Attacks on Computers and Computer Security: *Introduction, The need of Security, Security approaches, Principles of Security, Types of Security Attacks, Security Services, Security Mechanisms, A model for Network Security.*

Cryptography: Concepts and Techniques: *Introduction, Plain text and Cipher Text, Substitution Techniques, Transposition Techniques, Encryption and Decryption, Symmetric and Asymmetric Cryptography, Steganography, Key Range and Key Size, Possible types of Attacks.*

Introduction:

This is the age of universal electronic connectivity, where the activities like hacking, viruses, electronic fraud are very common. Unless security measures are taken, a network conversation or a distributed application can be compromised easily.

Some simple examples are:

i. Online purchases using a credit/debit card.
ii. A customer unknowingly being directed to a false website.
iii. A hacker sending a message to person pretending to be someone else.

Network Security has been affected by two major developments over the last several decades. First one is introduction of computers into organizations and the second one being introduction of distributed systems and the use of networks and communication facilities for carrying data between users & computers. These two developments lead to 'computer security' and 'network security', where the computer security deals with collection of tools designed to protect data and to thwart hackers. Network security measures are needed to protect data during transmission. But keep in mind that, it is the information and our ability to access that information that we are really trying to protect and not the computers and networks.

Why We Need Information Security?

Because there are threats:

Threats

A threat is an object, person, or other entity that represents a constant danger to an asset

The 2007 CSI survey

- 494 computer security practitioners
- 46% suffered security incidents
- 29% reported to law enforcement
- Average annual loss $350,424

- 1/5 suffered _targeted attack‘
- The source of the greatest financial losses?
- Most prevalent security problem
- Insider abuse of network access
- Email

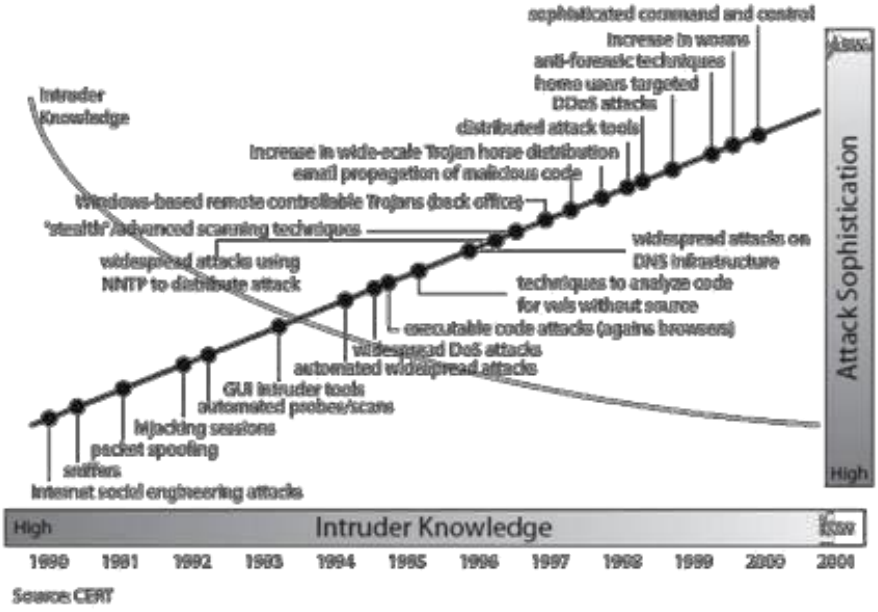
Threat Categories

- Acts of human error or failure
- Compromises to intellectual property
- Deliberate acts of espionage or trespass
- Deliberate acts of information extortion
- Deliberate acts of sabotage or vandalism
- Deliberate acts of theft
- Deliberate software attack
- Forces of nature
- Deviations in quality of service
- Technical hardware failures or errors
- Technical software failures or errors
- Technological obsolesce

Definitions

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
- **Network Security** - measures to protect data during their transmission
- **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks
- our focus is on **Internet Security**

- which consists of measures to deter, prevent, detect, and correct security violations that involve the transmission & storage of information



Aspects Of Security

consider 3 aspects of information security:

- **Security Attack**
- **Security Mechanism**
- **Security Service**

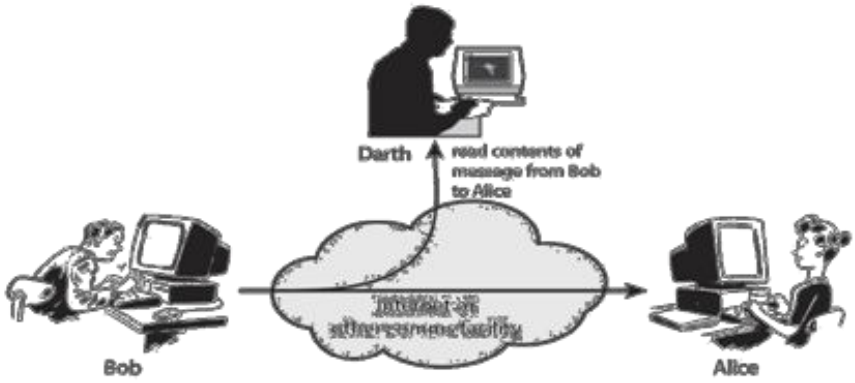
Security Attack

- any action that compromises the security of information owned by an organization
- information security is about how to prevent attacks, or failing that, to detect attacks on information-based systems
- often *threat* & *attack* used to mean same thing
- have a wide range of attacks
- can focus of generic types of attacks
    - Passive
    - Active

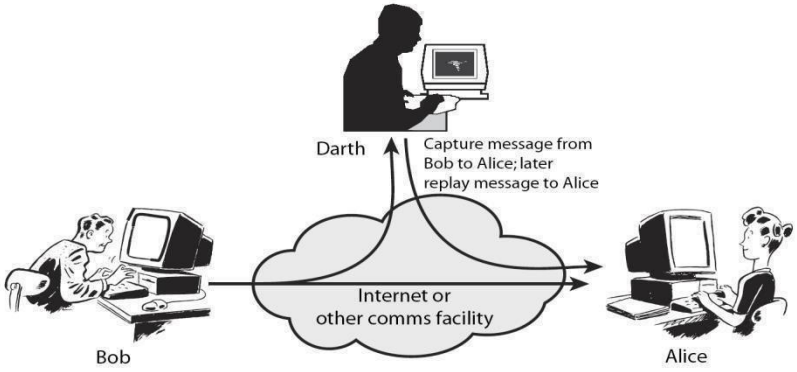


Figure 1.2 Active and Passive Security Threats

Passive Attack



Active Attack



Interruption

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

Examples:

- Destruction of some hardware
- Jamming wireless signals
- Disabling file management systems

Interception

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

Examples:

- Wire tapping to capture data in a network.
- Illicitly copying data or programs
- Eavesdropping

Modification

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

Examples:

- Changing data file
- 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:

- Hackers gaining access to a personal email and sending message
- Insertion of records in data files
- Insertion of spurious messages in a network

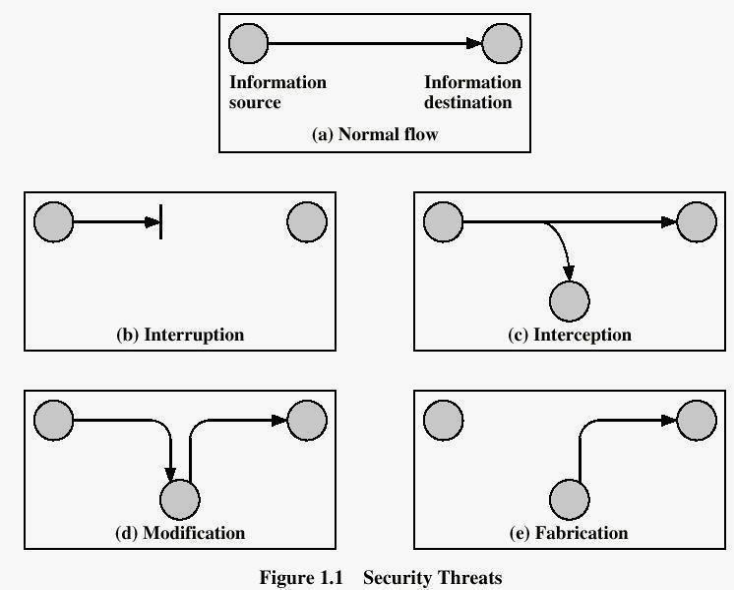


Figure 1.1 Security Threats

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. It has been defined as “ensuring that information is accessible only to those authorized to have access”.

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. For an ongoing interaction two aspects are involved. First, during connection initiation the service assures the authenticity of both parties. Second, the connection between the two hosts is not interfered allowing a third party to masquerade as one of the two parties. 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 Mediaconnectionestblishment and during data transmission. Provides confidence against a masquera or 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. Like confidentiality, it can be applied to a stream of messages, a single message or selected fields within a message. 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. Destruction of data is also covered here. Hence, it attends to both message stream modification and denial of service.

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.

Non-repudiation

Non-repudiation prevents either sender or receiver from denying a transmitted message. This capability is crucial to e-commerce. Without it an individual or entity can deny that he, she or it is responsible for a transaction, therefore not financially liable.

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. It is the ability to limit and control the access to host systems and applications via communication links. For this, each entity trying to gain access must first be identified or authenticated, so that access rights can be tailored to the individuals.

Availability

It is defined to be the property of a systemMediaorasystemresource being accessible and usable upon demand by an authorized system entity. The v ilability can significantly be affected by a variety of attacks, some amenable to automated counter measures i.e authentication and encryption and others need some sort of physical action to prevent or recover from loss of availability of elements of distributed system.

Security Mechanisms

According to X.800, the sec rity 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, whereas 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 cert in 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.

Model For Network Security

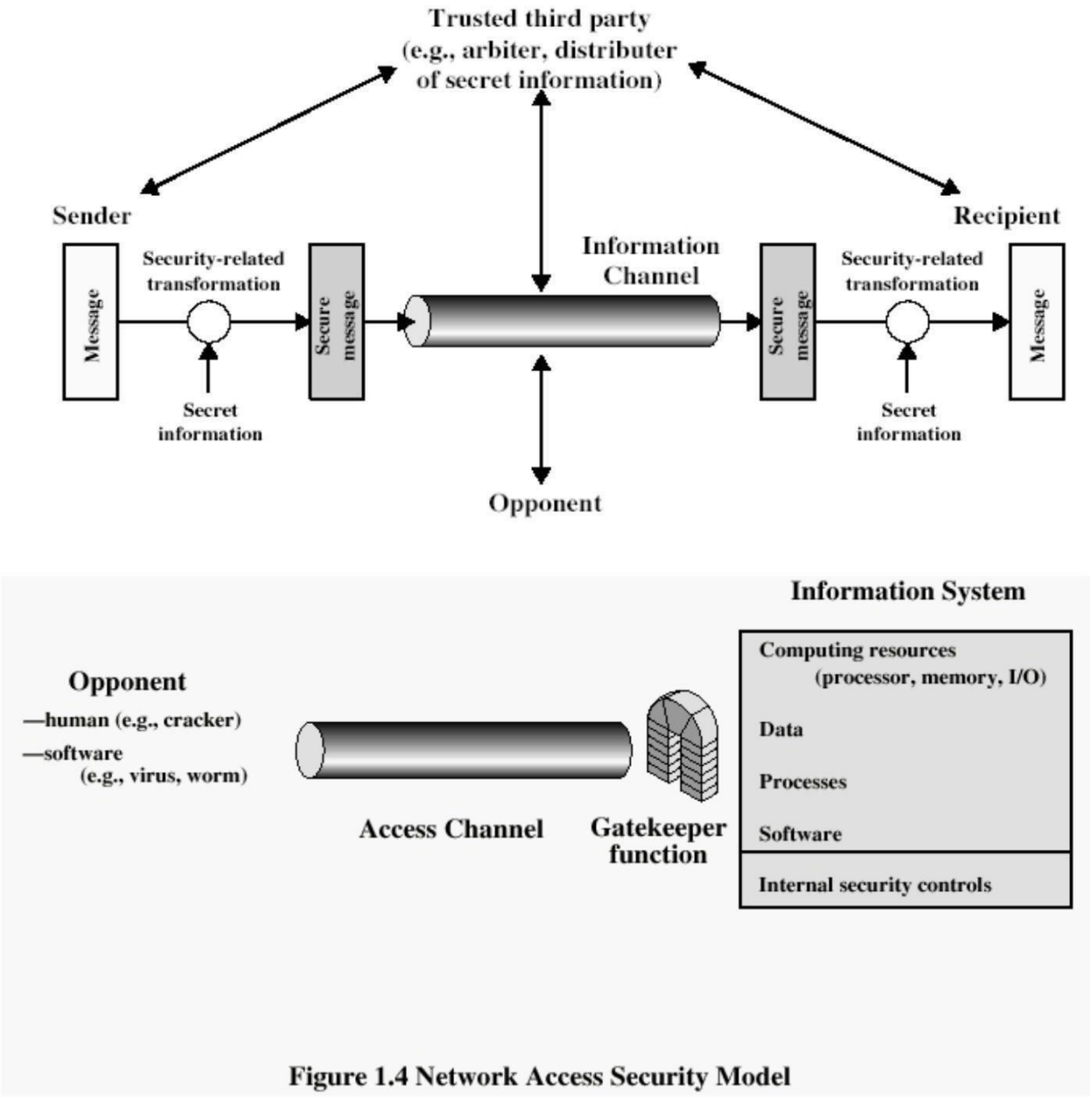


Figure 1.4 Network Access Security Model

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 almost all the security providing techniques.

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. Thealgorithm 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 various other threats to information system like unwanted access still exist.

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.

Some basic terminologies used

1. **CIPHER TEXT** - the coded message
2. **CIPHER** - algorithm for transforming plaintext to cipher text
3. **KEY** - info used in cipher known only to sender/receiver
4. **ENCIPHER (ENCRYPT)** - converting plaintext to cipher text
5. **ECIPHER (DECRYPT)** - recovering cipher text from plaintext
6. **CRYPTOGRAPHY** - study of encryption principles/methods
7. **CRYPTANALYSIS (CODEBREAKING)** - the study of principles/ methodsof deciphering cipher text *without* knowing key
8. **CRYPTOLOGY** - the field of both cryptography and cryptanalysis

Cryptography

Cryptographic systems are generally classified along 3 independent dimensions:

Type of operations used for transforming plain text to cipher text:

All the encryption algorithms are a based on two general principles: **substitution**, in which each element in the plaintext is mapped into another element, and **transposition**, in which elements in the plaintext are rearranged.

The number of keys used:

If the sender and receiver uses same key then it is s to be **symmetric key (or) single key (or) conventional encryption**. If the sender and receiver use different keys then it is said to be **public key encryption**.

The way in which the plain text is processed:

A **block cipher** processes the input and block of elements at a time, producing output block for each input block. A S**tream cipher** processes the input elements continuously, producing output element one at a time, as it goes along.

Cryptanalysis

The process of attempting to discover X or K or both is known as cryptanalysis. The strategy used by the cryptanalysis depends on the nature of the encryption scheme and the information available to the cryptanalyst. **There are various types of cryptanalytic attacks** based on the amount of information known to the cryptanalyst.

Cipher text only – A copy of cipher text alone is known to the cryptanalyst.

Known plaintext – The cryptanalyst has a copy of the cipher text and the corresponding plaintext.

Chosen plaintext – The cryptanalysts gains temporary access to the encryption machine. They cannot open it to find the key, however; they can encrypt a large number of suitably chosen plaintexts and try to use the resulting cipher texts to deduce the key.

Chosen cipher text – The cryptanalyst obtains temporary access to the decryption machine, uses it to decrypt several string of symbols, and tries to use the results to deduce the key.

Classical Encryption Techniques

There are two basic building blocks of all encryption techniques: substitution and transposition.

Substitution Techniques

In which each element in the plaintext is mapped into another element.

1. Caesar Cipher
2. Monoalphabetic cipher
3. Playfair Cipher
4. Hill Cipher
5. Polyalphabetic Cipher
6. One Time Pad

Caesar Cipher

It is a mono-alphabetic cipher wherein each letter of the plaintext is substituted by another letter to form the cipher text. It is a simplest form of substitution cipher scheme.

This cryptosystem is generally referred to as the **Shift Cipher**. The concept is to replace each alphabet by another alphabet which is 'shifted' by some fixed number between 0 and 25.

For this type of scheme, both sender and receiver agree on a 'secret shift number' for shifting the alphabet. This number which is between 0 and 25 becomes the key of encryption.

The name 'Caesar Cipher' is occasionally used to describe the Shift Cipher when the 'shift of three' is used.

Process of Shift Cipher

- In order to encrypt a plaintext letter, the sender positions the sliding ruler underneath the first set of plaintext letters and slides it to LEFT by the number of positions of the secret shift.
- The plaintext letter is then encrypted to the cipher text letter on the sliding ruler underneath. The result of this process is depicted in the following illustration for an agreed shift of three positions. In this case, the plaintext 'tutorial' is encrypted to the cipher text 'WXWRULDO'. Here is the cipher text alphabet for a Shift of 3 −

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- On receiving the cipher text, the receiver who also knows the secret shift, positions his sliding ruler underneath the cipher text alphabet and slides it to RIGHT by the agreed shift number, 3 in this case.
- He then replaces the cipher text letter by the plaintext letter on the sliding ruler underneath. Hence the cipher text 'WXWRULDO' is decrypted to 'tutorial'. To decrypt a message encoded with a Shift of 3, generate the plaintext alphabet using a shift of '-3' as shown below −

Ciphertext Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Plainrtext Alphabet	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

Security Value

Caesar Cipher is **not a secure** cryptosystem because there are only 26 possible keys to try out. An attacker can carry out an exhaustive key search with available limited computing resources.

Simple Substitution Cipher

It is an improvement to the Caesar Cipher. Instead of shifting the alphabets by some number, this scheme uses some permutation of the letters in alphabet.

For example, A.B…..Y.Z and Z.Y……B.A are two obvious permutation of all the letters in alphabet. Permutation is nothing but a jumbled up set of alphabets.

With 26 letters in alphabet, the possible permutations are 26! (Factorial of 26) which is equal to $4x10^{26}$. The sender and the receiver may choose any one of these possible permutation as a cipher text alphabet. This permutation is the secret key of the scheme.

Process of Simple Substitution Cipher

- Write the alphabets A, B, C,...,Z in the natural order.
- The sender and the receiver decide on a randomly selected permutation of the letters of the alphabet.
- Underneath the natural order alphabets, write out the chosen permutation of the letters of the alphabet. For encryption, sender replaces each plaintext letters by substituting the permutation letter that is directly beneath it in the table. This process is shown in the following illustration. In this example, the chosen permutation is K, D, G, O. The plaintext 'point' is encrypted to 'MJBXZ'.

Here is a jumbled Cipher text alphabet, where the order of the cipher text letters is a key.

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	K	D	G	F	N	S	L	V	B	W	A	H	E	X	J	M	Q	C	P	Z	R	T	Y	I	U	O

- On receiving the ciphertext, the receiver, who also knows the randomly chosen permutation, replaces each ciphertext letter on the bottom row with the corresponding plaintext letter in the top row. The ciphertext 'MJBXZ' is decrypted to 'point'.

Security Value

Simple Substitution Cipher is a considerable improvement over the Caesar Cipher. The possible number of keys is large (26!) and even the modern computing systems are not yet powerful enough to comfortably launch a brute force attack to break the system. However, the Simple Substitution Cipher has a simple design and it is prone to design flaws, say choosing obvious permutation, this cryptosystem can be easily broken.

Monoalphabetic and Polyalphabetic Cipher

Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if 'A' is encrypted as 'D', for any number of occurrence in that plaintext, 'A' will always get encrypted to 'D'.

All of the substitution ciphers we have discussed earlier in this chapter are monoalphabetic; these ciphers are highly susceptible to cryptanalysis.

Polyalphabetic Cipher is a substitution cipher in which the cipher alphabet for the plain alphabet may be different at different places during the encryption process. The next two examples, **playfair and Vigenere Cipher are polyalphabetic ciphers**.

Playfair Cipher

In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution cipher.

In playfair cipher, initially a key table is created. The key table is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I.

The sender and the receiver deicide on a particular key, say ‘tutorials’. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order. The key table works out to be −

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

Process of Playfair Cipher

- First, a plaintext message is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter. Let us say we want to encrypt the message “hide money”. It will be written as −

HI DE MO NE YZ

- The rules of encryption are −
  - If both the letters are in the same column, take the letter below each one (going back to the top if at the bottom)

T U O R I
A L S B C
D E F G H ‘H’ and ‘I’ are in same column, hence take letter below them to replace. HI → QC
K M N P Q
V W X Y Z

- If both letters are in the same row, take the letter to the right of each one (goingback to the left if at the farthest right)

T U O R I
A L S B C
D E F G H ‘D’ and ‘E’ are in same row, hence take letter to the right of them to replace. DE → EF
K M N P Q
V W X Y Z

- If neither of the preceding two rules are true, form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

T	U	O	R	I	'M' and 'O' nor on same column or same row, hence form rectangle as shown, and replace letter by picking up opposite corner letter on same row MO -> NU
A	L	S	B	C	
D	E	F	G	H	
K	M	N	P	Q	
V	W	X	Y	Z	

Using these rules, the result of the encryption of 'hide money' with the key of 'tutorials' would be −

QC EF NU MF ZV

Decrypting the Playfair cipher is as simple as doing the same process in reverse. Receiver has the same key and can create the same key table, and then decrypt any messages made using that key.

Security Value

It is also a substitution cipher and is difficult to break compared to the simple substitution cipher. As in case of substitution cipher, cryptanalysis is possible on the Playfair cipher as well, however it would be against 625 possible pairs of letters (25x25 alphabets) instead of 26 different possible alphabets.

The Playfair cipher was used mainly to protect important, yet non-critical secrets, as it is quick to use and requires no special equipment.

Vigenere Cipher

This scheme of cipher uses a text string (say, a word) as a key, which is then used for doing a number of shifts on the plaintext.

For example, let's assume the key is 'point'. Each alphabet of the key is converted to its respective numeric value: In this case,

p → 16, o → 15, i → 9, n → 14, and t → 20.

Thus, the key is: 16 15 9 14 20.

Process of Vigenere Cipher

- The sender and the receiver decide on a key. Say 'point' is the key. Numeric representation of this key is '16 15 9 14 20'.
- The sender wants to encrypt the message, say 'attack from south east'. Hewill arrange plaintext and numeric key as follows −

a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14

- He now shifts each plaintext alphabet by the number written below it to create ciphertext as shown below −

a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14
Q	I	C	O	W	A	U	A	C	G	I	D	D	H	B	U	P	B	H

- Here, each plaintext character has been shifted by a different amount – and that amount is determined by the key. The key must be less than or equal to the size of the message.
- For decryption, the receiver uses the same key and shifts received ciphertext in reverse order to obtain the plaintext.

Q	I	C	O	W	A	U	A	C	G	I	D	D	H	B	U	P	B	H
16	15	9	14	20	16	15	9	14	20	16	15	9	14	20	16	15	9	14
a	t	t	a	c	k	f	r	o	m	s	o	u	t	h	e	a	s	t

Security Value

Vigenere Cipher was designed by tweaking the standard Caesar cipher to reduce the effectiveness of cryptanalysis on the ciphertext and make a cryptosystem more robust. It is significantly **more secure than a regular Caesar Cipher**.

In the history, it was regularly used for protecting sensitive political and military information. It was referred to as the **unbreakable cipher** due to the difficulty it posed to the cryptanalysis.

Variants of Vigenere Cipher

There are two special cases of Vigenere cipher −

- The keyword length is same as plaintext message. This case is called **Vernam Cipher**. It is more secure than typical Vigenere cipher.
- Vigenere cipher becomes a cryptosystem with perfect secrecy, which is called **One-time pad**.

One-Time Pad

The circumstances are −

- The length of the keyword is same as the length of the plaintext.
- The keyword is a randomly generated string of alphabets.
- The keyword is used only once.

Security Value

Let us compare Shift cipher with one-time pad.

Shift Cipher − Easy to Break

In case of Shift cipher, the entire message could have had a shift between 1 and 25. This is a very small size, and very easy to brute force. However, with each character now having its own individual shift between 1 and 26, the possible keys grow exponentially for the message.

One-time Pad − Impossible to Break

Let us say, we encrypt the name "point" with a one-time pad. It is a 5 letter text. To break the cipher text by brute force, you need to try all possibilities of keys and conduct computation for (26 x 26 x 26 x 26 x 26) = 26^5 = 11881376 times. That's for a message with 5 alphabets. Thus, for a longer message, the computation grows exponentially with every additional alphabet. This makes it computationally impossible to break the cipher text by brute force.

Transposition Techniques

All the techniques examined so far involve the substitution of a cipher text symbol for a plaintext symbol. A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.

Rail fence is simplest of such cipher, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

Plaintext = meet at the school house

To encipher this message with a rail fence of depth 2,

We write the message as follows: m e a t e c o l o s e t t h s h o h u e

The encrypted message is MEATECOLOSETTHSHOHUE

Row Transposition Ciphers-A more complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of columns then becomes the key of the algorithm.

, plaintext = meet at the

school house Key = 4 3 1 2 5

6 7

PT = m e e t a t t h e s c h o o l h o u s e

CT = ESOTCUEEHMHLAHSTOETO

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is more complex permutation that is not easily reconstructed.

Steganography

A plaintext message may be hidden in any one of the two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text. A simple form of steganography, but one that is time consuming to construct is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message. e.g., (i) the sequence of first letters of each word of the overall message spells out the real (hidden) message. (ii) Subset of the words of the overall message is used to convey the hidden message. Various other techniques have been used historically, some of them are:

- **Character marking** – selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held to an angle to bright light.
- **Invisible ink** – a number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
- **Pin punctures** – small pin punctures on selected letters are ordinarily not visible unless the paper is held in front of the light.
- **Typewritten correction ribbon** – used between the lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Drawbacks of Steganography

- Requires a lot of overhead to hide a relatively few bits of information.
- Once the system is discovered, it becomes virtually worthless.

UNIT –2

Symmetric Key Ciphers: Block Cipher Principles and Algorithms (DES, AES, and Blowfish), Differential and Linear Cryptanalysis, Block Cipher Modes of Operations, Stream Ciphers, RC4, Location and Placement of encryption function, Key Distribution.

Asymmetric Key Ciphers: *Principles of Public Key Cryptosystems, Algorithms (RSA, Diffie-Hellman, ECC), Key Distribution.*

Conventional Encryption Principles

A Conventional/Symmetric encryption scheme has five ingredients:

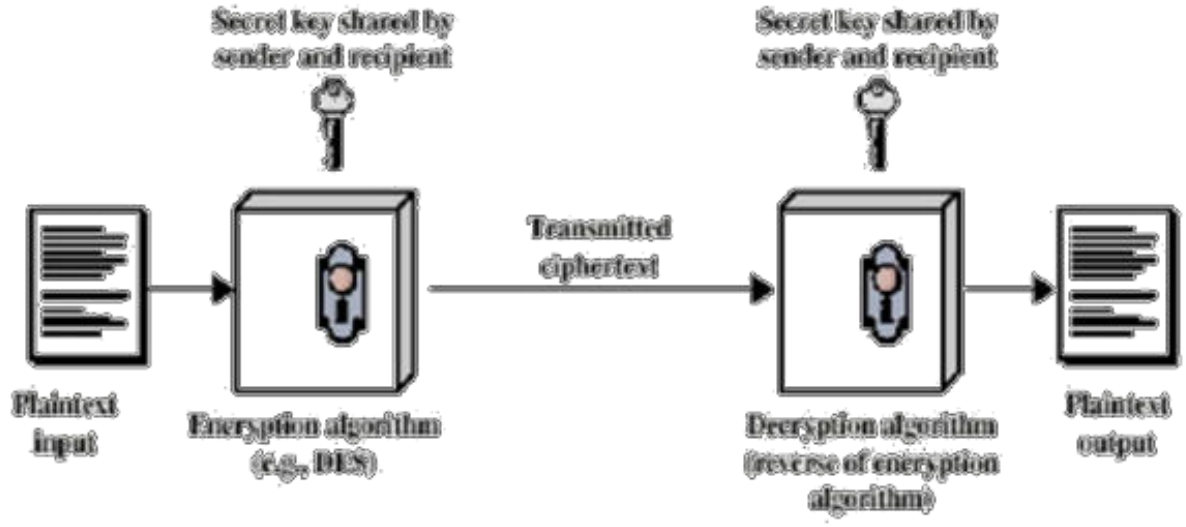
1. ***Plain Text***: This is the original message or data which is fed into the algorithm as input.

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 algor thm. The substitutions and transformations performed by algorithm depend on the key.

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.



Simplified Model of Conventional Encryption

The important point is that the security of conventional encrypt on depends on the secrecy of the key, not the secrecy of the algorithm i.e. it is not necessary to keep the algorithm secret, but only the key is to be kept secret. This feature that algorithm need not be kept secret made it feasible for wide spread use and enabled manufacturers develop low cost chip implementation of data encryption algorithms. With the use of conventional algorithm, the principal security problem is maintaining the secrecy of the key.

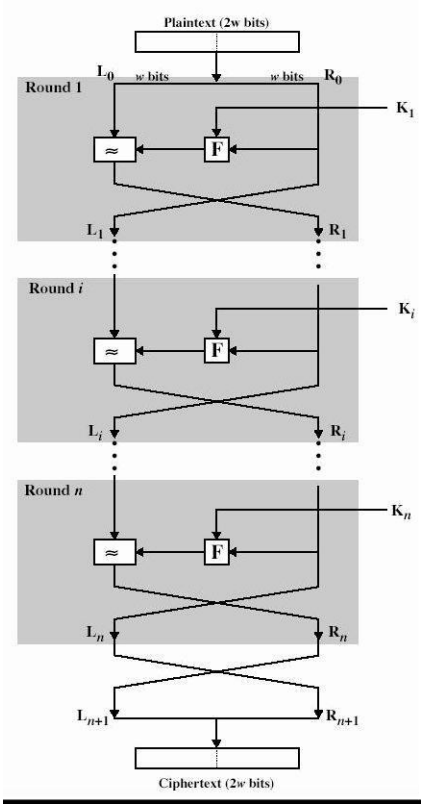
Feistel Cipher Structure

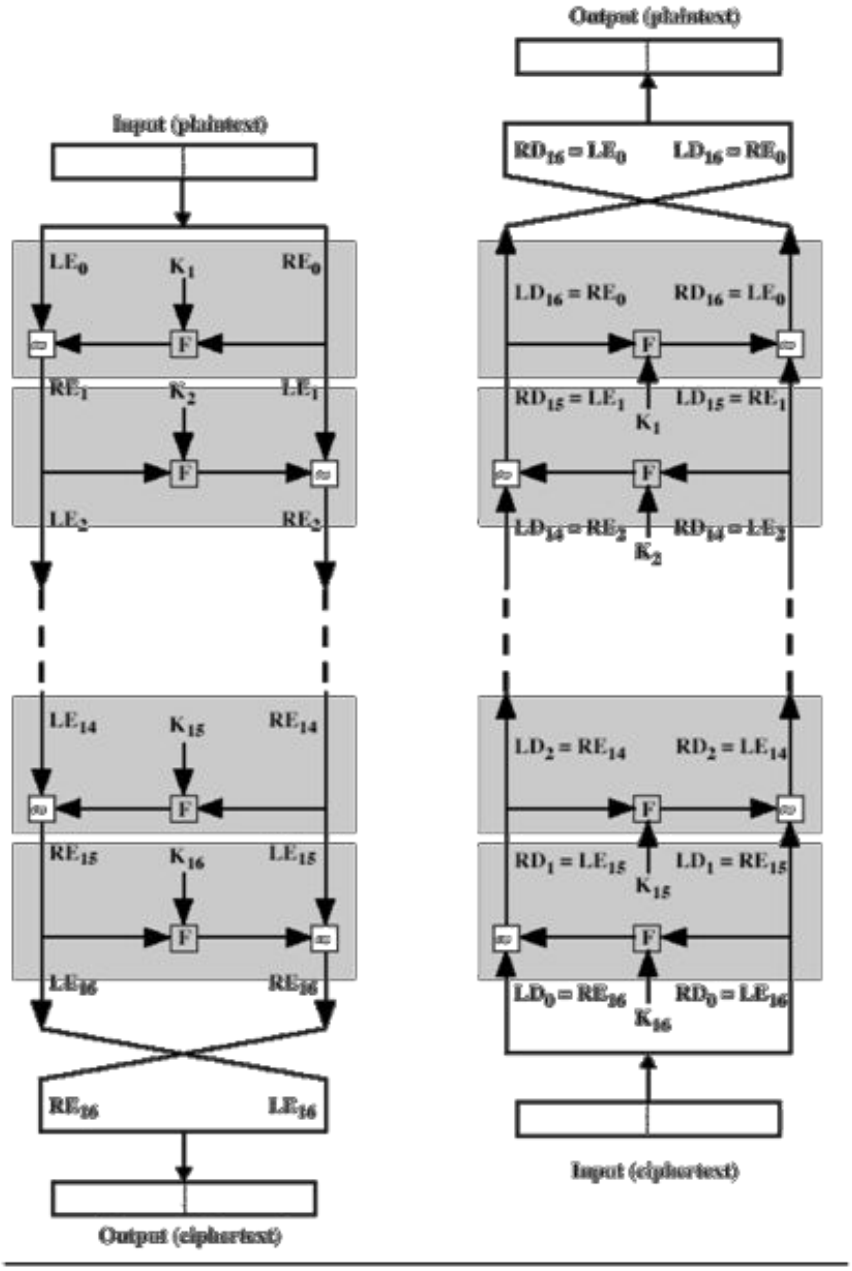
The input to the encryption algorithm are a plaintext block of length 2w bits and a key K. the plaintext block is divided into two halves L_0 and R_0. The two halves of the data pass through „n" rounds of processing and then combine to produce the cipher text block. Each round „i" has inputs L_{i-1} and R_{i-1}, derived from the previous round, as well as the subkey K_i, derived from the overall key K. in general, the subkeys K_i are different from K and from each other.

All rounds have the same structure. A substitution is performed on the left half of the data (as similar to S-DES). This is done by applying a round function F to the right half

of the data and then taking the XOR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey ki. Following this substitution, a permutation is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network. The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size** - Increasing size improves security, but slows cipher
- **Key size** - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher
- **Number of rounds** - Increasing number improves security, but slows cipher
- **Subkey generation** - Greater complexity can make analysis harder, butslows cipher
- **Round function** - Greater complexity can make analysis harder, but slows cipher
- **Fast software en/decryption & ease of analysis -** re more recent concerns for practical use and testing





The process of decryption is essentially the same as the encryption process. The rule is as follows: use the cipher text as input to the algorithm, but use the subkey k_i in reverse order. i.e., k_n in the first round, k_{n-1} in second round and so on. For clarity, we use the notation LE_i and RE_i for data traveling through the decryption algorithm. The diagram below indicates that, at each round, the intermediate value of the decryption process is same (equal) to the corresponding value of the encryption process with two halves of the value swapped.

, $RE_i \| LE_i$ (or) equivalently $RD_{16-i} \| LD_{16-i}$

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is $RE_{16} \| LE_{16}$. The output of that round is the cipher text. Now take the cipher text and use it as input to the same algorithm. The input to the first round is $RE_{16} \| LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process. Now we will see how the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process.

First consider the encryption process,

$LE_{16} = RE_{15}$

$RE_{16} = LE_{15} (+) F (RE_{15}, K_{16})$

On the decryption side, $LD_1 = RD_0 = LE_{16} = RE_{15}$

$RD_1 = LD_0 (+) F (RD_0, K_{16})$

$= RE_{16} F (RE_{15}, K_{16})$

$= [LE_{15} F (RE_{15}, K_{16})] F (RE_{15}, K_{16})$

$= LE_{15}$

Therefore, $LD_1 = RE_{15}$ $RD_1 = LE_{15}$ In general, for the i^{th} iteration of the encryption algorithm, $LE_i = RE_{i-1}$ $RE_i = LE_{i-1} F (RE_{i-1}, K_i)$

Finally, the output of the last round of the decryption process is $RE_0 \| LE_0$. A 32-bit swap recovers the original plaintext.

Definitions

Encryption: Converting a text into code or cipher.
Converting computer data and messages into something, incomprehensible use a key, so that only a holder of the matching key can reconvert them.

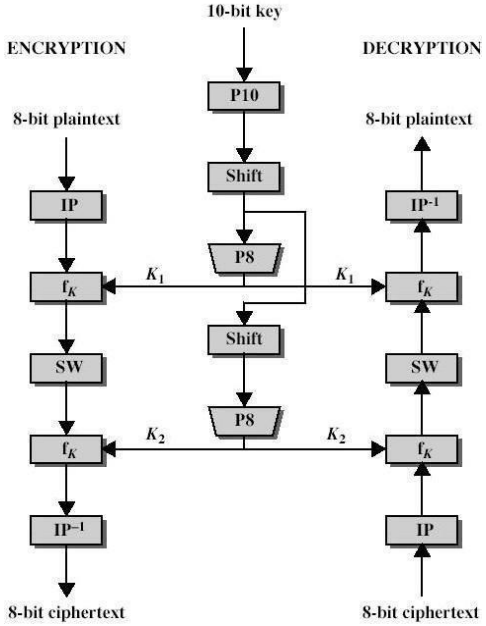
Conventional or Symmetric or Secret Key or Single Key encryption:
Uses the same key for encryption & decryption.
Public Key encryption: Uses different keys for encryption & decryption

Conventional Encryption Principles

An encryption scheme has five ingredients:

1. Plaintext – Original message or data.
2. Encryption algorithm – performs substitutions & transformations on plaintext.
3. Secret Key – exact substitutions & transformations depend on this
4. Cipher text - output ie scrambled input.
5. Decryption algorithm - converts cipher text back to plaintext.

Simplified Data Encryption Standard (S-DES)



The figure above illustrates the overall structure of the simplified DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of cipher text as output. The S-DES decryption algorithm takes an 8-bit block of cipher text and the same 10-bit key used to produce that cipher text as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function f_k again
- a permutation function that is the inverse of the initial permutation

The function f_k takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is us from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift o eration then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).

The encryption algorithm can be expressed as a composition composition1 of functions:

$$IP^{-1} \circ f_{K2} \circ SW \circ f_{k1} \circ IP$$

This can also be written as

$$Ciphertext = IP^{-1}(f_{K2}(SW(f_{k1}(IP(plaintext)))))$$

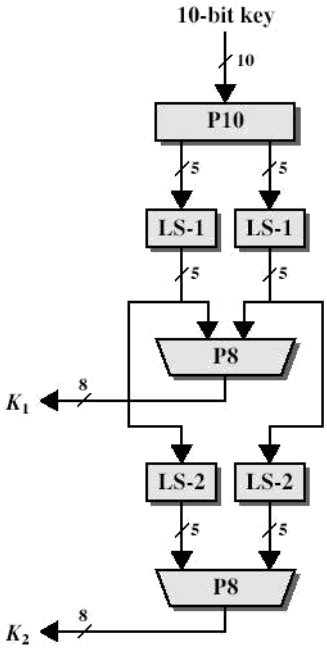
Where

K1 = P8 (Shift (P10 (Key)))

K2 = P8 (Shift (shift (P10 (Key))))

Decryption can be shown as

$$Plaintext = IP^{-1}(f_{K1}(SW(f_{k2}(IP(ciphertext)))))$$



S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use n particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10).

Then the permutation P10 is defined as:

P10 (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, K2, k7, k4, k10 10, k1, k9, k8, k6)

P10 can be concisely defined by the dis lay:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

S-DES encryption

Encryption involves the sequential application of five functions.

Initial and Final Permutations The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up.

Consider the plaintext to be 11110011.

Permuted output = 10111101

At the end of the algorithm, the inverse permutation is use :

IP –1							
4	1	3	5	7	2	8	6

The most complex Skyupscomponentof-DES is the function fk, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to f K, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let $f_k(L, R) = (L (+) F(R, SK), R)$
Where SK is a subkey and (+) is the bit-by-bit exclusive-OR function.

e.g., permuted output = 1011 1101 and suppose F (1101, SK) = (1110) for some key SK.

Then f K(10111101) = 10111110, 1101 = 01011101

We now describe the mapping F. The input is a 4-bit number (n1 n2 n3 n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

R= 1101 E/P output = 11101011 It is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey K1 = (k11, k12 12, k13 13, k14 14, k15 15, k16 16, k17 17, k18) is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits:

$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2- bit output.

These two boxes are defined as follows:
The S-boxes operate Skyups as follows. The first and fourth input bits are treated as a 2-bit

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

number that specify a row of the -box, and the second and third input bits specify a

column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if (p0,0 p0,3) =) (00) and (p0,1 p0,2) = (10), then the output is from row 0, column 2 of S0, which is 3, or (11) in) binary. Similarly, (p1,0 p1,3) and (p1,1 p1,2) are used to index into a row and column of S1 to produce an additional 2 bits. Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

The Switch Function The function f K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates

on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext

Data Encryption Standard (DES)

The main standard for encrypting data was a symmetric algorithm known as the Data Encryption Standard (DES). However, this has now been replaced by a new standard known as the Advanced Encryption Standard (AES) which we will look at later. DES is a 64 bit block cipher which means that it encrypts data 64 bits at a time. This is contrasted to a stream cipher in which only one bit at a time (or sometimes small groups of bits such as a byte) is encrypted. DES was the result of a research project set up by International Business Machines (IBM) corporation in the late 1960's which resulted in a cipher known as LUCIFER. In the early 1970's it was decided to commercialize LUCIFER and a number of significant changes were introduced. IBM was not the only one involved in these changes as they sought technical advice from the National Security Agency (NSA) (other outside consultants were involved but it is likely that the NSA were the major contributors from a technical point of view). The alt red version of LUCIFER was put forward as a proposal for the new national encryption standard requested by the National Bureau of Standards (NBS)3 . It was finally adopted in 1977 as the Data Encryption Standard - DES (FIPS PUB 46). Some of the changes made to LUCIFER have been the subject of much controversy even to the present day. The most notable of these was the key size. LUCIFER used a key size of 128 bits however this was reduced to 56 bits for DES. Even though DES actually accepts a 64 bit key as input, the remaining eight bits are used for parity checking and have no effect on DES's security. Outsiders were convinced that the 56 bit key was an easy target for a brute force attack4 due to its extremely small size. The need for the parity checking scheme was also questioned without satisfying answers. Another controversial issue was that the S-boxes used were designed under classified conditions and no reasons for their particular design were ever given. This led people to assume that the NSA had introduced a "trapdoor" through which they could decrypt any data encrypted by DES even without knowledge of the key. One startling discovery was that the S-boxes appeared to be secure against an attack known as Differential Cryptanalysis which was only publicly discovered by Biham and Shamir in 1990. This suggests that the NSA were aware of this attack in 1977; 13 years earlier! In

fact the DES designers claimed that the reason they never made the design specifications for the S-boxes available was that they knew about a number of attacks that weren't public knowledge at the time and they didn't want them leaking - this is quite a plausible claim as differential cryptanalysis has shown. However, despite all this controversy, in 1994 NIST reaffirmed DES for government use for a further five years for use in areas other than "classified". DES of course isn't the only symmetric cipher. There are many others, each with varying levels of complexity. Such ciphers include: IDEA, RC4, RC5, RC6 and the new Advanced Encryption Standard (AES). AES is an important algorithm and was originally meant to replace DES (and its more secure variant triple DES) as the standard algorithm for non-classified material. However as of 2003, AES with key sizes of 192 and 256 bits has been found to be secure enough to protect information up to top secret. Since its creation, AES had underdone intense scrutiny as one would expect for an algorithm that is to be used as the standard. To date it has withstood all attacks but the search is still on and it remains to be seen Media whetherornotthis will last. We will look at AES later in the course.

DES

DES (and most of the other major symmetric ciphers) is based on cipher known as the Feistel block cipher. It consists of a number of rounds where each round contains bit-shuffling, non-linear substitutions (S-boxes) and exclusive OR operations. As with most encryption schemes, DES expects two inputs - the plaintext to be encrypted and the secret key. The manner in which the plaintext is accepted, and the key arrangement used for encryption and decryption, both determine the type of cipher it is. DES is therefore a symmetric, 64 bit block cipher as it uses the same key for both encryption and decryption and only operates on 64 bit blocks of data at a time5 (be they plaintext or ciphertext). The key size used is 56 bits, however a 64 bit (or eight-byte) key is actually input. The least significant bit of each byte is either used for parity (odd for DES) or set arbitrarily and does not increase the security in any way. All blocks are numbered from left to right which makes the eight bit of each byte the parity bit.

Once a plain-text message is received to be encrypted, it is arranged into 64 bit blocks required for input. If the number of bits in the message is not evenly divisible by 64, then the last block will be padded. Multiple permutations and substitutions are incorporated throughout in order to increase the difficulty of performing a cryptanalysis on the cipher

Overall Structure

Figure below shows the sequence of events that occur during an encryption operation. DES performs an initial permutation on the entire 64 bit block of data. It is then split into 2, 32 bit sub-blocks, Li and Ri which are then passed into what is known as a round (see figure 2.3), of which there are 16 (the subscript i in Li and Ri indicates the current round). Each of the rounds are identical and the effectsMediaofincreasingtheir number is twofold - the algorithms security is increased and its temporal efficiency decreased. Clearly these are two conflicting outcomes and a compromise must be ma . For DES the number chosen was 16, probably to guarantee the elimination of any correlation between the cipher text and either the plaintext or key6 . At the end of the 16th round, the 32 bit Li and Ri output quantities are swapped to create what is known as the pre-output. This [R16, L16] concatenation is permuted using a function which is the exact inverse of the initial permutation. The output of this final permutation is the 64 bit cipher text.

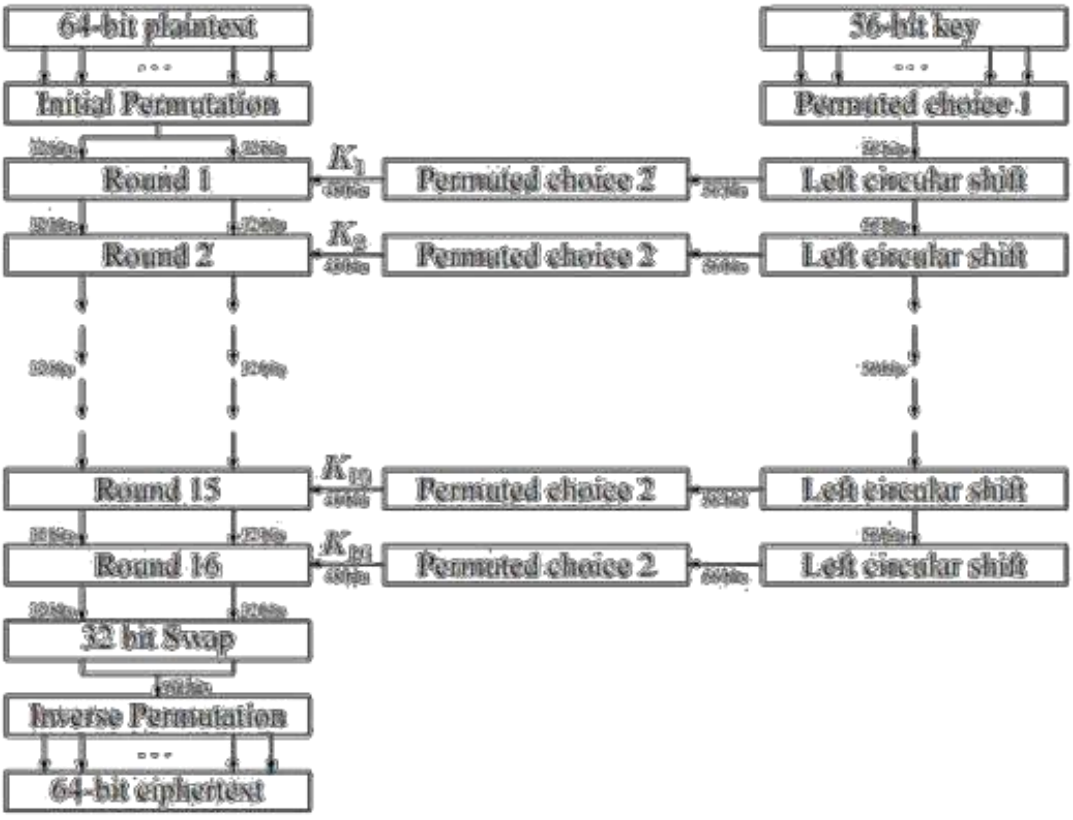


Figure : Flow Diagram of DES algorithm for encrypting data.

So in total the processing of the plaintext proceeds in three phases as can be seen from the left hand side of figure

1. Initial permutation (IP - defined in table 2.1) rearranging the bits to form the "permuted input".

2. Followed by 16 iterations of the same function (substitution and permutation). The output of the last iteration consists of 64 bits which is a function of the plaintext and key. The left and right halves are swapped to produce the pre-output.

3. Finally, the pre-output is passed through a permutation (IP−1 - defined in table 2.1) which is simply the inverse of the initial permutation (IP). The output of IP−1 is the 64-bit cipher text

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Table 2.1: Permutation tables used in DES.

As figure shows, the inputs to each round consist of the Li , Ri pair and a 48 bit subkey which is a shifted and contracted version of the original 56 bit key. The use of the key can be seen in the right hand portion of figure 2.2: • Initially the key is passed through a permutation function (PC1 - defined in table 2.2) • For each of the 16 iterations, a subkey (Ki) is produced by a combination of a left circular shift and a permutation (PC2 - defined in table 2.2) which is the same for each iteration. However, the resulting subkey is different for each iteration because of repeated shifts.

(a) Input Key

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

(d) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2.2: DES key schedule.

Details Of Individual Rounds

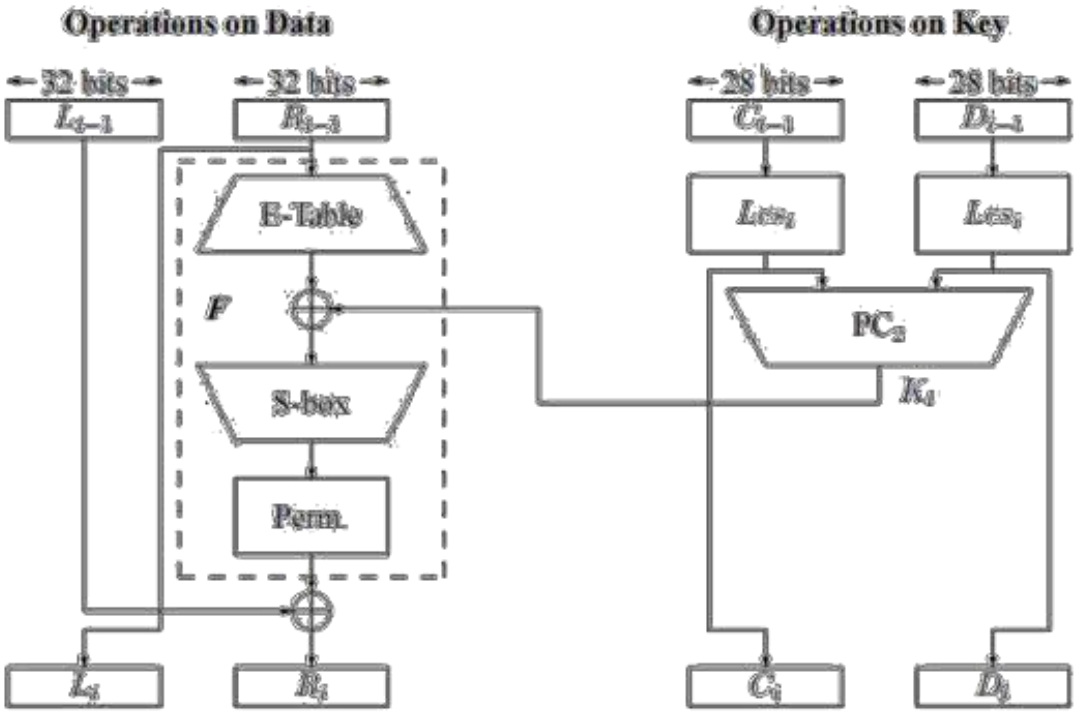


Figure : Details of a single DES round.

The main operations on the data are encompassed into what is referred to as the cipher function and is labeled F. This function accepts two different length inputs of 32 bits and 48 bits and outputs a single 32 bit number. Both the data and key are operated on in parallel, however the operations are quite different. The 56 bit key is split into two 28 bit halves Ci and Di (C and D being chosen so as not to be conf sed with L and R). The value of the key used in any round is simply a left cyclic shift and a permuted contraction of that used in the previous round. Mathematically, this can be written as

$$Ci = Lcsi(Ci-1),\ Di = Lcsi(Di-1)$$

$$Ki = P\ C2(Ci\ ,\ Di)$$

where Lcsi is the left cyclic shift for round i, Ci and Di are the outputs after the shifts, P C2(.) is a function which permutes and compresses a 56 bit number into a 48 bit number and Ki is the actual key used in round i. The number of shifts is either one or two and is determined by the round number i. For i = {1, 2, 9, 16} the number of shifts is one and for every other round it is two

OX Details

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 2.3: S-box details.

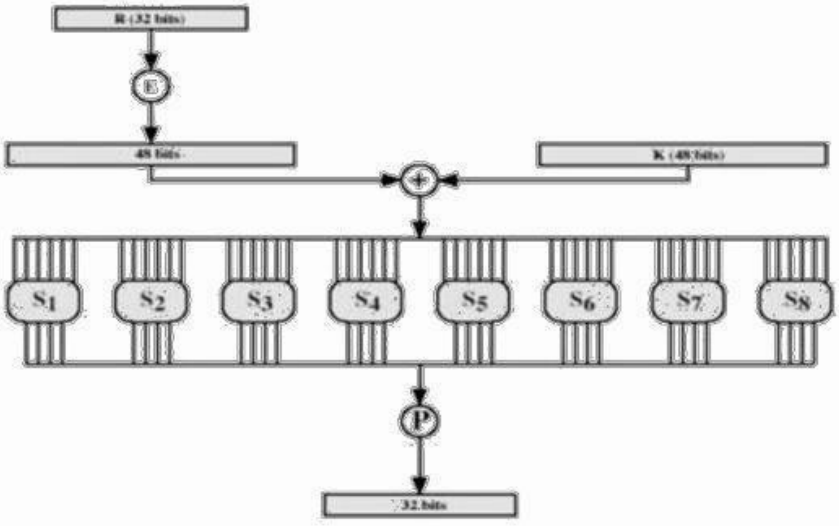


Figure 2.4: The complex F function of the DES algorithm.

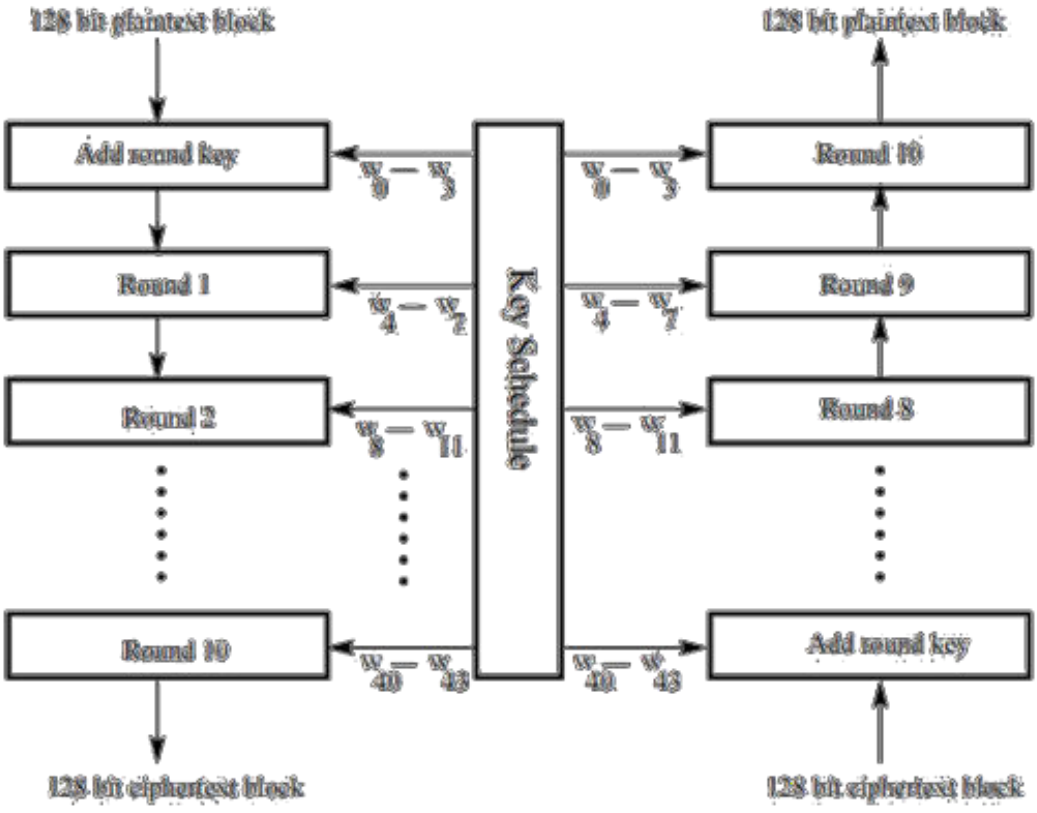
Advanced Encryption Algorithm (AES)

- AES is a block cipher with a block length of 128 bits.
- AES allows for three different key lengths: 128, 192, or 256 bits. Most of our discussion will assume that the key length is 128 bits.
- Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.
- Except for the last round in each case, all other rounds are identical.
- Each round of processing includes one single-byte based substitution step, a row-wise permutation step, a column-wise mixing step, and the addition of the round key. The order in which these four steps are executed is different for encryption and decryption.
- To appreciate the processing steps used in single round, it is best to think of a 128-bit block as consisting of a 4 × 4 matrix of bytes, rearranged as follows:

$$\begin{bmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{bmatrix}$$

Therefore, the first four bytes of a 128-bit input block occupy the first column in the 4 × 4 matrix of bytes. The next four bytes occupy the second column, and so on.

The 4×4 matrix of bytes shown above is referred to as the state array in AES.



AES Encryption AES Decryption

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages.

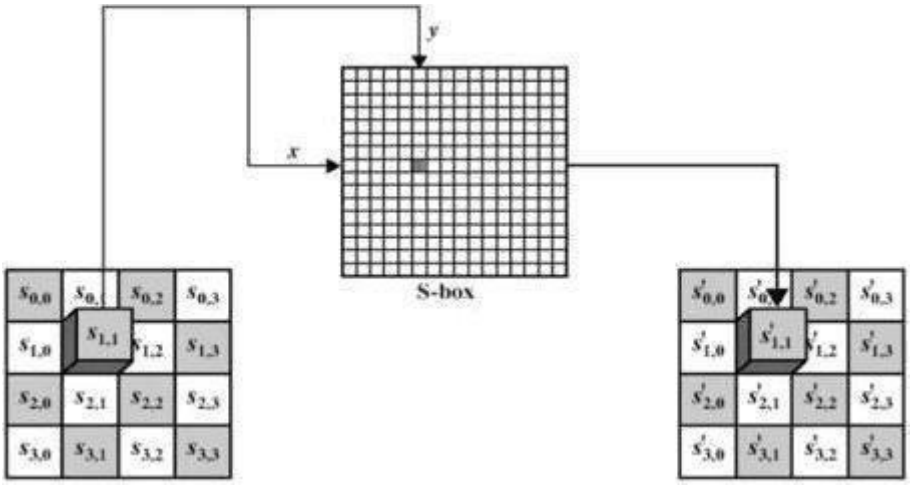
This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of its counterpart in the encryption algorithm.

The four stages are as follows: 1. Substitute bytes 2. Shift rows 3. Mix Columns 4. Add Round Key

Substitute Bytes

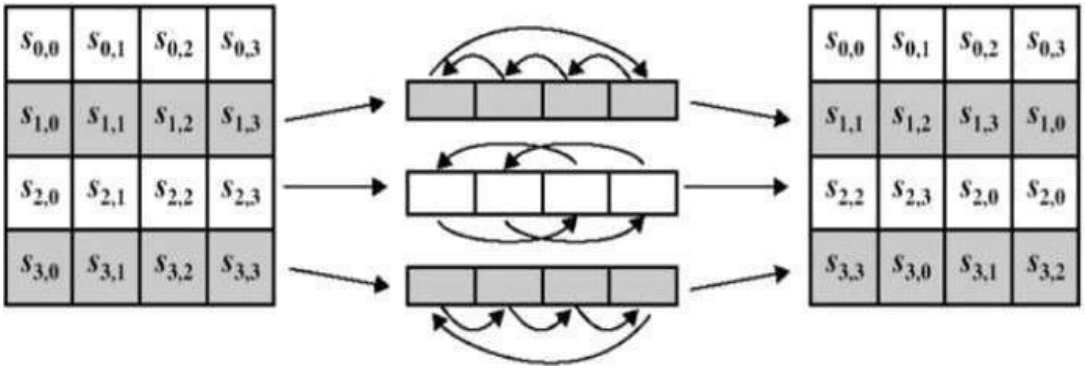
- This stage (known as SubBytes) is simply a table lookup using a 16 × 16 matrix of byte values called an s-box.
- This matrix consists of all the possible combinations of an 8 bit sequence (28 = 16 × 16 = 256).
- However, the s-box is not just a random permutation of these values and there is a well defined method for creating the s-box tables.

- The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given. Our concern will be how state is affected in each round.
- For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column.
- For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}.
- This is then used to update the state matrix.



Shift Row Transformation

- This stage (known as ShiftRows) is shown in figure below.
- Simple permutation an nothing more.
- It works as follow: – The first row of state is not altered. – The second row is shifted 1 bytes to the left in a circular manner. – The third row is shifted 2 bytes to the left in a circular manner. – The fourth row is shifted 3 bytes to the left in a circular manner.



Mix Column Transformation

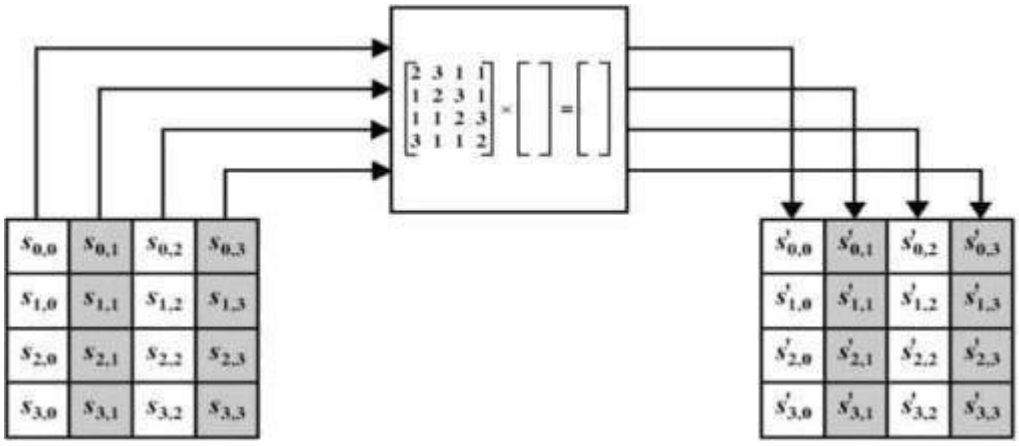
- This stage (known as MixColumn) is basically a substitution
- Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column.
- The transformation can be determined by the following matrix multiplication on state
- Each element of the product matrix is the sum of products of elements of one row and one column.
- In this case the individual additions and multiplications are performed in GF(28).
- The MixColumns transformation of a single column j ($0 \le j \le 3$) of state can be expressed as:

s ′ 0,j = (2 • s0,j) ⊕ (3 • s1,j) ⊕ s2,j ⊕ s3,j s

′ 1,j = s0,j ⊕ (2 • s1,j) ⊕ (3 • s2,j) ⊕ s3,j s ′

2,j = s0,j ⊕ s1,j ⊕ (2 • s2,j) ⊕ (3 • s3,j) s ′

3,j = (3 • s0,j) ⊕ s1,j ⊕ s2,j ⊕ (2 • s3,j)



Add Round Key Transformation

- In this stage (known as AddRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key.
- The operation is viewed as a column wise operation between the 4 bytes of a state column and one word of the round key.

- This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.
- The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure.
- Each word contains 32 bytes which means each subkey is 128 bits long. Figure 7 show pseudocode for generating the expanded key from the actual key.

Blowfish Algorithm

a symmetric block cipher designed by Bruce Schneier in 1993/94 •

characteristics:

- fast implementation on 32-bit CPUs
- compact in use of memory
- simple structure for analysis/implementation
- variable security by varying keysize

- has been implemented in various products

Blowfish Key Schedule

- uses a 32 to 448 bit key, 32-bit words store in K-array K_j ,j from 1 to 14
- used to generate
  - 18 32-bit subkeys stored in P array, P_1 ….P_{18}
  - four 8x32 S-boxes stored in $S_{i,j}$, each with 256 32-bitentries

Subkeys And S-Boxes Generation:

1. initialize P-arra and then 4 S-boxes in order using the fractional part of pi P_1 (left most 32-bit), and so on,,, $S_{4,255}$.
2. XOR P-array with key-Array (32-bit blocks) and reuse as needed: assume we have up to k_{10} then P_{10} XOR K_{10},, P_{11} XOR K_1 … P_{18} XOR K_8
3. Encrypt 64-bit block of zeros, and use the result to update P_1 and P_2.
4. Encrypting output from previous step using current P & S and replace P_3 and P_4. Then encrypting current output and use it to update successive pairs of P.
5. After updating all P's (last :P_{17} P_{18}), start updating S values using the encrypted output from previous step.
   - requires 521 encryptions, hence slow in re-keying
   - Not suitable for limited-memory applications.

Blowfish Encryption

- uses two main operations: addition modulo 2^{32} , and XOR
- data is divided into two 32-bit halves L_0 & R_0

for i = 1 to 16 do

$R_i = L_{i-1}$ XOR P_i;

$L_i = F[R_i]$ XOR R_{i-1};

$L_{17} = R_{16}$ XOR P_{18};

$R_{17} = L_{16}$ XOR P_{17};

- where

$F[a,b,c,d] = ((S_{1,a} + S_{2,b})$ XOR $S_{3,c}) + S_{4,d}$

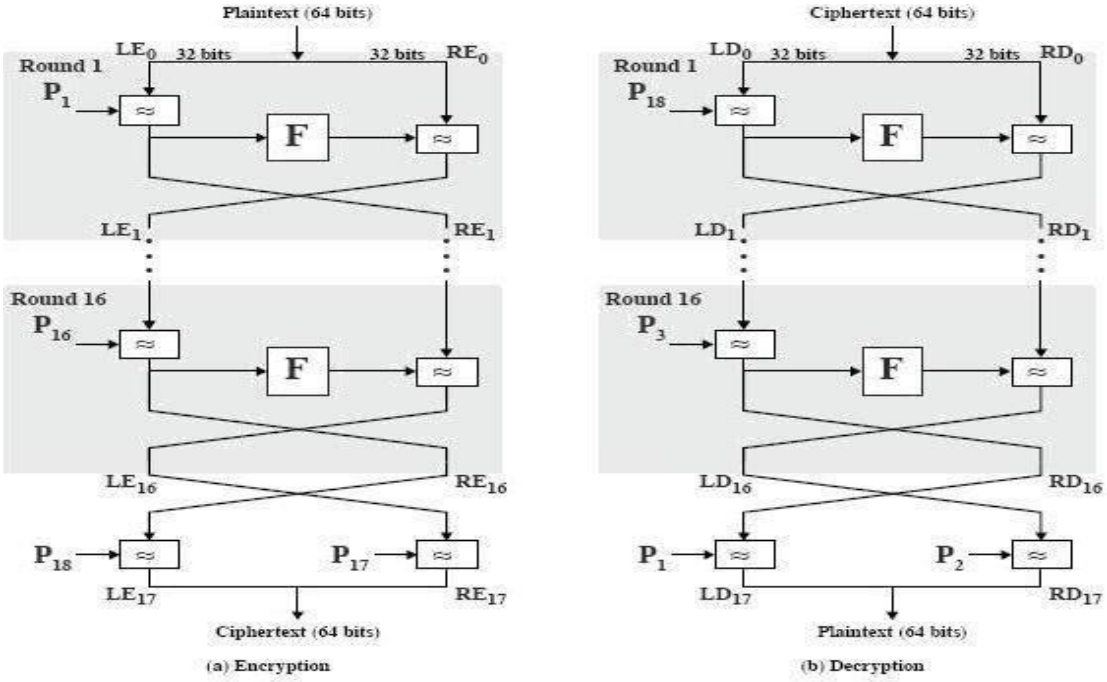


Figure 6.3 Blowfish Encryption and Decryption

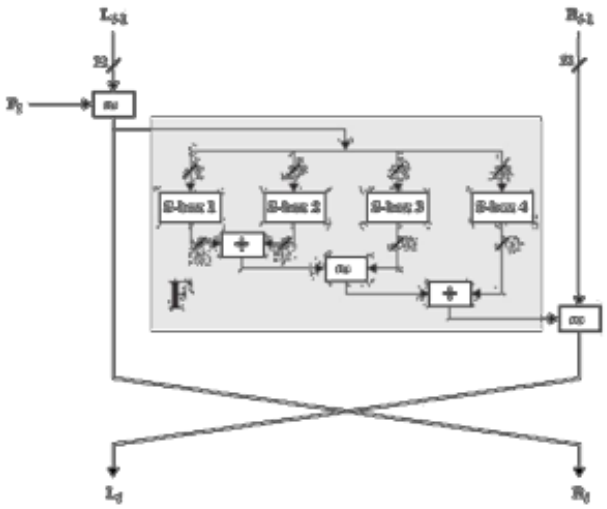


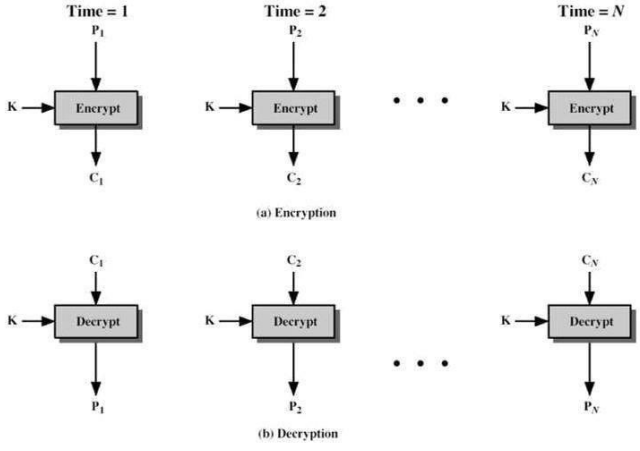
Figure 6.4 Detail of Single Blowfish Round

Block Cipher Modes Of Operations

- Direct use of a block cipher is in advisable
- Enemy can build up "code book" of plaintext/cipher text equivalents
- Beyond that, direct use only works on messages that is multiple of the cipher block size in length
- Solution: five standard Modes of Operation: Electronic Code Book (ECB), Cipher Block Chaining (CBC), CipherFeedback(CFB), Output Feedback (OFB), and Counter(CTR).

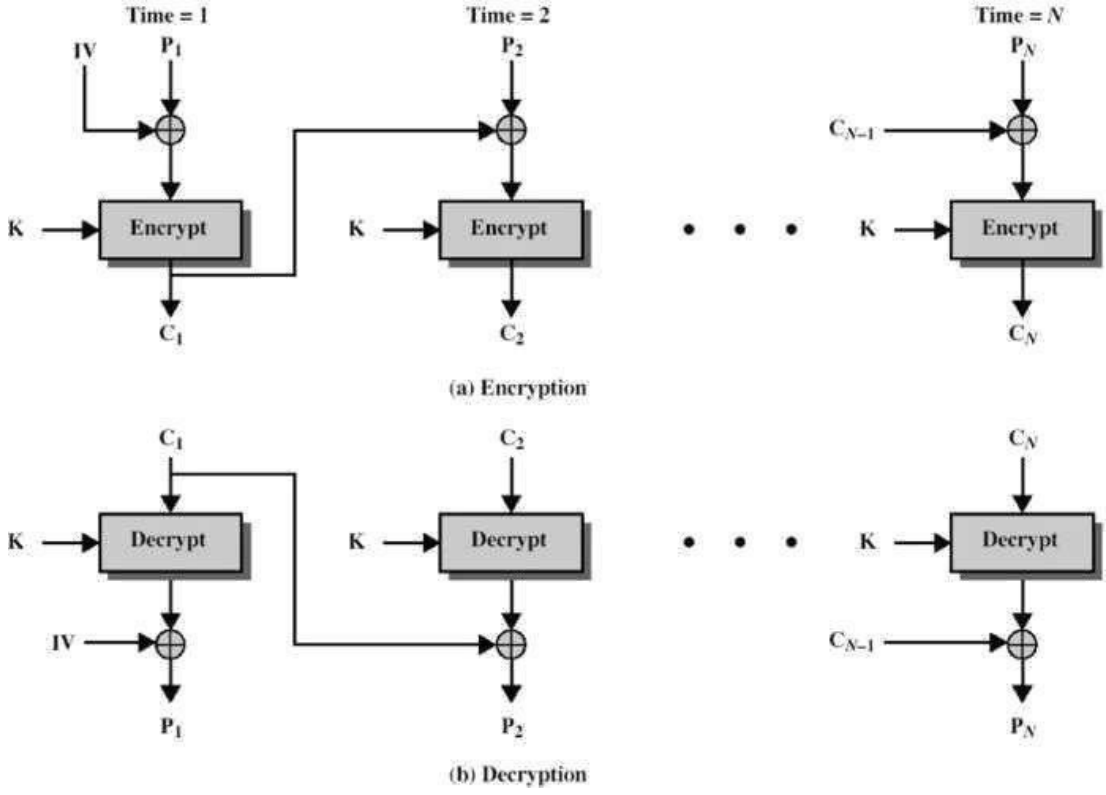
Electronic Code Book

- Direct use of the block cipher
- Used primarily to transmit encrypted keys
- Very weak if used for general-purpose encryption; never use it for a file or a message.
- • Attacker can build codebook; no semantic security
- We write {P}k → C to denote "encryption of plaintext P with key k to produce cipher text C"



Cipher Block Chaining

- We would like that same plaintext blocks produce different cipher text blocks.
- Cipher Block Chaining (see figure) allows this by XORing each plaintext with the
- Cipher text from the previous round (the first round using an Initialisation Vector (IV)).
- As before, the same key is used for each block.
- Decryption works as shown in the figure because of the properties of the XOR operation, i.e. IV ⊕ IV ⊕ P = P where IV is the Initialisation Vector and P is the plaintext.
- Obviously the IV needs to be known by both sender and receiver and it should be kept secret along with the key for maximum security.

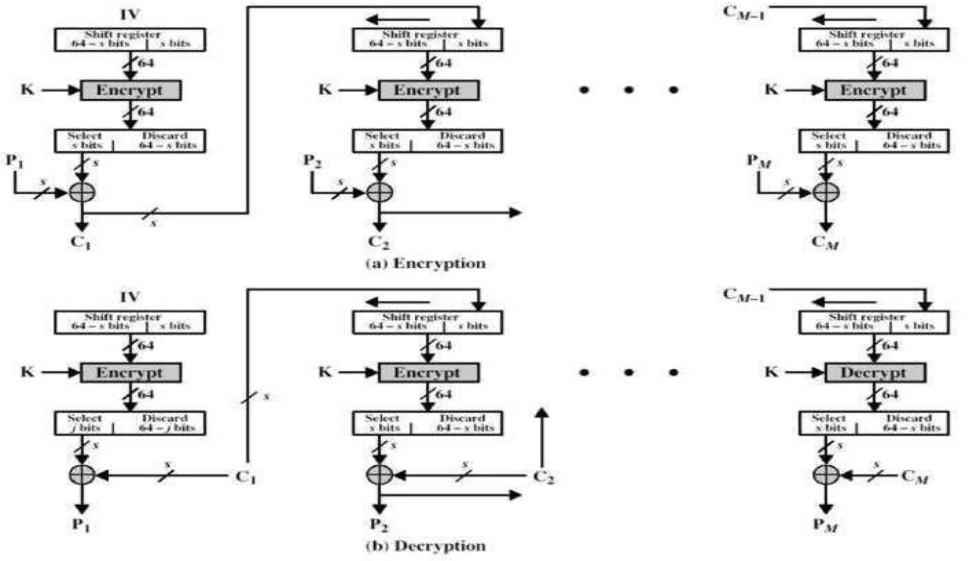


Cipher Feedback (CFB) Mode

- The Cipher Feedback and Output Feedback allows a block cipher to be converted into stream cipher.
- This eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time.
- Figure shows the CFB scheme.
- In this figure it assumed that the unit of transmission is s bits; a common value is s =8.

As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext (which is split into s bit segments).

- The input to the encryption function is a shift register equal in length to the block cipher of the algorithm (although the diagram shows 64 bits, which is block size used by DES, this can be extended to other block sizes such as the 128 bits of AES).
- This is initially set to some Initialisation Vector (IV).

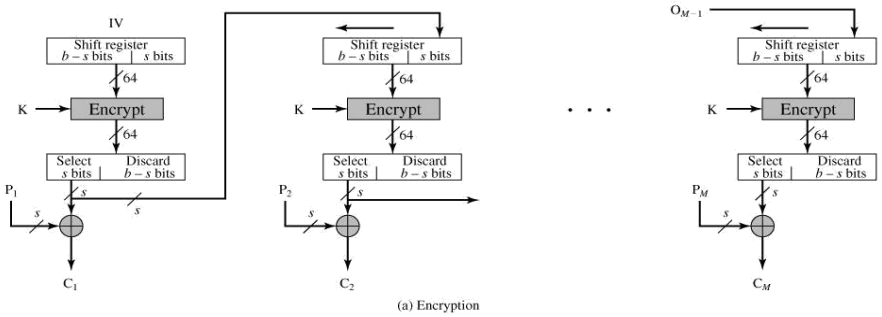


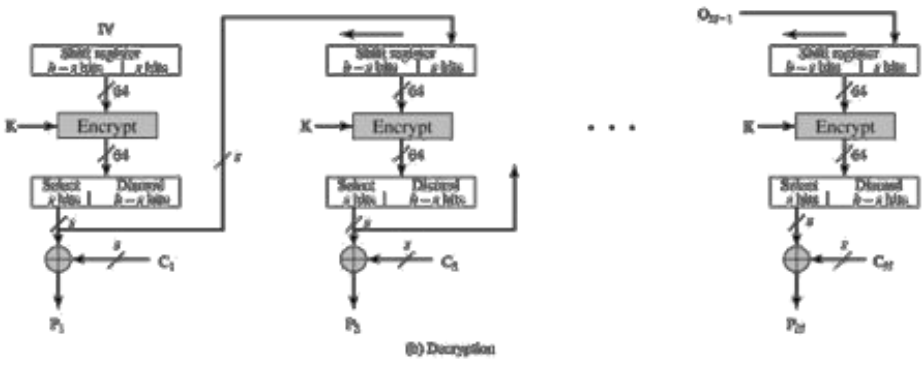
Output Feedback (Ofb) Mode

- The Output Feedback Mode is similar in structure to that of CFB, as seen in figure 13.
- As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the cipher text unit is fed back to the shift register.

- One advantage of the OFB method is that bit errors in transmission do not propagate.
- For example, if a bit error occurs in C1 only the recovered value of P1 is affected; subsequent plaintext units are not corrupted.

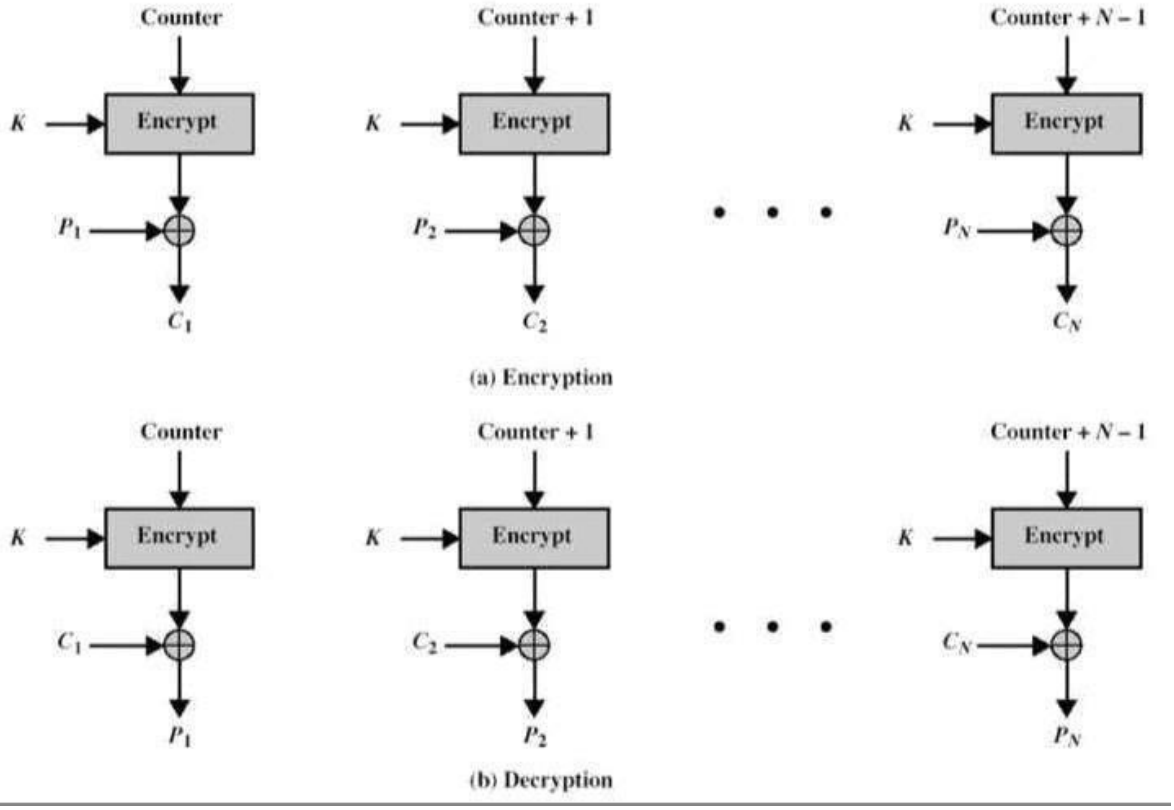
With CFB, C1 also serves as input to the shift register and therefore causes additional corruption downstream.





(b) Decryption

Counter Mode



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

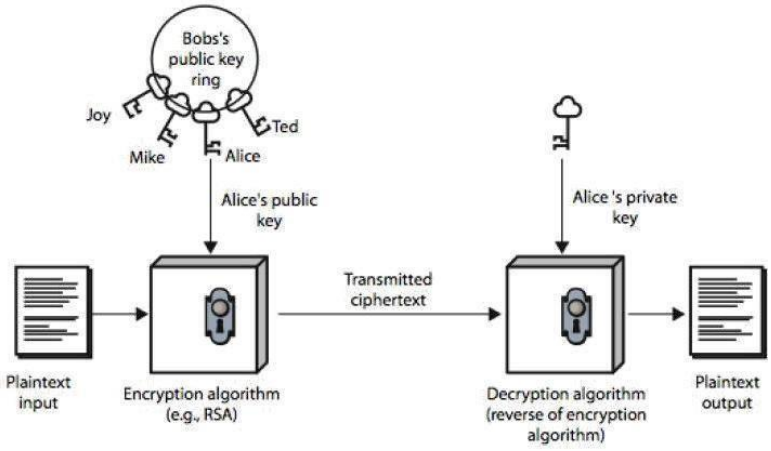
Public-key/two-key/asymmetric cryptography involves the use of **two** keys:

- a *public-key*, which may be known by anybody, and can be used to **encrypt messages**, 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 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
- either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms like RSA)

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, cipher text & decryption algorithm.



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.

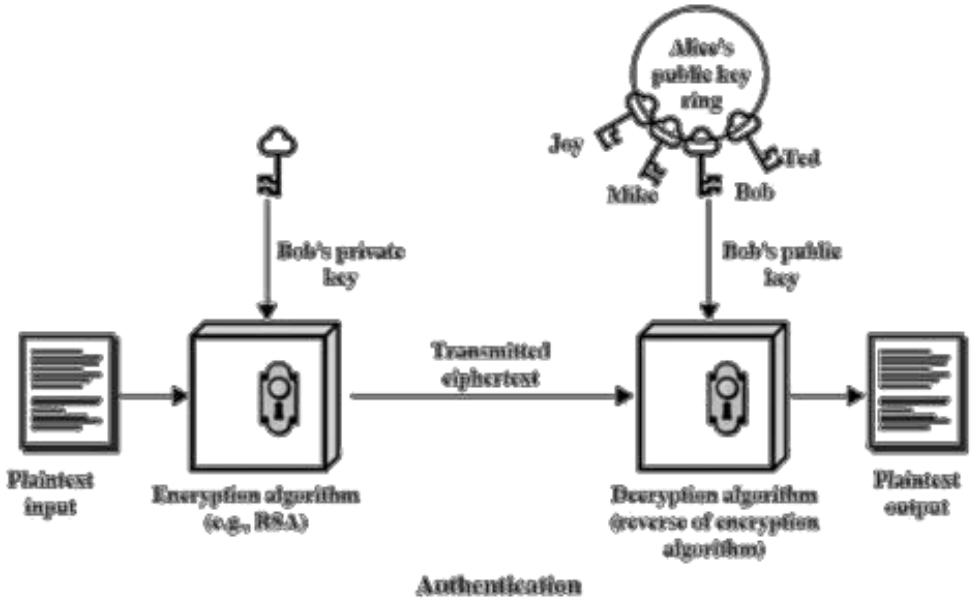
3.) If B wants to send a confidential message to A, B encrypts the message using A's public key.

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).

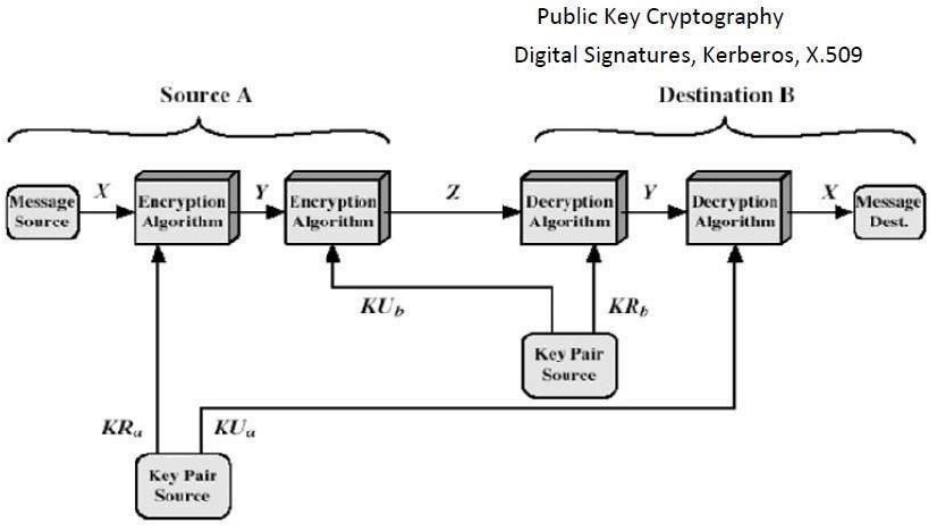
5.) If a user wishes to change his keys –generate another pair of keys and publish the public one: no interaction with other users is needed. Notations used in Public-key cryptography:

- The public key of user A will be denoted **KUA**.
- The private key of user A will be denoted **KRA**.
- 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 attacker may impersonate user B**: he sends a message $E(KU_A,X)$ and claims in the message to be B –A has 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:



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(KRB,X)

B encrypts Y with A's public key: Z=E(KUA,Y)

A will decrypt Z (and she is the only one capable of doing it): Y=D(KRA,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(KUB,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:

1. Computationally easy for a party B to generate a pair (public key KUb, privatekey KRb).
2. Easy for sender A to generate cipher text:
3. Easy for the receiver B to decrypt cipher text using private key:
4. Computationally infeasible to determine private key (KRb) knowing public key (KUb)
5. Computationally infeasible to recover message M, knowing KUb and cipher text C
6. either of the two keys can be used for encryption, with the other used for decryption:

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

Easy is defined to mean a problem that can be solv ing 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 -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** (cryptanalysis) problems. More generally the **hard** problem is known, it's just made too hard to do in practice.

- 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 ll 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-k y. 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 ø(n), where* ***ø(n)=(p-1)(q- 1)***
- *selecting at random the encry tion key e, where 1<e<ø(n),and* ***gcd(e,ø(n))=1***
- *solve following equation to find decryption key d:* ***e.d=1 mod ø(n)*** *and 0≤d≤n*
- *publish their public encr ption 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=Me mod n**, where 0≤M<n

To decrypt the ciphertext C the owner:

– uses their private key *KR={d,n}*

– computes: **M=Cd mod n = (Me) d mod n = Med 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 **Med = M mod n** for all M<n

b) It is relatively easy to calculate Me 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 **ap-1 ≡ 1 mod p. Corollary**: For any positive integer **a** and prime **p**, **ap ≡ 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 **aφ(n)≡1 mod n. Corollary: F**or ny integers a,n we have **aφ(n)+1≡a mod n Corollary:** Let p,q be two odd primes and n=pq. Then: φ(n)=(p-1)(q-

1) For any integer m with 0<m<n, m(p-1)(q-1)+1 ≡ m mod n For any integers k,m with 0<m<n, mk(p-1)(q-1)+1 ≡ m mod n Euler's theorem provides us the numbers d, e such that Med=M mod n. We have to choose d,e such that ed=kφ(n)+1, or equivalently, d≡e-1mod φ(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 = 887 mod 187 = 11

decryption: M = 1123 mod 187 = 88

Another example of RSA is given as,

Let **p = 11, q = 13, e = 11, m = 7**

n = pq i.e. n= 11*13 = 143

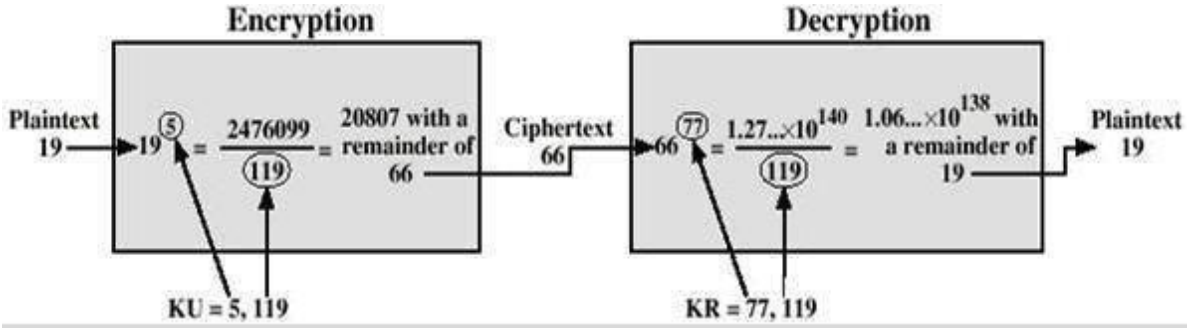
ø(n)= (p-1)(q-1) i.e. (11-1)(13-1) = 120

e.d=1 mod ø(n) i.e. 11d mod 120 = 1 i.e. (11*11) mod 120=1; so d

= 11 public key :{11,143} and private key: {11,143}

C=Me mod n, so ciphertext = 711mod143 = 727833 mod 143; i.e. C = 106

M=Cd mod n, plaintext = 10611 mod 143 = 1008 mod 143; i.e. M = 7



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 defense 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.

Diffie-Hellman Key Exchange

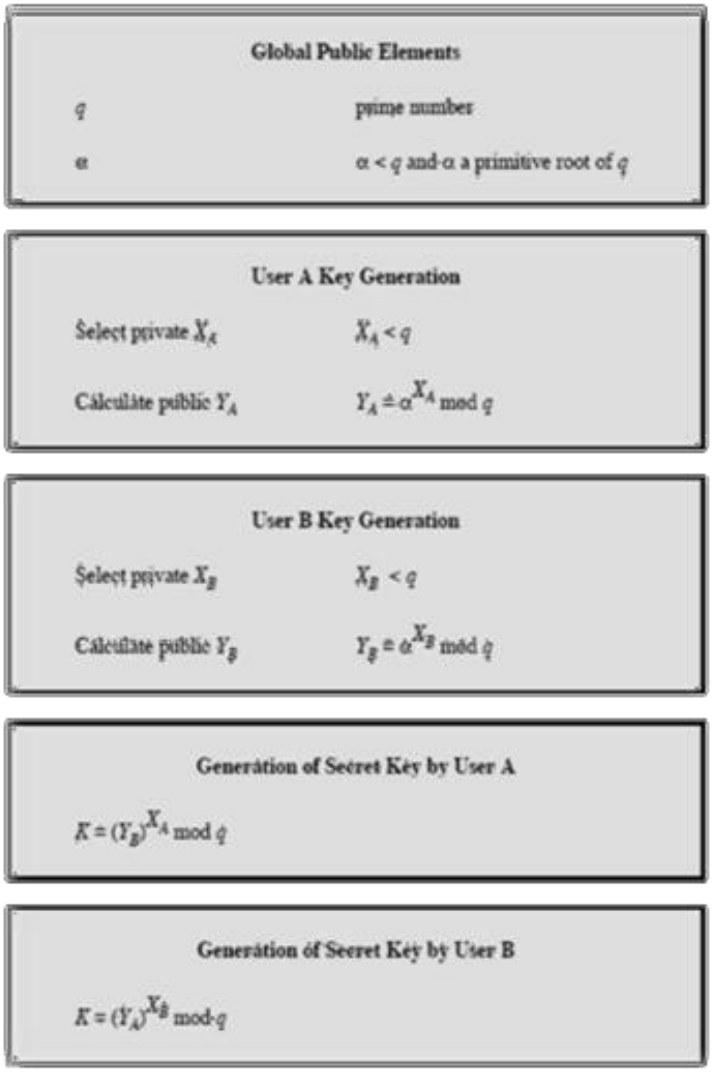
Diffie-Hellman key exchange (**D-H**) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications

using a symmetric key cipherMedia.TheD-Hlgorithm depends for its effectiveness on the difficulty of computing discrete logar thms.

First, a primitive root of a prime number p, can be fined 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, a^2 mod p,..., a^{p-1} mod 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 $b \equiv a^i \pmod{p}$ where $0 \leq i \leq (p\ 1)$.The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as $\boldsymbol{dlog_{a,p}(b)}$**.** The algorithm is summarized below:



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 XA < q and computes YA = αXA mod q. Similarly, user B independently selects a random integer XA < q and computes YB = αXB mod 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 = (YB)XA mod q and user B computes the key as K = (YA)XB mod 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 $ab \equiv c \pmod{p}$ 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 $13^{10} \pmod{37}$ which is 4 and sends that to Bob. Let Bob pick b = 7. Bob calculates $13^7 \pmod{37}$ 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 $32^{10} \pmod{37}$ which is 30, the secret key.
- Bob receives 4 and calculates $4^7 \pmod{37}$ which is 30, the same secret key.

2) Let p = 47 and g = 5. Let Alice pick a = 18. Alice calculates $5^{18} \pmod{47}$ which is 2 and sends that to Bob. Let Bob pick b = 22. Bob calculates $5^{22} \pmod{47}$ which is 28 and sends that to Alice.

- Alice receives 28 and calculates $28^{18} \pmod{47}$ which is 24, the secret key.
- Bob receives 2 and calculates $2^{22} \pmod{47}$ 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:

1. 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}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2 = $(Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates K1 = $(Y_{D1})^{X_E} \bmod q$.
5. Bob transmits X_A to Alice.
6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1 = $(Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates K2 = $(Y_{D2})^{X_A} \bmod q$.

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:

1. Alice sends an encrypted message M: E(K2, M).

2. Darth intercepts the encrypted message and decrypts it, to recover M.

3. 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.

Elliptic Curve Cryptography (Ecc)

Elliptic curve cryptography (ECC) is an approach to publ c-key cryptography based on the algebraic structure of elliptic curves over finite fiel s. The use of elliptic curves in cryptography was suggested independently by N al Koblitz and Victor S. Miller in 1985. The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bitSkyupssize,therebyreducing the processing overhead.

Elliptic Curve over GF(p)

Let GF(p) be a finite field, > 3, and let a, b

 GF(p) are constant such that ***4a3 + 27b2 ≡ 0 (mod p).*** An elliptic curve, E(a,b)(GF(p)),

is defined as the set of points (x,y) **C** GF(p) * GF(p) which satisfy the equation

y2 ≡ x3 + ax + b (mod p), together with a special point, O, *called the point at infinity*. Let P and Q be two points on E(a,b)(GF(p)) and O is the point at infinity.

• P+O = O+P = P

• If P = (x1,y1) then -P = (x1 ,-y1) and P + (-P) = O.

• If P = (x1,y1) and Q = (x2,y2), and P and Q are not O.

then P +Q = (x3 ,y3) where

x3 = ƛ 2 - x1 - x2

86

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \; for _ x_1 \neq x_2 \\ \dfrac{3x_1^{\,2} + a}{2y_1} \; for _ x_1 = x_2 \end{cases}$$

y3 = ƛ (x1 - x3) - y1 and

ƛ = (y2-y1)/(x2-x1) if P ≠ Q

ƛ = (3x12+a)/ 2y1 if P = Q

An elliptic curve may be defined over any finite field GF(q). For GF(2_m), the curve has a different form:- $y^2 + xy = x^3 + ax^2 + b$, *where b !=0.*

Cryptography with Elliptic Curves

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple additions are the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, some kind of hard problem such as discrete logarithm or factorization of prime numbers is needed. Considering the equation, Q=kP, where Q,P are points in an elliptic curve, it is "easy" to compute Q given k,P , but "hard" to find k given Q,P. This is known as the elliptic curve logarithm problem. K could be so large as to make brute-force fail.

Pick a prime number p= 2180 and elliptic curve parameters and b for the equation

y2 ≡ x3 + ax + b (mod p) which defines the elliptic group of points Ep(a,b). Select generator point G=(x1,y1) in Ep(a,b) uch that the smallest value for which nG=O be a very large prime number. Ep(a,b) and G are parameters of the cryptosystem known to all participants. The following steps take place:

ECC Key Exchange

- A & B select private keys nA<n, nB<n
- compute public keys: PA=nA×G, PB=nB×G
- Compute shared key: K=nA×PB, K=nB×PA {same since K=nA×nB×G }

ECC Encryption/Decryption As with key exchange system, an encryption/decryption system requires a point G and and elliptic group Ep(a,b) as parameters. First thing to be done is to encode the plaintext message m to be sent as an x-y point **Pm**. Each user chooses private key nA<n and computes public key PA=nA×G. To encrypt and send a message to Pm to B, A chooses a random positive integer k and produces the ciphertext **Cm** consisting of the pair of points **Cm={kG, Pm+kPb}.** here, A uses B's public key. To

decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point **Pm+*k*Pb – nB(*kG*) = Pm+*k*(nB*G*) – nB(*kG*) = Pm** A has masked the message **Pm** by adding ***k*Pb** to it. Nobody but A knows the value of k, so even though **Pb** is a public key, nobody can remove the mask k**Pb.** For an attacker to recover the message, he has to compute k given G and kG, which is assumed hard.

Security of ECC To **protect** a 128 bit AES key it would take a RSA Key Size of 3072 bits whereas an ECC Key Size of 256 bits.

Computational Effort for Cryptanalysis of Elliptic Curve Cryptography Compared to RSA

Key Size	MIPS-Years
150	3.8×10^{10}
205	7.1×10^{18}
234	1.6×10^{28}

(a) Elliptic Curve Logarithms using the Pollard rho Method

Key Size	MIPS-Years
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

(b) Integer Factorization using the General Number Field Sieve

Hence for similar security ECC offers significant computational advantages.

Applications of ECC:

- Wireless communication devices
- Smart cards
- Web servers that need to handle many encryption sessions
- Any application where security is needed but lacks the power, storage and computational power that is necessary for our current cryptosystems

Key Management

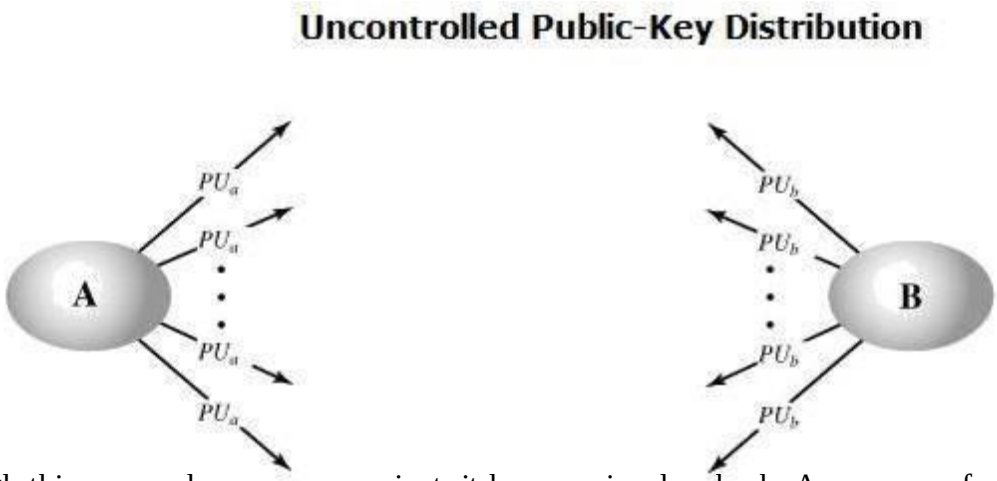
One of the major roles of public-key encryption has been to address the problem of key distribution. Two distinct aspects to use of public key encryption are present.

- 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.

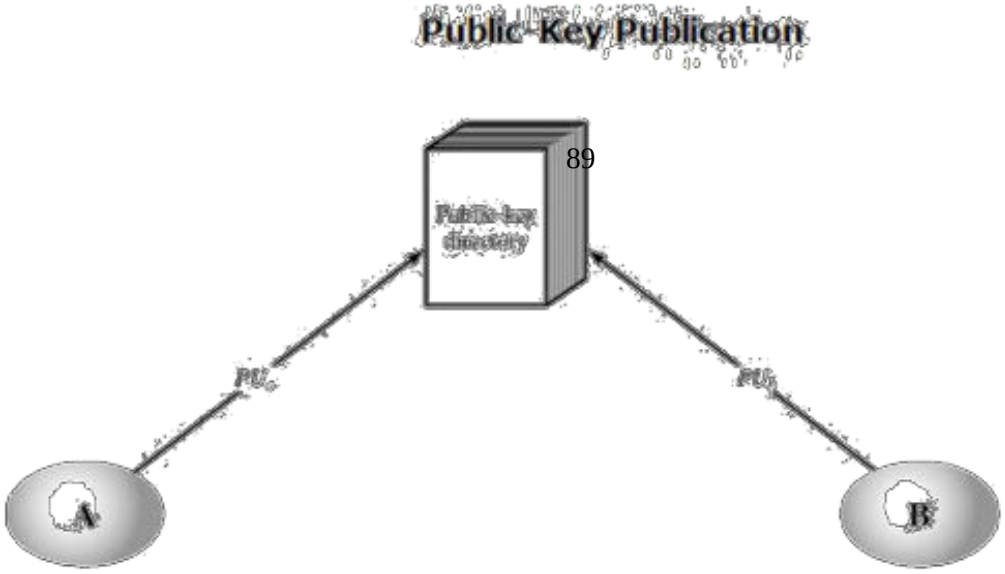


Though this approach seems convenient, it has a major drawback. Anyone can forge such a public announcement. Some user could pretend to be user and send a public key to another participant or broadcast such a public key. Until the time when A discovers about the forgery and alerts other participants, 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.

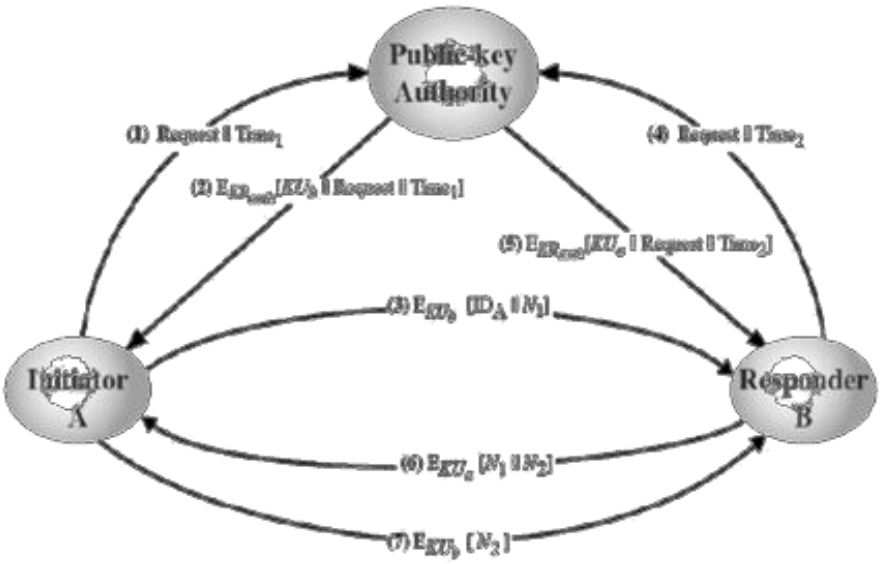


3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is

   Mandatory. This scheme has still got some vulnerability. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively a out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Or else, the adversary may tam er 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 w th its private key (for authentication) – message contains Bob's public k y 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 IDA 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 N1 plus a new generated nonce N2 (to identify uniquely the transaction)

7.) Alice replies once more encrypting Bob's nonce N2 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

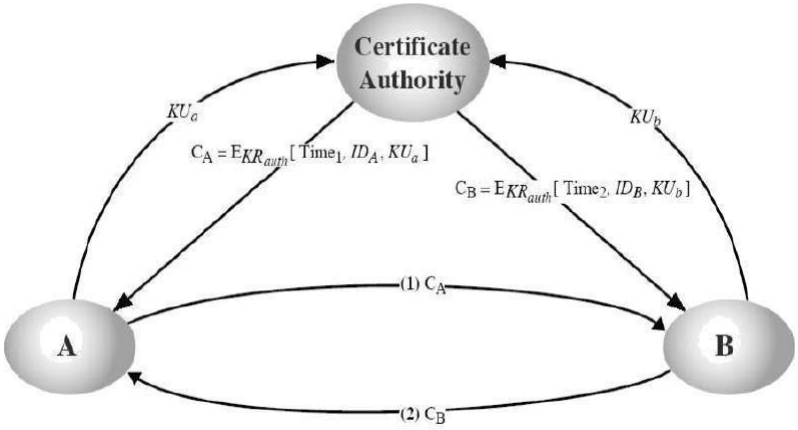
The above technique looks attractive, but still has some drawbacks. For any communication between any two users, the central authority must be consulted by both users to get the newest public keys i.e. the central authority must be online 24 hours/day. If the central authority goes offline, all secure 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 n me and public key of the certificate's owner.

2. Any participant can verify that the certificate originate 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.



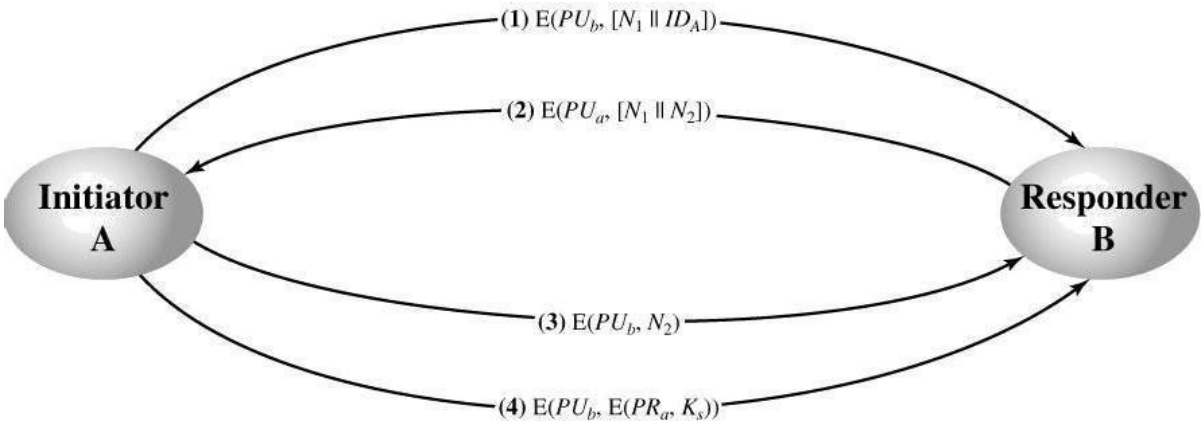
Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

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

One scheme has becomeSkyupsuniversallyaccepted for formatting public-key certificates: the X.509 standard. X.509 certificates are ed in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

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 (IDA) and a nonce (N1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PUa and containing A's nonce (N1) as well as a new nonce generated by B (N2) Because only B could have decrypted message (1), the presence of N1 in message (2) assures A that the correspondent is B.
3. A returns N2 encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key Ks and sends M = E(PUb, E(PRa, Ks)) 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(PUa, D(PRb, M)) to recover the secret key.

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

UNIT-3

Message Authentication Algorithms and Hash Function: Authentication Requirements, Functions, Message Authentication Codes, Hash Functions, Secure Hash Algorithms, Whirlpool, HMAC, CMAC, Digital Signatures, Knapsack Algorithm, ***Authentication Applications:*** *Kerberos, X.509 Authentication Services, Public-Key Infrastructure, Biometric Authentication*.

Message Authentication

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 re:

- ***Message encryption***–the ciphertext serves as auth nticator
- ***Message authentication code (MAC)***–a public function of the message and a secret key producing a fixed-length value to erve 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.

The different ways in which message encryption can provide authentication, confidentiality in both symmetric and asymmetric encryption techniques is explained with the table below:

Confidentiality and Authentication Implications of Message Encryption

A → B: $E_K[M]$ •Provides confidentiality —Only A and B share K •Provides a degree of authentication —Could come only from A —Has not been altered in transit —Requires some formatting/redundancy •Does not provide signature —Receiver could forge message —Sender could deny message (a) Symmetric encryption
A → B: $E_{KU_b}[M]$ •Provides confidentiality —Only B has KR_b to decrypt •Provides no authentication —Any party could use KU_b to encrypt message and claim to be A (b) Public-key encryption: confidentiality
A → B: $E_{KR_a}[M]$ •Provides authentication and signature —Only A has KR_a to encrypt —Has not been altered in transit —Requires some formatting/redundancy —Any party can use KU_a to verify signature (c) Public-key encryption: authentication and signature
A → B: $E_{KU_b}[E_{KR_a}(M)]$ •Provides confidentiality because of KU_b •Provides authentication and signature because of Kr_a (d) Public-key encryption: confidentiality, authentication, and signature

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: **MAC=Ck(M)** The message

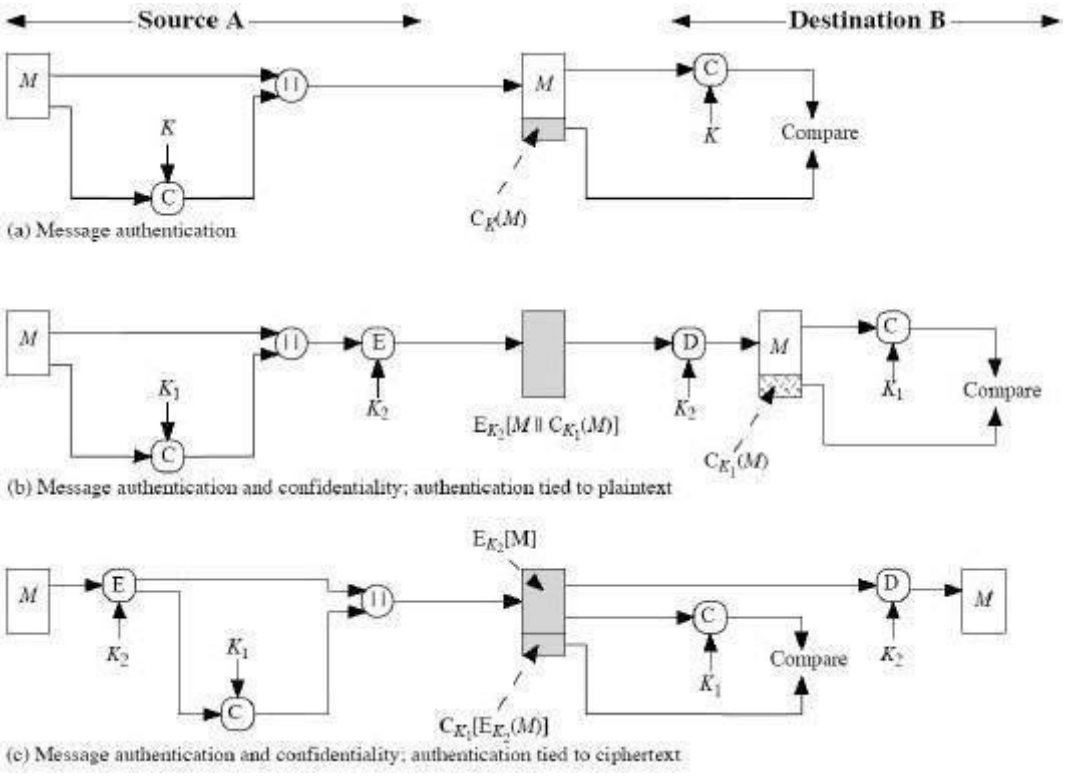
and the MAC are transmitted to the intended recipient, who upon receiving performs the same calculation on the received message, using the same secret key to generate a new MAC. The received MAC is compared to the calculated MAC and only if they match, then:

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.

Basic uses of Message Authentication Code (MAC) are shown in the figure:



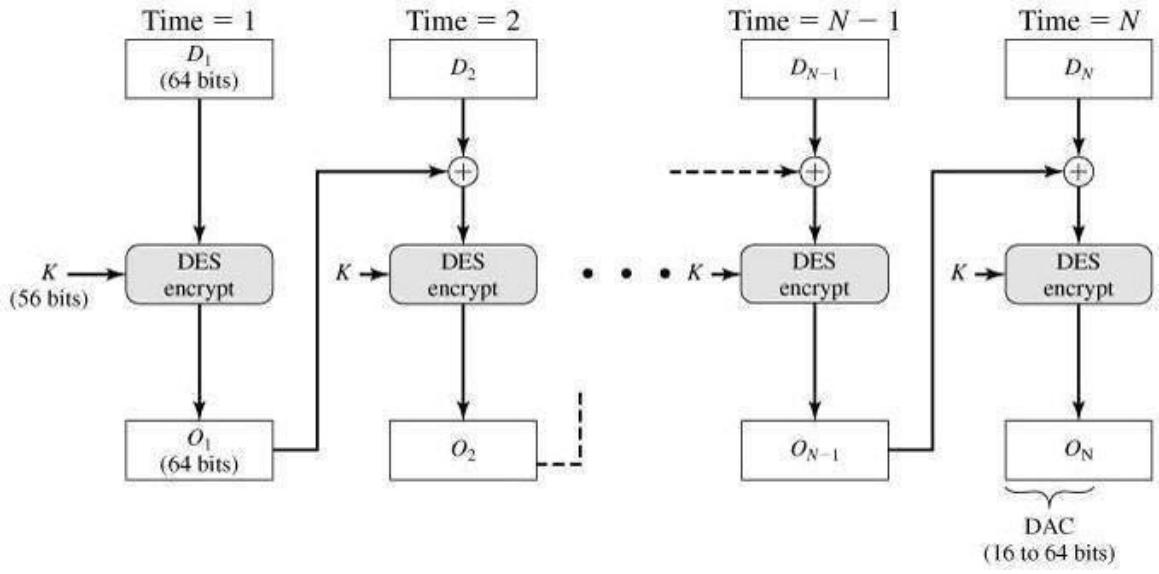
There are three different situations where use of a MAC is desirable:

- 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.

MESSAGE AUTHENTICATION CODE BASED ON DES

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). But, security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms. The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES shown below with an initialization vector of zero.

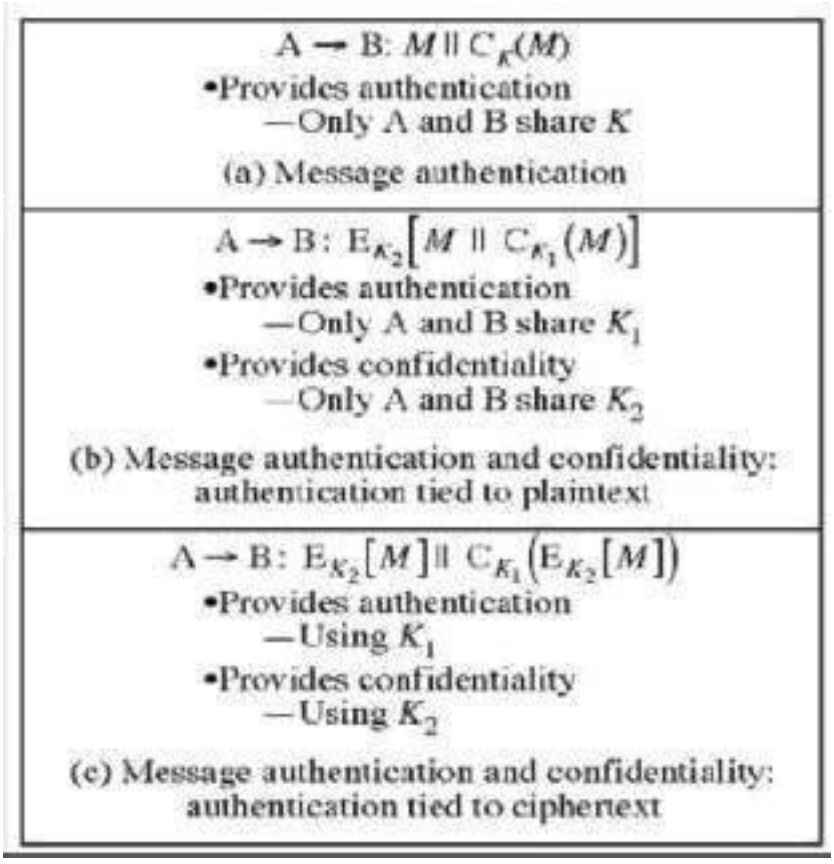


The data (e.g., message, record, file, or program) to be authenticated are grouped into contiguous 64-bit blocks: D1, D2,..., DN. If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm, E, and a secret key, K, a data authentication code (DAC) is calculated as follows:

$$O_1 = E(K, D_1)$$
$$O_2 = E(K, [D_2 \oplus O_1])$$
$$O_3 = (K, [D_3 \oplus O_2])$$
$$\bullet$$
$$\bullet$$
$$\bullet$$
$$O_N = E(K, [D_N \oplus O_{N1}])$$

The DAC consists of either the entire block ON or the leftmost M bits of the block, with $16 \leq M \leq 64$

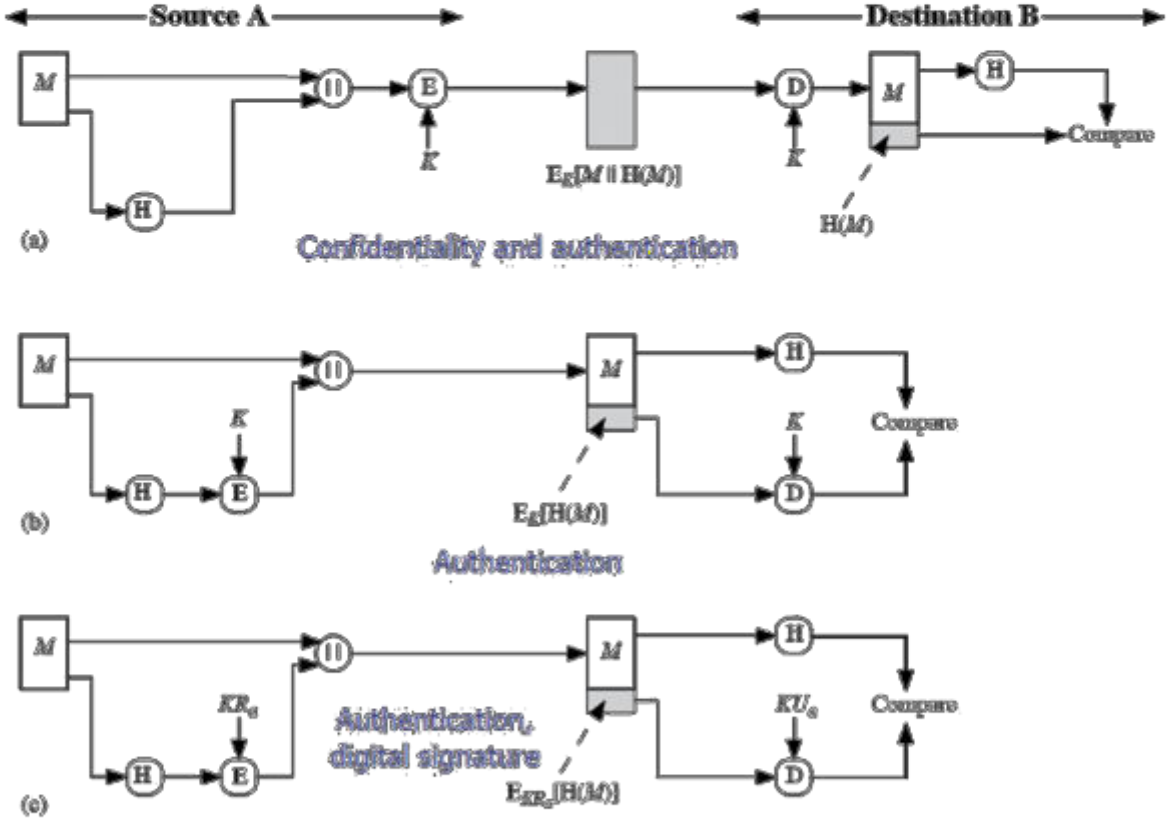
Use of MAC needs a shared secret key between the communicating parties and also MAC does not provide digital signature. The following table summarizes the confidentiality and authentication implications of the approaches shown above.

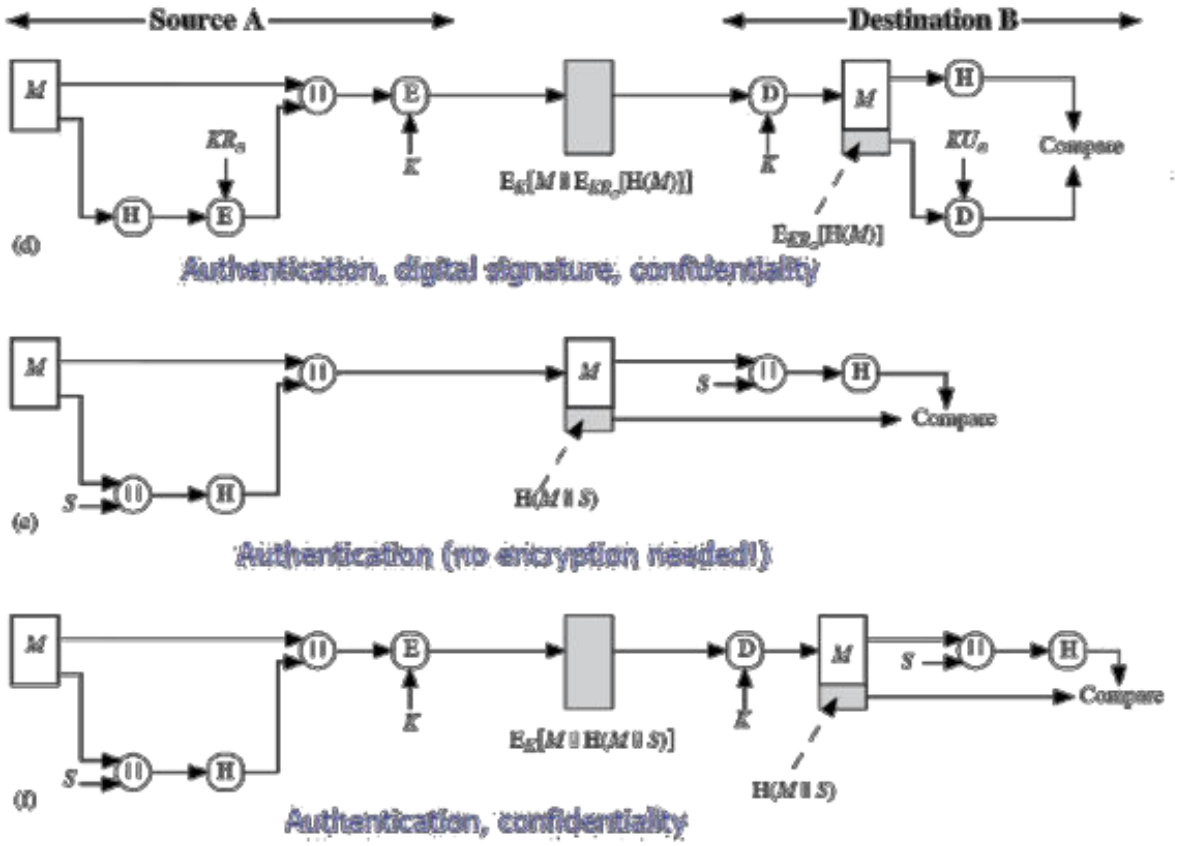


HASH FUNCTION

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 variety of ways in which a hash code can be used to provide message authentication is shown below and explained stepwise in the table.

$A \rightarrow B$: $E_K[M \parallel H(M)]$ •Provides confidentiality —Only A and B share K •Provides authentication —H(M) is cryptographically protected (a) Encrypt message plus hash code	$A \rightarrow B$: $E_K[M \parallel E_{KR_a}[H(M)]]$ •Provides authentication and digital signature •Provides confidentiality —Only A and B share K (d) Encrypt result of (c) - shared secret key
$A \rightarrow B$: $M \parallel E_K[H(M)]$ •Provides authentication —H(M) is cryptographically protected (b) Encrypt hash code - shared secret key	$A \rightarrow B$: $M \parallel H(M \parallel S)$ •Provides authentication —Only A and B share S (e) Compute hash code of message plus secret value
$A \rightarrow B$: $M \parallel E_{KR_a}[H(M)]$ •Provides authentication and digital signature —H(M) is cryptographically protected —Only A could create $E_{KR_a}[H(M)]$ (c) Encrypt hash code - sender's private key	$A \rightarrow B$: $E_K[M \parallel H(M) \parallel S]$ •Provides authentication —Only A and B share S •Provides confidentiality —Only A and B share K (f) Encrypt result of (e)





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.

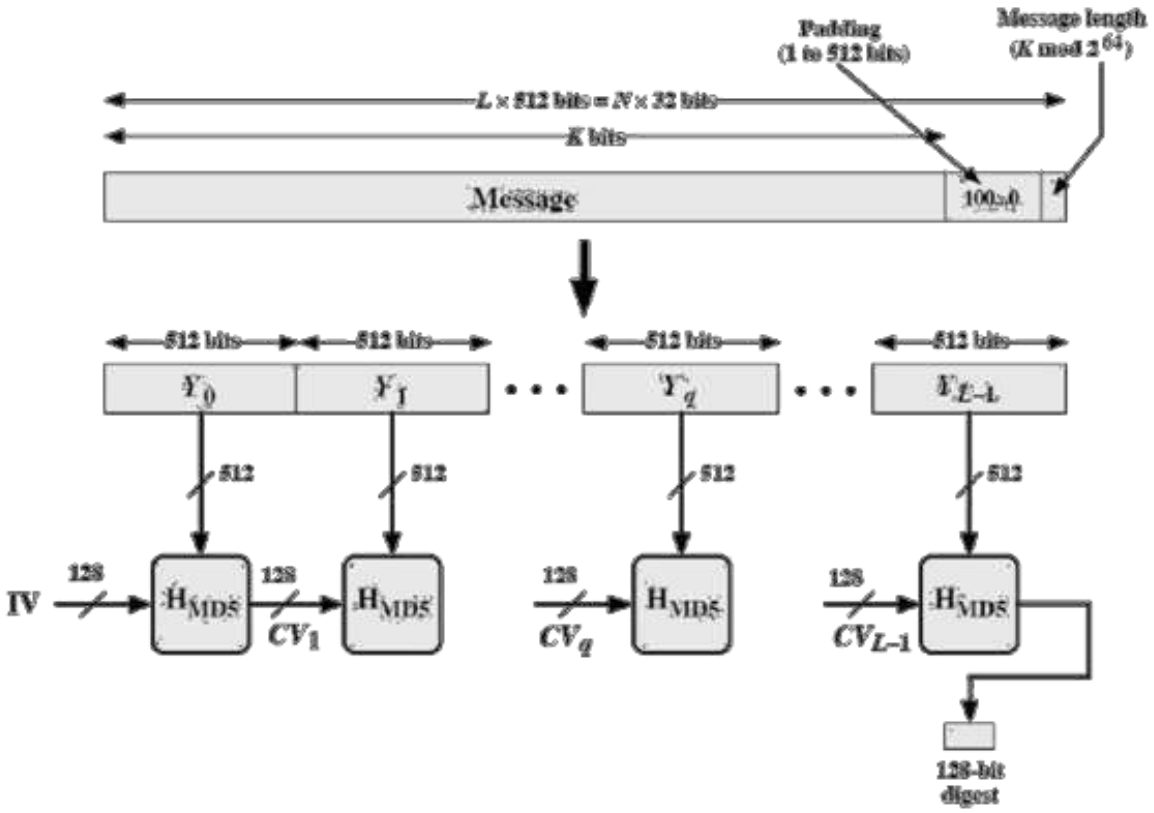
MD5 MESSAGE DIGEST ALGORITHM

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, 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.

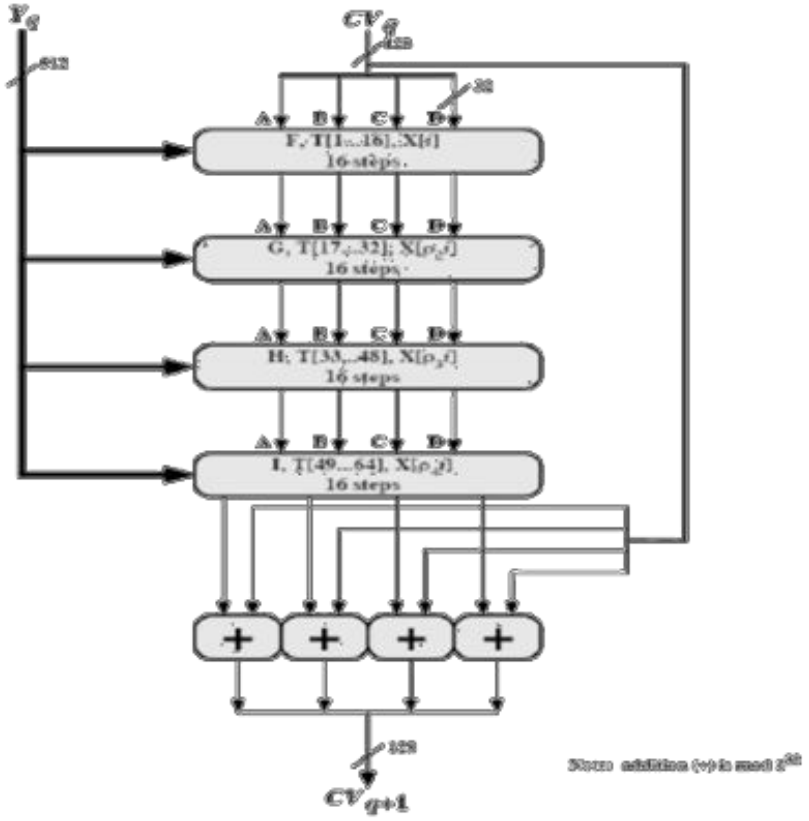
3.) *Initialize MD buffer*: A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D) and are initialized with A=0x01234567, B=0x89ABCDEF, C=0xFEDCBA98, D=0x76543210 i.e. 32-bit integers (hexadecimal values).



Message Digest Generation Using MD5

4.) *Process Message in 512-bit (16-word) blocks* : The h art 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 232 * 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 (CVq) to produce CVq+1. The addition is done independently for each of the four words in the buffer with each of the corresponding words in CVq, using addition modulo 232. This operation is shown in the figure below:



5.) *Output*: After all L 512-bit blocks have been proc ssed, the output from the Lth stage is the 128- bit message digest. MD5 can be summarized as follows:

$CV_0 = IV$ $CV_{q+1} = SUM_{32}(CV_q, RF_I Y_q RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$ $MD = CV_L$ Where,

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

Y_q = the q^{th} 512-bit block of the message

L = the number of blocks in the message

CV_q = chaining variable processed with the q^{th} block of the message.

RFx = round function using primitive logical function x.

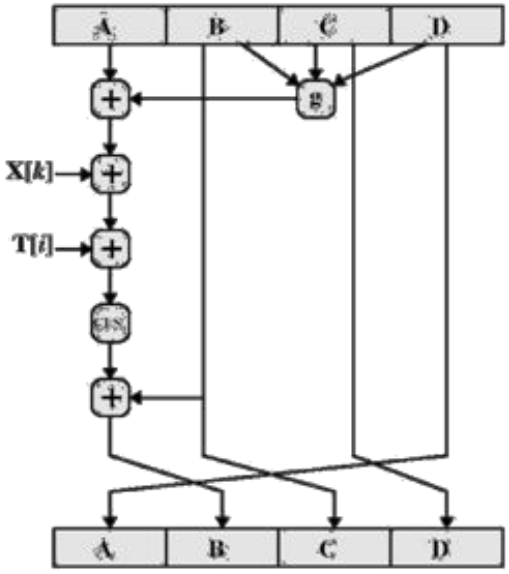
MD = final message digest value

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

MD5 Compression Function:

Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form, **a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)**

where a, b, c, d refer to the four words of the buffer but used in varying permutations. After 16 steps, each word is updated 4 times. g(b,c,d) is a different nonlinear function in each round (F,G,H,I). Elementary MD5 operation of a single step is shown below.



The primitive function g of the F,G,H,I is given as:

Round	Primitive function g	g(b, c, d)
1	F(b, c, d)	(b ∧ c) ∨ (b' ∧ d)
2	G(b, c, d)	(b ∧ d) ∨ (c ∧ d')
3	H(b, c, d)	b ⊕ c ⊕ d
4	I(b c, d)	c ⊕ (b ∨ d')

Truth table

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Where the logical operators (AND, OR, NOT, XOR) are represented by the symbols (∧,∨**, ~,(+)).**

Each round mixes the buffer input with the next "word" of the message in a complex, non-linear manner. A different non-linear function is used in each of the 4 rounds (but the same function for all 16 steps in a round). The 4 buffer words (a,b,c,d) are rotated from step to step so all are used and updated. g is one of the primitive functions F,G,H,I for the 4 rounds respectively. X[k] is the kth 32-bit word in the current message block. T[i] is the ith entry in the matrix of constants T. The addition of varying constants T and the use of different shifts helps ensure it is extremely difficult to compute collisions. The array of 32-bit words X[0..15] holds the value of current 512-bit input block being processed. Within a round, each of the 16 words of X[i] is used exactly once, during one step. The order in which these words is used varies from round to round. In the first round, the

words are used in their original order. For rounds 2 through 4, the following permutations are used

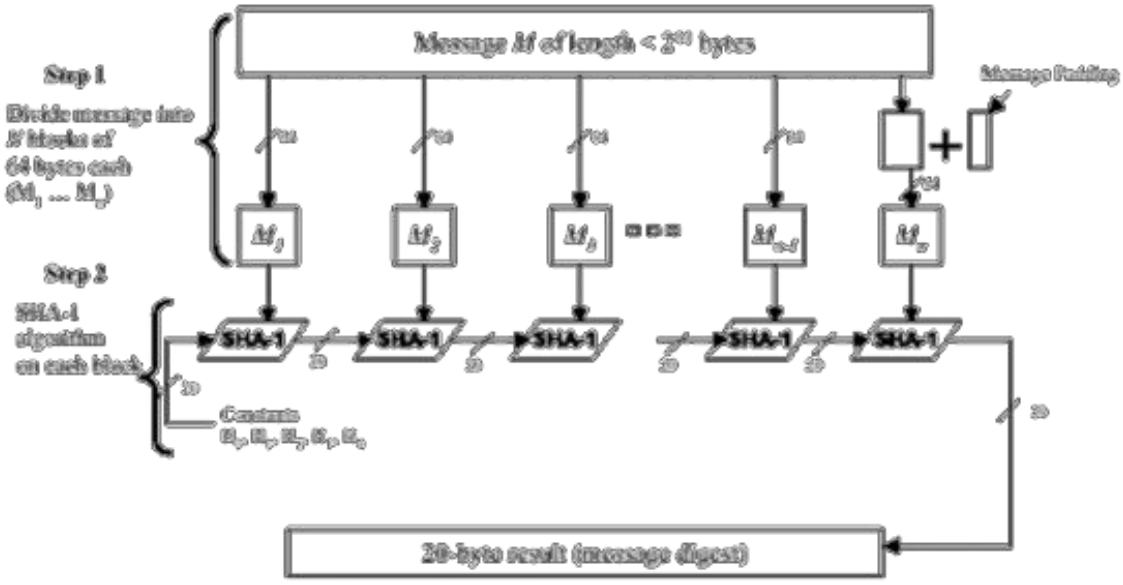
- p2(i) = (1 + 5i) mod 16
- p3(i) = (5 + 3i) mod 16
- p4(I) = 7i mod 16

MD4

- Precursor to MD5
- Design goals of MD4 (which are carried over to MD5)
- Security
- Speed
- Simplicity and compactness
- Favor little-endian architecture
- Main differences between MD5 and MD4
- A fourth round has been added.
- Each step now has a unique additive constant.
- The function g in round 2 was changed from (bc v bd v cd) to (bd v cd') to make g less symmetric.
- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
- The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.
- The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

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 264 bits and produces as output a 160-bit message digest.



The input is processed in 512-bit blocks. The overall processing of a message follows the structure of MD5 with block length of 512 bits and hash length and chaining variable length of 160 bits. The processing consists of following steps:

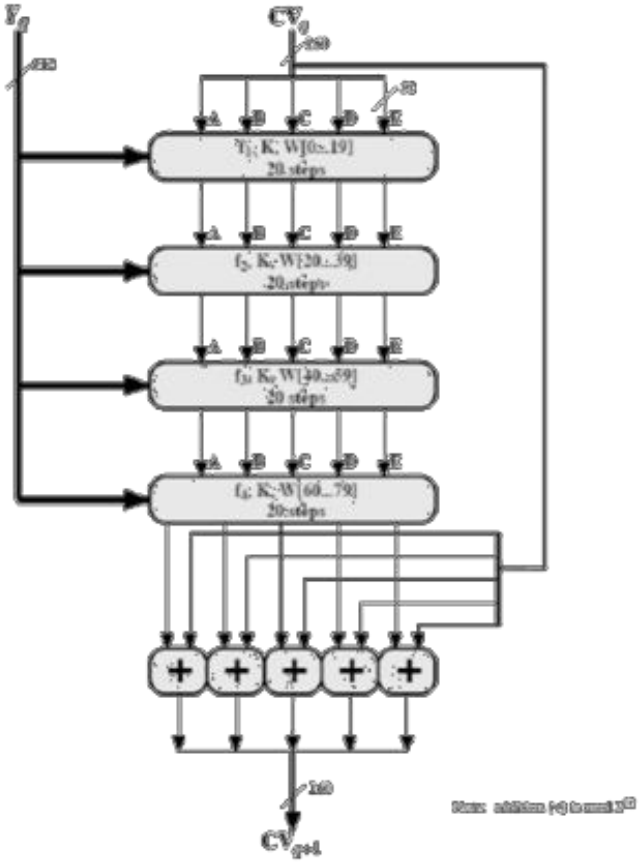
1.) ***Append Padding Bits:*** The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.

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

3.) ***Initialize MD buffer*****:**A160-bitbufferisued 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 bloc 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 CVq+1.

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

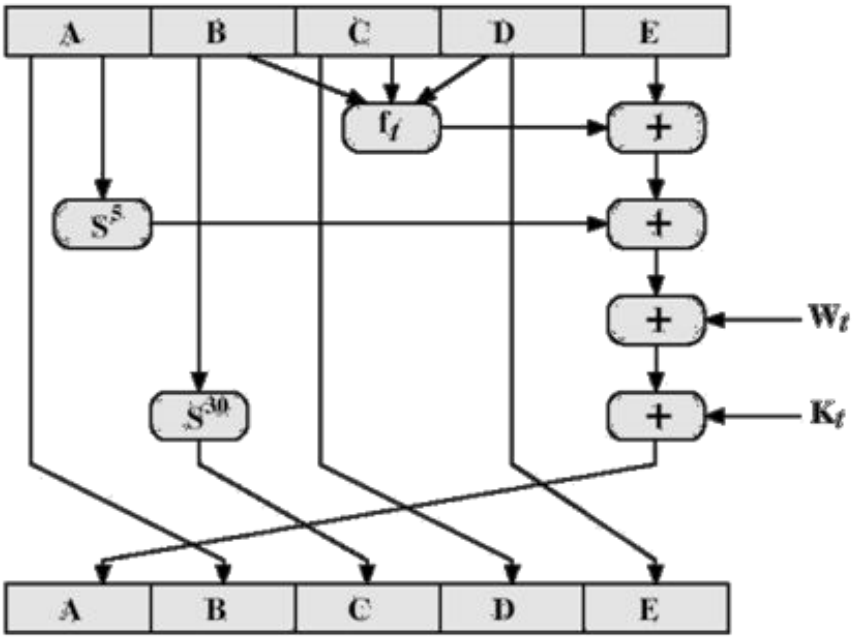


SHA-1 Processing of a Single 512-bit Block
(SHA-1 Compression Function)

The behavior of SHA-1 is as follows: **CV0 = IV CVq+1 = SUM32(CVq, ABCDEq) MD = CVL** Where, IV = initial value of ABCDE buffer ABCDEq = output of last round of processing of qth message block L = number of blocks in the message SUM32 = Addition modulo 232 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) <- (E + f(t,B,C,D) + S5(A)+ Wt+ Kt),A,S30(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 addition.



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 d recently, SHA-1 shuffles and

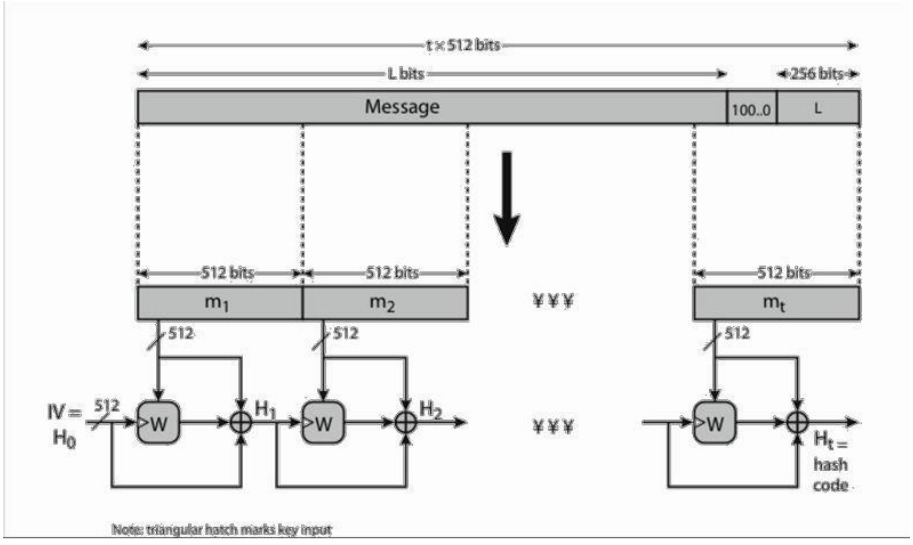
mixes them using rotates & XOR's to form more complex input, and greatly increases the difficulty of finding collisions. A sequence of logical functions f_0, f_1,..., f_{79} is used in the SHA-1. Each f_t, 0<=t<=79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. f_t(B,C,D) 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).**

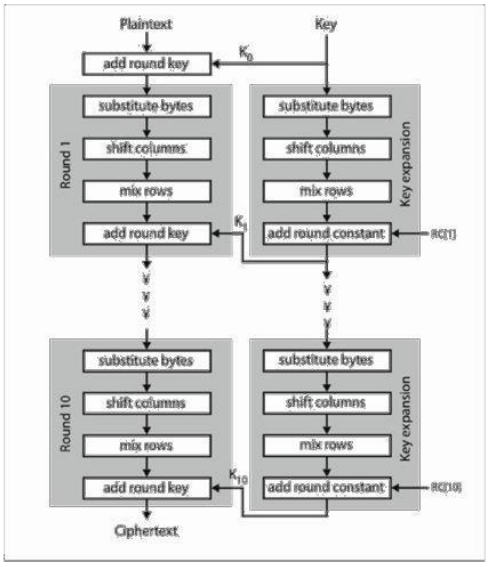
WHIRLPOOL HA H FUNCTION

- Created by Vincent Rijmen and Paulo S. L. M. Barreto
- Hashes messages of plaintext length 2^256
- Result is a 512 bit message
- Three versions have been released – WHIRLPOOL-0 – WHIRLPOOL-T – WHIRLPOOL
  - designed specifically for hash function use
  - with security and efficiency of AES
  - but with 512-bit block size and hence hash
  - similar structure & functions as AES but

- input is mapped row wise
- has 10 rounds
- a different primitive polynomial for GF(2^8)
- uses different S-box design & values
- "W" is a 512-bit block cipher
- "m" is the plaintext, split into 512 bit blocks
- "H" is the blocks formed from the hashes

WHIRLPOOL OVERVIEW





- The block cipher W is the core element of the Whirlpool hash function
- It is comprised of 4 steps.
  - Add Round Key

– Shift Columns

– Mix Rows

– Substitute bytes

Add Round Key

• During the Add Round Key step, the message is XOR'd with the key

• If this is the first message block being run through, the key is a block of all zeros

• If this is any block except the first, the key is the digest of the previous block

Shift Columns

• Starting from left to right, each column gets rotated vertically a number of bytes equal to which number column it is, from top to bottom –

Ex:

• [0,0][0,1][0,2] [0,0][2,1][1,2]

• [1,0][1,1][1,2] ------> [1,0][0,1][2,2]

• [2,0][2,1][2,2] [2,0][1,1][0,2]

Mix Rows

• Each row gets shifted horizontally by the numb r of row it is. Similar to the shift column function, but rotated left to right –

Ex:

• [0,0][0,1][0,2] [0,0][0,1][0,2]

• [1,0][1,1][1,2] ------> [1,2][1,0][1,2]

• [2,0][2,1][2,2] [2,1][2,2][0,2]

Substitute bytes

• Each byte in the message is passed through a set of s-boxes

• The output of this is then set to be the key for the next round

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 modificat on. Second is that if ever an existing hash function is to be replaced, the existing hash funct on module is removed and new module is dropped in. The last design obj ctive provides the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has ome 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.

HMAC Algorithm

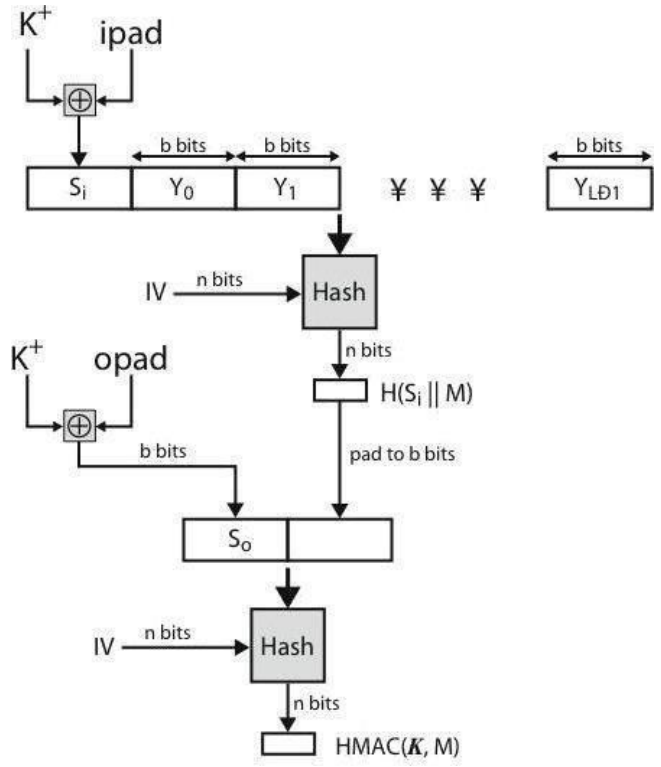
- Define the following terms

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 ≥ 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

$$HMAC_K = H[(K^+ \oplus opad) || H[K^+ \oplus ipad) || M]]$$

HMAC Structure



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.

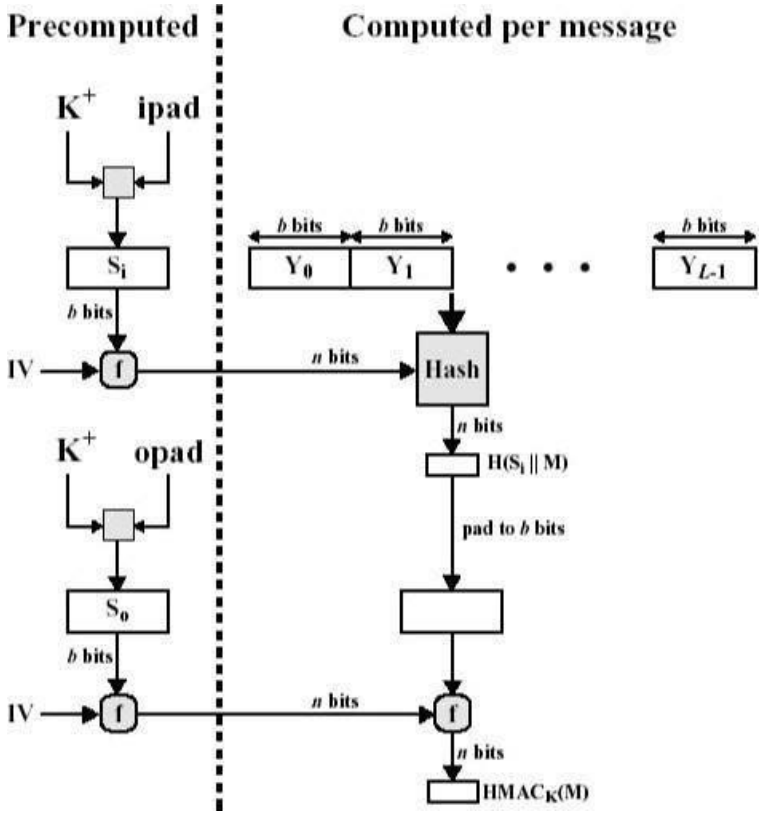
HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_0, S_i, and the block produced from the inner hash)

A more efficient implementation is possible. Two quantities are precomputed.

f(IV, (K^+ ⊕ ipad)

f(IV, (K^+ ⊕ 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.

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.

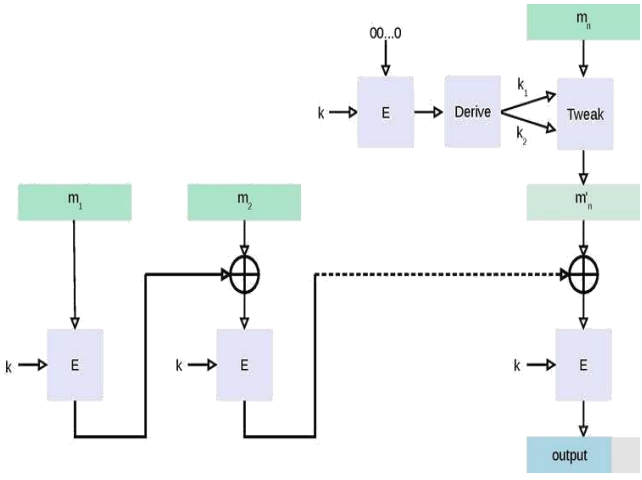
2. The attacker finds collisions in the hash function even when the IV is random and secret.

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.

CMAC

In cryptography, **CMAC** (Cipher-based Message Authentication Code)[1] is a block cipher-based message authentication code algorithm. It may be used to provide assurance of the authenticity and, hence, the integrity of binary data. This mode of operation fixes security deficiencies of CBC-MAC (CBC-MAC is secure only for fixed-length messages).

The core of the CMAC algorithm is variation of CBC-MAC that Black and Rogaway proposed and analyzed under the name XCBC[2] and submitted to NIST.[3] The XCBC algorithm efficiently addresses the security deficiencies of CBC-MAC, but requires three keys. Iwata and Kurosawa proposed an improvement of XCBC and named the resulting algorithm One-Key CBC-MAC (OMAC) in their papers.[4][5] They later submitted OMAC1[6], a refinement of OMAC, and additional security analysis.[7] The OMAC algorithm reduces the amount of key material required for XCBC. CMAC is equivalent to OMAC1.



To generate an ℓ-bit CMAC tag (t) of a message (m) using a b-bit block cipher (E) and a secret key (k), one first generates two b-bit sub-keys (k_1 and k_2) using the following algorithm (this is equivalent to multiplication by x and x^2 in a finite field GF(2^b)). Let $\ll$ denote the standard left-shift operator and $\oplus$ denote exclusive or:

1. Calculate a temporary value $k_0 = E_k(0)$.
2. If msb(k_0) = 0, then $k_1 = k_0 \ll 1$, else $k_1 = (k_0 \ll 1) \oplus C$; where C is a certain constant that depends only on b. (Specifically, C is the non-leading coefficients of the lexicographically first irreducible degree-b binary polynomial with the minimal number of ones.)
3. If msb(k_1) = 0, then $k_2 = k_1 \ll 1$, else $k_2 = (k_1 \ll 1) \oplus C$.
4. Return keys (k_1, k_2) for the MAC generation process.

As a small example, suppose $b = 4$, $C = 0011_2$, and $k_0 = E_k(0) = 0101_2$. Then $k_1 = 1010_2$ and $k_2 = 0100 \oplus 0011 = 0111_2$.

The CMAC tag generation process is as follows:

1. Divide message into b-bit blocks $m = m_1 \| \ldots \| m_{n-1} \| m_n$ where $m_1, \ldots, m_{n-1}$ are complete blocks. (The empty me age is treated as 1 incomplete block.)
2. If m_n is a complete block then $m_n' = k_1 \oplus m_n$ else $m_n' = k_2 \oplus (m_n \| 10\ldots0_2)$.
3. Let $c_0 = 00\ldots0_2$.
4. For $i = 1, \ldots, n\text{-}1$, calculate $c_i = E_k(c_{i-1} \oplus m_i)$.
5. $c_n = E_k(c_{n-1} \oplus m_n')$
6. Output $t = \text{msb}_\ell(c_n)$.

The verification process is as follows:

1. Use the above algorithm to generate the tag.
2. Check that the generated tag is equal to the received tag.

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 nd 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- ey algorithms, and the arbiter may or may not see the actual message contents. **Using Conventional encryption**

$$X \rightarrow A : M \,||\, E (Kxa ,[IDx \,||\, H (M)])$$

$$A \rightarrow Y : E(Kay ,[IDx \,||\, M \,||\, E (Kxa ,[IDx \,||H(M))]) \,||\, T])$$

- It is assumed that the sender X and the arbiter A share a secret key Kxa and that A and Y share secret key Kay. 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 IDx of X plus the hash value, all encrypted using Kxa.
- A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with Kay. The message includes IDx, the original message from X, the signature, and a timestamp.

- Arbiter sees message
- 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

X → A : IDx ||E(PRx,[IDx|| E (PUy, E(PRx, M))])

A → Y : E(PRa, [IDx ||E (PUy, E (PRx, M))|| T])

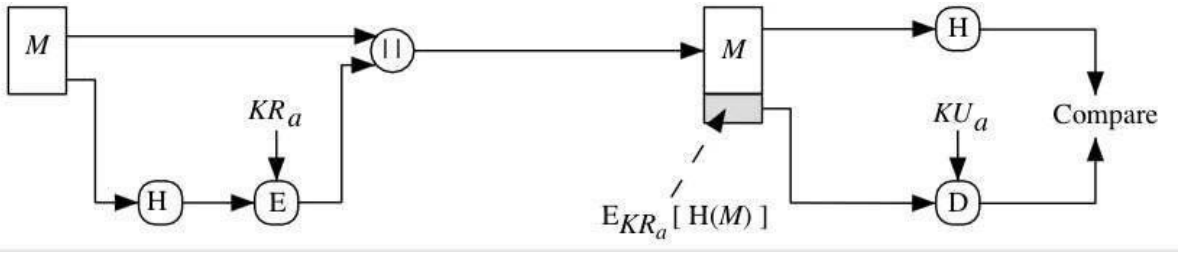
X double encrypts a message M first with X's private key, PRx, and then with Y's public key, PUy. This is a signed, secret version of the message. This signed message, together with X's identifier , is encrypted again with PRx and, together with IDx, is sent to A. The inner, double encrypted message 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 PRx). Then A transmits a message to Y, encrypted with PRa. The message includes IDx, the double encrypted message, and timestamp.
- Arbiter does not see message

Digital Signature Standard (DSS)

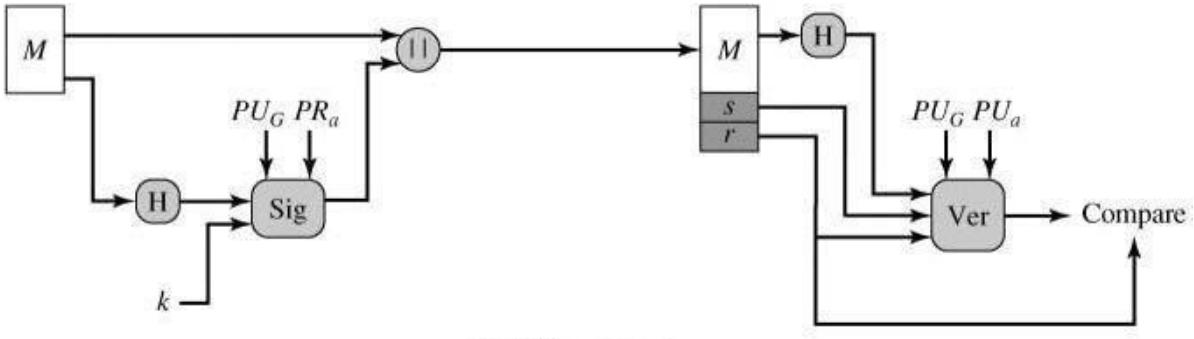
The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Ha h Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (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 (PRa) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PUG).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 (PUa), which is paired with the sender's private key. The outp t 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.

KNAPSACK ALGORITHM

Public-Key cryptography was invented in the 1970s by Whitfield Diffie, Martin Hellman and Ralph Merkle.

Public-key cryptography needs two keys. One key tells you how to encrypt (or code) a message and this is "public" so anyone can use it. The other key allows you to decode (or decrypt) the message. This decryption code is kept secret (or private) so only the person who knows the key can decrypt the message. It is also possible for the person

with the private key to encrypt a message with the private key, then anyone holding the public key can decrypt the message, although this seems to be of little use if you are trying to keep something secret!

The First General Public-Key Algorithm used what we call the Knapsack Algorithm. Although we now know that this algorithm is not secure we can use it to look at how these types of encryption mechanisms work.

The knapsack algorithm works like this:
Imagine you have a set of different weights which you can use to make any total weight that you need by adding combinations of any of these weights together. Let us look at an example:
Imagine you had a set of weights 1, 6, 8, 15 and 24. To pack a knapsack weighing 30, you could use weights 1, 6, 8 and 15. It would not be possible to p ck a knapsack that weighs 17 but this might not matter.
You might represent the weight 30 by the binary code 11110 (one 1, one 6, one 8, one 15 and no 24).
Example:

Plain text	10011	11010	01011	00000
Knapsack	1 6 8 15 24	1 6 8 15 24	1 68 15 24	1 68 15 24
Cipher text	1 + 15 + 24 = 40	1 + 6 + 15 = 22	6 +15 + 24 = 45	0 =0

What total weights is it possible to make?

So, if someone sends you the code 38 this can only have come from the plain text 01101. When the Knapsack Algorithm is used in public key cryptography, the idea is to create two different knapsack problems. One is easy to solve, the other not. Using the easy knapsack, the hard knapsack is derived from it. The hard knapsack becomes the public key. The easy knapsack is the private key. The public key can be used to encrypt messages, but cannot be used to decrypt messages. The private key decrypts the messages.

The Superincreasing Knapsack Problem

An easy knapsack problem is one in which the weights are in a superincreasing sequence. A superincreasing sequence is one in which the next term of the sequence is greater than the sum of all preceding terms. For example, the set {1, 2, 4, 9, 20, 38} is superincreasing, but the set {1, 2, 3, 9, 10, 24} is not because 10 < 1+2+3+9.

It is easy to solve a superincreasing knapsack. Simply take the total weight of the knapsack and compare it with the largest weight in the sequence. If the total weight is less than the number, then it is not in the knapsack. If the total weight is greater then the number, it is in the knapsack. Subtract the number from the total, and compare with the next highest number. Keep working this way until the total reaches zero. If the total doesn't reach zero, then there is no solution.

So, for example, if you have a knapsack that weighs 23 that has been made from the weights of the superincreasing series {1, 2, 4, 9, 20, 38} then it does not contain the weight 38 (as 38 > 23)
but it does contain the weight 20; leaving 3; which does not contain the weight 9 still leaving 3; which does not contain the weight 4 still leaving 3;
which contains the weight 2, leaving 1; which contains the weight 1.
The binary code is therefore 110010.

It is much harder to decrypt a non-superincreasing knapsack problem. Give a friend a non-super increasing knapsack and a total and see why this is the case.
One algorithm that uses a superincreasing knapsack for the private (easy) key and a non-superincreasing knapsack for the public key was created by Merkle and Hellman They did this by taking a superincreasing knapsack problem and converting it into a non-superincreasing one that could be made public, using modulus arithmetic.

Making the Public Key

To produce a normal knapsack sequence, take a superincreasing sequence; e.g. {1, 2, 4, 10, 20, 40}. Multiply all the values by a number, n, modulo m. The modulus should be a number greater than the sum of all the numbers in the sequence, for example, 110. The

multiplier should have no factors in common with the modulus. So let's choose 31. The normal knapsack sequence would be:

1×31 mod(110) = 31
2×31 mod(110) = 62
4×31 mod(110) = 14
10×31 mod(110) = 90
20×31 mod(110) = 70
40×31 mod(110) = 30

So the public key is: {31, 62, 14, 90, 70, 30} and
the private key is {1, 2, 4, 10, 20.40}.

Let's try to send a message that is in binary code:
100100111100101110
The knapsack contains six weights so we need to split the mess ge into groups of six:
100100
111100
101110
This corresponds to three sets of weights with totals as follows
100100 = 31 + 90 = 121
111100 = 31+62+14+90 = 197
101110 = 31+14+90+70 =205
So the coded message is 121 197 205.

Now the receiver has to decode the message...
The person decoding must know the two numbers 110 and 31 (the modulus and the multiplier). Let's call the modulus "m" and the number you multiply by "n".
We need n^{-1}, which is a multiplicative inverse of n mod m, i.e. $n(n^{-1}) = 1 \bmod m$
In this case I have calculated n^{-1} to be 71.

All you then have to do is multiply each of the codes 71 mod 110 to find the total in the knapsack which contains {1, 2, 4, 10, 20, 40} and hence to decode the message. The coded message is 121 197 205:

121×71 mod(110) = 11 = 100100
197×71 mod(110) = 17 = 111100
205×71 mod(110) = 35 = 101110

The decoded message is:
100100111100101110.
Just as I thought!

Simple and short knapsack codes are far too easy to break to be of any real use. For a knapsack code to be reasonably secure it would need well over 200 terms each of length 200 bits.

AUTHENTICATION APPLICATIONS

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an o en 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 necessary information 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.

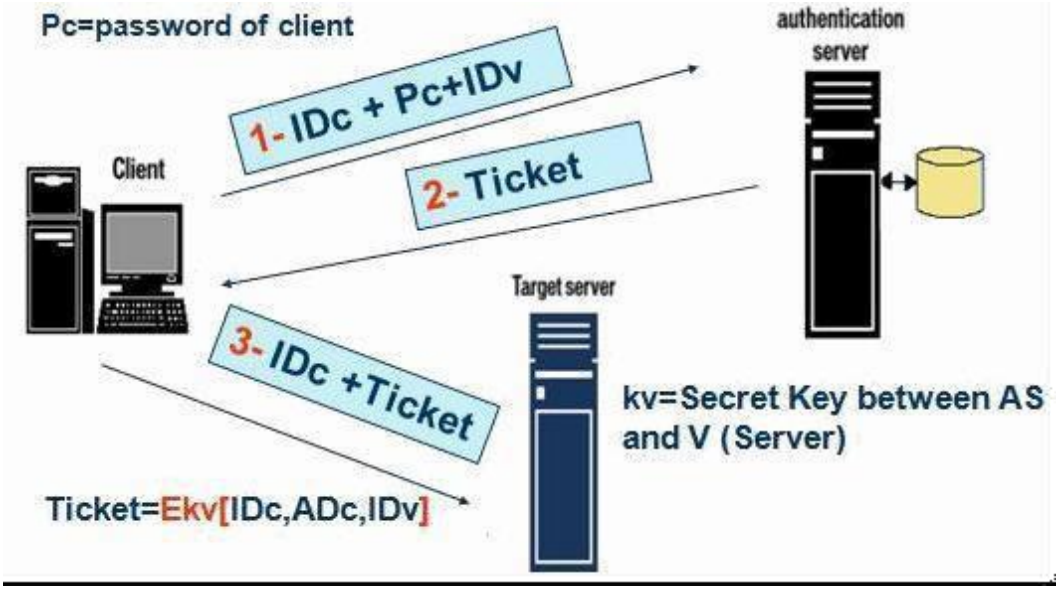
Transparent: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.

Scalable: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture

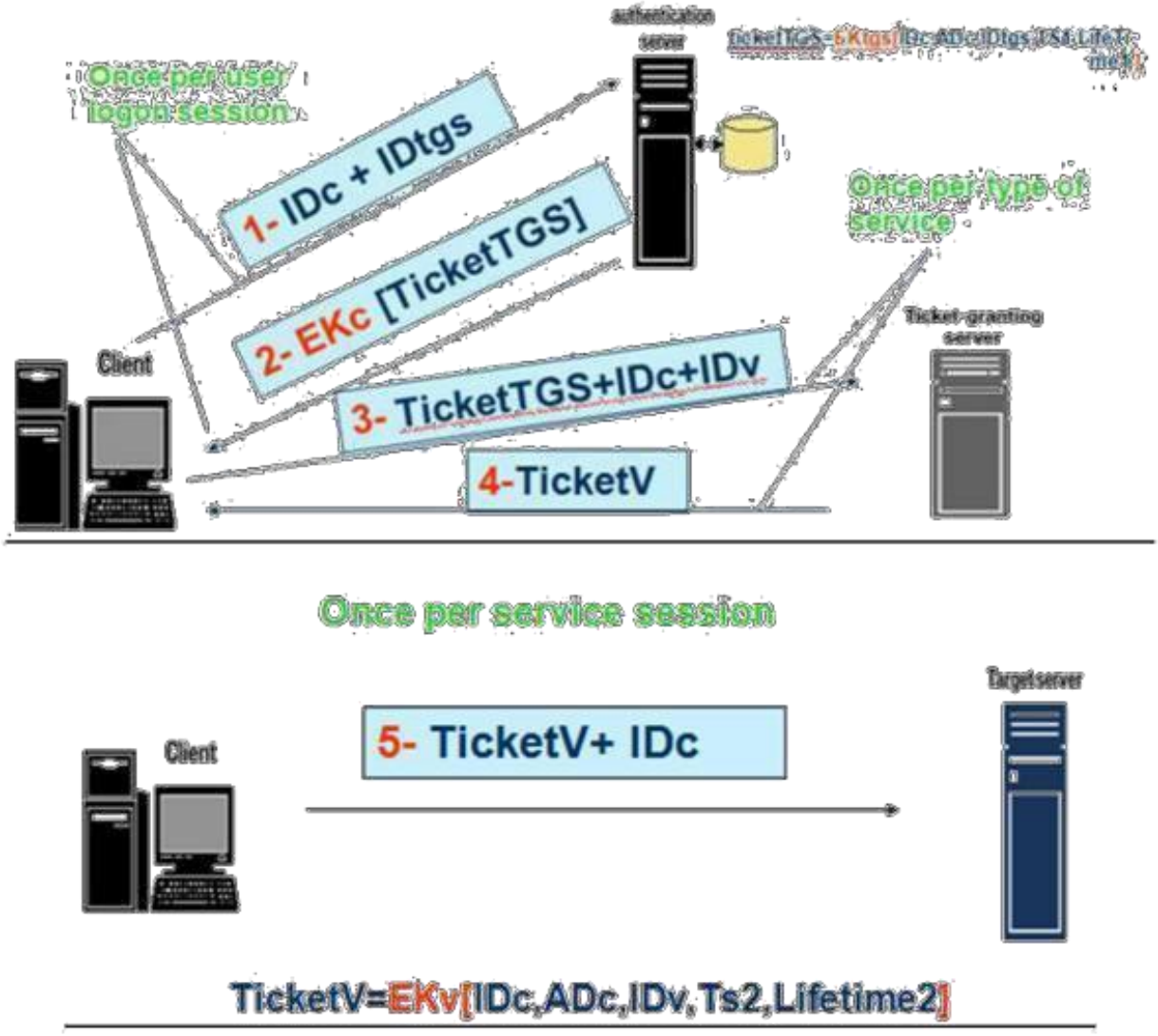
Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Vers on 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

KERBEROS VERSION 4

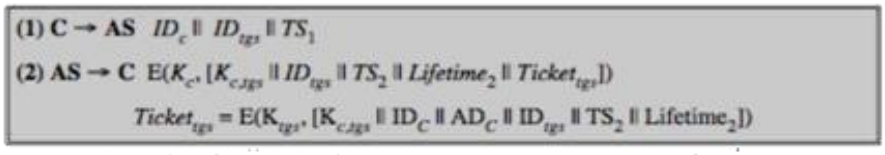
1.) **SIMPLE DIALOGUE**:



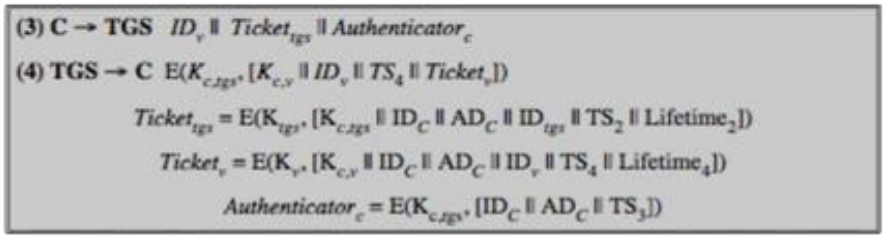
MORE SECURE DIALOGUE



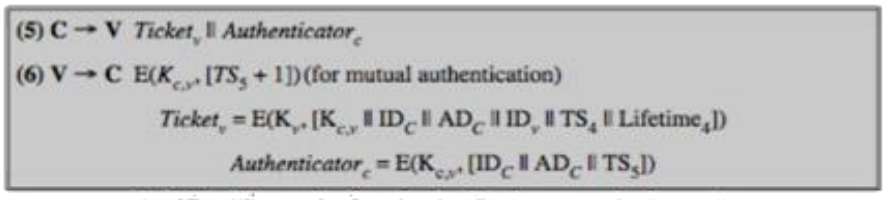
The Version 4 Authentication Dialogue The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.



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



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



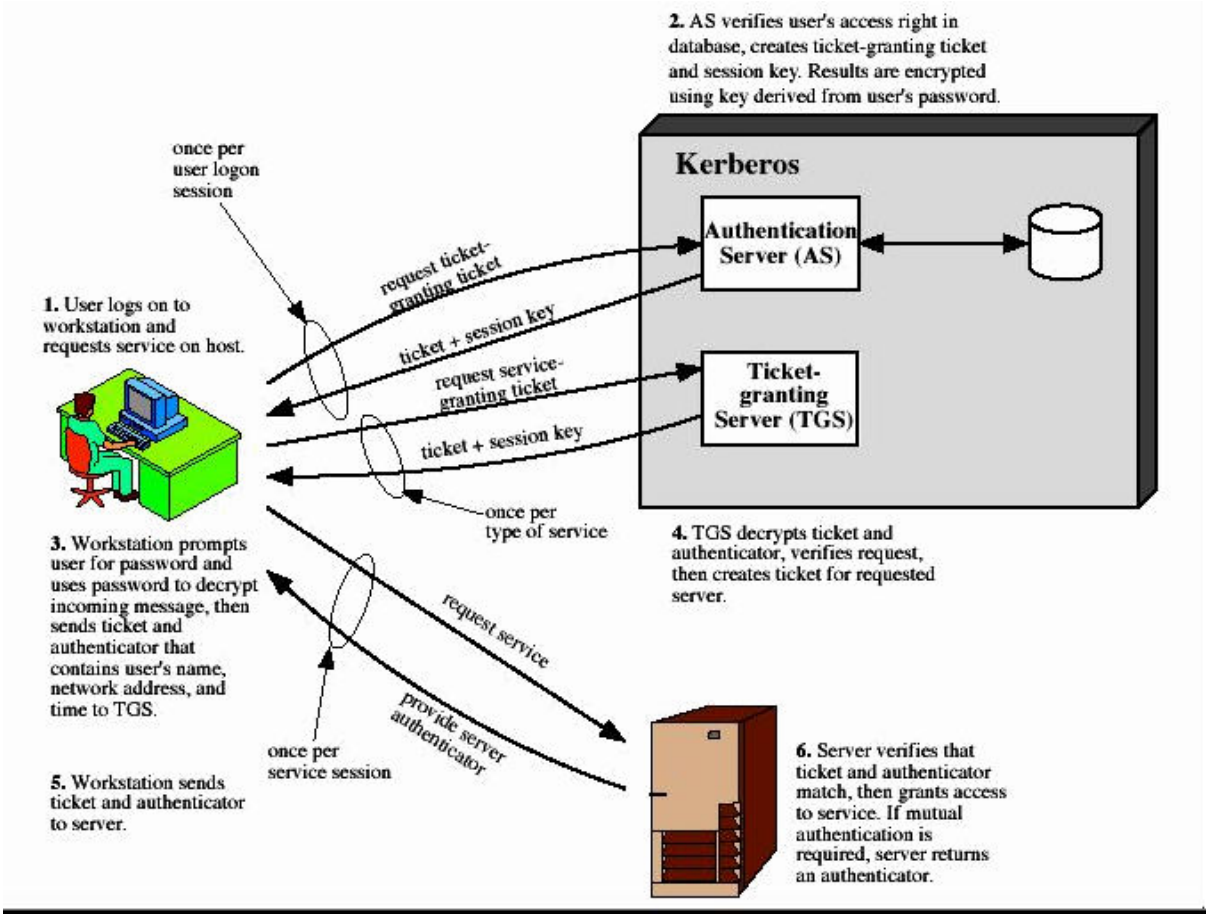
(c) Client/Server Authentication Exchange to obtain service

There is a problem of captured ticket-granting tickets nd 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 time stamp, 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.

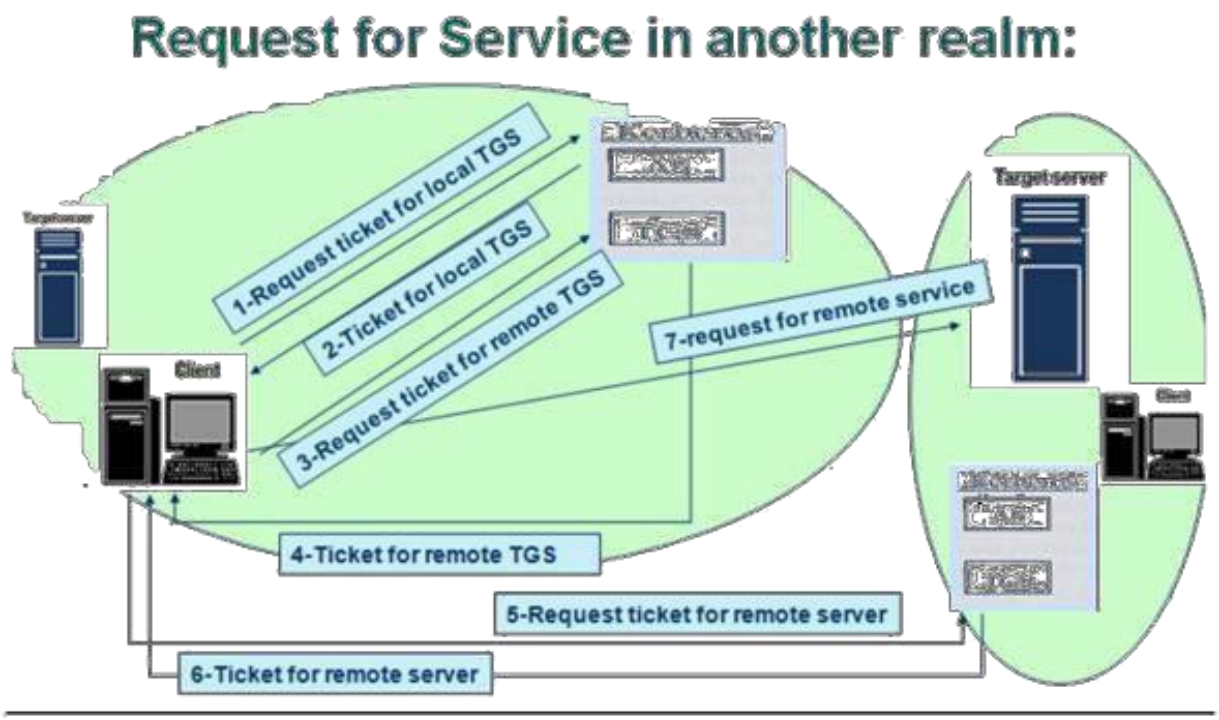
The server can decrypt the ticket, recover the session key, and decrypt the authenticator.If mutual authentication is required, the server can reply as shown in message (6).

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 key 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.



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

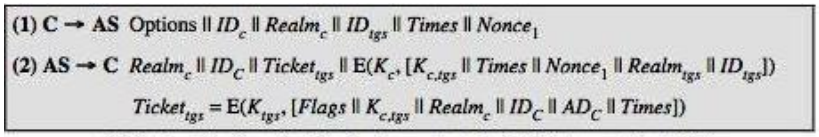
– From: the desired start time for the ticket

– Till: the requested expiration time

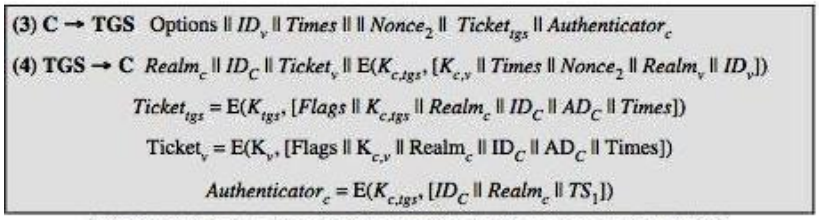
– Rtime: requested renew-till time

- Nonce: A random value to assure the response is fresh

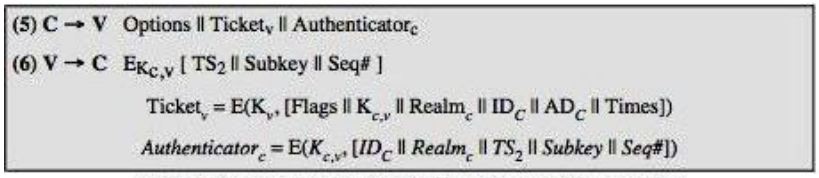
The basic Kerberos version 5 authentication dialogue is shown here First, consider the **authentication service exchange.**



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



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



(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:

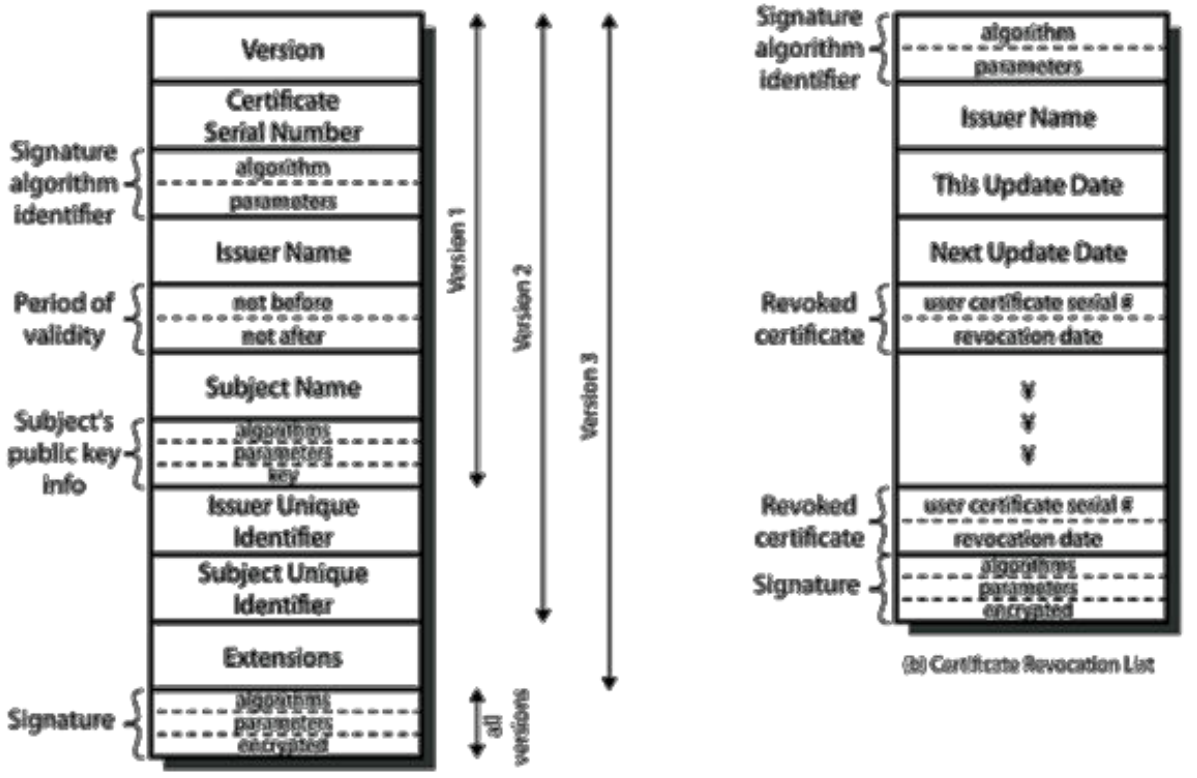
- User's passwords are never sent across the network, encrypted or in plain text
- Secret keys are *only* passed across the network in encrypted form
- Client and server systems mutually authenticate
- It limits the duration of their users' authentication.
- Authentications are **reusable** and **durable**
- 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 partMediaoftheX.500series of recommendations that define a directory service. The directory is, in effect, 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 w ll 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 a umed 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.

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+)



Media

- subject unique identifier (v2+)
- extension fields (v3)
- signature (of hash of all fields in certificate)

The standard uses the following notation to define certificate:

CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap}

Where Y<<X>>= the certificate of ser 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:

1. Any user with CA's public key can verify the user public key that was certified
2. No party other than the CA can modify the certificate without being detected
3. 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

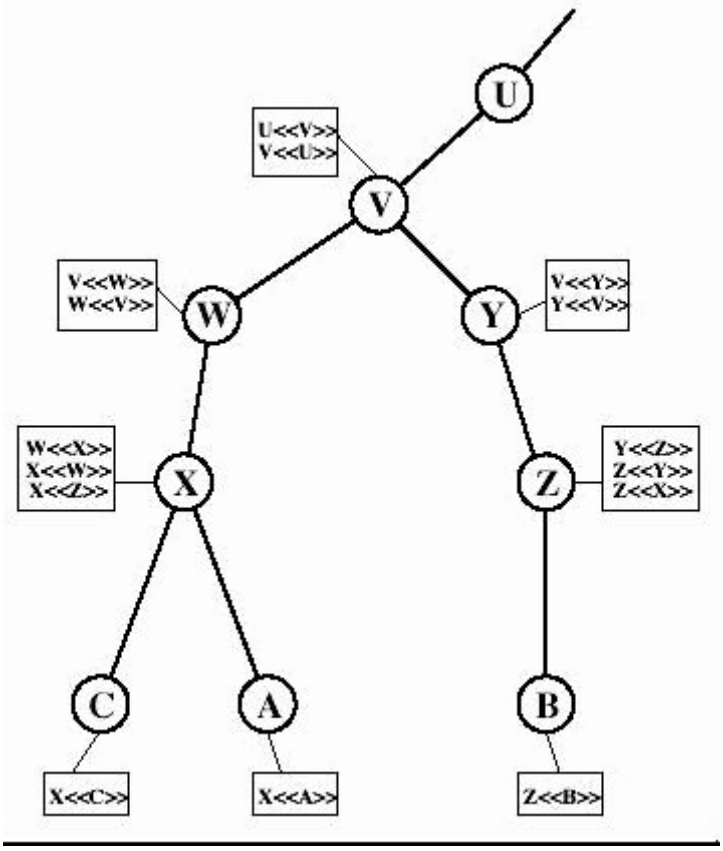
X.509 CA Hierarchy

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>>



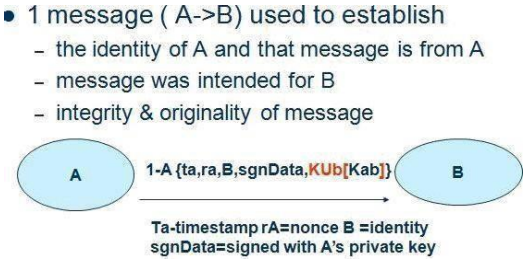
Revocation of Certificates Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- 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.

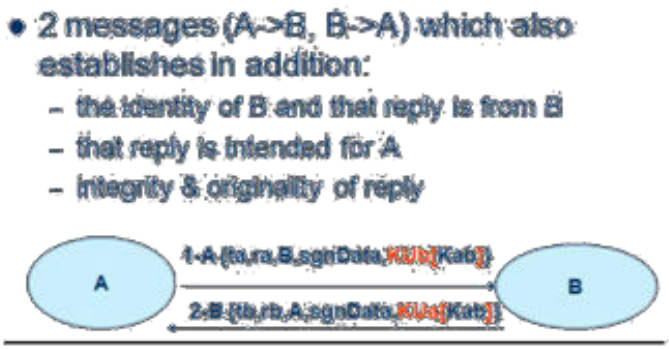
Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory. Each **certificate revocation list (CRL) posted** to the directory is signed by the issuer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

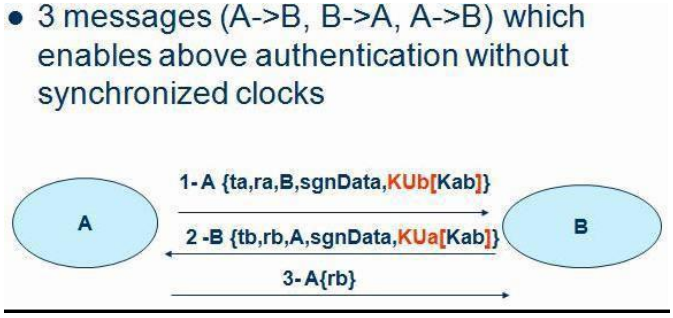
X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, there by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side. 1. One-Way Authentication: One way authentication involves a single transfer of information 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 ey for B.



Two-Way Authentication: Two-way authentication thus permits both parties in a communication 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.



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.



BIOMETRIC AUTHENTICATION

Biometric authentication is a type of system that relies on the unique biological characteristics of individuals to verify identity for secure access to electronic systems.

Biometric verification is considered a subset of biometric authentication. The biometric technologies involved are based on the ways in which individuals can be uniquely identified through one or more distinguishing biological traits, such as fingerprints, hand geometry, earlobe geometry, retina and iris patterns, voice waves, keystroke dynamics, DNA and signatures. Biometric authentication is the application of that proof of identity as part of a process validating a user for access to a system. Biometric technologies are used to secure a wide range of electronic communications, including enterprise security, online commerce and banking -- even just logging in to a computer or smartphone.

Biometric authentication systems compare the current biometric data capture to stored, confirmed authentic data in a database. If both samples of the biometric data match,

authentication is confirmed and access is granted. The process is sometimes part of a multifactor authentication system. For example, a Smartphone user might log on with his personal identification number (PIN) and then provide an iris scan to complete the authentication process.

Types of biometric authentication technologies:

Retina scans produce an image of the blood vessel pattern in the light-sensitive surface lining the individual's inner eye.

Iris recognition is used to identify individuals based on unique patterns within the ring-shaped region surrounding the pupil of the eye.

Finger scanning, the digital version of the ink-and-paper fingerprinting process, works with details in the pattern of raised areas and branches in human finger image.

Finger vein ID is based on the unique vascular pattern in an individual's finger.

Facial recognition systems work with numeric cod s called face prints, which identify 80 nodal points on a human face.

Voice identification systems rely on characteristics created by the shape of the speaker's mouth and throat, rather than more variable conditions.

Once seen mostly in spy movies (where it might be used to protect access to a top-secret military lab, for example), biometric authentication is becoming relatively commonplace. In addition to the security provided by hard-to-fake individual biological traits, the acceptance of biometric verification has also been driven by convenience: One can’t easily forget or lose ones biometrics.

The history of biometric verification:

The oldest known use of biometric verification is fingerprinting. Thumbprints made on clay seals were used as a means of unique identification as far back as ancient China. Modern biometric verification has become almost instantaneous, and is increasingly accurate with the advent of computerized databases and the digitization of analog data.

The market for biometrics products is still too fractured to name specific top providers. The physical characteristics of the biometrics products available today vary from the mundane, such as fingerprinting, to the esoteric, like typing speeds and electrophysiological signals.

Until recently, biometrics was typically used at a physical security level – protecting facilities at military bases or impenetrable bank vaults, for example. But, because single-factor authentication methods are easy to break, companies have started looking to two-factor solutions, like biometrics.

However, the following five fundamental barriers may limit the growth of biometric authentication:

1. Biometrics can be complicated and costly to deploy. All biometric deployments require installation of their own hardware and application servers.
2. The market is still fractured. Should you buy finger print reader, a voice recognition system or an iris scanner? Since each product differs greatly in its approach and installation, it is difficult to compare them during a typical company bid process.
3. Biometric data is like any other data. It its on servers, which are bait for hackers if not properly hardened and sec red. Therefore, when reviewing any biometric product, make sure it transmits data securely, meaning encrypted, from the biometric reader back to the authenticating server. And, make sure the authenticating server has been hardened, patched and protected.
4. Biometric readers are prone to errors. Fingerprints can smudge, faces and voices can be changed and all of them can be misread, blocking a legitimate user, or permitting access to an unauthorized or malicious user.
5. Difficulties with user acceptance. Properly trained employees may be willing to use biometrics devices, but customers, like those logging on to your Web site, may be more reluctant to use – or worse, forced to purchase – a device that's difficult to use or makes doing business, such as banking, on your site, a hassle instead of a

convenience. And both your employees and customers may be squeamish about exposing their eyes to devices like iris scanners, even if they appear harmless.

Despite these issues, biometrics is slowly gaining acceptance for two-factor authentication purposes. The products are getting better, lighter and easier to use. Error rates are going down, and fingerprint readers installed on tokens and laptops are getting smaller and less intrusive. And, like the rest of the security product industry, vendors will eventually merge and consolidate, uniting a fractured market, which will make it easier to choose a product that suits your business needs.

UNIT-4

***Email Privacy**: Pretty Good Privacy (PGP) and S/MIME. **IP Security:** IP Security Overview, IP Security Architecture, Authentication Header, Encapsulating Security Payload, Combining Security Associations and Key Management.*

Pretty Good Privacy

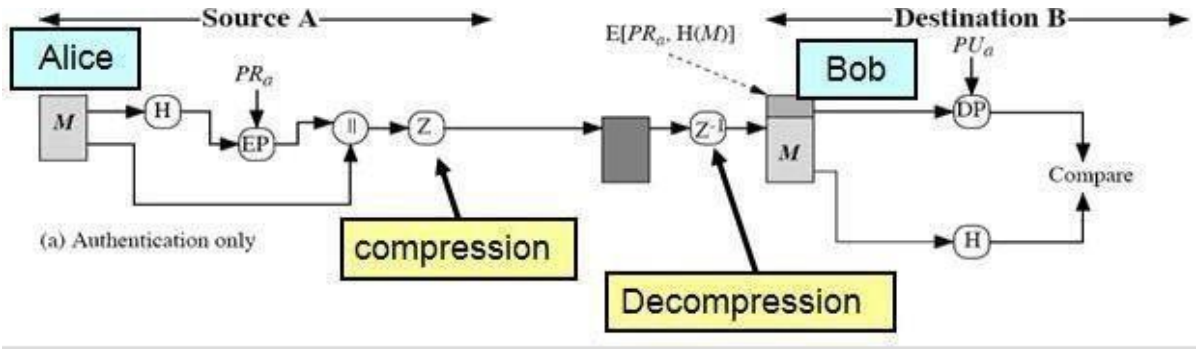
In virtually all distributed environments, electronic mail is the most heavily used network-based application. But current email services are roughly like "postcards", anyone who wants could pick it up and have a look as it's in transit or sitting in the recipients mailbox. PGP provides a confidentiality 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 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/Verif)*
- *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	= symm tric decryption
H	= hash function
||	= concatenation
Z	= compression using ZIP algorithm
R64	= conversion to radix 64 ASCII format

PGP Operation- Authentication



1. sender creates message

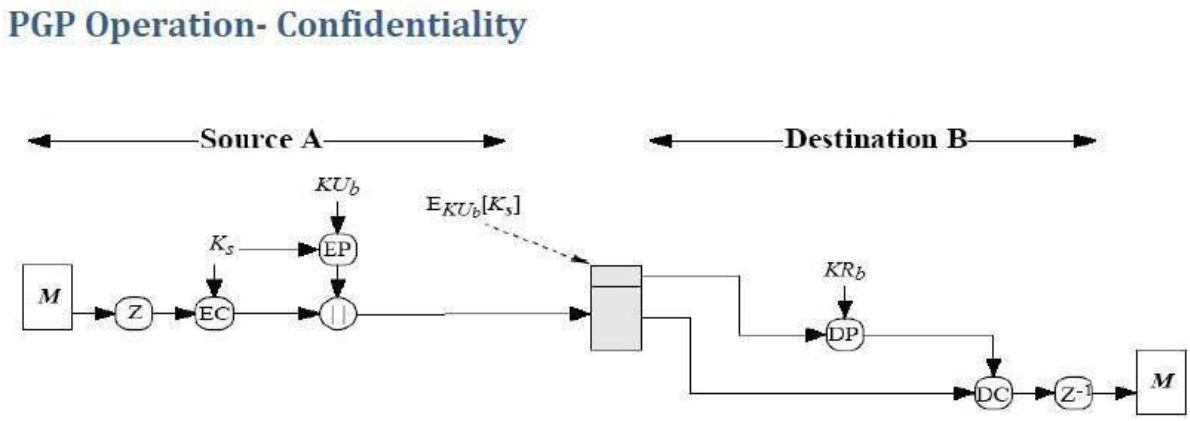
2. use SHA-1 to generate 160-bit hash of message

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

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

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

PGP Operation- Confidentiality



Sender:

1. Generates messageSkyupsandarandomnumber (session key) only for this message

2. Encrypts message with the session key ing AES, 3DES, IDEA orCAST-128

3. Encrypts session key itself with reci ient's public key using RSA

4. Attaches it to message

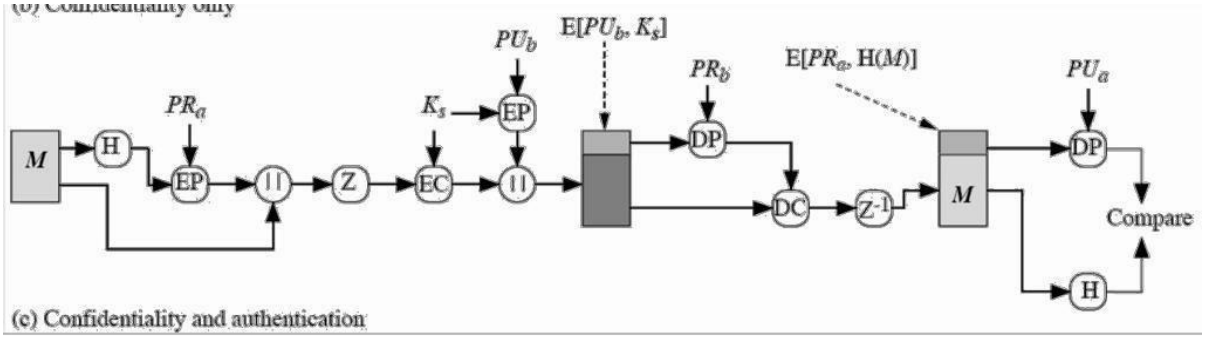
Receiver:

1. Recovers session key by decrypting using his private key

2. 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



can use both services on same message o create signature & attach to message o encrypt both message & signature

o attach RSA/ElGamal encrypted session key o

is called ***authenticated confidentiality***

PGP Operation – Compression

As a default, PGP compresses the messageMediaafterapplyngthe signature but before encryption. This has the benefit of saving space both for -ma l transmission and for file storage. The placement of the compression algorithm, in icated by Z for compression and Z-1 for decompression is critical. The compression algorithm used is ZIP.

-
- The signature is generated before compression for two reasons:

1. so that one can store only the ncom re ed message together with signature for later verification

2. 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

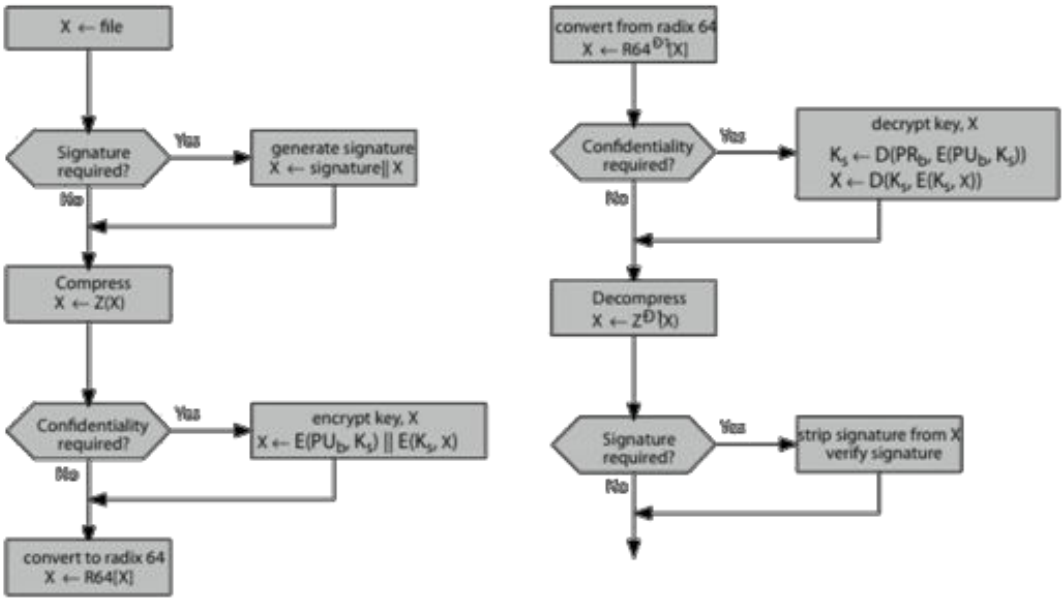
of converting the raw 8-bit binary stream to a stream of printable ASCII characters. It uses radix-64 conversion, in which each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message 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.



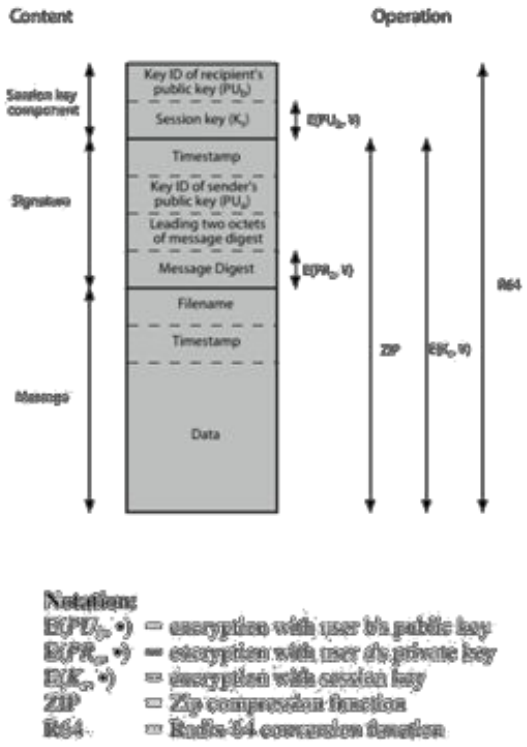
(a) Generic Transmission Diagram (from A)

(b) Generic Reception Diagram (to B)

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 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

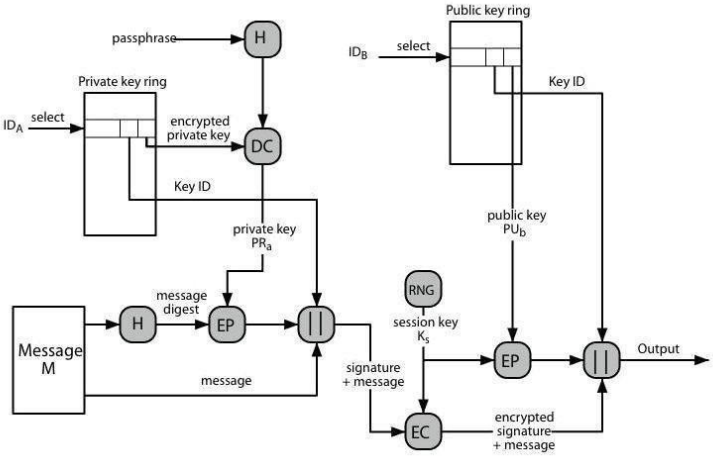


The *session key component* includes the session key and the dentifier of the recipient's public key that was used by the sender to encrypt the session key. The entire block is usually encoded with radix-64 encoding.

PGP 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.



The sending PGP entity performs the following steps:

Signing the message

a. PGP retrieves the sender's private key from the private-key ring using your user-id as an index. If your user-id was not provided in the command, the first private key on the ring is retrieved.

b. PGP prompts the user for the passphrase to recover the unencrypted private key. c.

The signature component of the message is constructed

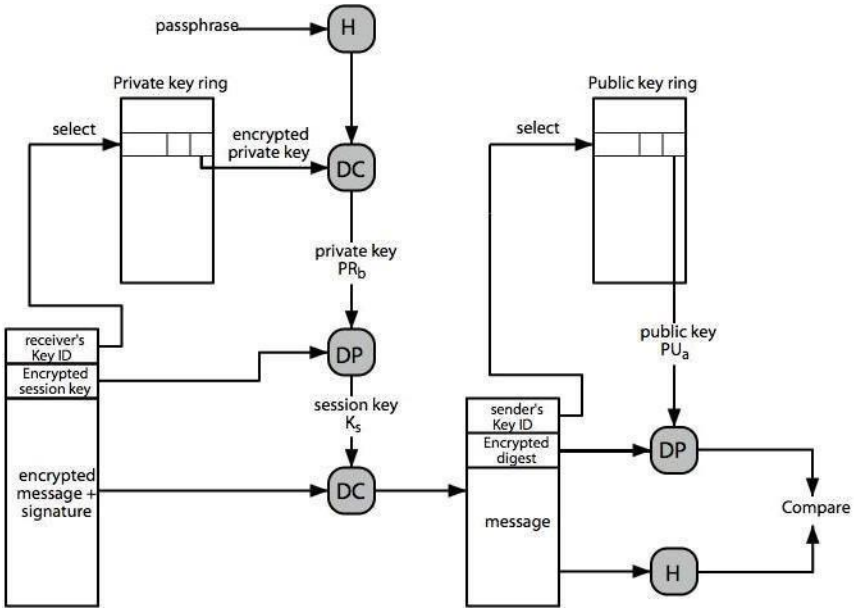
Encrypting the message

a. PGP generates a session key and encrypts the message.

b. PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.

c. The session key component of the message is constructed.

Message Reception



The receiving PGP entity performs the following steps:

Decrypting the message

a. 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.

b. PGP prompts the user for the passphrase to recover the unencrypted private key.

c. PGP then recovers the session key and decrypts the message.

Authenticating the message

a. 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. b. PGP recovers the transmitted message digest.

c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

S/MIME

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:

1. delete, add, or reorder CR and LF character
2.truncate or wrap lines longer than 76 character
3.remove trailing white space (tabs and spaces)
4.pad lines in a message to the same length convert
5.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:

1. Five new message header fields are defined, which prove information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. 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:

• *MIME-Version:* Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

• *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.

• *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.

• Content-ID: Used to identify MIME entities uniquely in multiple contexts.

• *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 uninterrupted 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 cross 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:

• **Enveloped data**: This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.

• **Signed data**: A digital signature is formed by taking the message digest of the content 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.

• **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 canMediaviewthemessge content, although they cannot verify the signature.

• **Signed and enveloped data**: Signed-only and ncrypted-only entities may be nested, so that encrypted data may be signed and sign data or clear-signed data may be encrypted.

IP SECURITY OVERVIEW

Definition: 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.

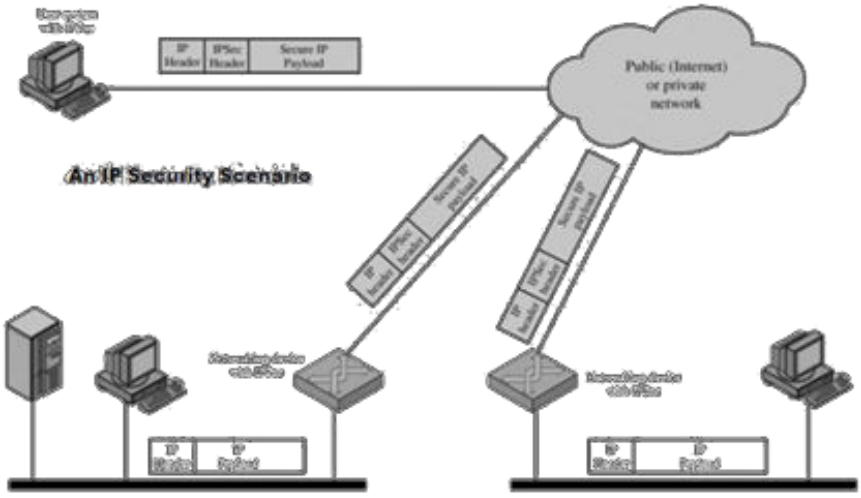
• ***Establishing extranet and intranet connectivity with p rtners***: IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.

• ***Enhancing electronic commerce security***: Ev n 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 in v du l users who dial into the WAN. Such user workstations must implement the IPS c 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 by ass

• 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 sub network 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

IP SECURITY ARCHITECTURE

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:

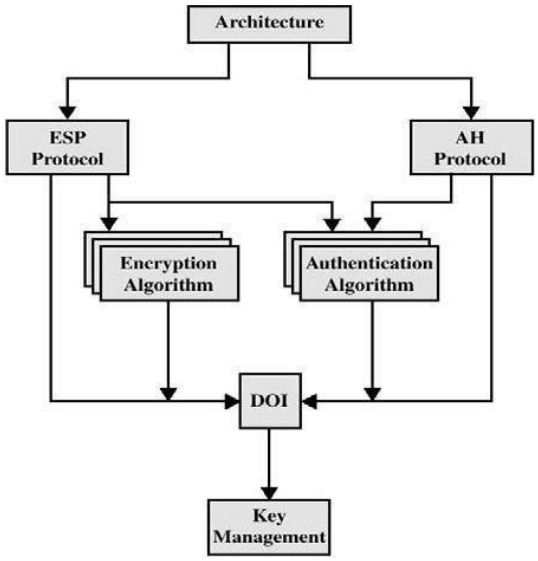
• RFC 2401: An overview of a security architecture

• 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 opt on l 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 by the IETF. The documents are divided into seven groups, as depicted in following fig re:



• **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 .

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 secur ty protocols, it uses Security Associations (SA) to specify the protocols to be us . SA s a database record which specifies security parameters controlling security op rations. 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 transmis ion 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 PI 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 implementat ons).

• **IPSec Protocol Mode:** Tunnel, transport, or w ldcard (required for all implementations). These modes are discussed lat r 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

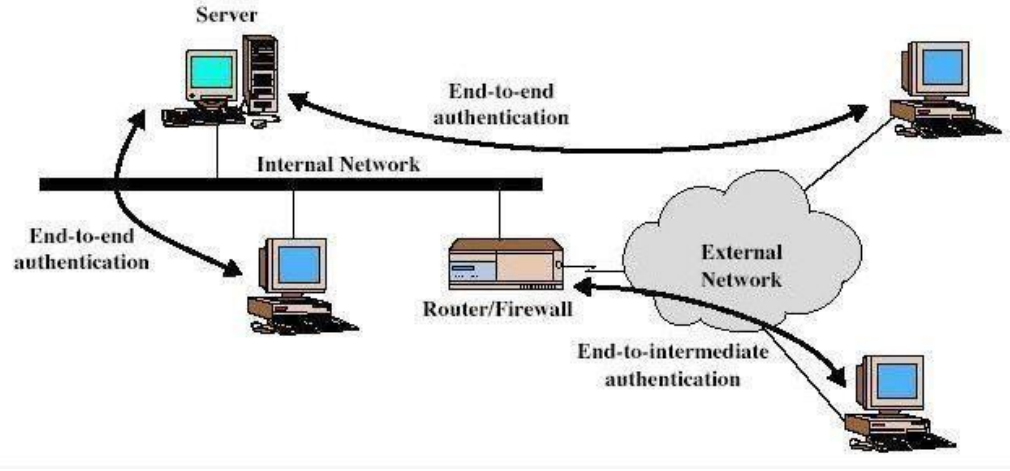
• **Transport mode**: the IP sec header is inserted just after the IP header –this contains the security information, such as SA identifier, encryption, authentication

- 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

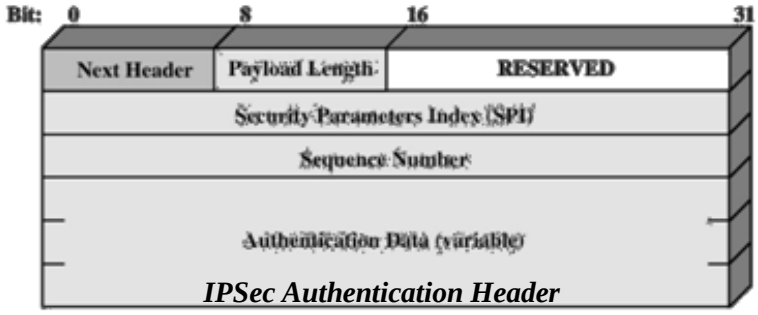
- Typically used in firewall-to-firewall
- communication Provides protection for the
- whole IP packet

No routers along the way will be able (and will not need) to check the content of the packets



AUTHENTICATION HEADER

The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feat re 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):** IdentifiesMediasecurityssociation.

• **Sequence Number (32 bits):** A monotonically increas ng counter value, discussed later.

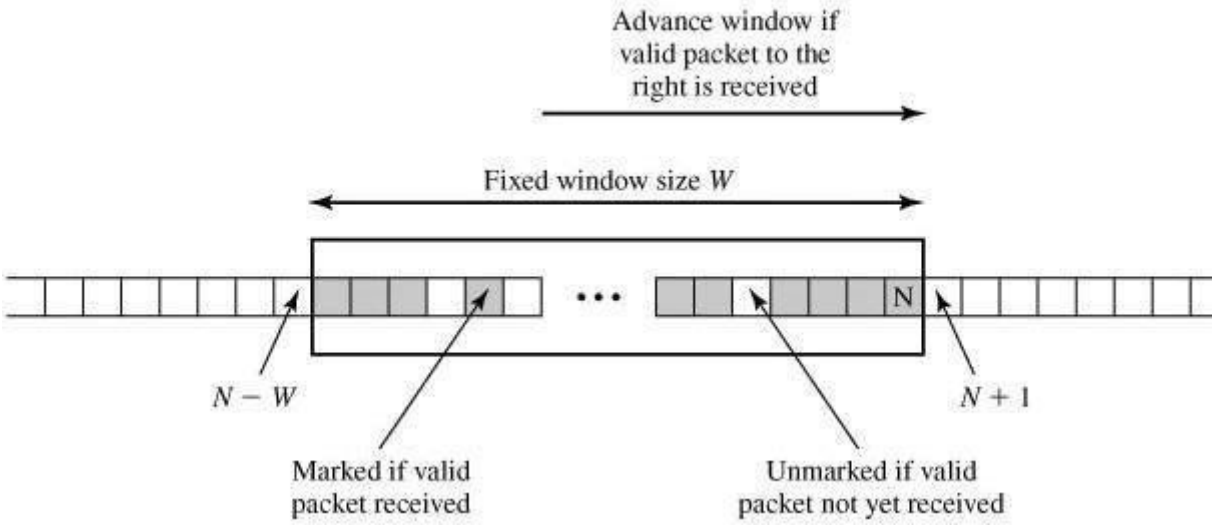
• **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet.

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 lot 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

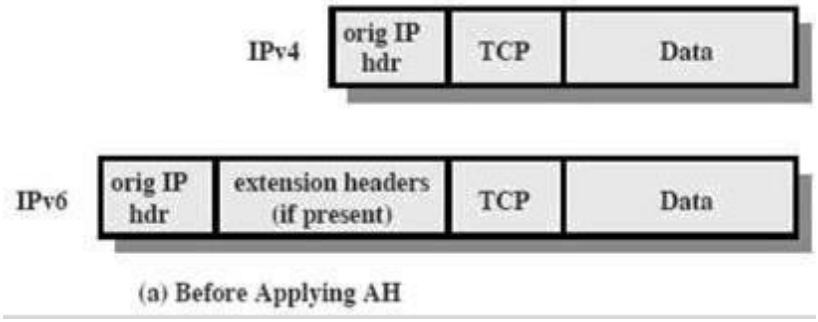
• IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival is unpredictable are set to zero for purposes of calculation at both source and destination.

• 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.

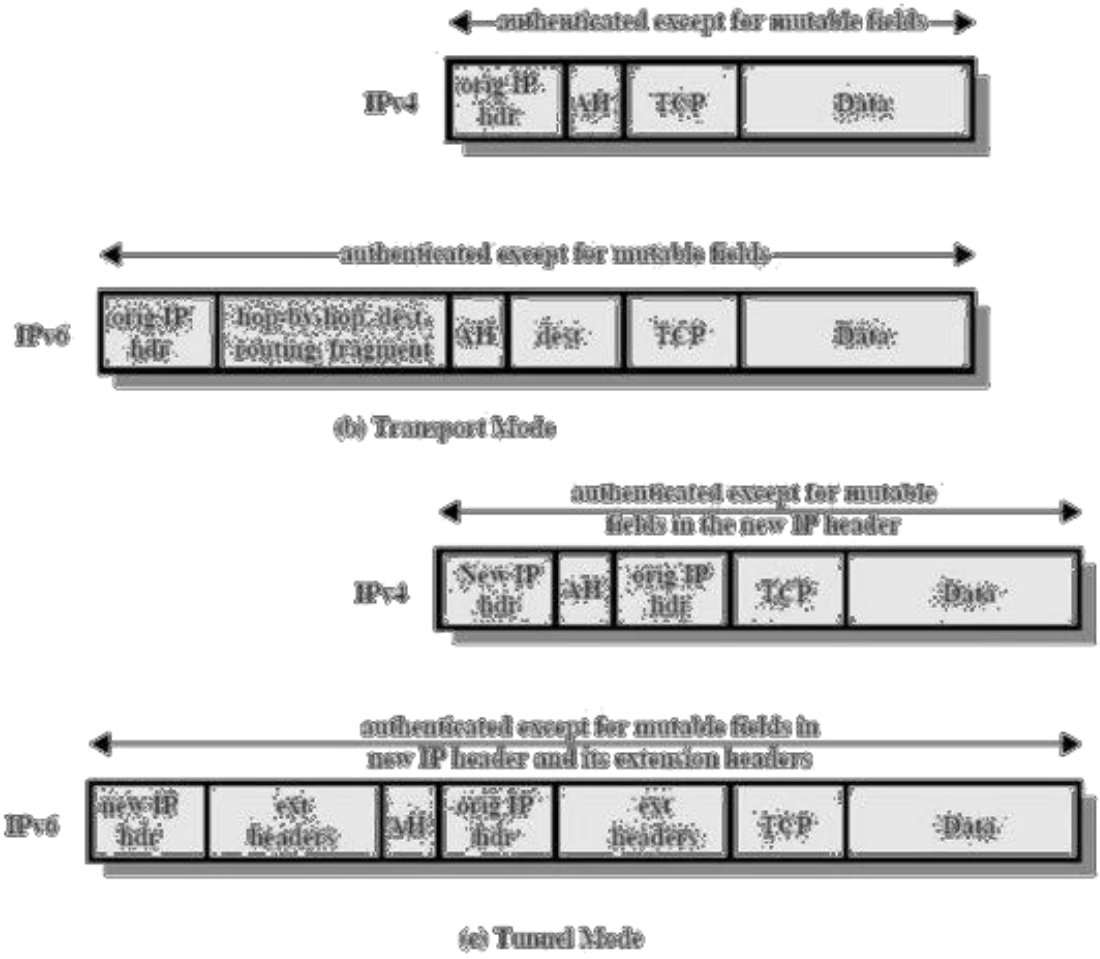
• The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).

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.



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.



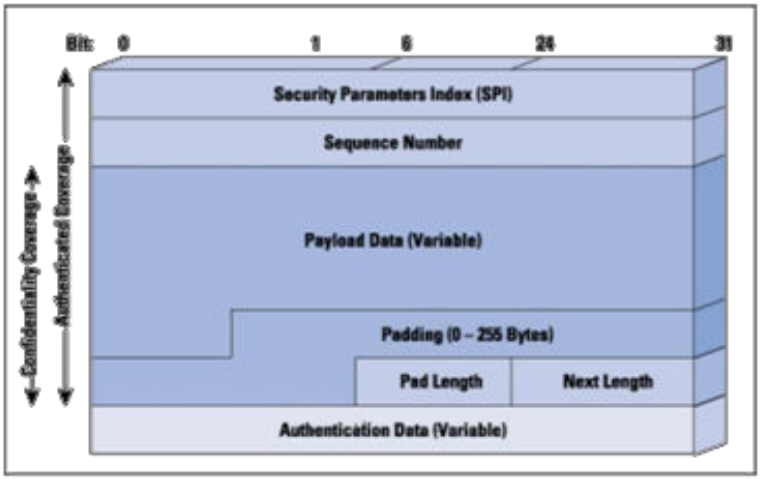
For tunnel mode AH, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header. The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses 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.

Encapsulating Security Payload

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:



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 ed to provide confidentiality.

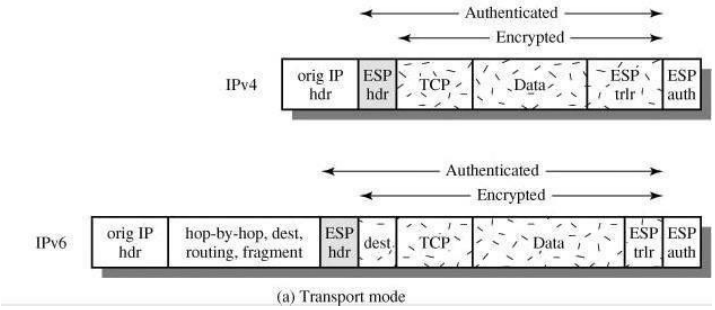
• **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.

• **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that ayload (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.

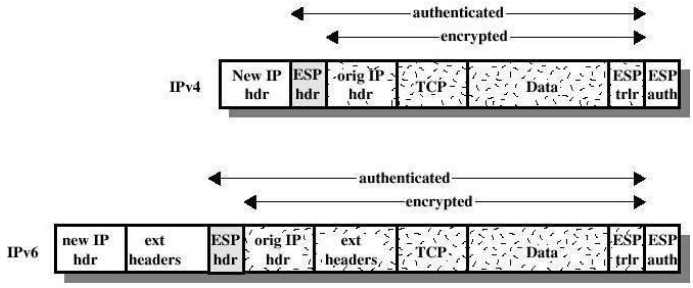
Adding encryption makes ESP a bit more complicated because the encapsulation *surrounds* the payload rather than *precedes* it as with AH: ESP includes header and trailer

Transport Mode ESP



(a) Transport mode

Tunnel Mode ESP

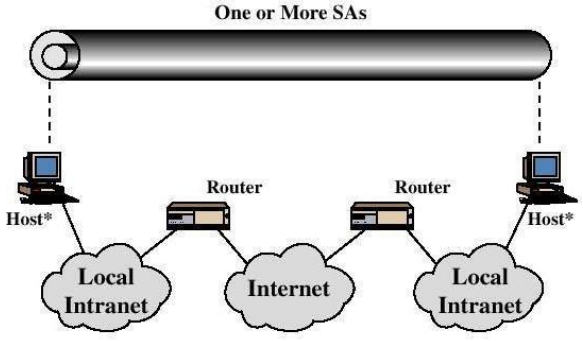


(b) Tunnel Mode

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:

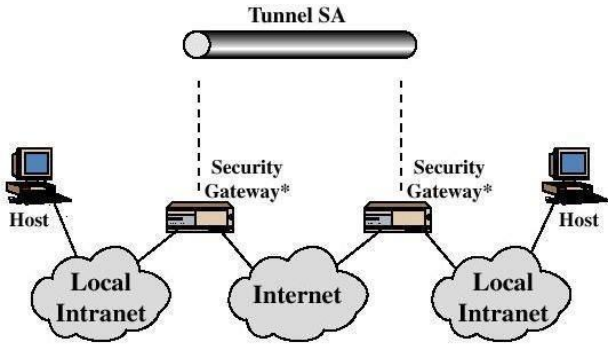
a) AH in transport mode

b) ESP in transport mode

c) ESP followed by AH in transport mode (an ESP SA inside an AH SA)

d) Any one of a, b, or c inside an AH or ESP in tunnel mode

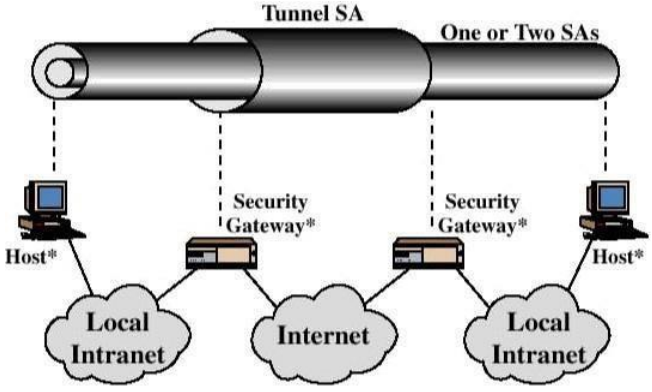
Case:-2



(b) Case 2

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 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.

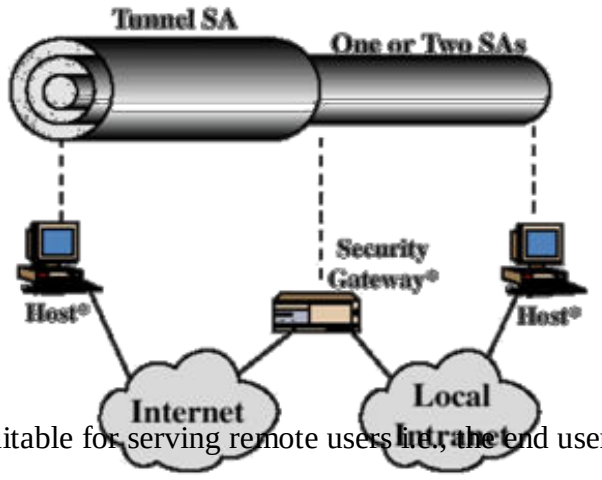
Case-3:-



(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.

KEY MANAGEMENT

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

• **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.

• **Automated:** An automated stem enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolvingconfiguration.

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.

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

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

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

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 intiator, 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

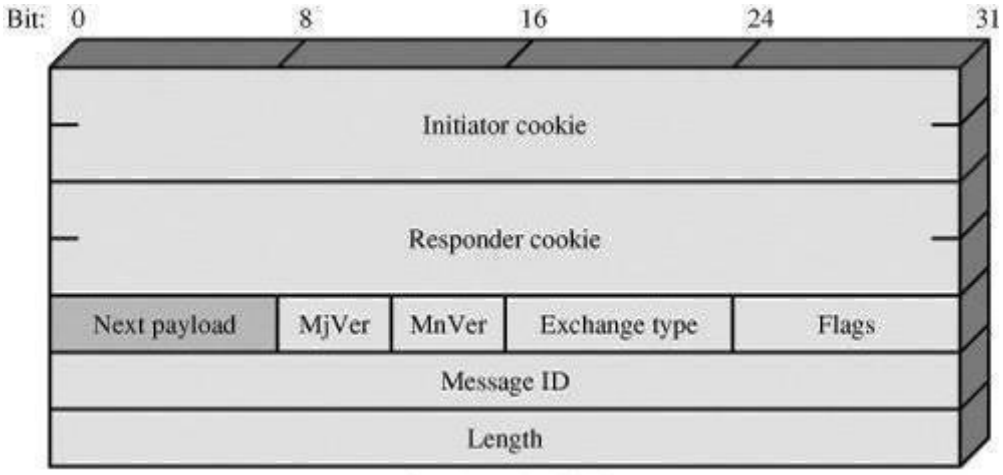
identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

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



(a) ISAKMP header

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:

- Initiator Cookie (64 bits): Cookie of entity that initiated SA establishment, SA notification, or SA deletion.
- Responder Cookie (64 bits): Cookie of responding entity; null in first message from initiator.
- Next Payload (8 bits): Indicates the type of the first payload in the message
- Major Version (4 bits): Indicates major version of ISAKMP in use.
- Minor Version (4 bits): Indicates minor version in use.

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

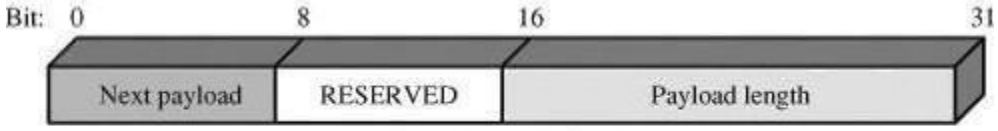
• Flags (8 bits): Indicates specific options set for this ISAKMP exchange. Two bits so far defined: The Encryption bit is set if all payloads following the header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to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

• Message ID (32 bits): Unique ID for this message.

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

ISAKMP Payload Types

All ISAKMP payloads begin with the same generic payload he der 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 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 tate. This payload is used to verify the integrity of the data in a message and may be for no repudiation services.

i. The Nonce payload contains random data used to guarantee likeness 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.

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** ⟶ **R**: SA; NONCE	Begin ISAKMP-SA negotiation
(2) **R** ⟶ **E**: SA; NONCE	Basic SA agreed upon
(3) **I** ⟶ **R**: KE; ID_I AUTH	Key generated; Initiator identity verified by responder
(4) **R** ⟶ **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** ⟶ **R**: SA	Begin ISAKMP-SA negotiation
(2) **R** ⟶ **E**: SA	Basic SA agreed upon
(3) **I** ⟶ **R**: KE; NONCE	Key generated
(4) **R** ⟶ **E**: KE; NONCE	Key generated
(5)* **I** ⟶ **R**: ID_I; AUTH	Initiator identity verified by responder
(6)* **R** ⟶ **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

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 nd 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

UNIT-5

Web Security: *Web Security Considerations, Secure Socket Layer (SSL) and Transport Layer Security (TLS), Secure Electronic Transaction (SET).* ***Intruders, Viruses and Firewalls:*** *Intruders, Intrusion Detection, Password Management, Virus and related threats, Countermeasures, Firewall Design Principles, Types of Firewalls.*

Case Studies on Cryptography and Security*: Secure Inter Branch Transactions, Cross Site Vulnerability, Virtual Elections.*

Introduction:

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 non repudiation. 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.

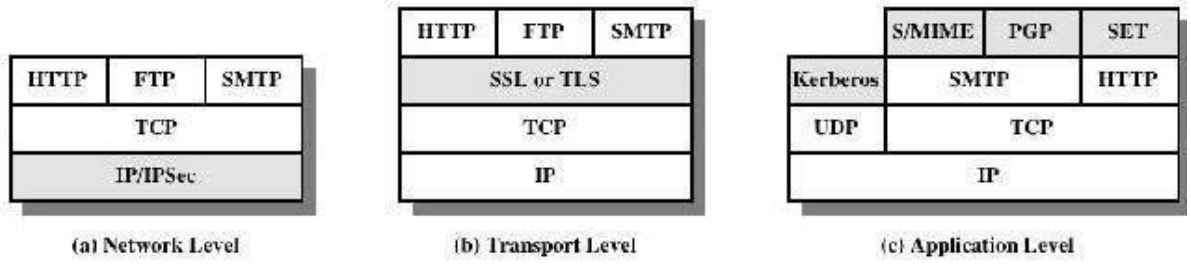
A summary of types of security threats faced in using the Web is given below:

	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

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 tr nsit 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

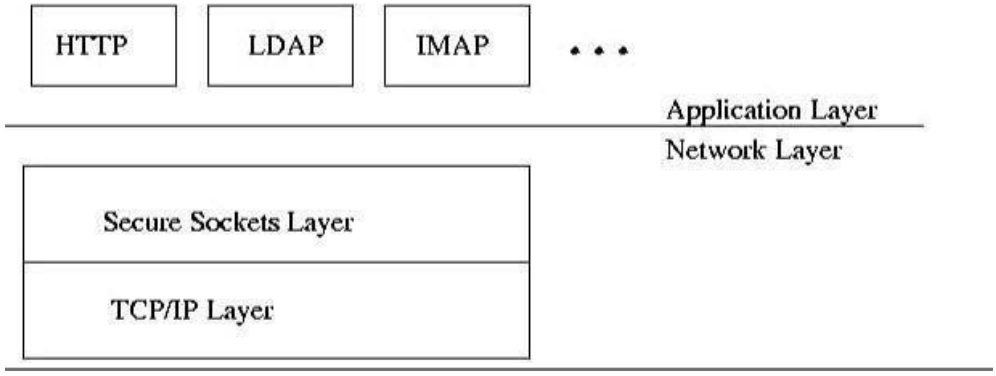
IPSec provides security at the network level and the main advantage is that it is transparent to end users and applications. In addition, IPSec includes a filtering capability so that only selected traffic can be processed. **Secure Socket Layer or Transport Layer Security (SSL/TLS)** provides security 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.

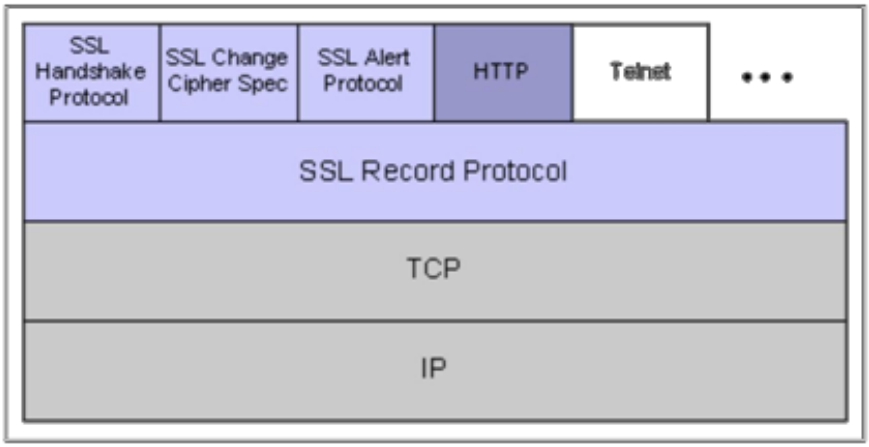
Secure Socket Layer/Transport Layer Security

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:

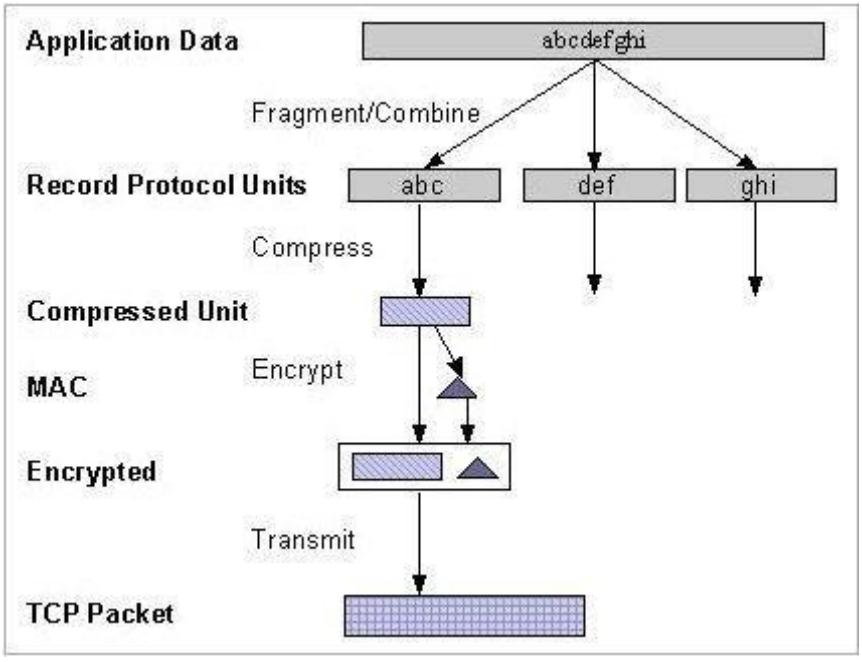
- ***Server and client random***: Byte sequences that are chosen by the server and client for each connection.
- ***Server write MAC secret***: The secret key used in MAC operations on data sent by the server.
- ***Client write MAC secret***: The secret key used in MAC operations on data sent by the 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 264-1.

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 214 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 ||

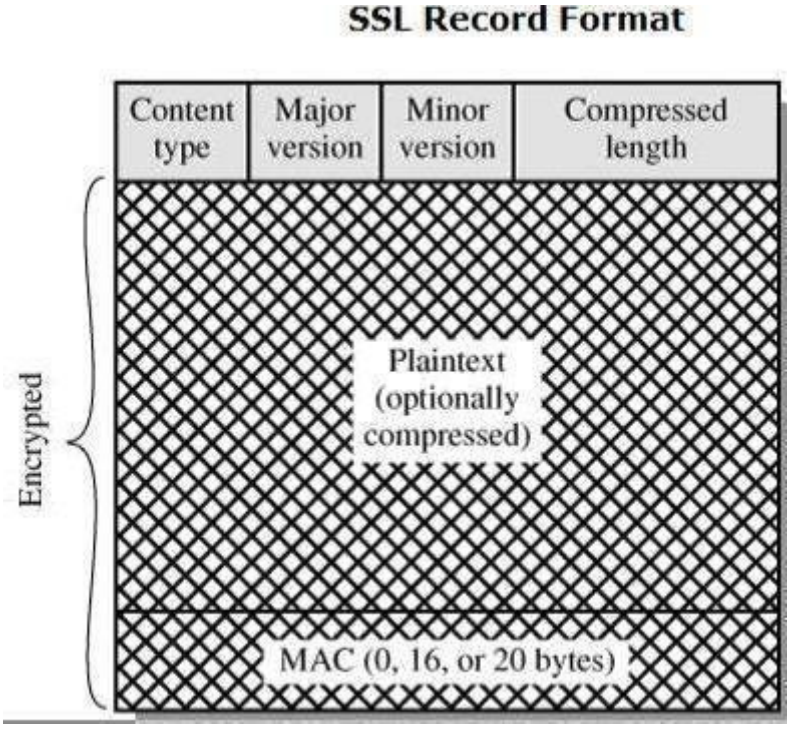
hash(MAC_write_secret || pad_1 || seq_num ||

SSLCompressed.type ||

SSLCompressed.length || SSLCompressed.fragment)) Where,

MAC_write_secret =	=	the byte 0x36 (0011 0110) repeated 48 times
Secret shared key pad_1		(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 m y 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.



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.

The sole purpose of this message is to cause the pending st te to be copied into the current state, which updates the cipher suite to be used on th s connection.

SSL Alert Protocol

The Alert Protocol is used to convey SSL-r lat 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 me age in this protocol consists of two bytes.

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, b t 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

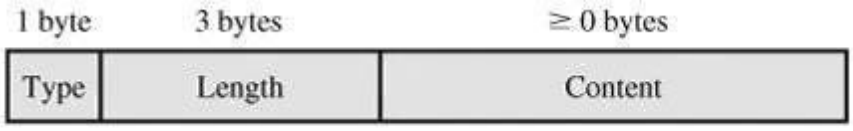
SSL Handshake protocol ensures establishment of reliable nd secure session between client and server and also allows server & client to:

• authenticate each other

• to negotiate encryption & MAC algorithms

• to negotiate cryptographic keys to be used

The Handshake Protocol consists of a eries 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.

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:

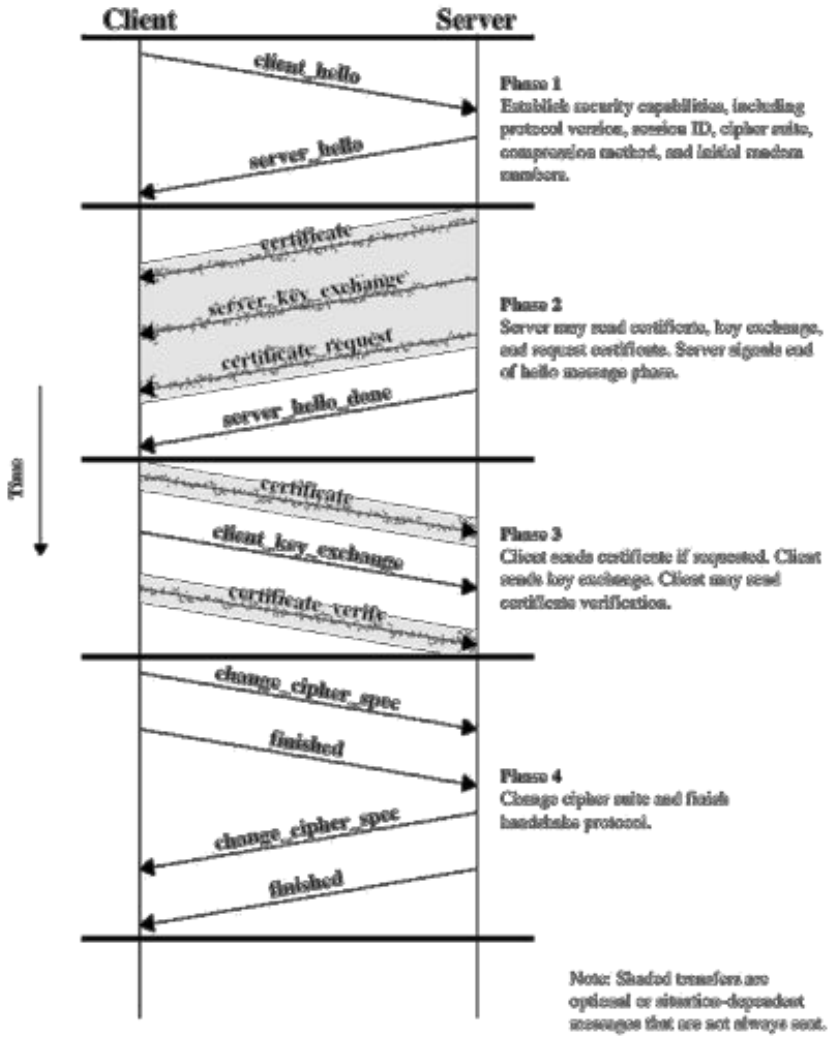
• Version: The highest SSL version understood by the client.

• 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 replayattacks.

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 cl ent wishes to establish a new connection on a new session.

• 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.

• Compression Method: This is a list of the compression methods the client supports.



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.

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.

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.

Transport Layer Security

TLS was released in response to the Int rnet 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, A(2) || seed) ||
HMAC_hash(secret, 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.

Set (Secure Electronic Transaction)

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 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 nd 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

- Cardholder: purchasers interact with merchants from personal computers over the Internet
- Merchant: a person or organization that has goods or services to sell to thecardholder
- Issuer: a financial institution, such as a bank, that provides the cardholder with the payment card.
- Acquirer: a financial institution that establishes an account with a merchant and processes payment card authorizations and payments
- Payment gateway: a function operated by the acquirer or a designated third party that processes merchant payment messages
- 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

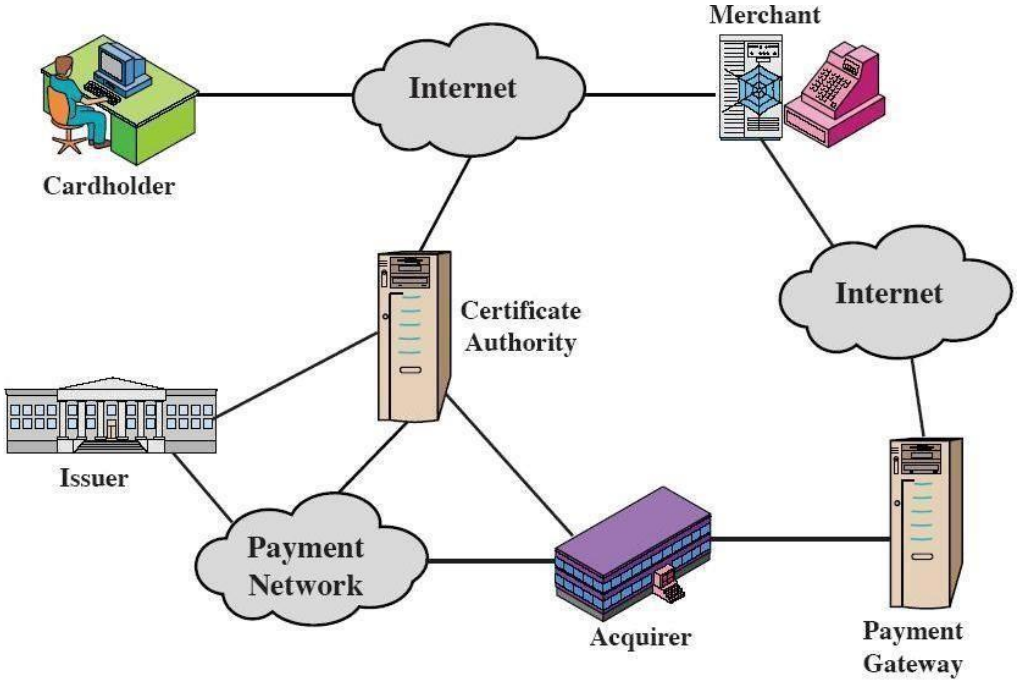
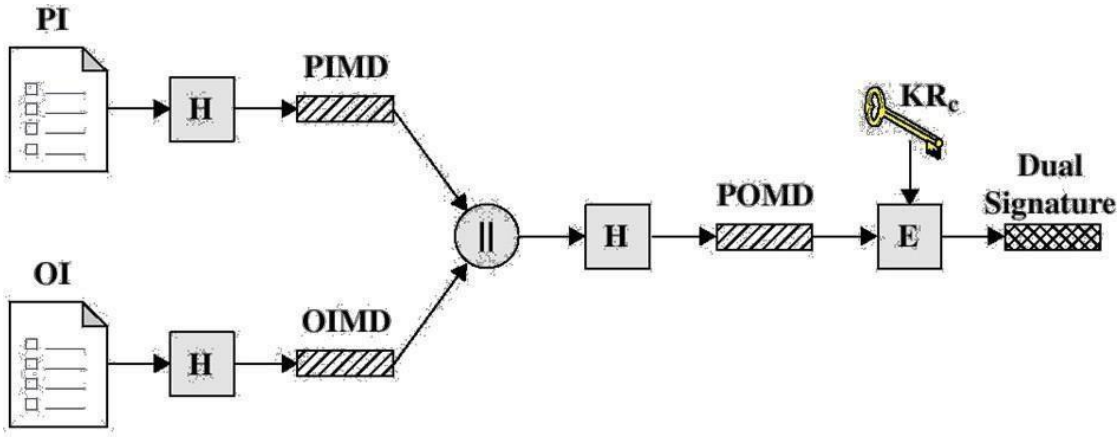


Figure 17.8 Secure Electronic Commerce Components

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
OI = Order Information
H = Hash function (SHA-1)
|| = Concatenation

PIMD = PI message digest
OIMD = OI message digest
POMD = Payment Order message digest
E = Encryption (RSA)
KR_c = 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 ha h of the result is taken. Finally, the customer encrypts the final hash with his or her rivate signature key, creating the dual signature. The operation can be summarized as

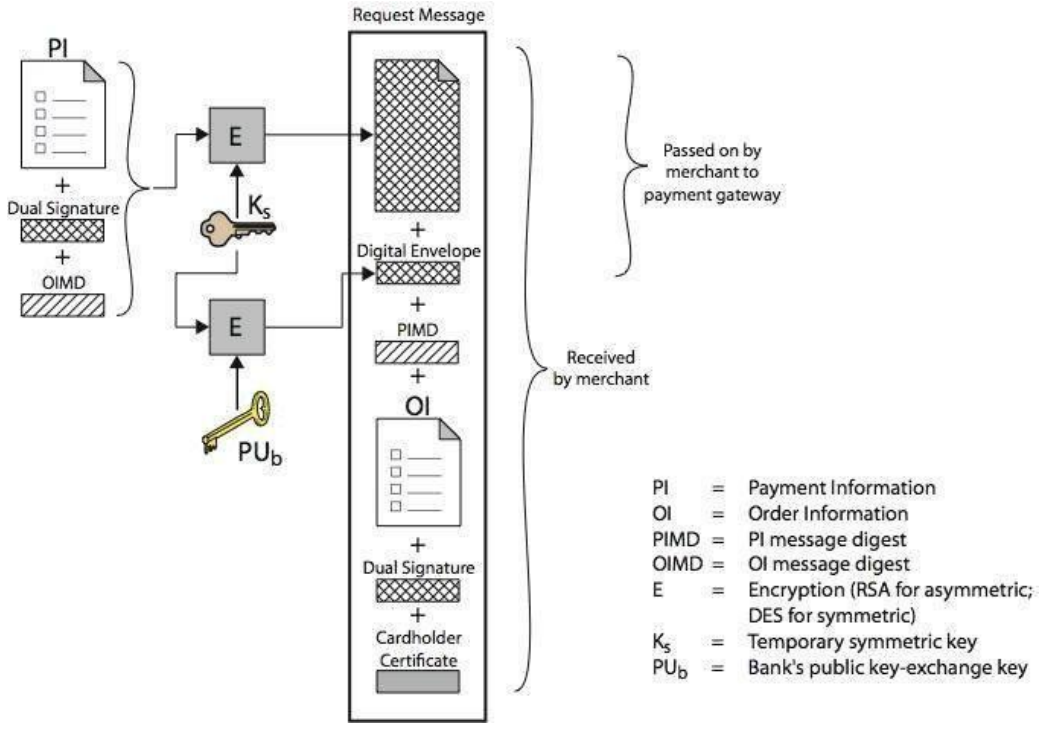
$$DS = E_{KR_c}[H(H(PI) \,||\, \mathrm{H(OI)})]$$

where KR_c 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 D_{KU_c}(DS) where KU_c 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 D_{KU_c}(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.

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.

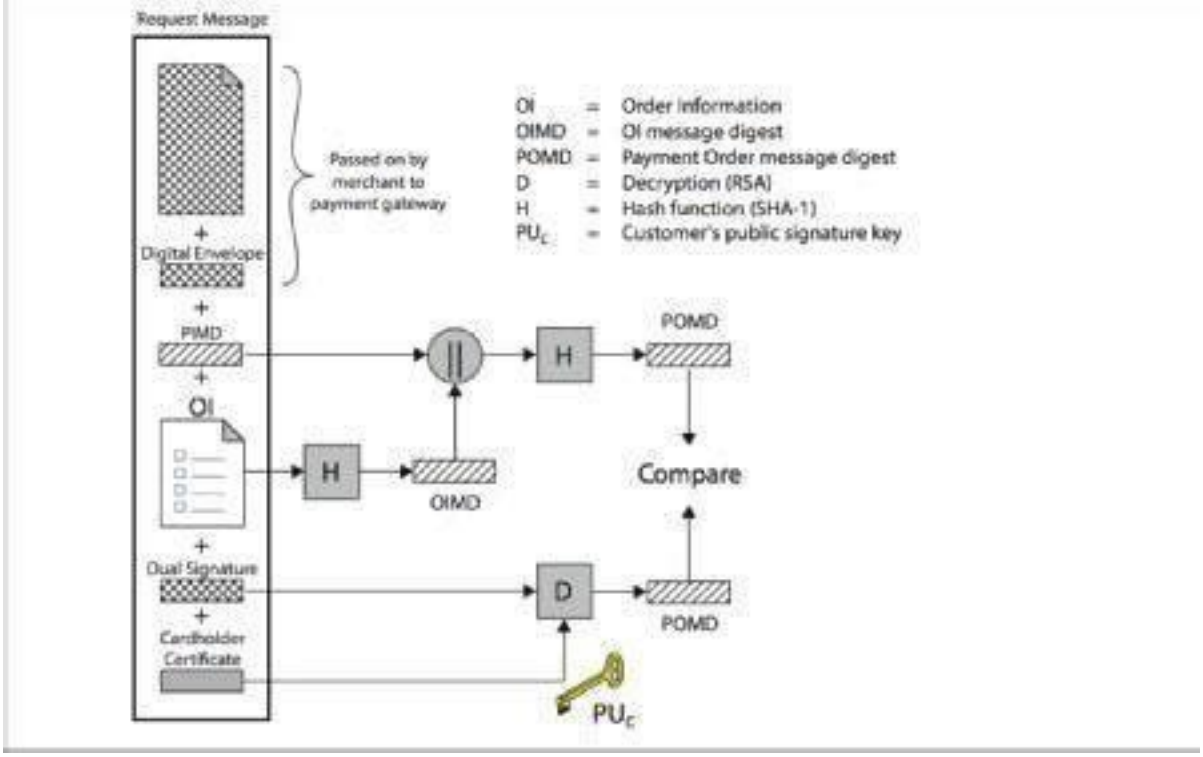
Merchant receives the Purchase Request message, the following actions are done:

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 gatewayfor 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.

Intruders

One of the most publicized attacks to security is the intruder, generally referred to as hacker or cracker. Three classes of intruders are as follows

- **Masquerader** – an individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account.
- **Misfeasor** – a legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuse his or her privileges.
- **Clandestine user** – an individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider. Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system. Benign intruders might be tolerable, lthough they do consume resources and may slow performance for legitimate users. However there is no way in advance to know whether an intruder will be benign or malign.

Intrusion techniques The objective of the intruders is to gain access to a system or to increase the range of privileges acce ible on a system. Generally, this requires the intruders to acquire information that should be protected. In most cases, the information is in the form of a user password. Ty ically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it. The password files can be protected in one of the two ways:

- **One way encryption** – the system stores only an encrypted form of user's password. In practice, the system usually performs a one way transformation (not reversible) in which the password is used to generate a key for the encryption function and in which a fixed length output is produced.
- **Access control** – access to the password file is limited to one or a very few accounts.

The following techniques are used for learning passwords.

- Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
- Exhaustively try all short passwords.
- Try words in the system's online dictionary or a list of likelypasswords.
- Collect information about users such as their full names, the name of their spouse and children, pictures in their office and books in their office that are related to hobbies.
- Try user's phone number, social security numbers and room numbers.
- Try all legitimate license plate numbers.
- Use a torjan horse to bypass restriction on access.
- Tap the line between a remote user and the host system.
- Detection – concerned with learning of an attack, either before or after its success.
- Prevention – challenging security goal and an uphill bottle at all times.

Intrusion Detection

Inevitably, the best intr sion revention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised.
2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

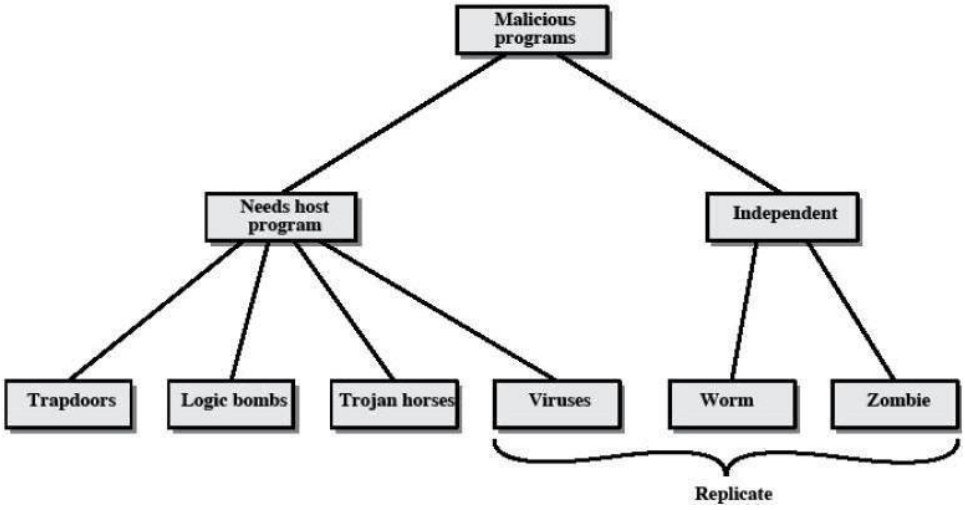
Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified.

Figure 18.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

VIRUSES AND RELATED THREATS

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems.

Malicious Programs



Name	Description
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs
Trojan horse	Program that contains unexpected additional functionality

Backdoor (trapdoor)	Program modification that allows unauthorized access to functionality
Exploits	Code specific to a single vulnerability or set of vulnerabilities
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access
Zombie	Program activated on n infected machine that is activated to l unch attacks on other machines

Malicious software can be divided into two categories: those that need a host program, and those that are independent.

The former are essentially fragments of rograms that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples.

The Nature of Viruses A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs. A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs. During its lifetime, a typical virus goes through the following four phases:

- **Dormant phase**: The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.
- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.
- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Virus Structure

A virus can be prepended or postpended to an execut ble program, or it can be embedded in some other fashion. The key to its op ration is that the infected program, when invoked, will first execute the virus code and th n execute the original code of the program.

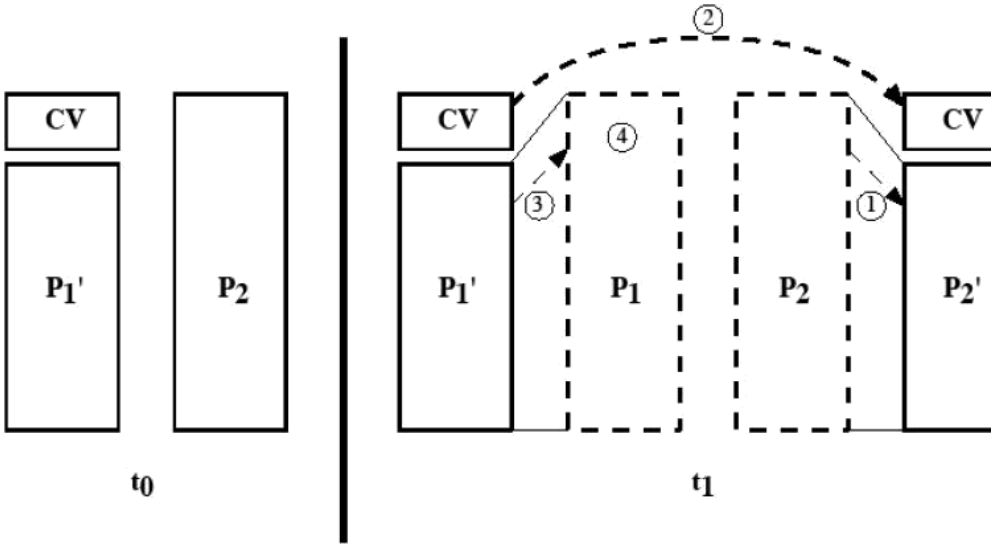
An infected program begins with the virus code and works as follows.

The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length.. The key lines in this virus are numbered, and

Figure 19.3 [COHE94] illustrates the operation. We assume that program P1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

- For each uninfected file P_2 that is found, the virus first compresses that file to produce P'_2, which is shorter than the original program by the size of the virus.
- A copy of the virus is prepended to the compressed program.
- The compressed version of the original infected program, P'1, is uncompressed
- The uncompressed original program is executed.



In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.

Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

Types of Viruses

Following categories as being among the most significant types of viruses:

- **Parasitic virus**: The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.
- **Memory-resident virus**: Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.
- **Boot sector virus**: Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.
- **Stealth virus**: A form of virus explicitly designed to hide itself from detection by antivirus software.
- **Polymorphic virus**: A virus that mutates with every infection, making detection by the "signature" of the virus impossible.
- **Metamorphic virus**: As with a polymorphicMediavirus,metamorphic virus mutates with every infection. The difference is that metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as th ir appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.

2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.

3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input. Successive releases of Word provide increased protection against macro viruses. For example,files and alerts the customer to the potential risk of opening file with macros.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melis a, made use of Microsoft Word macro embedded in an attachment. If the recipient o ens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mailpackage.
2. The virus does local damage.

Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions. To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- Electronic mail facility: A worm mails a copy of itself to other systems.Remote execution capability: A worm executes a copy of itself on another system.
- Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion. A

network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase.

The Morris Worm

The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation.

1. It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords

and corresponding user IDs. The assumption Media was that many users would use the same password on different systems. To obtain th passwords, the worm ran a password-cracking program that tried

a. Each user's account name and simple permutations of it
b. A list of 432 built-in passwords that Morris thought to be likely candidates
c. All the words in the local ystem directory

2. It exploited a bug in the finger rotocol, which reports the whereabouts of a remote user.
3. It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter.

Recent Worm Attacks In late 2001, a more versatile worm appeared, known as Nimda. Nimda spreads by multiple mechanisms:

- from client to client via e-mail
- from client to client via open network shares
- from Web server to client via browsing of compromised Web sites
- from client to Web server via active scanning for and exploitation of various Microsoft

Firewalls

A firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter, forming a single choke point where security and audit can be imposed. A firewall:

1. Defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks.

2. provides a location for monitoring security-related events

3. is a convenient platform for several Internet functions that are not security related, such as NAT and Internet usage audits or logs

4. A firewall can serve as the platform for IPSec to implement virtual private networks.

Design Goals of Firewalls

All traffic from inside to outside must pass through the f rew ll (physically blocking all access to the local network except via the firewall)

Only authorized traffic (defined by the local security police) will be allowed to pass

The firewall itself is immune to penetration (use of trusted system with a secure operating system)

The four general techniques that firewalls use to control access and enforce the sites security policies are:

- **Service control**: Determines the types of Internet services that can be accessed, inbound or outbound
- **Direction control**: Determines the direction in which particular service requests are allowed to flow
- **User control**: Controls access to a service according to which user is attemptingto access it
- **Behavior control**: Controls how particular services are used (e.g. filter e-mail)

The limitations of Firewalls are:

1. Cannot protect against attacks that bypass the firewall, eg PCs with dial-out capability to an ISP, or dial-in modem pool use.

2. do not protect against internal threats, eg disgruntled employee or one who cooperates with an attacker

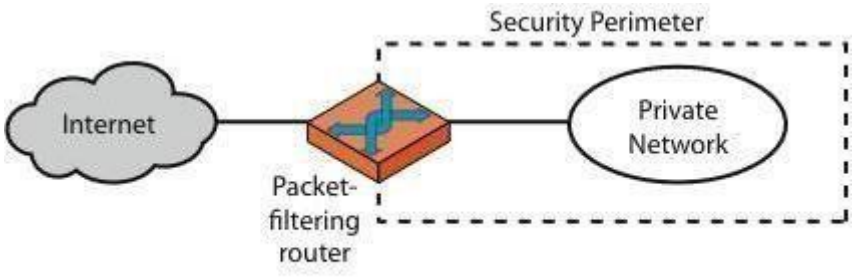
3. cannot protect against the transfer of virus-infected programs or files, given wide variety of O/S & applications supported

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.

The table gives some examples of packet-filtering rule sets. In each set, the rules are applied top to bottom.

Table 20.1 Packet-Filtering Examples

A

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

B

action	ourhost	port	theirhost	port	comment
block	*	*	*	*	default

C

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connection to their SMTP port

D

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		our packets to their SMTP port
allow	*	25	*	*	ACK	their replies

E

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

A. Inbound mail is allowed to a gateway host only (port 25 s for SMTP incoming

B. explicit statement of the default policy

C. tries to specify that any inside host can send mail to the outside, but has problem that an outside machine could be configured to have some other application linked to port 25 **D**. properly implements mail sending rule, by checking ACK flag of a TCP segment is set **E**. this rule set is one approach to handling FTP connections

Some of the attacks that can be made on packet-filtering routers & countermeasures are*:*

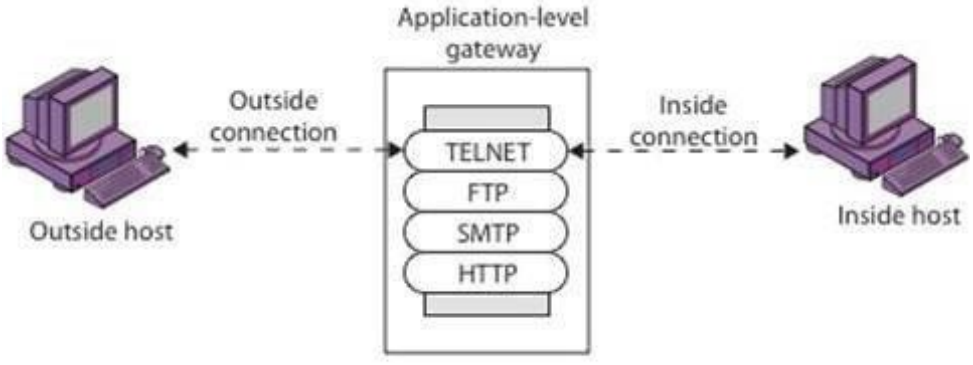
- **IP address spoofing**: where intruder transmits packets from the outside with internal host source IP addresses, need to filter & discard such packets
- **Source routing attacks**: where source specifies the route that a packet should take to bypass security measures, should discard all source routed packets
- **Tiny fragment attacks**: intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into separate fragments to circumvent filtering rules needing full header info, can enforce minimum fragment size to include full header.

Stateful Packet Filters

A traditional packet filter makes filtering decisions on an individual packet basis and does not take into consideration any higher layer context. A stateful inspection packet filter tightens up the rules for TCP traffic by creating a directory of outbound TCP connections, and will allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory. Hence they are better able to detect bogus packets sent out of context.

APPLICATION LEVEL GATEWAY

An application-level gateway (or proxy server), acts as a relay of application-level traffic. The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported nd cannot be forwarded across the firewall.



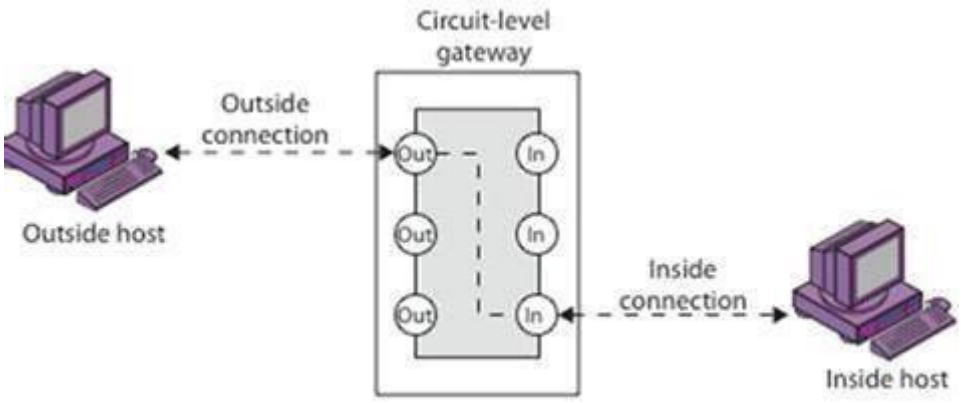
(b) Application-level gateway

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In addition, it is easy to log and audit all incoming traffic at the application level. A prime disadvantage of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

CIRCUIT LEVEL GATEWAY

A circuit-level gateway relays two TCP connections, one between itself and an inside TCP user, and the other between itself and a TCP user on an outside host. Once the two connections are established, it relays TCP data from one connection to the other without examining its contents. The security function consists of determining which connections will be allowed. It is typically used when internal users are trusted to decide what external services to access.

One of the most common circuit-level gateways is SOCKS, defined in RFC 1928. It consists of a SOCKS server on the firewall, and a SOCKS library & SOCKS-aware applications on internal clients. The protocol described here is designed to provide a framework for client-server applications in both the TCP and UDP domains to conveniently and securely use the services of a network firewall. The protocol is conceptually a "shim-layer" between the application layer and the transport layer, and as such does not provide network-layer gateway services, such as forwarding of ICMP messages.



(c) Circuit-level gateway

Bastion Host

A bastion host is a critical strong point in the network's security, serving as a platform for an application-level or circuit-level gateway, or for external services. It is thus potentially exposed to "hostile" elements and must be secured to withstand this. Common characteristics of a bastion host include that it:

- executes a secure version of its O/S, making it a trusted system
- has only essential services installed on the bastion host

- may require additional authentication before a user is allowed access to the proxy services
- is configured to support only a subset of the standard application's command set, with access only to specific hosts
- maintains detailed audit information by logging all traffic
- has each proxy module a very small software package specifically designed for network security
- has each proxy independent of other proxies on the bastion host
- have a proxy performs no disk access other than to read its initial configuration file
- have each proxy run as a non-privileged user in a private and secured directory
- A bastion host may have two or more network interfaces (or ports), and must be trusted to enforce trusted separation between these network connections, relaying traffic only according to policy.

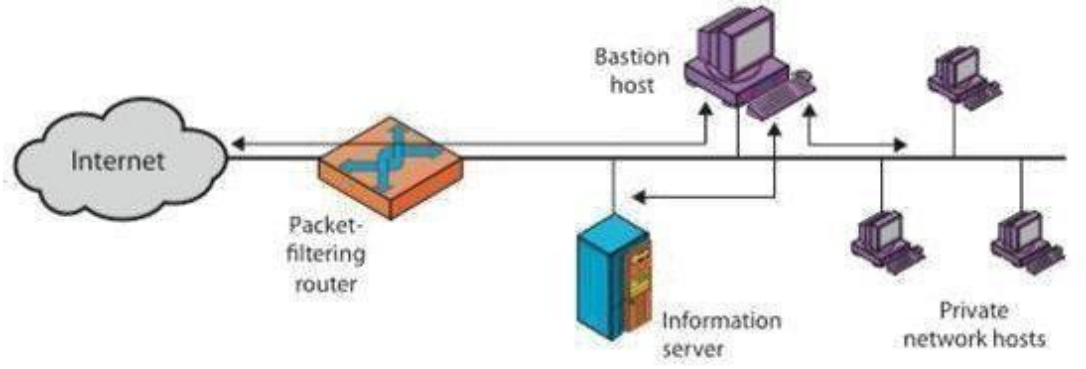
Firewall Configurations

In addition to the use of a simple configuration consisting of a single system, more complex configurations are possible and indeed more common. There are three common firewall configurations.

The following figure shows the "**screened host firewall, single-homed bastion configuration**", where the firewall consists of two systems:

• a packet-filtering router - allows Internet packets to/from bastion only

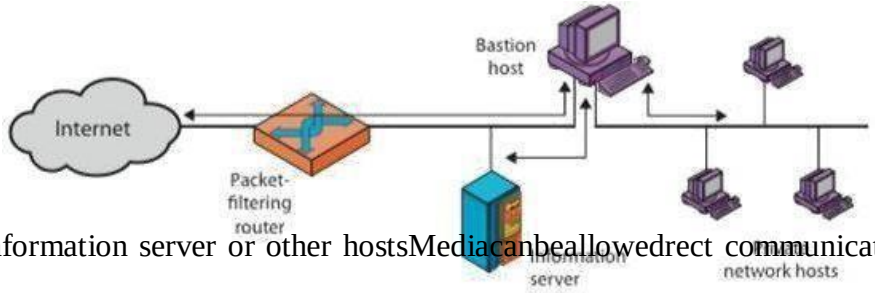
• a bastion host - performs authentication and proxy functions



(a) Screened host firewall system (single-homed bastion host)

This configuration has greater security, as it implements both packet-level & application-level filtering, forces an intruder to generally penetrate two separate systems to compromise internal security, & also affords flexibility in providing direct Internet access to specific internal servers (eg web) if desired.

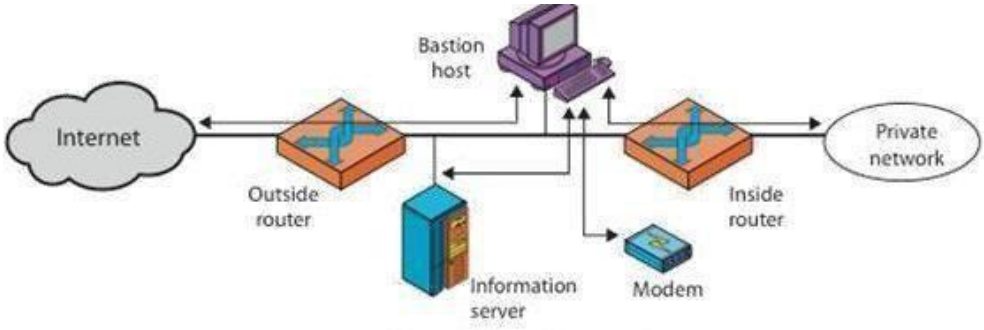
The next configuration illustrates the "**screened host firewall, dual-homed bastion configuration**" which physically separates the external and internal networks, ensuring two systems must be compromised to breach security. The advantages of dual layers of security are also present here.



(b) Screened host firewall system (dual-homed bastion host)

Again, an information server or other hostsMediacanbeallowedrect communication with the router if this is in accord with the security policy, but are now separated from the internal network.

The third configurations illustrated below shows the "**screened subnet firewall configuration**", being the most secure hown.



(c) Screened-subnet firewall system

It has two packet-filtering routers, one between the bastion host and the Internet and the other between the bastion host and the internal network, creating an isolated sub-network. This may consist of simply the bastion host but may also include one or more information servers and modems for dial-in capability. Typically, both the Internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked.

This configuration offers several advantages:

• There are now three levels of defense to thwart intruders

• The outside router advertises only the existence of the screened subnet to the Internet; therefore the internal network is invisible to the Internet

• Similarly, the inside router advertises only the existence of the screened subnet to the internal network; hence systems on the inside network cannot construct direct routes to the Internet

CASE STUDIES ON CRYPTOGRAPY AND SECURITY

Secure Inter-branch Payment Transactions

- **Points for classroom discussions**
  1. What is the technology to achieve non-repudiation? How is this guaranteed?
  2. How is the problem of key distribution resolved in PKI?
  3. Why are cryptographic toolkits required?
  4. How can smart cards be used in cryptography?

General Bank Of India (GBI) has implemented an Electronic Payment System called as *EPS* in about 1200 branches across the country. This system transfers payment instructions between two computerized branches of GBI. A central server is maintained at the EPS office located in Mumbai. The branch offices connect to the Local VSAT of a private network by using dial-up connection. The local VSAT has a connectivity established with the EPS office. GBI utilizes its proprietary messaging service called as *GBI-Transfer* to exchange payment instructions. Currently, EPS has minimal data security. As the system operates in a closed network, the current security infrastructure may suffice the need. The data moving across the network is in encrypted format.

Current EPS Architecture EPS is used to transmit payment details from the payer branch to the payee branch via the central server in Mumbai which is also described stepby- step.

- A data-entry person in the *Payer Branch* enters transaction details through the EPS interface.
- A Bank Officer checks the validity of the transaction through the EPS interface.
- After validating the transaction, the Bank Officer authorizes the transaction. Authorized transaction is stored in a local Payment Master (PM) database.
- Once the transaction is stored in PM, a copy of the same is encrypted and stored in a file. This transaction file is stored in OUT directory.
- The *GBI-Transfer* application looks for any pending transactions (i.e. for the presence of any files in the OUT directory) by a polling mechanism and if it finds such

transactions, it sends all these files one-by-one to the EPS central office located in Mumbai by dialing the local VSAT.

- The local VSAT gets connectivity to the EPS central office and the transaction is transferred and
  stored in the IN directory at the EPS central office.

- The interface program at the EPS central office collects the file pending in the IN directory and sends it to the PM application at that office.

- In order to send the Credit Request to PM, the transaction headers are changed. The transaction with changed headers in encrypted format is then placed in OUT directory of the EPS central office.

- The GBI-Transfer application at the EPS central office collects the transactions pending in the OUT directory and sends them to the *Payee Bank* through the VSAT.

- The transaction is transferred and stored in the IN directory of the *Payee Branch*.

- The interface program at the *Payee Branch* collects the transaction and posts it in PM.

- PM marks the credit entry and returns back an acknowledgement of the same. The acknowledgement is placed in OUT directory of the *Payee Branch*.

- The acknowledgement is picked by GBI-Transfer at the *Payee Branch* and sent to the EPS central office through the VSAT.

- The EPS central office receives the credit acknowledgement and forwards it to *Payer Branch*.

- The *Payer Branch* receives the credit acknowledgement receipt. This completes the transaction.

Requirements to Enhance EPS As GBI is in the process of complete automation and setting up connectivity over the Internet or a private network, they need to ensure stringent security measures, which demand the usage of a Public Key Infrastructure (PKI) framework.

As a part of implementing security, GBI wants the following aspects to be ensured:

• Non-repudiation (Digital Signatures)

• Encryption – 128-bit (Upgrade to the current 56-bit encryption)

• Smart card support for storing sensitive data & on-card digital signing

• Closed loop Public Key Infrastructure :

Proposed Solution Since providing cryptographic functionalities require the usage of a cryptographic toolkit, it is assumed that GBI will implement an appropriate Certification Authority (CA) infrastructure and a PKI infrastructure offering.

The transaction will be digitally signed and encrypted/decrypted at the Payer and Payee branches, as well as at the EPS central office. The signing operation can be performed on the system or on external hardware like a smart card. On the server side, a provision of automated signing without any manual intervention will be provided.

The transaction flow described earlier would now be split into two legs:

- The Payer Leg (*Payer Branch* to the EPS central office)
- The Payee Leg (EPS central office to the *Payee Branch*)

Cross Site Scripting Vulnerability (CSSV)

Points for classroom discussions

- What is the purpose of scripting technologies on the Internet?
- What can prevent CSSV attacks?
- What sort of testing can the creators of a Website perform in order to guard against possible CSSV attacks?

Cross Site Scripting Vulnerability (CSSV) is a relatively new form of attacks that exploits inadequate validations on the server-side. The term *Cross Server Scripting Vulnerability (CSSV)* is actually not completely correct. However, this term was coined when the problem was not completely understood and has stuck ever since. Cross-site scripting happens when malicious tags and/or scripts attack a Web browser via another site's dynamically generated Web pages. The attacker's target is not a Website, but rather its users (i.e. clients or browsers).

The idea of CSSV is quite simple to understand and is based on exploiting the scripting technologies, such as JavaScript, VBScript or JScript. Let us understand how this works. Consider the following Web page containing a form as shown in Fig. 10.9, in which the user is expected to enter her postal address. Suppose that the URL of the site sending this page is www.test.com and when the user submits this form, it would be processed by a server-side program called as address.asp.

We would typically expect the user to enter the house number, street name, city, postal code and country, etc. However, imagine that the user enters the following weird string, instead:

<SCRIPT>Hello World</SCRIPT>

As a result, the URL submitted would be something like www.test.com/address.asp?address= <SCRIPT>Hello World </SCRIPT>.

Now suppose that the server-side program address.asp does not validate the input sent by the user and simply sends the value of the field address to the next Web page. What would this

translate to? It would mean that the next Web page would receive the value of address as <SCRIPT>Hello World</ SCRIPT>.

As we know, this would most likely treat the value of the address field as a script, which would be executed as if it is written in a scripting language, such as JavaScript etc on the Web browser. Therefore, the user would get to see *Hello World*.

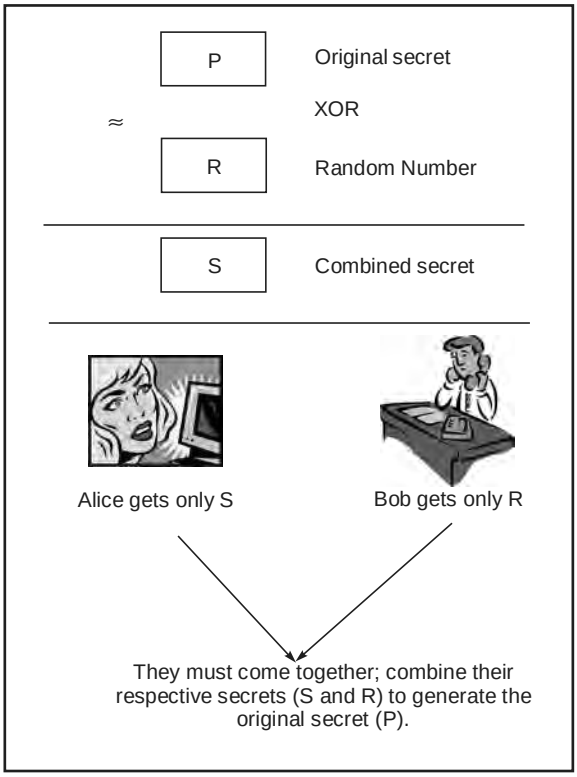
Virtual Elections

Points for classroom discussions

- Is it technically possible to have elections on the Internet? How? What sort of infrastructure would be needed for this?
- What would be the main concerns in such a virtual election?
- What would be the use of digital signatures and encryption in virtual elections?

Another situation where cryptography is useful is virtual elections. Computerized voting would become quite common in the next few decades. As such, it is important that the protocol for irtual

Fig. *Secret splitting*



elections should protect individual privacy and should also disallow cheating. Consider the following protocol in order that voters can send their votes electronically to the Election authority (EA).

- Each voter casts the vote and encrypts it with the public key of the EA.
- Each voter sends the encrypted vote to the EA.
- The EA decrypts all the votes to retrieve the original vote, tabulates all the votes and announces the result of the election.

Is this protocol secure and does it provide comfort both to the voters as well as to the EA? Not at all!

There are following problems in this scheme:

- The EA does not know whether the authorized voters have voted or it has received fake (bogus) votes.
- Secondly, there is no mechanism to prevent duplicate voting.

What is the advantage of this protocol? Clearly, no one would be able to change another voter's vote, because it is first encrypted with the EA's public key and is then sent to the EA. However, if we observe this scheme carefully, an attacker need not change someone's vote at all. The attacker can simply send duplicate votes!

How can we improve upon this protocol to make it more robust? Let us rewrite it, as follows:

- Each voter casts the vote and signs it with her private key.
- Each voter then encrypts the signed vote with the public key of the EA.
- Each voter sends the vote to the EA.
- The EA decrypts the voter with its private key and verifies the signature of the voter with the help of the voter's public key.
- The EA then tabulates all the votes and announces the result of the election.

This protocol would now ensure that duplicate voting is disallowed. Because the voter has signed the vote (with her private key) in Step 1, this can be checked. Similarly, no one can change another voter's vote. This is because a vote is digitally signed and any changes to it will be detected and exposed in the signature verification process.

Although this protocol is a lot better, the trouble with this scheme is that the EA would come to know who voted for whom, leading to privacy concerns. We shall leave it to the reader to figure out how this problem can be solved.