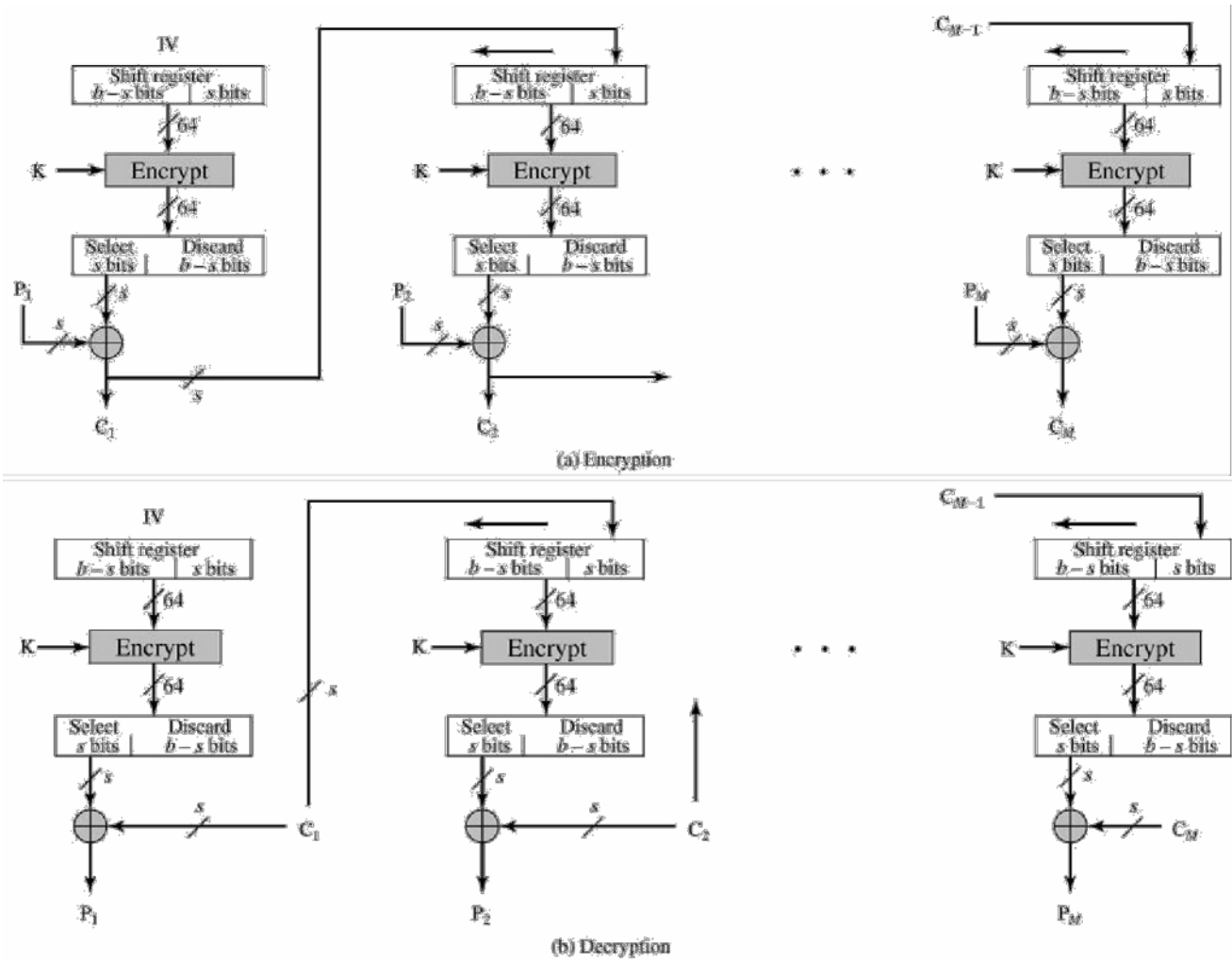
(b) Decryption

The input to the encryption function is a b-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment of plaintext P_1 to produce the first unit of ciphertext C_1, which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted. For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function.

$$C_i = P_i \text{ XOR } DES_{K1}(C_{i-1})$$

CFB is the usual stream mode. As long as can keep up with the input, doing encryptions every 8 bytes. A possible problem is that if its used over a "noisy" link, then any corrupted bit will destroy values in the current and next blocks (since the current block feeds as input to create the random bits for the next). So either must use over a reliable network transport layer (pretty usual) or use OFB.

Output Feedback Mode (OFB)

The output feedback (OFB) mode is similar in structure to that of CFB. It is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.

(a) Encryption

(b) Decryption
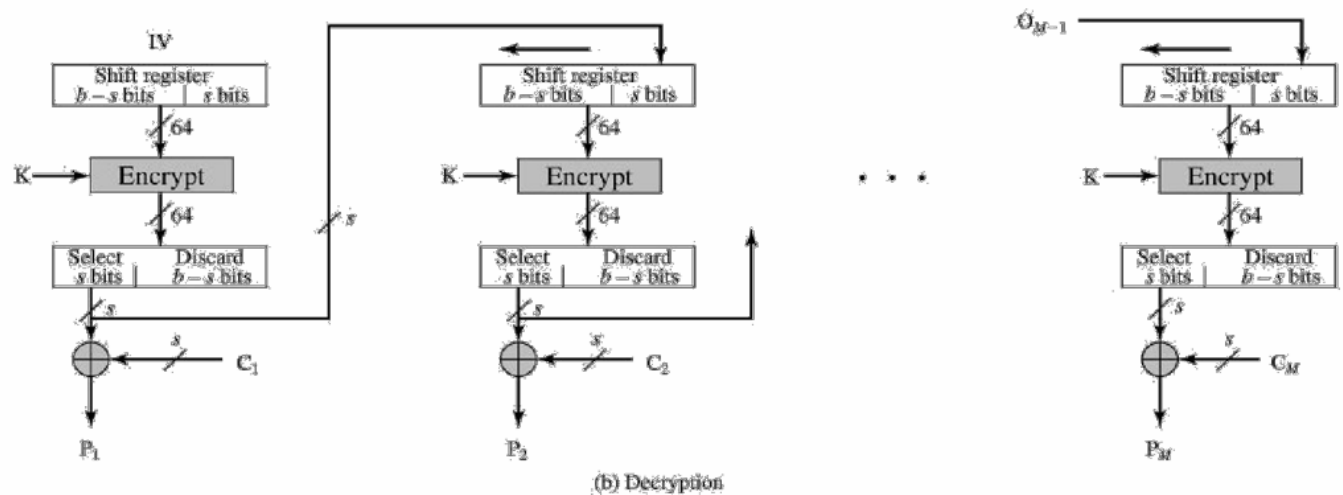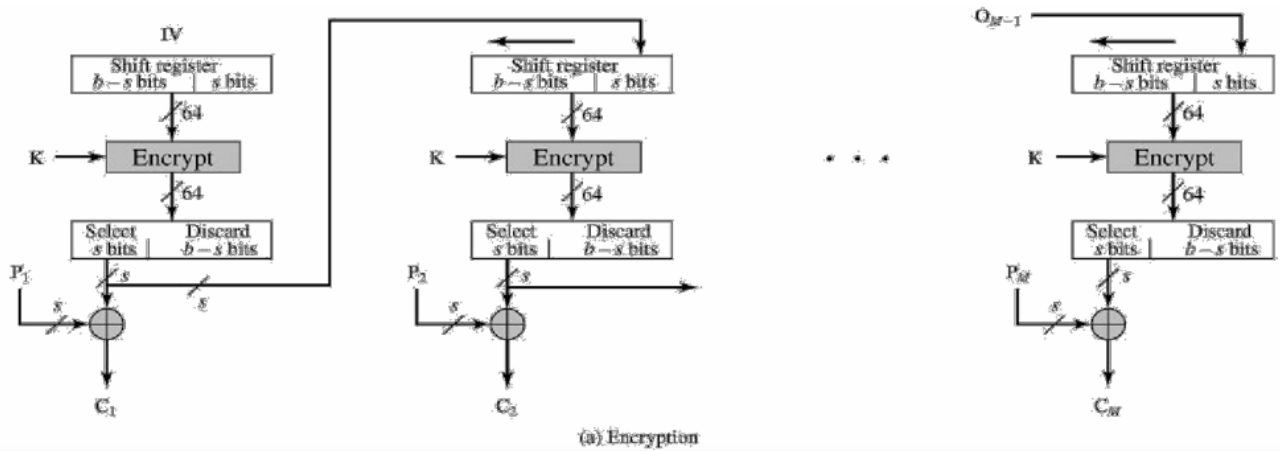
Keystream is independent of the data and can be computed in advance.

$$C_i = P_i \text{ XOR } O_i \, O_i$$
$$= DES_{K1}(O_{i-1})$$

Here the generation of the "random" bits is independent of the message being encrypted. The advantage is that firstly, they can be computed in advance, good for bursty traffic, and secondly, any bit error only affects a single bit. Thus this is good for noisy links (eg satellite TV transmissions etc). The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

Counter Mode (CTR)

The Counter (CTR) mode is a variant of OFB, but which encrypts a counter value (hence name). Although it was proposed many years before, it has only recently been standardized for use with AES along with the other existing 4 modes. It is being used with applications in ATM (asynchronous transfer mode) network security and IPSec (IP security).

All modes of operations except ECB make random access to the file impossible: to access data at the end of the file one has to decrypt everything. Plaintext is not encrypted directly. IV plus a constant is encrypted and the resulting ciphertext is XORed with the plaintext – add 1 to IV in each step.

(a) Encryption

(b) Decryption
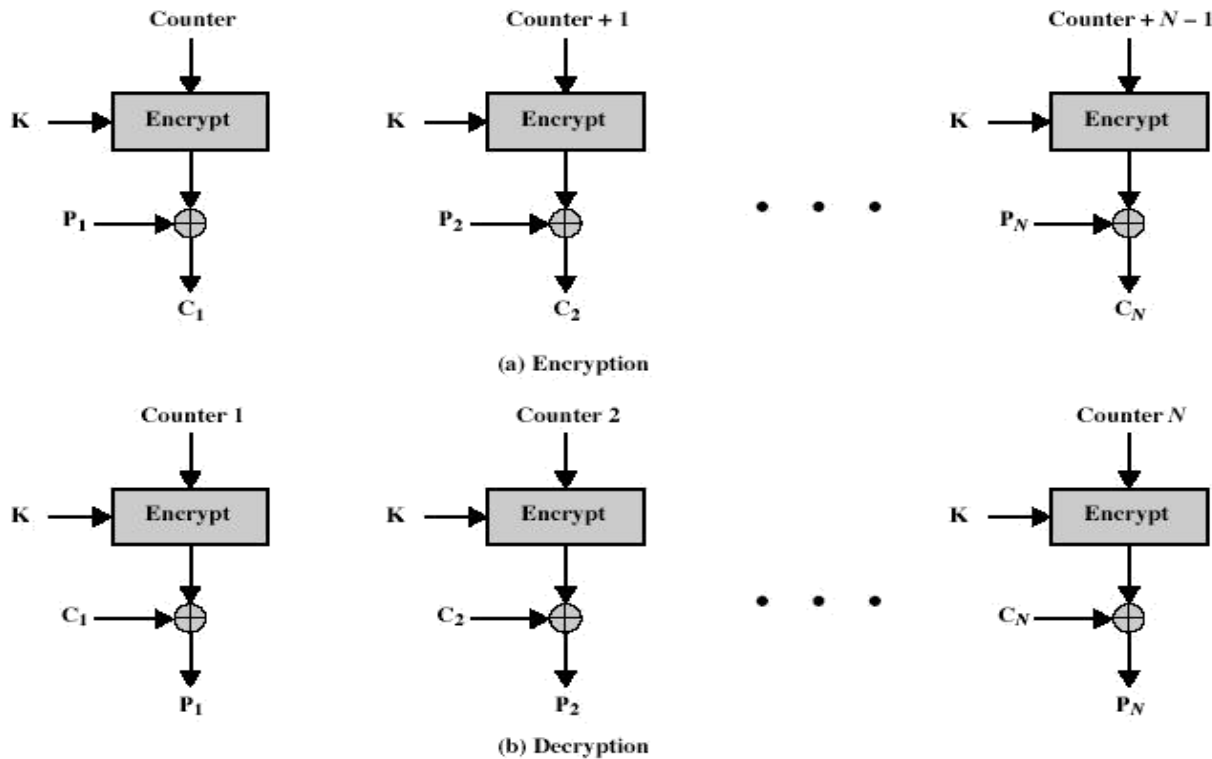
If the same IV is used twice with the same key, then cryptanalyst may XOR the ciphers to get the XOR of the plaintexts –this could be used in an attack. A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically the counter is initialized to some value and then incremented by 1 for each subsequent block.

CTR mode has a number of advantages in parallel h/w & s/w efficiency, can preprocess the output values in advance of needing to encrypt, can get random access to encrypted data blocks, and is simple. But like OFB have issue of not reusing the same key + counter value.

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. It is intended against the attacks like content modification, sequence modification, timing modification and repudiation. For repudiation, concept of digital signatures is used to counter it. There are three classes by which different types of functions that may be used to produce an authenticator. They are:

- *Message encryption*–the ciphertext serves as authenticator

- *Message authentication code (MAC)*–a public function of the message and a secret key producing a fixed-length value to serve as authenticator. This does not provide a digital signature because A and B share the same key.

- *Hash function*–a public function mapping an arbitrary length message into a fixed-length hash value to serve as authenticator. This does not provide a digital signature because there is no key.

Message Encryption:

Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes. The message must have come from the sender itself, because the ciphertext can be decrypted using his (secret or public) key. Also, none of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext.
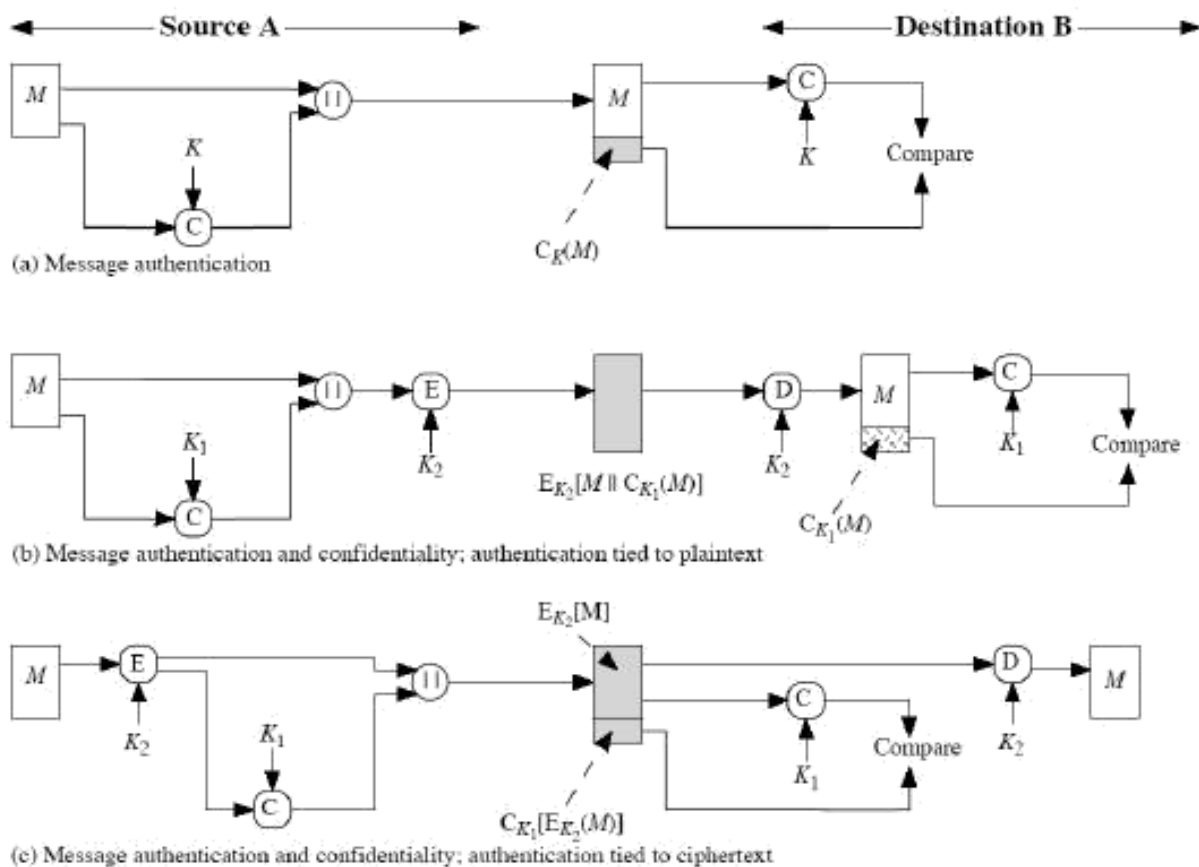
Often one needs alternative authentication schemes than just encrypting the message.
- Sometimes one needs to avoid encryption of full messages due to legal requirements.
- Encryption and authentication may be separated in the system architecture.

Message Authentication Code

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as cryptographic checksum or MAC, which is appended to the message. This technique assumes that both the communicating parties say A and B share a common secret key K. When A has a message to send to B, it calculates MAC as a function C of key and message given as: $\mathbf{MAC=C_k(M)}$

The message and the MAC are transmitted to the intended recipient, who upon receiving performs the same calculation on the received message, using the same secret key to generate a new MAC. The received MAC is compared to the calculated MAC and only if they match, then:

1. The receiver is assured that the message has not been altered: Any alternations been done the MAC's do not match.

2. The receiver is assured that the message is from the alleged sender: No one except the sender has the secret key and could prepare a message with a proper MAC.

3. If the message includes a sequence number, then receiver is assured of proper sequence as an attacker cannot successfully alter the sequence number.
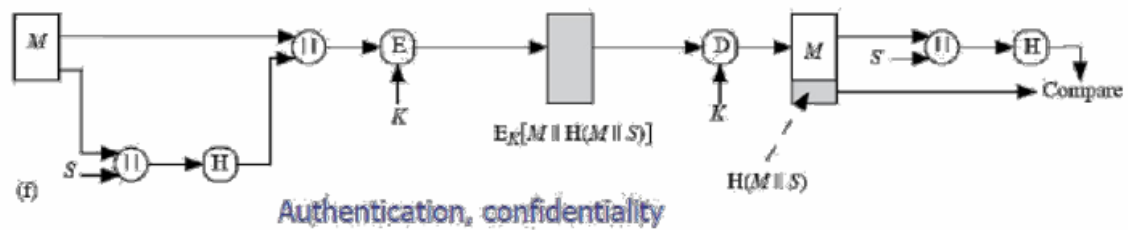
(a) Message authentication

(b) Message authentication and confidentiality; authentication tied to plaintext

(c) Message authentication and confidentiality; authentication tied to ciphertext

There are three different situations where use of a MAC is desirable:

- If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity –message will be sent in plain with an attached authenticator.

- If one side has a heavy load, it cannot afford to decrypt all messages –it will just check the authenticity of some randomly selected messages.

- Authentication of computer programs in plaintext is very attractive service as they need not be decrypted every time wasting of processor resources. Integrity of the program can always be checked by MAC.

Hash Function

A variation on the message authentication code is the one-way hash function. As with the message authentication code, the hash function accepts a variable-size message M as input and produces a fixed-size hash code H(M), sometimes called a message digest, as output. The hash code is a function of all bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

(a) Confidentiality and authentication

$E_K[M \| H(M)]$

(b) Authentication

$E_K[H(M)]$

(c) Authentication, digital signature

$E_{KR_a}[H(M)]$

(d) Authentication, digital signature, confidentiality

$E_K[M \| E_{KR_a}[H(M)]]$

(e) Authentication (no encryption needed!)

$H(M \| S)$

(f) Authentication, confidentiality

$E_K[M \| H(M \| S)]$
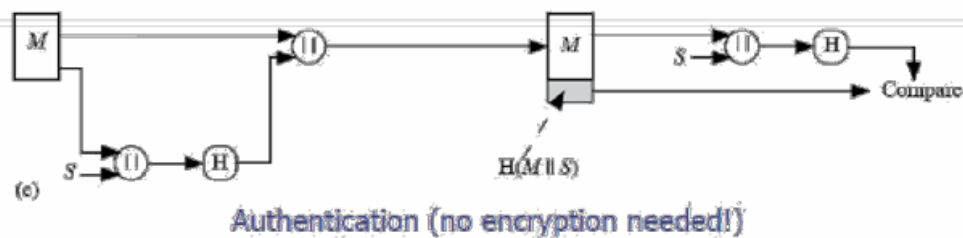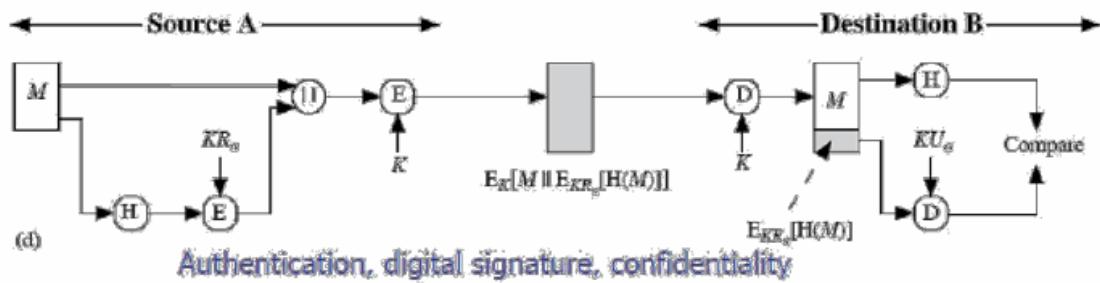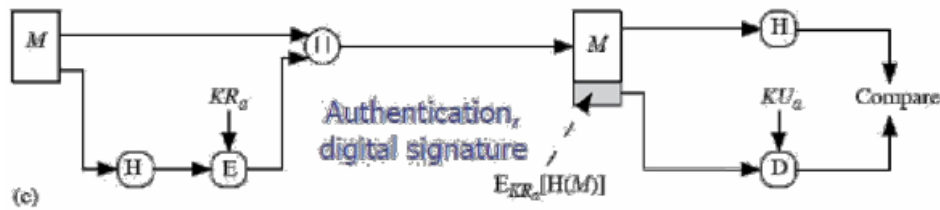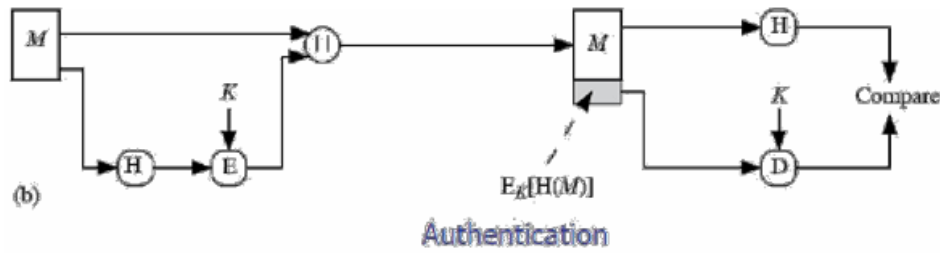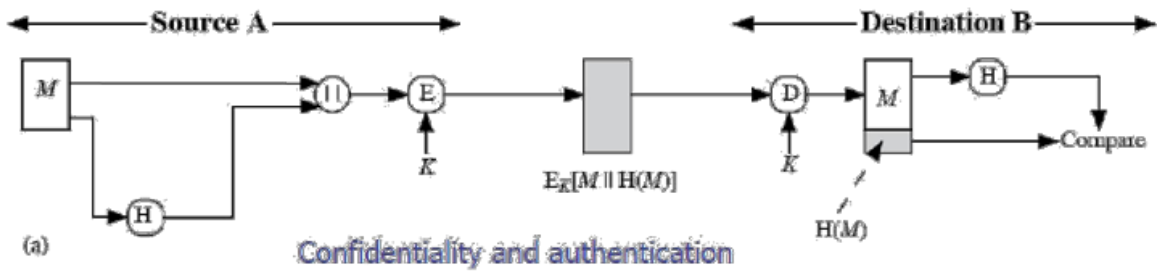
In cases where confidentiality is not required, methods b and c have an advantage over those that encrypt the entire message in that less computation is required. Growing interest for techniques that avoid encryption is due to reasons like, Encryption software is quite slow and may be covered by patents. Also encryption hardware costs are not negligible and the algorithms are subject to U.S export control.

A fixed-length hash value h is generated by a function H that takes as input a message of arbitrary length:

h=H(M).

- A sends M and H(M)
- B authenticates the message by computing H(M) and checking the match

Requirements for a hash function: The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be used for message authentication, the hash function H must have the following properties

- H can be applied to a message of any size
- H produces fixed-length output
- Computationally easy to compute H(M) for any given M
- Computationally infeasible to find M such that H(M)=h, for a given h, referred to as the *one-way property*
- Computationally infeasible to find M' such that H(M')=H(M), for a given M, referred to as *weak collision resistance*.

- Computationally infeasible to find M,M' with H(M)=H(M') (to resist to birthday attacks), referred to as *strong*

collision resistance. Examples of simple hash functions are:
- Bit-by-bit XOR of plaintext blocks: h= D1⊕D2⊕…⊕DN
- rotated XOR –before each addition the hash value is rotated to the left with 1 bit
- Cipher block chaining technique without a secret key.

MD5 Message Digest Algorithm

The MD5 message-digest algorithm was developed by Ron Rivest at MIT and it remained as the most popular hash algorithm until recently. The algorithm takes as input, a message of arbitrary length and produces as output, a 128-bit message digest. The input is processed in 512-bit blocks. The processing consists of the following steps:

1.) *Append Padding bits*: The message is padded so that its length in bits is congruent to 448 modulo 512 i.e. the length of the padded message is 64 bits less than an integer multiple of 512 bits.Padding is always added, even if the message is already of the desired length. Padding consists of a single 1-bit followed by the necessary number of 0-bits.

2.) *Append length*: A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step-1. If the length is larger than 264, the 64 least representative bits are taken.

3.) *Initialize MD buffer*: A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D) and are initialized with A=0x01234567, B=0x89ABCDEF, C=0xFEDCBA98, D=0x76543210 i.e. 32-bit integers (hexadecimal values).
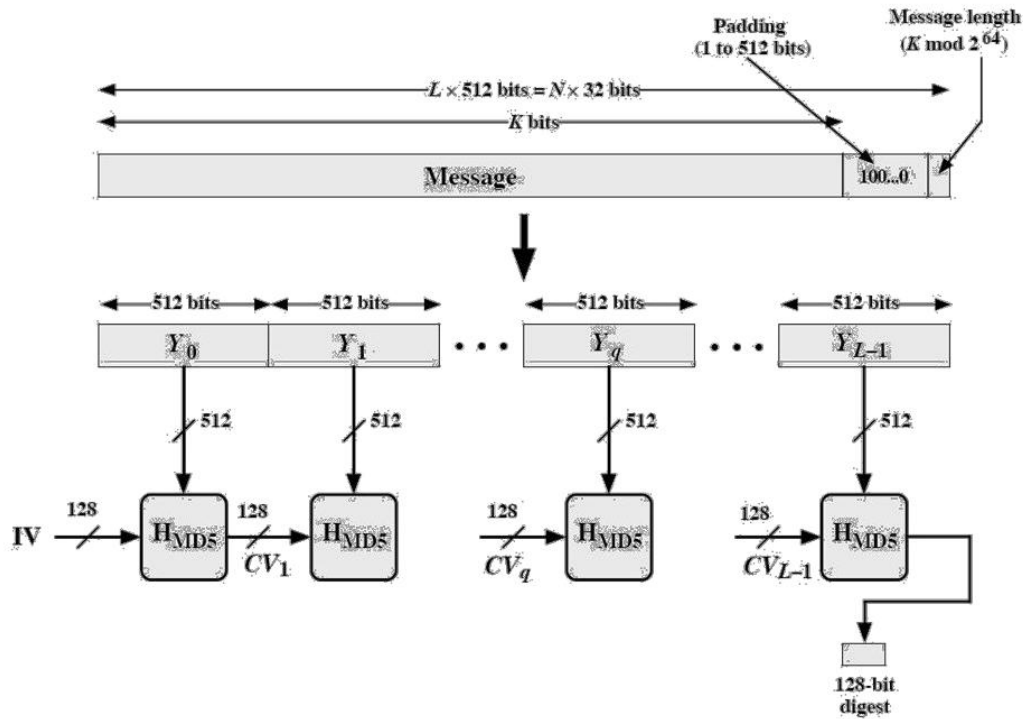
Message Digest Generation Using MD5

4.) *Process Message in 512-bit (16-word) blocks*: The heart of algorithm is the compression function that consists of four rounds of processing and this module is labeled HMD5 in the above figure and logic is illustrated in the following figure. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H and I in the specification. Each block takes as input the current 512-bit block being processed Yq and the 128-bit buffer value ABCD and updates the contents of the buffer. Each round also makes use of one-fourth of a 64-element table T*1....64+, constructed from the sine function. The ith element of T, denoted T[i], has the value equal to the integer part of 2^{32} * abs(sin(i)), where i is in radians. As the value of abs(sin(i)) is a value between 0 and 1, each element of T is an integer that can be represented in

32-bits and would eliminate any regularities in the input data. The output of fourth round is added to the input to the first round (CV_q) to produce CV_{q+1}. The addition is done independently for each of the four words in the buffer with each of the corresponding words in CV_q, using addition modulo 2^{32}. This operation is shown in the figure below:
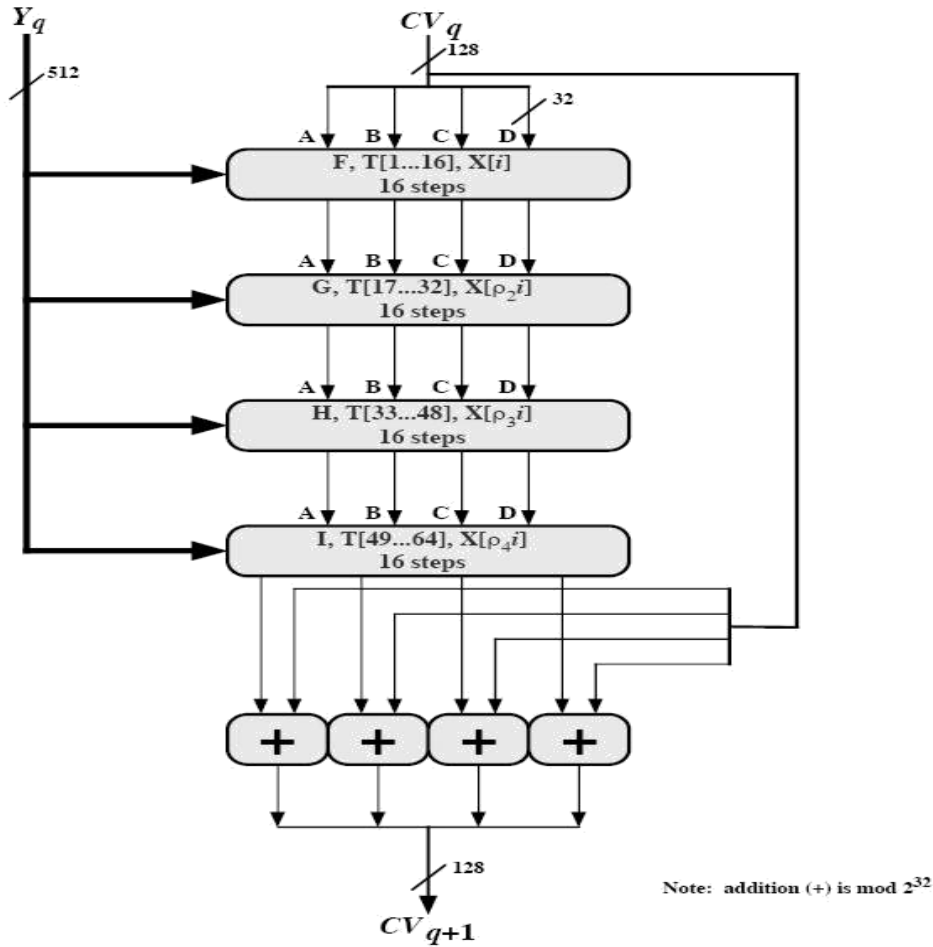
5.) <u>*Output*</u>: After all L 512-bit blocks have been processed, the output from the Lth stage is the 128-bit message digest.
MD5 can be summarized as follows:

$$CV0 = IV$$
$$CV_{q+1} = SUM_{32}(CV_q, RF_I Y_q RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$$
$$MD = CV_L$$
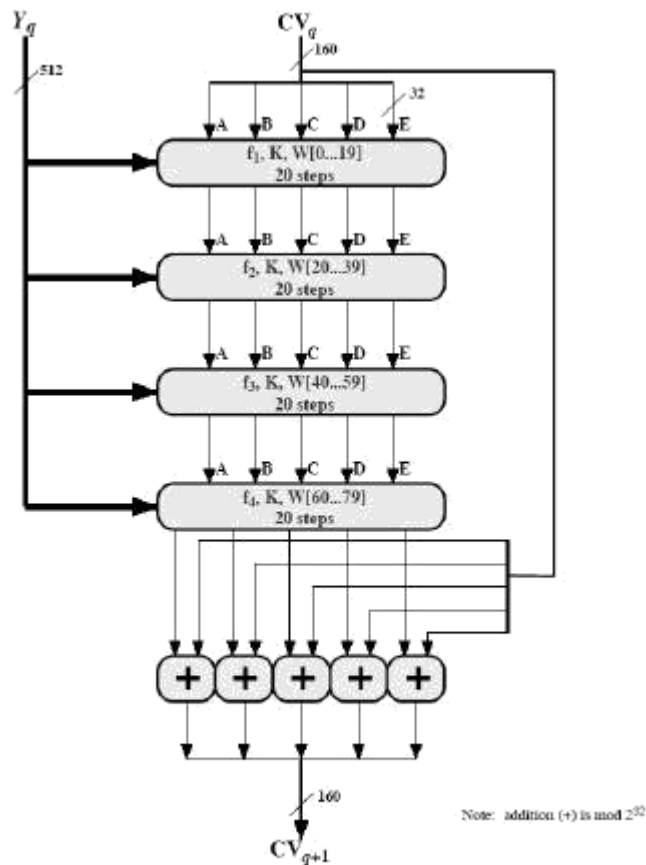
Where,

IV = initial value of ABCD buffer, defined in step 3.

Y_q = the qth 512-bit block of the message

L = the number of blocks in the message

CV_q = chaining variable processed with the qth block of the message.

RFx = round function using primitive logical function x.

MD = final message digest value

SUM_{32} = Addition modulo 2^{32} performed separately.

Secure Hash Algorithm:

The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST). SHA-1 is the best established of the existing SHA hash functions, and is employed in several widely used security applications and protocols. The algorithm takes as input a message with a maximum length of less than 2^{64} bits and produces as output a 160-bit message digest

1 .) *Append Padding Bits:* The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.

2.) *Append Length:* a block of 64 bits containing the length of the original message is added.

3.) *Initialize MD buffer*: A 160-bit buffer is used to hold intermediate and final results on the hash function. This is formed by 32-bit registers A,B,C,D,E. Initial values: A=0x67452301, B=0xEFCDAB89, C=0x98BADCFE, D=0x10325476, E=C3D2E1F0. Stores in big-endian format i.e. the most significant bit in low address.

4.) *Process message in blocks 512-bit (16-word) blocks*: The processing of a single 512-bit block is shown above. It consists of four rounds of processing of 20 steps each. These four rounds have similar structure, but uses a different primitive logical function, which we refer to as f1, f2, f3 and f4. Each round takes as input the current 512-bit block being processed and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of four distinct additive constants Kt. The output of the fourth round i.e. eightieth step is added to the input to the first round to produce CV_{q+1}.

5.) *Output:* After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.

SHA-1 Processing of a Single 512-bit Block
(SHA-1 Compression Function)

49

The behavior of SHA-1 is as follows:

$CV_0 = IV$

$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$

$MD = CV_L$

Where, IV = initial value of ABCDE buffer

$ABCDE_q$ = output of last round of processing of qth message block

L = number of blocks in the message

SUM_{32} = Addition modulo 2^{32}

MD = final message digest value.

SHA-1 Compression Function:

Each round has 20 steps which replaces the 5 buffer words. The logic present in each one of the 80 rounds present is given as

$$(A,B,C,D,E) <\text{-} (E + f(t,B,C,D) + S^5(A) + Wt + Kt), A, S^{30}(B), C, D$$

Where,   A, B, C, D, E = the five words of the buffer

t = step number; 0 < t < 79

f(t,B,C,D) = primitive logical function for step t

S^k = circular left shift of the 32-bit argument by k bits

Wt = a 32-bit word derived from current 512-bit input block.

Kt = an additive constant; four distinct values are used

+ = modulo additon
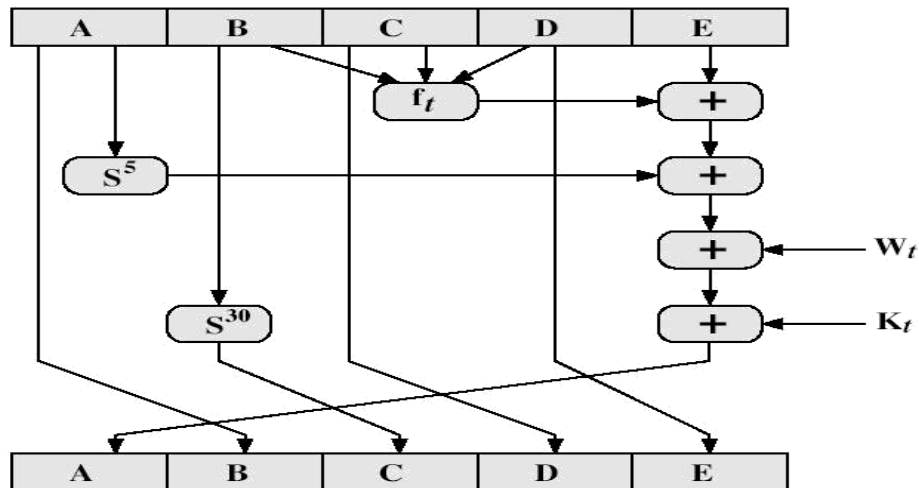
Elementary SHA operation (single step)

SHA shares much in common with MD4/5, but with 20 instead of 16 steps in each of the 4 rounds. Note the 4 constants are based on sqrt(2,3,5,10). Note also that instead of just splitting the input block into 32-bit words and using them directly, SHA-1 shuffles and mixes them using rotates & XOR's to form a more complex input, and greatly increases the difficulty of finding collisions. A sequence of logical functions $f_0, f_1,..., f_{79}$ is used in the SHA-1.

Each f_t, 0<=t<=79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. $f_t(B,C,D)$ is defined as follows: for words B, C, D,

$f_t(B,C,D)$ = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

$f_t(B,C,D)$ = B XOR C XOR D (20 <= t <= 39)

$f_t(B,C,D)$ = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

$f_t(B,C,D)$ = B XOR C XOR D (60 <= t <= 79).

HMAC

Interest in developing a MAC, derived from a cryptographic hash code has been increasing mainly because hash functions are generally faster and are also not limited by export restrictions unlike block ciphers. Additional reason also would be that the library code for cryptographic hash functions is widely available. The original proposal is for incorporation of a secret key into an existing hash algorithm and the approach that received most support is HMAC. HMAC is specified as Internet standard RFC2104. It makes use of the hash function on the given message. Any of MD5, SHA-1, RIPEMD-160 can be used.

HMAC Design Objectives

- To use, without modifications, available hash functions
- To allow for easy replaceability of the embedded hash function
- To preserve the original performance of the hash function
- To use and handle keys in a simple way
- To have a well understood cryptographic analysis of the strength of the MAC based on reasonable assumptions on the embedded hash function

The first two objectives are very important for the acceptability of HMAC. HMAC treats the hash function as a "black box", which has two benefits. First is that an existing implementation of the hash function can be used for implementing HMAC making the bulk of HMAC code readily available without modification. Second is that if ever an existing hash function is to be replaced, the existing hash function module is removed and new module is dropped in. The last design objective provides the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths.

Steps involved in HMAC algorithm:

1. Append zeroes to the left end of K to create a b-bit string K^+ (ex: If K is of length 160-bits and b = 512, then K will be appended with 44 zero bytes).
2. XOR(bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Now apply H to the stream generated in step-3
5. XOR K^+ with opad to produce the b-bit block S_0.
6. Append the hash result from step-4 to S_0.
7. Apply H to the stream generated in step-6 and output the result.

· Define the following terms

H	= embedded hash function
M	= message input to HMAC
Y_i	= i^{th} block of M, $0 \le i \le L - 1$
L	= number of blocks in M
b	= number of bits in a block
n	= length of hash code produced by embedded hash function
K	= secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length $\ge$ n
K^+	= K padded with 0's on the left so that the result is b bits in length
ipad	= 00110110 repeated b/8 times
opad	= 01011100 repeated b/8 times

· Then HMAC can be expressed as

$$\text{HMAC}_K = H[\ (K^+ \oplus \text{opad})\ ||\ H[K^+ \oplus \text{ipad})\ ||\ M]\ ]$$

HMAC Structure:

52
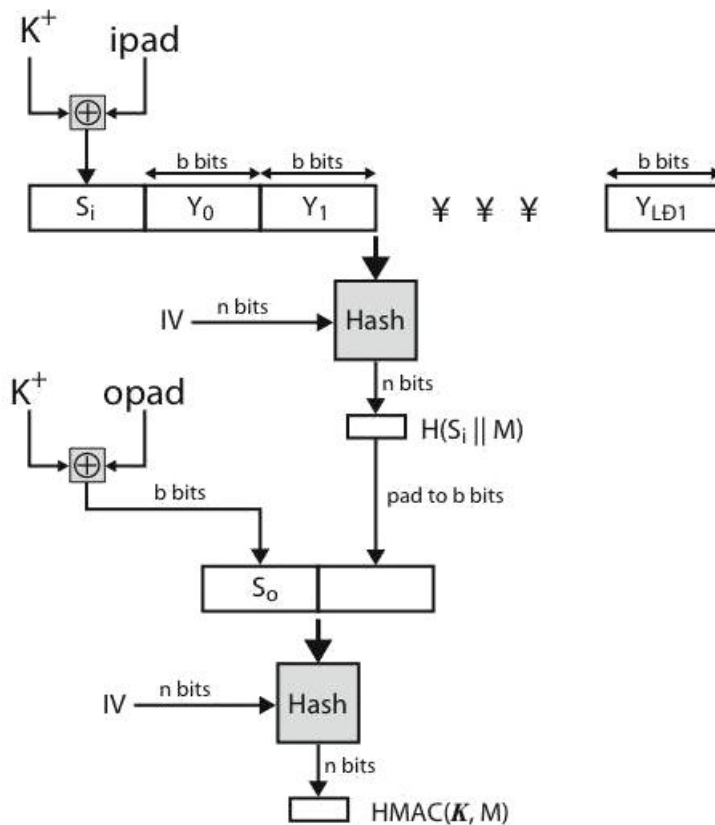
The XOR with ipad results in flipping one-half of the bits of K. Similarly, XOR with opad results in flipping one-half of the bits of K, but different set of bits. By passing S_i and S_0 through the compression function of the hash algorithm, we have pseudorandomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_0, S_i and the block produced from the inner hash)

A more efficient implementation is possible. Two quantities are precomputed. $f(IV, (K^+ \oplus$
$ipad)$
$f(IV, (K^+ \oplus opad)$

where f is the compression function for the hash function which takes as arguments a chaining variable of n bits and a block of b-bits and produces a chaining variable of n bits.
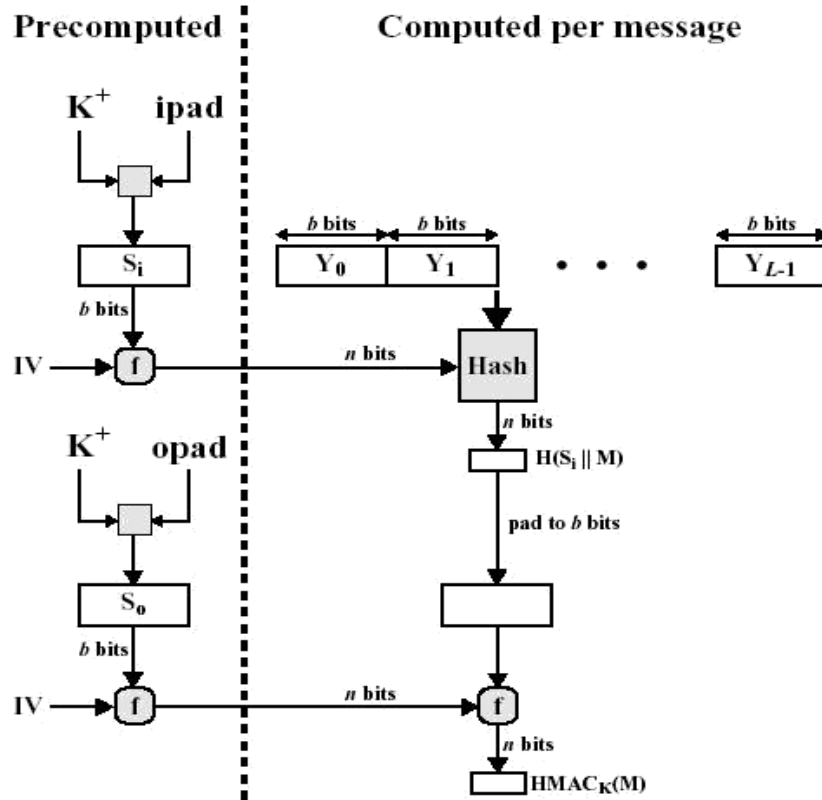
As shown in the above figure, the values are needed to be computed initially and every time a key changes. The precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This implementation is worthwhile if most of the messages for which a MAC is computed are short.

53

Security of HMAC:

The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC. The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key. Have two classes of attacks on the embedded hash function:

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker.
2. The attacker finds collisions in the hash function even when the IV is random and secret.

These attacks are likely to be caused by brute force attack on key used which has work of order 2^n; or a birthday attack which requires work of order $2^{(n/2)}$ - but which requires the attacker to observe 2^n blocks of messages using the same key - very unlikely. So even MD5 is still secure for use in HMAC given these constraints.

Public key cryptography principles, public key cryptography algorithms, digital signatures, digital certificates, certificate authority and key management kerberos, x.509 directory authentication service.

Public Key Cryptography:

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. It is *asymmetric*, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. Public key schemes are neither more nor less secure than private key (security depends on the key size for both). Public-key cryptography *complements rather than replaces* symmetric cryptography. Both also have issues with key distribution, requiring the use of some suitable protocol.

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

1.) *key distribution* – how to have secure communications in general without having to trust a KDC with your key
2.) *digital signatures* – how to verify a message comes intact from the claimed sender

Public-key/two-key/asymmetric cryptography involves the use of **two** keys:

- a *public-key*, which may be known by anybody, and can be used to **encryptmessages**, and **verify signatures**
- a *private-key*, known only to the recipient, used to **decrypt messages**, and **sign**
  (create) **signatures**.
- is **asymmetric** because those who encrypt messages or verify signatures cannot decrypt messages or create signatures

Public-Key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- it is computationally infeasible to find decryption key knowing only algorithm & encryption key
- it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known

The following figure illustrates public-key encryption process and shows that a public-key encryption scheme has six ingredients: plaintext, encryption algorithm, public & private keys, ciphertext& decryption algorithm.
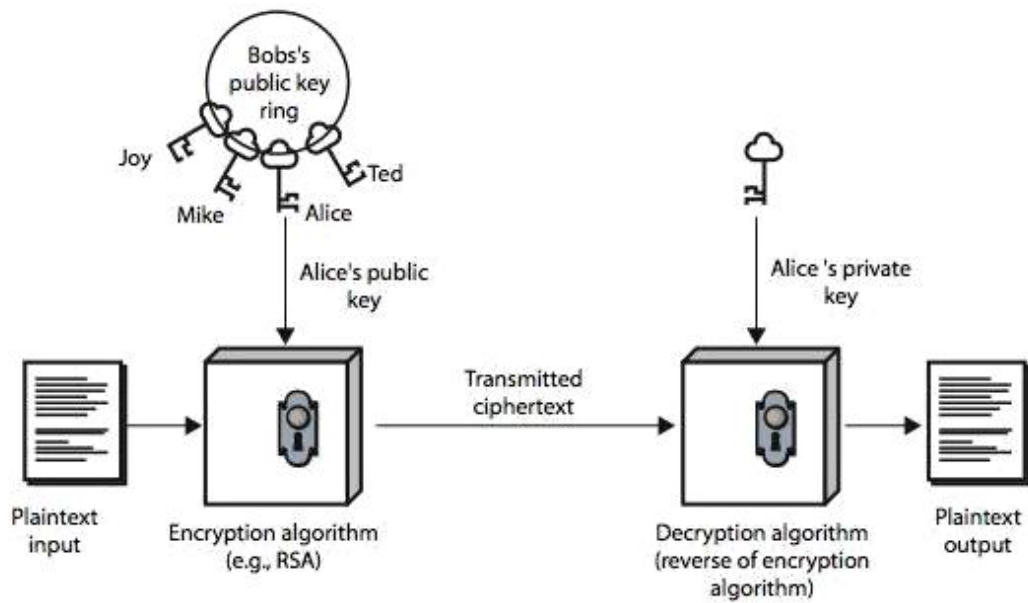
(a) Encryption

The essential steps involved in a public-key encryption scheme are given below:

1.) Each user generates a pair of keys to be used for encryption and decryption.

2.) Each user places one of the two keys in a public register and the other key is kept private.

3.) If B wants to send a confidential message to A, B encrypts the message using A's public key.

4.) When A receives the message, she decrypts it using her private key. Nobody else can decrypt the message because that can only be done using A's private key (Deducing a private key should be infeasible).

5.) If a user wishes to change his keys –generate another pair of keys and publish the public one: no interaction with other users is needed.

Notations used in Public-key cryptography:
- The public key of user A will be denoted **KU$_A$**.
- The private key of user A will be denoted **KR$_A$.**
- Encryption method will be a function E.

- Decryption method will be a function D.
- If B wishes to send a plain message X to A, then he sends the cryptotext Y=E(KU$_A$,X)
- The intended receiver A will decrypt the message: D(KR$_A$,Y)=X

The first attack on Public-key Cryptography is the attack on Authenticity. **An attackermay impersonate user B**: he sends a message E(KU$_A$,X) and claims in the message to be B–Ahas no guarantee this is so. To overcome this, B will encrypt the message using his private key: Y=E(KR$_B$,X). Receiver decrypts using B's public key KR$_B$. This shows the authenticity of the sender because (supposedly) he is the only one who knows the private key. The entire encrypted message serves as a digital signature. This scheme is depicted in the following figure:
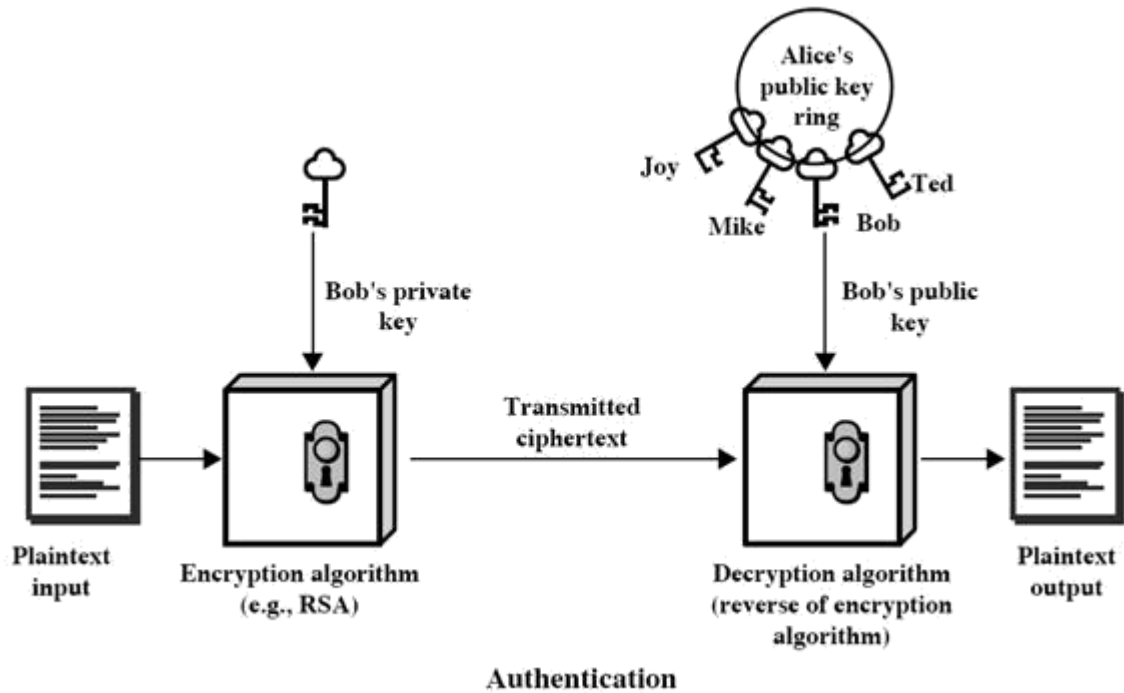
Authentication

But, a drawback still exists. Anybody can decrypt the message using B's public key. So, secrecy or confidentiality is being compromised.

One can provide both *authentication and confidentiality* using the public-key scheme twice:
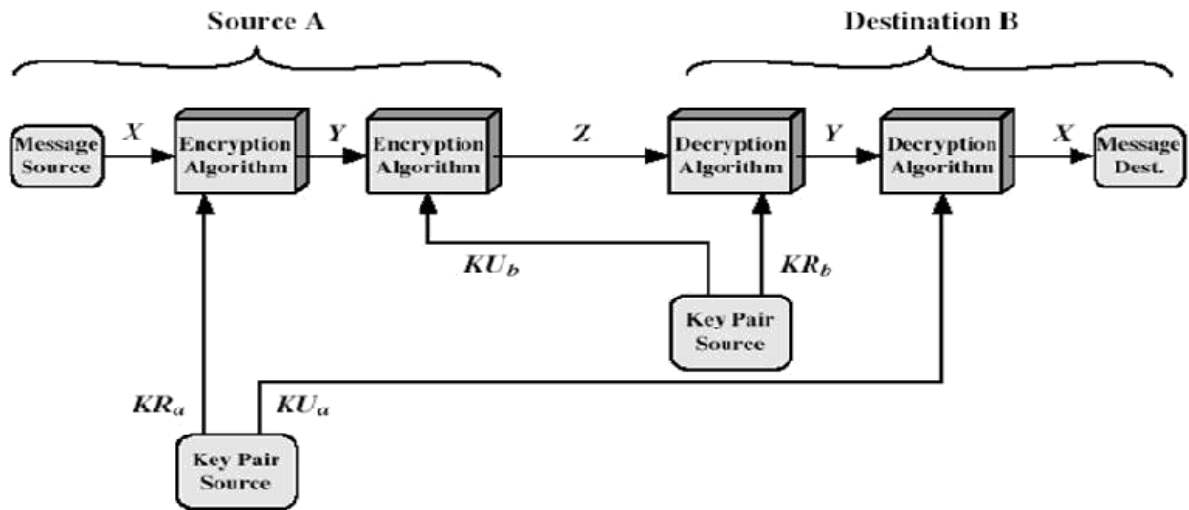
Public-Key Cryptosystem: Secrecy and Authentication

- □ B encrypts X with his private key: $Y=E(KR_B,X)$
- □ B encrypts Y with A's public key: $Z=E(KU_A,Y)$
- □ A will decrypt Z (and she is the only one capable of doing it): $Y=D(KR_A,Z)$
- □ A can now get the plaintext and ensure that it comes from B (he is the only one who knows his private key): decrypt Y using B's public key: $X=E(KU_B,Y)$.

Applications for public-key cryptosystems:

1.) **Encryption/decryption**: sender encrypts the message with the receiver's public key.

2.) **Digital signature**: sender "signs" the message (or a representative part of the message) using his private key

3.) **Key exchange**: two sides cooperate to exchange a secret key for later use in a secret-key cryptosystem.

The main requirements of Public-key cryptography are:

- □ Computationally easy for a party B to generate a pair (public key KUb, private key KR_b).
- □ Easy for sender A to generate ciphertext: $C=E_{KUb}(M)$
- □ Easy for the receiver B to decrypt ciphertect using private key:
$$D_{KRb}(C)=D_{KRb}[E_{KUb}(M)]$$
4. Computationally infeasible to determine private key (KR_b) knowing public key (KU_b)
5. Computationally infeasible to recover message M, knowing KU_b and ciphertext C
6. Either of the two keys can be used for encryption, with the other used for decryption:

$$M \square D_{KRb}[E_{KUb}(M)] \square D_{KUb}[E_{KRb}(M)]$$

Easy is defined to mean a problem that can be solved in polynomial time as a function of input length. A problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. Public-key cryptosystems usually rely on difficult math functions rather than S-P networks as classical cryptosystems. **One-way function** is one, easy to calculate in one direction, infeasible to calculate in the other direction (i.e., the inverse is infeasible to compute). **Trap-door function** is a difficult function that becomes easy if some extra information is known. Our aim to find a *trap-door one-way function*, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.

Security of Public-key schemes:

☐ Like private key schemes brute force **exhaustive search** attack is always theoretically possible. But keys used are too large (>512bits).

☐ Security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems. More generally the **hard** problem is known, its just made too hard to do in practise.

☐ Requires the use of **very large numbers,** hence is **slow** compared to private key schemes

RSA algorithm

RSA is the best known, and by far the most widely used general public key encryption algorithm, and was first published by Rivest, Shamir &Adleman of MIT in 1978 [RIVE78]. Since that time RSA has reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption. The RSA scheme is a block cipher in which the plaintext and the ciphertext are integers between 0 and n-1 for some fixed n and typical size for n is 1024 bits (or 309 decimal digits). It is based on exponentiation in a finite (Galois) field over integers modulo a prime, using large integers (eg. 1024 bits). Its security is due to the cost of factoring large numbers.

RSA involves a public-key and a private-key where the public key is known to all and is used to encrypt data or message. The data or message which has been encrypted using a public key can only be decryted by using its corresponding private-key. Each user generates a key pair i.e. public and private key using the following steps:

- *each user selects two large primes at random - p, q*
- *compute their system modulus n=p.q*
- *calculate $\phi(n)$, where $\phi(n)=(p-1)(q-1)$*
- *selecting at random the encryption key e, where $1<e<\phi(n)$,and **gcd(e,$\phi(n)$)=1***
- *solve following equation to find decryption key d: **e.d=1 mod $\phi(n)$** and $0\leq d\leq n$*
- *publish their public encryption key: **KU={e,n}***
- *keep secret private decryption key: **KR={d,n}***

Both the sender and receiver must know the values of n and e, and only the receiver knows the value of d. Encryption and Decryption are done using the following equations.

To encrypt a message M the sender:
- obtains **public key** of recipient *KU={e,n}*
- computes: **C=M^e mod n**, where $0\leq M<n$

To decrypt the ciphertext C the owner:
- uses their private key *KR={d,n}*
- computes: **M=C^d mod n = (M^e)^d mod n = M^{ed} mod n**

For this algorithm to be satisfactory, the following requirements are to be met.

a) Its possible to find values of e, d, n such that **M^{ed}= M mod n** for all M<n
b) It is relatively easy to calculate M^e and C for all values of M < n.
c) It is impossible to determine d given e and n

The way RSA works is based on Number theory:

Fermat's little theorem: if **p** is prime and **a** is positive integer not divisible by **p**, then

$$a^{p-1}\equiv 1 \bmod p.$$

Corollary: For any positive integer **a** and prime **p**, a$^p\equiv$ a mod p.

Fermat's theorem, as useful as will turn out to be does not provide us with integers d,e we are looking for –Euler's theorem (a refinement of Fermat's) does. Euler's function associates to any positive integer **n,** a number φ(**n**): the number of positive integers smaller than **n** and relatively prime to **n.** For example, φ(**37**) = **36** i.e. φ(**p**) = **p-1** for any prime **p.** For any two primes **p,q**, φ(**pq**)=(**p-1**)(**q-1**).

Euler's theorem: for any relatively prime integers a,n we have $a^{\varphi(n)} \equiv 1 \bmod n$.
Corollary: For any integers a,n we have $a^{\varphi(n)+1} \equiv a \bmod n$
Corollary: Let p,q be two odd primes and n=pq. Then:

$$\varphi(n)=(p-1)(q-1)$$

For any integer m with 0<m<n, $m^{(p-1)(q-1)+1} \equiv m \bmod n$ For any
integers k,m with 0<m<n, $m^{k(p-1)(q-1)+1} \equiv m \bmod n$

Euler's theorem provides us the numbers d, e such that $M^{ed}=M \bmod n$. We have to choose d,e such that ed=kφ(n)+1, or equivalently, $d \equiv e^{-1} \bmod \varphi(n)$

An example of RSA can be given as,
- Select primes: *p*=17 &*q*=11
- Compute $n = pq$ =17×11=187
- Compute ø(*n*)=(*p*–1)(*q*-1)=16×10=160
- Select e *:*gcd(e,160)=1; choose *e*=7
- Determine d*: de*=1 mod 160 and *d*< 160 Value is d=23 since 23×7=161= 10×160+1
- Publish public key KU={7,187}
- Keep secret private key KR={23,187}
- Now, given message M = 88 (nb. 88<187)
- encryption: C = 88^7 mod 187 = 11
- decryption: M = 11^{23} mod 187 = 88

Another example of RSA is given as,
Let **p = 11, q = 13, e = 11, m = 7**

n = pqi.e. n= 11*13 = 143

ø(n)= (p-1)(q-1) i.e. (11-1)(13-1) = 120

e.d=1 mod ø(n) i.e. 11d mod 120 = 1 i.e. (11*11) mod 120=1; so d = 11public key
:{11,143} and private key: {11,143}
C=M^e mod n, so ciphertext = 7^{11}mod143 = 727833 mod 143; i.e. C = 106
M=C^d mod n, plaintext = 106^{11}mod 143 = 1008 mod 143; i.e. M = 7

Another example is:
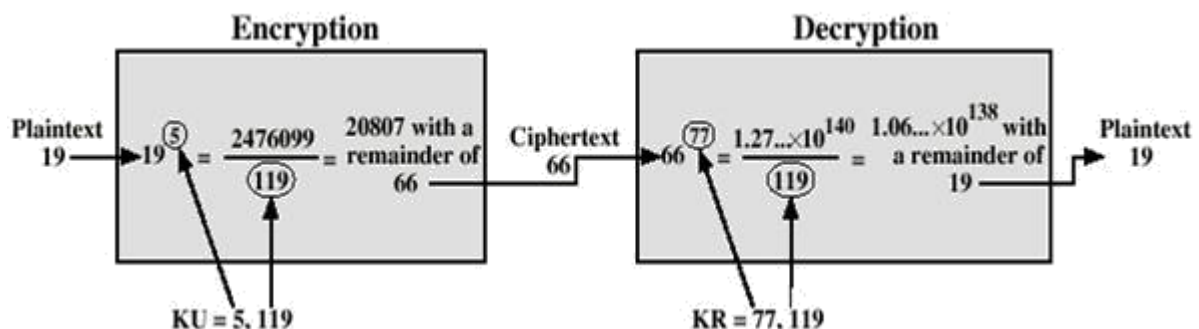
For RSA key generation,

▢ users of RSA must:
  – determine two primes at random - p, q
  – select either e or d and compute the other

▢ primes p,q must not be easily derived from modulus N=p.q
  – means must be sufficiently large
  – typically guess and use probabilistic test

▢ exponents e, d are inverses, so use Inverse algorithm to compute the other

Security of RSA

There are three main approaches of attacking RSA algorithm.

Brute force key search(infeasible given size of numbers)

As explained before, involves trying all possible private keys. Best defence is using large keys.

Mathematical attacks(based on difficulty of computing ø(N), by factoring modulus N)

There are several approaches, all equivalent in effect to factoring the product of two

primes.   Some of them are given as:
  – factor N=p.q, hence find ø(N) and then d
  – determine ø(N) directly and find d
  – find d directly

The possible defense would be using large keys and also choosing large numbers for p and q, which should differ only by a few bits and are also on the order of magnitude 10^{75} to 10^{100}. And gcd (p-1, q-1) should be small.

<u>Timing attacks</u>(on running of decryption)

These attacks involve determination of a private key by keeping track of how long a computer takes to decipher a message (ciphertext-only attack) –this is essentially an attack on the fast exponentiation algorithm but can be adapted for any other algorithm. Though these attacks are a quite serious threat, there are some simple countermeasures that can be used. They are explained below:

Constant exponentiation time: Ensure that all exponentiations take the same time beforereturning a result: degrade performance of the algorithm.

Random Delay: A random delay can be added to exponentiation algorithm to confuse thetiming attack. If there is not enough noise added by defenders, the attackers can succeed.

Blinding: Multiply the ciphertext by a**random**number before performing exponentiation–in this way the attacker does not know the input to the exponentiation algorithm.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C^d$ mod n is implemented as follows:

- Generate a secret random number r between 0 and n-1
- Compute $C' = C(r^e)$ mod n where e is the public exponent
- Compute $M' = (C'^d)$ mod n with the ordinary exponentiation
- Compute $M = M' r^{-1}$ mod n
- Reported performance penalty: 2 to 10%

Key Management

One of the major roles of public-key encryption has been to address the problem of key distribution. Two distinct aspects to use of public key encryption are present.

- The distribution of public keys.
- Use of public-key encryption to distribute secret keys.

Distribution of Public Keys
The most general schemes for distribution of public keys are given below

PUBLIC ANNOUNCEMENT OF PUBLIC KEYS

Here any participant can send his or her public key to any other participant or broadcast the key to the community at large. For example, many PGP users have adopted the practice of appending their public key to messages that they send to public forums.

Uncontrolled Public-Key Distribution
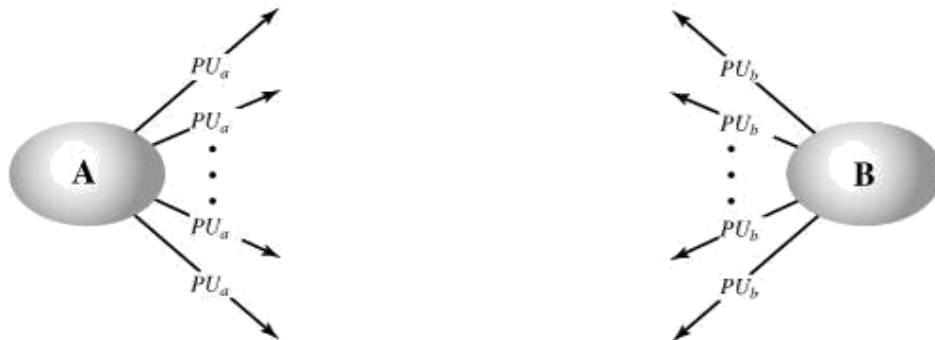
Though this approach seems convenient, it has a major drawback. Anyone can forge such a public announcement. Some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until the time when A discovers about the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

PUBLICLY AVAILABLE DIRECTORY

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. It includes the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.

2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.

Public-Key Publication
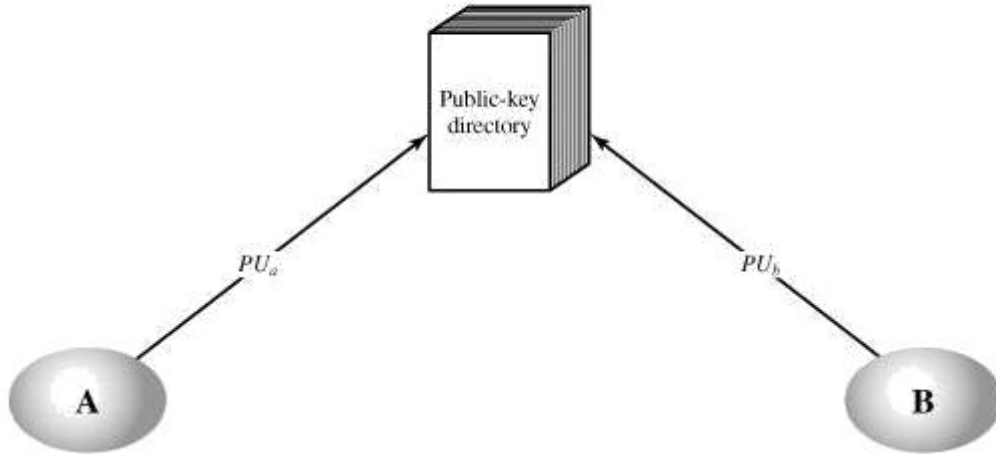
3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.

4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme has still got some vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Or else, the adversary may tamper with the records kept by the authority.

PUBLIC-KEY AUTHORITY

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. This scenario assumes the existence of a public authority (whoever that may be) that maintains a dynamic directory of public keys of all users. The public authority has its own (private key, public key) that it is using to communicate to users. Each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key.

For example, consider that Alice and Bob wish to communicate with each other and the following steps take place and are also shown in the figure below:
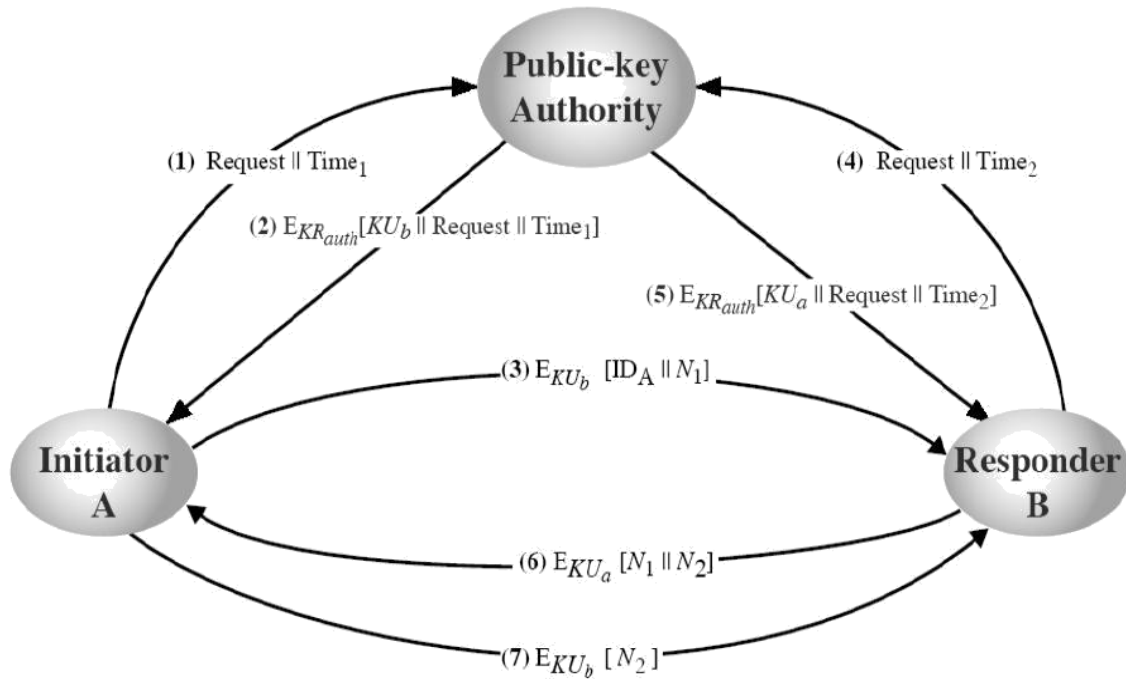
1.) Alice sends a **timestamped** message to the central authority with a request for Bob's public key (the time stamp is to mark the moment of the request)

2.) The authority sends back a message encrypted with its private key (for authentication) –message contains Bob's public key and the original message of Alice

   –this way Alice knows this is not a reply to an old request;

3.) Alice starts the communication to Bob by sending him an encrypted message containing her identity ID_A and a nonce N_1 (to identify uniquely this transaction)

4.) Bob requests Alice's public key in the same way (step 1)

5.) Bob acquires Alice's public key in the same way as Alice did. (Step-2)

6.) Bob replies to Alice by sending an encrypted message with N_1 plus a new generated nonce N_2 (to identify uniquely the transaction)

7.) Alice replies once more encrypting Bob's nonce N_2 to assure bob that its correspondent is Alice

Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

PUBLIC-KEY CERTIFICATES

The above technique looks attractive, but still has some drawbacks. For any communication between any two users, the central authority must be consulted by both users to get the newest public keys i.e. the central authority must be online 24 hours/day. If the central authority goes offline, all secure communications get to a halt. This clearly leads to an undesirable bottleneck.

A further improvement is to use certificates, which can be used to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA). A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority.

This certificate issuing scheme does have the following requirements:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.
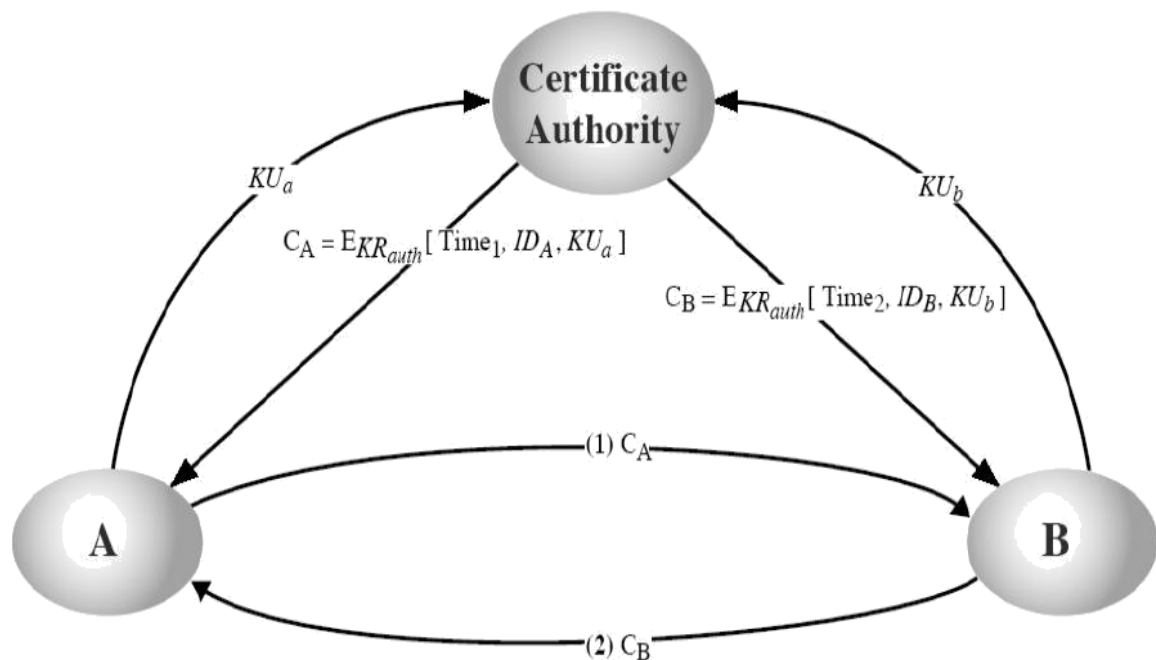
Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

$$C_A = E(PR_{auth}, [T||ID_A||PU_a])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T||ID_A||PU_a])) = (T||ID_A||PU_a)$$

The recipient uses the authority's public key, PU_{auth} to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages. In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

Public Key Distribution of Secret Keys

Public-key encryption is usually viewed as a vehicle for the distribution of secret keys to be used for conventional encryption and the main reason for this is the relatively slow data rates associated with public-key encryption.

Simple Secret Key Distribution:

If A wishes to communicate with B, the following procedure is employed:
1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

In this case, if an adversary, E, has control of the intervening communication channel, then E

can compromise the communication in the following fashion without being detected:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e \| ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message, and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to E. A and B can now exchange messages using K_s E no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s E can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

It is assumed that A and B have exchanged public keys by one of the schemes described earlier. Then the following steps occur:
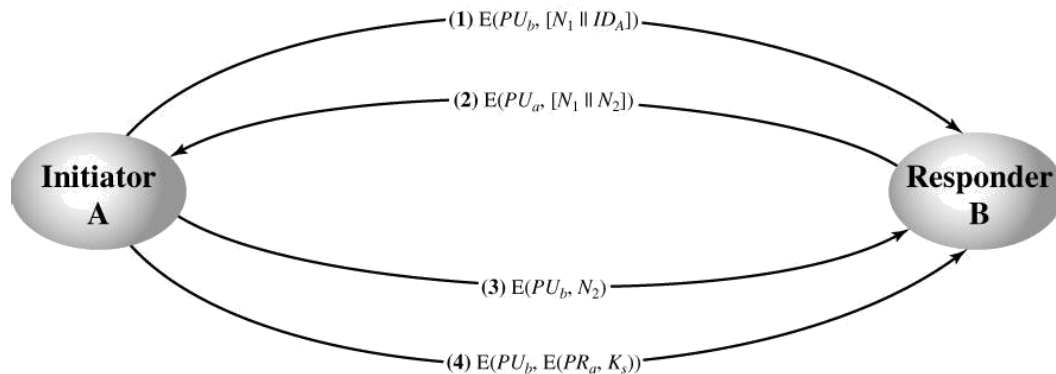
1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2) Because only B could have decrypted message (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = E(PU_b, E(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $D(PU_a, D(PR_b, M))$ to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

Diffie-Hellman Key Exchange

Diffie-Hellman key exchange (**D-H**) is a cryptographic protocol that allows two parties thathave no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The D-H algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

First, a primitive root of a prime number p, can be defined as one whose powers generate all the integers from 1 to p-1. If a is a primitive root of the prime number p, then the

numbers, *a mod p, a²mod p,..., aᵖ⁻¹mod p*, are distinct and consist of the integers from 1 through p 1 in some permutation.

For any integer **b** and a primitive root **a** of prime number **p**, we can find a unique exponent **i**

such that   .The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as ***dlog$_{a,p}$(b).*** The algorithm is summarized below:

<table>
<tr><td>Global Public Elements</td></tr>
<tr><td>q</td><td>prime number</td></tr>
<tr><td>α</td><td>$\alpha < q$ and α a primitive root of q</td></tr>
</table>

<table>
<tr><td colspan="2">User A Key Generation</td></tr>
<tr><td>Select private X_A</td><td>$X_A < q$</td></tr>
<tr><td>Calculate public Y_A</td><td>$Y_A = \alpha^{X_A} \bmod q$</td></tr>
</table>

<table>
<tr><td colspan="2">User B Key Generation</td></tr>
<tr><td>Select private X_B</td><td>$X_B < q$</td></tr>
<tr><td>Calculate public Y_B</td><td>$Y_B = \alpha^{X_B} \bmod q$</td></tr>
</table>

<table>
<tr><td>Generation of Secret Key by User A</td></tr>
<tr><td>$K = (Y_B)^{X_A} \bmod q$</td></tr>
</table>

<table>
<tr><td>Generation of Secret Key by User B</td></tr>
<tr><td>$K = (Y_A)^{X_B} \bmod q$</td></tr>
</table>
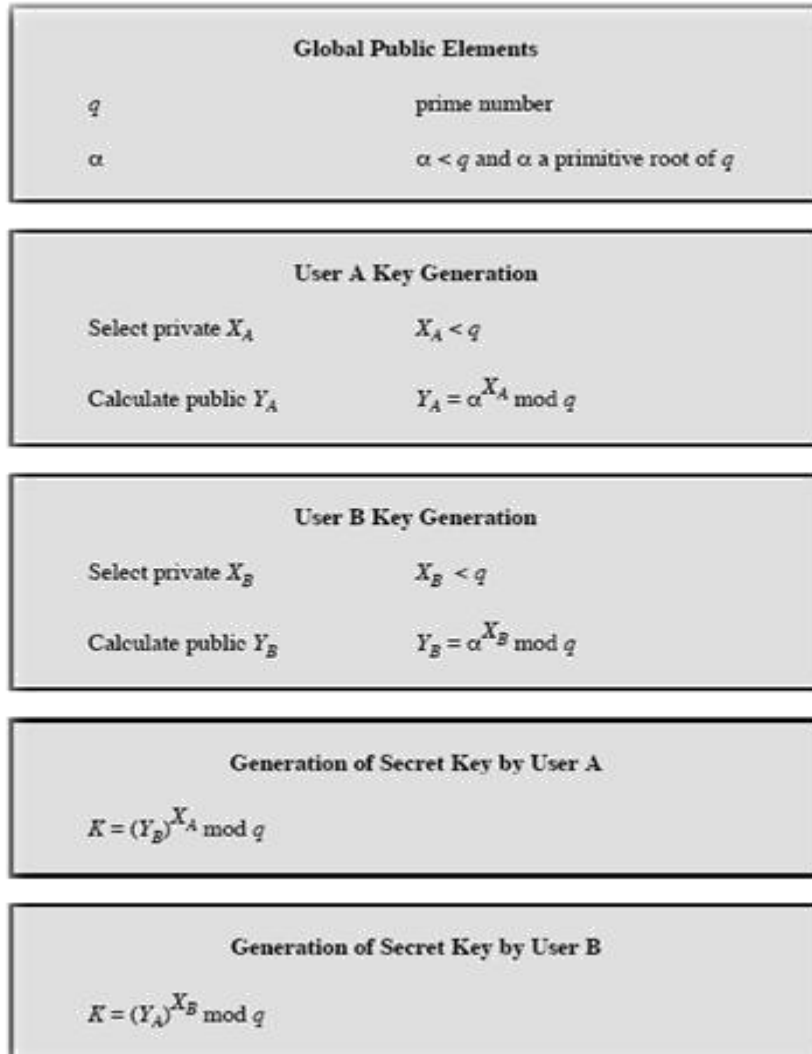
For this scheme, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_A < q$ and computes $Y_B = \alpha^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{XA} \bmod q$ and user B computes the key as $K = (Y_A)^{XB} \bmod q$. These two calculations produce identical results.

Discrete Log Problem

The (discrete) exponentiation problem is as follows: Given a base a, an exponent b and a modulus p, calculate c such that $a^b \equiv c \pmod{p}$ and $0 \leq c < p$. It turns out that this problem is fairly easy and can be calculated "quickly" using fast-exponentiation.

The discrete log problem is the inverse problem:

Given a base a, a result c ($0 \leq c < p$) and a modulus p, calculate the exponent b such that
$$a^b \equiv c \pmod{p}.$$
It turns out that no one has found a quick way to solve this problem

With DLP, if P had 300 digits, X_a and X_b have more than 100 digits, it would take longer than the life of the universe to crack the method.

Examples for D-H key distribution scheme:

1) Let p = 37 and g = 13.

Let Alice pick a = 10. Alice calculates 13^{10} (mod 37) which is 4 and sends that to Bob. Let Bob pick b = 7. Bob calculates 13^7 (mod 37) which is 32 and sends that to Alice.
(Note: 6 and 7 are secret to Alice and Bob, respectively, but both 4 and 32 are known by all.)

- ☐ Alice receives 32 and calculates 32^{10} (mod 37) which is 30, the secret key.
- ☐ Bob receives 4 and calculates 4^7 (mod 37) which is 30, the same secret key.

2) Let p = 47 and g = 5.

Let Alice pick a = 18. Alice calculates 5^{18} (mod 47) which is 2 and sends that to Bob. Let Bob pick b = 22. Bob calculates 5^{22} (mod 47) which is 28 and sends that to Alice.

- ☐ Alice receives 28 and calculates 28^{18} (mod 47) which is 24, the secret key.
- ☐ Bob receives 2 and calculates 2^{22} (mod 47) which is 24, the same secret key

Man-in-the-Middle Attack on D-H protocol

Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack

proceeds as follows:

- ☐ Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
- ☐ Alice transmits Y_A to Bob.
- ☐ Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2 = $(Y_A)^{X_{D2}}$ mod q.
- ☐ Bob receives Y_{D1} and calculates K1 = $(Y_{D1})^{X_E}$ mod q.
- ☐ Bob transmits X_A to Alice.
- ☐ Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1 = $(Y_B)^{X_{D1}}$ mod q.
- ☐ Alice receives Y_{D2} and calculates K2 = $(Y_{D2})^{X_A}$ mod q.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2. All future communication between Bob and Alice is compromised in the following way:

- Alice sends an encrypted message M: E(K2, M).
- Darth intercepts the encrypted message and decrypts it, to recover M.
- Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

Digital Signature

The most important development from the work on public-key cryptography is the digital signature. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties:
- It must verify the author and the date and time of the signature
- It must to authenticate the contents at the time of the signature
- It must be verifiable by third parties, to resolve disputes

Thus, the digital signature function includes the authentication function. A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire

message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost/stolen and signatures forged. Need time-stamps and timely key revocation.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public-key algorithms, and the arbiter may or may not see the actual message contents.

Using Conventional encryption

$X \rightarrow A : M \| E (K_{xa}, [ID_x \| H (M)])$

$A \rightarrow Y : E(K_{ay} ,[ID_x \| M \| E (K_{xa} ,[ID_x \| H(M)]) \| T])$

2. It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay}. X constructs a message M and computes its hash value

   H(m) . Then X transmits the message plus a signature to A. the signature consists of an identifier ID_x of X plus the hash value, all encrypted using K_{xa}.

3. A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with K_{ay}. The message includes ID_x, the original message from X, the signature, and a timestamp.

4. Arbiter sees message

5. Problem : the arbiter could form an alliance with sender to deny a signed message, or with the receiver to forge the sender's signature.

Using Public Key Encryption

$\rightarrow A : ID_x \| E(PR_x, [ID_x \| E (PU_y, E(PR_x, M))])$

$A \rightarrow Y : E(PR_a, [ID_x \| E (PU_y, E (PR_x, M))\| T])$
B

☐ X double encrypts a message M first with X's private key, PR_x, and then with Y's public key, PU_y. This is a signed, secret version of the message. This signed message, together with X's identifier , is encrypted again with PR_x and, together with ID_x, is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else except Y)

☐ A can decrypt the outer encryption to assure that the message must have come from

X (because only X has PR_x). Then A transmits a message to Y, encrypted with PR_a. The message includes ID_x, the double encrypted message, and a timestamp. ☐ Arbiter does not see message

Digital Signature Standard (DSS)

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS uses an algorithm that is designed to provide only the digital signature function and cannot be used for encryption or key exchange, unlike RSA.

The RSA approach is shown below. The message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted.
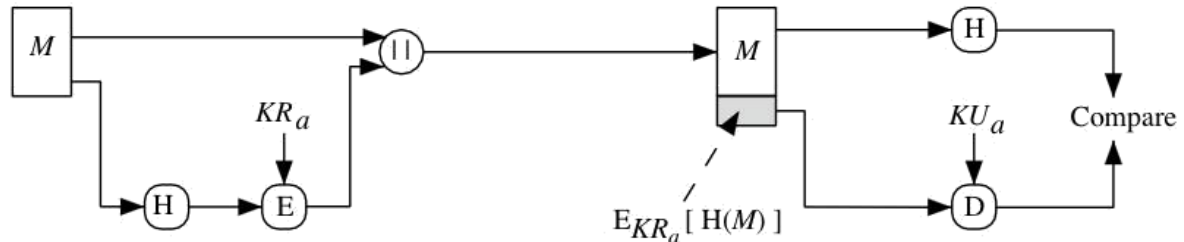
The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G). The result is a signature consisting of two components, labeled s and r.
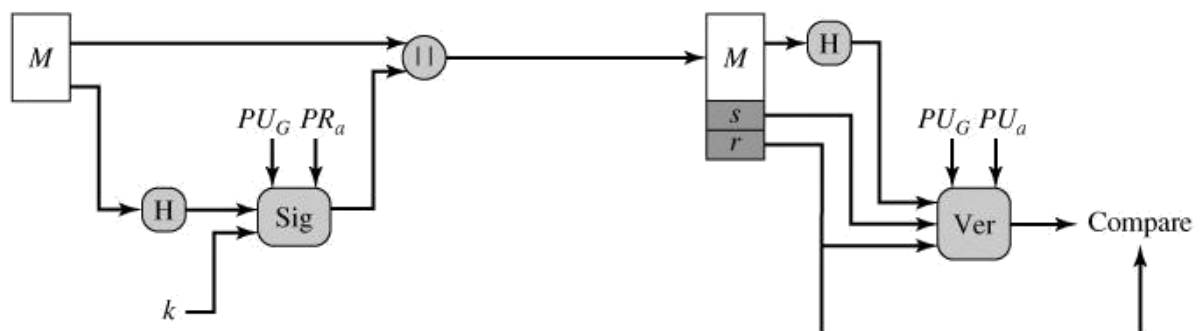
(b) DSS approach

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's

private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos listed the following requirements:

- Secure: A network eavesdropper should not be able to obtain the necessaryinformation to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- Reliable: For all services that rely on Kerberos for access control, lack of availability of

    the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- Transparent: Ideally, the user should not be aware that authentication is taking place,beyond the requirement to enter a password.
- Scalable:The system should be capable of supporting large numbers of clients andservers. This suggests a modular, distributed architecture

Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Version 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

Kerberos Version 4

1.) **Simple dialogue**:
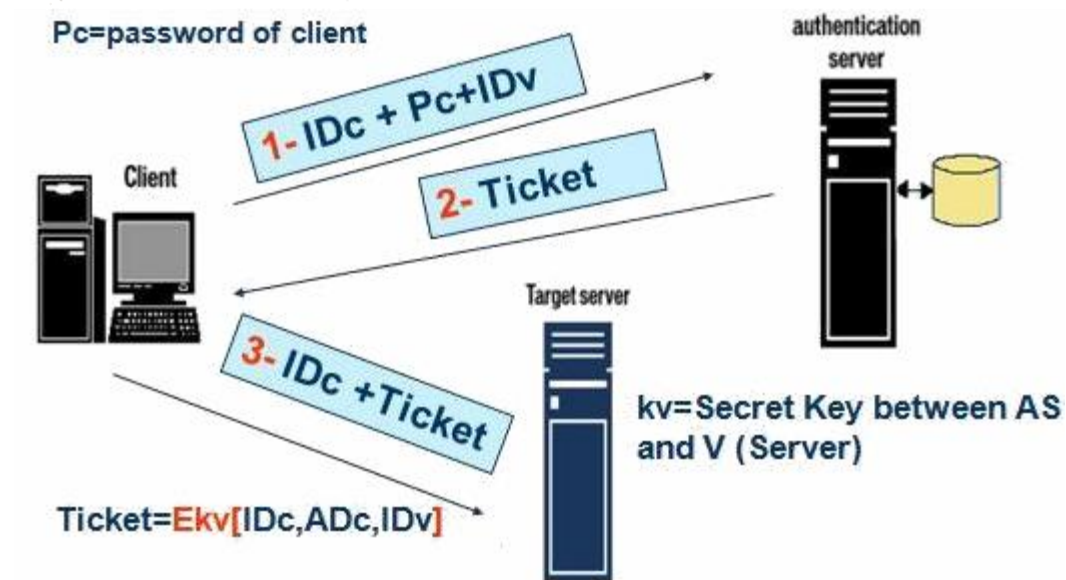
More Secure Dialogue

The Version 4 Authentication Dialogue

The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.

$$(1)\ C \rightarrow AS\quad ID_c \parallel ID_{tgs} \parallel TS_1$$

$$(2)\ AS \rightarrow C\quad E(K_c, [K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}])$$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

$$(3)\ C \rightarrow TGS\quad ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$$

$$(4)\ TGS \rightarrow C\quad E(K_{c,tgs}, [K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v])$$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$$
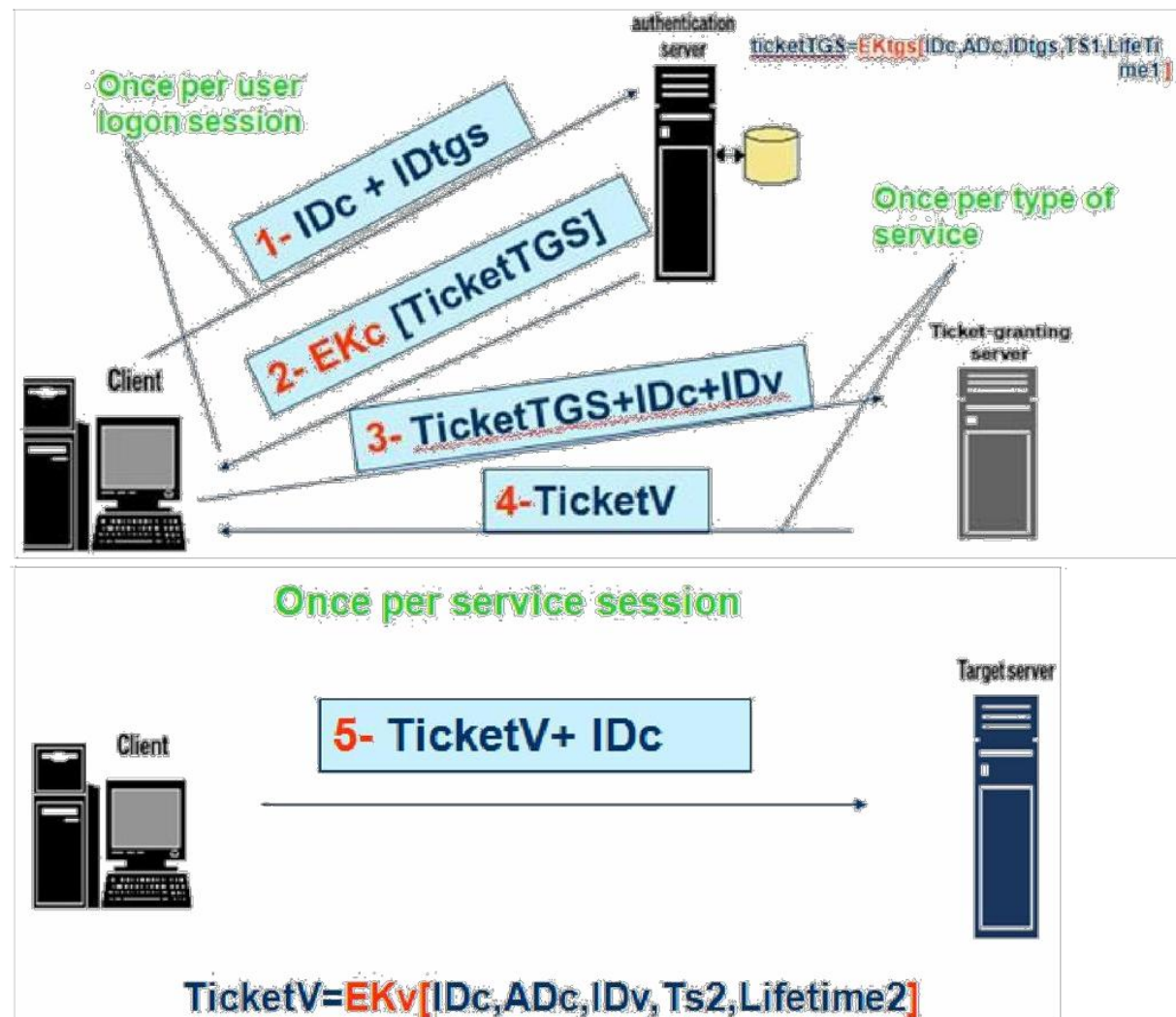
$$Authenticator_c = E(K_{c,tgs}, [ID_C \parallel AD_C \parallel TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

$$(5)\ C \rightarrow V\quad Ticket_v \parallel Authenticator_c$$

$$(6)\ V \rightarrow C\quad E(K_{c,v}, [TS_5 + 1])\,(\text{for mutual authentication})$$

$$Ticket_v = E(K_v, [K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \parallel AD_C \parallel TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

There is a problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information.

Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator.

The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6).

Overview of Kerberos
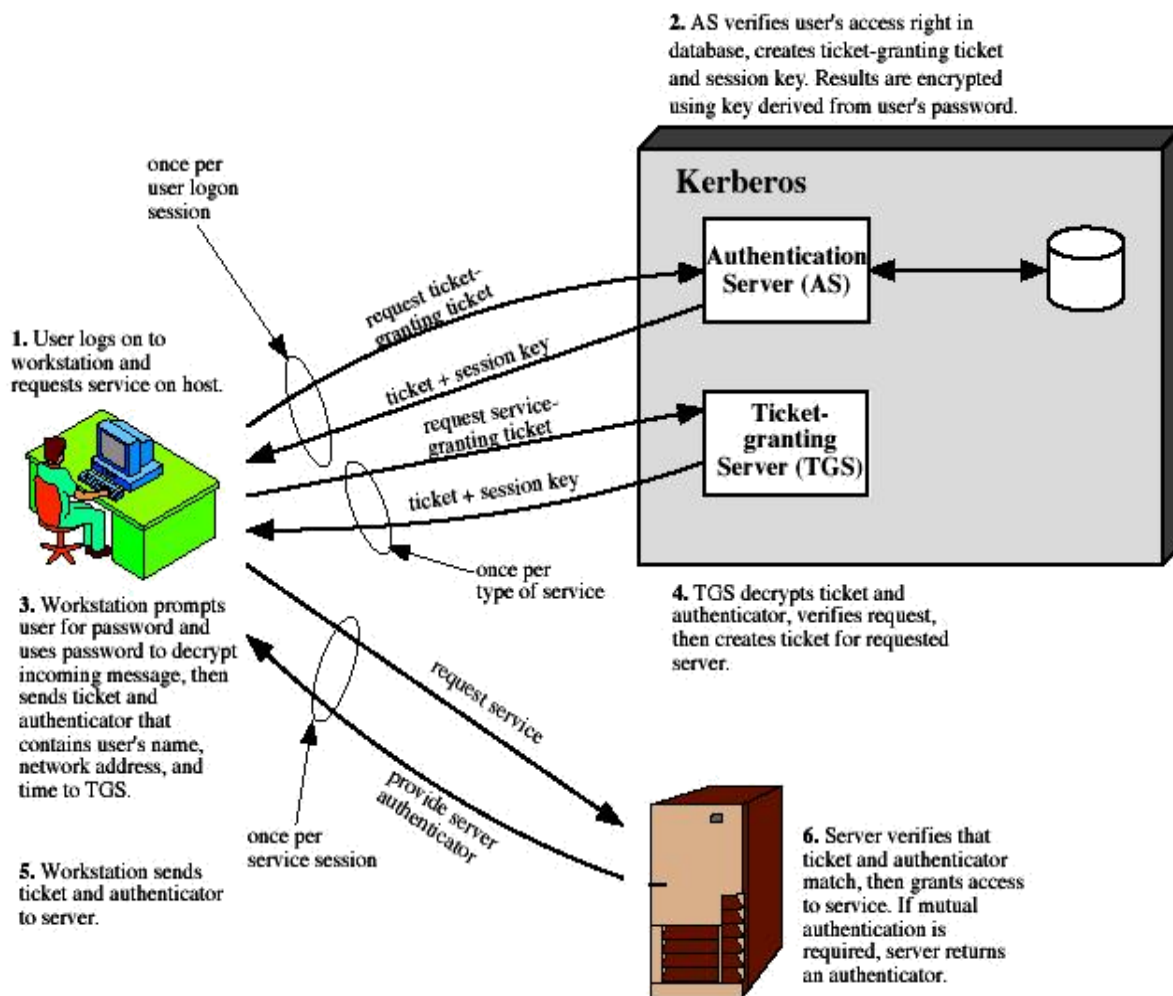
Kerberos Realms

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share keys and trust each other.

The following figure shows the authentication messages where service is being requested from another domain. The ticket presented to the remote server indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request. One problem presented by the foregoing approach is that it does not scale well to many realms, as each pair of realms need to share a key.

Request for Service in another realm:
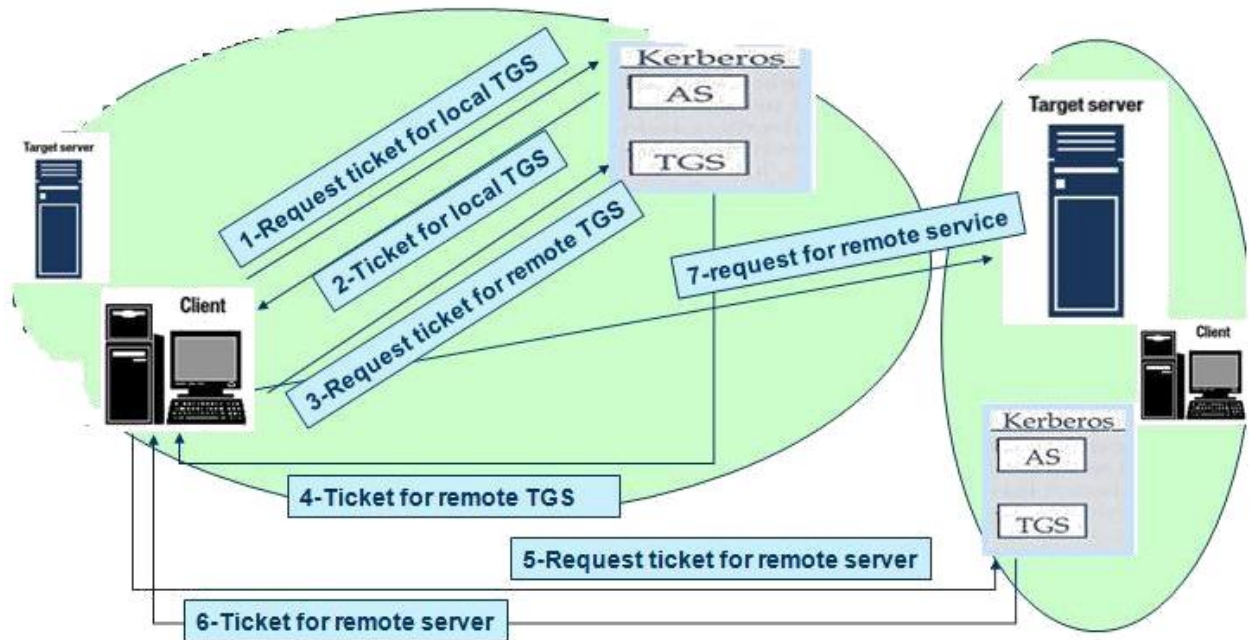
The limitations of Kerberos version-4 are categorised into two types:
- Environmental shortcomings of Version 4:
  - Encryption system dependence: DES
  - Internet protocol dependence
  - Ticket lifetime
  - Authentication forwarding
  - Inter-realm authentication
- Technical deficiencies of Version 4:
  - Double encryption
  - Session Keys
  - Password attack

Kerberos version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:
- Realm: Indicates realm of the user
- Options
- Times
  - From: the desired start time for the ticket
  - Till: the requested expiration time
  - Rtime: requested renew-till time
- Nonce: A random value to assure the response is fresh

The basic Kerberos version 5 authentication dialogue is shown here First, consider the **authentication service exchange.**

(1) C → AS $Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$

(2) AS → C $Realm_c \| ID_C \| Ticket_{tgs} \| E(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $Options \| ID_v \| Times \| \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $Realm_c \| ID_C \| Ticket_v \| E(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Options \| Ticket_v \| Authenticator_c$

(6) V → C $E_{K_{c,v}} [TS_2 \| Subkey \| Seq\#]$

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

Message (1) is a client request for a ticket-granting ticket.

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS.

Now compare the **ticket-granting service** exchange for versions 4 and 5. See that message

(3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS. Finally, for the client/server authentication exchange, several new features appear in version 5, such as a request for mutual authentication. If required, the server responds with message (6) that includes the timestamp from the authenticator. The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

Advantages of Kerberos:

- User's passwords are never sent across the network, encrypted or in plain text
- Secret keys are *only* passed across the network in encrypted form
- Client and server systems mutually authenticate
- It limits the duration of their users' authentication.
- Authentications are **reusable** and **durable**
- Kerberos has been scrutinized by many of the top programmers, cryptologists and security experts in the industry

X.509 Authentication Service

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 is based on the use of **public-key cryptography** and **digital signatures**.

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

(b) Certificate Revocation List
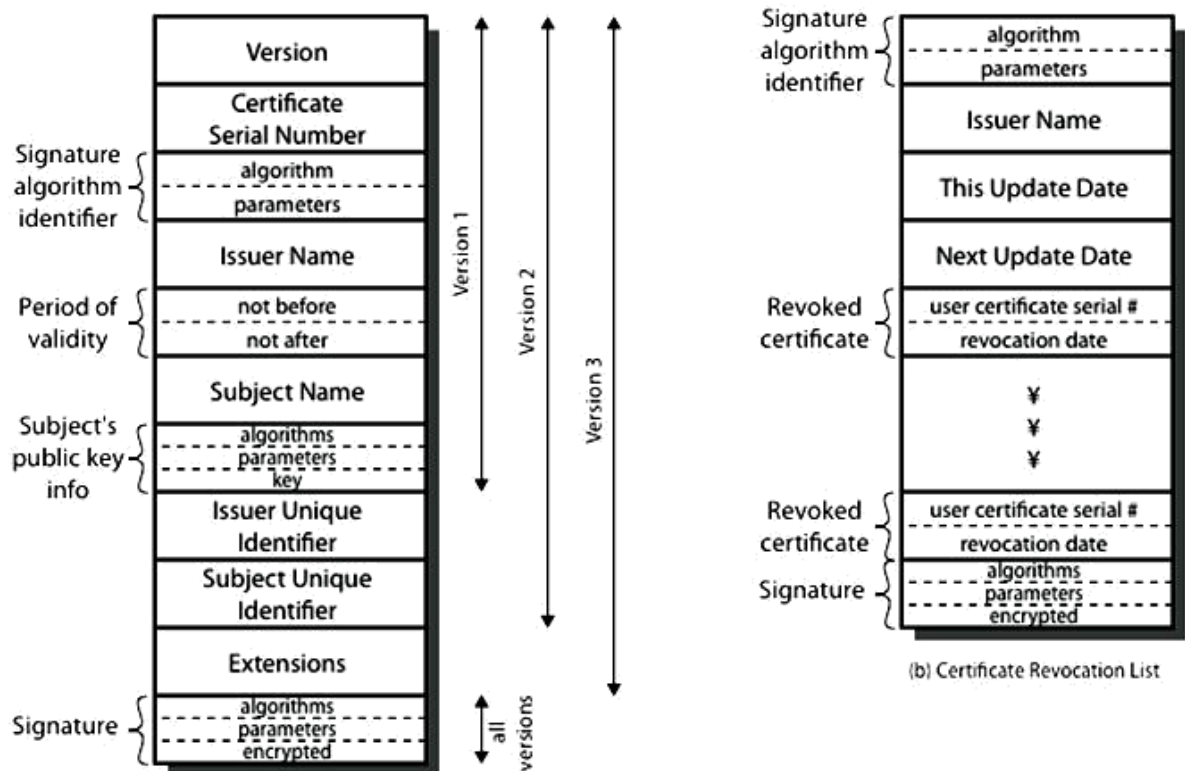
The general format of a certificate is shown above, which includes the following elements:

- version 1, 2, or 3
- serial number (unique within CA) identifying certificate
- signature algorithm identifier
- issuer X.500 name (CA)
- period of validity (from - to dates)
- subject X.500 name (name of owner)
- subject public-key info (algorithm, parameters, key)
- issuer unique identifier (v2+)
- subject unique identifier (v2+)
- extension fields (v3)
- signature (of hash of all fields in certificate)

The standard uses the following notation to define a certificate:

$$CA<<AW>> = CA \{V, SN, AI, CA, T_A, A, Ap\}$$

Where,

$Y <<X>>$ = the certificate of user X issued by certification authority Y

$Y \{I\}$ = the signing of **I** by **Y**. It consists of **I** with an encrypted Hash code appended

User certificates generated by a CA have the following characteristics:
- Any user with CA's public key can verify the user public key that was certified
- No party other than the CA can modify the certificate without being detected
- because they cannot be **forged**, certificates can be placed in a public directory

Scenario: Obtaining a User Certificate

If both users share a common CA then they are assumed to know its public key. Otherwise CA's must form a hierarchy and use certificates linking members of hierarchy to validate other CA's. Each CA has certificates for clients (forward) and parent (backward). Each client trusts parents certificates. It enables verification of any certificate from one CA by users of all other CAs in hierarchy.

A has obtained a certificate from the CA X1. B has obtained a certificate from the CA X2. A can read the B's certificate but cannot verify it. In order to solve the problem ,the Solution:

X1<<X2> X2<<B>>. A obtain the certificate of X2 signed by X1 from directory. obtain X2's public key. A goes back to directory and obtain the certificate of B signed by X2. obtain
B's public key securely. The directory entry for each CA includes two types of certificates: Forward certificates: Certificates of X generated by other CAs
Reverse certificates: Certificates generated by X that are the certificates of other CAs

31

X.509 CA Hierarchy
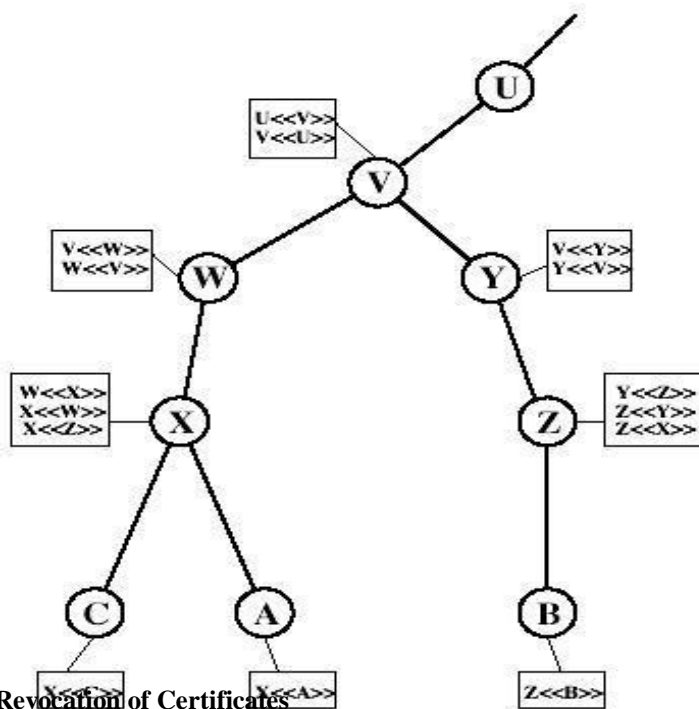
A acquires B certificate using chain:
X<<W>>W<<V>>V<<Y>>Y<< Z>> Z<<B>>

B acquires A certificate using chain:
Z<<Y>>Y<<V>>V<<W>>W<< X>> X<<A>>

U<<V>>
V<<U>>

V<<W>>
W<<V>>

V<<Y>>
Y<<V>>

W<<X>>
X<<W>>
X<<Z>>

Y<<Z>>
Z<<Y>>
Z<<X>>

X<<C>>   X<<A>>

Z<<B>>

Revocation of Certificates

Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- The user's private key is assumed to be compromised.
- The user is no longer certified by this CA.
- The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each **certificate revocation list(CRL)posted** to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

Authentication Procedures

X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side.

1. One-WayAuthentication: One way authentication involves a single transfer ofinformation from one user (A) to another (B), and establishes the details shown above. Note that only the identity of the initiating entity is verified in this process, not that of the responding entity. At a minimum, the message includes a timestamp ,a nonce, and the identity of B and is signed with A's private key. The message may also include information to be conveyed, such as a session key for B.

- 1 message (A->B) used to establish
    - the identity of A and that message is from A
    - message was intended for B
    - integrity & originality of message

A   1-A {ta,ra,B,sgnData,KUb[Kab]}   B

Ta-timestamp rA=nonce B =identity
sgnData=signed with A's private key

2. Two-Way Authentication:Two-way authentication thus permits both parties in acommunication to verify the identity of the other, thus additionally establishing the above details. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B, and possible additional information for A.

- 2 messages (A->B, B->A) which also establishes in addition:
    - the identity of B and that reply is from B
    - that reply is intended for A
    - integrity & originality of reply

A   1-A {ta,ra,B,sgnData,KUb[Kab]}
   2-B {tb,rb,A,sgnData,KUa[Kab]}   B

3. Three-Way Authentication: Three-Way Authentication includes a final message from A to B, which contains a signed copy of the nonce, so that timestamps need not be checked, for use when synchronized clocks are not available.

- # 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks


```
1- A {ta,ra,B,sgnData,KUb[Kab]}
2 -B {tb,rb,A,sgnData,KUa[Kab]}
3- A{rb}
```

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed.

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.

2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.

3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.

4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.

5. It is important to be able to identify different keys used by the same owner at different

   times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. X.509 version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored or not.

UNIT-IV

Email privacy: pretty good privacy (pgp) and s/mime

PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidentiality services.

The Pretty Good Privacy (PGP) secure email program, is a remarkable phenomenon, has grown explosively and is now widely used. Largely the effort of a single person, Phil Zimmermann, who selected the best available crypto algorithms to use & integrated them into a single program, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. It is independent of government organizations and runs on a wide range of systems, in both free & commercial versions.

*There are **five** important services in PGP*

- ☐ *Authentication (Sign/Verify)*
- ☐ *Confidentiality (Encryption/Decryption)*
- ☐ *Compression*
- ☐ *Email compatibility*
- ☐ *Segmentation and Reassembly*
- ☐ ☐The last three are☐**transparent**☐to the user

PGP Notations:

Ks   =session key used in symmetric encryption scheme

PRa =private key of user A, used in public-key encryption scheme

PUa =public key of user A, used in public-key encryption scheme

EP   = public-key encryption

DP = public-key decryption

EC = symmetric encryption

DC = symmetric decryption

H   = hash function

☐   = concatenation

Z = compression using ZIP algorithm R64 =

conversion to radix 64 ASCII format

PGP Operation- Authentication

(a) Authentication only
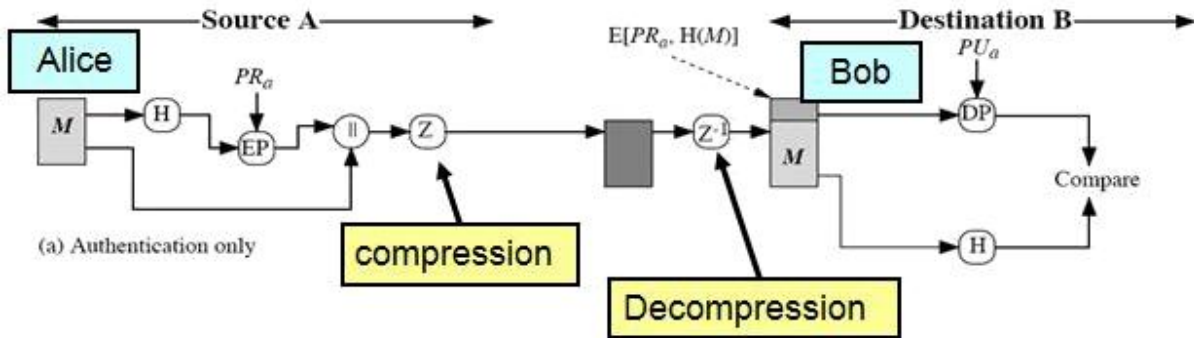
- sender creates message

- use SHA-1 to generate 160-bit hash of message

- signed hash with RSA using sender's private key, and is attached to message

- receiver uses RSA with sender's public key to decrypt and recover hash code

- receiver verifies received message using hash of it and compares with decrypted hash code

PGP Operation- Confidentiality
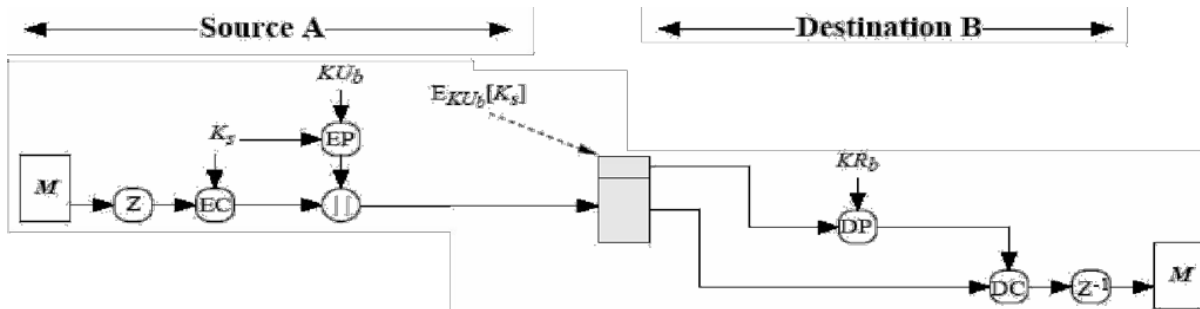
Sender:

- Generates message and a random number (session key) only for this message
- Encrypts message with the session key using AES, 3DES, IDEA or CAST-128
- Encrypts session key itself with recipient's public key using RSA
- Attaches it to message

Receiver:

- Recovers session key by decrypting using his private key
- Decrypts message using the session key

Confidentiality service provides no assurance to the receiver as to the identity of sender (i.e. no authentication). Only provides confidentiality for sender that only the recipient can read the message (and no one else)

PGP Operation – Confidentiality & Authentication
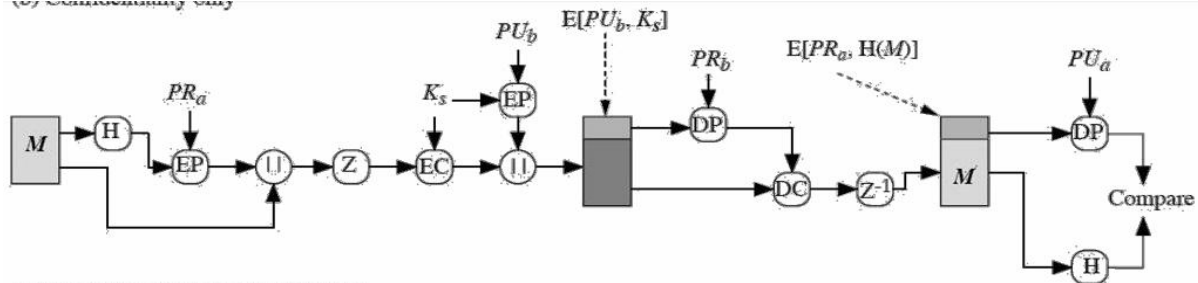
(c) Confidentiality and authentication

☐☐can use both services on same message
    ocreate signature & attach to messageoencrypt
    both message & signature
    oattach RSA/ElGamal encrypted session key o *is*

*called**authenticated confidentiality***

PGP Operation – Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage. The placement of the compression algorithm, indicated by Z for compression and Z⁻
for decompression is critical. The compression algorithm used is ZIP.

The signature is generated before compression for two reasons:

so that one can store only the uncompressed message together with signature for later verification

Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm as the PGP compression algorithm is not deterministic

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

PGP Operation – Email Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted, and thus consists of a stream of arbitrary 8-bit octets. However many electronic mail systems only permit the use of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters. It uses radix-64 conversion, in which each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message by 33%, but still an overall compression of about one-third can be achieved.

PGP Operation - Segmentation/Reassembly

E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. Reassembly at the receiving end is required before verifying signature or decryption

PGP Operations – Summary

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key, and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key, and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation	—	To accommodate maximum message size limitations, PGP performs segmentation and reassembly.
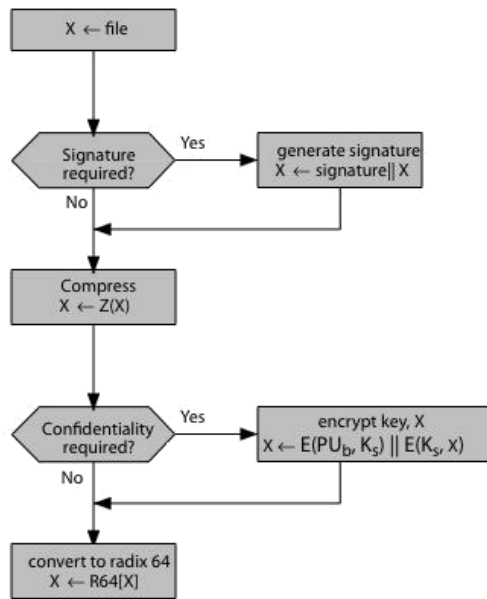
(a) Generic Transmission Diagram (from A)   (b) Generic Reception Diagram (to B)
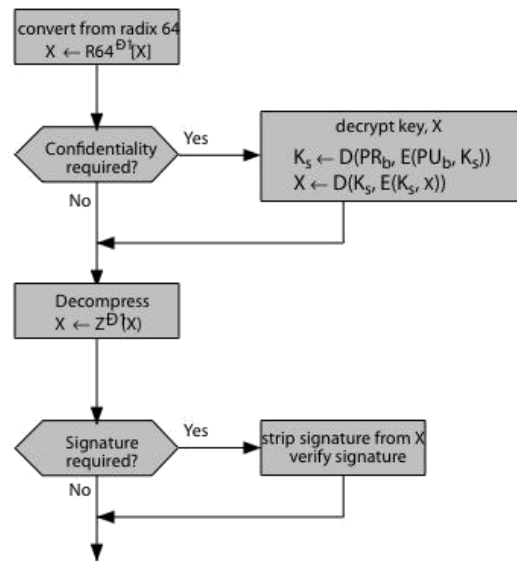
Cryptographic Keys and Key Rings

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys. Three separate requirements can be identified with respect to these keys:

1. a means of generating unpredictable session keys is needed.
2. a user is allowed to have multiple public-key/private-key pairs.
3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

PGP Session Keys

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message. Random numbers are generated using the algorithm specified in ANSI X12.17, with inputs based on keystroke input from the user, where both the keystroke timing and the actual keys struck are used to generate a randomized stream of numbers.

Key Identifiers

In PGP, any given user may have multiple public/private key pairs. That means, a user may have many public/private key pairs at his disposal. He wishes to encrypt or sign a message using one of his keys. But, the problem of informing the other party, which key he has used arises. Attaching the whole public key every time is inefficient. Rather PGP uses *akey identifier* based on the least significant 64-bits of the key, which will very likely be unique. That is, the key ID of public PUa is (PUa mod 2^{64}). Then only the much shorter key ID would need to be transmitted with any message. A key ID is also required for the PGP digital signature.

PGP Message Format

A message consists of three components: the message component, a signature (optional), and a session key component (optional).

The *message component* includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation. The *signature component* includes the following:

- *Timestamp*: The time at which the signature was made.
- *Message digest*: The 160-bit SHA-1 digest, encrypted with the sender's private signature key.
- *Leading two octets of message digest*: To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.
- *Key ID of sender's public key*: Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest
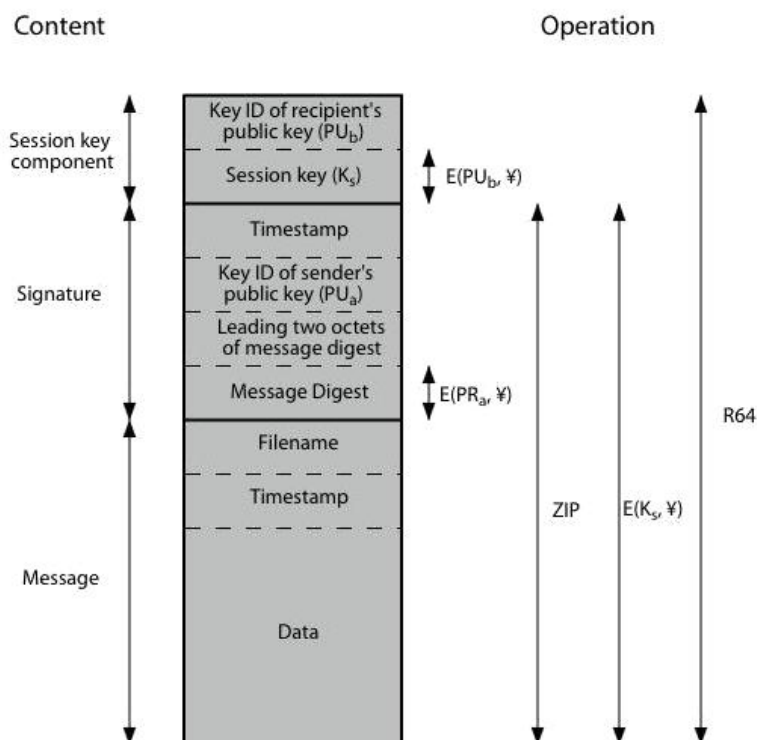
Notation:

$E(PU_b, \bullet)$ = encryption with user b's public key
$E(PR_a, \bullet)$ = encryption with user a's private key
$E(K_s, \bullet)$ = encryption with session key
ZIP = Zip compression function
R64 = Radix-64 conversion function

The *session key component* includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key. The entire block is usually encoded with radix-64 encoding.

PGP Key Rings

Keys & key IDs are critical to the operation of PGP. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. PGP uses a pair of data structures, one to store the user's public/private key pairs - their ***private-key ring***; and one to store the public keys of other known users, their ***public-key ring***.

General Structure of Private- and Public-Key Rings

a) Private-Key Ring

Private-Key Ring				
Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
• • •	• • •	• • •	• • •	• • •

The Private-Key ring can be viewed as a table, in which each row represents one of the public/private key pairs owned by this user. Each row contains the following entries:

- *Timestamp*: The date/time when this key pair was generated.
- *Key ID*:The least significant 64 bits of the public key for this entry.
- *Public key*: The public-key portion of the pair.
- *Private key*: The private-key portion of the pair; this field is encrypted.
- *User ID*: Typically, this will be the user's e-mail address (e.g., stallings@acm.org).However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same User ID more than once

The private-key ring is intended to be stored only on the machine of the user that created and owns the key pairs, and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

- The user selects a passphrase to be used for encrypting private keys.

- When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.

- The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code. . As in any system based on passwords, the security of this system depends on the security of the password, which should be not easily guessed but easily remembered.

- **Public-key Ring**
  This data structure is used to store public keys of other users that are known to this user.

Public-Key Ring							
Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	$trust\_flag_i$	User i	$trust\_flag_i$		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

- Timestamp: The date/time when this entry was generated.
- Key ID: The least significant 64 bits of the public key for this entry.
- Public Key: The public key for this entry.
- User ID: Identifies the owner of this key. Multiple user IDs may be associated with a single public key

PGP Message Transmission and Reception

Message transmission

The following figure shows the steps during message transmission assuming that the message is to be both signed and encrypted.
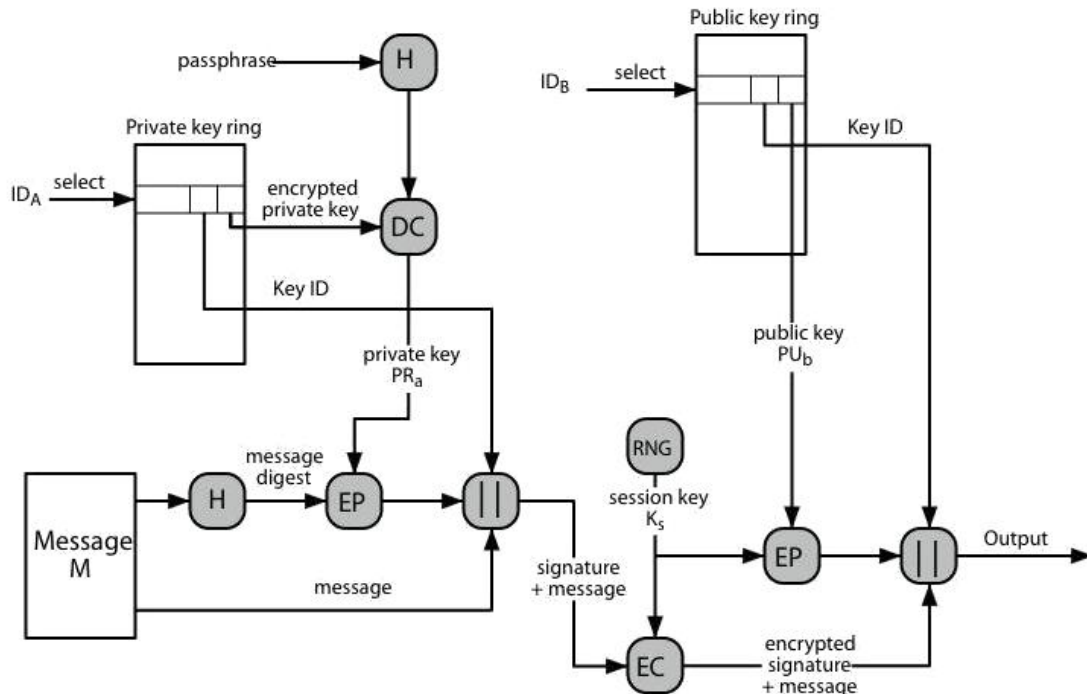
PGP Message Generation (from User A to User B; no compression or radix 64 conversion)

The sending PGP entity performs the following steps:

- Signing the message

  - PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.
  - PGP prompts the user for the passphrase to recover the unencrypted private key.
  - The signature component of the message is constructed.

- Encrypting the message

  - PGP generates a session key and encrypts the message.
  - PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.
  - The session key component of the message is constructed.

Message Reception
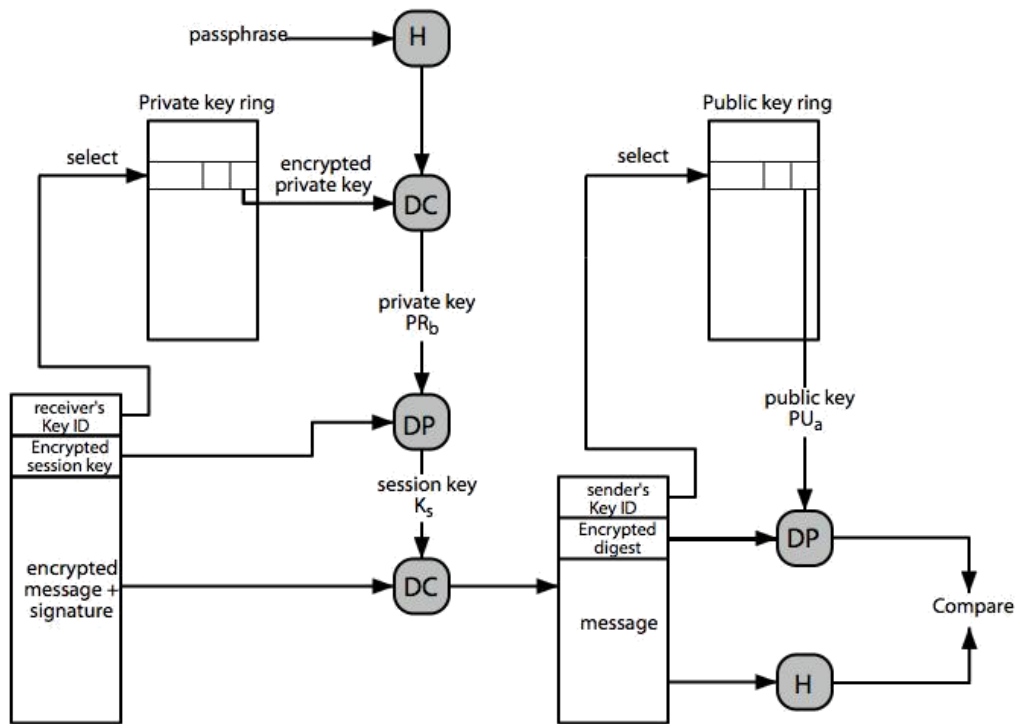
PGP Message Reception (from User A to User B; no compression or radix 64 conversion)

The receiving PGP entity performs the following steps:

☐ Decrypting the message

> PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.
> PGP prompts the user for the passphrase to recover the unencrypted private key.
> PGP then recovers the session key and decrypts the message.

☐ Authenticating the message

> PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
> PGP recovers the transmitted message digest.
> PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

Public Key Management

PGP contains a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service and also addresses the problem of public-key management.

Various Approaches for Public Key Management

A number of approaches are possible for minimizing the risk that a user's public-key ring contains false public keys. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

1. B could store her public key (PUb) on a floppy disk and hand it to A. This is a very secure method but has obvious practical limitations.
2. B could transmit her key in an e-mail message to A. A could have PGP generate a 160-bit SHA-1 digest of the key and display it in hexadecimal format; this is referred to as the "*fingerprint*" of the key. A could then call B and ask her to dictate the fingerprint over the phone. If the two fingerprints match, the key is verified. This is a more practical approach and for this A has to recognize the voice of B over the telephone.
3. Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key. D generates an SHA-1 digest of this certificate, encrypts it with her private key, and attaches the signature to the certificate. Because only D could have created the signature, no one else can create a false public key and pretend that it is signed by D. The signed certificate could be sent directly to A by B or D, or could be posted on a bulletin board.
4. Obtain B's public key from a trusted certifying authority. Again, a public key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.

The Use of Trust

PGP provides a better way of using trust, utilizing trust information and linking trust with public keys. The information about trust is stored in a 'trust flag byte'. Its structure consists of three fields:

1. key legitimacy field – KEYLEGITFIELD
2. signature trust field – SIGTRUST FIELD
3. owner trust field – OWNERTRUST FIELD

Key Legitimacy Field

It is computed by PGP. This field specifies the level of PGP's trust about the validity of user's public key. Based on the extent of trust, the user ID is bound to the key. A KEYLEGIT field can hold the following information:

1. unknown or undefined trust
2. key ownership not trusted
3. marginal trust in key ownership
4. complete trust in key ownership

A WARNONLY bit is set if user wants only to be warned when key that is not fully validated is used for encryption

Signature Trust Field

A key ring owner collects all the signatures that are related to the entries. Each signature has its own signature-trust-field that specifies the level of PGP user's trust towards the signer, so that all its public keys can be certified. A SIGTRUST FIELD can hold values like:

1. undefined trust
2. unknown user
3. usually not trusted to sign other keys
4. usually trusted to sign other keys
5. always trusted to sign other keys
6. this key is present in secret key ring (ultimate trust)

It also has a CONTIG bit that is set if signature tends to a contiguous trusted certification path that will ultimately reach the trusted key ring owner

Owner Trust Field

Each entry in the public key ring represents a public key that is related to a particular owner along with a owner-trust-field. This field specifies the extent of trust towards the public key, so that it can be used to sign other public-key-certificates. User is supposed to assign this field. An OWNERTRUST field can hold values like:

1. undefined trust
2. unknown user
3. usually not trusted to sign other keys
4. usually trusted to sign other keys
5. always trusted to sign other keys
6. this key is present in secret key ring (ultimate trust)

It also has a BUCKSTOP bit that is automatically set, if the key is present in the secret key ring.

Operation of Trust Processing

Consider the public key ring of User-A, then the operation of trust processing is described as follows:

1. When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of ultimate trust is automatically assigned to the trust field. Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify that this owner is unknown, untrusted, marginally trusted, or completely trusted.

2. When the new public key is entered, one or more signatures may be attached to it. More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an unknown user value is assigned

3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of ultimate, then the key legitimacy value is set to complete. Otherwise, PGP computes a weighted sum of the trust values. ~~A weight of 1/X is given to signatures that are always trusted and 1/Y to signatures that are usually trusted, where X and Y are user configurable parameters. When the total of weights of the introducers of a key/UserID combination reaches 1, the binding is considered to be trustworthy, and the key legitimacy value is set to complete. Thus, in the absence of ultimate trust, at least X signatures that are always trusted or Y signatures that are usually trusted or some combination is needed.~~

PGP scans the public key ring in a top-down manner for assuring consistency. Each OWNERTRUST field is scanned by PGP for all signatures with the authorization of that owner in order to update SIGTRUST field, so that it becomes equal to the OWNERTRUST field. To start this process, it selects the keys with 'ultimate trust' first and then determines all the KEYLEGIT fields that are based on the attached signatures.

Revoking Public Keys

When a user suspects that his opponent might have acquired his unencrypted private key or if he doesn't want to use the same key for a long period, he must revoke(cancel) his current public key. In order to revoke a public key, the owner will have to issue a signed key revocation certificate. To sign this certificate, corresponding private key is used. This certificate is similar to that of the general signature certificates except that, this certificate is used for revoking its public key. The owner will then broadcast this certificate as soon as possible so that others can update their public key rings.

PGP "Web of Trust"

The idea behind the various trust fields in the public key ring is to establish a "Web of Trust" among a community of users.
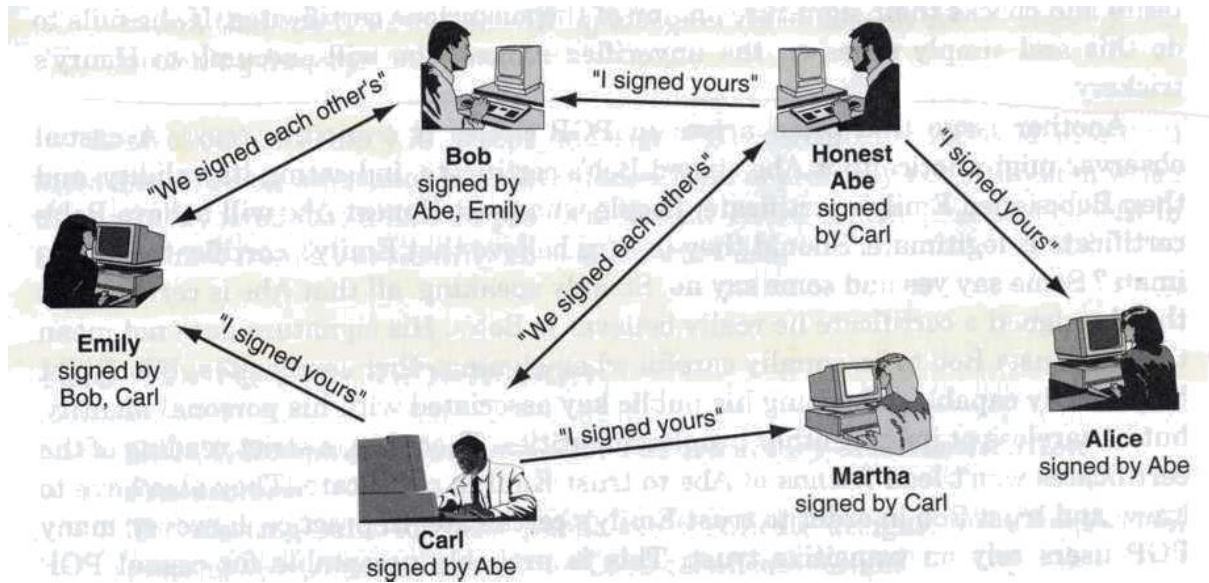
If Alice trusts <u>only</u> Abe to sign certificates, then she won't believe certificates from Martha or Emily are genuine. If she also trusts Bob's judgment about signing certificates, she can trust Emily's certificate; if she also trusts Carl, she can trust everyone's certificate.

Figure 12.7 PGP Trust Model Example

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, which in turn provided support for varying content types and multi-part messages over the text only support in the original Internet RFC822 email standard. MIME allows encoding of binary data to textual form for transport over traditional RFC822 email systems. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851 and S/MIME support is now included in many modern mail agents.

RFC 822

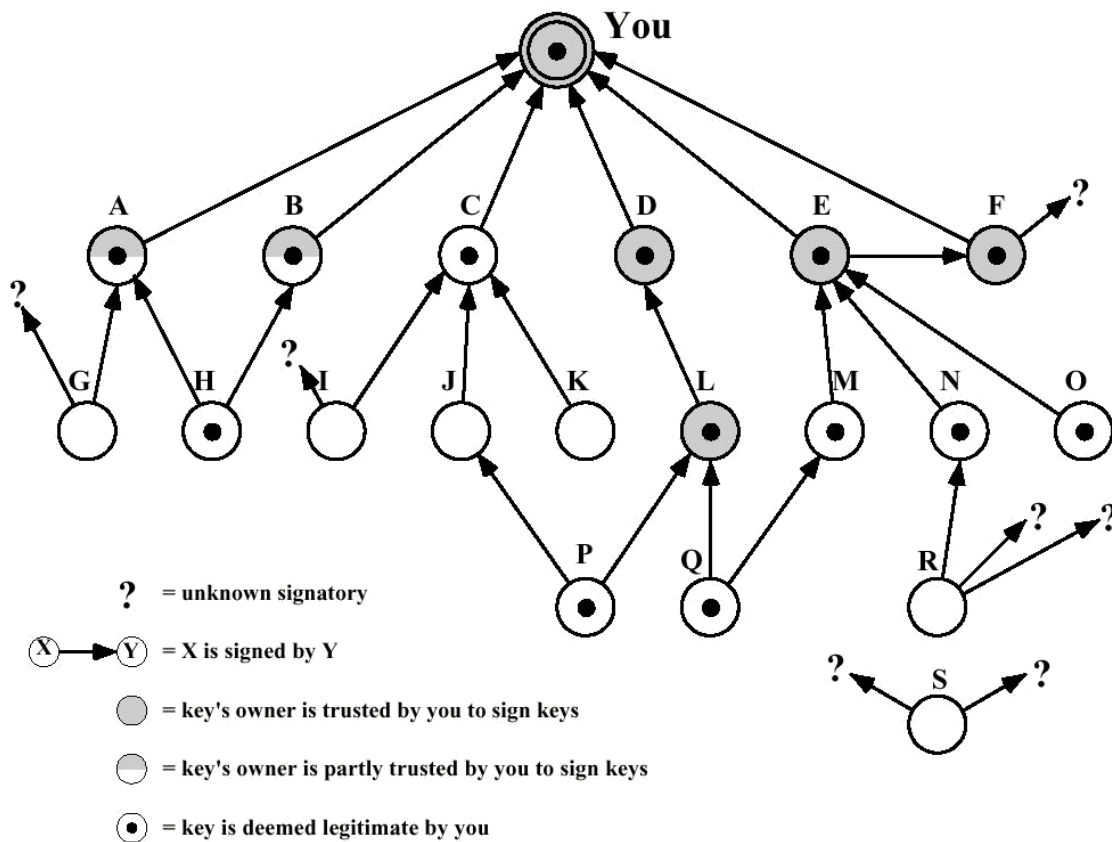RFC 822 defines a format for text messages that are sent using electronic mail and it has been the standard for Internet-based text mail message. The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line. A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From, To*, *Subject*, and *Date*.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail.

Problems with RFC 822 and SMTP

- Executable files or other binary objects must be converted into ASCII. Various schemes exist (e.g., Unix UUencode), but a standard is needed
- Text data that includes special characters (e.g., Hungarian text) cannot be transmitted as SMTP is limited to 7-bit ASCII
- Some servers reject mail messages over a certain size
- Some common problems exist with the SMTP implementations which do not adhere completely to the SMTP standards defined in RFC 821. They are:
  - delete, add, or reorder CR and LF characters
  - truncate or wrap lines longer than 76 characters
  - remove trailing white space (tabs and spaces)
  - pad lines in a message to the same length
  - convert tab characters into multiple spaces

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations and the specification is provided in RFC's 2045 through 2049.

The MIME specification includes the following elements:

5. Five new message header fields are defined, which provide information about the body of the message.
6. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
7. Transfer encodings are defined that protect the content from alteration by the mail system.

MIME - New header fields

The five header fields defined in MIME are as follows:

5. _MIME-Version:_ Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
6. _Content-Type_: Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
7. _Content-Transfer-Encoding_: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
8. Content-ID: Used to identify MIME entities uniquely in multiple contexts.
9. _Content-Description_: A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

For the text type of body, the primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility.

The multipart type indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter called boundary that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header. There are four subtypes of the multipart type, all of which have the same overall syntax.

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet-stream	General binary data consisting of 8-bit bytes.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message. The message/partial subtype enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments. The message/external-body subtype indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values. Three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters. The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semanties of the media type, which may have strong implications for any character set conversion (e.g. with regard to syntactically meaningful characters in a text subtype other than "plain").

S/MIME Functionality

S/MIME has a very similar functionality to PGP. Both offer the ability to sign and/or encrypt messages.

Functions

S/MIME provides the following functions:

6. **Enveloped data**: This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.
7. **Signed data**: A digital signature is formed by taking the message digest of thecontent to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
8. **Clear-signed data**: As with signed data, a digital signature of the content is formed.However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
9. **Signed and enveloped data**: Signed-only and encrypted-only entities may be nested,so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Cryptographic Algorithms

S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:

6. *Must:* The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
7. *Should:* There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

The following table summarizes the cryptographic algorithms used in S/MIME.

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys; in fact, S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal. As an alternative, RSA, can be used for both signatures and session key encryption. These are the same algorithms used in PGP and provide a high level of security. For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME. As there is justifiable concern about the security of MD5, SHA-1 is clearly the preferred alternative.

A sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference any message that it sends out. A receiving agent may store that information for future use.

The following rules, in the following order, should be followed by a sending agent:

- If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.
- If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.
- If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.
- If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown below:

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs 7-mime	signedData	A signed S/MIME entity.
	pkcs 7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs 7-mime	degenerate signedData	An entity containing only public- key certificates.
	pkcs 7-mime	CompressedData	A compressed S/MIME entity
	pkcs 7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

S/MIME Content Types

Securing a MIME Entity

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object. A PKCS object is then treated as message content and wrapped in MIME (provided with appropriate MIME headers).

EnvelopedData

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique smime-type parameter. In all cases, the resulting entity, referred to as an object, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The BER format consists of arbitrary octet strings and is therefore binary data. Such an object should be transfer encoded with base64 in the outer MIME message. We first look at envelopedData.

The steps for preparing an envelopedData MIME entity are as follows:

☐ Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or tripleDES).

2. For each recipient, encrypt the session key with the recipient's public RSA key.

☐ For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate,[3] an identifier of the algorithm used to encrypt the session key, and the encrypted session key.

☐ Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

Clear Signing

Clear signing is achieved using the multipart content type with a signed subtype. This signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted-printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature.The protocol parameter indicates that this is a two-part clear-signed entity. The micalg parameter indicates the type of message digest used. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

Registration Request

Typically, an application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request. The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key. The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

Certificates-Only Message

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with ansmime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509. The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists, needed to verify incoming signatures and to encrypt outgoing messages. But certificates are signed by trusted certification authorities.

User Agent Role

An S/MIME user has several key-management functions to perform:

- **Key generation:** The user of some related administrative utility (e.g., one associatedwith LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs.
- **Registration:** A user's public key must be registered with a certification authority inorder to receive an X.509 public-key certificate.
- **Certificate storage and retrieval:** A user requires access to a local list of certificates inorder to verify incoming signatures and to encrypt outgoing messages.

S/MIME – Certification Authorities

"Certificate Authority" (CA), or "Trust Center", is the name used for an organisation that acts as the agent of trust in a PKI (Public Key Infrastructure) and also for the piece of software. PKI needed for secure use of public key based protocols

A CA performs 5 main functions:

1) Verifies users' identities - this may be done by the CA itself, or on its behalf by a Local Registration Authority (LRA)
2) Issues users with keys (though sometimes users may generate their own key pair)
3) Certifies users' public keys
4) Publishes users' certificates
5) Issues certificate revocation lists (CRLs)

VeriSign Certificates

There are several companies that provide certification authority (CA) services. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

- Owner's public key
- Owner's name or alias
- Expiration date of the Digital ID
- Serial number of the Digital ID
- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve.

☐ For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.

☐ For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.

☐ For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Radix-64 Conversion

Both PGP and S/MIME make use of an encoding technique referred to as radix-64 conversion. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

6. The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set.
7. The character set consists of 65 printable characters, one of which is used for padding. With $2^6 = 64$ available characters, each character can be used to represent 6 bits of input
8. No control characters are included in the set
9. The hyphen character ("-") is not used.

6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

For example, consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:

001000 110101 110010 010001

The extracted 6-bit decimal values are 8, 53, 50, 17. Looking these up in above table yields the radix-64 encoding as the following characters: I1yR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

01001001 00110001 01111001 01010010

In hexadecimal, this is 49317952. To summarize,

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	I1yR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

Def: Internet Protocol security (IPSec) is a framework of open standards for protecting communications over Internet Protocol (IP) networks through the use of cryptographic security services. IPSec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

Need for IPSec

In Computer Emergency Response Team (CERT)'s 2001 annual report it listed 52,000 security incidents in which most serious types of attacks included **IP spoofing**, in which intruders create packets with false IP addresses and exploit applications that use authentication based on IP and various forms of **eavesdropping and packet sniffing**, in which attackers read transmitted information, including logon information and database contents. In response to these issues, the IAB included authentication and encryption as necessary security features in the next-generation IP i.e. IPv6.

Applications of IPSec

IPSec provides the capability to secure communications across a LAN, across private and public wide area networks (WAN's), and across the Internet.

☐ *Secure branch office connectivity over the Internet*: A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
☐ *Secure remote access over the Internet*: An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for travelling employees and telecommuters.
☐ *Establishing extranet and intranet connectivity with partners*: IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
☐ *Enhancing electronic commerce security*: Even though some Web and electronic commerce applications have built-in security protocols, the use of IPSec enhances that security.

The principal feature of IPSec enabling it to support varied applications is that it can encrypt and/or authenticate all traffic at IP level. Thus, all distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured.

The following figure shows a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Non secure IP traffic is conducted on each LAN.
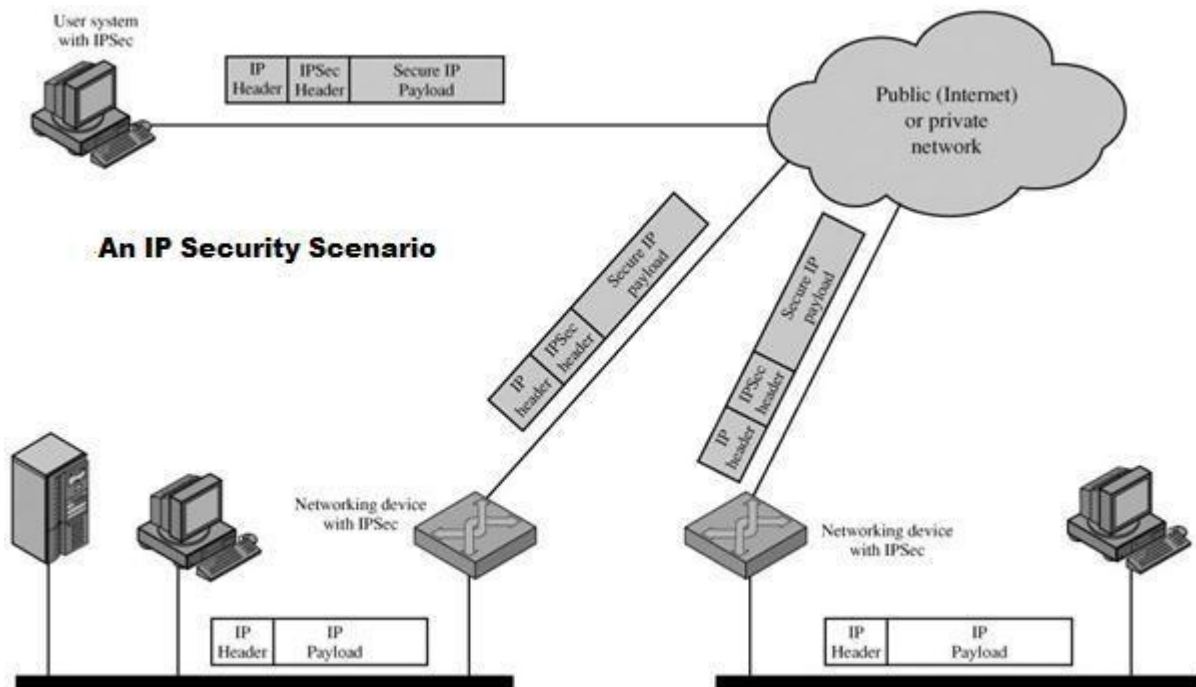
The IPSec protocols operate in networking devices, such as a router or firewall that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

Benefits of IPSec
The benefits of IPSec are listed below:
- IPSec in a firewall/router provides strong security to all traffic crossing the perimeter

- IPSec in a firewall is resistant to bypass

- IPSec is below transport layer(TCP,UDP), hence transparent to applications

- IPSec can be transparent to end users

- IPSec can provide security for individual users if needed (useful for offsite workers and setting up a secure virtual subnetwork for sensitive applications)

Routing Applications
IPSec also plays a vital role in the routing architecture required for internetworking. It assures that:
- router advertisements come from authorized routers
- neighbor advertisements come from authorized routers
- redirect messages come from the router to which initial packet was sent
- A routing update is not forged

To understand IP Security architecture, we examine IPSec documents first and then move on to IPSec services and Security Associations.

IPSec Documents

The IPSec specification consists of numerous documents. The most important of these, issued in November of 1998, are RFCs 2401, 2402, 2406, and 2408:

- RFC 2401: An overview of a security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

Support for these features is mandatory for IPv6 and optional for IPv4. In both cases, the security features are implemented as extension headers that follow the main IP header. The extension header for authentication is known as the Authentication header; that for encryption is known as the Encapsulating Security Payload (ESP) header. In addition to these four RFCs, a number of additional drafts have been published by the IP Security Protocol Working Group set up by the IETF. The documents are divided into seven groups, as depicted in following figure:
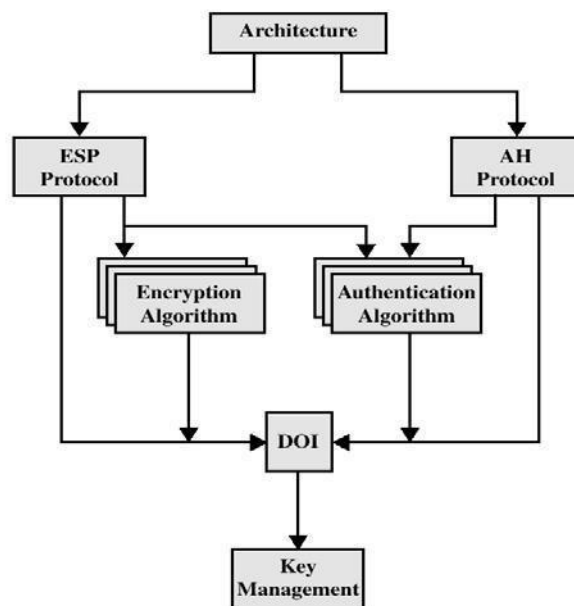
- **Architecture**: Covers the general concepts, security requirements, definitions, and mechanisms defining IPSec technology
- **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.
- **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.

- □ **Encryption Algorithm**: A set of documents that describe how various encryption algorithms are used for ESP.
- □ **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.
- □ **Key Management**: Documents that describe key management schemes.
- □ **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

IPSec Services

IPSec architecture makes use of two major protocols (i.e., Authentication Header and ESP protocols) for providing security at IP level. This facilitates the system to beforehand choose an algorithm to be implemented, security protocols needed and any cryptographic keys required to provide requested services. The IPSec services are as follows:

- □ **Connectionless Integrity:-** Data integrity service is provided by IPSec via AH which prevents the data from being altered during transmission.
- □ **Data Origin Authentication**:- This IPSec service prevents the occurrence of replay attacks, address spoofing etc., which can be fatal
- □ **Access Control**:- The cryptographic keys are distributed and the traffic flow is controlled in both AH and ESP protocols, which is done to accomplish access control over the data transmission.
- □ **Confidentiality**:- Confidentiality on the data packet is obtained by using an encryption technique in which all the data packets are transformed into ciphertext packets which are unreadable and difficult to understand.
- □ **Limited Traffic Flow Confidentiality**:- This facility or service provided by IPSec ensures that the confidentiality is maintained on the number of packets transferred or received. This can be done using padding in ESP.
- □ **Replay packets Rejection**:- The duplicate or replay packets are identified and discarded using the sequence number field in both AH and ESP.

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

Security Associations

Since IPSEC is designed to be able to use various security protocols, it uses Security Associations (SA) to specify the protocols to be used. SA is a database record which specifies security parameters controlling security operations. They are referenced by the sending host and established by the receiving host. An index parameter called the Security Parameters Index (SPI) is used. SAs are in one direction only and a second SA must be established for the transmission to be bi-directional. A security association is uniquely identified by three parameters:

- **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- **IP Destination Address**: Currently, only unicast addresses are allowed; this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.
- **Security Protocol Identifier**: This indicates whether the association is an AH or ESP security association.

SA Parameters

In each IPSec implementation, there is a nominal Security Association Database that defines the parameters associated with each SA. A security association is normally defined by the following parameters:

- **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers
- **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).
- **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay
- **AH Information:** Authentication algorithm, keys, key lifetimes, and related parameters being used with AH (required for AH implementations).
- **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations).
- **Lifetime of This Security Association:** A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations).
- **IPSec Protocol Mode:** Tunnel, transport, or wildcard (required for all implementations). These modes are discussed later in this section.
- **Path MTU:** Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations).

Transport and Tunnel Modes

Both AH and ESP support two modes of use: transport and tunnel mode.

	Transport Mode SA	**Tunnel Mode SA**
AH	**Authenticates** IP payload and selected portions of IP header and IPv6 extension headers	**Authenticates** entire inner IP packet plus selected portions of outer IP header
ESP	**Encrypts** IP payload and any IPv6 extesion header	**Encrypts** inner IP packet
ESP with authentication	**Encrypts** IP payload and any IPv6 extesion header. **Authenticates** IP payload but no IP header	**Encrypts** inner IP packet. **Authenticates** inner IP packet.

IP sec can be used (both AH packets and ESP packets) in two modes

- **Transport mode**: the IP sec header is inserted just after the IP header –this contains the security information, such as SA identifier, encryption, authentication
    Typically used in end-to-end communication
    IP header not protected

- **Tunnel mode**: the entire IP packet, header and all, is encapsulated in the body of a new IP packet with a completely new IP header
    Typically used in firewall-to-firewall communication
    Provides protection for the whole IP packet
    No routers along the way will be able (and will not need) to check the content of the packets
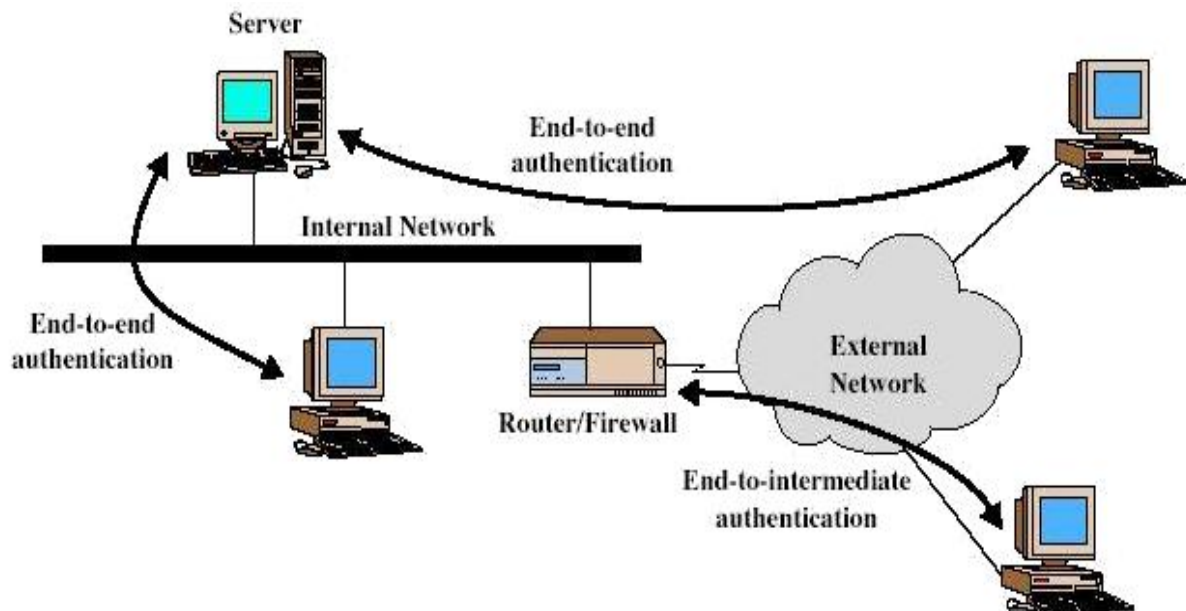
End-to-End versus End-to-Intermediate Authentication

The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feature ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly; it also prevents the address spoofing attacks observed in today's Internet. The AH also guards against the replay attack. Authentication is based on the use of a message authentication code (MAC), hence the two parties must share a secret key. The Authentication Header consists of the following fields:


IPSec Authentication Header
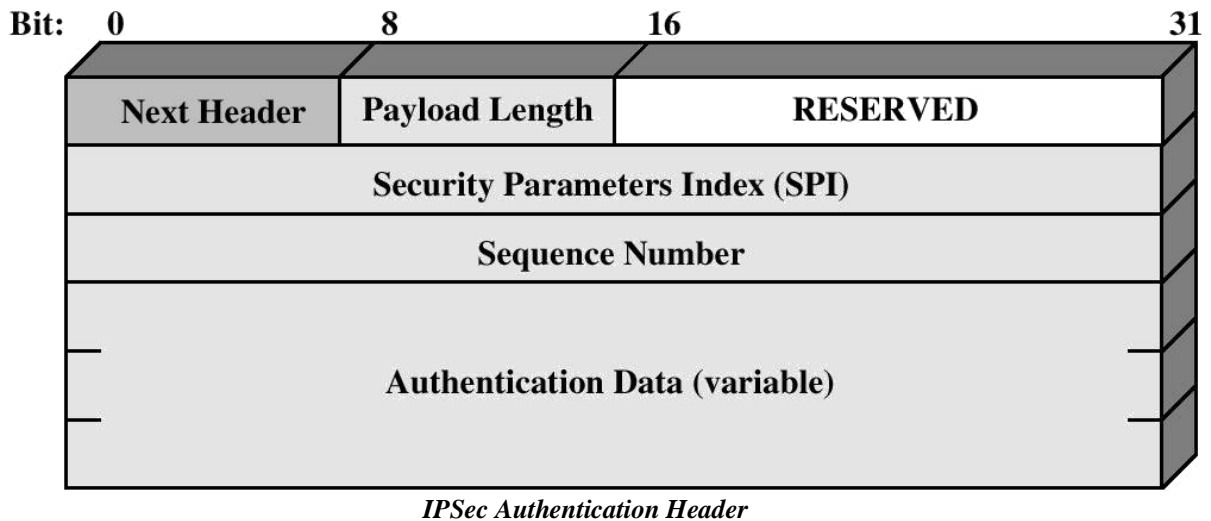
- **Next Header (8 bits):** Identifies the type of header immediately following this header.
- **Payload Length (8 bits):** Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.
- **Reserved (16 bits):** For future use.
- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value, discussed later.
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet.

Anti-Replay Service

Anti-replay service is designed to overcome the problems faced due to replay attacks in which an intruder intervenes the packet being transferred, make one or more duplicate copies of that authenticated packet and then sends the packets to the desired destination, thereby causing inconvenient processing at the destination node. The Sequence Number field is designed to thwart such attacks.

When a new SA is established, the sender initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. This value goes on increasing with respect to the number of packets being transmitted. The sequence number field in each packet represents the value of this counter. The maximum value of the sequence number field can go up to $2^{32}-1$. If the limit of $2^{32}-1$ is reached, the sender should terminate this SA and negotiate a new SA with a new key.

The IPSec authentication document dictates that the receiver should implement a window of size W, with a default of W = 64. The right edge of the window represents the highest sequence number, N, so far received for a valid packet. For any packet with a sequence number in the range from N-W+1 to N that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked as shown. Inbound processing proceeds as follows when a packet is received:
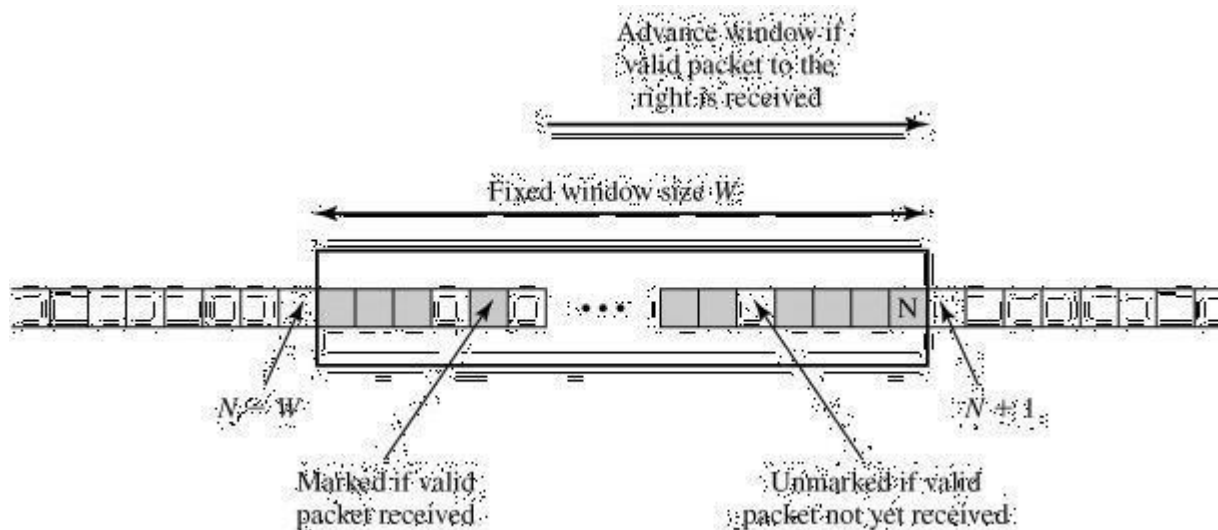
Antireplay Mechanism

1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.
3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

Integrity Check Value

ICV is the value present in the authenticated data field of ESP/AH, which is used to determine any undesired modifications made to the data during its transit. ICV can also be referred as MAC or part of MAC algorithm. MD5 hash code and SHA-1 hash code are implemented along with HMAC algorithms i.e.,

- HMAC-MD5-96
- HMAC-SHA-1-96

In both cases, the full HMAC value is calculated but then truncated by using the first 96 bits, which is the default length for the Authentication Data field. The MAC is calculated over

- IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival is unpredictable are set to zero for purposes of calculation at both source and destination.
- The AH header other than the Authentication Data field. The Authentication Data field is set to zero for purposes of calculation at both source and destination.
- The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).
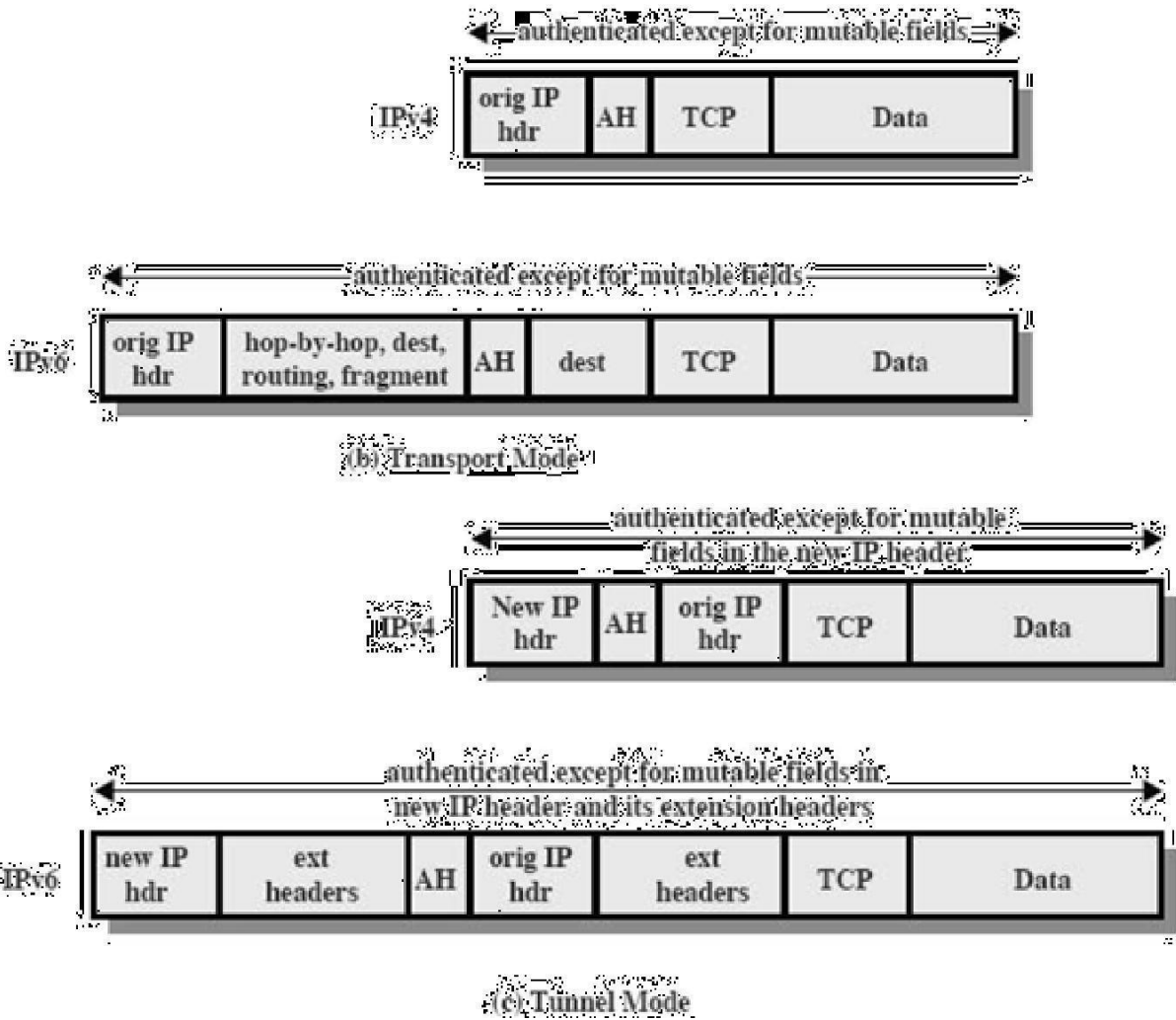
Transport and Tunnel Modes

The following figure shows typical IPv4 and IPv6 packets. In this case, the IP payload is a TCP segment; it could also be a data unit for any other protocol that uses IP, such as UDP or ICMP.

(a) Before Applying AH
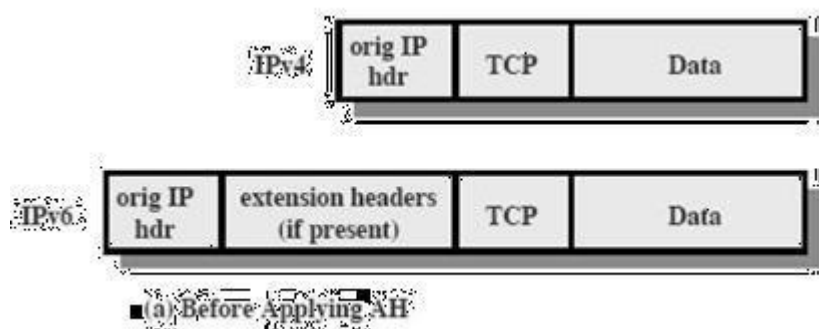
For transport mode AH using IPv4, the AH is inserted after the original IP header and before the IP payload (e.g., a TCP segment) shown below. Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation. In the context of IPv6, AH is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the AH appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the AH header, depending on the semantics desired. Again, authentication covers the entire packet, excluding mutable fields that are set to zero for MAC calculation.

IPv4	orig IP hdr	AH	TCP	Data

authenticated except for mutable fields

IPv6	orig IP hdr	hop-by-hop, dest, routing, fragment	AH	dest	TCP	Data

(b) Transport Mode

authenticated except for mutable fields in the new IP header

IPv4	New IP hdr	AH	orig IP hdr	TCP	Data

authenticated except for mutable fields in new IP header and its extension headers

| IPv6	new IP hdr	ext headers	AH	orig IP hdr	ext headers	TCP	Data

(c) Tunnel Mode

For tunnel mode AH, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header. The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses of firewalls or other security gateways). With tunnel mode, the entire inner IP packet, including the entire inner IP header is protected by AH. The outer IP header (and in the case of IPv6, the outer IP extension headers) is protected except for mutable and unpredictable fields.

The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.

ESP Format
The following figure shows the format of an ESP packet. It contains the following fields:
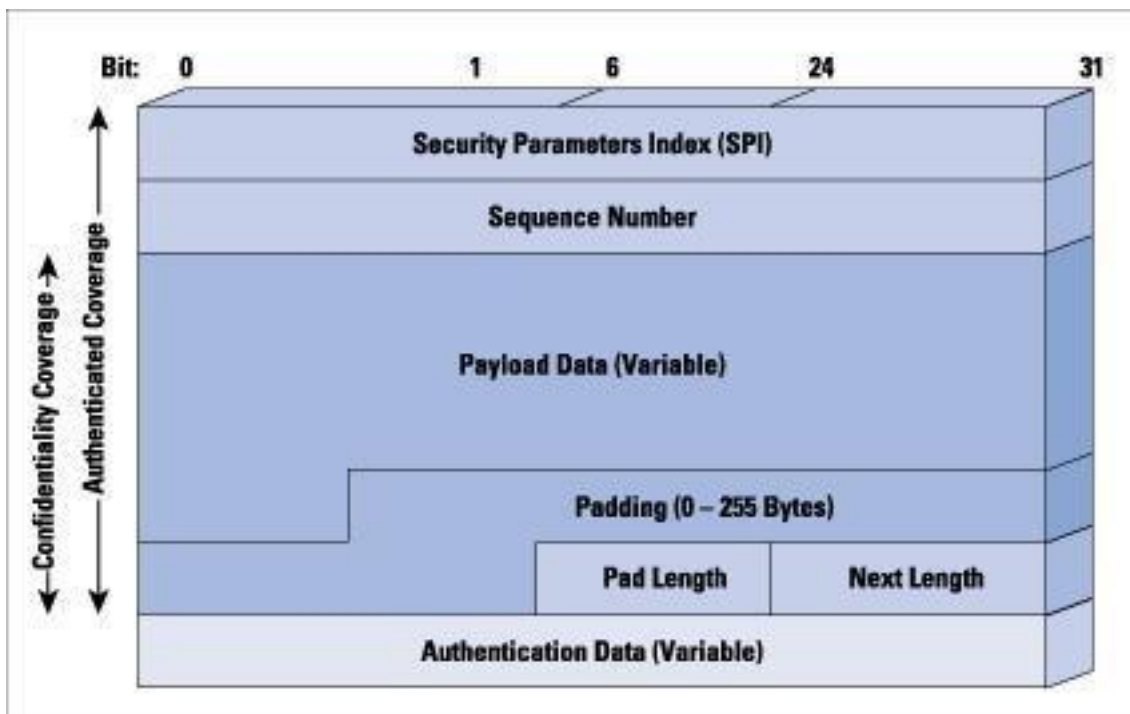

IPSec ESP format

- **Security Parameters Index** (32 bits): Identifies a security association.
- **Sequence Number** (32 bits): A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.
- **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.
- **Padding (0-255 bytes):** This field is used to make the length of the plaintext to be a multiple of some desired number of bytes. It is also added to provide confidentiality.
- **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.
- **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

Adding encryption makes ESP a bit more complicated because the encapsulation *surrounds* the payload rather than *precedes* it as with AH: ESP includes header and trailer

fields to support the encryption and optional authentication. It also provides Tunnel and Transport modes. The IPSec RFCs don't insist upon any particular encryption algorithms, but we find DES, triple-DES, AES, and Blowfish in common use to shield the payload from prying eyes. The algorithm used for a particular connection is specified by the Security Association and this SA includes not only the algorithm, but the key used. Unlike AH, which provides a small header *before* the payload, ESP *surrounds* the payload it's protecting. The Security Parameters Index and Sequence Number serve the same purpose as in AH, but we find padding, the next header, and the optional Authentication Data at the end, in the ESP Trailer.
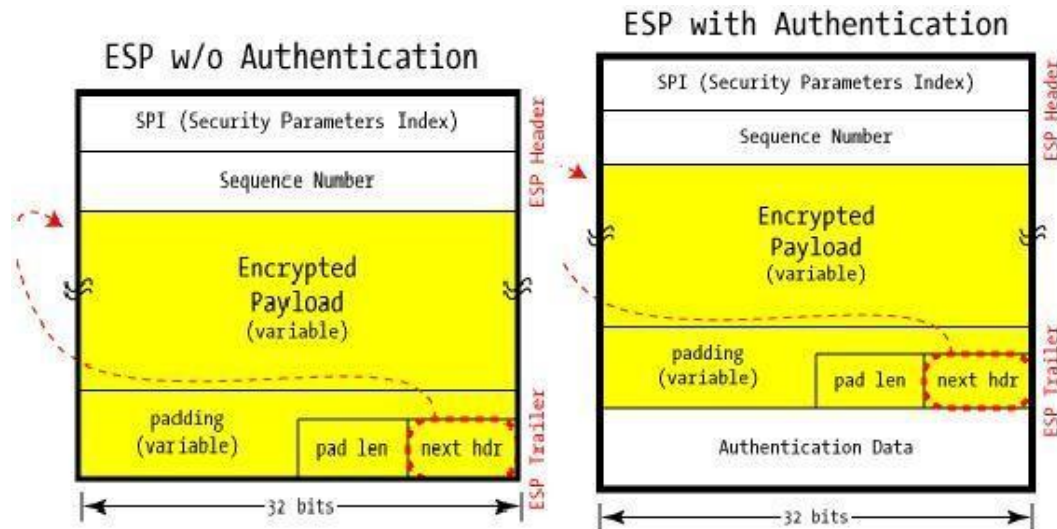

It's possible to use ESP without any actual encryption (to use a NULL algorithm), which nonetheless structures the packet the same way. This provides no confidentiality, and it only makes sense if combined with ESP authentication. Padding is provided to allow block-oriented encryption algorithms room for multiples of their block size, and the length of that padding is provided in the **pad len** field. The **next hdr** field gives the type (IP, TCP, UDP, etc.) of the payload in the usual way, though it can be thought of as pointing "backwards" into the packet rather than forward as we've seen in AH. In addition to encryption, ESP can also optionally provide authentication, with the same HMAC as found in AH. Unlike AH, however, this authentication is *only for the ESP header and encrypted payload*: it does not cover the full IP packet.

Transport Mode ESP

Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment). For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP) and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet; if authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header. In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment

extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.
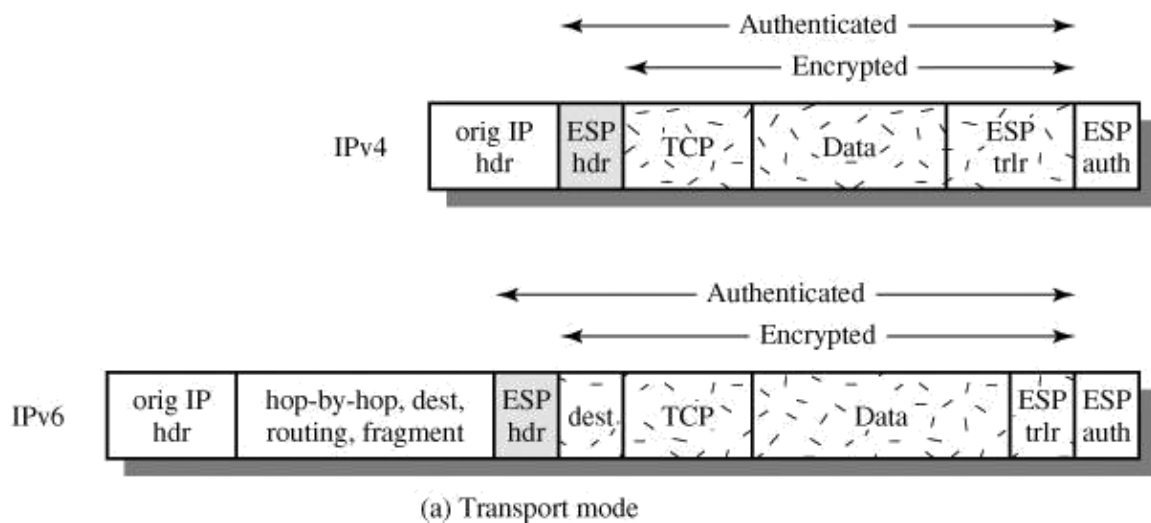
(a) Transport mode

Transport mode operation may be summarized as follows:

5. At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

6. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.

7. The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thus avoiding the need to implement confidentiality in every individual application. This mode of operation is also reasonably efficient, adding little to the total length of the IP packet. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

Tunnel Mode ESP

In case of tunnel mode ESP, ESP header and the ESP trailer are attached before and after the IP packet respectively, then the complete IP packet which includes IP header, Transport header and data field along with the ESP trailer is encrypted. Tunnel mode ESP is used to protect against the traffic flow analysis. But if ESP header precedes the IP header, the routers cannot identify and process this packet as the routing information and other parameters needed are present in the IP header of the packet. To overcome this problem,

the complete structure which contains ESP header, encrypted text as well as authentication data are encapsulated in a new IP packet with a new IP header. This new IP header has enough routing information inorder to process the packet to the appropriate destination.
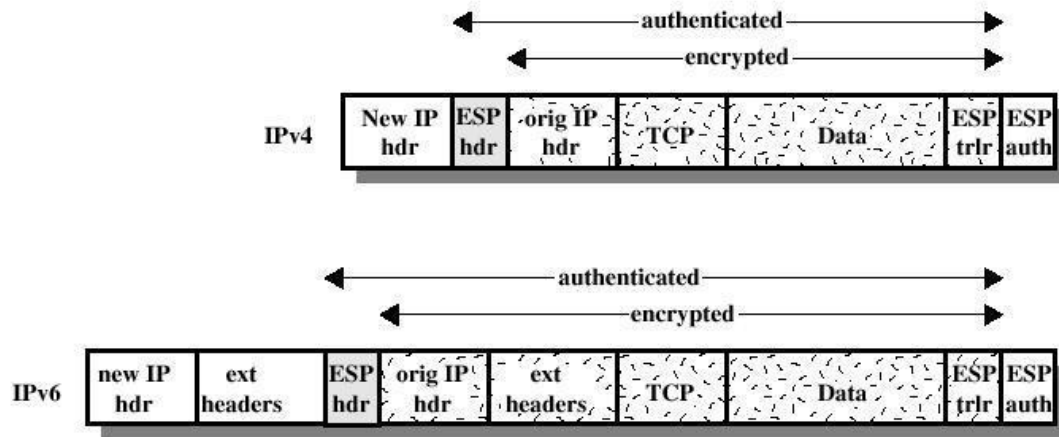
(b) Tunnel Mode

The transport mode is suitable for protecting connections between hosts that support the ESP feature and the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. Consider a case in which an external host wishes to communicate with a host on an internal network protected by a firewall, and in which ESP is implemented in the external host and the firewalls. The following steps occur for transfer of a transport-layer segment from the external host to the internal host:

4. The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms the outer IP packet.
5. The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus anyouter IP extension headers but does not need to examine the ciphertext.
6. The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet. This packet is then transmitted in the internal network.
7. The inner packet is routed through zero or more routers in the internal network to the destination host.

An individual SA can implement either the AH or ESP protocol but not both. Multiple SAs must be employed for traffic flow to achieve the desired IPSec services. The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPSec services. The SAs in a bundle may terminate at different endpoints or at the same endpoints. Security associations may be combined into bundles in two ways:

5. **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet, without invoking tunnelling.
6. **Iterated tunnelling:** Refers to the application of multiple layers of security protocols effected through IP tunnelling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPSec site along the path.

Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. There are several approaches for this:

ESP with Authentication Option

In this approach, the encryption is carried out on a data packet prior to its authentication. This can be represented using the following two cases:

7. Transport Mode ESP
8. Tunnel Mode ESP

Transport Adjacency

Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

Transport-Tunnel Bundle

The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for

later reference. It is more convenient to do this if the authentication information applies to the unencrypted message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

Basic Combinations of Security Associations

The IPSec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPSec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router).

case:-1
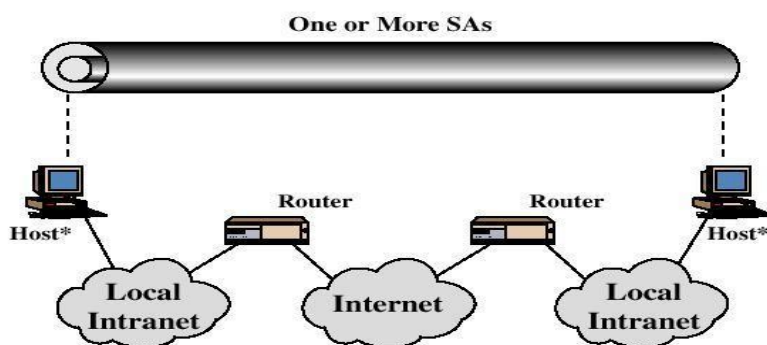
(a) Case 1

All security is provided between end systems that implement IPSec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations:
7. AH in transport mode
8. ESP in transport mode
9. ESP followed by AH in transport mode (an ESP SA inside an AH SA)
10. Any one of a, b, or c inside an AH or ESP in tunnel mode

Case:-2

(b) Case 2

128
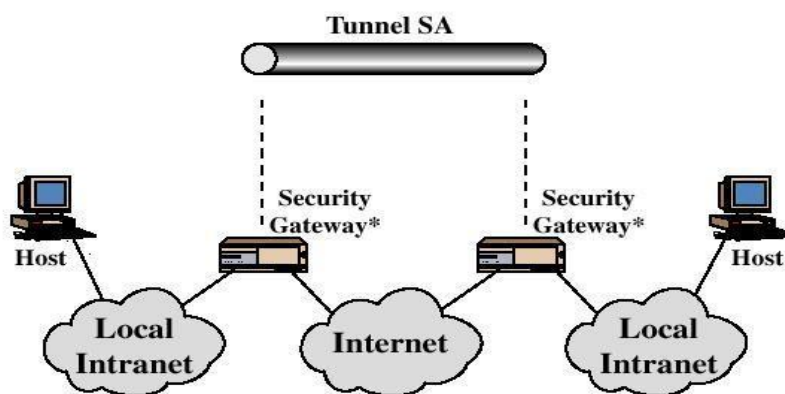
Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.

Case-3:-
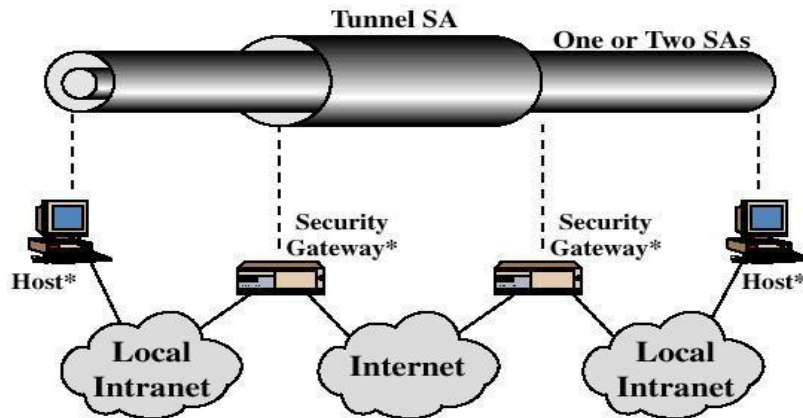
(c) Case 3

The third combination is similar to the second, but in addition provides security even to nodes. This combination makes use of two tunnels first for gateway to gateway and second for node to node. Either authentication or the encryption or both can be provided by using gateway to gateway tunnel. An additional IPSec service is provided to the individual nodes by using node to node tunnel.

Case:-4
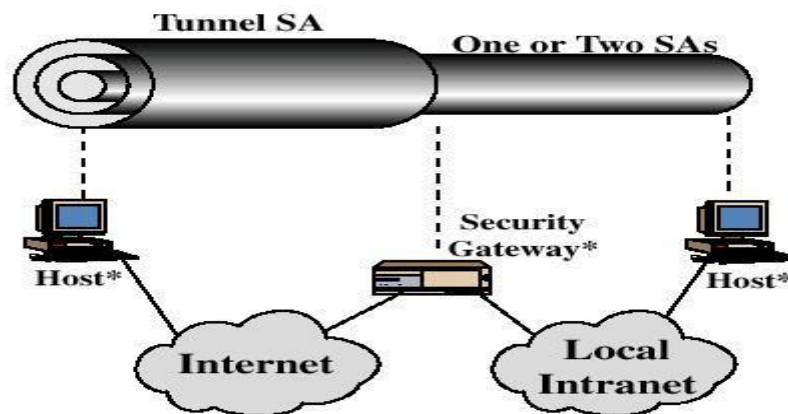
(d) Case 4

This combination is suitable for serving remote users i.e., the end user sitting anywhere in the world can use the internet to access the organizational workstations via the firewall. This combination states that only one tunnel is needed for communication between a remote user and an organizational firewall.

The key management portion of IPSec involves the determination and distribution of secret keys. The IPSec Architecture document mandates support for two types of key management:

4. **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
5. **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPSec is referred to as ISAKMP/Oakley and consists of the following elements:

• **Oakley Key Determination Protocol**: Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
• **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

Oakley Key Determination Protocol

Oakley is a refinement of the Diffie-Hellman key exchange algorithm. The Diffie-Hellman algorithm has two attractive features:
• Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.
• The exchange requires no pre-existing infrastructure other than an agreement on the global parameters.

However, Diffie-Hellman has got some weaknesses:
• No identity information about the parties is provided.
• It is possible for a man-in-the-middle attack
• It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys.

Oakley is designed to retain the advantages of Diffie-Hellman while countering its weaknesses.

Features of Oakley

The Oakley algorithm is characterized by five important features:
1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a group; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

In clogging attacks, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can clog the victim's system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie-Hellman key exchange. The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value. Oakley supports the use of different **groups** for the Diffie-Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. Oakley employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use. Three different authentication methods can be used with Oakley are digital signatures, public-key encryption and Symmetric-key encryption.

Aggressive Oakley Key Exchange

Aggressive key exchange is a technique used for exchanging the message keys and is so called because only three messages are allowed to be exchanged at any time.

$$I \rightarrow R: \quad CKY_I, OK\_KEYX, GRP, g^x, EHAO, NIDP, ID_I, ID_R, N_I, S_{KI}[ID_I \| ID_R \| N_I \| GRP \| g^x \| EHAO]$$

$$R \rightarrow I: \quad CKY_R, CKY_I, OK\_KEYX, GRP, g^y, EHAS, NIDP, ID_R, ID_I, N_R, N_I, S_{KR}[ID_R \| ID_I \| N_R \| N_I \| GRP \| g^y \| g^x \| EHAS]$$

$$I \rightarrow R: \quad CKY_I, CKY_R, OK\_KEYX, GRP, g^x, EHAS, NIDP, ID_I, ID_R, N_I, N_R, S_{KI}[ID_I \| ID_R \| N_I \| N_R \| GRP \| g^x \| g^y \| EHAS]$$

Notation:

I	=	Initiator
R	=	Responder
CKY_I, CKY_R	=	Initiator, responder cookies
OK_KEYX	=	Key exchange message type
GRP	=	Name of Diffie-Hellman group for this exchange
g^x, g^y	=	Public key of initiator, responder; g^{xy} = session key from this exchange
EHAO, EHAS	=	Encryption, hash authentication functions, offered and selected
NIDP	=	Indicates encryption is not used for remainder of this message
ID_I, ID_R	=	Identifier for initiator, responder
N_I, N_R	=	Random nonce supplied by initiator, responder for this exchange
$S_{KI}[X], S_{KR}[X]$	=	Indicates the signature over X using the private key (signing key) of initiator, responder

Example of Aggressive Oakley Key Exchange

In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms. When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce for this exchange. Finally, R

appends a signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

ISAKMP

ISAKMP defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, ISAKMP defines payloads for exchanging key generation and authentication data.

ISAKMP Header Format

An ISAKMP message consists of an ISAKMP header followed by one or more payloads and must follow UDP transport layer protocol for its implementation. The header format of an ISAKMP header is shown below:

(a) ISAKMP header
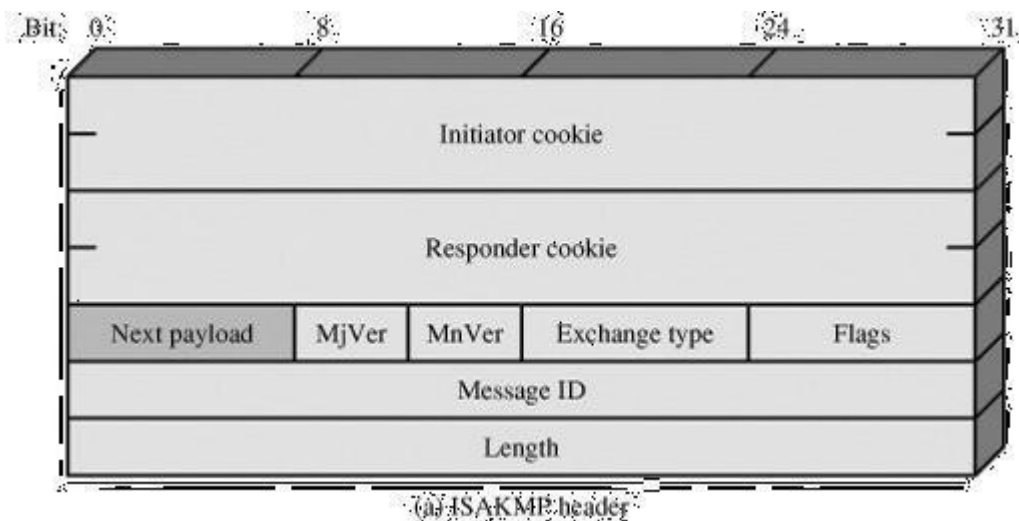
☐ (64 bits): Cookie of entity that initiated SA establishment, SA notification, or SA deletion.

☐ (64 bits): Cookie of responding entity; null in first message from initiator.

☐ (8 bits): Indicates the type of the first payload in the message

☐ (4 bits): Indicates major version of ISAKMP in use.

☐ (4 bits): Indicates minor version in use.

☐ (8 bits): Indicates the type of exchange. Can be informational, aggressive, authentication only, identity protection or base exchange (S).

☐ (8 bits): Indicates specific options set for this ISAKMP exchange. Two bits so far defined: The Encryption bit is set if all payloads following the header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to completion of SA establishment.

☐ (32 bits): Unique ID for this message.

☐ (32 bits): Length of total message (header plus all payloads) in octets.

ISAKMP Payload Types

All ISAKMP payloads begin with the same generic payload header shown below.
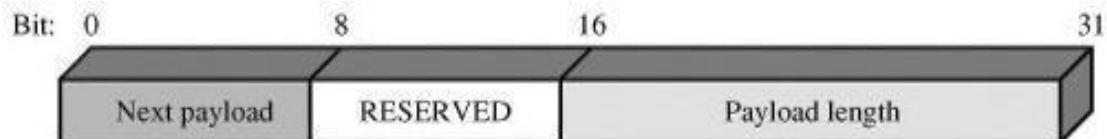
(b) Generic payload header

The Next Payload field has a value of 0 if this is the last payload in the message; otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header. There are many different ISAKMP payload types. They are:

a. The **SA payload** is used to begin the establishment of an SA. The Domain of Interpretation parameter identifies the DOI under which negotiation is taking place. The Situation parameter defines the security policy for this negotiation; in essence, the levels of security required for encryption and confidentiality are specified (e.g., sensitivity level, security compartment).

b. The **Proposal payload** contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH) for which services and mechanisms are being negotiated. The payload also includes the sending entity's SPI and the number of transforms. Each transform is contained in a transform payload.

c. The **Transform payload** defines a security transform to be used to secure the communications channel for the designated protocol. The Transform # parameter serves to identify this particular payload so that the responder may use it to indicate acceptance of this transform. The Transform-ID and Attributes fields identify a specific transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attributes (e.g., hash length).

d. The **Key Exchange payload** can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP. The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

e. The Identification payload is used to determine the identity of communicating peers and may be used for determining authenticity of information. Typically the ID Data field will contain an IPv4 or IPv6 address.

f. The **Certificate payload** transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which may include SPKI, ARL, CRL, PGP info etc. At any point in an ISAKMP exchange, the sender may include a Certificate Request payload to request the certificate of the other communicating entity.

g. The **Hash payload** contains data generated by a hash function over some part of the message and/or ISAKMP state. This payload may be used to verify the integrity of the data in a message or to authenticate negotiating entities.

h. The **Signature payload** contains data generated by a digital signature function over some part of the message and/or ISAKMP state. This payload is used to verify the integrity of the data in a message and may be used for nonrepudiation services.

i. The **Nonce payload** contains random data used to guarantee liveness during an exchange and protect against replay attacks.

j. The **Notification payload** contains either error or status information associated with this SA or this SA negotiation. Some of the ISAKMP error messages that have been defined are Invalid Flags, Invalid Cookie, Payload Malformed etc

k. The **Delete payload** indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

133

ISAKMP Exchanges

ISAKMP provides a framework for message exchange, with the payload types serving as the building blocks. The specification identifies five default exchange types that should be supported.

1. *Base Exchange*: allows key exchange and authentication material to be transmitted together. This minimizes the number of exchanges at the expense of not providing identity protection.

(a) Base Exchange

(1)I$\longrightarrow$R: SA; NONCE	Begin ISAKMP-SA negotiation
(2)R$\longrightarrow$E: SA; NONCE	Basic SA agreed upon
(3)I$\longrightarrow$R: KE; ID$_I$ AUTH	Key generated; Initiator identity verified by responder
(4)R$\longrightarrow$E: KE; ID$_R$ AUTH	Responder identity verified by initiator; Key generated; SA established

The first two messages provide cookies and establish an SA with agreed protocol and transforms; both sides use a nonce to ensure against replay attacks. The last two messages exchange the key material and user IDs, with an authentication mechanism used to authenticate keys, identities, and the nonces from the first two messages.

2. *Identity Protection Exchange:* expands the Base Exchange to protect the users' identities.

(b) Identity Protection Exchange

(1)I$\longrightarrow$R: SA	Begin ISAKMP-SA negotiation
(2)R$\longrightarrow$E: SA	Basic SA agreed upon
(3)I$\longrightarrow$R: KE; NONCE	Key generated
(4)R$\longrightarrow$E: KE; NONCE	Key generated
(5)*I$\longrightarrow$R: ID$_I$; AUTH	Initiator identity verified by responder
(6)*R$\longrightarrow$E: ID$_R$; AUTH	Responder identity verified by initiator; SA established

The first two messages establish the SA. The next two messages perform key exchange, with nonces for replay protection. Once the session key has been computed, the two parties exchange encrypted messages that contain authentication information, such as digital signatures and optionally certificates validating the public keys.

3. *Authentication Only Exchange*: used to perform mutual authentication, without a key exchange

(c) Authentication Only Exchange

(1) I ⟶ R: SA; NONCE — Begin ISAKMP-SA negotiation

(2) R ⟶ E: SA; NONCE; ID_R; AUTH — Basic SA agreed upon; Responder identity verified by initiator

(3) I ⟶ R: ID_I; AUTH — Initiator identity verified by responder; SA established

The first two messages establish the SA. In addition, the responder uses the second message to convey its ID and uses authentication to protect the message. The initiator sends the third message to transmit its authenticated ID.

4. *Aggressive Exchange*: minimizes the number of exchanges at the expense of not providing identity protection.

(d) Aggressive Exchange

(1) I ⟶ R: SA; KE; NONCE; ID_I; — Begin ISAKMP-SA negotiation and key exchange

(2) R ⟶ E: SA; KE; NONCE; ID_R; AUTH — Initiator identity verified by responder; Key generated; Basic SA agreed upon

(3) *I ⟶ R: AUTH — Responder identity verified by initiator; SA established

In the first message, the initiator proposes an SA with associated offered protocol and transform options. The initiator also begins the key exchange and provides its ID. In the second message, the responder indicates its acceptance of the SA with a particular protocol and transform, completes the key exchange, and authenticates the transmitted information. In the third message, the initiator transmits an authentication result that covers the previous information, encrypted using the shared secret session key.

5. *Informational Exchange*: used for one-way transmittal of information for SA management.

**(e) Informational Exchange**

(1) *I ⟶ R: N/D — Error or status notification, or deletion

IPv4 Header

IPv6 Header
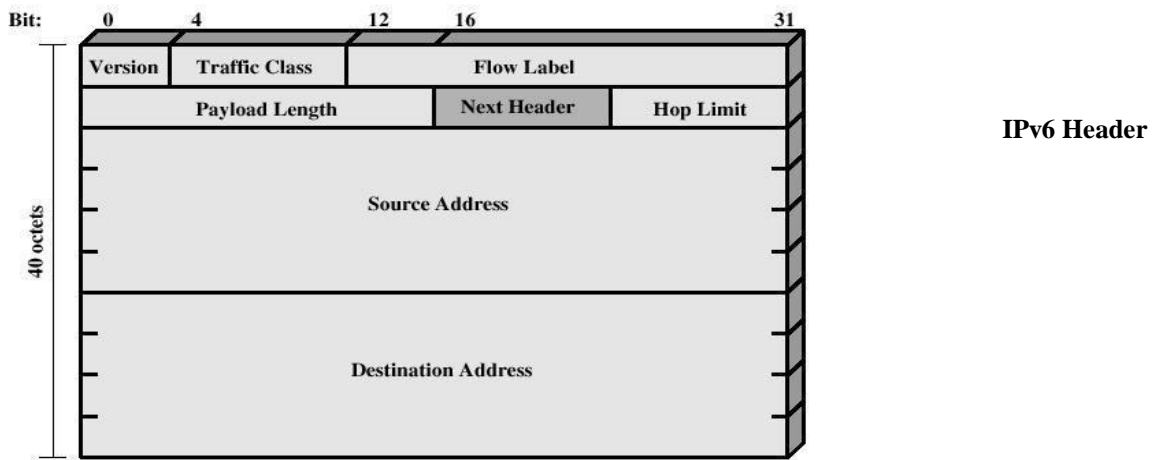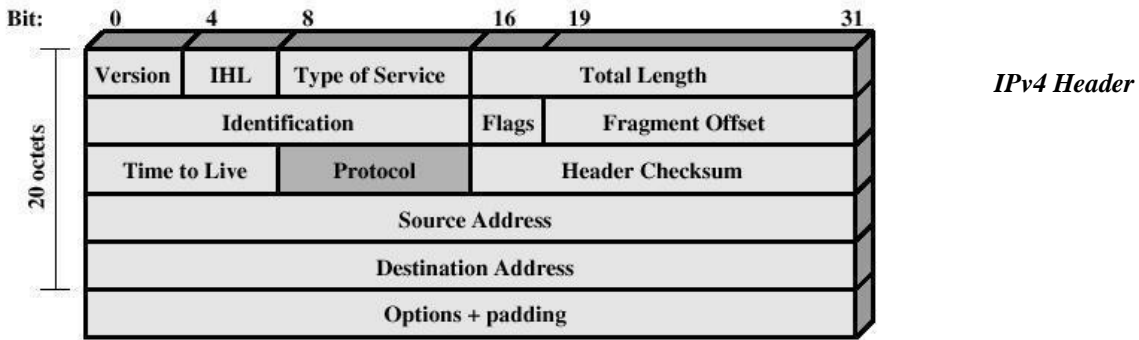
An internet protocol (IP) provides the functionality for interconnecting end systems across multiple networks. For this purpose, IP is implemented in each end system and in routers, which are devices that provide connection between networks. Higher-level data at a source end system are encapsulated in an IP protocol data unit (PDU) for transmission. This PDU is then passed through one or more networks and connecting routers to reach the destination end system.
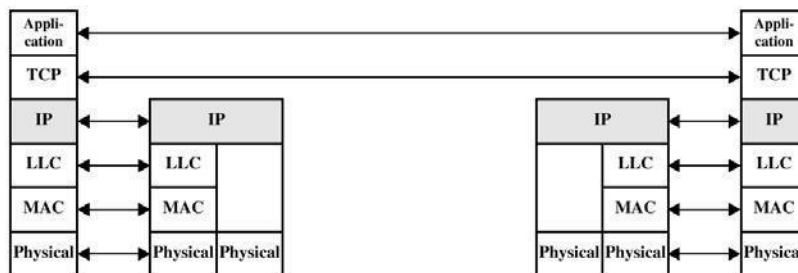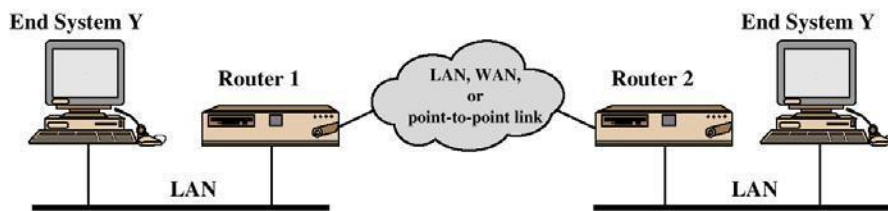
TCP/IP Example

UNIT – VI

Web security requirements, secure socket layer (SSL) and transport layer security (TLS), secure electronic transaction (SET)

Usage of internet for transferring or retrieving the data has got many benefits like speed, reliability, security etc. Much of the Internet's success and popularity lies in the fact that it is an open global network. At the same time, the fact that it is open and global makes it not very secure. The unique nature of the Internet makes exchanging information and transacting business over it inherently dangerous. The faceless, voiceless, unknown entities and individuals that share the Internet may or may not be who or what they profess to be. In addition, because the Internet is a global network, it does not recognize national borders and legal jurisdictions. As a result, the transacting parties may not be where they say they are and may not be subject to the same laws or regulations.

For the exchange of information and for commerce to be secure on any network, especially the Internet, a system or process must be put in place that satisfies requirements for confidentiality, access control, authentication, integrity, and nonrepudiation. These requirements are achieved on the Web through the use of encryption and by employing digital signature technology. There are many examples on the Web of the practical application of encryption. One of the most important is the SSL protocol.

A summary of types of security threats faced in using the Web is given below:

	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the Net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus threats • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

A Comparison of threats on the web

....

One way of grouping the security threats is in terms of passive and active attacks. *Passive attacks* include eavesdropping on network traffic between browser and server and gaining access to information on a website that is supposed to be restricted. *Active attacks* include impersonating another user, altering messages in transit between client and server and altering information on a website. Another way of classifying these security threats is in terms of location of the threat: Web server, Web browser and network traffic between browser and server.

Web Traffic Security Approaches

Various approaches for providing Web Security are available, where they are similar in the services they provide and also similar to some extent in the mechanisms they use. They differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack. The main approaches are IPSec, SSL or TLS and SET.
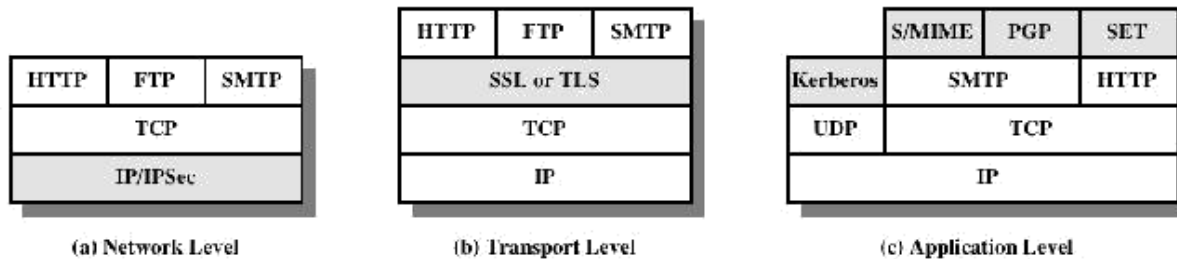

Relative location of Security Faculties in the TCP/IP Protocol Stack

IPSec provides security at the network level and the main advantage is that it is transparent to end users and applications. In addition, IPSec includes a filtering capability so that only selected traffic can be processed. **Secure Socket Layer or Transport Layer Security (SSL/TLS)** provides security just above the TCP at transport layer. Two implementation choices are present here. Firstly, the SSL/TLS can be implemented as a part of TCP/IP protocol suite, thereby being transparent to applications. Alternatively, SSL can be embedded in specific packages like SSL being implemented by Netscape and Microsoft Explorer browsers. **Secure Electronic Transaction (SET)** approach provides application-specific services i.e., according to the security requirements of a particular application. The main advantage of this approach is that service can be tailored to the specific needs of a given application.

SSL was developed by Netscape to provide security when transmitting information on the Internet. The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP).

SSL runs above TCP/IP and below high-level application protocols
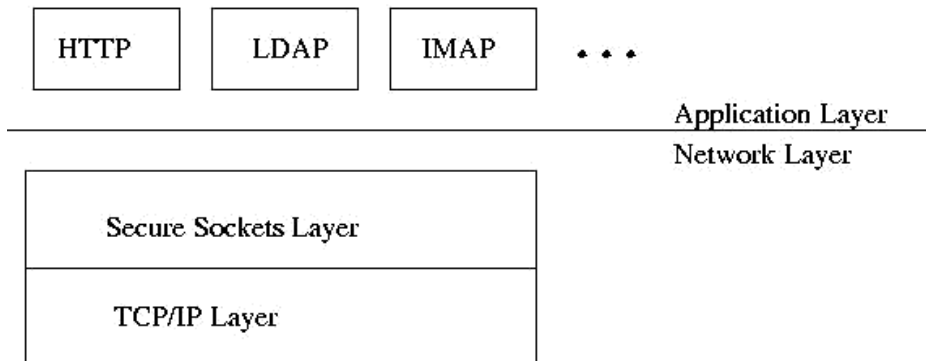
SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy. SSL protocol has different versions such as SSLv2.0, SSLv3.0, where SSLv3.0 has an advantage with the addition of support for certificate chain loading. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard. SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol, but rather two layers of protocols as shown below:
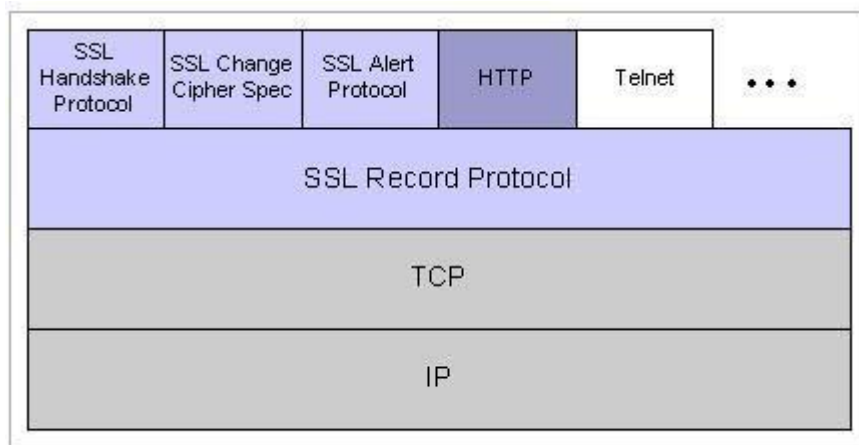
SSL Protocol Stack

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are

defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

- **Connection**: A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session**: An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

An SSL session is *stateful*. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states. An SSL session may include multiple secure connections; in addition, parties may have multiple simultaneous sessions.

A **session state** is defined by the following parameters:

- *Session identifier*: An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- *Peer certificate*: An X509.v3 certificate of the peer. This element of the state may be null.
- *Compression method*: The algorithm used to compress data prior to encryption.
- *Cipher spec*: Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- *Master secret*: 48-byte secret shared between the client and server.
- *Is resumable*: A flag indicating whether the session can be used to initiate new connections.

A **connection state** is defined by the following parameters:

- *Server and client random*: Byte sequences that are chosen by the server and client for each connection.
- *Server write MAC secret*: The secret key used in MAC operations on data sent by the server.
- *Client write MAC secret*: The secret key used in MAC operations on data sent by the client.
- *Server write key*: The conventional encryption key for data encrypted by the server and decrypted by the client.
- *Client write key*: The conventional encryption key for data encrypted by the client and decrypted by the server.
- *Initialization vectors*: When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.

- *Sequence numbers*: Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64}-1$.

SSL Record Protocol

The SSL Record Protocol provides two services for SSL connections:

- **Confidentiality**: The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- **Message Integrity**: The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users. The overall operation of the SSL Record Protocol is shown below:
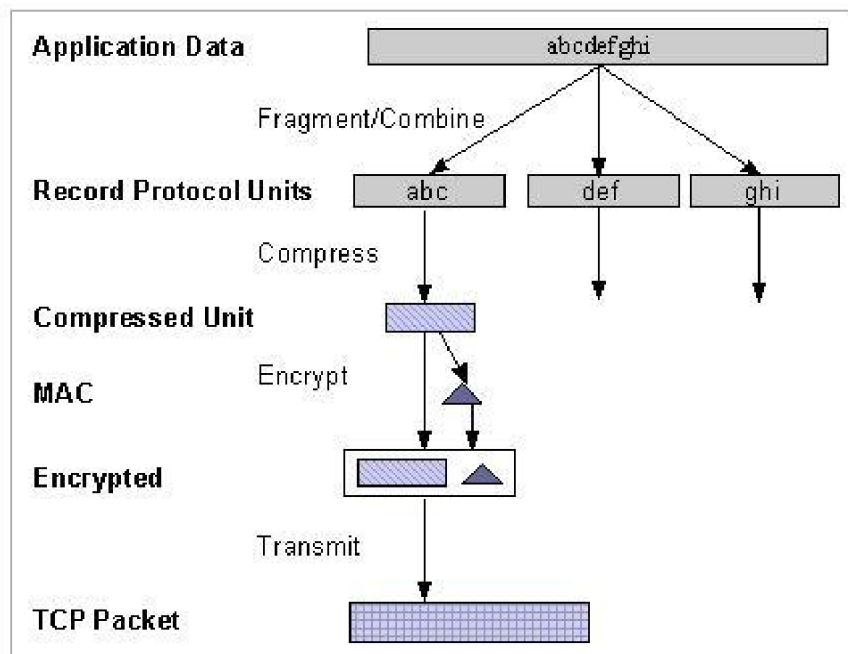
The first step is fragmentation. Each upper-layer message is fragmented into blocks of 2^{14} bytes (16384 bytes) or less. Next, compression is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes. The next step in processing is to compute a message authentication code over the compressed data. For this purpose, a shared secret key is used. The calculation is defined as:

hash(MAC_write_secret || pad_2 ||

       hash(MAC_write_secret || pad_1 || seq_num ||

       SSLCompressed.type ||

   SSLCompressed.length || SSLCompressed.fragment))   Where,

MAC_write_secret =   Secret shared key

pad_1   =   the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times for

pad_2   =   the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1

The main   difference between HMAC and above calculation is that   the two pads are concatenated in SSLv3 and are XORed in HMAC. Next, the compressed message plus the MAC are encrypted using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed $2^{14} + 2048$. The encryption algorithms allowed are AES-128/256, IDEA-128, DES-40, 3DES-168, RC2-40, Fortezza, RC4-40 and RC4-128. For stream encryption, the compressed message plus the MAC are encrypted whereas, for block encryption, padding may be added after the MAC prior to encryption.

SSL Record Format
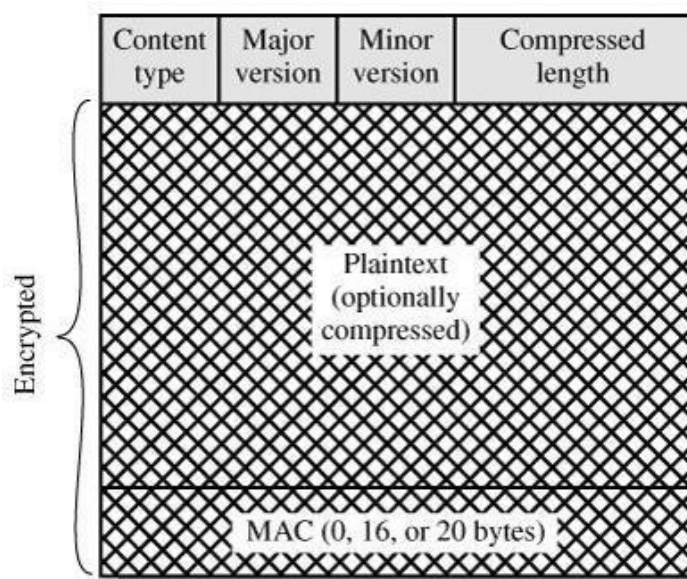
The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

- Content Type (8 bits): The higher layer protocol used to process the enclosed fragment.
- Major Version (8 bits): Indicates major version of SSL in use. For SSLv3, the value is 3.
- Minor Version (8 bits): Indicates minor version in use. For SSLv3, the value is 0.
- Compressed Length (16 bits): The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is $2^{14} + 2048$.

The content types that have been defined are change_cipher_spec, alert, handshake, and application_data.

SSL Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message, which consists of a single byte with the value 1.
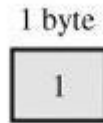
(a) Change Cipher Spec Protocol

The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

SSL Alert Protocol

The Alert Protocol is used to convey SSL-related alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state. Each message in this protocol consists of two bytes.

(b) Alert Protocol

The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. The second byte contains a code that indicates the specific alert. The fatal alerts are listed below

- **unexpected_message**: An inappropriate message was received.
- **bad_record_mac**: An incorrect MAC was received.
- **decompression_failure**: The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure**: Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter**: A field in a handshake message was out of range or inconsistent with other fields.

The remainder of the alerts are given below:

- **close_notify**: Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.
- **no_certificate**: May be sent in response to a certificate request if no appropriate certificate is available.

- **bad_certificate**: A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate**: The type of the received certificate is not supported.
- **certificate_revoked**: A certificate has been revoked by its signer.
- **certificate_expired**: A certificate has expired.
- **certificate_unknown**: Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

SSL Handshake Protocol

SSL Handshake protocol ensures establishment of reliable and secure session between client and server and also allows server & client to:
- authenticate each other
- to negotiate encryption & MAC algorithms
- to negotiate cryptographic keys to be used

The Handshake Protocol consists of a series of messages exchanged by client and server. All of these have the format shown below and each message has three fields:

(c) Handshake Protocol

- **Type (1 byte)**: Indicates one of 10 messages.
- **Length (3 bytes):** The length of the message in bytes.
- **Content (>=0 bytes):** The parameters associated with this message

The following figure shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.in phases

- o Establish Security Capabilities
- o Server Authentication and Key Exchange
- o Client Authentication and Key Exchange
- o Finish

Phase 1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a client_hello message with the following parameters:

- Version: The highest SSL version understood by the client.
- Random: A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.

145

5. Session ID: A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.
6. CipherSuite: This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.
7. Compression Method: This is a list of the compression methods the client supports.
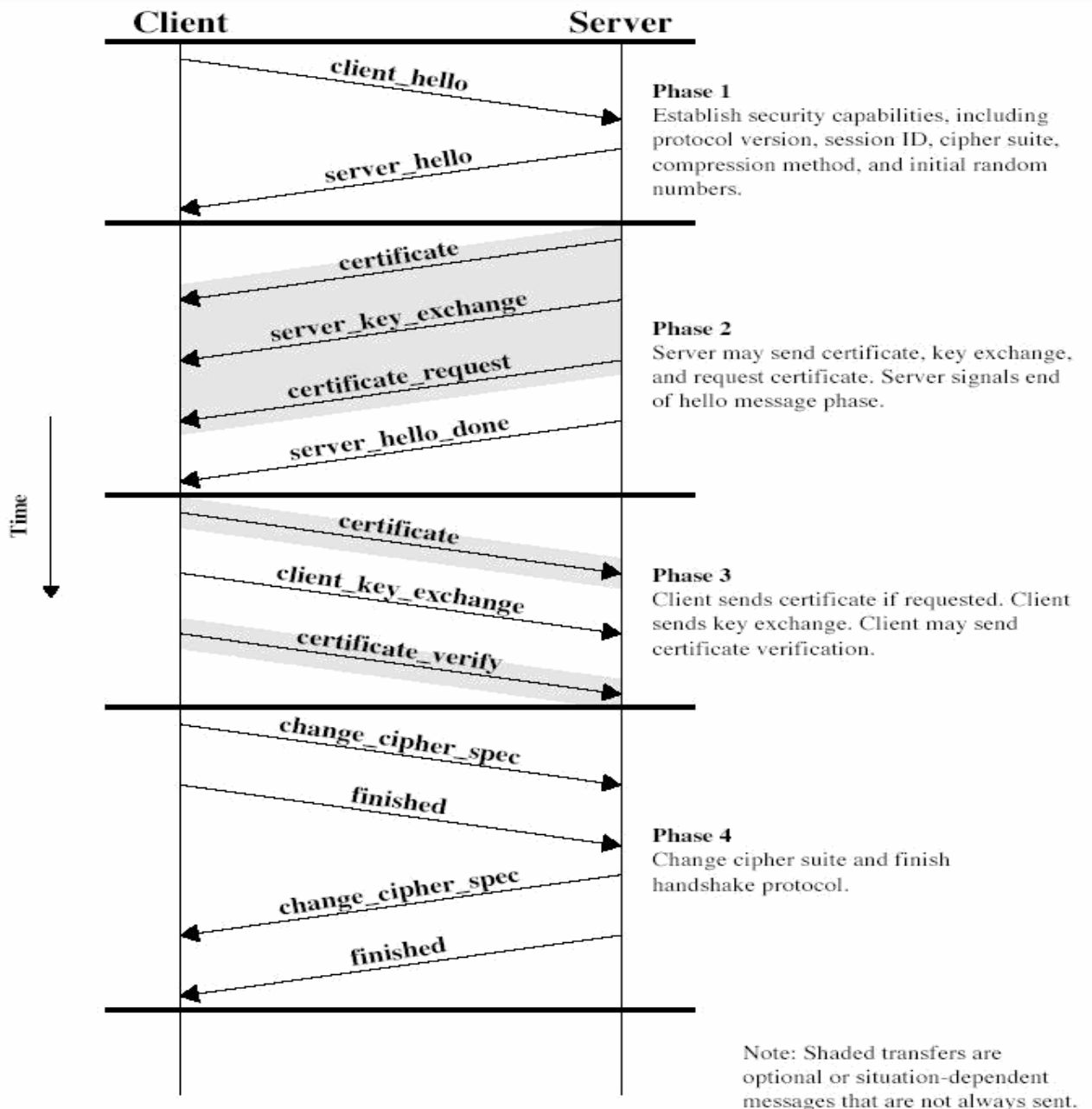
Handshake Protocol Action

After sending the client_hello message, the client waits for the server_hello message, which contains the same parameters as the client_hello message. For the server_hello message, the following conventions apply. The Version field contains the lower of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client. The first element of Cipher Suite parameter is key exchange method and the selected methods are:

- **RSA:** Secret key is encrypted with receivers RSA public key and a public key certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman**: This method produces a fixed secret key between two peers based on the Diffie-Hellman calculation using fixed public keys. Servers certificate contains the D-H public parameters signed by CA and clients public key parameters are provided either in a certificate or in a key exchange message.
- **Ephemeral Diffie- Hellman:** Temporary, One-time secret keys are generated using D-H scheme where public keys are exchanged signed using senders private RSA of DSS key. Most secure as temporary, authenticated keys are generated
- **Anonymous Diffie- Hellman:** The base D-H algorithm is used with no authentication. This is vulnerable to man-in-the-middle attack.
- **Fortezza**: the fortezza parameters are defined for both client and server

After the key exchange method, is the CipherSpec, which includes the following fields:

- **CipherAlgorithm**: Any of the algorithms: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- **MACAlgorithm**: MD5 or SHA-1
- **CipherType**: Stream or Block
- **IsExportable**: True or False
- **HashSize**: 0, 16 (for MD5), or 20 (for SHA-1) bytes
- **Key Material**: A sequence of bytes that contain data used in generating the write keys
- **IV Size**: The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

Phase 2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate via a certificate message, which contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie-Hellman. Next, a **server_key_exchange** message may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or (2) RSA key exchange is to be used. The server_key_exchange message is needed for the following:

4. Anonymous Diffie-Hellman: The message content consists of the two global Diffie-Hellman values (a prime number and a primitive root of that number) plus the server's public Diffie-Hellman key.
5. Ephemeral Diffie-Hellman: The message content includes the three Diffie-Hellman parameters provided for anonymous Diffie-Hellman, plus a signature of those parameters.
6. RSA key exchange, in which the server is using RSA but has a signature-only RSA key: The server creates a temporary RSA public/private key pair and use the server_key_exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key (exponent and modulus) plus a signature of those parameters.
7. Fortezza

Hash can be defined as **hash(ClientHello.random || ServerHello.random || ServerParams),** where hash covers not only the Diffie-Hellman or RSA parameters, but also the two nonces from the initial hello messages. This ensures against replay attacks and misrepresentation.

Next, a nonanonymous server (server not using anonymous Diffie-Hellman) can request a certificate from the client. The **certificate_request** message includes two parameters: certificate_type and certificate_authorities. The certificate type indicates the public-key algorithm and its use like RSA/Signature only, DSS/Signature only, RSA/Fixed Diffie-Hellman, DSS/Fixed Diffie-Hellman, RSA/Ephemeral DH, DSS/Ephemeral DH, Fortezza. The second parameter in the certificate_request message is a list of the distinguished names of acceptable certificate authorities. The final message in Phase 2, and one that is always required, is the **server_done** message(with no parameters), which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response.

Phase 3. Client Authentication and Key Exchange

Once the server_done message is received by client, it should verify whether a valid certificate is provided and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server. If the server has requested a certificate, the client begins this phase by sending a **certificate message** . If no suitable certificate is available, the client sends a no_certificate alert instead. Next is the **client_key_exchange** message, for which the content of the message depends on the type of key exchange, as follows:

- RSA: The client generates a 48-byte pre-master secret and encrypts with the public key from the server's certificate or temporary RSA key from a server_key_exchange message.
- Ephemeral or Anonymous Diffie-Hellman: The client's public Diffie-Hellman parameters are sent.
- Fixed Diffie-Hellman: The client's public Diffie-Hellman parameters were sent in a certificate message, so the content of this message is null.
- Fortezza: The client's Fortezza parameters are sent.

Finally, in this phase, the client may send a **certificate_verify** message to provide explicit verification of a client certificate. This message signs a hash code based on the preceding messages, defined as follows:

CertificateVerify.signature.md5_hash
        MD5(master_secret || pad_2 || MD5(handshake_messages || master_secret ||
            pad_1));
Certificate.signature.sha_hash
         SHA(master_secret || pad_2 || SHA(handshake_messages || master_secret ||

            pad_1));

handshake_messages indicate Handshake Protocol messages sent or received starting at client_hello but not including this message. If the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If it's RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. The main purpose is to verify the client's ownership of the private key for the client certificate.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a **change_cipher_spec** message and copies the pending CipherSpec into the current CipherSpec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values:

MD5(master_secret || pad2 || MD5(handshake_messages ||
     Sender || master_secret || pad1))
SHA(master_secret || pad2 || SHA(handshake_messages ||
     Sender || master_secret || pad1))

where Sender is a code that identifies that the sender is the client and handshake_messages is all of the data from all handshake messages up to but not including this message. In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point the handshake is complete and the client and server may begin to exchange application layer data.

Cryptographic Computations

It includes, the creation of a shared master secret by means of the key exchange, and the generation of cryptographic parameters from the master secret.

Master Secret Creation

The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a pre_master_secret is exchanged. Second, the master_secret is calculated by both parties. For pre_master_secret exchange, there are two possibilities: First is RSA, where pre_master_secret is generated by

client, encrypted by servers public key and then decrypted by server to recover the pre_master_secret. Second is Diffie Hellman, where the Server and Client exchange the public key information and then calculate to create the pre_master_secret. Both sides now compute the master_secret as follows:

master_secret = MD5(pre_master_secret || SHA('A' || pre_master_secret
||ClientHello.random ||
ServerHello.random)) ||
MD5(pre_master_secret || SHA('BB' ||
pre_master_secret || ClientHello.random ||
ServerHello.random)) ||
MD5(pre_master_secret || SHA('CCC' || pre_master_secret ||
ClientHello.random || ServerHello.random))

where ClientHello.random and ServerHello.random are the two nonce values exchanged in the initial hello messages.

Generation of Cryptographic Parameters

CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters. The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret:

key_block = MD5(master_secret || SHA('A' || master_secret || ServerHello.random ||
ClientHello.random)) || MD5(master_secret || SHA('BB' || master_secret ||
ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('CCC' || master_ secret ||
ServerHello.random || ClientHello.random)) || . . .

until enough output has been generated. The result of this algorithmic structure is a pseudorandom function.

Transport Layer Security

TLS was released in response to the Internet community's demands for a standardized protocol. TLS (Transport Layer Security), defined in RFC 2246, is a protocol for establishing a secure connection between a client and a server. TLS (Transport Layer Security) is capable of authenticating both the client and the server and creating a encrypted connection between the two. Many protocols use TLS (Transport Layer Security) to ~~establish secure connections, including HTTP, IMAP, POP3, and SMTP. The TLS Handshake Protocol first~~ negotiates key exchange using an asymmetric algorithm such as RSA or Diffie-Hellman. The TLS Record Protocol then begins opens an encrypted channel using a symmetric algorithm such as RC4, IDEA, DES, or 3DES. The TLS Record Protocol is also responsible for ensuring that the communications are not altered in transit. Hashing algorithms such as MD5 and SHA are used for this purpose. RFC 2246 is very similar to SSLv3. There are some minor differences ranging from protocol version numbers to generation of key material.

Version Number: The TLS Record Format is the same as that of the SSL Record Format and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the Major Version is 3 and the Minor Version is 1.

Message Authentication Code: Two differences arise one being the actual algorithm and the other being scope of MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. For TLS, the MAC calculation encompasses the fields indicated in the following expression:

HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||

   TLSCompressed.version || TLSCompressed.length ||

   TLSCompressed.fragment)

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed.

Pseudorandom Function: TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The PRF is based on the following data expansion function:

$$P\_hash(secret, seed) = HMAC\_hash(secret, A(1) \ || \ seed) \ ||$$
$$HMAC\_hash(secret, \qquad A(2) \ || \ seed) \ ||$$
$$HMAC\_hash(secret, \qquad A(3) \ || \ seed) \ || \ ...$$

where A() is defined as
A(0) = seed
A(i) = HMAC_hash (secret, A(i - 1))

The data expansion function makes use of the HMAC algorithm, with either MD5 or SHA-1 as the underlying hash function. As can be seen, P_hash can be iterated as many times as necessary to produce the required quantity of data. each iteration involves two executions of HMAC, each of which in turn involves two executions of the underlying hash algorithm.
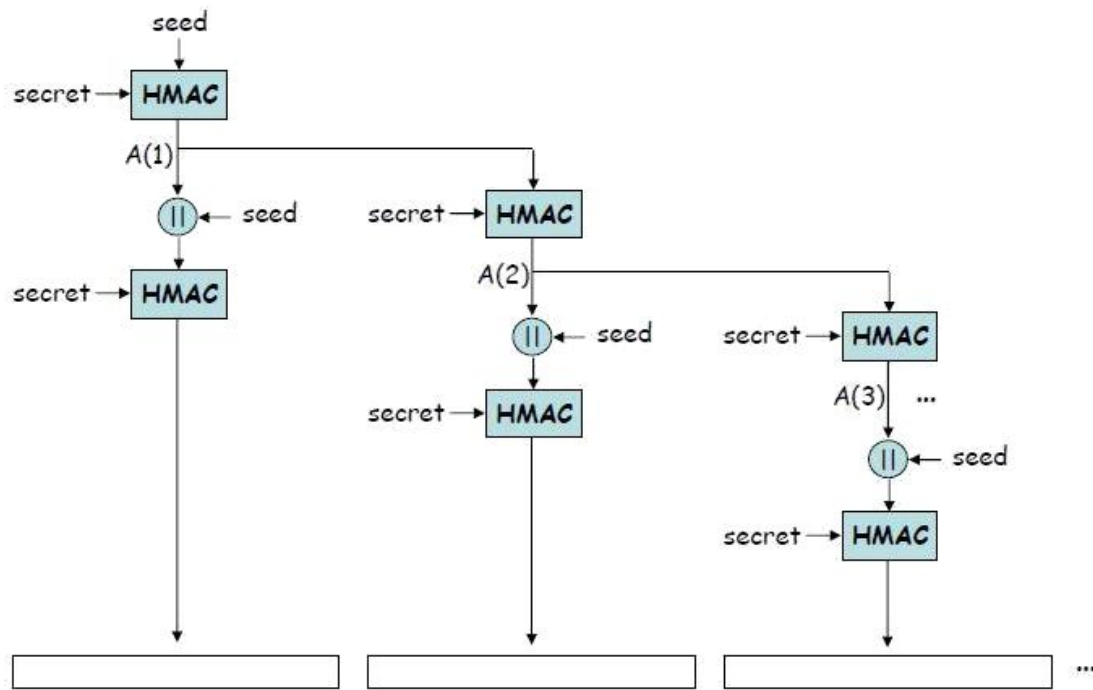
To make PRF as secure as possible, it uses two hash algorithms SHA-1 and MD5 on each half of data and then taking a XOR to produce the output. It is defined as:

PRF(secret, label, seed) = P_MD5(S1, label || seed) XOR
$$P\_SHA\text{-}1(S2, label || seed)$$

Alert Codes: TLS supports all of the alert codes defined in SSLv3 with the exception of no_certificate. A number of additional codes are defined in TLS; of these, always fatal are

- ☐ decryption_failed: A ciphertext decrypted in an invalid way; either it was not an even multiple of the block length or its padding values, when checked, were incorrect.
- ☐ record_overflow: A TLS record was received with a payload (ciphertext) whose length exceeds $2^{14} + 2048$ bytes, or the ciphertext decrypted to a length of greater than $2^{14} + 1024$ bytes.
- ☐ unknown_ca: A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- ☐ access_denied: A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- ☐ decode_error: A message could not be decoded because a field was out of its specified range or the length of the message was incorrect.
- ☐ export_restriction: A negotiation not in compliance with export restrictions on key length was detected.
- ☐ protocol_version: The protocol version the client attempted to negotiate is recognized but not supported.
- ☐ insufficient_security: Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- ☐ internal_error: An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.

The remainder of the new alerts include the following:

- □ decrypt_error: A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- □ user_canceled: This handshake is being canceled for some reason unrelated to a protocol failure.
- □ no_renegotiation: Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Cipher Suites: TLS supports all of the key exchange algorithms and also the symmetric encryption algorithms found in SSLv3 with the exception of Fortezza scheme.

Client Certificate Types: TLS defines the following certificate types to be requested in a certificate_request message: rsa_sign, dss_sign, rsa_fixed_dh, and dss_fixed_dh. These are all defined in SSLv3 along with others. Once again Fortezza scheme is not included.

Certificate_Verify and Finished messages: In case of TLS certificate_verify messages, the SHA-1 and MD5 hashes are calculated only over handshake messages, where as in SSLv3, it also includes master secret and pads. The finished message in TLS is a hash based on the shared master_secret, the previous handshake messages, and a label that identifies client or server. The calculation is somewhat different. For TLS, we have

PRF(master_secret, finished_label, MD5(handshake_messages)|| SHA-
    1(handshake_messages))
where finished_label is the string "client finished" for the client and "server finished" for the server.

Cryptographic Computations: The pre_master_secret for TLS is calculated in the same way as in SSLv3 and the calculation of master_secret is defined as:

master_secret = PRF(pre_master_secret, "master secret", ClientHello.random ||

ServerHello.random)

The algorithm is performed until 48 bytes of pseudorandom output are produced. The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is
defined as follows:
        key_block = PRF(master_secret, "key expansion",
                    SecurityParameters.server_random ||
                    SecurityParameters.client_random)
until enough output has been generated.

Padding: In SSL, the padding added prior to encryption of user data is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher's block length. In TLS, the padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes.

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. SET is not itself a payment system. Rather it is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- Provides a secure communications channel among all parties involved in a transaction
- Provides trust by the use of X.509v3 digital certificates
- Ensures privacy because the information is only available to parties in a transaction when and where necessary

SET Requirements

- Provide confidentiality of payment and ordering information
- Ensure the integrity of all transmitted data
- Provide authentication that a cardholder is a legitimate user of a credit card account
- Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution
- Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction
- Create a protocol that neither depends on transport security mechanisms nor prevents their use
- Facilitate and encourage interoperability among software and network providers

SET Key Features
To meet the requirements, SET incorporates the following features:

- Confidentiality of information
- Integrity of data
- Cardholder account authentication
- Merchant authentication

SET Participants

1. Cardholder: purchasers interact with merchants from personal computers over the Internet
2. Merchant: a person or organization that has goods or services to sell to the cardholder
3. Issuer: a financial institution, such as a bank, that provides the cardholder with the payment card.
4. Acquirer: a financial institution that establishes an account with a merchant and processes payment card authorizations and payments
5. Payment gateway: a function operated by the acquirer or a designated third party that processes merchant payment messages
6. Certification authority (CA): an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways

Events in a transaction

1. The customer obtains a credit card account with a bank that supports electronic payment and SET
2. The customer receives a X.509v3 digital certificate signed by the bank.
3. Merchants have their own certificates
4. The customer places an order
5. The merchant sends a copy of its certificate so that the customer can verify that it's a valid store
6. The order and payment are sent
7. The merchant requests payment authorization
8. The merchant confirms the order
9. The merchant ships the goods or provides the service to the customer
10. The merchant requests payment
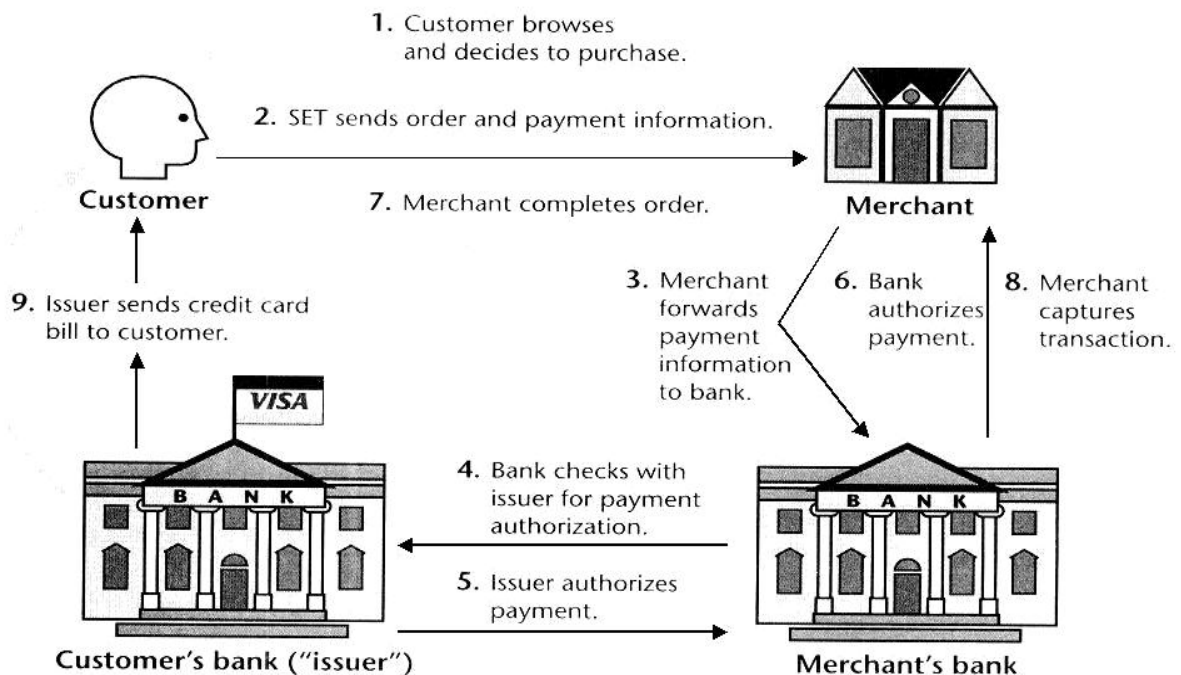
DUAL SIGNATURE

The purpose of the dual signature is to link two messages that are intended for two different recipients. The customer wants to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. The two items must be linked and the link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.
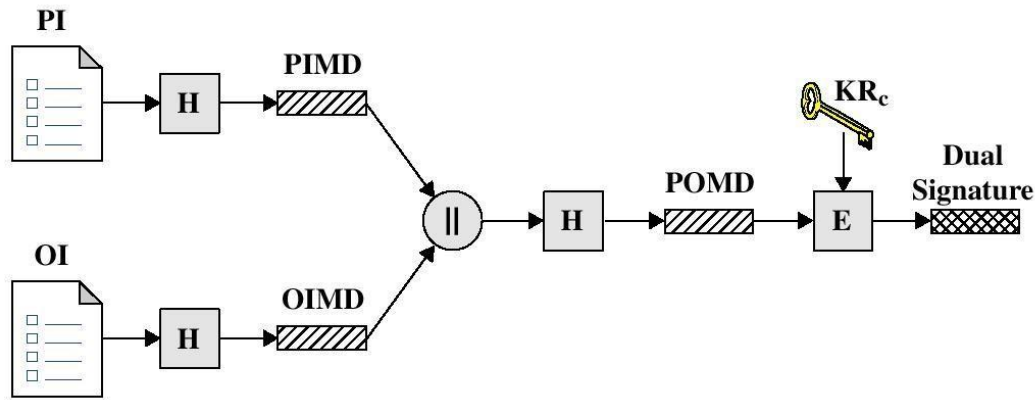
```
PI   = Payment Information        PIMD = PI message digest
OI   = Order Information          OIMD = OI message digest
H    = Hash function (SHA-1)      POMD = Payment Order message digest
||   = Concatenation              E    = Encryption (RSA)
                                  KRc  = Customer's private signature key
```

The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as

$$DS = E_{KRc} \left[H \left(H \left(PI \right) \| H(OI) \right) \right]$$

where KRc is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the quantities H(PIMS‖H[OI]) and DKUc(DS) where KUc is the customer's public signature key. If these two quantities are equal, then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute H(H[OI]‖OIMD) and DKUc(DS). Again, if these two quantities are equal, then the bank has verified the signature. To summarize:

- The merchant has received OI and verified the signature.
- The bank has received PI and verified the signature.
- The customer has linked the OI and PI and can prove the linkage.

For a merchant to substitute another OI, he has to find another OI whose hash exactly matches OIMD, which is deemed impossible. So, the OI cannot be linked with another PI.

SET Transaction Types

Cardholder registration	Authorization reversal
Merchant registration	Capture reversal
Purchase request	Credit
Payment authorization	Credit reversal
Payment capture	Payment gateway certificate request
Certificate inquiry and status	Batch administration
Purchase inquiry	Error message

<u>Purchase Request</u>

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response. The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response. In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the *Initiate_Request* message, sent to the merchant. It also includes, the brand of customers card, customer assigned ID for the request/response pair and a nonce.

The merchant generates a response and signs it with its private signature key. The Initiate_Response message includes the merchant's signature certificate, the payment gateway's key exchange certificate and a transaction ID along with the customer's nonce and merchant's nonce. The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. Next, the cardholder prepares the *Purchase_Request message* with Purchase-related information, Order-related information and customers certificate as shown below:
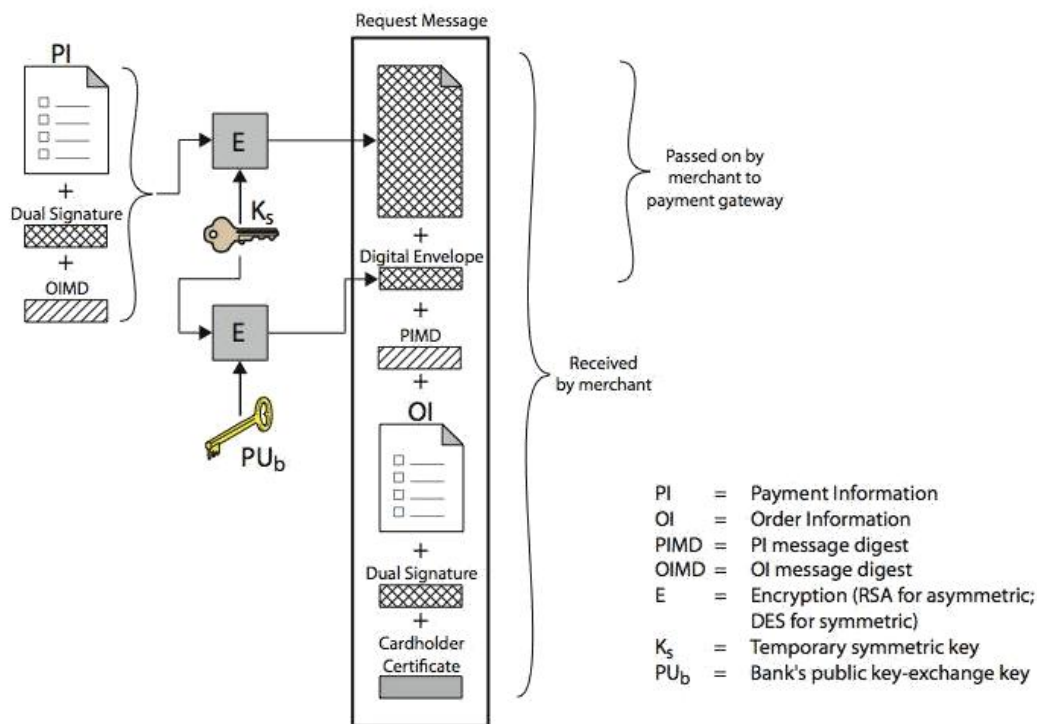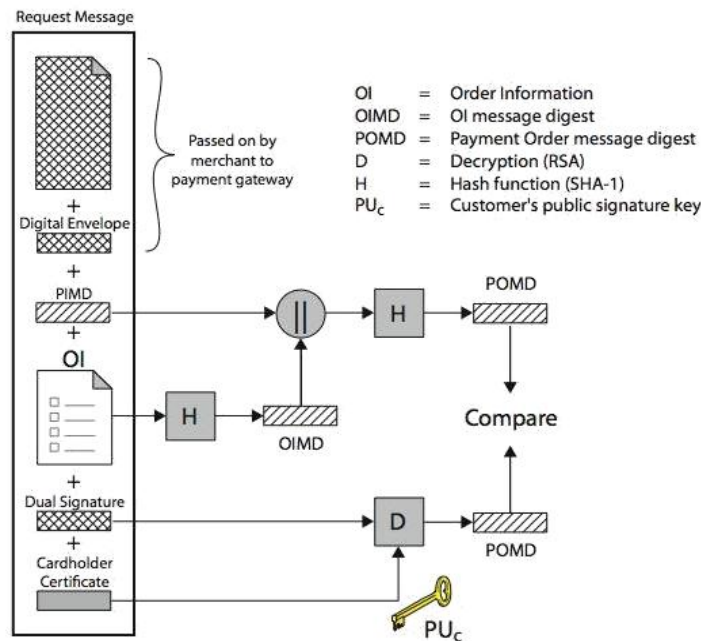

Cardholder Sends Purchase Request

The message includes the following:
1. Purchase-related information, which will be forwarded to the payment gateway by the merchant and consists of: PI, dual signature & OI message digest (OIMD). These are encrypted using Ks. A digital envelope is also present which is formed by encrypting Ks with the payment gateway's public key-exchange key.

2. Order-related information, needed by the merchant and consists of: OI, dual signature, PI message digest (PIMD). OI is sent in the clear.
3. ~~Cardholder certificate. This~~ contains the cardholder's public signature key. It is needed by the merchant and payment gateway.

Merchant receives the Purchase Request message, the following actions are done:

1. verifies cardholder certificates using CA sigs
2. verifies dual signature using customer's public signature key to ensure order has not been tampered with in transit & that it was signed using cardholder's private signature key
3. processes order and forwards the payment information to the payment gateway for authorization
4. sends a purchase response to cardholder

The Purchase Response message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate. Necessary action will be taken by cardholder's software upon verification of the certificates and signature.

Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer, guarantees the merchant will receive payment, so merchant can provide services or goods to customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response. The merchant sends an Authorization Request message to the payment gateway consisting of the following:

158

- do not protect against internal threats, eg disgruntled employee or one who cooperates with an attacker
- cannot protect against the transfer of virus-infected programs or files, given wide variety of O/S & applications supported

Types of Firewalls

Firewalls are generally classified as three types: packet filters, application-level gateways, & circuit-level gateways.

Packet-filtering Router

A packet-filtering router applies a set of rules to each incoming and outgoing IP packet to forward or discard the packet. Filtering rules are based on information contained in a network packet such as src & dest IP addresses, ports, transport protocol & interface.
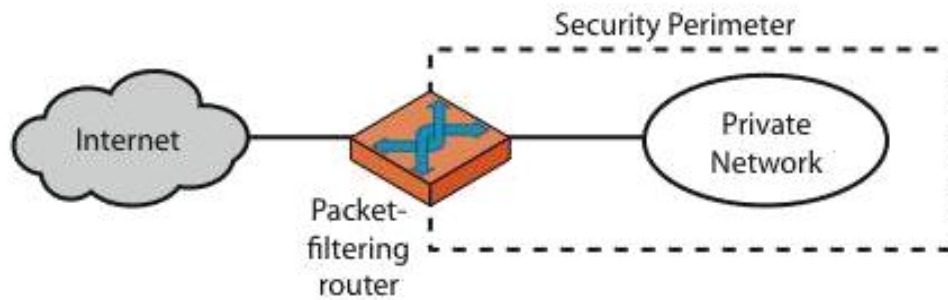
(a) Packet-filtering router

If there is no match to any rule, then one of two default policies are applied:

- that which is not expressly permitted is prohibited (default action is discard packet), conservative policy
- that which is not expressly prohibited is permitted (default action is forward packet), permissive policy

The default discard policy is more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known. One advantage of a packet-filtering router is its simplicity. Also, packet filters typically are transparent to users and are very fast.