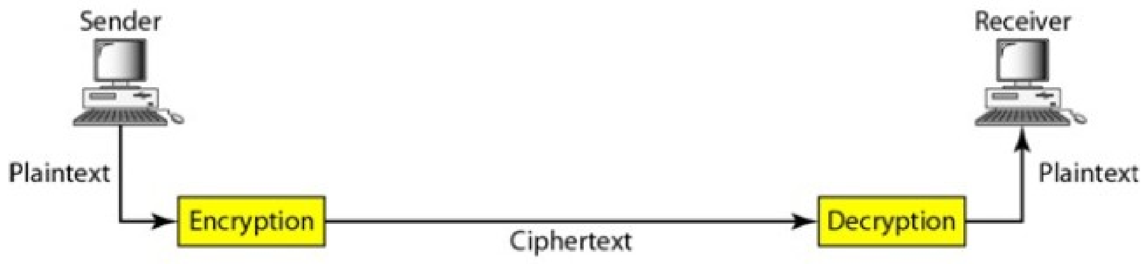
UNIT 5
Cryptography

The science and art of transforming messages to make them secure and immune to attacks. Figure 30.1 shows the components involved in cryptography.

Figure 30.1 *Cryptography components*



Plaintext and Ciphertext

The original message, before being transformed, is called plaintext. After the message is transformed, it is called ciphertext. An encryption algorithm transforms the plaintext into ciphertext; a decryption algorithm transforms the ciphertext back into plaintext. The sender uses an encryption algorithm, and the receiver uses a decryption algorithm.

Cipher

We refer to encryption and decryption algorithms as ciphers. The term *cipher* is also used to refer to different categories of algorithms in cryptography. This is not to say that every sender-receiver pair needs their very own unique cipher for a secure communication. On the contrary, one cipher can serve millions of communicating pairs.

Key

A key is a number (or a set of numbers) that the cipher, as an algorithm, operates on. To encrypt a message, we need an encryption algorithm, an encryption key, and the plaintext. These create the ciphertext. To decrypt a message, we need a decryption algorithm, a decryption key, and the ciphertext. These reveal the original plaintext.

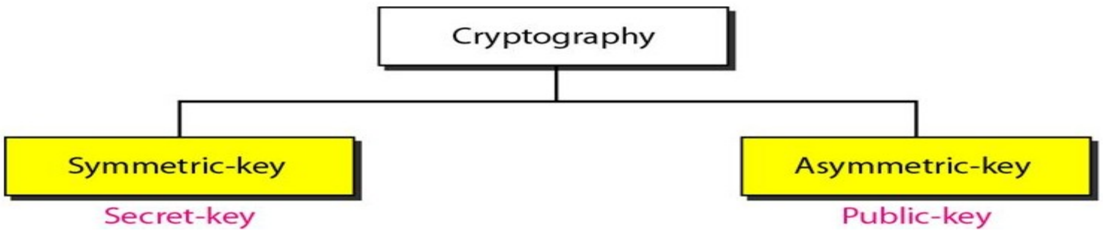
Alice, Bob, and Eve

In cryptography, it is customary to use three characters in an information exchange scenario; we use Alice, Bob, and Eve. Alice is the person who needs to send secure data. Bob is the recipient of the data. Eve is the person who somehow disturbs the communication between Alice and Bob by intercepting messages to uncover the data or by sending her own disguised messages. These three names represent computers or processes that actually send or receive data, or intercept or change data.

Two Categories

We can divide all the cryptography algorithms (ciphers) into two groups: **symmetric key** (also called secret-key) cryptography algorithms and **asymmetric** (also called public-key) cryptography algorithms. Figure 30.2 shows the taxonomy.

Figure 30.2 *Categories of cryptography*

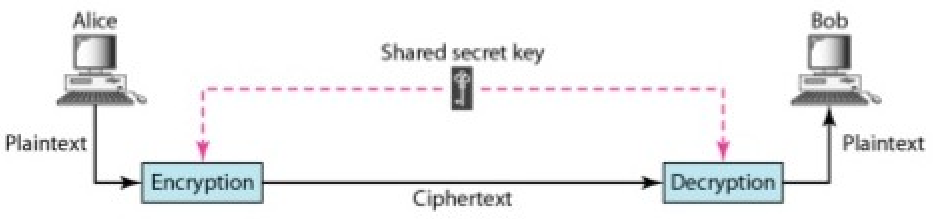


Symmetric Key Cryptography

In symmetric-key cryptography, the same key is used by both parties. The sender uses this key and an encryption algorithm to encrypt data; the receiver uses the same key and the corresponding decryption algorithm to decrypt the data.

In symmetric-key cryptography, the same key is used by the sender (for encryption) and the receiver (for decryption). The key is shared.

Figure 30.3 *Symmetric-key cryptography*



Asymmetric-Key Cryptography

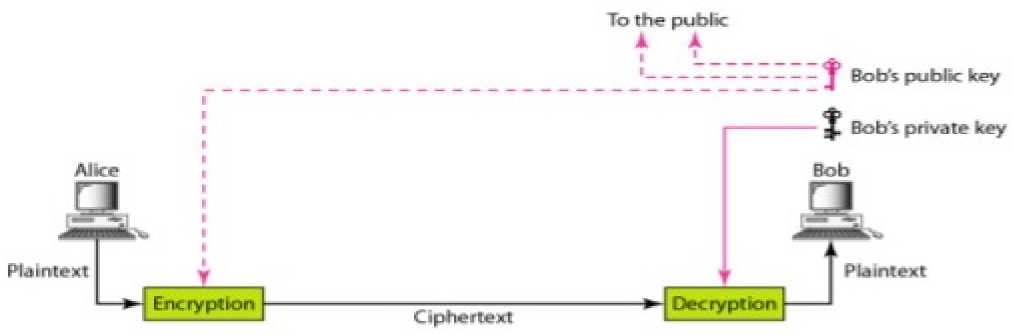
In asymmetric or public-key cryptography, there are two keys:

- a private key
- a public key.

The private key is kept by the receiver. The public key is announced to the public. In Figure 30.4, imagine Alice wants to send a message to Bob. Alice uses the public key to encrypt the message. When the message is received by Bob, the private key is used to decrypt the message.

In public-key encryption/decryption, the public key that is used for encryption is different from the private key that is used for decryption. The public key is available to the public,the private key is available only to an individual.

Figure 30.4 *Asymmetric-key cryptography*



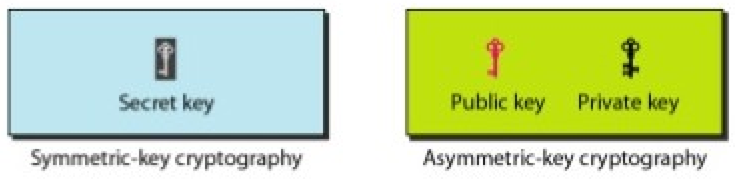
Three Types o f Keys

We are dealing with three types of keys in cryptography:

- The secret key
- The public key
- The private key.

The first, the secret key, is the shared key used in symmetric-key cryptography. The second and the third are the public and private keys used in asymmetric-key cryptography.

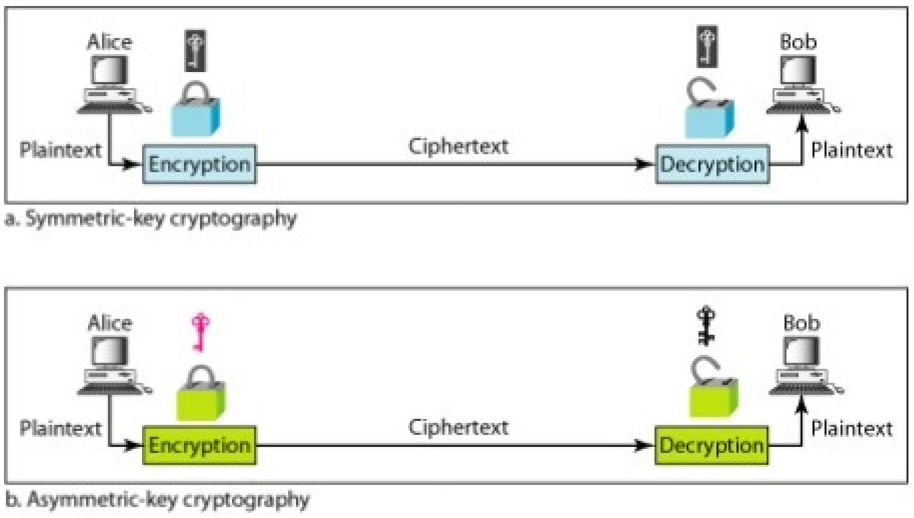
Figure 30.5 *Keys used in cryptography*



Encryption can be thought of as electronic locking; decryption as electronic unlocking. The sender puts the message in a box and locks the box by using a key; the receiver unlocks the box with a key and takes out the message. The difference lies in the mechanism of the locking and unlocking and the type of keys used.

In symmetric-key cryptography, the same key locks and unlocks the box. In asymmetric-key cryptography, one key locks the box, but another key is needed to unlock it.

Figure 30.6 *Comparison between two categories of cryptography*



SYMMETRIC-KEY CRYPTOGRAPHY

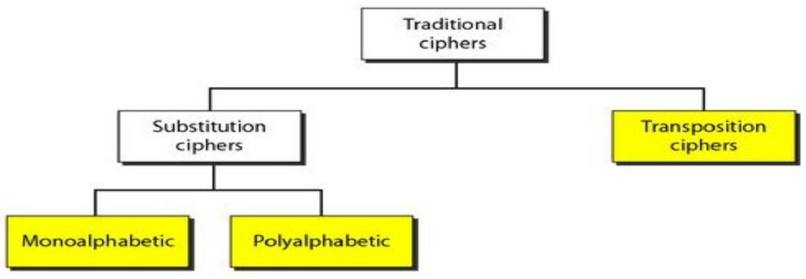
Today's ciphers are much more complex. The traditional algorithms were character-oriented. The modern algorithms are bit-oriented.

Traditional Ciphers

Traditional symmetric-key ciphers can be divided into two broad categories:

- Substitution ciphers
- Transposition ciphers, as shown in Figure 30.7.

Figure 30.7 *Traditional ciphers*



Substitution Cipher

A substitution cipher substitutes one symbol with another. If the symbols in the plaintext are alphabetic characters, we replace one character with another. For eg, we can replace character A with D, and character T with Z. If the symbols are digits (0 to 9), we can replace 3 with 7, and 2 with 6. Substitution ciphers can be categorized as either monoalphabetic or polyalphabetic ciphers.

A substitution cipher replaces one symbol with another.

In a **monoalphabetic** cipher, a character (or a symbol) in the plaintext is always changed to the same character (or symbol) in the ciphertext regardless of its position in the text. For eg, if the algorithm says that character A in the plaintext is changed to character D, every character A is changed to character D.

In a **polyalphabetic** cipher, each occurrence of a character can have a different substitute. The relationship between a character in the plaintext to a character in the ciphertext is a one-to-many relationship. For eg, character A could be changed to D in the beginning of the text, but it could be changed to N at the middle. It is obvious that if the relationship between plaintext characters and ciphertext characters is one-to-many, the key must tell us which of the many possible characters can be chosen for encryption. To achieve this goal, we need to divide the text into groups of characters and use a set of keys. For eg, we can divide the text "THISISANEASYTASK" into groups of 3 characters and then apply the encryption using a set of 3 keys. We then repeat the procedure for the next 3 characters.

Shift Cipher:

The simplest monoalphabetic cipher is probably the **shift cipher.** We assume that the plaintext and ciphertext consist of uppercase letters (A to Z) only. In this cipher, the encryption algorithm is "shift *key* characters down," with *key* equal to some number. The decryption algorithm is "shift *key* characters up." For eg, if the key is 5, the encryption algorithm is "shift 5 characters down" (toward the end of the alphabet). The decryption algorithm is "shift 5 characters up" (toward the beginning of the alphabet). Of course, if we reach the end or beginning of the alphabet, we wrap around.

Julius Caesar used the shift cipher to communicate with his officers. For this reason, the shift cipher is sometimes referred to as the Caesar cipher. Caesar used a key of 3 for his communications.

Example:

Use the shift cipher with key = 15 to encrypt the message "HELLO."

Solution

We encrypt one character at a time. Each character is shifted 15 characters down. Letter H is encrypted to W. Letter E is encrypted to T. The first L is encrypted to A. The second L is also encrypted to A. And 0 is encrypted to D. The cipher text is WTAAD.

Use the shift cipher with key =15 to decrypt the message "WTAAD."

Solution

We decrypt one character at a time. Each character is shifted 15 characters up. Letter W is decrypted to H. Letter T is decrypted to E. The first A is decrypted to L. The second A is decrypted to L. And, finally, D is decrypted to O. The plaintext is HELLO.

Transposition Ciphers

In a transposition cipher, there is no substitution of characters; instead, their locations change. A character in the first position of the plaintext may appear in the tenth position of the ciphertext. A character in the eighth position may appear in the first position. In other words, a transposition cipher reorders the symbols in a block of symbols.

A transposition cipher reorders (permutes) symbols in a block of symbols.

Key: In a transposition cipher, the key is a mapping between the position of the symbols in the plaintext and cipher text. For example, the following shows the key using a block of 4 characters:

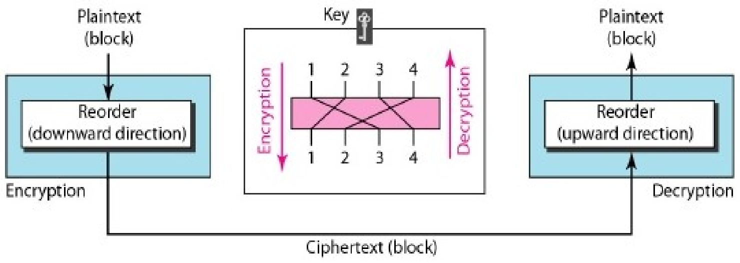
Plaintext:

Ciphertext: 2 4 1 3

1 2 3 4

In encryption, we move the character at position 2 to position 1, the character at position 4 to position 2, and so on. In decryption, we do the reverse. To be more effective, the key should be long, which means encryption and decryption of long blocks of data. Figure 30.8 shows encryption and decryption for our four-character block using the above key. The figure shows that the encryption and decryption use the same key. The encryption applies it from downward while decryption applies it upward.

Figure 30.8 ***Transposition cipher***



Example

Encrypt the message "HELLO MY DEAR," using the above key.

Solution

We first remove the spaces in the message. We then divide the text into blocks of four characters. We add a bogus character Z at the end of the third block. The result is HELL OMYD EARZ. We create a three-block ciphertext ELHLMDOYAZER.

Using Example 30.5, decrypt the message "ELHLMDOYAZER".

Solution

The result is HELL OMYD EARZ. After removing the bogus character and combining the characters, we get the original message "HELLO MY DEAR."

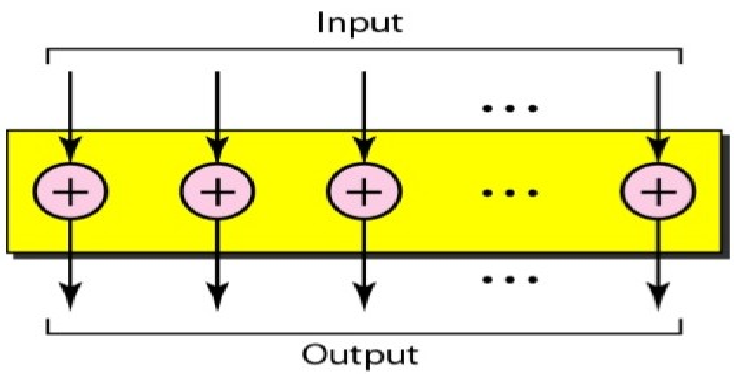
Simple Modern Ciphers

A modern symmetric cipher is a combination of simple ciphers.

XOR *Cipher*

Modern ciphers today are normally made of a set of **simple ciphers,** which are simple predefined functions in mathematics or computer science. The **XOR cipher** because it uses the exclusive-or operation as defined in computer science. Figure 30.9 shows an XOR cipher

Figure 30.9 ***XOR cipher***

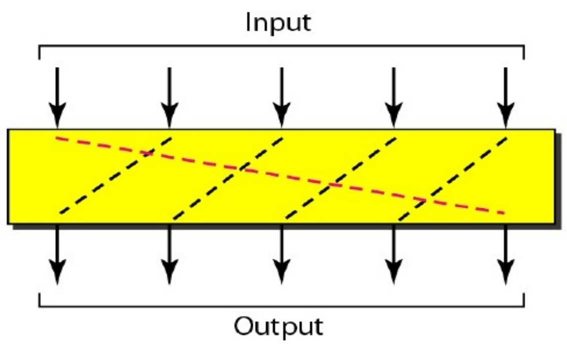


An XOR operation needs 2 data inputs plaintext, as the first and a key as the second. In other words, one of the inputs is the block to be the encrypted, the other input is 2a key; the result is the encrypted block. Note that in an XOR cipher, the size of the key, the plaintext, and the ciphertext are all the same. XOR ciphers have a very interesting property: the encryption and decryption are the same.

Rotation Cipher

Another common cipher is the rotation cipher, in which the input bits are rotated to the left or right. The rotation cipher can be keyed or keyless. In keyed rotation, the value of the key defines the number of rotations; in keyless rotation the number of rotations is fixed. Figure 30.10 shows an example of a rotation cipher. Note that the rotation cipher can be considered a special case of the transpositional cipher using bits instead of characters.

Figure 30.10 ***Rotation cipher***



If the length of the original stream is *N,* after *N* rotations, we get the original input stream. This means that it is useless to apply more than *N* - 1 rotations. In other words, the number of rotations must be between 1 and *N-1.*

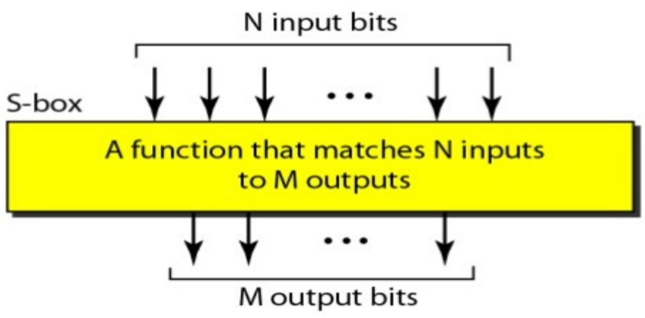
The decryption algorithm for the rotation cipher uses the same key and the opposite rotation direction. If we use a right rotation in the encryption, we use a left rotation in decryption and vice versa.

Substitution Cipher: S-box

An S-box (substitution box) parallels the traditional substitution cipher for characters. The input to an S-box is a stream of bits with length *N;* the result is another stream of bits with length *M.* N and *M* are not necessarily the same. Figure 30.11 shows an S-box.

The S-box is normally keyless and is used as an intermediate stage of encryption or decryption. The function that matches the input to the output may be defined mathematically or by a table.

Figure 30.11 ***S-box***



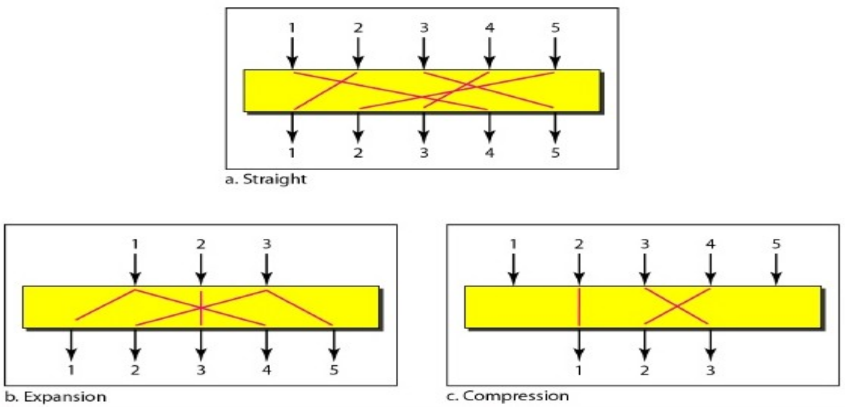
Transposition Cipher: P-box

A P-box (permutation box) for bits parallels the traditional transposition cipher for characters. It performs a transposition at the bit level; it transposes bits. It can be implemented in software or hardware, but hardware is faster. P-boxes, like S-boxes, are normally keyless.

We can have 3 types of permutations in P-boxes:

- **Straight permutation,**
- **Expansion permutation**
- **Compression permutation** as shown in Figure 30.12.

Figure 30.12 ***P-boxes: straight, expansion, and compression***



A straight permutation cipher or a straight P-box has the same number of inputs as outputs. In other words, if the number of inputs is *N,* the number of outputs is also *N.* In an expansion permutation cipher, the number of output ports is greater than the number of input ports. In a compression permutation cipher, the number of output ports is less than the number of input ports.

Modern Round Ciphers

The ciphers of today are called **round ciphers** because they involve multiple **rounds,** where each round is a complex cipher made up of the simple ciphers. The key used in each round is a subset or variation of the general key called the round key. If the cipher has *N* rounds, a key generator produces *N* keys, K_1, K_2, ..., K_N, where K_1 is used in round 1, K_2 in round 2, and so on.

Two modern symmetric-key ciphers:

- DES
- AES.

These ciphers are referred to as **block ciphers** because they divide the plaintext into blocks and use the same key to encrypt and decrypt the blocks. DES has been the de facto standard until recently. AES is the formal standard now.

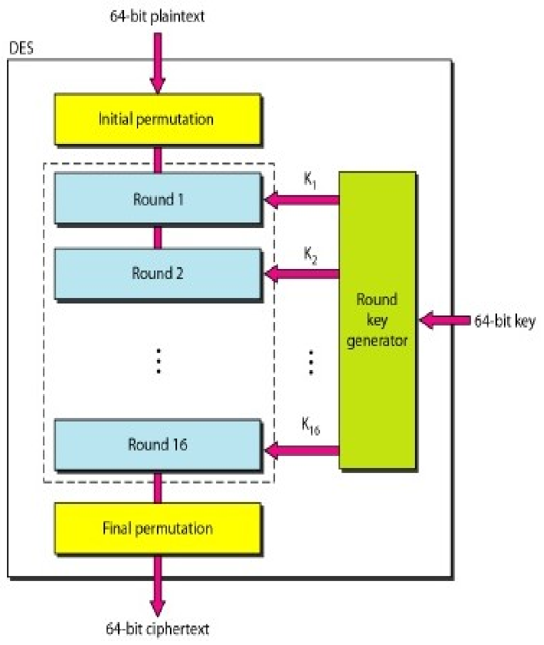
Data Encryption Standard (DES)

One example of a complex block cipher is the **Data Encryption Standard** (DES). DES was designed by IBM and adopted by the U.S. government as the standard encryption method for non-military and non-classified use. The algorithm encrypts a 64-bit plaintext block using a 64-bit key, as shown in Figure 30.13.

DES has two transposition blocks (P-boxes) and 16 complex round ciphers (they are repeated). Although the 16 iteration round ciphers are conceptually the same, each uses a different key derived from the original key.

The initial and final permutations are keyless straight permutations that are the inverse of each other. The permutation takes a 64-bit input and permutes them according to predefined values.

Figure 30.13 ***DES***



Each round of DES is a complex round cipher, as shown in Figure 30.14. Note that the structure of the encryption round ciphers is different from that of the decryption one.

Advanced Encryption Standard (AES)

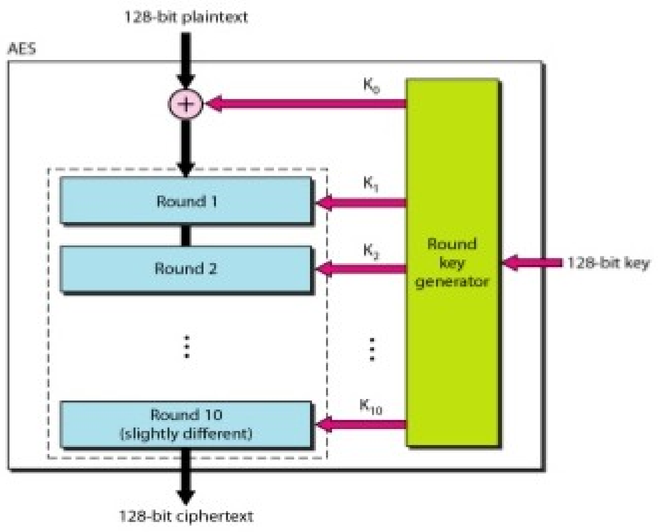
The Advanced Encryption Standard (AES) was designed because DES's key was too small. Although Triple DES ODES) increased the key size, the process was too slow. The National Institute of Standards and Technology (NIST) chose the Rijndael algorithm, named after its two Belgian inventors, Vincent Rijmen and Joan Daemen, as the basis of AES. AES is a very complex round cipher. AES is designed with three key sizes: 128, 192, or 256 bits. Table 30.1 shows the relationship between the data block, number of rounds, and key size.

Table 30.1 ***AES configuration***

Size of Data Block	*Number of Rounds*	*Key Size*
128 bits	10	128 bits
	12	192 bits
	14	256 bits

The general structure is shown in Figure 30.17. There is an initial XOR operation followed by 10 round ciphers. The last round is slightly different from the preceding rounds; it is missing one operation. Although the 10 iteration blocks are almost identical, each uses a different key derived from the original key.

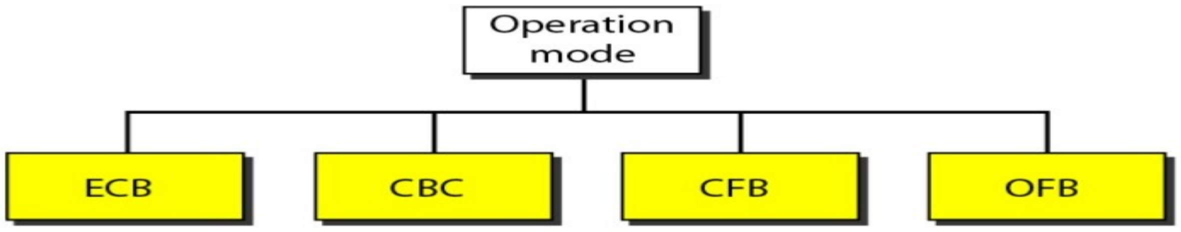
Figure 30.17 ***AES***



Mode of Operation

A mode of operation is a technique that employs the modern block ciphers such as DES and AES.

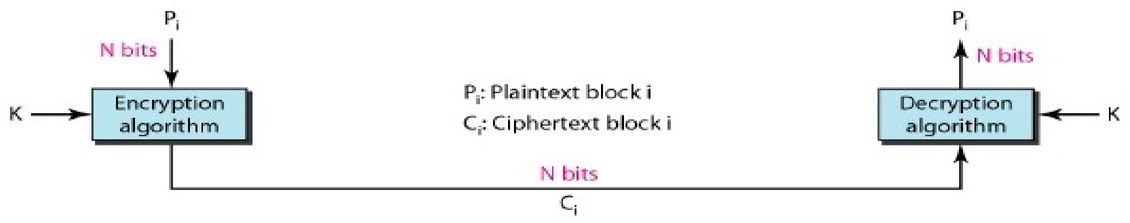
Figure 30.19 ***Modes of operation for block ciphers***



Electronic Code Book

The electronic code book (ECB) mode is a purely block cipher technique. The plaintext is divided into blocks of *N* bits. The cipher text is made of blocks of *N* bits. The value of *N* depends on the type of cipher used. Figure 30.20 shows the method.

Figure 30.20 ***ECB mode***



The 4 characteristics of this mode:

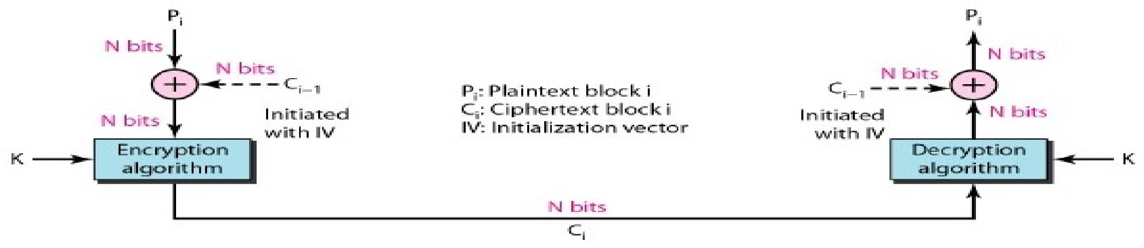
1. Because the key and the encryption/decryption algorithm are the same, equal blocks in the plaintext become equal blocks in the ciphertext. For example, if plaintext blocks 1, 5, and 9 are the same, ciphertext blocks I, 5, and 9 are also the same. This can be a security problem; the adversary can guess that the plaintext blocks are the same if the corresponding ciphertext blocks are the same.
2. If we reorder the plaintext block, the ciphertext is also reordered.
3. Blocks are independent of each other. Each block is encrypted or decrypted independently. A problem in encryption or decryption of a block does not affect other blocks.
4. An error in one block is not propagated to other blocks. If one or more bits are corrupted during transmission, it only affects the bits in the corresponding plaintext after decryption. Other plaintext blocks are not affected. This is a real advantage if the channel is not noise-free.

Cipher Block Chaining

The cipher block chaining (CBC) mode tries to alleviate some of the problems in ECB by including the previous cipher block in the preparation of the current block. If the current block is *i,* the previous ciphertext block C_{i-1} is included in the encryption of block *i.*

In other words, when a block is completely enciphered, the block is sent, but a copy of it is kept in a register (a place where data can be held) to be used in the encryption of the next block. The reader may wonder about the initial block. There is no ciphertext block before the first block. In this case, a phony block called the initiation vector (IV) is used. Both the sender and receiver agree upon a specific predetermined IV. In other words, the IV is used instead of the nonexistent CO, Figure 30.21 shows the CBC mode.

Figure 30.21 ***CBC mode***



The following are the characteristics of CBC.
1. Even though the key and the encryption/decryption algorithm are the same, equal blocks in the plaintext do not become equal blocks in the ciphertext. For example, if plaintext blocks 1, 5, and 9 are the same, ciphertext blocks 1, 5, and 9 will not be the same. An adversary will not be able to guess from the ciphertext that two blocks are the same.
2. Blocks are dependent on each other. Each block is encrypted or decrypted based on a previous block. A problem in encryption or decryption of a block affects other blocks.
3. The error in one block is propagated to the other blocks. If one or more bits are corrupted during the transmission, it affects the bits in the next blocks of the plaintext after decryption.

Cipher Feedback

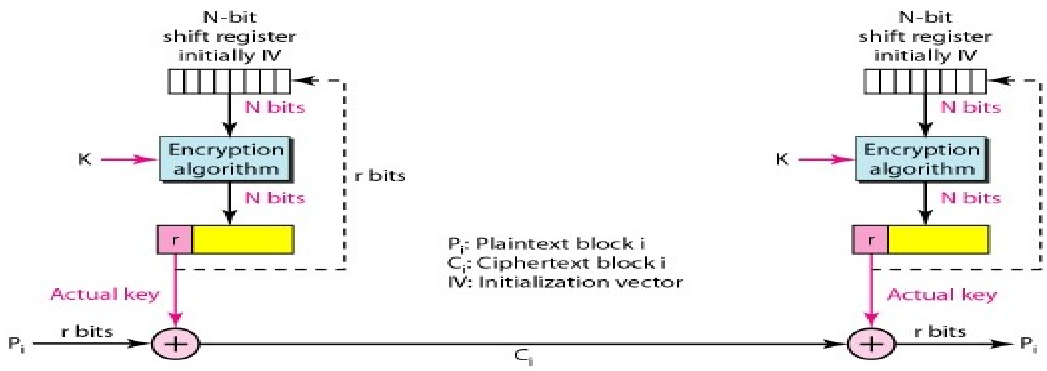
The cipher feedback (CFB) mode was created for those situations in which we need to send or receive *r* bits of data, where *r* is a number different from the underlying block size of the encryption cipher used. The value of *r* can be 1, 4, 8, or any number of bits. Since all block ciphers work on a block of data at a time, the problem is how to encrypt just *r* bits. The

solution is to let the cipher encrypt a block of bits and use only the first *r* bits as a new key (stream key) to encrypt the *r* bits of user data. Figure 30.22 shows the configuration.
The following are some characteristics of the CFB mode:
1. If we change the IV from one encryption to another using the same plaintext, the ciphertext is different.
2. The ciphertext Ci depends on both *Pi* and the preceding ciphertext block.
3. Errors in one or more bits of the ciphertext block affect the next ciphertext blocks.

Output Feedback

The **output** feedback (OFB) mode is very similar to the CFB mode with one difference. Each bit in the ciphertext is independent of the previous bit or bits. This avoids error propagation. If an error occurs in transmission, it does not affect the future bits. As in CFB, both the sender and the receiver use the encryption algorithm. Note also that in OFB, block ciphers such as DES or AES can only be used to create the key stream. The feedback for creating the next bit stream comes from the previous bits of the key stream instead of the ciphertext. The ciphertext does not take part in creating the key stream. Figure 30.23 shows the OFB mode.

Figure 30.23 ***OFB mode***



The following are some of the characteristics of the OFB mode.
1. If we change the IV from one encryption to another using the same plaintext, the ciphertext will be different.
2. The ciphertext Ci depends on the plaintext *Pi'*
3. Errors in one or more bits of the ciphertext do not affect future ciphertext blocks.

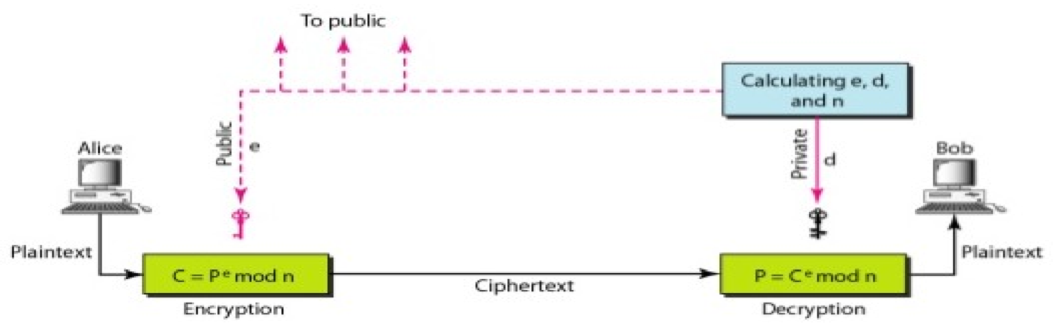
ASYMMETRIC-KEY CRYPTOGRAPHY

An asymmetric-key (or public-key) cipher uses two keys: **one private and one public**.
Two algorithms used here are: **RSA and Diffie-Hellman**.
RSA
The most common public key algorithm is RSA, named for its inventors Rivest, Shamir, and Adleman (RSA). It uses two numbers, e and d, as the public and private keys, as shown in Figure 30.24.
The two keys, *e* and *d,* have a special relationship to each other, a discussion of this relationship is beyond the scope of this book. We just show how to calculate the keys without proof.
Selecting Keys
Bob use the following steps to select the private and public keys:
1. Bob chooses two very large prime numbers *p* and *q*. Remember that a prime number is one that can be divided evenly only by 1 and itself.

2. Bob multiplies the above two primes to find *n,* the modulus for encryption and decryption. In other words, *n* ::: *p* X *q*.
3. Bob calculates another number <1> ::: *(p* -1) X *(q* - 1).
4. Bob chooses a random integer *e.* He then calculates *d* so that *d* x *e::*: 1 mod <1>.
5. Bob announces *e* and *n* to the public; he keeps <1> and *d* secret.

In RSA, *e* and *n* are announced to the public; *d* and <I> are kept secret.

Figure 30.24 ***RSA***



Encryption
Anyone who needs to send a message to Bob can use nand *e.* For example, if Alice needs to send a message to Bob, she can change the message, usually a short one, to an integer. This is the plaintext. She then calculates the ciphertext, using *e* and *n.*

$$C=p^e(\text{mod n}) .$$

Alice sends C, the ciphertext, to Bob.

Decryption
Bob keeps and *d* private. When he receives the ciphertext, he uses his private key *d* to decrypt the message:

$$P= Cd(\text{mod n})$$

Restriction
For RSA to work, the value of *P* must be less than the value of *n.* If *P* is a large number, the plaintext needs to be divided into blocks to make *P* less than *n.*

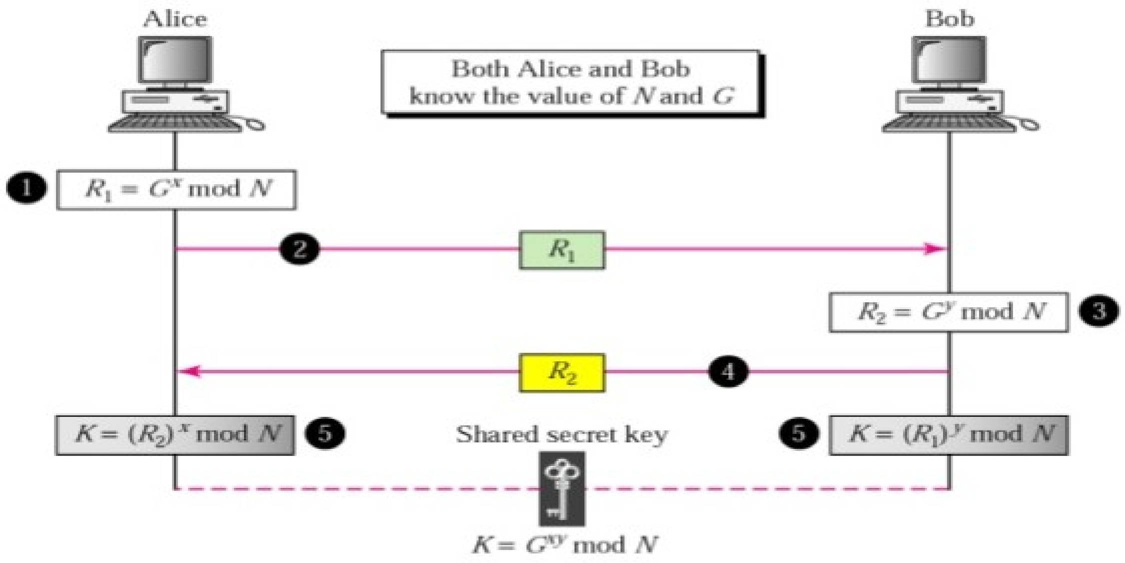
Applications
Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long. RSA is useful for short messages such as a small message digest or a symmetric key to be used for a symmetric-key cryptosystem.
RSA is used in digital signatures and other cryptosystems that often need to encrypt a small message without having access to a symmetric key. RSA is also used for authentication.

Diffie-Hellman
RSA is a public-key cryptosystem that is often used to encrypt and decrypt symmetric keys. Diffie-Hellman, on the other hand, was originally designed for key exchange. In the **Diffie-Hellman** cryptosystem, two parties create a symmetric session key to exchange data without having to remember or store the key for future use. They do not have to meet to agree on the key; it can be done through the Internet. Before establishing a symmetric key, the two parties need to choose two numbers *p* and *g.* The first number, *p,* is a large prime number on

the order of 300 decimal digits (1024 bits). The second number is a random number. These two numbers need not be confidential. They can be sent through the Internet; they can be public.

Procedure

Figure 30.26 shows the procedure. The steps are as follows:



- Step 1: Alice chooses a large random number *x* and calculates *R*1=g^x *mod p*
- Step 2: Bob chooses another large random number *y* and calculates *R2* = *gY* mod *p.*
- Step 3: Alice sends *R*1 to Bob. Note that Alice does not send the value of *x;* she sends only *R1-*
- Step 4: Bob sends *R2* to Alice. Again, note that Bob does not send the value of *y,* he sends only *R2.*
- Step 5: Alice calculates *K* = *(R2)*^{x}mod *p.*
- Step 6: Bob also calculates *K* = *(R1)*^{y}mod *p.*

The symmetric (shared) key in the Diffie-Hellman protocol is *K*=g^{xy} *mod p*

UNIT 5

APPLICATION LAYER: HTTP, FTP, SMTP, DNS

HTTP

The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web. HTTP functions as a combination of FTP and SMTP. It is similar to
FTP because it transfers files and uses the services of TCP. However, it is much simpler than FTP because it uses only one TCP connection. There is no separate control connection; only data are transferred between the client and the server.
HTTP is like SMTP because the data transferred between the client and the server look like SMTP messages. In addition, the format of the messages is controlled by MIME-like headers. Unlike SMTP, the HTTP messages are not destined to be read by humans; they are read and interpreted by the HTTP server and HTTP client (browser).
SMTP messages are stored and forwarded, but HTTP messages are delivered immediately. The commands from the client to the server are embedded in a request message.
The contents of the requested file or other information are embedded in a response message. HTTP uses the services of TCP on well-known port 80.

HTTP Transaction

Figure 27.12 illustrates the HTTP transaction between the client and server. Although HTTP uses the services of TCP, HTTP itself is a stateless protocol. The client initializes the transaction by sending a request message. The server replies by sending a response.

Messages

The formats of the request and response messages are similar; both are shown in Figure 27.13. A request message consists of a request line, a header, and sometimes a body. A response message consists of a status line, a header, and sometimes a body.

Figure 27.1 HTTP transaction

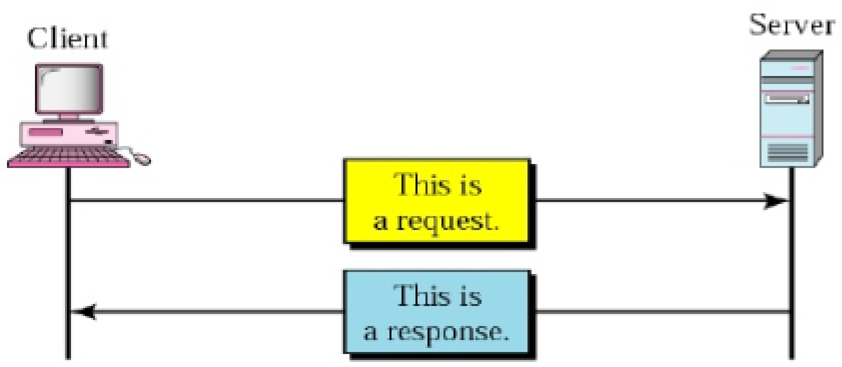
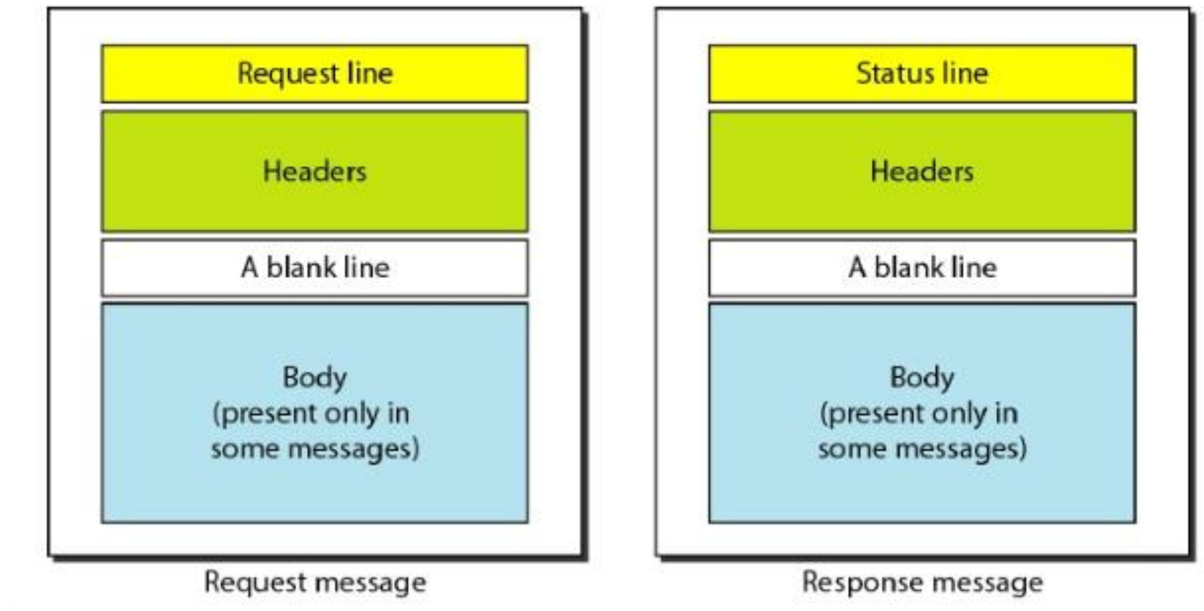
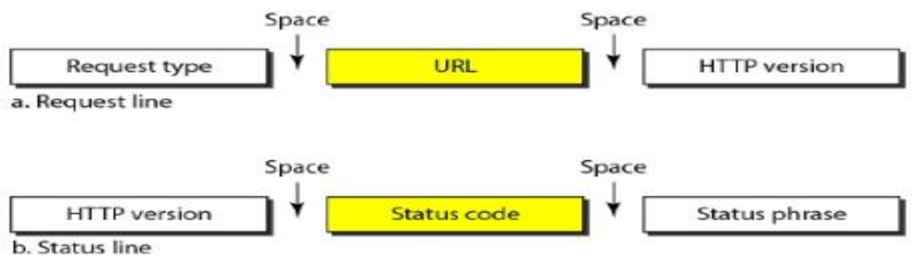


Figure 27.13 ***Request and response messages***



Request and Status Lines: The first line in a request message is called a request line; the first line in the response message is called the status line. There is one common field, as shown in Figure 27.14.

Figure 27.14 ***Request and status lines***



Request type: This field is used in the request message. In version 1.1 of HTTP, several request types are defined. The request type is categorized into *methods* as defined in Table 27.1.

Table 27.1 ***Methods***

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
POST	Sends some information from the client to the server
PUT	Sends a document from the server to the client
TRACE	Echoes the incoming request
CONNECT	Reserved
OPTION	Inquires about available options

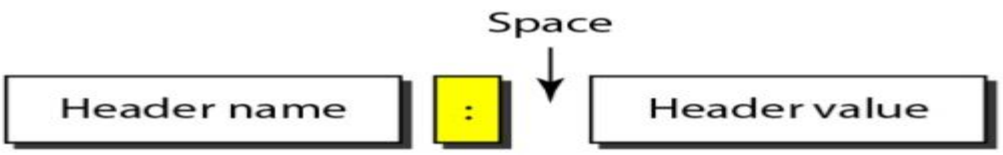
- **URL:**Uniform Resource Locator
- **Version.** The most current version of HTTP is 1.1.
- **Status code.** This field is used in the response message. The status code field is similar to those in the FTP and the SMTP protocols. It consists of three digits. Whereas the codes in the 100 range are only informational, the codes in the 200 range indicate a successful request. The codes in the 300 range redirect the client to another URL, and the codes in the 400 range indicate an error at the client site. Finally, the codes in the 500 range indicate an error at the server site.
- **Status phrase.** This field is used in the response message. It explains the status code in text form. Table 27.2 also gives the status phrase.

Header: The header exchanges additional information between the client and the server.
For eg, the client can request that the document be sent in a special format, or the server can send extra information about the document. The header can consist of one or more header lines. Each header line has a header name; a colon, a space, and a header value (Figure 27.15).
A header line belongs to one of four categories:

- General header
- Request header
- Response header
- Entity header.

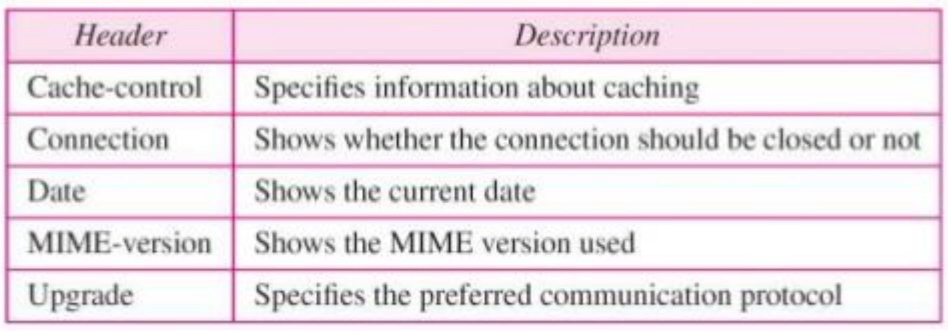
A request message can contain only general, request, and entity headers. A response message, on the other hand, can contain only general, response, and entity headers.

Figure 27.15 *Header format*



General header: The general header gives general information about the message and can be present in both a request and a response.

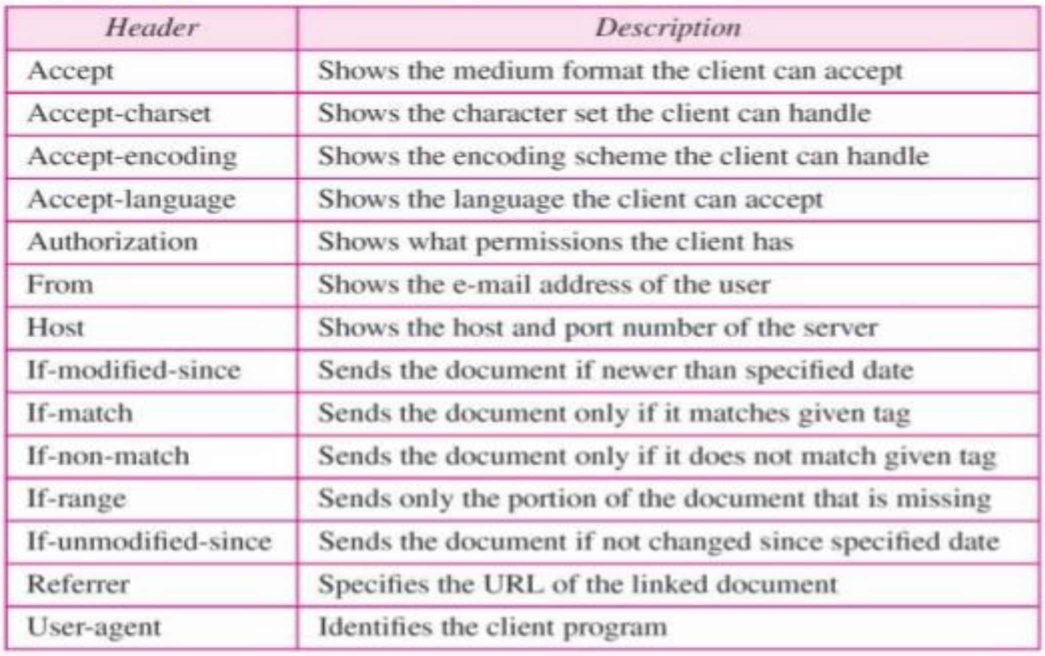
Table 27.3 *General headers*



Header	*Description*
Cache-control	Specifies information about caching
Connection	Shows whether the connection should be closed or not
Date	Shows the current date
MIME-version	Shows the MIME version used
Upgrade	Specifies the preferred communication protocol

Request header The request header can be present only in a request message. It specifies the client's configuration and the client's preferred document format.

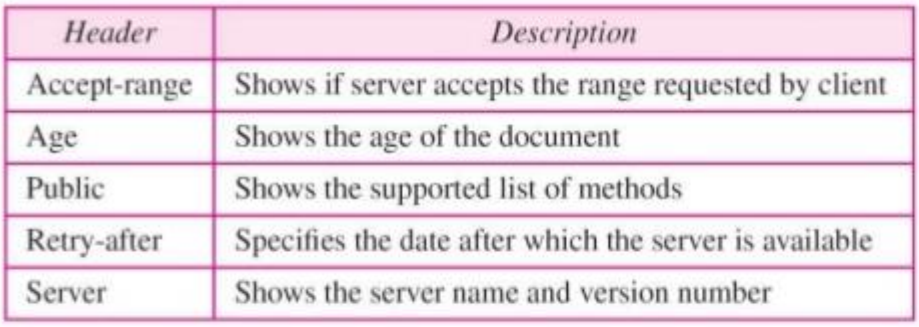
Table 27.4 ***Request headers***



Header	*Description*
Accept	Shows the medium format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
From	Shows the e-mail address of the user
Host	Shows the host and port number of the server
If-modified-since	Sends the document if newer than specified date
If-match	Sends the document only if it matches given tag
If-non-match	Sends the document only if it does not match given tag
If-range	Sends only the portion of the document that is missing
If-unmodified-since	Sends the document if not changed since specified date
Referrer	Specifies the URL of the linked document
User-agent	Identifies the client program

Response header: The response header can be present only in a response message. It specifies the server's configuration and special information about the request.

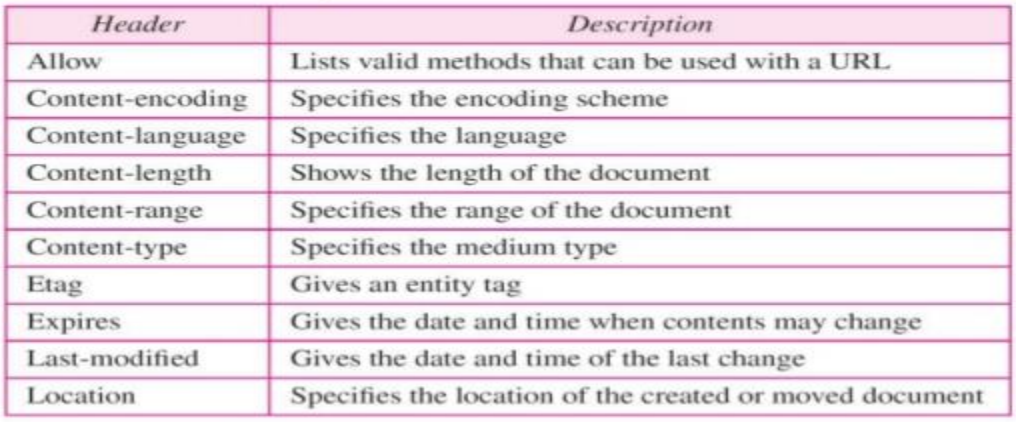
Table 27.5 ***Response headers***



Header	*Description*
Accept-range	Shows if server accepts the range requested by client
Age	Shows the age of the document
Public	Shows the supported list of methods
Retry-after	Specifies the date after which the server is available
Server	Shows the server name and version number

Entity header: The entity header gives information about the body of the document. Although it is mostly present in response messages, some request messages, such as POST or PUT methods, that contain a body also use this type of header. See Table 27.6 for a list of some entity headers and their descriptions.

Table 27.6 ***Entity headers***



Header	*Description*
Allow	Lists valid methods that can be used with a URL
Content-encoding	Specifies the encoding scheme
Content-language	Specifies the language
Content-length	Shows the length of the document
Content-range	Specifies the range of the document
Content-type	Specifies the medium type
Etag	Gives an entity tag
Expires	Gives the date and time when contents may change
Last-modified	Gives the date and time of the last change
Location	Specifies the location of the created or moved document

Body: The body can be present in a request or response message. Usually, it contains the document to be sent or received.

Persistent Versus Non-persistent Connection

HTTP prior to version 1.1 specified a non-persistent connection, while a persistent connection is the default in version 1.1.

Non-persistent Connection

In a non-persistent connection, one TCP connection is made for each request/response.
The following lists the steps in this strategy:
1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

In this strategy, for *N* different pictures in different files, the connection must be opened and closed *N* times. The non-persistent strategy imposes high overhead on the server because the server needs *N* different buffers and requires a slow start procedure each time a connection is opened.

Persistent Connection

HTTP version 1.1 specifies a persistent connection by default. In a persistent connection, the server leaves the connection open for more requests after sending a response. The server can close the connection at the request of a client or if a time-out has been reached. The sender usually sends the length of the data with each response. However, there are some occasions when the sender does not know the length of the data. This is the case when a document is created dynamically or actively. In these cases, the server informs the client that the length is not known and closes the connection after sending the data so the client knows that the end of the data has been reached.

FILE TRANSFER

Transferring files from one computer to another is one of the most common tasks expected from a networking or internetworking environment. As a matter of fact, the volume of data exchange in the Internet today is due to file transfer. A protocol involved in transferring files is: File Transfer Protocol *(FTP).*

File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is the standard mechanism provided by *TCP/IP* for copying a file from one host to another. Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different ways to represent text and data. Two systems may have different directory structures. All these problems have been solved by FTP in a very simple and elegant approach.

FTP differs from other client/server applications in that it establishes two connections between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication.

We need to transfer only a line of command or a line of response at a time. The data connection needs more complex rules due to the variety of data types transferred. However, the difference in complexity is at the FTP level, not TCP.
For TCP, both connections are treated the same. FTP uses two well-known TCP ports: Port 21 is used for the control connection, and port 20 is used for the data connection. Figure 26.21 shows the basic model of FTP.
The client has three components:

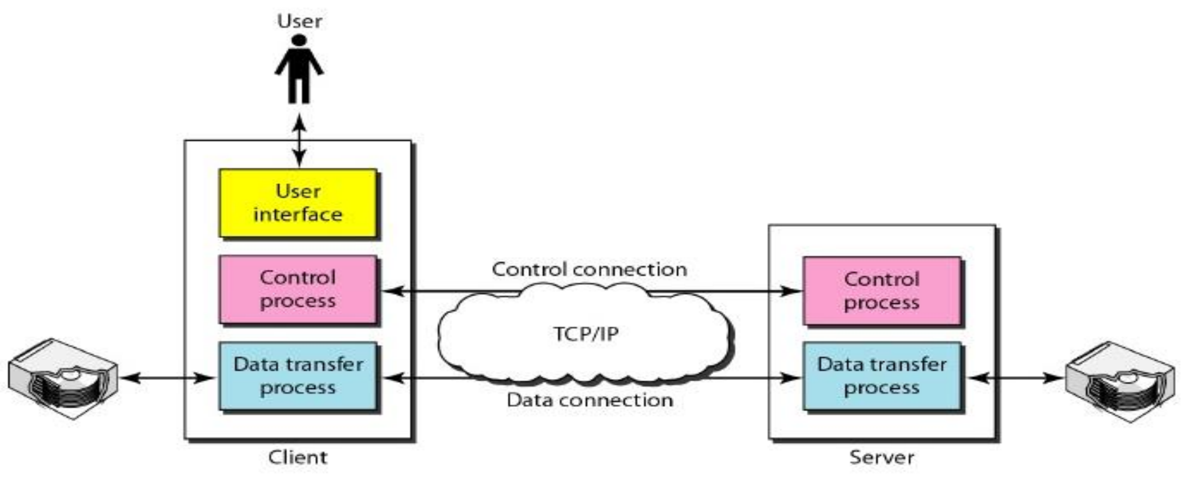
- User interface,
- Client control process,
- The client data transfer process.

The server has two components:

- The server control process and
- The server data transfer process.

The control connection is made between the control processes.
The data connection is made between the data transfer processes.

Figure 26.21 ***FTP***

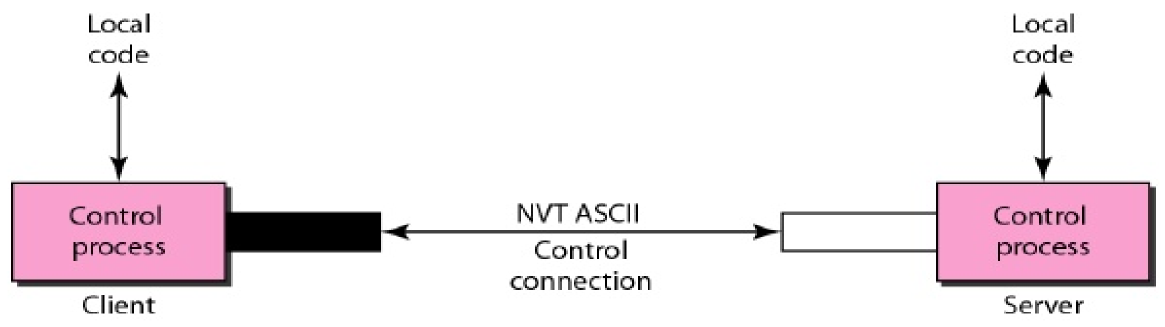


The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transferred. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. When a user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.

Communication over Control Connection

FTP uses the same approach as SMTP to communicate across the control connection. It uses the 7-bit ASCII character set (see Figure 26.22). Communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each command or response is only one short line, so we need not worry about file format or file structure. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

Figure 26.22 *Using the control connection*



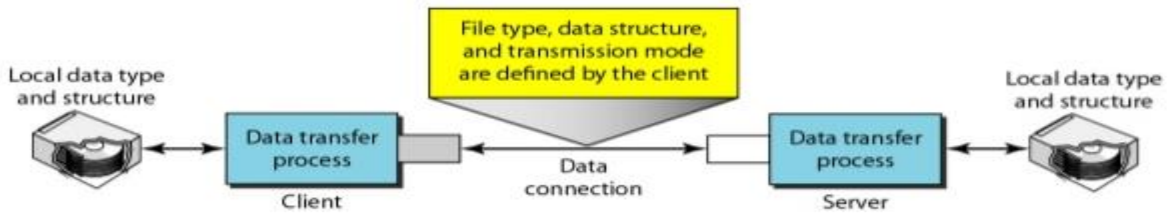
Communication over Data Connection

The purpose of the data connection is different from that of the control connection. File transfer occurs over the data connection under the control of the commands sent over the control connection.

- A file is to be copied from the server to the client. This is called *retrieving a file.* It is done under the supervision of the RETR command,
- A file is to be copied from the client to the server. This is called *storing a file.* It is done under the supervision of the STOR command.
- A list of directory or file names is to be sent from the server to the client. This is done under the supervision of the LIST command. Note that FTP treats a list of directory or file names as a file. It is sent over the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode. Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission mode (see Figure 26.23).

File Type FTP can transfer one of the following file types across the data connection: an ASCII file, EBCDIC file, or image file. The ASCII file is the default format for transferring text files. Each character is encoded using 7-bit ASCII. The sender transforms the file from its own representation into ASCII characters, and the receiver transforms the ASCII characters to its own representation. If one or both ends of the connection use EBCDIC encoding (the file format used by IBM), the file can be transferred using EBCDIC encoding. The image file is the default format for transferring binary files. The file is sent as continuous streams of bits without any interpretation or encoding. This is mostly used to transfer binary files such as compiled programs.

Figure 26.23 *Using the data connection*


File type, data structure, and transmission mode are defined by the client
Local data type and structure
Data transfer process
Client
Data connection
Data transfer process
Server
Local data type and structure

Data Structure FTP can transfer a file across the data connection by using one of the following interpretations about the structure of the data:

- file structure,
- record structure
- page structure.

In the **file structure** format, the file is a continuous stream of bytes.
In the **record structure**, the file is divided into records. This can be used only with text files.
In the **page structure**, the file is divided into pages, with each page having a page number and a page header. The pages can be stored and accessed randomly or sequentially.

Transmission Mode: FTP can transfer a file across the data connection by using one of the following three transmission modes:

- stream mode,
- block mode
- compressed mode.

The **stream mode** is the default mode. Data are delivered from FTP to TCP as a continuous stream of bytes. TCP is responsible for chopping data into segments of appropriate size. If the data are simply a stream of bytes (file structure), no end-of-file is needed. End-of-file in this case is the closing of the data connection by the sender. If the data are divided into records (record structure), each record will have a I-byte end-of- record (EOR) character and the end of the file will have a 1-byte end-of-file (EOF) character.
In **block mode**, data can be delivered from FTP to TCP in blocks. In this case, each block is preceded by a 3-byte header. The first byte is called the *block descriptor;* the next 2 bytes define the size of the block in bytes.
In the **compressed mode**, if the file is big, the data can be compressed. The compression method normally used is run-length encoding. In this method, consecutive appearances of a data unit are replaced by one occurrence and the number of repetitions. In a text file, this is usually spaces (blanks). In a binary file, null characters are usually compressed.

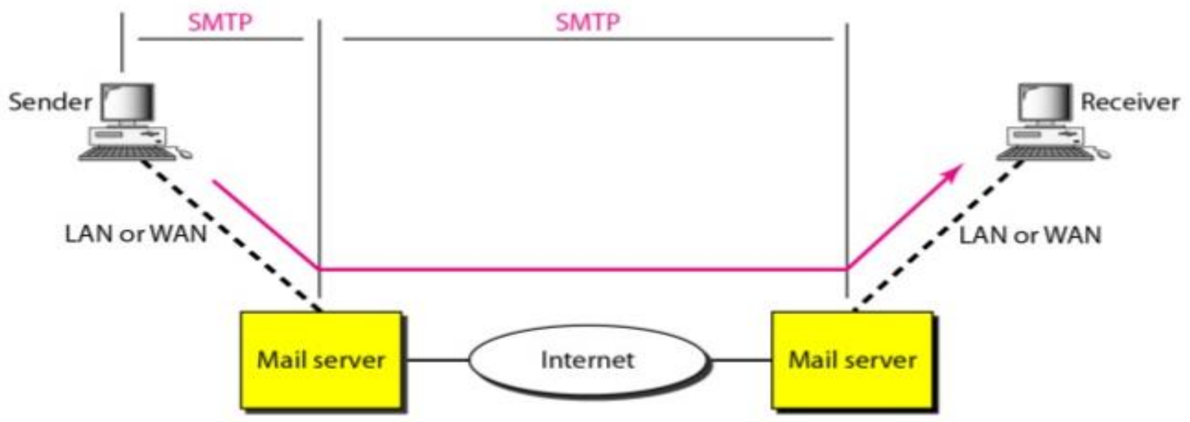
Anonymous FTP

To use FTP, a user needs an account (user name) and a password on the remote server. Some sites have a set of files available for public access, to enable anonymous FTP. To access these files, a user does not need to have an account or password. Instead, the user can use *anonymous* as the user name and *guest* as the password.
User access to the system is very limited. Some sites allow anonymous users only a subset of commands. For example, most sites allow the user to copy some files, but do not allow navigation through the directories.

SMTP

The actual mail transfer is done through message transfer agents. To send mail, a system must have the client MTA (Message Transfer Agent), and to receive mail, a system must have a server MTA. The formal protocol that defines the MTA client and server in the Internet is called **the Simple Mail Transfer Protocol (SMTP)**. Two pairs of MTA client/server programs are used in the most common situation (fourth scenario). Figure 26.16 shows the range of the SMTP protocol in this scenario.

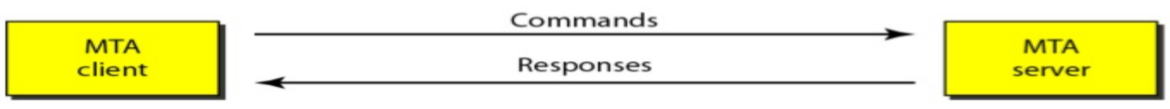
Figure 26.16 ***SMTP range***



SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. A protocol is needed between the mail server and the receiver. SMTP simply defines how commands and responses must be sent back and forth. Each network is free to choose a software package for implementation.

Commands and Responses
SMTP uses commands and responses to transfer messages between an MTA client and an MTA server.

Figure 26.17 ***Commands and responses***



Each command or reply is terminated by a two-character (carriage return and line feed) end-of-line token.
Commands: Commands are sent from the client to the server. The format of a command is shown in Figure 26.18. It consists of a keyword followed by zero or more arguments.
SMTP defines 14 commands. The first five are mandatory; every implementation must support these five commands. The next three are often used and highly recommended. The last six are seldom used.

Figure 26.18 ***Command format***

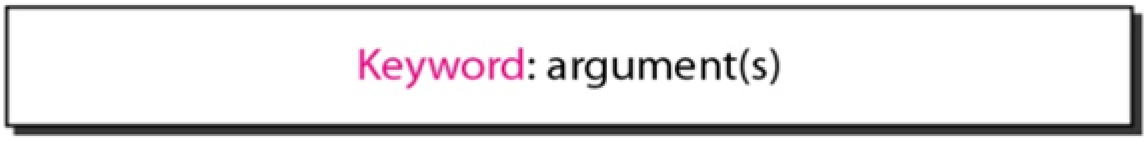
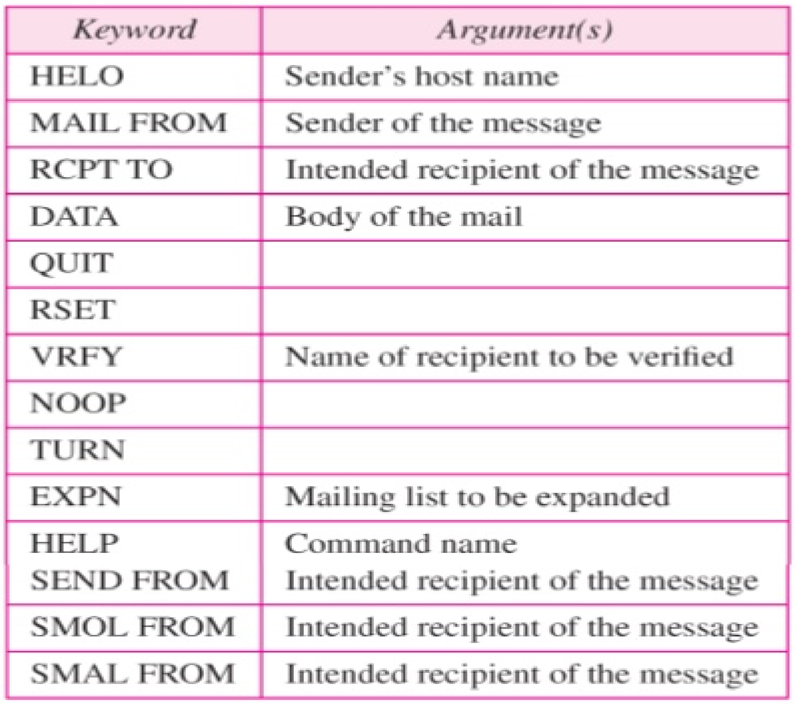


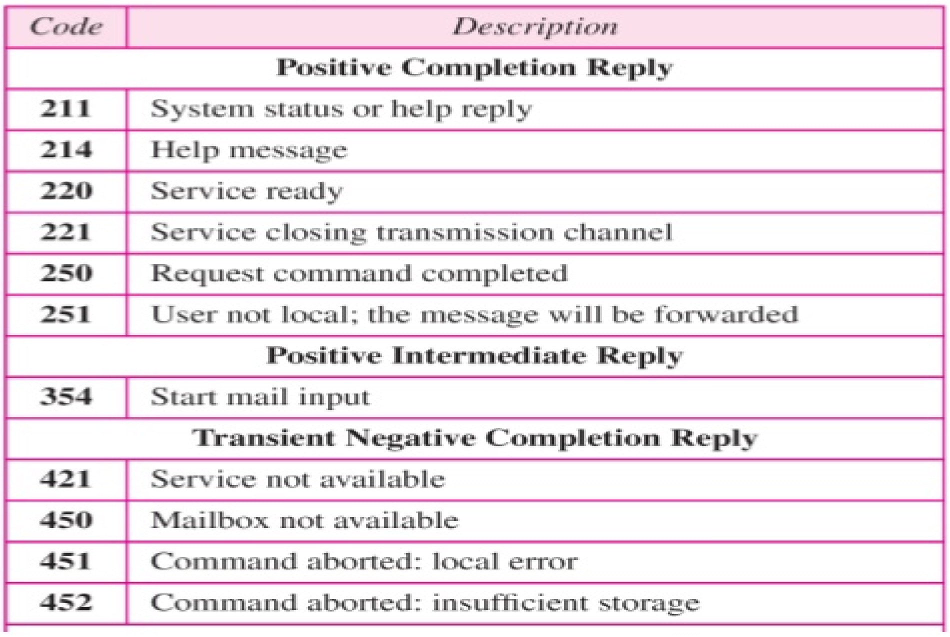
Table 26.7 ***Commands***



Keyword	*Argument(s)*
HELO	Sender's host name
MAIL FROM	Sender of the message
RCPT TO	Intended recipient of the message
DATA	Body of the mail
QUIT	
RSET	
VRFY	Name of recipient to be verified
NOOP	
TURN	
EXPN	Mailing list to be expanded
HELP	Command name
SEND FROM	Intended recipient of the message
SMOL FROM	Intended recipient of the message
SMAL FROM	Intended recipient of the message

Responses: Responses are sent from the server to the client. A response is a three digit code that may be followed by additional textual information. Table 26.8 lists some of the responses.

Table 26.8 ***Responses***



Code	*Description*
	Positive Completion Reply
211	System status or help reply
214	Help message
220	Service ready
221	Service closing transmission channel
250	Request command completed
251	User not local; the message will be forwarded
	Positive Intermediate Reply
354	Start mail input
	Transient Negative Completion Reply
421	Service not available
450	Mailbox not available
451	Command aborted: local error
452	Command aborted: insufficient storage

Domain Name System

There are several applications in the application layer of the Internet model that follow the client/server paradigm. The client/server programs can be divided into two categories:
1) Those that can be directly used by the user, such as e-mail,
2) Those that support other application programs.
The Domain Name System (DNS) is a supporting program that is used by other programs such as e-mail.
When the Internet was small, mapping was done by using a host file. The host file had only two columns:

- Name
- Address.

Every host could store the host file on its disk and update it periodically from a master host file. When a program or a user wanted to map a name to an address, the host consulted the host file and found the mapping.

Today, it is impossible to have one single host file to relate every address with a name and vice versa. The host file would be too large to store in every host. In addition, it would be impossible to update all the host files every time there was a change.

One solution would be to store the entire host file in a single computer and allow access to this centralized information to every computer that needs mapping. But this would create a huge amount of traffic on the Internet.

Another solution, the one used today, is to divide this huge amount of information into smaller parts and store each part on a different computer. In this method, the host that needs mapping can contact the closest computer holding the needed information.
This method is used by the **Domain Name System (DNS).**

NAME SPACE

The names must be unique because the addresses are unique. A name space that maps each address to a unique name can be organized in two ways:

- flat
- hierarchical.

Flat Name Space

In a flat name space, a name is assigned to an address. A name in this space is a sequence of characters without structure. The names may or may not have a common section; if they do, it has no meaning.
The main **disadvantage** of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication.

Hierarchical Name Space

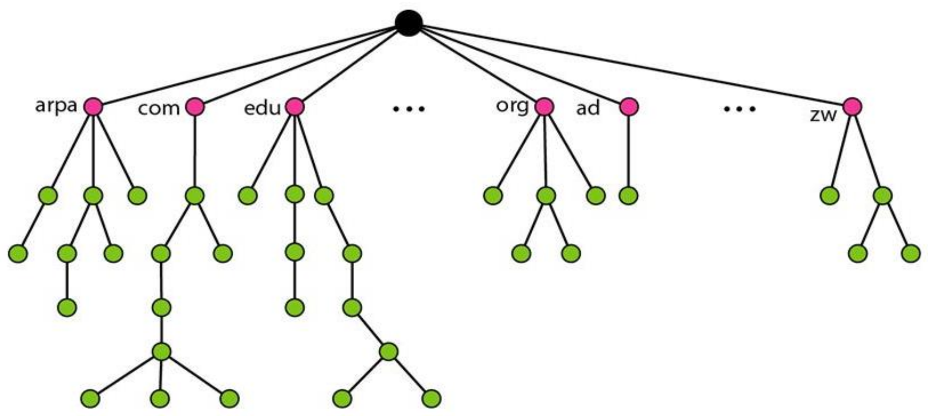
In a hierarchical name space, each name is made of several parts.
The **first part** can define **the nature of the organization,** the **second part** can define the **name** of an organization,the **third** part can define **departments** in the organization, and so on.
In this case, the authority to assign and control the name spaces can be decentralized. A central authority can assign the part of the name that defines the nature of the organization and the name of the organization. The responsibility of the rest of the name can be given to

the organization itself. The organization can add suffixes (or prefixes) to the name to define its host or resources.
The management of the organization need not worry that the prefix chosen for a host is taken by another organization because, even if part of an address is the same, the whole address is different.
For example, assume two colleges and a company call one of their computers *challenger*. The first college is given a name by the central authority such as *jhda.edu,* the second college is given the name *berkeley.edu,* and the company is given the name *smart. com.* When these organizations add the name *challenger* to the name they have already been given, the end result is three distinguishable names: *challenger.jhda.edu, challenger.berkeley.edu,* and *challenger.smart.com.* The names are unique without the need for assignment by a central authority. The central authority controls only part of the name, not the whole.

DOMAIN NAME SPACE

To have a hierarchical name space, a domain name space was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 (see Figure 25.2).

Figure 25.2 ***Domain name space***



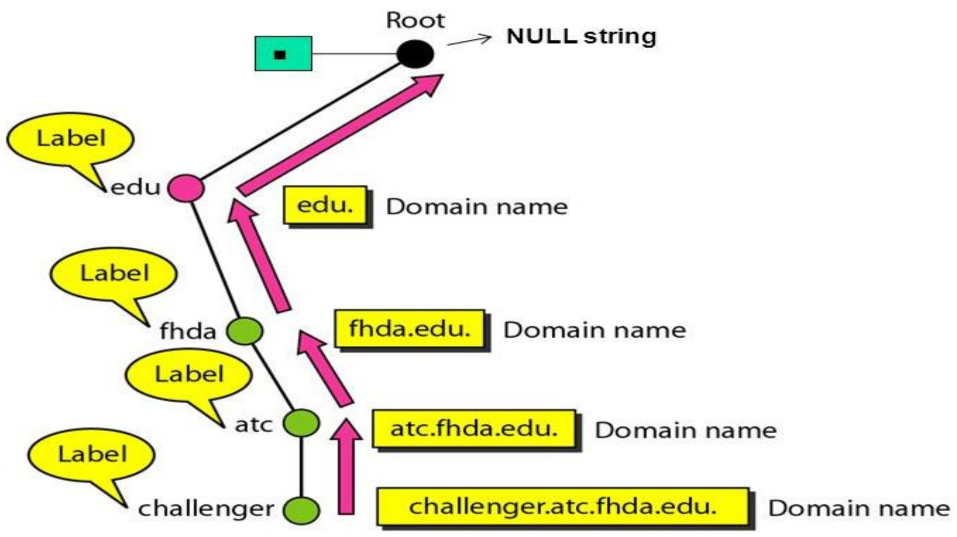
Label

Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

Domain Name

Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing. Figure 25.3 shows some domain names.

Figure 25.3 *Domain names and labels*



Root
NULL string
.
Label
edu
edu.
Domain name
Label
fhda
fhda.edu.
Domain name
Label
atc
atc.fhda.edu.
Domain name
Label
challenger
challenger.atc.fhda.edu.
Domain name