ANNAMALAI UNIVERSITY
FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

MASTER OF COMPUTER APPLICATION (MCA)

MCAE 405- NETWORK SECURITY

Ms. V. Narmatha, M.Sc., M.Phil.,
Assistant professor/Programmer
Department of computer and information science
Faculty of science
Annamalai university

MCAE405- NETWORK SECURITY

AIM: To study the various issues concerning Network security, Database security and Program security

Unit-I

Symmetric Ciphers: Classical Encryption Techniques- Block Ciphers and the Data Encryption Standard – Finite Fields – Advanced Encryption Standard – Symmetric Ciphers – Confidentiality using Symmetric Encryption.

Unit-II

Public Key Encryption and Hash Functions: Introduction to Number Theory – Public Key Cryptography and RSA - Key Management- other Public Key Cryptosystem – Message Authentication and Hash Functions – Hash and MAC Algorithms – Digital Signatures and Authentication Protocols.

Unit-III

Program Security: Secure Programs – Non-Malicious Program Errors – Viruses and Others Malicious Code – Targeted Malicious Code – Control Against Threats.

Unit-IV

Database Security: Introduction to Database – Security Requirement – Reliability and Integrity – Sensitive Data – Inference – Multilevel Databases - Multilevel Security

Unit-V

Network Security: networks Concepts – Threats in Networks – Network Security Controls – Firewalls – l. Electronic Mail Security – IP Security – Web Security.

Text Books:

1. Charles B. Pfleeger - Shari Lawrence Pfleeger , " Security in Computing ", Third Edition, Pearson Education, 2003.
2. William Stallings, "Cryptography and Network Security – Principles and Practices ", Pearson Education, Fourth Edition, 2003.

Unit-I

Symmetric Ciphers: Classical Encryption Techniques- Block Ciphers and the Data Encryption Standard – Finite Fields – Advanced Encryption Standard – Symmetric Ciphers – Confidentiality using Symmetric Encryption.

INTRODUCTION

Network Security is used to protect both equipment and information.

Computer data often travels from one computer to another, leaving the safety of its protected physical surroundings. Once the data is out of hand, people with bad intention could modify or forge your data, either for amusement or for their own benefit.

Cryptography can reformat and transform our data, making it safer on its trip between computers. The technology is based on the essentials of secret codes, augmented by modern mathematics that protects our data in powerful ways.

• **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.

BASIC CONCEPTS

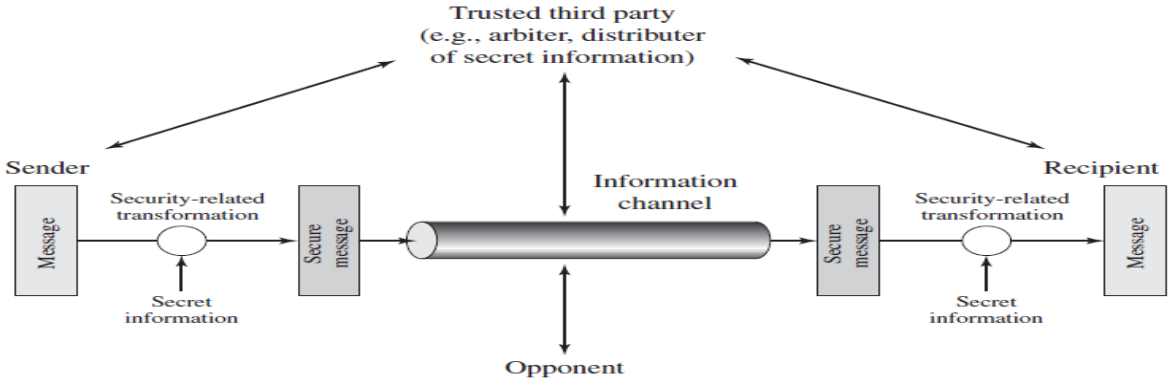
- **Cryptography :**The art or science encompassing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form.
- **Plaintext:** The original intelligible message.
- **Cipher text:** The transformed message.
- **Cipher text:** An algorithm for transforming an intelligible message into one that is unintelligible by transposition and/or substitution methods.
- **Key:** Some critical information used by the cipher, known only to the sender& receiver
- **Encipher** (encode) : The process of converting plaintext to cipher text using a cipher and a key
- **Decipher** (decode) : The process of converting cipher text back into plaintext using a cipher and a key

- **Cryptanalysis:** The study of principles and methods of transforming an unintelligible message back into an intelligible message *without* knowledge of the key. Also called **code breaking**
- **Cryptology:** Both cryptography and cryptanalysis
- **Code** An algorithm for transforming an intelligible message into an unintelligible one using a code-book.

CLASSICAL ENCRYPTION TECHNIQUES

- Symmetric Cipher Model
- Transposition Techniques
- Substitution Techniques
- Steganography

SYMMETRIC CIPHER MODEL



Symmetric cipher model consists of five components

1. Plain text: An Original message is known as plaintext. That is feed into the algorithm as input.
2. Encryption Algorithm: The encryption algorithm perform various transposition and substitution techniques.
3. Secret Key: The secret key is also input to the encryption algorithm. The value is independent of the plain text and encryption algorithm.
4. Cipher text: This is the scrambled message produced as output. It depends on the plaintext and secret key.
5. Decryption Algorithm: The encryption algorithm run in the reverse. It take the cipher text and the secret key and produce the original message.

The symmetric encryption is also known as Conventional encryption or single key encryption.

CRYPTOGRAPHY

Cryptographic systems are generally classified along three independent dimensions:

- **Type of operations used for transforming plain text to cipher text**

  All the encryption algorithms are 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 said 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 **stream cipher** processes the input elements continuously, producing output element one at a time, as it goes along.

Types of Cryptosystems

Fundamentally, there are two types of cryptosystems based on the manner in which encryption-decryption is carried out in the system −

- Symmetric Key Encryption
- Asymmetric Key Encryption

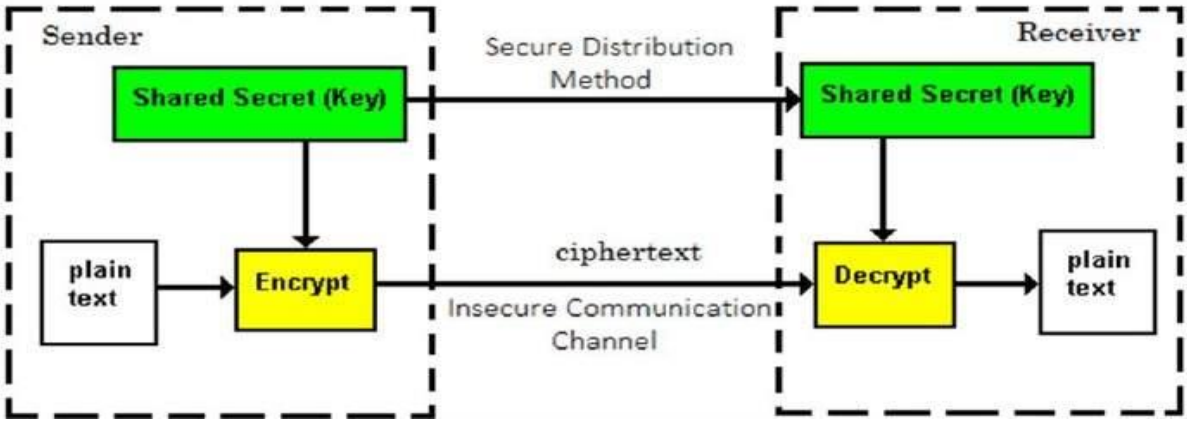
The main difference between these cryptosystems is the relationship between the encryption and the decryption key. Logically, in any cryptosystem, both the keys are closely associated. It is practically impossible to decrypt the ciphertext with the key that is unrelated to the encryption key.

Symmetric Key Encryption

The encryption process where **same keys are used for encrypting and decrypting** the information is known as Symmetric Key Encryption.

The study of symmetric cryptosystems is referred to as **symmetric cryptography**. Symmetric cryptosystems are also sometimes referred to as **secret key cryptosystems**.

A few well-known examples of symmetric key encryption methods are − Digital Encryption Standard (DES), Triple-DES (3DES), IDEA, and BLOWFISH.



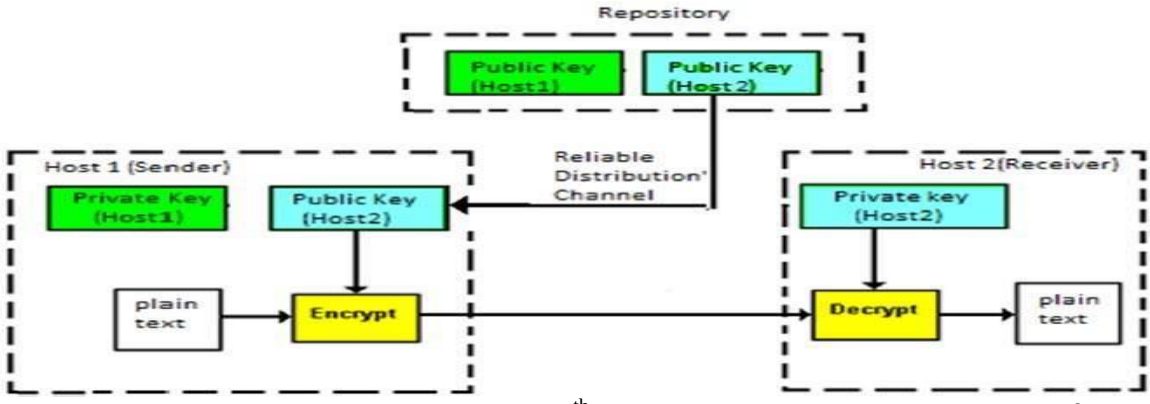
Prior to 1970, all cryptosystems employed symmetric key encryption. Even today, its relevance is very high and it is being used extensively in many cryptosystems. It is very unlikely that this encryption will fade away, as it has certain advantages over asymmetric key encryption.

The salient features of cryptosystem based on symmetric key encryption are −

- Persons using symmetric key encryption must share a common key prior to exchange of information.
- Keys are recommended to be changed regularly to prevent any attack on the system.
- A robust mechanism needs to exist to exchange the key between the communicating parties. As keys are required to be changed regularly, this mechanism becomes expensive and cumbersome.
- In a group of **n** people, to enable two-party communication between any two persons, the number of keys required for group is **n × (n – 1)/2**.
- Length of Key (number of bits) in this encryption is smaller and hence, process of encryption-decryption is faster than asymmetric key encryption.
- Processing power of computer system required to run symmetric algorithm is less.

Asymmetric Key Encryption

The encryption process where **different keys are used for encrypting and decrypting the information** is known as Asymmetric Key Encryption. Though the keys are different, they are mathematically related and hence, retrieving the plaintext by decrypting ciphertext is feasible. The process is depicted in the following illustration −



Asymmetric Key Encryption was invented in the 20th century to come over the necessity of pre-shared secret key between communicating persons. The salient features of this encryption scheme are as follows −

- Every user in this system needs to have a pair of dissimilar keys, **private key** and **public key**. These keys are mathematically related − when one key is used for encryption, the other can decrypt the ciphertext back to the original plaintext.
- It requires to put the public key in public repository and the private key as a well-guarded secret. Hence, this scheme of encryption is also called **Public Key Encryption**.
- Though public and private keys of the user are related, it is computationally not feasible to find one from another. This is a strength of this scheme.

- When *Host1* needs to send data to *Host2*, he obtains the public key of *Host2* from repository, encrypts the data, and transmits.
- *Host2* uses his private key to extract the plaintext.
- Length of Keys (number of bits) in this encryption is large and hence, the process of encryption-decryption is slower than symmetric key encryption.
- Processing power of computer system required to run asymmetric algorithm is higher.

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.

- **Ciphertext Only Attacks (COA)** − In this method, the attacker has access to a set of ciphertext(s). He does not have access to corresponding plaintext. COA is said to be successful when the corresponding plaintext can be determined from a given set of ciphertext. Occasionally, the encryption key can be determined from this attack. Modern cryptosystems are guarded against ciphertext-only attacks.
- **Known Plaintext Attack (KPA)** − In this method, the attacker knows the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext using this information. This may be done by determining the key or via some other method. The best example of this attack is *linear cryptanalysis* against block ciphers.
- **Chosen Plaintext Attack (CPA)** − In this method, the attacker has the text of his choice encrypted. So he has the ciphertext-plaintext pair of his choice. This simplifies his task of determining the encryption key. An example of this attack is *differential cryptanalysis* applied against block ciphers as well as hash functions. A popular public key cryptosystem, RSA is also vulnerable to chosen-plaintext attacks.
- **Dictionary Attack** − This attack has many variants, all of which involve compiling a 'dictionary'. In simplest method of this attack, attacker builds a dictionary of ciphertexts and corresponding plaintexts that he has learnt over a period of time. In future, when an attacker gets the ciphertext, he refers the dictionary to find the corresponding plaintext.
- **Brute Force Attack (BFA)** − In this method, the attacker tries to determine the key by attempting all possible keys. If the key is 8 bits long, then the number of possible keys is $2^8 = 256$. The attacker knows the ciphertext and the algorithm, now he attempts all the 256 keys one by one for decryption. The time to complete the attack would be very high if the key is long.
- **Birthday Attack** − This attack is a variant of brute-force technique. It is used against the cryptographic hash function. When students in a class are asked about their birthdays, the answer is one of the possible 365 dates. Let us assume the first student's birthdate is 3rd Aug. Then to find the next student whose birthdate is 3rd Aug, we need to enquire $1.25^{*} \cdot \sqrt{365} \approx 25$ students.

Similarly, if the hash function produces 64 bit hash values, the possible hash values are $1.8x10^{19}$. By repeatedly evaluating the function for different inputs, the same output is expected to be obtained after about $5.1x10^{9}$ random inputs.

If the attacker is able to find two different inputs that give the same hash value, it is a **collision** and that hash function is said to be broken.

- **Man in Middle Attack (MIM)** − The targets of this attack are mostly public key cryptosystems where key exchange is involved before communication takes place.
    - Host *A* wants to communicate to host *B*, hence requests public key of *B*.
    - An attacker intercepts this request and sends his public key instead.
    - Thus, whatever host *A* sends to host *B*, the attacker is able to read.
    - In order to maintain communication, the attacker re-encrypts the data after reading with his public key and sends to *B*.
    - The attacker sends his public key as *A*'s public key so that *B* takes it as if it is taking it from *A*.
- **Side Channel Attack (SCA)** − This type of attack is not against any particular type of cryptosystem or algorithm. Instead, it is launched to exploit the weakness in physical implementation of the cryptosystem.
- **Timing Attacks** − They exploit the fact that different computations take different times to compute on processor. By measuring such timings, it is be possible to know about a particular computation the processor is carrying out. For example, if the encryption takes a longer time, it indicates that the secret key is long.
- **Power Analysis Attacks** − These attacks are similar to timing attacks except that the amount of power consumption is used to obtain information about the nature of the underlying computations.
- **Fault analysis Attacks** − In these attacks, errors are induced in the cryptosystem and the attacker studies the resulting output for useful information.

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.

SECURITY SERVICES

- The classification of security services are as follows:
- **Confidentiality:** Ensures that the information in a computer system a n d transmitted information are accessible only for reading by authorized parties.
  E.g. Printing, displaying and other forms of disclosure.
- **Authentication:** Ensures that the origin of a message or electronic document is correctly identified, with an assurance that the identity is not false.
- **Integrity:** Ensures that only authorized parties are able to modify computer system assets and transmitted information. Modification includes writing, changing status, deleting, creating and delaying or replaying of transmitted messages.
- **Non repudiation**: Requires that neither the sender nor the receiver of a message be able to deny the transmission.
- **Access control**: Requires that access to information resources may be controlled by or the target system.
- **Availability**: Requires that computer system assets be available to authorized parties when needed.

SECURITY ATTACKS

There are four general categories of attack which are listed below.

Interruption

An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability e.g., destruction of piece of hardware, cutting of a communication line or Disabling of file management system.

Interception

An unauthorized party gains access to an asset. This is an attack on confidentiality. Unauthorized party could be a person, a program or a computer. e.g., wire tapping to capture data in the network, illicit copying of files.

Modification

An unauthorized party not only gains access to but tampers with an asset. This is an attack on integrity. e.g., changing values in data file, altering a program, modifying the contents of messages being transmitted in a network.

Fabrication

An unauthorized party inserts counterfeit objects into the system. This is an attack on authenticity. e.g., insertion of spurious message in a network or addition of records to a file.

CRYPTOGRAPHIC ATTACKS

Passive Attacks

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain Information that is being transmitted.

Passive attacks are of two types:

- **Release of message contents:** A telephone conversation, an e-mail message and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.
- **Traffic analysis**: If we had encryption protection in place, an opponent might still be able to observe the pattern of the message. The opponent could determine the location and identity of communication hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of data. However, it is feasible to prevent the success of these attacks.

Active Attacks

These attacks involve some modification of the data stream or the creation of a false stream. These attacks can be classified in to four categories:

- **Masquerade** – One entity pretends to be a different entity.
- **Replay** – involves passive capture of a data unit and its subsequent transmission to produce an unauthorized effect.
- **Modification of messages** – Some portion of message is altered or the messages are delayed or recorded, to produce an unauthorized effect.
- **Denial of service** – Prevents or inhibits the normal use or management of communication
- facilities. Another form of service denial is the disruption of an entire network, either by disabling the network or overloading it with messages so as to degrade performance.

It is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them.

CLASSICAL ENCRYPTION TECHNIQUES

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

SUBSTITUTION TECHNIQUES

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with cipher text bit patterns.

Caesar cipher (or) shift cipher

The earliest known use of a substitution cipher and the simplest was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing 3 places further down the alphabet.

e.g., plain text : pay more money

Cipher text: SDB PRUH PRQHB

Note that the alphabet is wrapped around, so that letter following „z" is „a".

For each plaintext letter p, substitute the cipher text letter c such that

$$C = E(p) = (p+3) \bmod 26$$

A shift may be any amount, so that general Caesar algorithm is

$$C = E(p) = (p+k) \bmod 26$$

Where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$$P = D(C) = (C-k) \bmod 26$$

Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Recall the assignment for the Caesar cipher:

plain: 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

cipher: 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

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4 x 1026 possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher.**

Playfair cipher

The best known multiple letter encryption cipher is the playfair, which treats diagrams in the plaintext as single units and translates these units into cipher text diagrams. The playfair algorithm is based on the use of 5x5 matrix of letters constructed using a keyword. Let the keyword be „monarchy". The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetical order. The letter „i" and „j" count as one letter. Plaintext is encrypted two letters at a time.

According to the following rules:

- Repeating plaintext letters that would fall in the same pair are separated with a Filler letter such as „x".
- Plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row following the last.
- Plaintext letters that fall in the same column are replaced by the letter beneath, with the top element of the column following the last.

Otherwise, each plaintext letter is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter.

M	O	N	A	R
C	H	Y	B	D

E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Plaintext : meet me at the school house
Splitting two letters as a unit => me et me at th es ch o x ol ho us ex
Corresponding cipher text => CL KL CL RS PD IL HY AV MP HF XL IU

Polyalphabetic ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is polyalphabetic cipher. All the techniques have the following features in common.

- A set of related monoalphabetic substitution rules are used
- A key determines which particular rule is chosen for a given transformation.

Vigenere cipher

In this scheme, the set of related monoalphabetic substitution rules consisting of 26 caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter.

e.g., Caesar cipher with a shift of 3 is denoted by the key value 'd‟ (since a=0, b=1, c=2 and so on). To aid in understanding the scheme, a matrix known as vigenere tableau is Constructed Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top.

The process ofEEncryption is simple: Given a key letter X and a plaintext letter y, the cipher text is at the intersection of the row labeled x and the column labeled y; in this case, the ciphertext is V. To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword.

e.g., key = d e c e p t i v e d e c e p t i v e d e c e p t i v e
PT = w e a r e d i s c o v e r e d s a v e y o u r s e l f
CT = ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Decryption is equally simple. The key letter again identifies the row. The position of the cipher text letter in that row determines the column, and the plaintext letter is at the top of that column.

Strength of Vigenere cipher

o There are multiple cipher text letters for each plaintext letter.
o Letter frequency information is obscured.

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.

e.g., plaintext = meet at the school house

Key = 4 3 1 2 5 6 7

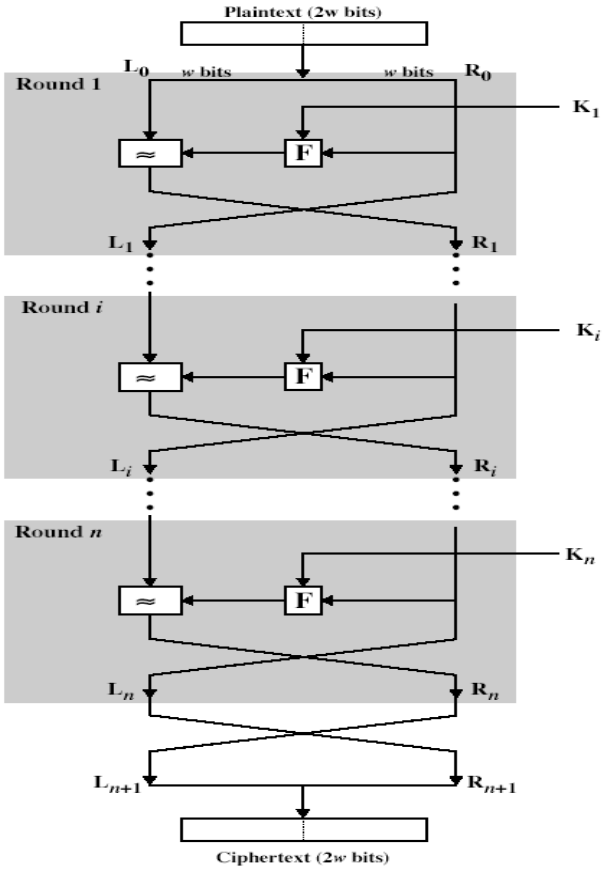
PT = m e e t a t t
h e s c h o o
l h o u s e

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.

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 L0 and R0. The two halves of the data pass through „n" rounds of processing and then combine to produce the ciphertext block. Each round „i" has inputs Li-1 and Ri-1, derived from the previous round, as well as the subkey Ki, derived from the overall key K. in general, the subkeys Ki are different from K and from each other.



Classical Feistel Network

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 sub key 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

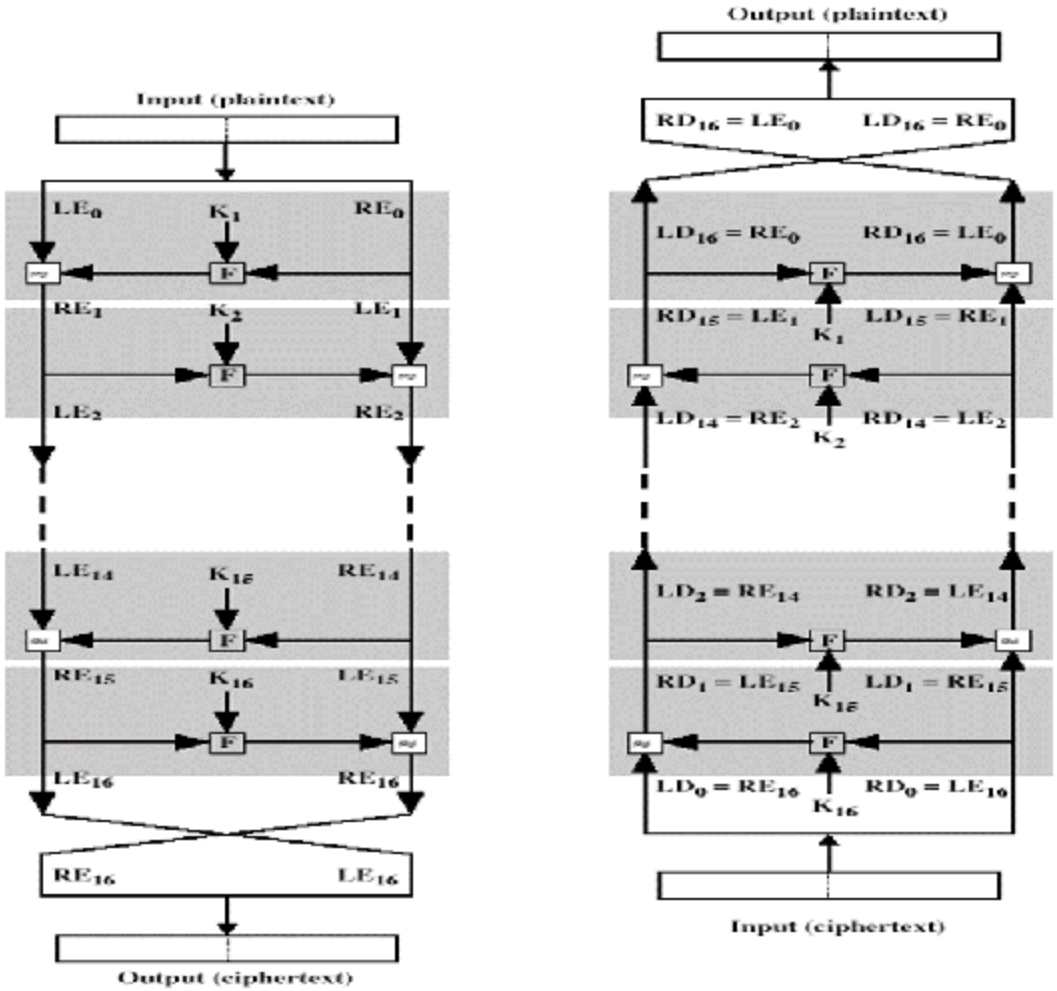
Key size - Increasing size improves security, makes exhaustive key searching harder, but may slow cipher

Number of rounds - Increasing number improves security, but slows cipher

Subkey generation - Greater complexity can make analysis harder, but slows cipher

Round function - Greater complexity can make analysis harder, but slows cipher

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



Feistel encryption and decryption

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 ki in reverse order. i.e., kn in the first round, kn-1 in second round and so on. For clarity, we use the notation LEi and REi 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.

i.e., REi || LEi (or) equivalently RD16-i || LD16-i

After the last iteration of the encryption process, the two halves of the output are swapped, so that the cipher text is RE16 || LE16. 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 RE16 || LE16, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

BLOCK CIPHER PRINCIPLES

Virtually, all symmetric block encryption algorithms in current use are based on a structure referred to as Fiestel block cipher. For that reason, it is important to examine the design

principles of the Fiestel cipher. We begin with a **comparison of stream cipher with block cipher.**

- **A stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. E.g, vigenere cipher.
- **A block cipher** is one in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically a block size of 64 or 128 bits is used.

Block cipher principles

- Most symmetric block ciphers are based on a **Feistel Cipher Structure** needed since must be able to **decrypt** ciphertext to recover messages efficiently. block ciphers look like an extremely large substitution
- Would need table of 264 entries for a 64-bit block
- Instead create from smaller building blocks
- Using idea of a product cipher in 1949 Claude Shannon introduced idea of substitution-permutation (S-P) networks called modern substitution-transposition product cipher
- these form the basis of modern block ciphers
- S-P networks are based on the two primitive cryptographic operations we have seen before:
    - *substitution* (S-box)
    - *permutation* (P-box)
- provide *confusion* and *diffusion* of message
    - **diffusion** – dissipates statistical structure of plaintext over bulk of ciphertext.
    - **confusion** – makes relationship between ciphertext and key as complex as possible.

BLOCK CIPHER MODES OF OPERATION.

i. Electronic Codebook(ECB)
ii. Cipher Block Chaining(CBC)
iii. Cipher Feedback(CFB)
iv. Output Feedback(OFB)
v. Counter(CTR)

A block cipher processes the data blocks of fixed size. Usually, the size of a message is larger than the block size. Hence, the long message is divided into a series of sequential message blocks, and the cipher operates on these blocks one at a time.

Electronic Code Book (ECB) Mode

This mode is a most straightforward way of processing a series of sequentially listed message blocks.

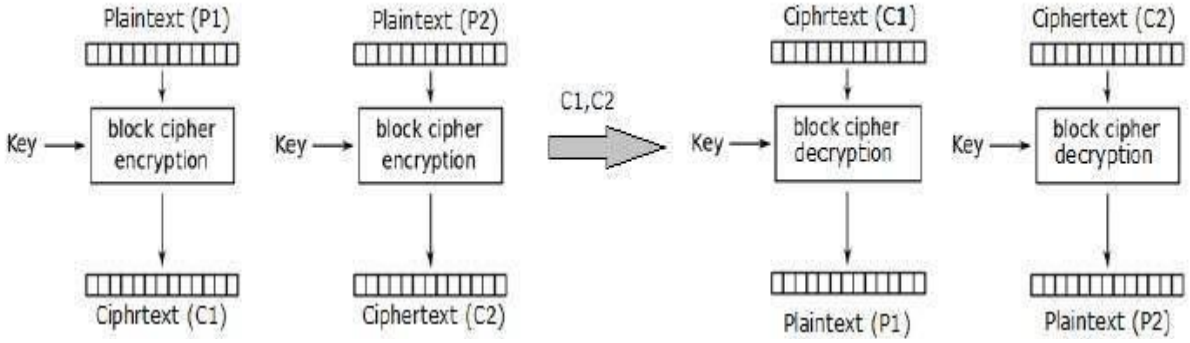
Operation

- The user takes the first block of plaintext and encrypts it with the key to produce the first block of ciphertext.

- He then takes the second block of plaintext and follows the same process with same key and so on so forth.

The ECB mode is **deterministic**, that is, if plaintext block P1, P2,…, Pm are encrypted twice under the same key, the output ciphertext blocks will be the same.

In fact, for a given key technically we can create a codebook of ciphertexts for all possible plaintext blocks. Encryption would then entail only looking up for required plaintext and select the corresponding ciphertext. Thus, the operation is analogous to the assignment of code words in a codebook, and hence gets an official name − Electronic Codebook mode of operation (ECB). It is illustrated as follows −



Analysis of ECB Mode

In reality, any application data usually have partial information which can be guessed. For example, the range of salary can be guessed. A ciphertext from ECB can allow an attacker to guess the plaintext by trial-and-error if the plaintext message is within predictable.

For example, if a ciphertext from the ECB mode is known to encrypt a salary figure, then a small number of trials will allow an attacker to recover the figure. In general, we do not wish to use a deterministic cipher, and hence the ECB mode should not be used in most applications.

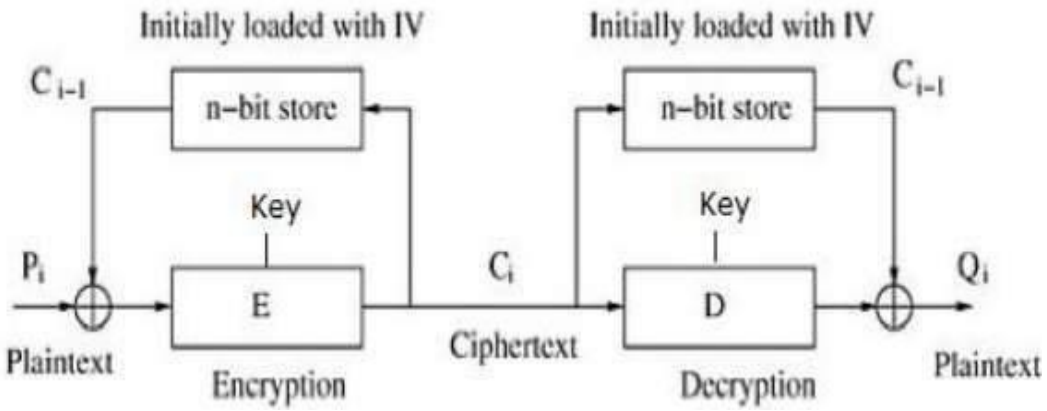
Cipher Block Chaining (CBC) Mode

CBC mode of operation provides message dependence for generating ciphertext and makes the system non-deterministic.

Operation

The operation of CBC mode is depicted in the following illustration. The steps are as follows −

- Load the n-bit Initialization Vector (IV) in the top register.
- XOR the n-bit plaintext block with data value in top register.
- Encrypt the result of XOR operation with underlying block cipher with key K.
- Feed ciphertext block into top register and continue the operation till all plaintext blocks are processed.
- For decryption, IV data is XORed with first ciphertext block decrypted. The first ciphertext block is also fed into to register replacing IV for decrypting next ciphertext block.



Analysis of CBC Mode

In CBC mode, the current plaintext block is added to the previous ciphertext block, and then the result is encrypted with the key. Decryption is thus the reverse process, which involves decrypting the current ciphertext and then adding the previous ciphertext block to the result.

Advantage of CBC over ECB is that changing IV results in different ciphertext for identical message. On the drawback side, the error in transmission gets propagated to few further block during decryption due to chaining effect.

It is worth mentioning that CBC mode forms the basis for a well-known data origin authentication mechanism. Thus, it has an advantage for those applications that require both symmetric encryption and data origin authentication.

Cipher Feedback (CFB) Mode

In this mode, each ciphertext block gets 'fed back' into the encryption process in order to encrypt the next plaintext block.

Operation

The operation of CFB mode is depicted in the following illustration. For example, in the present system, a message block has a size 's' bits where $1 < s < n$. The CFB mode requires an initialization vector (IV) as the initial random n-bit input block. The IV need not be secret. Steps of operation are −

- Load the IV in the top register.
- Encrypt the data value in top register with underlying block cipher with key K.
- Take only 's' number of most significant bits (left bits) of output of encryption process and XOR them with 's' bit plaintext message block to generate ciphertext block.
- Feed ciphertext block into top register by shifting already present data to the left and continue the operation till all plaintext blocks are processed.
- Essentially, the previous ciphertext block is encrypted with the key, and then the result is XORed to the current plaintext block.

- Similar steps are followed for decryption. Pre-decided IV is initially loaded at the start of decryption.

Analysis of CFB Mode

CFB mode differs significantly from ECB mode, the ciphertext corresponding to a given plaintext block depends not just on that plaintext block and the key, but also on the previous ciphertext block. In other words, the ciphertext block is dependent of message.

CFB has a very strange feature. In this mode, user decrypts the ciphertext using only the encryption process of the block cipher. The decryption algorithm of the underlying block cipher is never used.

Apparently, CFB mode is converting a block cipher into a type of stream cipher. The encryption algorithm is used as a key-stream generator to produce key-stream that is placed in the bottom register. This key stream is then XORed with the plaintext as in case of stream cipher.

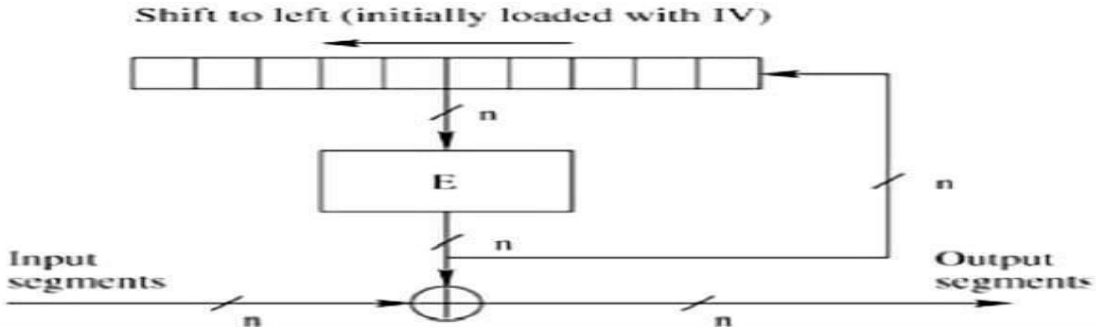
By converting a block cipher into a stream cipher, CFB mode provides some of the advantageous properties of a stream cipher while retaining the advantageous properties of a block cipher.On the flip side, the error of transmission gets propagated due to changing of blocks.

Output Feedback (OFB) Mode

It involves feeding the successive output blocks from the underlying block cipher back to it. These feedback blocks provide string of bits to feed the encryption algorithm which act as the key-stream generator as in case of CFB mode.

The key stream generated is XOR-ed with the plaintext blocks. The OFB mode requires an IV as the initial random n-bit input block. The IV need not be secret.

The operation is depicted in the following illustration −



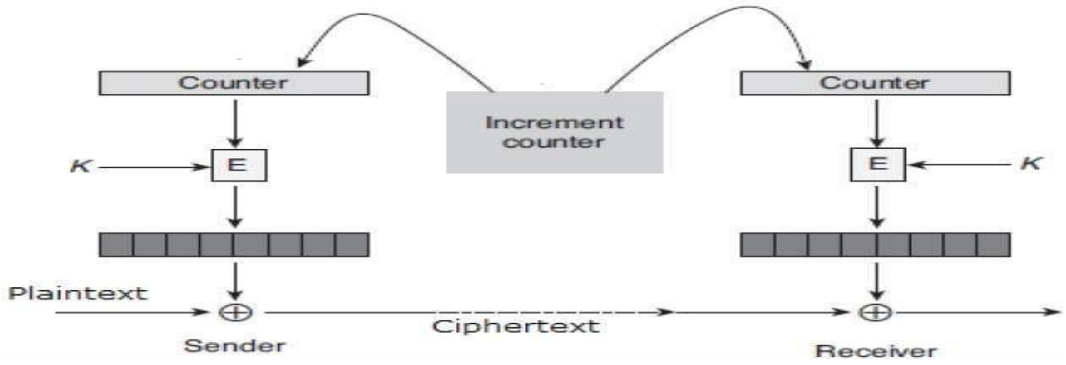
Counter (CTR) Mode

It can be considered as a counter-based version of CFB mode without the feedback. In this mode, both the sender and receiver need to access to a reliable counter, which computes a new shared value each time a ciphertext block is exchanged. This shared counter is not necessarily a secret value, but challenge is that both sides must keep the counter synchronized.

Operation

Both encryption and decryption in CTR mode are depicted in the following illustration. Steps in operation are −

- Load the initial counter value in the top register is the same for both the sender and the receiver. It plays the same role as the IV in CFB (and CBC) mode.
- Encrypt the contents of the counter with the key and place the result in the bottom register.
- Take the first plaintext block P1 and XOR this to the contents of the bottom register. The result of this is C1. Send C1 to the receiver and update the counter. The counter update replaces the ciphertext feedback in CFB mode.
- Continue in this manner until the last plaintext block has been encrypted.
- The decryption is the reverse process. The ciphertext block is XORed with the output of encrypted contents of counter value. After decryption of each ciphertext block counter is updated as in case of encryption.



Analysis of Counter Mode

It does not have message dependency and hence a ciphertext block does not depend on the previous plaintext blocks.

Like CFB mode, CTR mode does not involve the decryption process of the block cipher. This is because the CTR mode is really using the block cipher to generate a key-stream, which is encrypted using the XOR function. In other words, CTR mode also converts a block cipher to a stream cipher.

The serious disadvantage of CTR mode is that it requires a synchronous counter at sender and receiver. Loss of synchronization leads to incorrect recovery of plaintext.

However, CTR mode has almost all advantages of CFB mode. In addition, it does not propagate error of transmission at all.

DATA ENCRYPTION STANDARD (DES)

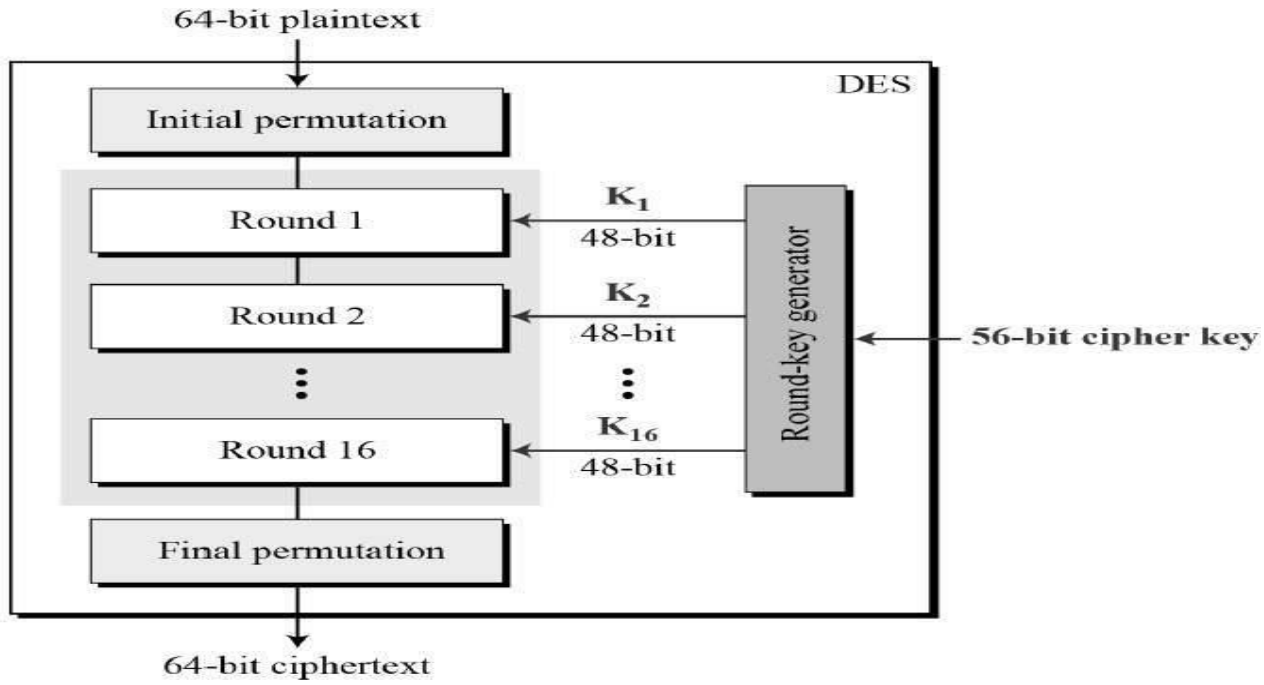
The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits,

since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration

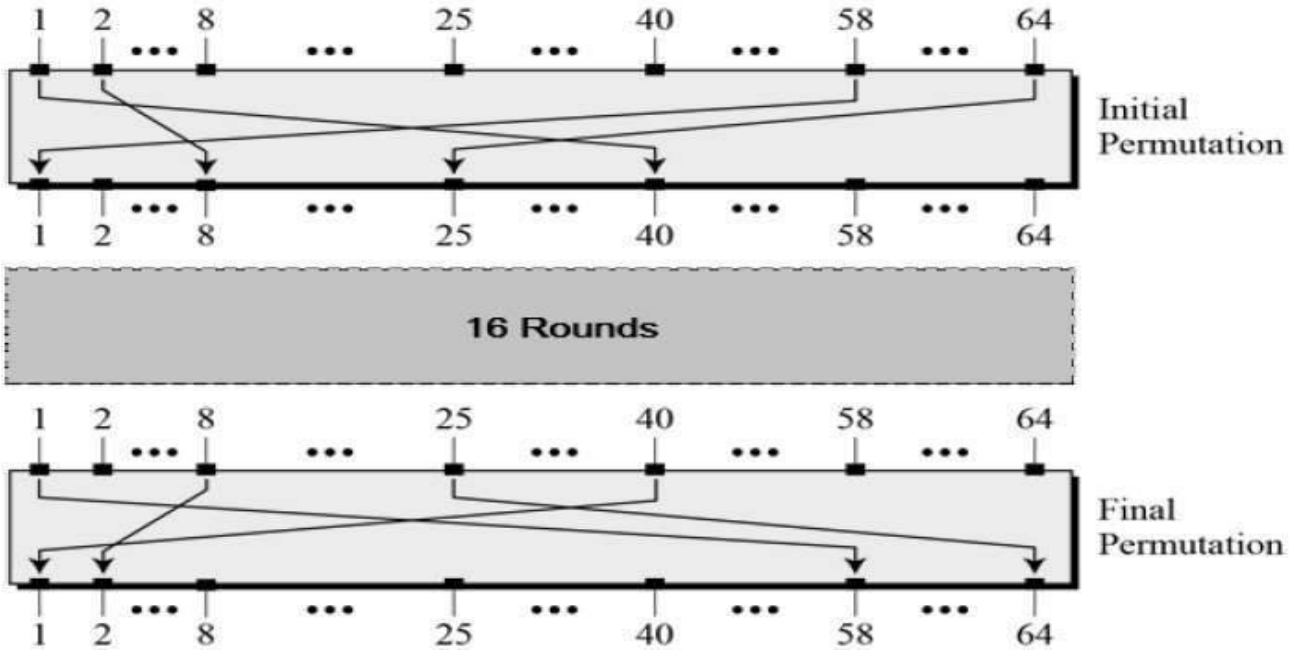
Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation



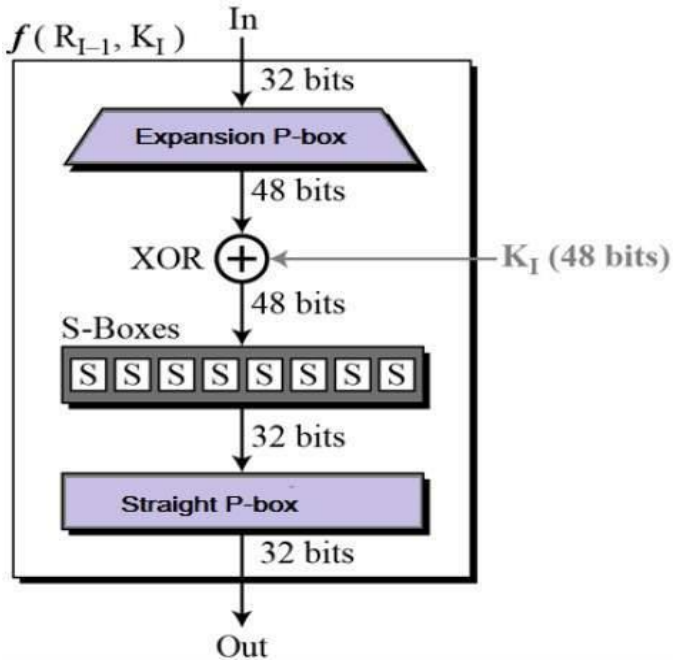
Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows −

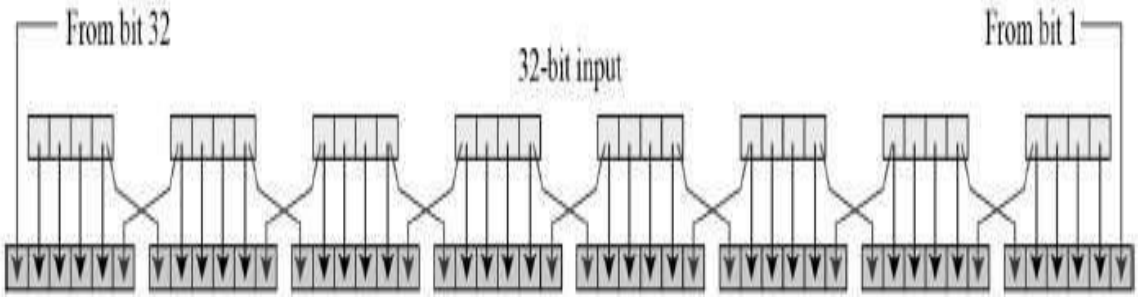


Round Function

The heart of this cipher is the DES function, *f*. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



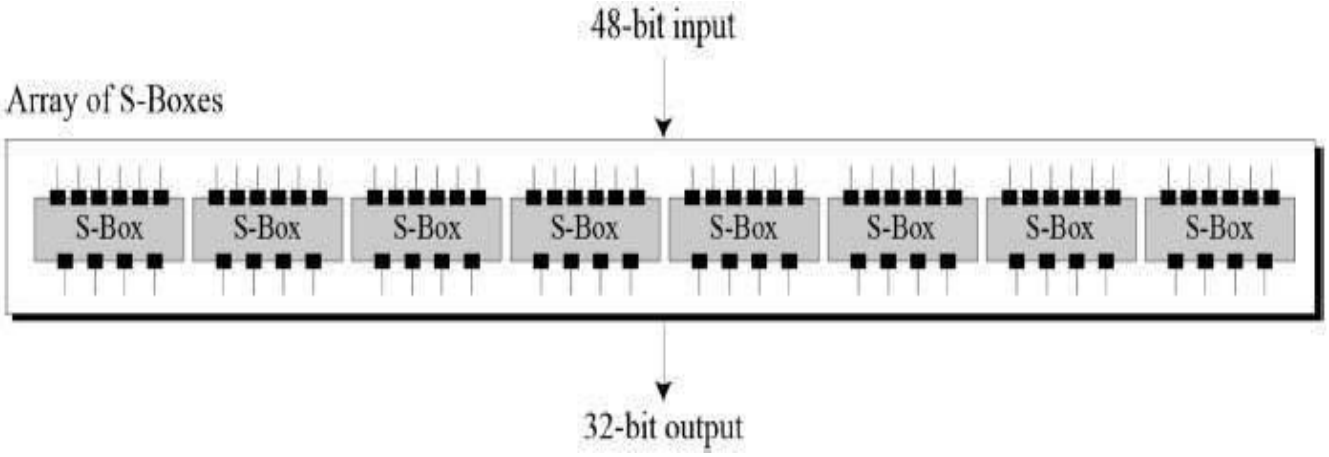
- **Expansion Permutation Box** − Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration −



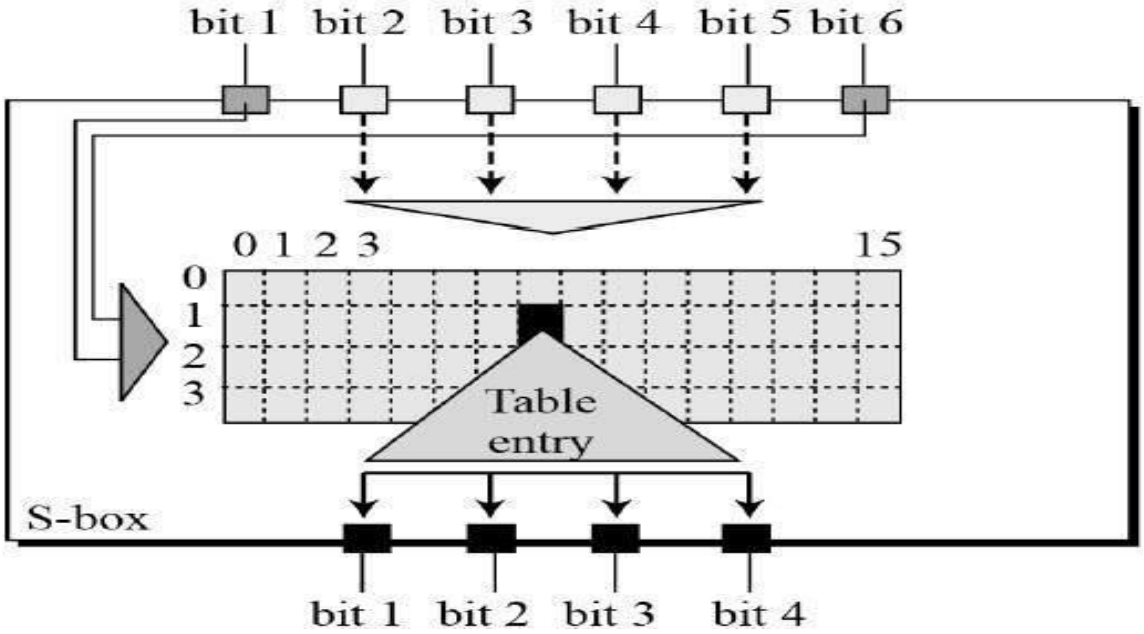
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown −

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

- **XOR (Whitener).** − After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
- **Substitution Boxes.** − The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration −



- The S-box rule is illustrated below −

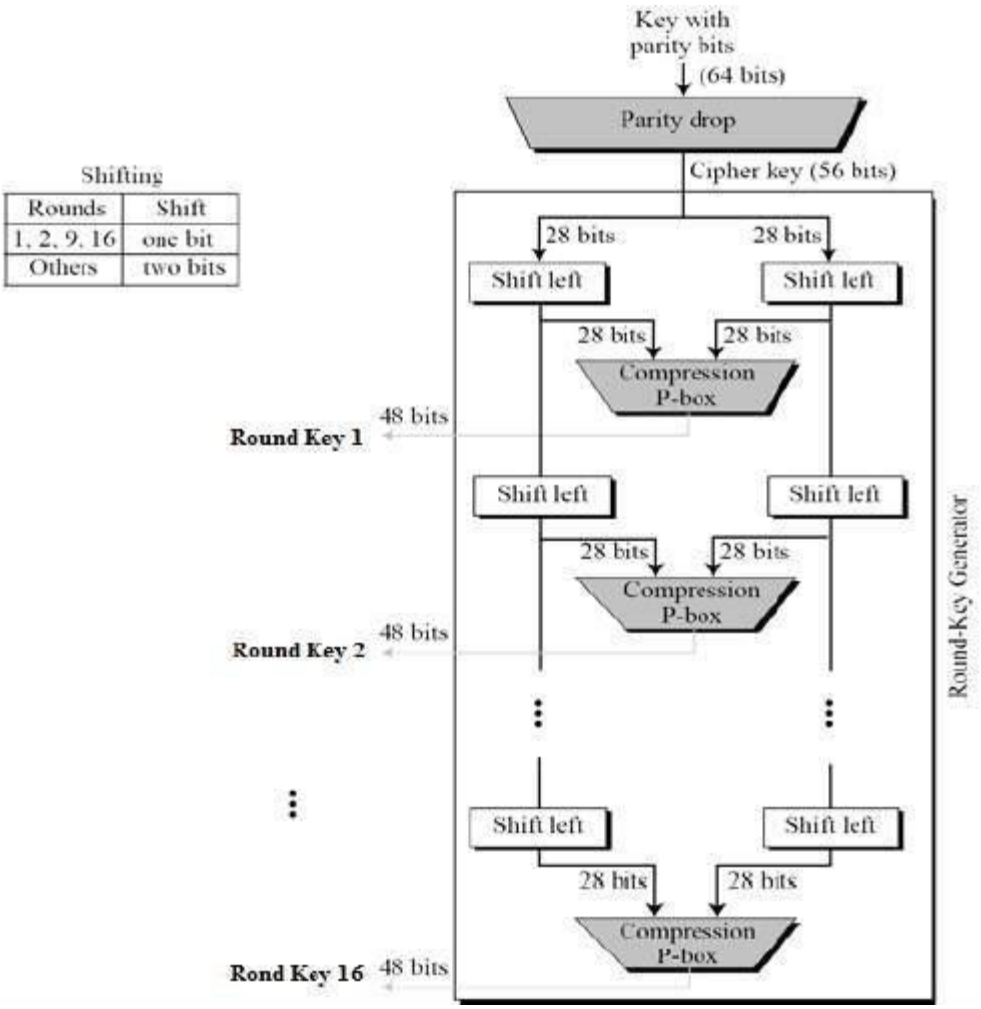


- There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.
- **Straight Permutation** − The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration −



The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plaintext results in the very great change in the ciphertext.
- **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

TRIPLE DES

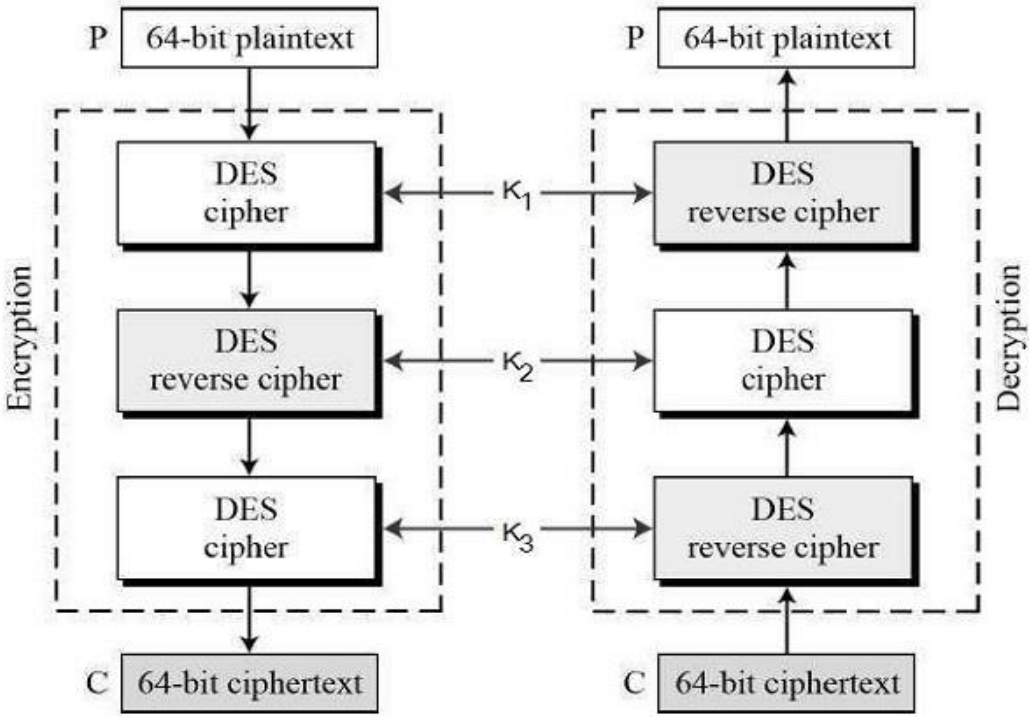
The speed of exhaustive key searches against DES after 1990 began to cause discomfort amongst users of DES. However, users did not want to replace DES as it takes an enormous amount of time and money to change encryption algorithms that are widely adopted and embedded in large security architectures.

The pragmatic approach was not to abandon the DES completely, but to change the manner in which DES is used. This led to the modified schemes of Triple DES (sometimes known as 3DES).

Incidentally, there are two variants of Triple DES known as 3-key Triple DES (3TDES) and 2-key Triple DES (2TDES).

3-KEY Triple DES

Before using 3TDES, user first generate and distribute a 3TDES key K, which consists of three different DES keys K_1, K_2 and K_3. This means that the actual 3TDES key has length 3×56 = 168 bits. The encryption scheme is illustrated as follows −



The encryption-decryption process is as follows −

- Encrypt the plaintext blocks using single DES with key K_1.
- Now decrypt the output of step 1 using single DES with key K_2.
- Finally, encrypt the output of step 2 using single DES with key K_3.
- The output of step 3 is the ciphertext.
- Decryption of a ciphertext is a reverse process. User first decrypt using K_3, then encrypt with K_2, and finally decrypt with K_1.

Due to this design of Triple DES as an encrypt–decrypt–encrypt process, it is possible to use a 3TDES (hardware) implementation for single DES by setting K_1, K_2, and K_3 to be the same value. This provides backwards compatibility with DES.

Second variant of Triple DES (2TDES) is identical to 3TDES except that $K_3$is replaced by K_1. In other words, user encrypt plaintext blocks with key K_1, then decrypt with key K_2, and finally encrypt with K_1 again. Therefore, 2TDES has a key length of 112 bits.

Triple DES systems are significantly more secure than single DES, but these are clearly a much slower process than encryption using single DES.

FINITE FIELD

Groups, Rings, and Fields

Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra. In abstract algebra, we are concerned with sets on whose elements we can operate algebraically;

Groups

A **group** G, sometimes denoted by $\{G, \cdot\}$ is a set of elements with a binary operation, denoted by $\cdot$, that associates to each ordered pair (a, b) of elements in G an element $(a \cdot b)$ in G, such that the following axioms are obeyed:

The operator $\cdot$ is generic and can refer to addition, multiplication, or some other mathematical operation.

(A1) Closure: If a and b belong to G, then $a \cdot b$ is also in G.

(A2) Associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b, c in G.

(A3) Identity element: There is an element e in G such that $a \cdot e = e \cdot a = a$ for all a in G.

(A4) Inverse element: For each a in G there is an element a' in G such that $a \cdot a' = a' \cdot a = e$.

If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**. A group is said to be **abelian** if it satisfies the following additional condition:

(A5) Commutative: $a \cdot b = b \cdot a$ for all a, b in G.

Rings

A **ring** R, sometimes denoted by $\{R, +, \times\}$, is a set of elements with two binary operations, called *addition* and *multiplication* such that for all a, b, c in R the following axioms are obeyed:

(A1-A5) R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as $-a$.

(M1) Closure under multiplication: If a and b belong to R, then ab is also in R.

(M2) Associativity of multiplication: $a(bc) = (ab)c$ for all a, b, c in R.

(M3) Distributive laws: $a(b + c) = ab + ac$ for all a, b, c in R.

$(a + b)c = ac + bc$ for all a, b, c in R.

A ring is said to be **commutative** if it satisfies the following additional condition:

(M4) Commutativity of multiplication: $ab = ba$ for all a, b in R.

Next, we define an **integral domain**, which is a commutative ring that obeys the following axioms:

(M5) Multiplicative identity: There is an element 1 in R such that $a1 = 1a = a$ for all a in R.

(M6) No zero divisors: If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Fields

A **field** F, sometimes denoted by $\{F, +, \times\}$, is a set of elements with two binary operations, called *addition* and *multiplication*, such that for all a, b, c in F the following axioms are obeyed:

(A1M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.

(M7) Multiplicative inverse: For each a in F, except 0, there is an element a-1 in F such that aa-1 = (a- 1)a = 1.

In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b\text{-}1)$.

ADVANCED ENCRYPTION STANDARD

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

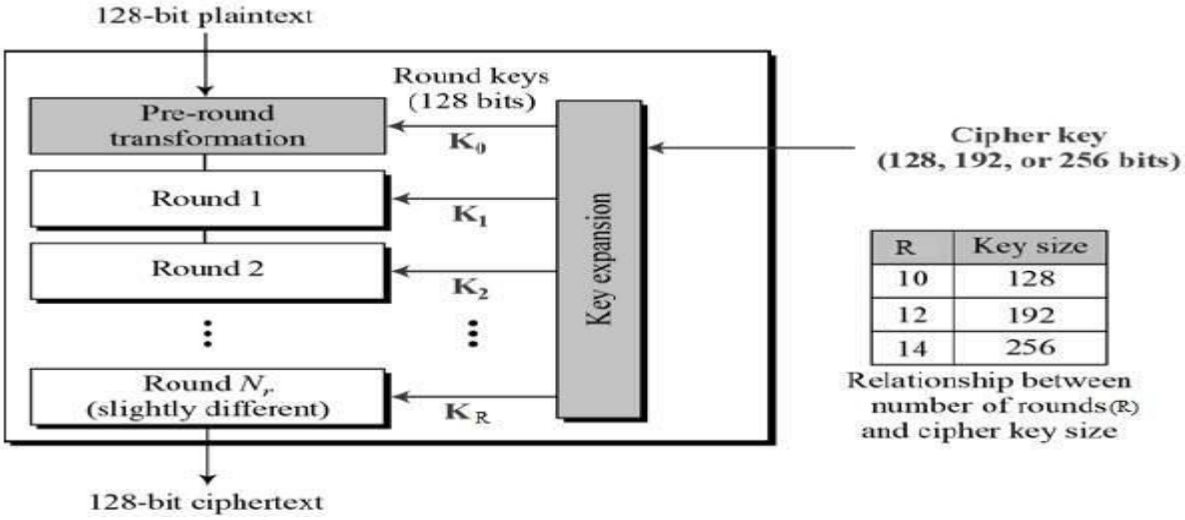
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

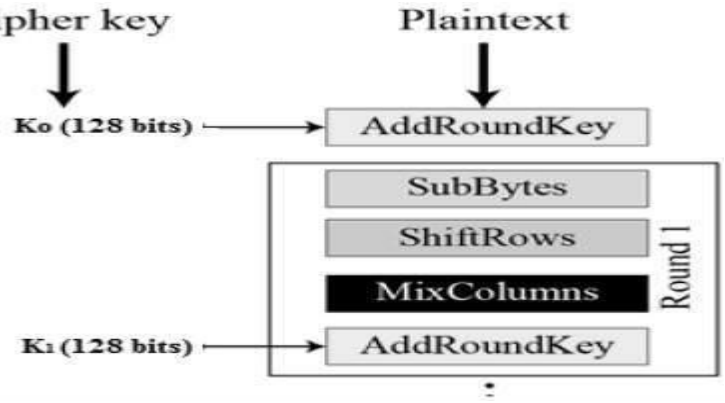


R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

CONFIDENTIALITY USING SYMMETRIC ENCRYPTION

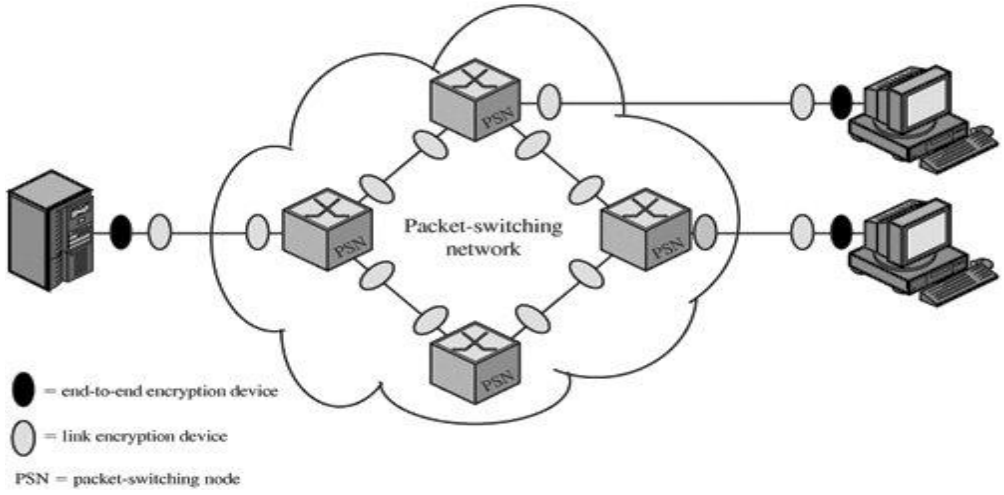
Two approaches are

1. Link encryption
2. End to end encryption

Link versus End-to-End Encryption

With link encryption, each vulnerable communications link is equipped on both ends with an encryption device. Thus, all traffic over all communications links is secured. Although this recourse requires a lot of encryption devices in a large network, its value is clear. One of its disadvantages is that the message must be decrypted each time it enters a switch (such as a frame relay switch) because the switch must read the address (logical connection number) in the packet header in order to route the frame. Thus, the message is vulnerable at each switch. If working with a public network, the user has no control over the security of the nodes. Several implications of link encryption should be noted. For this strategy to be effective, all the potential

links in a path from source to destination must use link encryption. Each pair of nodes that share a link should share a unique key, with a different key used on each link. Thus, many keys must be provided.



With end-to-end encryption, the encryption process is carried out at the two end systems. The source host or terminal encrypts the data. The data in encrypted form are then transmitted unaltered across the network to the destination terminal or host. The destination shares a key with the source and so is able to decrypt the data. This plan seems to secure the transmission against attacks on the network links or switches. Thus, end-to-end encryption relieves the end user of concerns about the degree of security of networks and links that support the communication. There is, however, still a weak spot.

Consider the following situation. A host connects to a frame relay or ATM network, sets up a logical connection to another host, and is prepared to transfer data to that other host by using end-to-end encryption. Data are transmitted over such a network in the form of packets that consist of a header and some user data. What part of each packet will the host encrypt? Suppose that the host encrypts the entire packet, including the header. This will not work because, remember, only the other host can perform the decryption. The frame relay or ATM switch will receive an encrypted packet and be unable to read the header. Therefore, it will not be able to route the packet. It follows that the host may encrypt only the user data portion of the packet and must leave the header in the clear.

Thus, with end-to-end encryption, the user data are secure. However, the traffic pattern is not, because packet headers are transmitted in the clear. On the other hand, end-to-end encryption does provide a degree of authentication. If two end systems share an encryption key, then a recipient is assured that any message that it receives comes from the alleged sender, because only that sender shares the relevant key. Such authentication is not inherent in a link encryption scheme.To achieve greater security, both link and end-to-end encryption are needed,
When both forms of encryption are employed, the host encrypts the user data portion of a packet using an end-to-end encryption key. The entire packet is then encrypted using a link encryption key. As the packet traverses the network, each switch decrypts the packet, using a link

encryption key to read the header, and then encrypts the entire packet again for sending it out on the next link. Now the entire packet is secure except for the time that the packet is actually in the memory of a packet switch, at which time the packet header is in the clear.

Link Encryption	**Link Encryption**
Security within End Systems and Intermediate Systems	
Message exposed in sending host	Message encrypted in sending host
Message exposed in intermediate nodes	Message encrypted in intermediate nodes
Role of User	
Applied by sending host	Applied by sending process
Transparent to user	User applies encryption
Host maintains encryption facility	User must determine algorithm
One facility for all users	Users selects encryption scheme
Can be done in hardware	Software implementation
All or no messages encrypted	User chooses to encrypt, or not, for each message
Implementation Concerns	
Requires one key per (host-intermediate node) pair and (intermediate node-intermediate node) pair	Requires one key per user pair
Provides host authentication	Provides user authentication

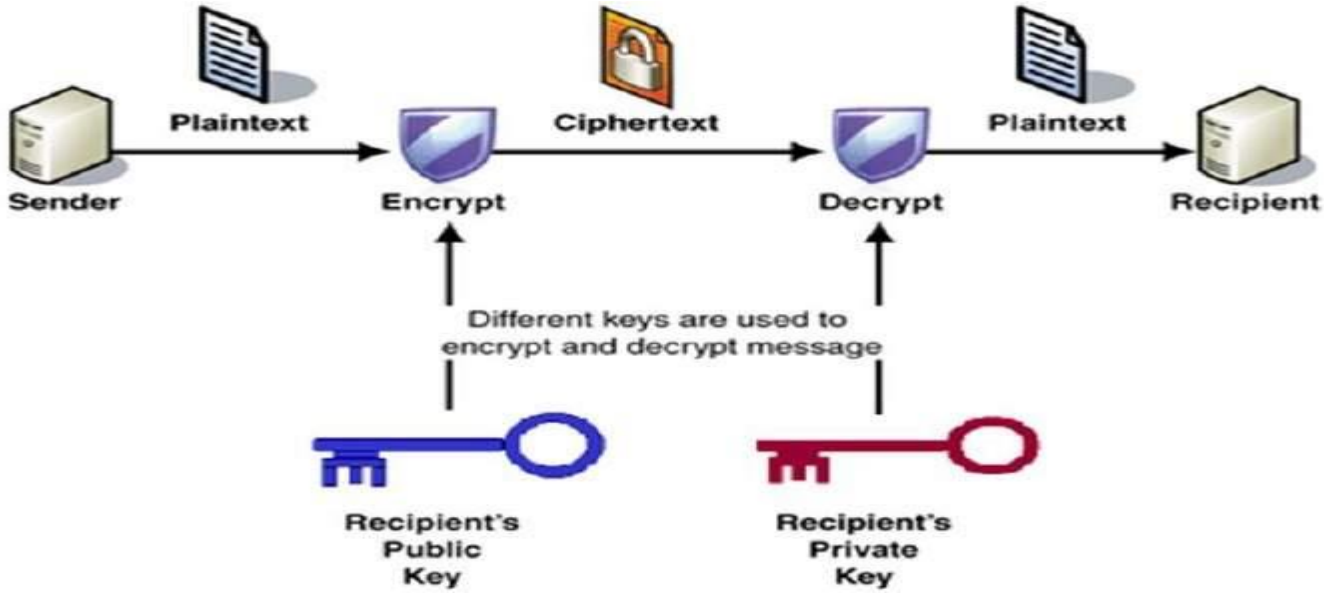
Unit-II

Public Key Encryption and Hash Functions: Introduction to Number Theory – Public Key Cryptography and RSA - Key Management- other Public Key Cryptosystem – Message Authentication and Hash Functions – Hash and MAC Algorithms – Digital Signatures and Authentication Protocols.

PUBLIC KEY CRYPTOGRAPHY

Unlike symmetric key cryptography, we do not find historical use of public-key cryptography. It is a relatively new concept. Symmetric cryptography was well suited for organizations such as governments, military, and big financial corporations were involved in the classified communication. With the spread of more unsecure computer networks in last few decades, a genuine need was felt to use cryptography at larger scale. The symmetric key was found to be non-practical due to challenges it faced for key management. This gave rise to the public key cryptosystems.

The process of encryption and decryption is depicted in the following illustration −



The most important properties of public key encryption scheme are −

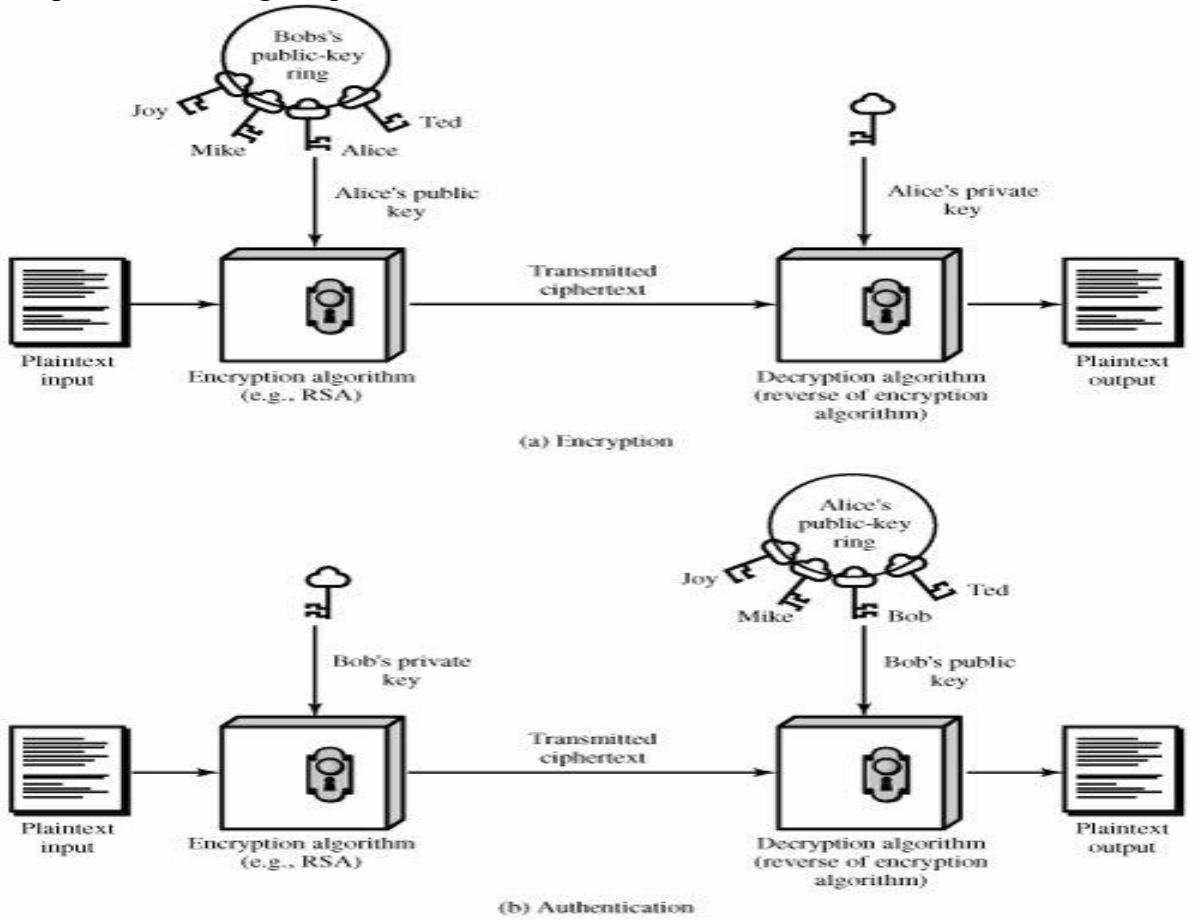
- Different keys are used for encryption and decryption. This is a property which set this scheme different than symmetric encryption scheme.
- Each receiver possesses a unique decryption key, generally referred to as his private key.
- Receiver needs to publish an encryption key, referred to as his public key.
- Some assurance of the authenticity of a public key is needed in this scheme to avoid spoofing by adversary as the receiver. Generally, this type of cryptosystem involves trusted third party which certifies that a particular public key belongs to a specific person or entity only.
- Encryption algorithm is complex enough to prohibit attacker from deducing the plaintext from the ciphertext and the encryption (public) key.

Though private and public keys are related mathematically, it is not be feasible to calculate the private key from the public key. In fact, intelligent part of any public-key cryptosystem is in designing a relationship between two keys.

A public-key encryption scheme has six ingredients

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.



RSA Algorithm

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem.

We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

- **Generate the RSA modulus (n)**
    - Select two large primes, p and q.

- Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

- **Find Derived Number (e)**
  - Number **e** must be greater than 1 and less than $(p - 1)(q - 1)$.
  - There must be no common factor for e and $(p - 1)(q - 1)$ except for 1. In other words two numbers e and $(p – 1)(q – 1)$ are co prime.
- **Form the public key**
  - The pair of numbers (n, e) form the RSA public key and is made public.
  - Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.
- **Generate the private key**
  - Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.
  - Number d is the inverse of e modulo $(p - 1)(q – 1)$. This means that d is the number less than $(p - 1)(q - 1)$ such that when multiplied by e, it is equal to 1 modulo $(p - 1)(q - 1)$.
  - This relationship is written mathematically as follows −

$$ed = 1 \bmod (p - 1)(q - 1)$$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and $(p - 1)(q - 1) = 6 \times 12 = 72$, except for 1.
- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.
- Check that the d calculated is correct by computing −

$$de = 29 \times 5 = 145 = 1 \bmod 72$$

- Hence, public key is (91, 5) and private keys is (91, 29).

Encryption and Decryption

Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.

Interestingly, RSA does not directly operate on strings of bits as in case of symmetric key encryption. It operates on numbers modulo n. Hence, it is necessary to represent the plaintext as a series of numbers less than n.

RSA Encryption

- Suppose the sender wish to send some text message to someone whose public key is (n, e).
- The sender then represents the plaintext as a series of numbers less than n.
- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as −

$$C = P^e \bmod n$$

- In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.
- Returning to our Key Generation example with plaintext P = 10, we get ciphertext C −

$$C = 10^5 \bmod 91$$

RSA Decryption

- The decryption process for RSA is also very straightforward. Suppose that the receiver of public-key pair (n, e) has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

$$\text{Plaintext} = C^d \bmod n$$

- Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −

$$\text{Plaintext} = 82^{29} \bmod 91 = 10$$

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Analysis

The security of RSA depends on the strengths of two separate functions. The RSA cryptosystem is most popular public-key cryptosystem strength of which is based on the practical difficulty of factoring the very large numbers.

- **Encryption Function** − It is considered as a one-way function of converting plaintext into ciphertext and it can be reversed only with the knowledge of private key d.
- **Key Generation** − The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one way function, going from p & q values to modulus n is easy but reverse is not possible.

The strength of RSA encryption drastically goes down against attacks if the number p and q are not large primes and/ or chosen public key e is a small number.

KEY MANAGEMENT

One of the major roles of public-key encryption has been to address the problem of key distribution. There are actually two distinct aspects to the use of public-key cryptography in this regard:

- The distribution of public keys
- The use of public-key encryption to distribute secret keys

Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Public Announcement of Public Keys

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large.

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers 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.

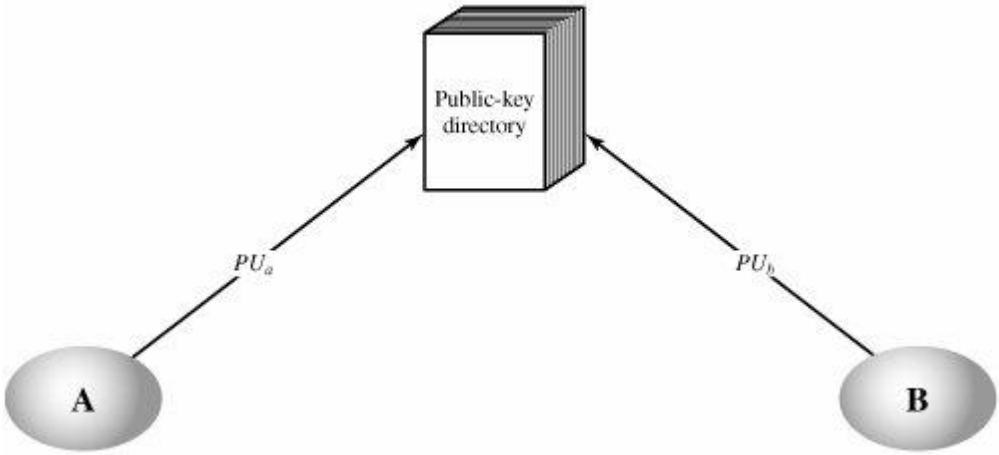
Such a scheme would include the following elements:

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. R

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 is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. The scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps

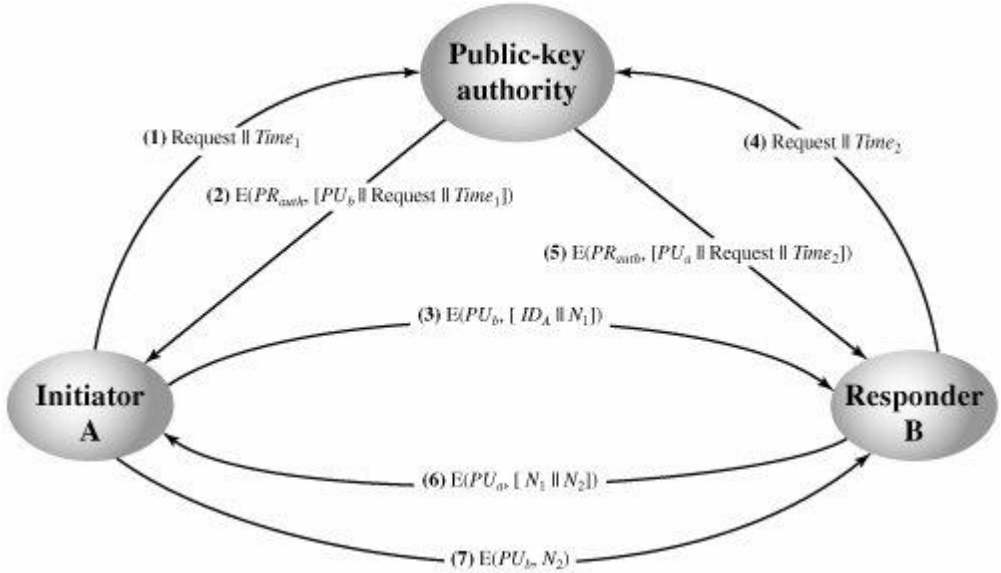
1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.
2. The authority responds with a message that is encrypted using the authority's private key, *PRauth*

Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

- B's public key, *PUb* which A can use to encrypt messages destined for B
- The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority.

The original timestamp, so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (*IDA*) and a nonce (*N*1), which is used to identify this transaction uniquely.
4. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.
5. At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:
6. B sends a message to A encrypted with *PUa* and containing A's nonce (*N*1) as well as a new nonce generated by B (*N*2) Because only B could have decrypted message (3), the presence of *N*1 in message (6) assures A that the correspondent is B.
7. A returns *N*2, encrypted using B's public key, to assure B that its correspondent is A.



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 scenario of Figure 10.3 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, to use **certificates** that can be used by participants 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. In essence, a certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. 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.

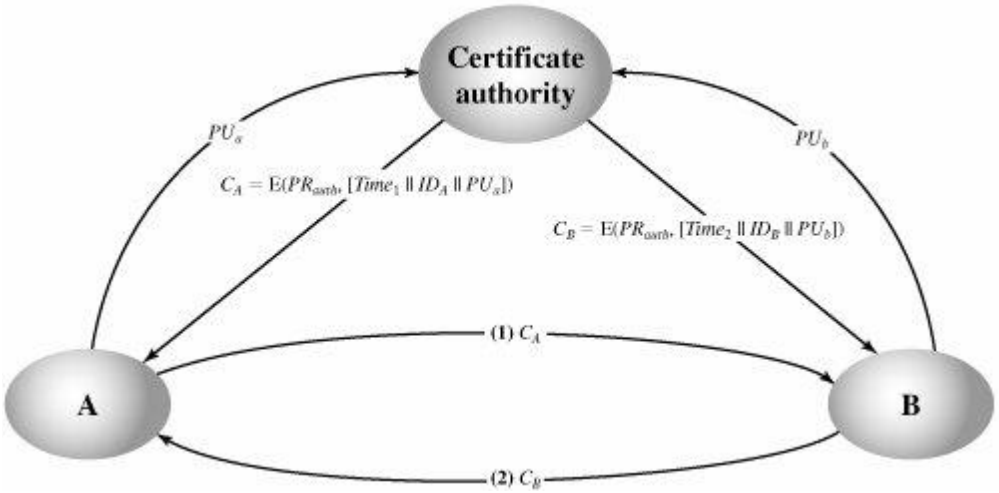
Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.

2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.

4. Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in Figure. Each participant applies to the certificate authority, supplying a public key and requesting a 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

CA = E(*PRauth*, [*T*||*IDA*||*PUa*])

where *PRauth* 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:

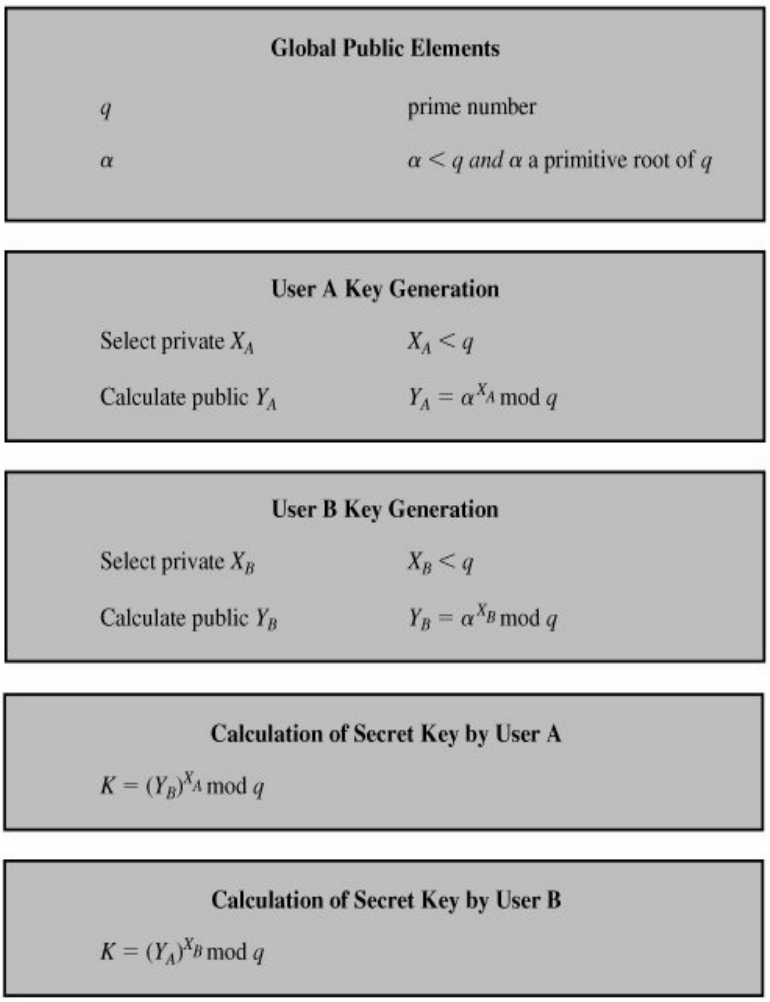
D(*PUauth*, *CA*) = D(*PUauth*, E(*PR*auth, [*T*||*IDA*||*PUa*])) = (*T*||*IDA*||*PUa*)

The recipient uses the authority's public key, *PUauth* 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 *IDA* and *PUa* provide the recipient with the name and public key of the certificate's holder. The timestamp *T* validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

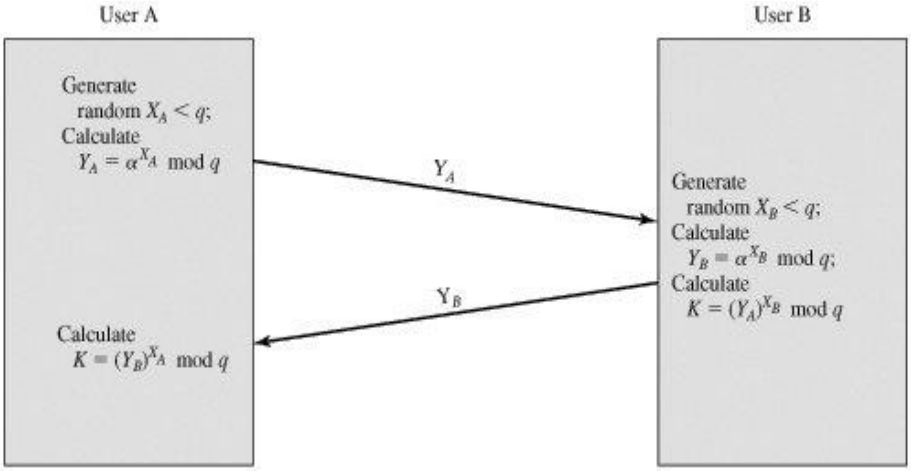
DIFFIE- HELLMAN KEY EXCHANGE.

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography and is generally referred to as Diffie-Hellman key exchange. The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

- It is not an encryption algorithm
- It Exchange Secret/Symmetric key between end users
- It use Asymmetric encryption



i. Assume X_A (Private Key of user A) X_A= q
ii. Calculate Y_A(Public key of user A) $Y_A = \alpha^{XA} \bmod q$
iii. Assume X_B (Private Key of user A) X_B= q
iv. Calculate Y_B (Public key of user A) $Y_B = \alpha^{XB} \bmod q$
v. K= $(Y_B)^{XA} \bmod q$ for User A
vi. K= $(Y_A)^{XB} \bmod q$ for User B



Example

i. q=11 , α = 2

ii. α is a primitive root of p, that is α mod p, α^2mod p,… α^{p-1}mod p is 1,2,3…p-1.

iii. Select $X_A = 8$

$Y_A = \alpha^{XA} \bmod q$

$Y_A = 2^8 \bmod 11$

$Y_A = 3$

iv. Select $X_B = 4$

$Y_B = \alpha^{XB} \bmod q$

$Y_B = 2^4 \bmod 11$

$Y_B = 5$

v. For User A

$K = (Y_B)^{XA} \bmod q$

$K = (5)^8 \bmod 11$

K = 4

vi. For User B

$K = (Y_A)^{XB} \bmod q$

$K = (3)^4 \bmod 11$

K = 4

User A and B have a same Key values.

ElGamal Cryptosystem

Along with RSA, there are other public-key cryptosystems proposed. Many of them are based on different versions of the Discrete Logarithm Problem.

ElGamal cryptosystem, called Elliptic Curve Variant, is based on the Discrete Logarithm Problem. It derives the strength from the assumption that the discrete logarithms cannot be found in practical time frame for a given number, while the inverse operation of the power can be computed efficiently.

Let us go through a simple version of ElGamal that works with numbers modulo p. In the case of elliptic curve variants, it is based on quite different number systems.

Generation of ElGamal Key Pair

Each user of ElGamal cryptosystem generates the key pair through as follows −

- **Choosing a large prime p.** Generally a prime number of 1024 to 2048 bits length is chosen.
- **Choosing a generator element g.**
    - This number must be between 1 and p − 1, but cannot be any number.
    - It is a generator of the multiplicative group of integers modulo p. This means for every integer m co-prime to p, there is an integer k such that g^k=a mod n.

      For example, 3 is generator of group 5 (Z_5 = {1, 2, 3, 4}).

N	3^n	3^n mod 5
1	3	3
2	9	4
3	27	2
4	81	1

- **Choosing the private key.** The private key x is any number bigger than 1 and smaller than p−1.
- **Computing part of the public key.** The value y is computed from the parameters p, g and the private key x as follows −

$$y = g^x \bmod p$$

- **Obtaining Public key.** The ElGamal public key consists of the three parameters (p, g, y).

  For example, suppose that p = 17 and that g = 6 (It can be confirmed that 6 is a generator of group Z_{17}). The private key x can be any number bigger than 1 and smaller than 71, so we choose x = 5. The value y is then computed as follows −

$y = 6^5 \bmod 17 = 7$

- Thus the private key is 62 and the public key is (17, 6, 7).

Encryption and Decryption

The generation of an ElGamal key pair is comparatively simpler than the equivalent process for RSA. But the encryption and decryption are slightly more complex than RSA.

ElGamal Encryption

Suppose sender wishes to send a plaintext to someone whose ElGamal public key is (p, g, y), then −

- Sender represents the plaintext as a series of numbers modulo p.
- To encrypt the first plaintext P, which is represented as a number modulo p. The encryption process to obtain the ciphertext C is as follows −
    - Randomly generate a number k;
    - Compute two values C1 and C2, where −

$$C1 = g^k \bmod p$$
$$C2 = (P^* y^k) \bmod p$$

- Send the ciphertext C, consisting of the two separate values (C1, C2), sent together.
- Referring to our ElGamal key generation example given above, the plaintext P = 13 is encrypted as follows −
    - Randomly generate a number, say k = 10
    - Compute the two values C1 and C2, where −

$$C1 = 6^{10} \bmod 17$$
$$C2 = (13^* 7^{10}) \bmod 17 = 9$$

- Send the ciphertext C = (C1, C2) = (15, 9).

ElGamal Decryption

- To decrypt the ciphertext (C1, C2) using private key x, the following two steps are taken −
    - Compute the modular inverse of $(C1)^x$ modulo p, which is $(C1)^{-x}$, generally referred to as decryption factor.
    - Obtain the plaintext by using the following formula −

$$C2 \times (C1)^{-x} \bmod p = \text{Plaintext}$$

- In our example, to decrypt the ciphertext C = (C1, C2) = (15, 9) using private key x = 5, the decryption factor is

$15^{-5} \mod 17 = 9$

- Extract plaintext P = (9 × 9) mod 17 = 13.

ElGamal Analysis

In ElGamal system, each user has a private key x. and has **three components** of public key − **prime modulus p, generator g, and public $Y = g^x \mod p$**. The strength of the ElGamal is based on the difficulty of discrete logarithm problem.

The secure key size is generally > 1024 bits. Today even 2048 bits long key are used. On the processing speed front, Elgamal is quite slow, it is used mainly for key authentication protocols. Due to higher processing efficiency, Elliptic Curve variants of ElGamal are becoming increasingly popular.

MESSAGE AUTHENTICATION AND HASH FUNCTIONS

Authentication Requirements

In the context of communications across a network, the following attacks can be identified:

1. Disclosure: Release of message contents to any person or process not possessing the appropriate cryptographic key.

2. Traffic analysis: Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

3. Masquerade: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.

4. Content modification: Changes to the contents of a message, including insertion, deletion, transposition, and modification.

5. Sequence modification: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. Timing modification: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.

7. Source repudiation: Denial of transmission of message by source.

8. Destination repudiation: Denial of receipt of message by destination.

AUTHENTICATION FUNCTIONS

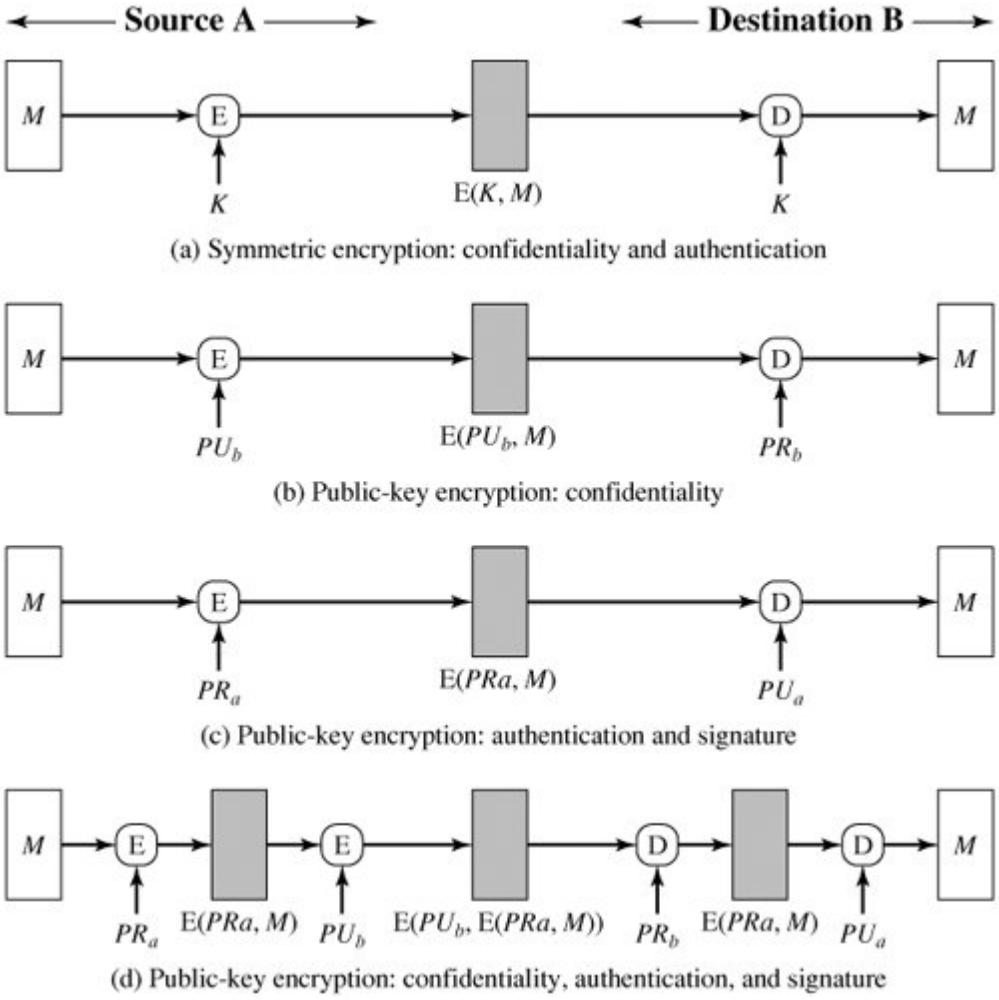
- **Message encryption:** The ciphertext of the entire message serves as its authenticator
- **Message authentication code (MAC)**: A function of the message and a secret key that produces a fixed-length value that serves as the authenticator
- **Hash function:** A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator.

Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Symmetric Encryption

Consider the straightforward use of symmetric encryption in the below figure. A message *M* transmitted from source A to destination B is encrypted using a secret key *K* shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.



MESSAGE AUTHENTICATION CODE

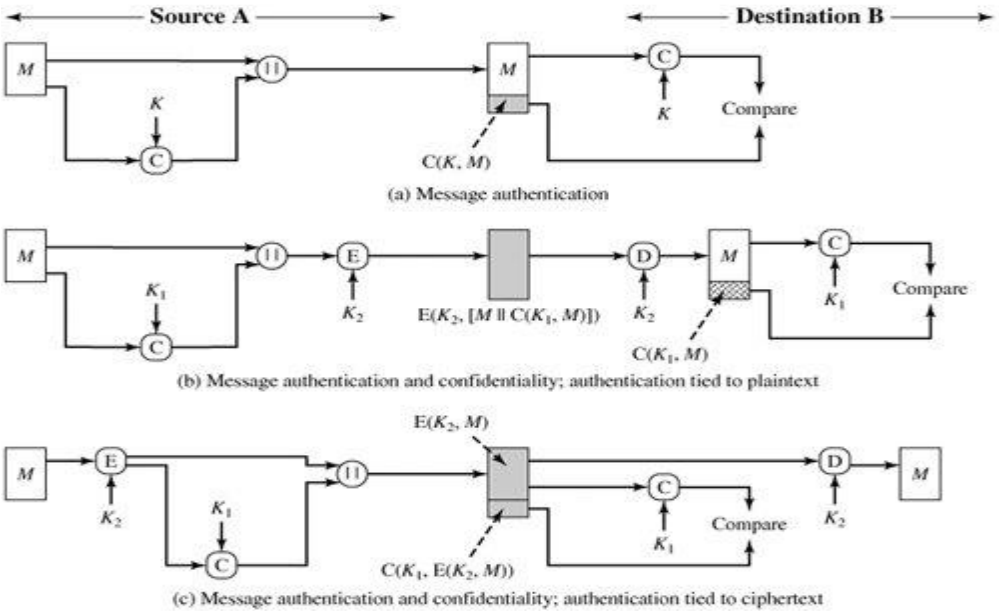
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key *K*. When A has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

where

M = input message
C = MAC function
K = shared secret key
MAC = message authentication code

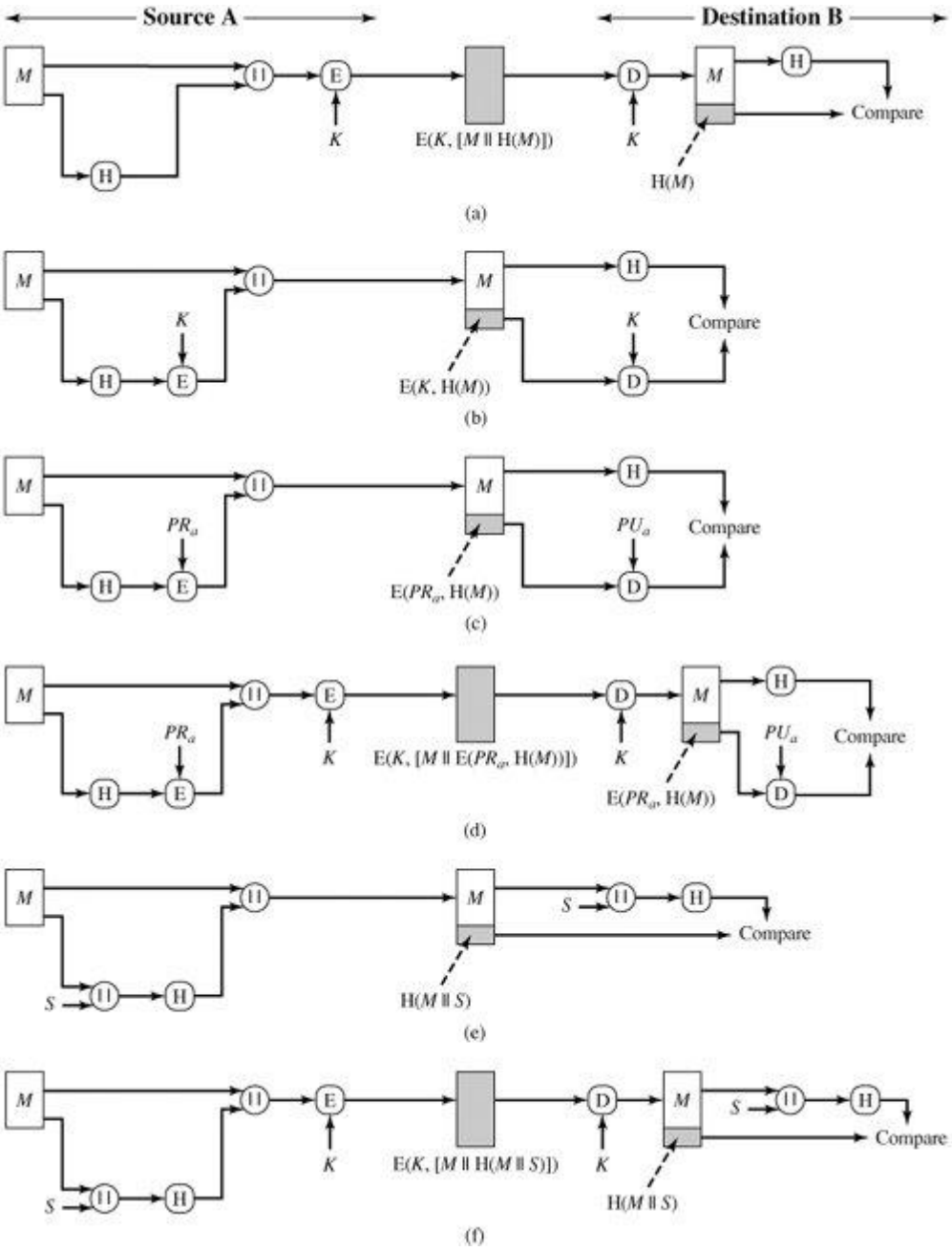
The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.



HASH FUNCTION

A variation on the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message M as input and produces a fixed size output, referred to as a **hash code** H(M). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a **message digest** or **hash value**. The hash code is a function of all the 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.



Basic Uses of Hash Function

a. The message plus concatenated hash code is encrypted using symmetric encryption. A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

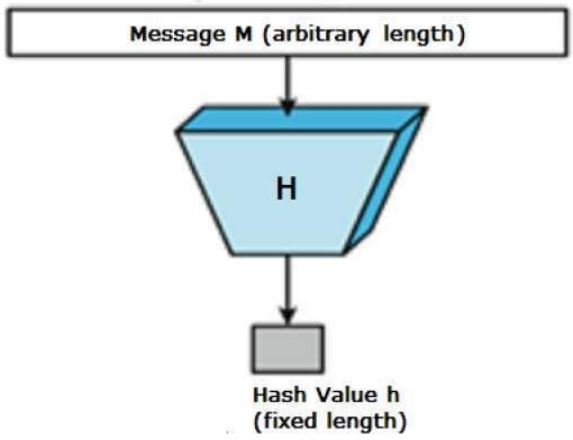
b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

c. Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

d. If confidentiality as well as a digital signature is desired, then the message plus the private-key encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

e. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value *S*. A computes the hash value over the concatenation of *M* and *S* and appends the resulting hash value to *M*. Because B possesses *S*, it can re-compute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.
f. Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

Hash functions are extremely useful and appear in almost all information security applications. A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length. Values returned by a hash function are called **message digest** or simply **hash values**. The following picture illustrated hash function.



Features of Hash Functions

The typical features of hash functions are

- **Fixed Length Output (Hash Value)**
    - Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as **hashing the data**.
    - In general, the hash is much smaller than the input data, hence hash functions are sometimes called **compression functions**.

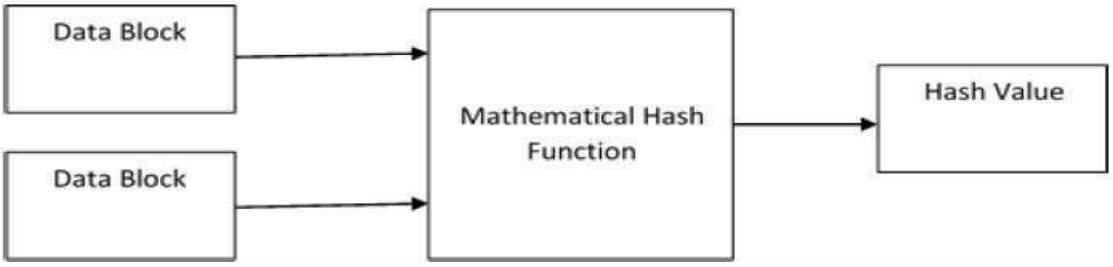
- Since a hash is a smaller representation of a larger data, it is also referred to as a **digest**.
- Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.

- **Efficiency of Operation**
  - Generally for any hash function h with input x, computation of h(x) is a fast operation.
  - Computationally hash functions are much faster than a symmetric encryption.

Design of Hashing Algorithms

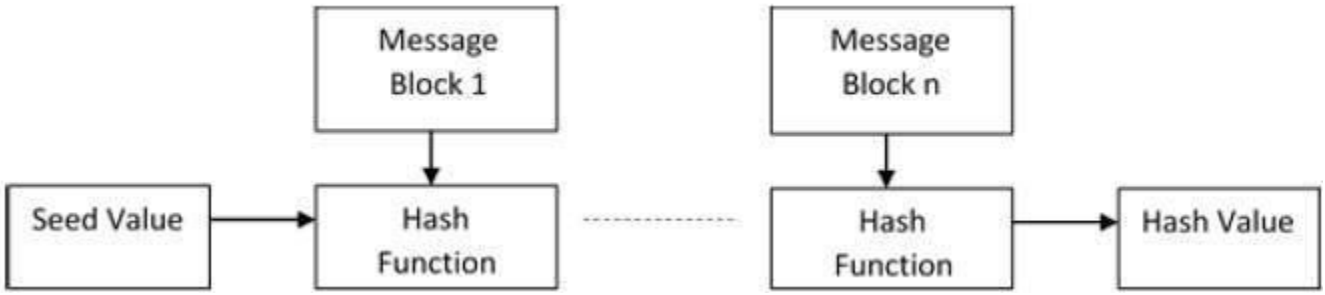
At the heart of a hashing is a mathematical function that operates on two fixed-size blocks of data to create a hash code. This hash function forms the part of the hashing algorithm.

The size of each data block varies depending on the algorithm. Typically the block sizes are from 128 bits to 512 bits. The following illustration demonstrates hash function −



Hashing algorithm involves rounds of above hash function like a block cipher. Each round takes an input of a fixed size, typically a combination of the most recent message block and the output of the last round.

This process is repeated for as many rounds as are required to hash the entire message. Schematic of hashing algorithm is depicted in the following illustration −



Since, the hash value of first message block becomes an input to the second hash operation, output of which alters the result of the third operation, and so on. This effect, known as an **avalanche** effect of hashing. Avalanche effect results in substantially different hash values for two messages that differ by even a single bit of data. Understand the difference between hash

function and algorithm correctly. The hash function generates a hash code by operating on two blocks of fixed-length binary data. Hashing algorithm is a process for using the hash function, specifying how the message will be broken up and how the results from previous message blocks are chained together.

Secure Hash Function (SHA)

Family of SHA comprise of four SHA algorithms; SHA-0, SHA-1, SHA-2, and SHA-3. Though from same family, there are structurally different.

- The original version is SHA-0, a 160-bit hash function, was published by the National Institute of Standards and Technology (NIST) in 1993. It had few weaknesses and did not become very popular. Later in 1995, SHA-1 was designed to correct alleged weaknesses of SHA-0.
- SHA-1 is the most widely used of the existing SHA hash functions. It is employed in several widely used applications and protocols including Secure Socket Layer (SSL) security.
- In 2005, a method was found for uncovering collisions for SHA-1 within practical time frame making long-term employability of SHA-1 doubtful.
- SHA-2 family has four further SHA variants, SHA-224, SHA-256, SHA-384, and SHA-512 depending up on number of bits in their hash value. No successful attacks have yet been reported on SHA-2 hash function.
- Though SHA-2 is a strong hash function. Though significantly different, its basic design is still follows design of SHA-1. Hence, NIST called for new competitive hash function designs.

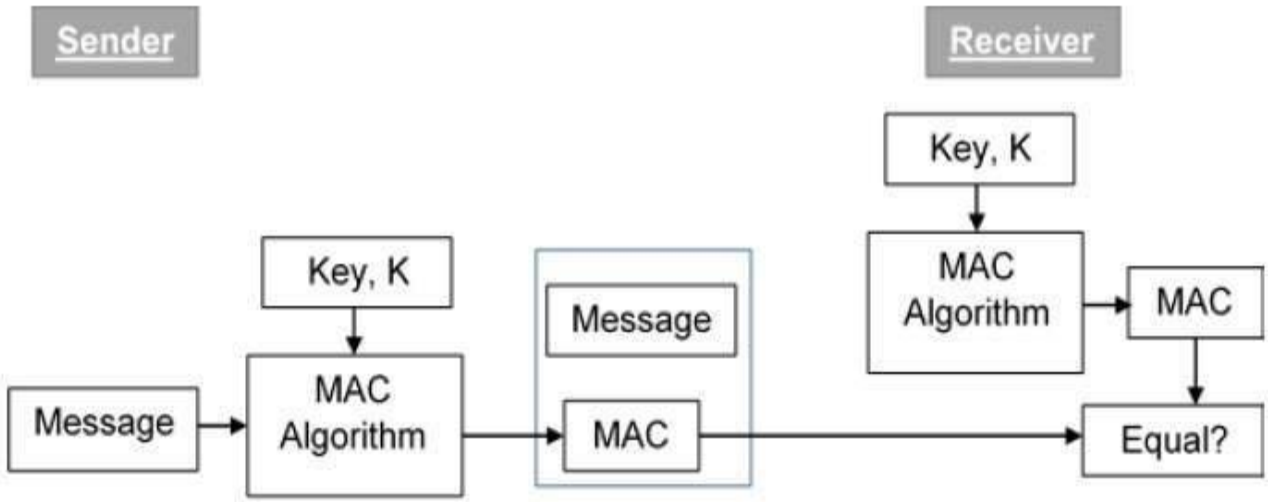
- In October 2012, the NIST chose the Keccak algorithm as the new SHA-3 standard. Keccak offers many benefits, such as efficient performance and good resistance for attacks.

Message Authentication Code (MAC)

MAC algorithm is a symmetric key cryptographic technique to provide message authentication. For establishing MAC process, the sender and receiver share a symmetric key K.

Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

The process of using MAC for authentication is depicted in the following illustration −



Let us now try to understand the entire process in detail −

- The sender uses some publicly known MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, MAC function also compresses an arbitrary long input into a fixed length output. The major difference between hash and MAC is that MAC uses secret key during the compression.
- The sender forwards the message along with the MAC. Here, we assume that the message is sent in the clear, as we are concerned of providing message origin authentication, not confidentiality. If confidentiality is required then the message needs encryption.
- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.
- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.
- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified. As a bottom-line, a receiver safely assumes that the message is not the genuine.

Limitations of MAC

There are two major limitations of MAC, both due to its symmetric nature of operation −

- **Establishment of Shared Secret.**
    - It can provide message authentication among pre-decided legitimate users who have shared key.
    - This requires establishment of shared secret prior to use of MAC.
- **Inability to Provide Non-Repudiation**

- Non-repudiation is the assurance that a message originator cannot deny any previously sent messages and commitments or actions.
- MAC technique does not provide a non-repudiation service. If the sender and receiver get involved in a dispute over message origination, MACs cannot provide a proof that a message was indeed sent by the sender.
- Though no third party can compute the MAC, still sender could deny having sent the message and claim that the receiver forged it, as it is impossible to determine which of the two parties computed the MAC.

Both these limitations can be overcome by using the public key based digital signatures discussed in following section.

Digital signatures are the public-key primitives of message authentication. In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message.

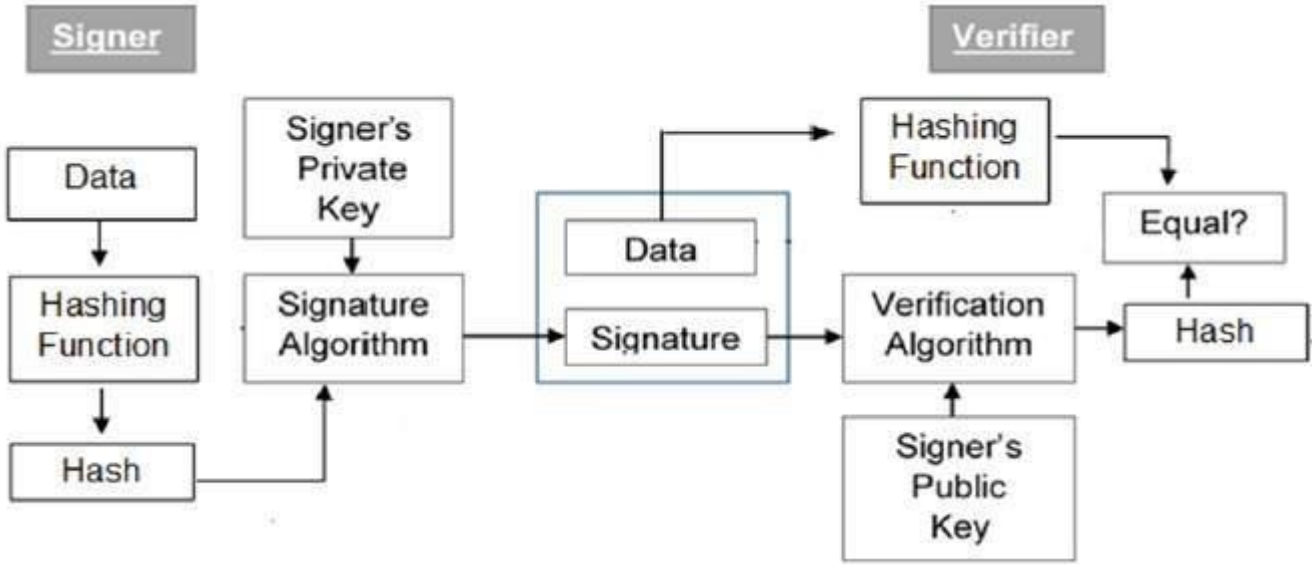
Similarly, a digital signature is a technique that binds a person/entity to the digital data. This binding can be independently verified by receiver as well as any third party.

Digital signature is a cryptographic value that is calculated from the data and a secret key known only by the signer.

In real world, the receiver of message needs assurance that the message belongs to the sender and he should not be able to repudiate the origination of that message. This requirement is very crucial in business applications, since likelihood of a dispute over exchanged data is very high.

DIGITAL SIGNATURE

The digital signature scheme is based on public key cryptography. The model of digital signature scheme is depicted in the following illustration −



The following points explain the entire process in detail −

- Each person adopting this scheme has a public-private key pair.

- Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as the signature key and the public key as the verification key.
- Signer feeds data to the hash function and generates hash of data.
- Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier.
- Verifier feeds the digital signature and the verification key into the verification algorithm. The verification algorithm gives some value as output.
- Verifier also runs same hash function on received data to generate hash value.
- For verification, this hash value and output of verification algorithm are compared. Based on the comparison result, verifier decides whether the digital signature is valid.
- Since digital signature is created by 'private' key of signer and no one else can have this key; the signer cannot repudiate signing the data in future.

It should be noticed that instead of signing data directly by signing algorithm, usually a hash of data is created. Since the hash of data is a unique representation of data, it is sufficient to sign the hash in place of data. The most important reason of using hash instead of data directly for signing is efficiency of the scheme.

Let us assume RSA is used as the signing algorithm. As discussed in public key encryption chapter, the encryption/signing process using RSA involves modular exponentiation.

Signing large data through modular exponentiation is computationally expensive and time consuming. The hash of the data is a relatively small digest of the data, hence **signing a hash is more efficient than signing the entire data**.

Importance of Digital Signature

Out of all cryptographic primitives, the digital signature using public key cryptography is considered as very important and useful tool to achieve information security. Apart from ability to provide non-repudiation of message, the digital signature also provides message authentication and data integrity. Let us briefly see how this is achieved by the digital signature

- **Message authentication** − When the verifier validates the digital signature using public key of a sender, he is assured that signature has been created only by sender who possess the corresponding secret private key and no one else.
- **Data Integrity** − In case an attacker has access to the data and modifies it, the digital signature verification at receiver end fails. The hash of modified data and the output provided by the verification algorithm will not match. Hence, receiver can safely deny the message assuming that data integrity has been breached.
- **Non-repudiation** − Since it is assumed that only the signer has the knowledge of the signature key, he can only create unique signature on a given data. Thus the receiver can present data and the digital signature to a third party as evidence if any dispute arises in the future.

HMAC Algorithm

Notations

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
IV = initial value input to hash function
M = message input to HMAC(including the padding specified in the embedded hash function)
Yi = *i*th block of *M*, 0 *i* (*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 recommended length is *n*; if key length is greater than *b*; the key is input to the hash function to produce an *n*-bit key
K+ = *K* padded with zeros on the left so that the result is *b* bits in length
ipad = 00110110 (36 in hexadecimal) repeated *b*/8 times

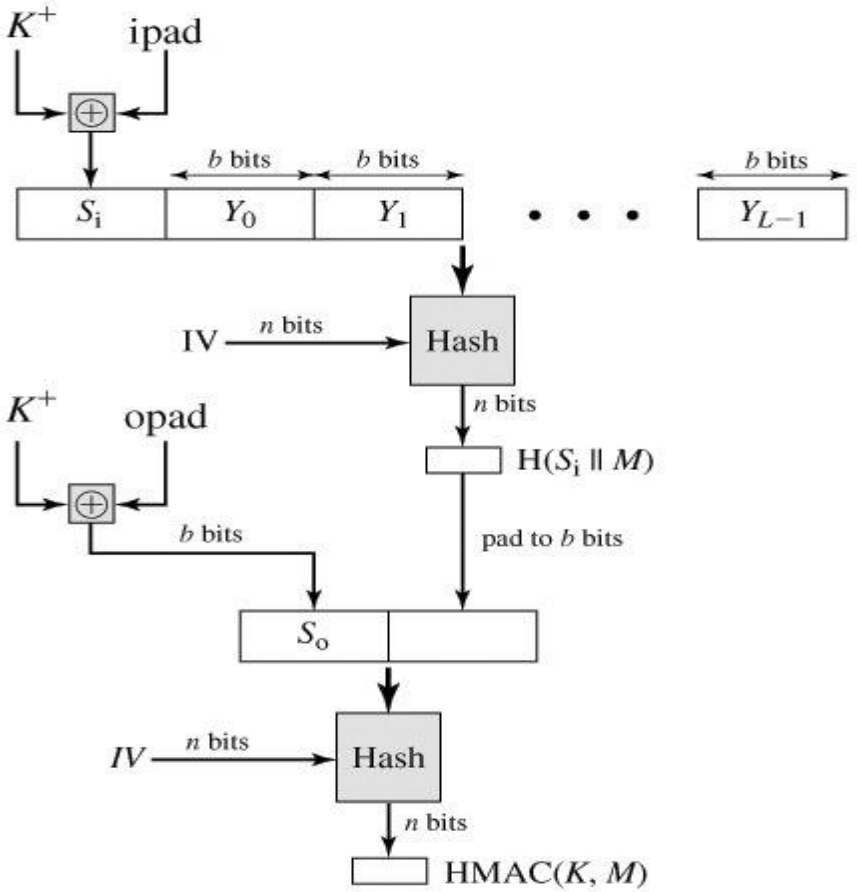
opad = 01011100 (5C in hexadecimal) repeated *b*/8 times

Then HMAC can be expressed as follows:

HMAC(*K*,*M*) = H[(*K*+ opad)||H[(*K*+ ipad)||*M*]]

In words,

1. Append zeros to the left end of *K* to create a *b*-bit string *K*+(e.g., if *K* is of length 160 bits and *b*= 512 then *K* will be appended with 44 zero bytes 0 x 00).



2. XOR (bitwise exclusive-OR) *K*+ with ipad to produce the *b*-bit block *S*i.
3. Append *M* to *S*i.
4. Apply H to the stream generated in step 3.
5.XOR *K*+ with opad to produce the b-bit block *S*o
6.Append the hash result from step 4 to *S*o
7.Apply H to the stream generated in step 6 and output the result.

Authentication Protocols

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

- Authentication Protocols(ap1.o) - Simple authentication Protocol.
- Authentication Protocols(ap2.o) - Network address(IP address)
- Authentication Protocols(ap3.o) - Secret password
- Authentication Protocols(ap3.1) - Encrypted Secret password
- Authentication Protocols(ap4..o) – Nonce(Different password eachtime)
- Authentication Protocols(ap5.o) – Nonce(Asymmetric key encryption.

Unit-III

Program Security: Secure Programs – NonMalicious Program Errors – Viruses and Others Malicious Code – Targeted Malicious Code – Control Against Threats.

NONMALICIOUS PROGRAM ERRORS

Buffer-Overflow:

- This is a stack based buffer overflow, also known as smashing the stack. Stack smashing has been called the attack of the decade for the 1990s.
- Assume a Web form that asks the user to enter data, such as name, age and date of birth. The entered information is then sent to a server and the server writes the data entered to a buffer that can hold N characters.
- If the server software does not verify that the length of the data is at most N characters, then a buffer overflow might occur. It might seem that a buffer overflow may cause less harm but it is not the case.
- It is likely that any overflowing data will overwrite something important and cause the computer to crash. If so then attacker Trudy might be able to use this flaw to launch a denial of service (DoS) attack.
- The problem can be explained using a software that is used for authentication. The authentication decision resides in a single bit. If a buffer overflow overwrites this authentication bit, then Trudy can authenticate herself as Alice the actual user.
- This is shown in the figure 1 below where the "F" in the position of the boolean flag indicates failed authentication.

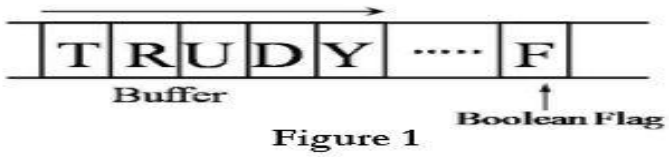


Figure 1

- If a buffer overflow overwrites the memory position where the boolean flag is stored, Trudy can overwrite "F" with "T" and the software will believe that Trudy has been authenticated. This attack is shown in figure 2.

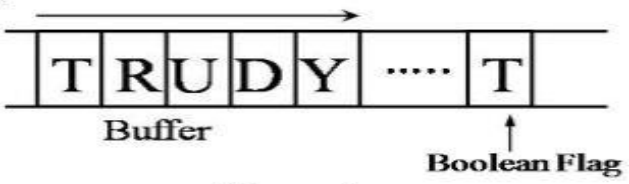


Figure 2

- Example: Consider a C source code given below-

```
int main()
{
int buffer [10];
buffer [20] =37;
}
```

- When this code is executed, a buffer overflow occurs. The impact of this buffer overflow depends on what resides in memory at the location corresponding to buffer [20].
- The buffer overflow might overwrite user data or code, or it could overwrite system data or code, or it might overwrite unused space.

Incomplete mediation:

- In C program a function strcpy(buffer, input) copies the contents of the input string input to the array buffer. A buffer overflow will occur if the length of input is greater than the length of buffer.
- To prevent such a buffer overflow, the program validates the input by checking the length of input before attempting to write it to buffer. Failure to do so is an example of incomplete mediation.

Race-Condition:

- Race conditions arise when a security-critical process occurs in stages instead of "all at once".
- In such cases, an attacker may be able to make a change between the stages and thereby break the security.
- The term race condition refers to a race between the attacker and the next stage of the process, although it is not so much a race as a matter of careful timing for the attacker.
- Example: The race condition considered here can be implemented by the Unix command mkdir, which creates a new directory.
- The creation of the directory is done in stages where a stage that determines authorization is followed by a stage that transfers ownership.
- If Trudy can make a change after the authorization stage but before the transfer of ownership, then she can become the owner of some directory that she should not be able to access.
- The working of mkdir is shown in the below figure 3.Here mkdir is not atomic and that is the source of the race condition.

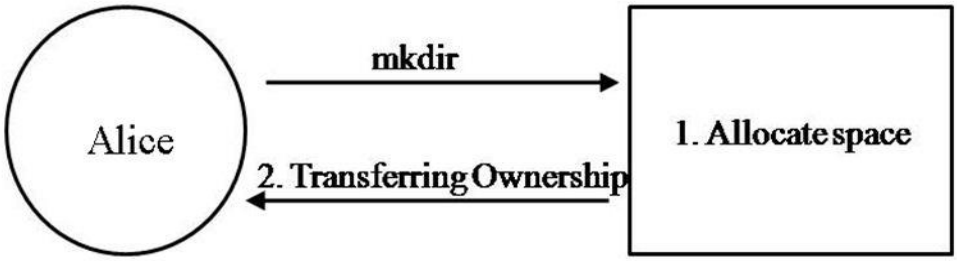


Figure 3

- Trudy can exploit this particular mkdir race condition if she can somehow implement the attack that is shown in figure 4.

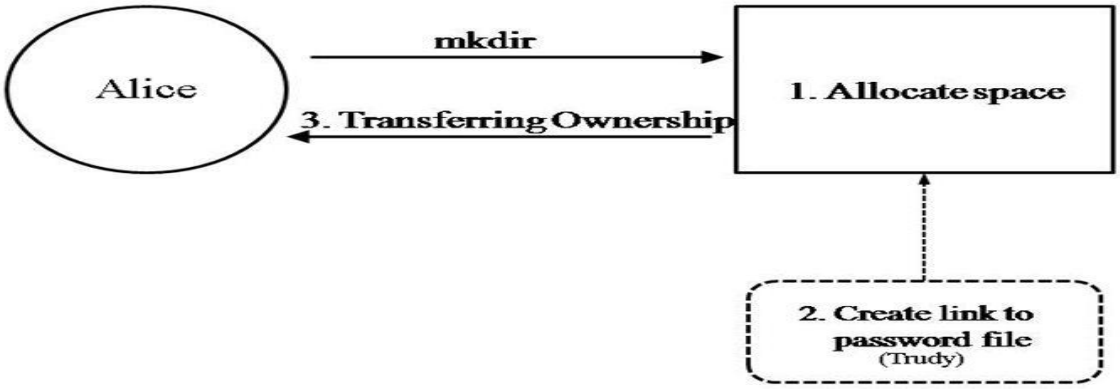


Figure 4

- In the attack above, after the space for the new directory is allocated to Trudy, a link is established from the password file.
- Trudy is not authorized to access to this newly created space, before ownership of the new directory is transferred to Trudy.
- This attack is not really a race, but instead it requires careful timing by Trudy.
- Race conditions are probably fairly common and with the trend towards increased parallelism they can become even more prevalent.
- Each race condition is unique, so there is no standard formula for such an attack while they are certainly more difficult to exploit.
- Real-world attacks based on race conditions require careful timing which makes it more difficult to exploit than buffer overflow.
- Race conditions can be prevented by making the security-critical processes atomic, which is easy to say but difficult to implement

TARGETED MALICIOUS CODE

So far, we have looked at anonymous code written to affect users and machines indiscriminately. Another class of malicious code is written for a particular system, for a particular application, and for a particular purpose. Many of the virus writers' techniques apply, but there are also some new ones.

Trapdoors

A trapdoor is an undocumented entry point to a module. Developers insert trapdoors during code development, perhaps to test the module, to provide "hooks" by which to connect future modifications or enhancements, or to allow access if the module should fail in the future. In addition to these legitimate uses, trapdoors can allow a programmer access to a program once it is placed in production.

Examples of Trapdoors

Because computing systems are complex structures, programmers usually develop and test systems in a methodical, organized, modular manner, taking advantage of the way the

system is composed of modules or components. Often, programmers first test each small component of the system separate from the other components, in a step called **unit testing**, to ensure that the component works correctly by itself. Then, developers test components together during **integration testing**, to see how they function as they send messages and data from one to the other. Rather than paste all the components together in a "big bang" approach, the testers group logical clusters of a few components, and each cluster is tested in a way that allows testers to control and understand what might make a component or its interface fail.

To test a component on its own, the developer or tester cannot use the surrounding routines that prepare input or work with output. Instead, it is usually necessary to write "stubs" and "drivers," simple routines to inject data in and extract results from the component being tested. As testing continues, these stubs and drivers are discarded because they are replaced by the actual components whose functions they mimic.

Poor error checking

It is another source of trapdoors. A good developer will design a system so that any data value is checked before it is used; the checking involves making sure the data type is correct as well as ensuring that the value is within acceptable bounds. But in some poorly designed systems, unacceptable input may not be caught and can be passed on for use in unanticipated ways. For example, a component's code may check for one of three expected sequences; finding none of the three, it should recognize an error. Suppose the developer uses a CASE statement to look for each of the three possibilities. A careless programmer may allow a failure simply to fall through the CASE without being flagged as an error. The fingerd flaw exploited by the Morris worm occurs exactly that way: A C library I/O routine fails to check whether characters are left in the input buffer before returning a pointer to a supposed next character.

Hardware processor design provides another common example of this kind of security flaw. Here, it often happens that not all possible binary opcode values have matching machine instructions. The undefined opcodes sometimes implement peculiar instructions, either because of an intent to test the processor design or because of an oversight by the processor designer. Undefined opcodes are the hardware counterpart of poor error checking for software.

As with viruses, trapdoors are not always bad. They can be very useful in finding security flaws. Auditors sometimes request trapdoors in production programs to insert fictitious but identifiable transactions into the system. Then, the auditors trace the flow of these transactions through the system. However, trapdoors must be documented, access to them should be strongly controlled, and they must be designed and used with full understanding of the potential consequences.

VIRUSES AND OTHER MALICIOUS CODE

By themselves, programs are seldom security threats. The programs operate on data, taking action only when data and state changes trigger it. Much of the work done by a program is invisible to users, so they are not likely to be aware of any malicious activity. For instance, when was the last time you saw a bit? Do you know in what form a document file is stored? If you

know a document resides somewhere on a disk, can you find it? Can you tell if a game program does anything in addition to its expected interaction with you? Which files are modified by a word processor when you create a document? Most users cannot answer these questions. However, since computer data are not usually seen directly by users, malicious people can make programs serve as vehicles to access and change data and other programs. Let us look at the possible effects of malicious code and then examine in detail several kinds of programs that can be used for interception or modification of data.

Why Worry About Malicious Code?

None of us likes the unexpected, especially in our programs. Malicious code behaves in unexpected ways, thanks to a malicious programmer's intention. We think of the malicious code as lurking inside our system: all or some of a program that we are running or even a nasty part of a separate program that somehow attaches itself to another (good) program.

Malicious Code Can Do Much (Harm)

Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file. Or malicious code can do nothing at all right now; it can be planted to lie dormant, undetected, until some event triggers the code to act. The trigger can be a time or date, an interval (for example, after 30 minutes), an event (for example, when a particular program is executed), a condition (for example, when communication occurs on a modem), a count (for example, the fifth time something happens), some combination of these, or a random situation. In fact, malicious code can do different things each time, or nothing most of the time with something dramatic on occasion. In general, malicious code can act with all the predictability of a two-year-old child: We know in general what two-year-olds do, we may even know what a specific two-year-old often does in certain situations, but two-year-olds have an amazing capacity to do the unexpected.

Malicious code runs under the user's authority. Thus, malicious code can touch everything the user can touch, and in the same ways. Users typically have complete control over their own program code and data files; they can read, write, modify, append, and even delete them. And well they should. But malicious code can do the same, without the user's permission or even knowledge.

Kinds of Malicious Code

Malicious code or a **rogue program** is the general name for unanticipated or undesired effects in programs or program parts, caused by an agent intent on damage. This definition eliminates unintentional errors, although they can also have a serious negative effect. This definition also excludes coincidence, in which two benign programs combine for a negative effect. The **agent** is the writer of the program or the person who causes its distribution. By this definition, most faults found in software inspections, reviews, and testing do not qualify as malicious code, because we think of them as unintentional. However, keep in mind as you read

this chapter that unintentional faults can in fact invoke the same responses as intentional malevolence; a benign cause can still lead to a disastrous effect.

You are likely to have been affected by a virus at one time or another, either because your computer was infected by one or because you could not access an infected system while its administrators were cleaning up the mess one made. In fact, your virus might actually have been a worm: The terminology of malicious code is sometimes used imprecisely. A **virus** is a program that can pass on malicious code to other nonmalicious programs by modifying them. The term "virus" was coined because the affected program acts like a biological virus: It infects other healthy subjects by attaching itself to the program and either destroying it or coexisting with it. Because viruses are insidious, we cannot assume that a clean program yesterday is still clean today. Moreover, a good program can be modified to include a copy of the virus program, so the infected good program itself begins to act as a virus, infecting other programs. The infection usually spreads at a geometric rate, eventually overtaking an entire computing system and spreading to all other connected systems.

A virus can be either transient or resident. A **transient** virus has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends. (During its execution, the transient virus may have spread its infection to other programs.) A **resident** virus locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.

A **Trojan horse** is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect.[1] As an example of a computer Trojan horse,

A **logic bomb** is a class of malicious code that "detonates" or goes off when a specified condition occurs. A **time bomb** is a logic bomb whose trigger is a time or date.

A **trapdoor** or **backdoor** is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges. For instance, an automated bank teller program might allow anyone entering the number 990099 on the keypad to process the log of everyone's transactions at that machine. In this example, the trapdoor could be intentional, for maintenance purposes, or it could be an illicit way for the implementer to wipe out any record of a crime.

A **worm** is a program that spreads copies of itself through a network. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files). Additionally, the worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.

Types of Malicious Code.

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies

	of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs
Time bomb	Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resource

Because "virus" is the popular name given to all forms of malicious code and because fuzzy lines exist between different kinds of malicious code, we will not be too restrictive in the following discussion. We want to look at how malicious code spreads, how it is activated, and what effect it can have. A virus is a convenient term for mobile malicious code, and so in the following sections we use the term "virus" almost exclusively. The points made apply also to other forms of malicious code.

Homes for Viruses

The virus writer may find these qualities appealing in a virus:

- It is hard to detect.
- It is not easily destroyed or deactivated.
- It spreads infection widely.
- It can reinfect its home program or other programs.
- It is easy to create.
- It is machine independent and operating system independent.

One-Time Execution

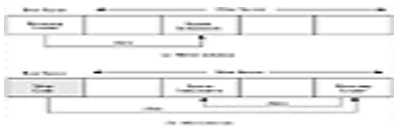
The majority of viruses today execute only once, spreading their infection and causing their effect in that one execution. A virus often arrives as an e-mail attachment of a document virus. It is executed just by being opened.

Boot Sector Viruses

A special case of virus attachment, but formerly a fairly popular one, is the so-called **boot sector virus.** When a computer is started, control begins with firmware that determines which hardware components are present, tests them, and transfers control to an operating system. A given hardware platform can run many different operating systems, so the operating system is not coded in firmware but is instead invoked dynamically, perhaps even by a user's choice, after the hardware test.

The operating system is software stored on disk. Code copies the operating system from disk to memory and transfers control to it; this copying is called the **bootstrap** (often **boot**) load because the operating system figuratively pulls itself into memory by its bootstraps. The firmware does its control transfer by reading a fixed number of bytes from a fixed location on the disk (called the **boot sector**) to a fixed address in memory and then jumping to that address (which will turn out to contain the first instruction of the bootstrap loader). The bootstrap loader then reads into memory the rest of the operating system from disk. To run a different operating system, the user just inserts a disk with the new operating system and a bootstrap loader. When the user reboots from this new disk, the loader there brings in and runs another operating system. This same scheme is used for personal computers, workstations, and large mainframes.

To allow for change, expansion, and uncertainty, hardware designers reserve a large amount of space for the bootstrap load. The boot sector on a PC is slightly less than 512 bytes, but since the loader will be larger than that, the hardware designers support "chaining," in which each block of the bootstrap is chained to (contains the disk location of) the next block. This chaining allows big bootstraps but also simplifies the installation of a virus. The virus writer simply breaks the chain at any point, inserts a pointer to the virus code to be executed, and reconnects the chain after the virus has been installed.



Boot Sector Virus Relocating Code.

The boot sector is an especially appealing place to house a virus. The virus gains control very early in the boot process, before most detection tools are active, so that it can avoid, or at least complicate, detection. The files in the boot area are crucial parts of the operating system. Consequently, to keep users from accidentally modifying or deleting them with disastrous results, the operating system makes them "invisible" by not showing them as part of a normal listing of stored files, preventing their deletion. Thus, the virus code is not readily noticed by users.

Memory-Resident Viruses

Some parts of the operating system and most user programs execute, terminate, and disappear, with their space in memory being available for anything executed later. For very

frequently used parts of the operating system and for a few specialized user programs, it would take too long to reload the program each time it was needed. Such code remains in memory and is called "resident" code. Examples of resident code are the routine that interprets keys pressed on the keyboard, the code that handles error conditions that arise during a program's execution, or a program that acts like an alarm clock, sounding a signal at a time the user determines. Resident routines are sometimes called TSRs or "terminate and stay resident" routines.

Virus writers also like to attach viruses to resident code because the resident code is activated many times while the machine is running. Each time the resident code runs, the virus does too. Once activated, the virus can look for and infect uninfected carriers. For example, after activation, a boot sector virus might attach itself to a piece of resident code. Then, each time the virus was activated it might check whether any removable disk in a disk drive was infected and, if not, infect it. In this way the virus could spread its infection to all removable disks used during the computing session.

Virus Effects and Causes.

Virus Effect	**How It Is Caused**
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory handler address table	• Intercept interrupt by modifying interrupt • Load self in nontransient memory area
Infect disks	• Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self falsify result	• Intercept system calls that would reveal self and • Classify self as "hidden" file

Spread infection	• Infect boot sector • Infect systems program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation deactivation	• Activate before deactivating program and block • Store copy to reinfect after deactivation

Transmission Patterns

A virus is effective only if it has some means of transmission from one location to another. As we have already seen, viruses can travel during the boot process, by attaching to an executable file or traveling within data files. The travel itself occurs during execution of an already infected program. Since a virus can execute any instructions a program can, virus travel is not confined to any single medium or execution pattern. For example, a virus can arrive on a diskette or from a network connection, travel during its host's execution to a hard disk boot sector, reemerge next time the host computer is booted, and remain in memory to infect other diskettes as they are accessed.

Polymorphic Viruses

The virus signature may be the most reliable way for a virus scanner to identify a virus. If a particular virus always begins with the string 47F0F00E08 (in hexadecimal) and has string 00113FFF located at word 12, it is unlikely that other programs or data files will have these exact characteristics. For longer signatures, the probability of a correct match increases.

If the virus scanner will always look for those strings, then the clever virus writer can cause something other than those strings to be in those positions. For example, the virus could have two alternative but equivalent beginning words; after being installed, the virus will choose one of the two words for its initial word. Then, a virus scanner would have to look for both patterns. A virus that can change its appearance is called a **polymorphic virus.** (*Poly* means "many" and *morph* means "form".) A two-form polymorphic virus can be handled easily as two independent viruses. Therefore, the virus writer intent on preventing detection of the virus will want either a large or an unlimited number of forms so that the number of possible forms is too large for a virus scanner to search for. Simply embedding a random number or string at a fixed place in the executable version of a virus is not sufficient, because the signature of the virus is just the constant code excluding the random part. A polymorphic virus has to randomly reposition all parts of itself and randomly change all fixed data. Thus, instead of containing the fixed (and therefore searchable) string "HA! INFECTED BY A VIRUS," a polymorphic virus has to change even that pattern sometimes.

Trivially, assume a virus writer has 100 bytes of code and 50 bytes of data. To make two virus instances different, the writer might distribute the first version as 100 bytes of code followed by all 50 bytes of data. A second version could be 99 bytes of code, a jump instruction, 50 bytes of data, and the last byte of code. Other versions are 98 code bytes jumping to the last two, 97 and three, and so forth. Just by moving pieces around the virus writer can create enough different appearances to fool simple virus scanners. Once the scanner writers became aware of these kinds of tricks, however, they refined their signature definitions.

A more sophisticated polymorphic virus randomly intersperses harmless instructions throughout its code. Examples of harmless instructions include addition of zero to a number, movement of a data value to its own location, or a jump to the next instruction. These "extra" instructions make it more difficult to locate an invariant signature.

A simple variety of polymorphic virus uses encryption under various keys to make the stored form of the virus different. These are sometimes called **encrypting** viruses. This type of virus must contain three distinct parts: a decryption key, the (encrypted) object code of the virus, and the (unencrypted) object code of the decryption routine. For these viruses, the decryption routine itself or a call to a decryption library routine must be in the clear, and so that becomes the signature.

To avoid detection, not every copy of a polymorphic virus has to differ from every other copy. If the virus changes occasionally, not every copy will match a signature of every other copy.

The Source of Viruses

Since a virus can be rather small, its code can be "hidden" inside other larger and more complicated programs. Two hundred lines of a virus could be separated into one hundred packets of two lines of code and a jump each; these one hundred packets could be easily hidden inside a compiler, a database manager, a file manager, or some other large utility.

Virus discovery could be aided by a procedure to determine if two programs are equivalent. However, theoretical results in computing are very discouraging when it comes to the complexity of the equivalence problem. The general question, "are these two programs equivalent?" is undecidable (although that question *can* be answered for many specific pairs of programs). Even ignoring the general undecidability problem, two modules may produce subtly different results that may—or may not—be security relevant. One may run faster, or the first may use a temporary file for work space whereas the second performs all its computations in memory. These differences could be benign, or they could be a marker of an infection. Therefore, we are unlikely to develop a screening program that can separate infected modules from uninfected ones.

Although the general is dismaying, the particular is not. If we know that a particular virus may infect a computing system, we can check for it and detect it if it is there. Having found the virus, however, we are left with the task of cleansing the system of it. Removing the virus in a running system requires being able to detect and eliminate its instances faster than it can spread.

Prevention of Virus Infection

The only way to prevent the infection of a virus is not to share executable code with an infected source. This philosophy used to be easy to follow because it was easy to tell if a file was executable or not. For example, on PCs, a *.exe* extension was a clear sign that the file was executable. However, as we have noted, today's files are more complex, and a seemingly nonexecutable file may have some executable code buried deep within it. For example, a word processor may have commands within the document file; as we noted earlier, these commands, called macros, make it easy for the user to do complex or repetitive things. But they are really executable code embedded in the context of the document. Similarly, spreadsheets, presentation slides, and other office- or business-related files can contain code or scripts that can be executed in various ways—and thereby harbor viruses. And, as we have seen, the applications that run or use these files may try to be helpful by automatically invoking the executable code, whether you want it run or not! Against the principles of good security, e-mail handlers can be set to automatically open (without performing access control) attachments or embedded code for the recipient, so your e-mail message can have animated bears dancing across the top.

Another approach virus writers have used is a little-known feature in the Microsoft file design. Although a file with a *.doc* extension is expected to be a Word document, in fact, the true document type is hidden in a field at the start of the file. This convenience ostensibly helps a user who inadvertently names a Word document with a *.ppt* (Power-Point) or any other extension. In some cases, the operating system will try to open the associated application but, if that fails, the system will switch to the application of the hidden file type. So, the virus writer creates an executable file, names it with an inappropriate extension, and sends it to the victim, describing it is as a picture or a necessary code add-in or something else desirable. The unwitting recipient opens the file and, without intending to, executes the malicious code.

More recently, executable code has been hidden in files containing large data sets, such as pictures or read-only documents. These bits of viral code are not easily detected by virus scanners and certainly not by the human eye. For example, a file containing a photograph may be highly granular; if every sixteenth bit is part of a command string that can be executed, then the virus is very difficult to detect.

Since you cannot always know which sources are infected, you should assume that any outside source is infected. Fortunately, you know when you are receiving code from an outside source; unfortunately, it is not feasible to cut off all contact with the outside world.

- *Use only commercial software acquired from reliable, well-established vendors.*
- *Test all new software on an isolated computer.*
- *Open attachments only when you know them to be safe.*
- *Make a recoverable system image and store it safely*
- *Make and retain backup copies of executable system files.*
- *Use virus detectors (often called virus scanners) regularly and update them daily.*

CONTROLS AGAINST PROGRAM THREATS

The picture we have just described is not pretty. There are many ways a program can fail and many ways to turn the underlying faults into security failures. It is of course better to focus on prevention than cure; how do we use controls during **software development**—the specifying, designing, writing, and testing of the program—to find and eliminate the sorts of exposures we have discussed? The discipline of software engineering addresses this question more globally, devising approaches to ensure the quality of software. In this book, we provide an overview of several techniques that can prove useful in finding and fixing security flaws. Three types of controls: developmental, operating system, and administrative. We discuss each in turn.

Developmental Controls

Many controls can be applied during software development to ferret out and fix problems. So let us begin by looking at the nature of development itself, to see what tasks are involved in specifying, designing, building, and testing software.

The Nature of Software Development

Software development is often considered a solitary effort; a programmer sits with a specification or design and grinds out line after line of code. But in fact, software development is a collaborative effort, involving people with different skill sets who combine their expertise to produce a working product. Development requires people who can

- *specify* the system, by capturing the requirements and building a model of how the system should work from the users' point of view
- *design* the system, by proposing a solution to the problem described by the requirements and building a model of the solution
- *implement* the system, by using the design as a blueprint for building a working solution
- *test* the system, to ensure that it meets the requirements and implements the solution as called for in the design
- *review* the system at various stages, to make sure that the end products are consistent with the specification and design models
- *document* the system, so that users can be trained and supported
- *manage* the system, to estimate what resources will be needed for development and to track when the system will be done

- *maintain* the system, tracking problems found, changes needed, and changes made, and evaluating their effects on overall quality and functionality

One person could do all these things. But more often than not, a team of developers works together to perform these tasks. Sometimes a team member does more than one activity; a tester can take part in a requirements review, for example, or an implementer can write documentation. Each team is different, and team dynamics play a large role in the team's success.

We can examine both product and process to see how each contributes to quality and in particular to security as an aspect of quality. Let us begin with the product, to get a sense of how we recognize high-quality secure software.

Modularity, Encapsulation, and Information Hiding

Code usually has a long shelf-life, and it is enhanced over time as needs change and faults are found and fixed. For this reason, a key principle of software engineering is to create a design or code in small, self-contained units, called **components** or **modules**; when a system is written this way, we say that it is **modular**. Modularity offers advantages for program development in general and security in particular.

If a component is isolated from the effects of other components, then it is easier to trace a problem to the fault that caused it and to limit the damage the fault causes. It is also easier to maintain the system, since changes to an isolated component do not affect other components. And it is easier to see where vulnerabilities may lie if the component is isolated. We call this isolation **encapsulation**.

Information hiding is another characteristic of modular software. When information is hidden, each component hides its precise implementation or some other design decision from the others. Thus, when a change is needed, the overall design can remain intact while only the necessary changes are made to particular components.

Modularity

Modularization is the process of dividing a task into subtasks. This division is done on a logical or functional basis. Each component performs a separate, independent part of the task. The goal is to have each component meet four conditions:

- *single-purpose*: performs one function
- *small*: consists of an amount of information for which a human can readily grasp both structure and content
- *simple*: is of a low degree of complexity so that a human can readily understand the purpose and structure of the module

- *independent*: performs a task isolated from other modules

Often, other characteristics, such as having a single input and single output or using a limited set of programming constructs, help a component be modular. From a security standpoint, modularity should improve the likelihood that an implementation is correct.

In particular, smallness is an important quality that can help security analysts understand what each component does. That is, in good software, design and program units should be only as large as needed to perform their required functions. There are several advantages to having small, independent components.

- *Maintenance.* If a component implements a single function, it can be replaced easily with a revised one if necessary. The new component may be needed because of a change in requirements, hardware, or environment. Sometimes the replacement is an enhancement, using a smaller, faster, more correct, or otherwise better module. The interfaces between this component and the remainder of the design or code are few and well described, so the effects of the replacement are evident.

- *Understandability.* A system composed of many small components is usually easier to comprehend than one large, unstructured block of code.

- *Reuse.* Components developed for one purpose can often be reused in other systems. Reuse of correct, existing design or code components can significantly reduce the difficulty of implementation and testing.

- *Correctness.* A failure can be quickly traced to its cause if the components perform only one task each.

- *Testing.* A single component with well-defined inputs, output, and function can be tested exhaustively by itself, without concern for its effects on other modules (other than the expected function and output, of course).

Security analysts must be able to understand each component as an independent unit and be assured of its limited effect on other components.

A modular component usually has high cohesion and low coupling. By **cohesion**, we mean that all the elements of a component have a logical and functional reason for being there; every aspect of the component is tied to the component's single purpose. A highly cohesive component has a high degree of focus on the purpose; a low degree of cohesion means that the component's contents are an unrelated jumble of actions, often put together because of time-dependencies or convenience.

Coupling refers to the degree with which a component depends on other components in the system. Thus, low or loose coupling is better than high or tight coupling, because the loosely coupled components are free from unwitting interference from other components.

Peer Reviews

We turn next to the process of developing software. Certain practices and techniques can assist us in finding real and potential security flaws (as well as other faults) and fixing them before the system is turned over to the users. Of the many practices available for building what they call "solid software," Pfleeger et al. recommend several key techniques: [PFL01a]

- peer reviews
- hazard analysis
- testing
- good design
- prediction
- static analysis
- configuration management
- analysis of mistakes

Here, we look at each practice briefly, and we describe its relevance to security controls. We begin with peer reviews.

You have probably been doing some form of review for as many years as you have been writing code: desk-checking your work or asking a colleague to look over a routine to ferret out any problems. Today, a software review is associated with several formal process steps to make it more effective, and we review any artifact of the development process, not just code. But the essence of a review remains the same: sharing a product with colleagues able to comment about its correctness. There are careful distinctions among three types of peer reviews:

- *Review:* The artifact is presented informally to a team of reviewers; the goal is consensus and buy-in before development proceeds further.

- *Walk-through:* The artifact is presented to the team by its creator, who leads and controls the discussion. Here, education is the goal, and the focus is on learning about a single document.

- *Inspection:* This more formal process is a detailed analysis in which the artifact is checked against a prepared list of concerns. The creator does not lead the discussion, and the fault identification and correction are often controlled by statistical measurements.

Administrative Controls

Not all controls can be imposed automatically by the computing system. Sometimes controls are applied instead by the declaration that certain practices will be followed. These controls, encouraged by managers and administrators, are called administrative controls. We look at them briefly here and in more depth in Chapter 8.

Standards of Program Development

No software development organization worth its salt allows its developers to produce code at any time in any manner. The good software development practices described earlier in this chapter have all been validated by many years of practice. Although none is Brooks's mythical "silver bullet" that guarantees program correctness, quality, or security, they all add demonstrably to the strength of programs. Thus, organizations prudently establish standards on how programs are developed. Even advocates of agile methods, which give developers an unusual degree of flexibility and autonomy, encourage goal-directed behavior based on past experience and past success. Standards and guidelines can capture wisdom from previous projects and increase the likelihood that the resulting system will be correct. In addition, we want to ensure that the systems we build are reasonably easy to maintain and are compatible with the systems with which they interact.

We can exercise some degree of administrative control over software development by considering several kinds of standards or guidelines.

- standards of *design*, including using specified design tools, languages, or methodologies, using design diversity, and devising strategies for error handling and fault tolerance
- standards of *documentation*, *language*, and *coding style*, including layout of code on the page, choices of names of variables, and use of recognized program structures
- standards of *programming*, including mandatory peer reviews, periodic code audits for correctness, and compliance with standards
- standards of *testing*, such as using program verification techniques, archiving test results for future reference, using independent testers, evaluating test thoroughness, and encouraging test diversity
- standards of *configuration management*, to control access to and changes of stable or completed program units

Standardization improves the conditions under which all developers work by establishing a common framework so that no one developer is indispensable. It also allows carryover from one project to another; lessons learned on previous projects become available for use by all on the next project. Standards also assist in maintenance, since the maintenance team can find required

information in a well-organized program. However, we must take care so that the standards do not unnecessarily constrain the developers.

Firms concerned about security and committed to following software development standards often perform **security audits**. In a security audit, an independent security evaluation team arrives unannounced to check each project's compliance with standards and guidelines. The team reviews requirements, designs, documentation, test data and plans, and code. Knowing that documents are routinely scrutinized, a developer is unlikely to put suspicious code in a component in the first place.

Program Controls in General

This section has explored how to control for faults during the program development process. Some controls apply to how a program is developed, and others establish restrictions on the program's use. The best is a combination, the classic layered defense.

Is one control essential? Can one control be skipped if another is used? Although these are valid questions, the security community does not have answers. Software development is both an art and science. As a creative activity, it is subject to the variety of human minds, but also to the fallibility of humans. We cannot rigidly control the process and get the same results time after time, as we can with a machine.

But creative humans can learn from their mistakes and shape their creations to account for fundamental principles. Just as a great painter will achieve harmony and balance in a painting, a good software developer who truly understands security will incorporate security into all phases of development. Thus, even if you never become a security professional, this exposure to the needs and shortcomings of security will influence many of your future actions. Unfortunately, many developers do not have the opportunity to become sensitive to security issues, which probably accounts for many of the unintentional security faults in today's programs.

Unit-IV

Database Security: Introduction to Database – Security Requirement – Reliability and Integrity – Sensitive Data – Inference – Multilevel Databases - Multilevel Security.

INTRODUCTION TO DATABASE

Concept of a Database

A **database** is a collection of *data* and a set of *rules* that organize the data by specifying certain relationships among the data. Through these rules, the user describes a *logical* format

for the data. The data items are stored in a file, but the precise *physical* format of the file is of no concern to the user. A **database administrator** is a person who defines the rules that organize the data and also controls who should have access to what parts of the data. The user interacts with the database through a program called a **database manager** or a **database management system (DBMS)**, informally known as a **frontend.**

Components of Databases

The database file consists of **records,** each of which contains one related group of data. Each record contains **fields** or **elements**, the elementary data items themselves.The logical structure of a database is called a **schema**. A particular user may have access to only part of the database, called a **subschema.** The name of each column is called an **attribute** of the database. A **relation** is a set of columns.

ADAMS	212 Market St.	Columbus	OH	43210
BENCHLY	501 Union St.	Chicago	IL	60603
CARTER	411 Elm St.	Columbus	OH	43210

Queries

Users interact with database managers through commands to the DBMS that retrieve, modify, add, or delete fields and records of the database. A command is called a **query**. Database management systems have precise rules of syntax for queries. Most query languages use an English-like notation, and many are based on SQL, a structured query language originally developed by IBM. We have written the example queries in this chapter to resemble English sentences so that they are easy to understand. For example, the query

SELECT NAME = 'ADAMS'

retrieves all records having the value *ADAMS* in the NAME field.

Advantages of Using Databases

The logical idea behind a database is this: A database is a single collection of data, stored and maintained at one central location, to which many people have access as needed. However, the actual implementation may involve some other physical storage arrangement or access. The essence of a good database is that the users are unaware of the physical arrangements; the unified logical arrangement is all they see. As a result, a database offers many advantages over a simple file system:

- *shared access,* so that many users can use one common, centralized set of data
- *controlled access,* so that only authorized users are allowed to view or to modify data values

- *minimal redundancy,* so that individual users do not have to collect and maintain their own sets of data
- *data consistency*, so that a change to a data value affects all users of the data value*data integrity,* so that data values are protected against accidental or malicious undesirable changes

Security Requirements of Databases

The basic security requirements of database systems are not unlike those of other computing systems we have studied. The basic problems—access control, exclusion of spurious data, authentication of users, and reliability—have appeared in many contexts so far in this book. Following is a list of requirements for database security.

- *Physical database integrity*. The data of a database are immune from physical problems, such as power failures, and someone can reconstruct the database if it is destroyed through a catastrophe.
- *Logical database integrity*. The structure of the database is preserved. With logical integrity of a database, a modification to the value of one field does not affect other fields,
- *Element integrity*. The data contained in each element are accurate.
- *Auditability*. It is possible to track who or what has accessed (or modified) the elements in the database.
- *Access control*. A user is allowed to access only authorized data, and different users can be restricted to different modes of access (such as read or write).
- *User authentication*. Every user is positively identified, both for the audit trail and for permission to access certain data.
- *Availability*. Users can access the database in general and all the data for which they are authorized.

Integrity of the Database

If a database is to serve as a central repository of data, users must be able to trust the accuracy of the data values. This condition implies that the database administrator must be assured that updates are performed only by authorized individuals. It also implies that the data must be protected from corruption, either by an outside illegal program action or by an outside force such as fire or a power failure. Two situations can affect the integrity of a database: when the whole database is damaged or when individual data items are unreadable.

Integrity of the database as a whole is the responsibility of the DBMS, the operating system, and the (human) computing system manager. From the perspective of the operating system and the computing system manager, databases and DBMSs are files and programs, respectively. Therefore, one way of protecting the database as a whole is to regularly back up all files on the system. These periodic backups can be adequate controls against catastrophic failure.

Sometimes an administrator needs to be able to reconstruct the database at the point of a

failure. For instance, when the power fails suddenly, a bank's clients may be in the middle of making transactions or students may be registering online for their classes. In these cases, owners want to be able to restore the systems to a stable point without forcing users to redo their recently completed transactions.

To handle these situations, the DBMS must maintain a log of transactions. For example, suppose the banking system is designed so that a message is generated in a log (electronic or paper or both) each time a transaction is processed. In the event of a system failure, the system can obtain accurate account balances by reverting to a backup copy of the database and reprocessing all later transactions from the log.

Element Integrity

The **integrity** of database elements is their correctness or accuracy. Ultimately, authorized users are responsible for entering correct data in databases. However, users and programs make mistakes collecting data, computing results, and entering values. Therefore, DBMSs sometimes take special action to help catch errors as they are made and to correct errors after they are inserted.

This corrective action can be taken in three ways: by field checks, through access control, and with change log.

First, the DBMS can apply **field checks**, activities that test for appropriate values in a position. A field might be required to be numeric, an uppercase letter, or one of a set of acceptable characters. The check ensures that a value falls within specified bounds or is not greater than the sum of the values in two other fields. These checks prevent simple errors as the data are entered.

A second integrity action is afforded by **access control**. To see why, consider life without databases. Data files may contain data from several sources, and redundant data may be stored in several different places. For example, a student's mailing address may be stored in many different campus files: in the registrar's office for formal correspondence, in the food service office for dining hall privileges, at the bookstore for purchases, and in the financial aid office for accounting. Indeed, the student may not even be aware that each separate office has the address on file. If the student moves from one residence to another, each of the separate files requires correction.

Without a database, you can imagine the risks to the data's integrity. First, at a given time, some data files could show the old address (they have not yet been updated) and some simultaneously have the new address (they have already been updated). Second, there is always the possibility that someone misentered a data field, again leading to files with incorrect information. Third, the student may not even be aware of some files, so he or she does not know to notify the file owner about updating the address information. These problems are solved by databases. They enable collection and control of this data at one central source, ensuring the student and users of having the correct address.

However, the centralization is easier said than done. Who owns this shared central file? Who is authorized to update which elements? What if two people apply conflicting modifications? What if modifications are applied out of sequence? How are duplicate records detected? What action is taken when duplicates are found? These are policy questions that

must be resolved by the database administrator.

The third means of providing database integrity is maintaining a **change log** for the database. A change log lists every change made to the database; it contains both original and modified values. Using this log, a database administrator can undo any changes that were made in error. For example, a library fine might erroneously be posted against Charles W. Robertson, instead of Charles M. Robertson, flagging Charles W. Robertson as ineligible to participate in varsity athletics. Upon discovering this error, the database administrator obtains Charles W.'s original eligibility value from the log and corrects the database.

Auditability

For some applications administrators may want to generate an audit record of all access (read or write) to a database. Such a record can help to maintain the database's integrity, or at least to discover after the fact who had affected what values and when. A second advantage, as we see later, is that users can access protected data incrementally; that is, no single access reveals protected data, but a set of sequential accesses viewed together reveals the data, much like discovering the clues in a detective novel. In this case, an audit trail can identify which clues a user has already been given, as a guide to whether to tell the user more.

Access Control

Databases are often separated logically by user access privileges. For example, all users can be granted access to general data, but only the personnel department can obtain salary data and only the marketing department can obtain sales data. Databases are useful because they centralize the storage and maintenance of data. Limited access is both a responsibility and a benefit of this centralization.

The database administrator specifies who should be allowed access to which data, at the view, relation, field, record, or even element level. The DBMS must enforce this policy, granting access to all specified data or no access where prohibited. Furthermore, the number of modes of access can be many. A user or program may have the right to read, change, delete, or append to a value, add or delete entire fields or records, or reorganize the entire database.

Superficially, access control for a database seems like access control for an operating system or any other component of a computing system. However, the database problem is more complicated, as we see throughout this chapter. Operating system objects, such as files, are unrelated items, whereas records, fields, and elements are related. Although a user probably cannot determine the contents of one file by reading others, a user might be able to determine one data element just by reading others. The problem of obtaining data values from others is called **inference**, and we consider it in depth later in this chapter.

It is important to notice that you can access data by inference without needing direct access to the secure object itself. Restricting inference may mean prohibiting certain paths to prevent possible inferences. However, restricting access to control inference also limits queries from users who do not intend unauthorized access to values. Moreover, attempts to check requested accesses for possible unacceptable inferences may actually degrade the DBMS's performance.

User Authentication

The DBMS can require rigorous user authentication. For example, a DBMS might insist that a user pass both specific password and time-of-day checks. This authentication supplements the authentication performed by the operating system. Typically, the DBMS runs as an application program on top of the operating system. This system design means that there is no trusted path from the DBMS to the operating system, so the DBMS must be suspicious of any data it receives, including a user identity from the operating system. Thus, the DBMS is forced to do its own authentication.

Availability

A DBMS has aspects of both a program and a system. It is a program that uses other hardware and software resources, yet to many users it is the only application run. Users often take the DBMS for granted, employing it as an essential tool with which to perform particular tasks. But when the system is not available—busy serving other users or down to be repaired or upgraded—the users are very aware of a DBMS's unavailability. For example, two users may request the same record, and the DBMS must arbitrate; one user is bound to be denied access for a while. Or the DBMS may withhold unprotected data to avoid revealing protected data, leaving the requesting user unhappy. We examine these problems in more detail later in this chapter. Problems like these result in high availability requirements for a DBMS.

Integrity/Confidentiality/Availability

The three aspects of computer security—integrity, confidentiality, and availability—clearly relate to database management systems. As we have described, integrity applies to the individual elements of a database as well as to the database as a whole. Integrity is also a property of the structure of the database (elements in one table correspond one to one with those of another) and of the relationships of the database (records having the same unique identifier, called a key, are related). Thus, integrity is a major concern in the design of database management systems.

Reliability and Integrity

Databases amalgamate data from many sources, and users expect a DBMS to provide access to the data in a reliable way. When software engineers say that software has **reliability**, they mean that the software runs for very long periods of time without failing. Users certainly expect a DBMS to be reliable, since the data usually are key to business or organizational needs. Moreover, users entrust their data to a DBMS and rightly expect it to protect the data from loss or damage. Concerns for reliability and integrity are general security issues, but they are more apparent with databases.

A DBMS guards against loss or damage in several ways, which we study in this section. However, the controls we consider are not absolute: No control can prevent an authorized user from inadvertently entering an acceptable but incorrect value.

Database concerns about reliability and integrity can be viewed from three dimensions:

- *Database integrity:* concern that the database as a whole is protected against damage, as

from the failure of a disk drive or the corruption of the master database index. These concerns are addressed by operating system integrity controls and recovery procedures.

- *Element integrity:* concern that the value of a specific data element is written or changed only by authorized users. Proper access controls protect a database from corruption by unauthorized users.
- *Element accuracy:* concern that only correct values are written into the elements of a database. Checks on the values of elements can help prevent insertion of improper values. Also, constraint conditions can detect incorrect values.

Redundancy/Internal Consistency

Many DBMSs maintain additional information to detect internal inconsistencies in data. The additional information ranges from a few check bits to duplicate or shadow fields, depending on the importance of the data.

Error Detection and Correction Codes

One form of redundancy is error detection and correction codes, such as parity bits, Hamming codes, and cyclic redundancy checks. These codes can be applied to single fields, records, or the entire database. Each time a data item is placed in the database, the appropriate check codes are computed and stored; each time a data item is retrieved, a similar check code is computed and compared to the stored value. If the values are unequal, they signify to the DBMS that an error has occurred in the database. Some of these codes point out the place of the error; others show precisely what the correct value should be. The more information provided, the more space required to store the codes.

Shadow Fields

Entire attributes or entire records can be duplicated in a database. If the data are irreproducible, this second copy can provide an immediate replacement if an error is detected. Obviously, redundant fields require substantial storage space.

Recovery

In addition to these error correction processes, a DBMS can maintain a log of user accesses, particularly changes. In the event of a failure, the database is reloaded from a backup copy and all later changes are then applied from the audit log.

Concurrency/Consistency

Database systems are often multiuser systems. Accesses by two users sharing the same database must be constrained so that neither interferes with the other. Simple locking is done by the DBMS. If two users attempt to read the same data item, there is no conflict because both obtain the same value.

If both users try to modify the same data items, we often assume that there is no conflict because each knows what to write; the value to be written does not depend on the previous value of the data item. However, this supposition is not quite accurate.

Sensitive Data

Sensitive data are data that should not be made public. Determining which data items and fields are sensitive depends both on the individual database and the underlying meaning of the data. Obviously, some databases, such as a public library catalog, contain no sensitive data; other databases, such as defense-related ones, are wholly sensitive. These two cases—nothing sensitive and everything sensitive—are the easiest to handle, because they can be covered by access controls to the database as a whole. Someone either is or is not an authorized user. These controls can be provided by the operating system.

The more difficult problem, which is also the more interesting one, is the case in which *some but not all* of the elements in the database are sensitive. There may be varying degrees of sensitivity. For example, a university database might contain student data consisting of name, financial aid, dorm, drug use, sex, parking fines, and race. An example of this database is shown in Table . Name and dorm are probably the least sensitive; financial aid, parking fines, and drug use the most; sex and race somewhere in between. That is, many people may have legitimate access to name, some to sex and race, and relatively few to financial aid, parking fines, or drug use. Indeed, knowledge of the existence of some fields, such as drug use, may itself be sensitive. Thus, securityconcerns not only the data elements but their context and meaning.

Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey

Furthermore, we must account for different degrees of sensitivity. For instance, although all the fields are highly sensitive, the financial aid, parking fines, and drug-use fields may not have the same kinds of access restrictions. Our security requirements may demand that a few people be authorized to see each field, but no one be authorized to see all three. The challenge of the access control problem is to limit users' access so that they can obtain only the data to which they have legitimate access. Alternatively, the access control problem forces us to ensure that sensitive data are not released to unauthorized people.

Several factors can make data sensitive.

- *Inherently sensitive*. The value itself may be so revealing that it is sensitive. Examples are the locations of defensive missiles or the median income of barbers in a town with only one barber.
- *From a sensitive source*. The source of the data may indicate a need for confidentiality. An example is information from an informer whose identity would be compromised if the information were disclosed.
- *Declared sensitive*. The database administrator or the owner of the data may have declared the data to be sensitive. Examples are classified military data or the name of the anonymous donor of a piece of art.
- Part of a *sensitive attribute* or *record.* In a database, an entire attribute or record may be classified as sensitive. Examples are the salary attribute of a personnel database or a

record describing a secret space mission.

- Sensitive *in relation to previously disclosed information*. Some data become sensitive in the presence of other data. For example, the longitude coordinate of a secret gold mine reveals little, but the longitude coordinate in conjunction with the latitude coordinate pinpoints the mine.

All of these factors must be considered when the sensitivity of the data is being determined.

Database Security and Threats

Data security is an imperative aspect of any database system. It is of particular importance in distributed systems because of large number of users, fragmented and replicated data, multiple sites and distributed control.

Threats in a Database

- **Availability loss** − Availability loss refers to non-availability of database objects by legitimate users.
- **Integrity loss** − Integrity loss occurs when unacceptable operations are performed upon the database either accidentally or maliciously. This may happen while creating, inserting, updating or deleting data. It results in corrupted data leading to incorrect decisions.
- **Confidentiality loss** − Confidentiality loss occurs due to unauthorized or unintentional disclosure of confidential information. It may result in illegal actions, security threats and loss in public confidence.

Measures of Control

The measures of control can be broadly divided into the following categories −

- **Access Control** − Access control includes security mechanisms in a database management system to protect against unauthorized access. A user can gain access to the database after clearing the login process through only valid user accounts. Each user account is password protected.
- **Flow Control** − Distributed systems encompass a lot of data flow from one site to another and also within a site. Flow control prevents data from being transferred in such a way that it can be accessed by unauthorized agents. A flow policy lists out the channels through which information can flow. It also defines security classes for data as well as transactions.
- **Data Encryption** − Data encryption refers to coding data when sensitive data is to be communicated over public channels. Even if an unauthorized agent gains access of the data, he cannot understand it since it is in an incomprehensible format.

Multi-Level Security (MLS)

Protecting sensitive or confidential data is paramount in many businesses. In the event such information is made public, businesses may face legal or financial ramifications. At the very least, they will suffer a loss of customer trust. In most cases, however, they can recover from these financial and other losses with appropriate investment or compensation.

The same cannot be said of the defense and related communities, which includes military services, intelligence organizations and some areas of police service. These organizations cannot easily recover should sensitive information be leaked, and may not recover at all. These communities require higher levels of security than those employed by businesses and other organizations.

Having information of different security levels on the same computer systems poses a real threat. It is not a straight-forward matter to isolate different information security levels, even though different users log in using different accounts, with different permissions and different access controls.

Some organizations go as far as to purchase dedicated systems for each security level. This is often prohibitively expensive, however. A mechanism is required to enable users at different security levels to access systems simultaneously, without fear of information contamination.

Unit-V

Network Security: networks Concepts – Threats in Networks – Network Security Controls – Firewalls – l. Electronic Mail Security – IP Security – Web Security.

Network Concepts

A network is a little more complicated than a local computing installation. A set of components are computers, printers, storage devices, and so forth and wires. A wire is point to point, with essentially no leakage between end points, although wiretapping does allow anyone with access to the wire to intercept, modify, or even block the transmission. In a local environment, the physical wires are frequently secured physically or perhaps visually so wiretapping is not a major issue. With remote communication, the same notion of wires applies, but the wires are outside the control and protection of the user, so tampering with the transmission is a serious threat. The nature of that threat depends in part on the medium of these "wires," which can actually be metal wire, glass fibers, or electromagnetic signals such as radio communications. In a moment we look at different kinds of communications media.

Returning our attention to the local environment with a wire for each pair of devices, to send data from one device to another the sender simply uses the one wire to the destination. With a remote network, ordinarily the sender does not have one wire for each possible recipient, because the number of wires would become unmanageable. Instead, as you probably know, the sender precedes data with what is essentially a mailing label, a tag showing to where (and often from where) to transmit data. At various points along the transmission path devices inspect the label to determine if that device is the intended recipient and, if not, how to forward the data to get nearer to the destination. This processing of a label is called routing.

THREATS IN NETWORKS

There are four general categories of attack which are listed below.

Interruption

An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability e.g., destruction of piece of hardware, cutting of a communication line or Disabling of file management system.

Interception

An unauthorized party gains access to an asset. This is an attack on confidentiality. Unauthorized party could be a person, a program or a computer. e.g., wire tapping to capture data in the network, illicit copying of files.

Modification

An unauthorized party not only gains access to but tampers with an asset. This is an attack on integrity. e.g., changing values in data file, altering a program, modifying the contents of messages being transmitted in a network.

Fabrication

An unauthorized party inserts counterfeit objects into the system. This is an attack on authenticity. e.g., insertion of spurious message in a network or addition of records to a file.

WIRETAPPING

Passive Attacks

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain Information that is being transmitted.

Passive attacks are of two types:

- **Release of message contents:** A telephone conversation, an e-mail message and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.
- **Traffic analysis**: If we had encryption protection in place, an opponent might still be able to observe the pattern of the message. The opponent could determine the location and identity of communication hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of data. However, it is feasible to prevent the success of these attacks.

Active Attacks

These attacks involve some modification of the data stream or the creation of a false stream. These attacks can be classified in to four categories:

- **Masquerade** – One entity pretends to be a different entity.
- **Replay** – involves passive capture of a data unit and its subsequent transmission to produce an unauthorized effect.
- **Modification of messages** – Some portion of message is altered or the messages are delayed or recorded, to produce an unauthorized effect.
- **Denial of service** – Prevents or inhibits the normal use or management of communication
- facilities. Another form of service denial is the disruption of an entire network, either by disabling the network or overloading it with messages so as to degrade performance.

It is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them.

FIREWALLS

Firewalls can either be software or hardware, though it's best to have both. A software firewall is a program installed on each computer and regulates traffic through port numbers and

applications, while a physical firewall is a piece of equipment installed between your network and gateway.

Packet-filtering firewalls, the most common type of firewall, examine packets and prohibit them from passing through if they don't match an established security rule set. This type of firewall checks the packet's source and destination IP addresses. If packets match those of an "allowed" rule on the firewall, then it is trusted to enter the network.

Packet-filtering firewalls are divided into two categories: stateful and stateless. Stateless firewalls examine packets independently of one another and lack context, making them easy targets for hackers. In contrast, stateful firewalls remember information about previously passed packets and are considered much more secure.

While packet-filtering firewalls can be effective, they ultimately provide very basic protection and can be very limited—for example, they can't determine if the contents of the request that's being sent will adversely affect the application it's reaching. If a malicious request that was allowed from a trusted source address would result in, say, the deletion of a database, the firewall would have no way of knowing that. Next-generation firewalls and proxy firewalls are more equipped to detect such threats.

Next-generation firewalls (NGFW) combine traditional firewall technology with additional functionality, such as encrypted traffic inspection, intrusion prevention systems, anti-virus, and more. Most notably, it includes deep packet inspection (DPI). While basic firewalls only look at packet headers, deep packet inspection examines the data within the packet itself, enabling users to more effectively identify, categorize, or stop packets with malicious data.

Proxy firewalls filter network traffic at the application level. Unlike basic firewalls, the proxy acts an intermediary between two end systems. The client must send a request to the firewall, where it is then evaluated against a set of security rules and then permitted or blocked. Most notably, proxy firewalls monitor traffic for layer 7 protocols such as HTTP and FTP, and use both stateful and deep packet inspection to detect malicious traffic.

Network address translation (NAT) firewalls allow multiple devices with independent network addresses to connect to the internet using a single IP address, keeping individual IP addresses hidden. As a result, attackers scanning a network for IP addresses can't capture specific details, providing greater security against attacks. NAT firewalls are similar to proxy firewalls in that they act as an intermediary between a group of computers and outside traffic.

Stateful multilayer inspection (SMLI) firewalls filter packets at the network, transport, and application layers, comparing them against known trusted packets. Like NGFW firewalls, SMLI also examine the entire packet and only allow them to pass if they pass each layer individually. These firewalls examine packets to determine the state of the communication (thus the name) to ensure all initiated communication is only taking place with trusted sources.

Electronic Mail Security

Two Schemes are

i. PGP
ii. S/MIME

PGP (Pretty Good Privacy)

It provide five type of services

i. Authentication
ii. Confidentiality
iii. Compression
iv. E-mail Compatibility
v. Segmentation

Notations

Ks =session key used in symmetric encryption scheme

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

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

EP = public-key encryption

DP = public-key decryption

EC = symmetric encryption

DC = symmetric decryption

H = hash function

|| = concatenation

Z = compression using ZIP algorithm

R64 = conversion to radix 64 ASCII format

Authentication

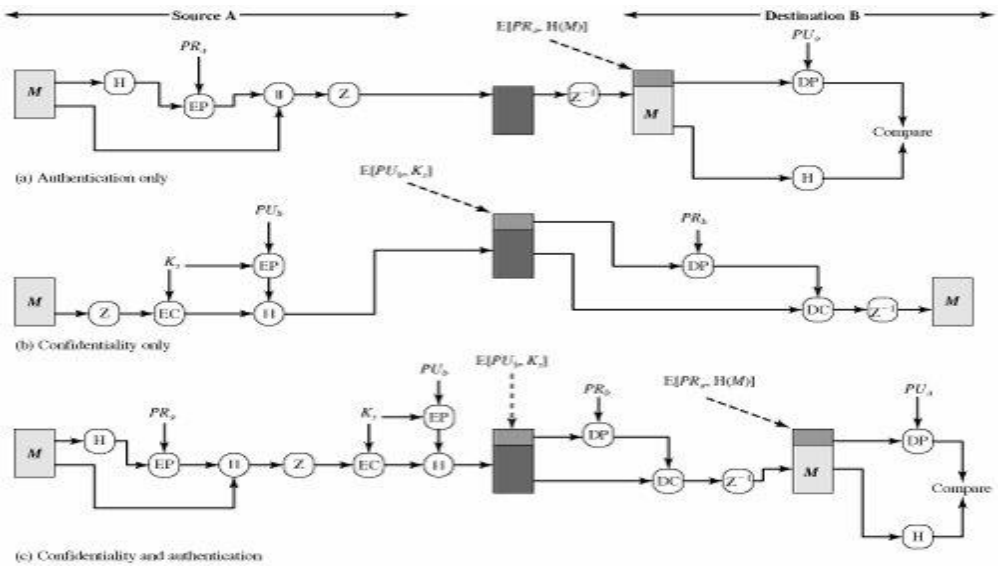
1.The sender creates a message.

2.SHA-1 is used to generate a 160-bit hash code of the message.

3.The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.

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

5.The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.



Confidentiality

1.The sender generates a message and a random 128-bit number to be used as a session key for this message only.

2.The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.

3.The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.

4.The receiver uses RSA with its private key to decrypt and recover the session key.

5.The session key is used to decrypt the message.

Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

E-mail Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting 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.

Segmentation and 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. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing.

S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet email format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users.

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 lists the following limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a defacto standard.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
    - Deletion, addition, or reordering of carriage return and linefeed
    - Truncating or wrapping lines longer than 76 characters
    - Removal of trailing white space (tab and space characters)
    - Padding of lines in a message to the same length
    - Conversion of tab characters into multiple space characters

MIME – Header Fields

- **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.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

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 can view the message content, although they cannot verify the signature.
- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

IP SECURITY

Benefits of IPSec

When IPSec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.

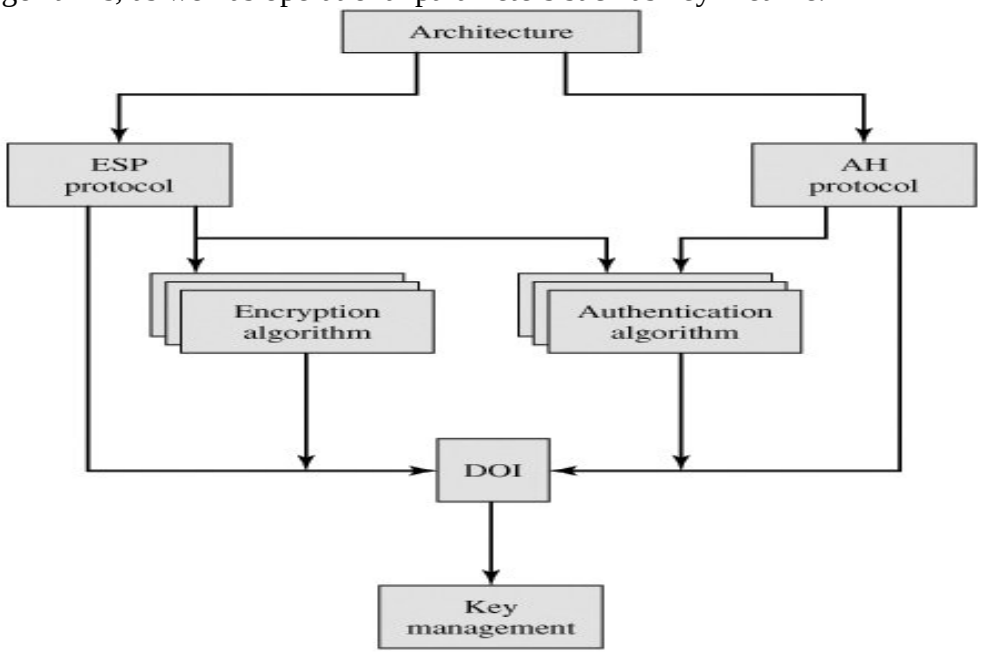
- IPSec in a firewall is resistant to bypass if all traffic from the outside must use IP, and the firewall is the only means of entrance from the Internet into the organization.
- IPSec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when IPSec is

implemented in the firewall or router. Even if IPSec is implemented in end systems, upper-layer software, including applications, is not affected.

- IPSec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.
- IPSec can provide security for individual users if needed. This is useful for offsite workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

IP Security Architecture

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



IPSec Services

IPSec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any

cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP).

The services are

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic flow confidentiality

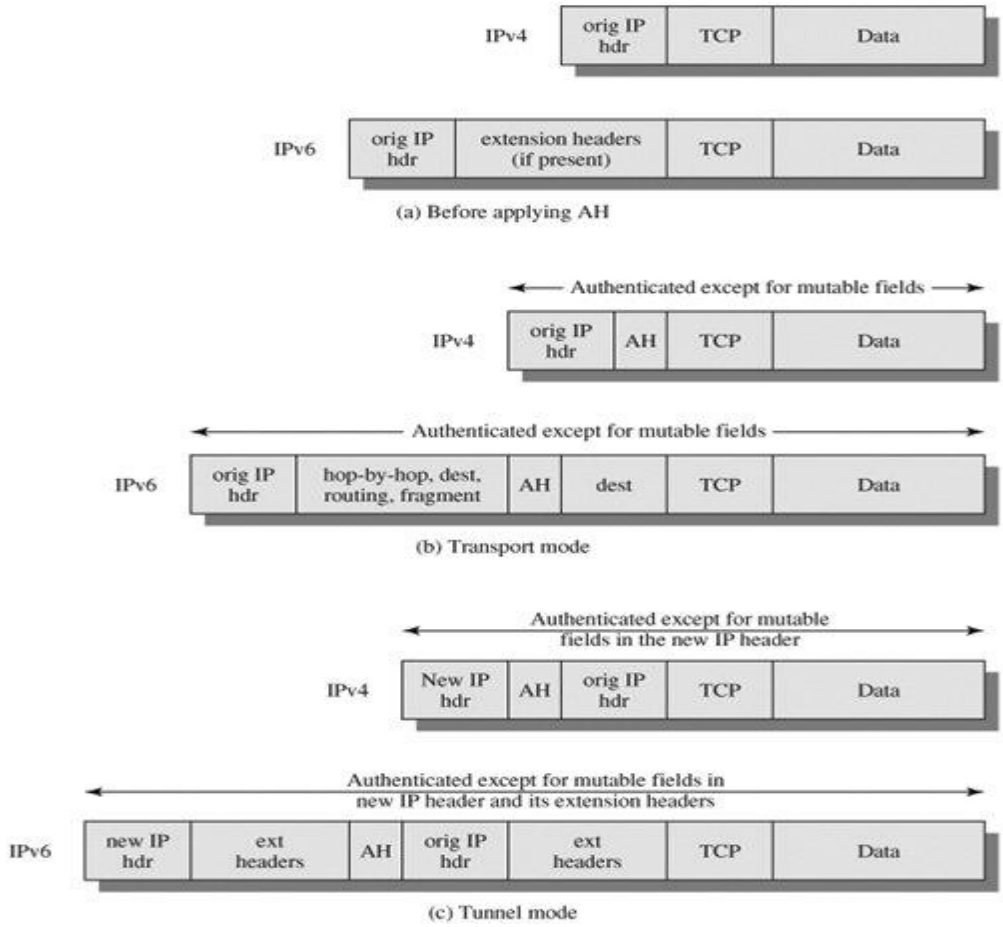
Transport and Tunnel Modes

Transport Mode

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet. Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection. ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new "outer" IP packet with a new outer IP header. The entire original, or inner, packet travels through a "tunnel" from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of an SA are a security gateway, such as a firewall or router that implements IPSec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPSec software in the firewall or secure router at the boundary of the local network.



WEB SECURITY

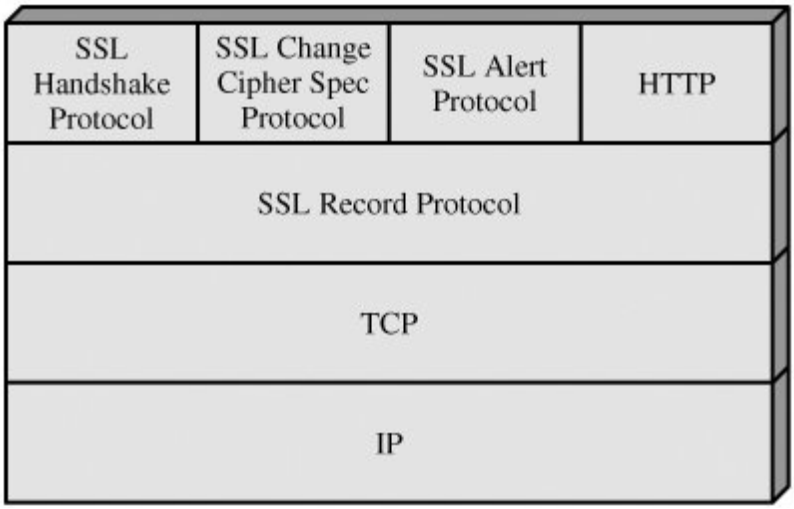
Areas:

1. SSL – Secure Socket Layer
2. TLS – Transport Layer Security
3. SET – Secure Electronic Transaction

SECURE SOCKET LAYER AND TRANSPORT LAYER SECURITY

SSL Architecture

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



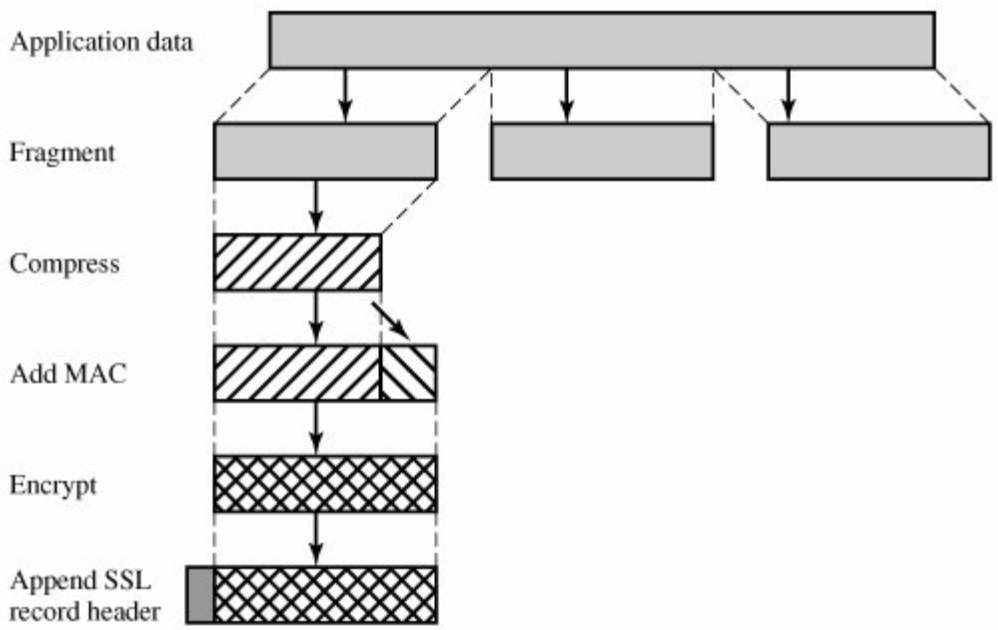
The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: 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.

SSL Record Protocol

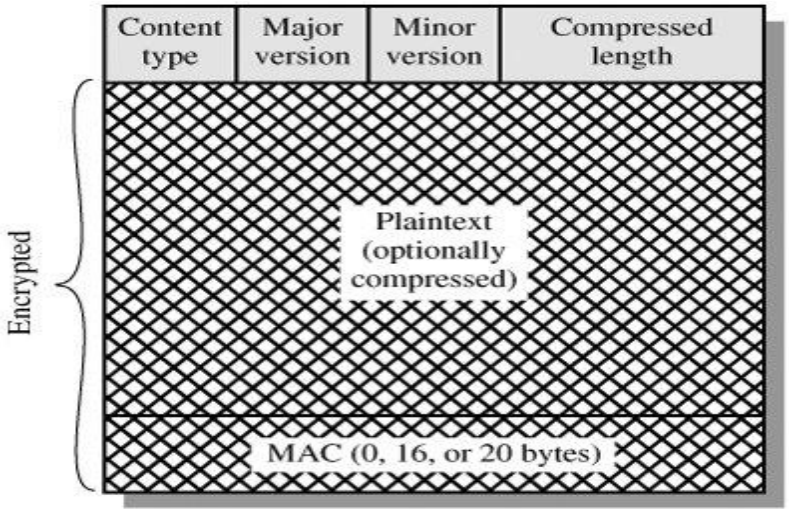
The SSL Record Protocol provides two services for SSL connections:

- **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 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. In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null. 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.

SSL Record Format



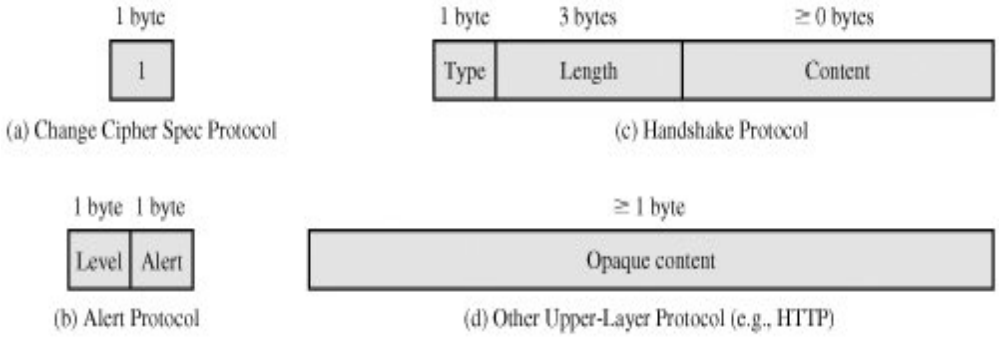
- **Content Type (8 bits):** The higher layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of SSL in use. For SSLv3, the value is 3.
- **Minor Version (8 bits):** Indicates minor version in use. For SSLv3, the value is 0.

- **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 Record Protocol Payload

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 state to be copied into the current state, which updates the cipher suite to be used on this connection.



Alert Protocol

The Alert Protocol is used to convey SSL-related alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state.

Handshake Protocol

The most complex part of SSL is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. The Handshake Protocol is used before any application data is transmitted.

SECURE ELECTRONIC TRANSACTION

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. The current version, SETv1, emerged from a call for security standards by MasterCard and Visa in February 1996. A wide range of companies were involved in developing the initial specification, including IBM, Microsoft, Netscape, RSA, Terisa, and Verisign.

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,

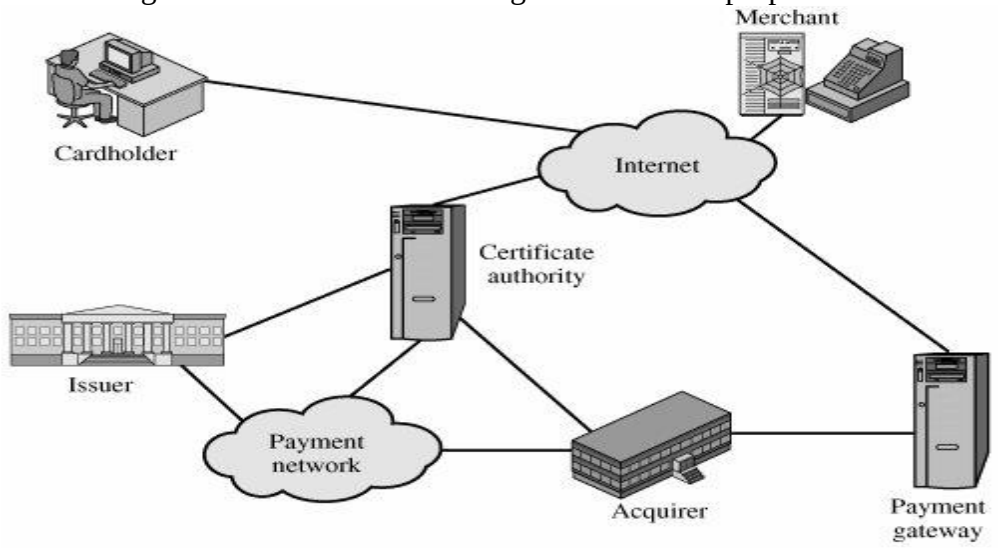
SET provides three services:

- 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

Key Features of SET

To meet the requirements just outlined, SET incorporates the following features:

- **Confidentiality of information:** Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.
- **Integrity of data:** Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.
- **Cardholder account authentication:** SET enables merchants to verify that a cardholder is a legitimate user of a valid card account number. SET uses X.509v3 digital certificates with RSA signatures for this purpose.
- **Merchant authentication:** SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509v3 digital certificates with RSA signatures for this purpose.



SET Participants

- **Cardholder:** In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., MasterCard, Visa) that has been issued by an issuer.

- **Merchant:** A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.
- **Issuer**: This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person.
- **Acquirer**: This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.
- **Payment gateway**: This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.
- **Certification authority (CA):** This is an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. A hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

The sequence of events that are required for a transaction

1. **The customer opens an account.** The customer obtains a credit card account, such as MasterCard or Visa, with a bank that supports electronic payment and SET.

2. **The customer receives a certificate.** After suitable verification of identity, the customer receives an X.509v3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.

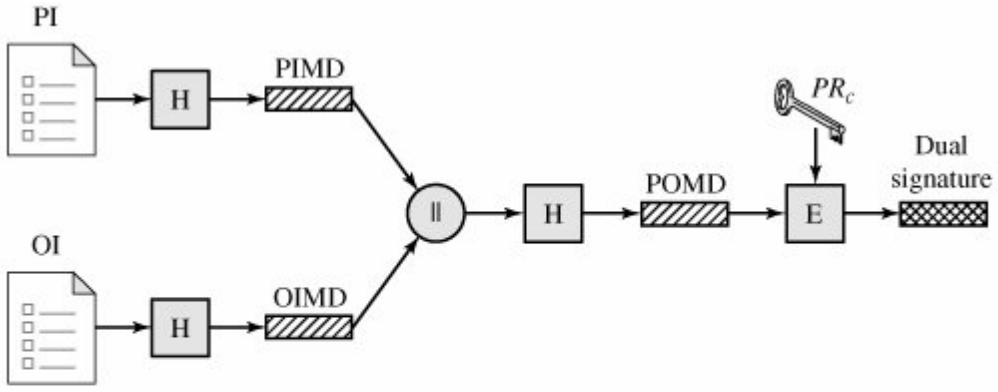
3. **Merchants have their own certificates.** A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

4. **The customer places an order.** This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.

5. **The merchant is verified.** In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.

6. **The order and payment are sent.** The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.
7. **The merchant requests payment authorization.** The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.
8. **The merchant confirms the order.** The merchant sends confirmation of the order to the customer.
9. **The merchant provides the goods or service.** The merchant ships the goods or provides the service to the customer.
10. **The merchant requests payment.** This request is sent to the payment gateway, which handles all of the payment processing.

Dual Signature



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)
PR_c = Customer's private signature key

The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, 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. However, the two items must be linked in a way that can be used to resolve disputes if necessary. 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. To see the need for the link, suppose that the customers send the merchant two messages: a signed OI and a signed PI, and the merchant passes the PI on to the bank. If the merchant can capture another OI from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature.