Data Communications and Networking Fourth Edition

Forouzan

# Chapter 30

# Cryptography

# 30-1 INTRODUCTION

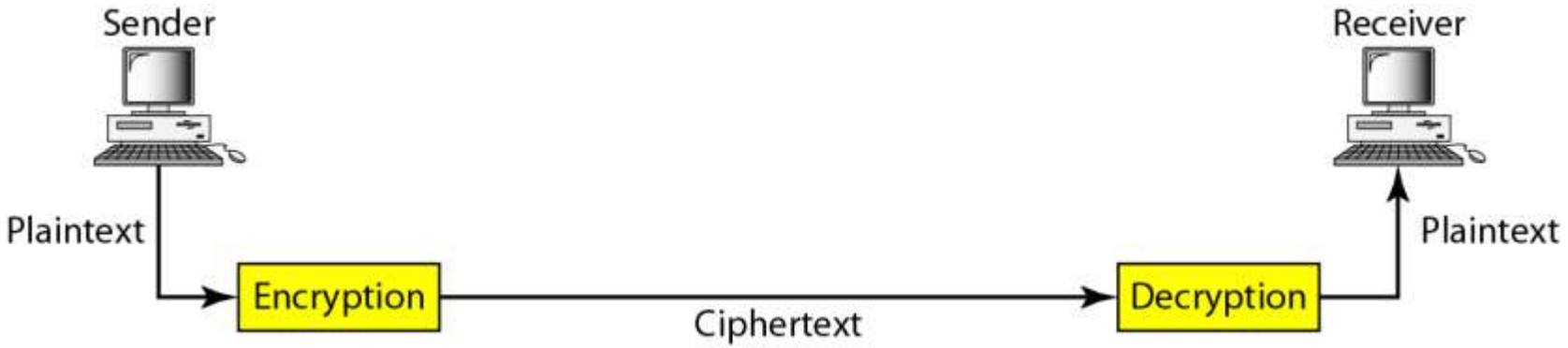
***Let us introduce the issues involved in cryptography. First, we need to define some terms; then we give some taxonomies.***

***Topics discussed in this section:***

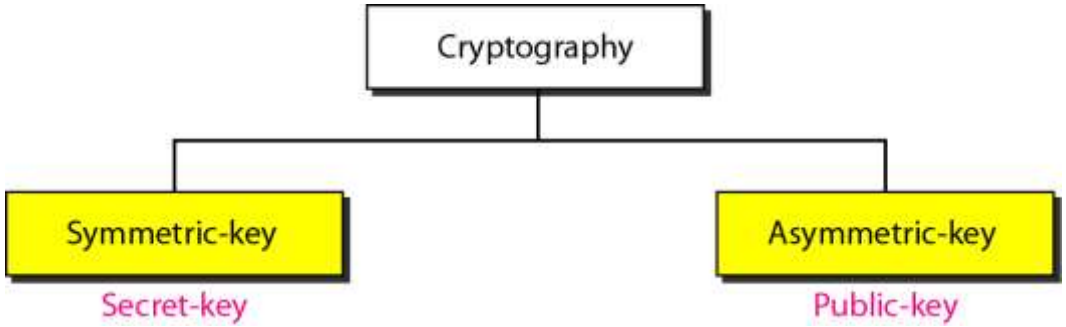
**Definitions**

**Two Categories**

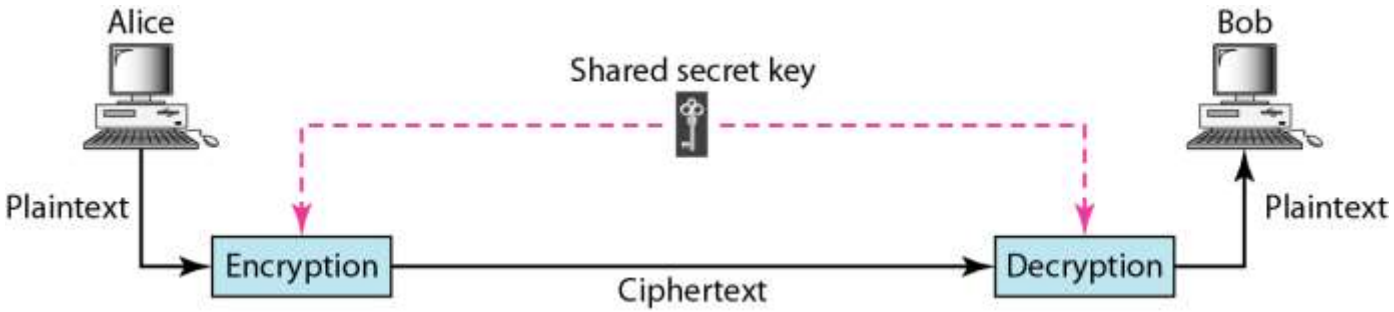
## Figure 30.1 *Cryptography components*



## Figure 30.2 *Categories of cryptography*



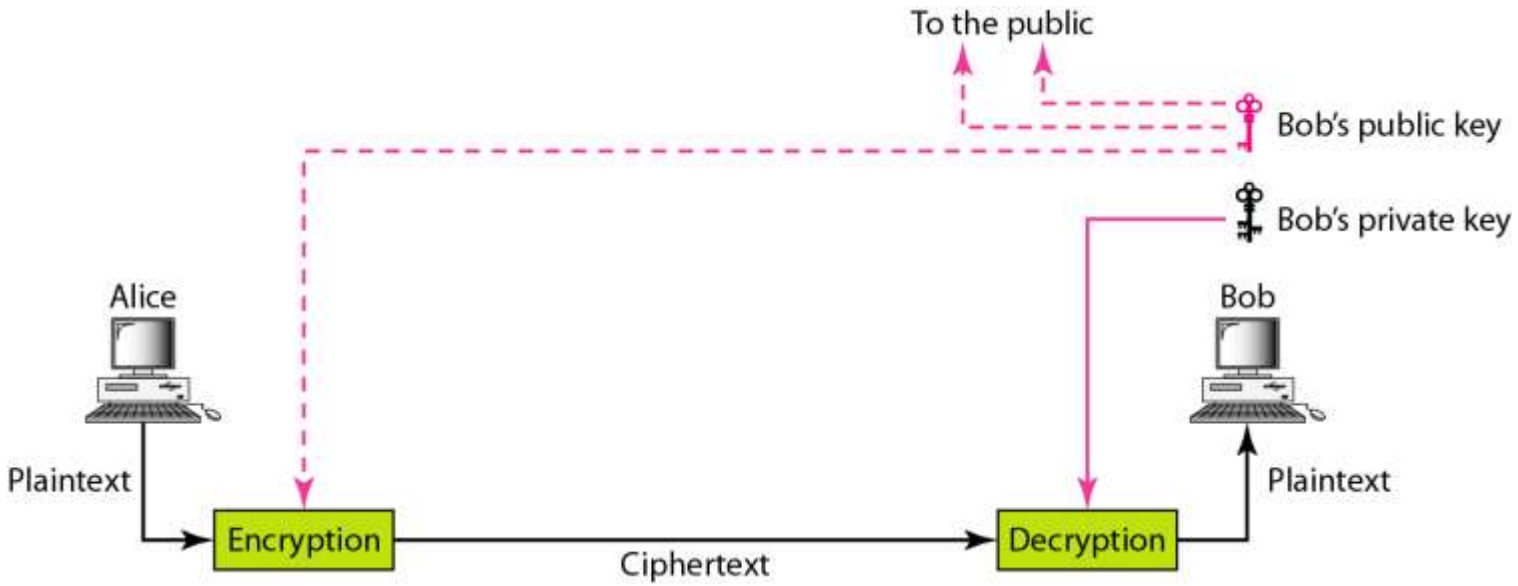
## Figure 30.3 *Symmetric-key cryptography*



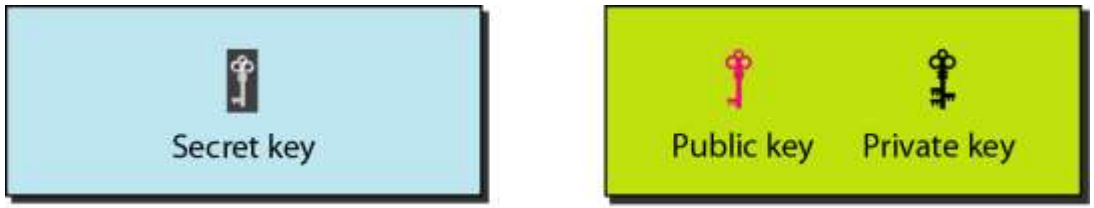
*Note*

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

## Figure 30.4 *Asymmetric-key cryptography*



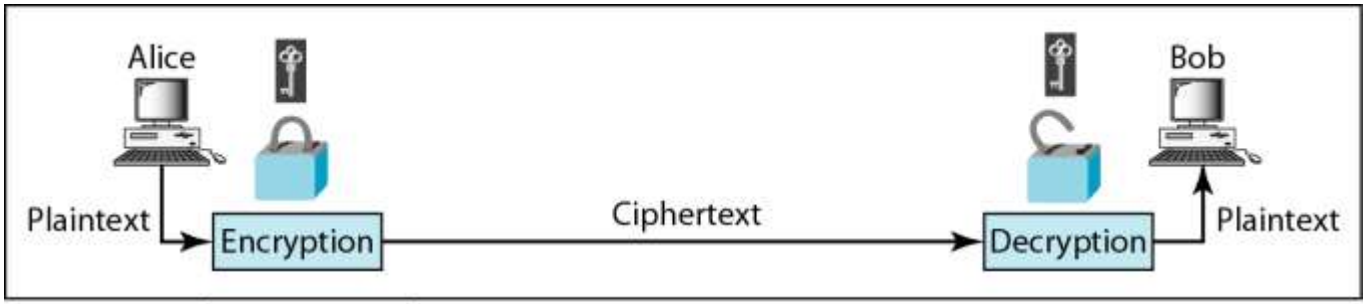
## Figure 30.5 *Keys used in cryptography*



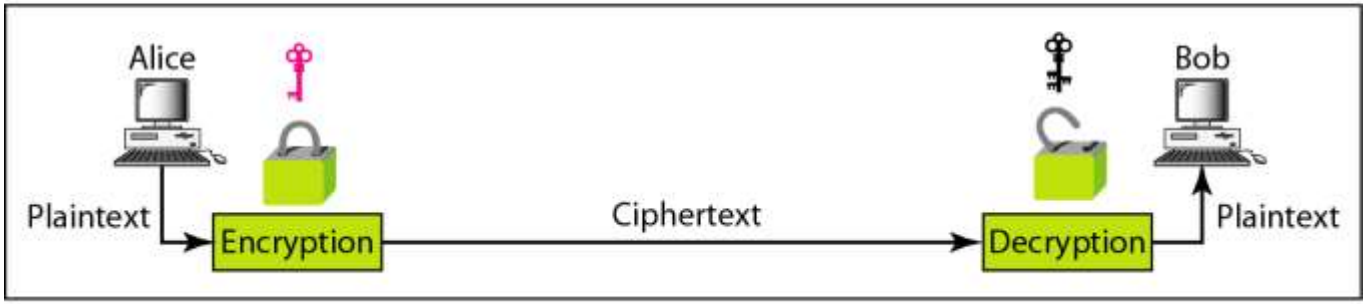

Symmetric-key cryptography

Asymmetric-key cryptography

## Figure 30.6 *Comparison between two categories of cryptography*




a. Symmetric-key cryptography




b. Asymmetric-key cryptography

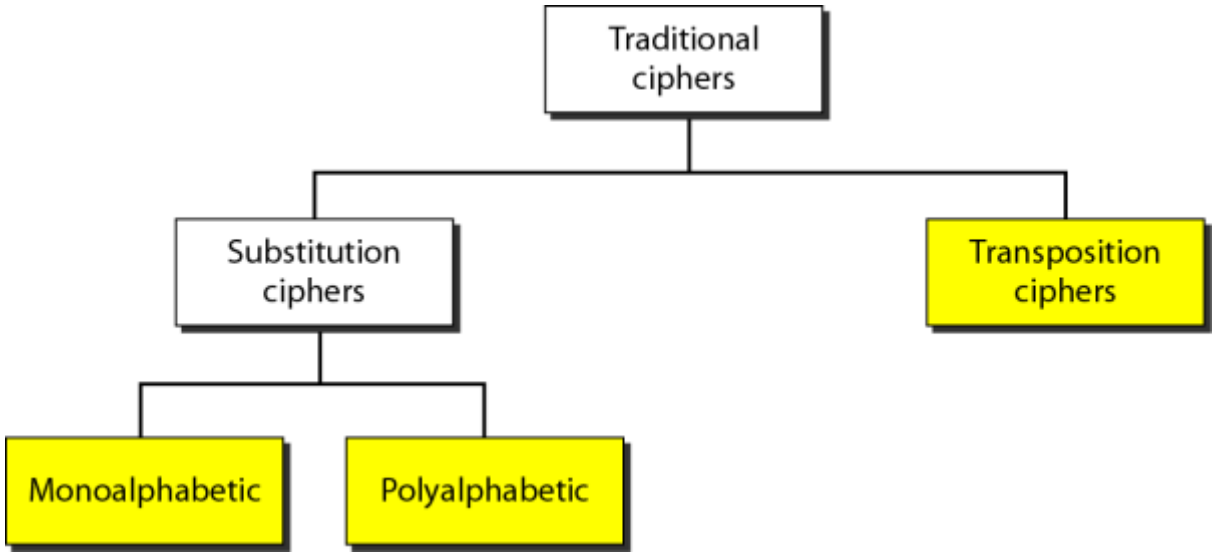
# 30-2 SYMMETRIC-KEY CRYPTOGRAPHY

***Symmetric-key cryptography started thousands of years ago when people needed to exchange secrets (for example, in a war). We still mainly use symmetric-key cryptography in our network security.***

***Topics discussed in this section:***

**Traditional Ciphers**
**Simple Modern Ciphers**
**Modern Round Ciphers**
**Mode of Operation**

## Figure 30.7 *Traditional ciphers*



*Note*

**A substitution cipher replaces one symbol with another.**

# Example 30.1

*The following shows a plaintext and its corresponding ciphertext. Is the cipher monoalphabetic?*

**Plaintext:** HELLO
**Ciphertext:** KHOOR

## Solution

*The cipher is probably monoalphabetic because both occurrences of L's are encrypted as O's.*

# Example 30.2

*The following shows a plaintext and its corresponding ciphertext. Is the cipher monoalphabetic?*

**Plaintext:** HELLO
**Ciphertext:** ABNZF

## Solution

*The cipher is not monoalphabetic because each occurrence of L is encrypted by a different character. The first L is encrypted as N; the second as Z.*

*Note*

**The shift cipher is sometimes referred to as the Caesar cipher.**

# Example 30.3

*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 O is encrypted to D. The cipher text is WTAAD.*

# Example 30.4

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

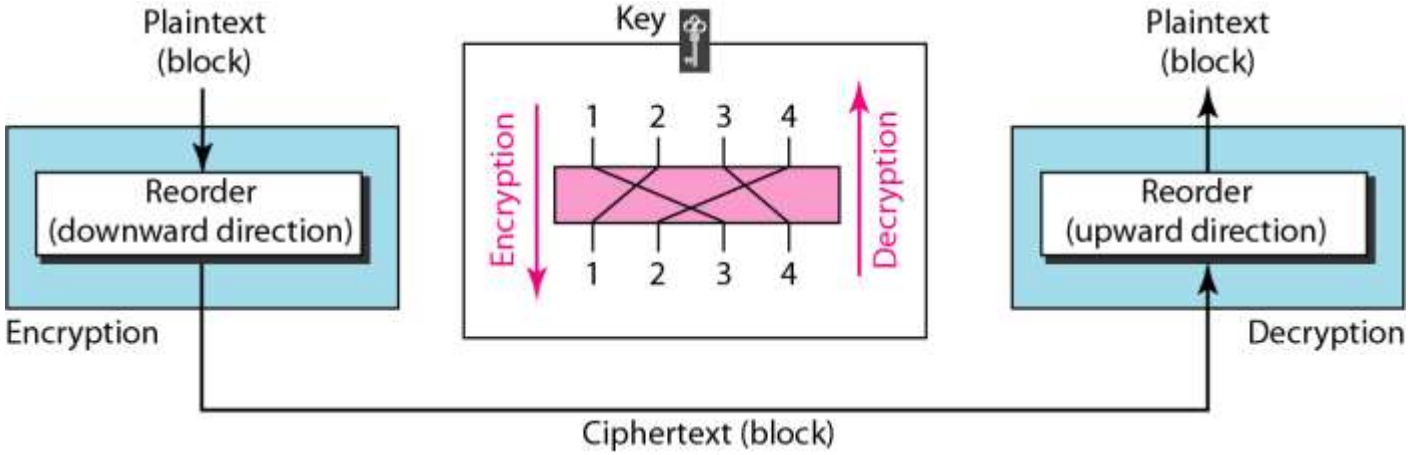
**Solution**

*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.*

*Note*

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

## Figure 30.8 *Transposition cipher*



# Example 30.5

***Encrypt the message "HELLO MY DEAR," using the key shown in Figure 30.8.***

## 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.***

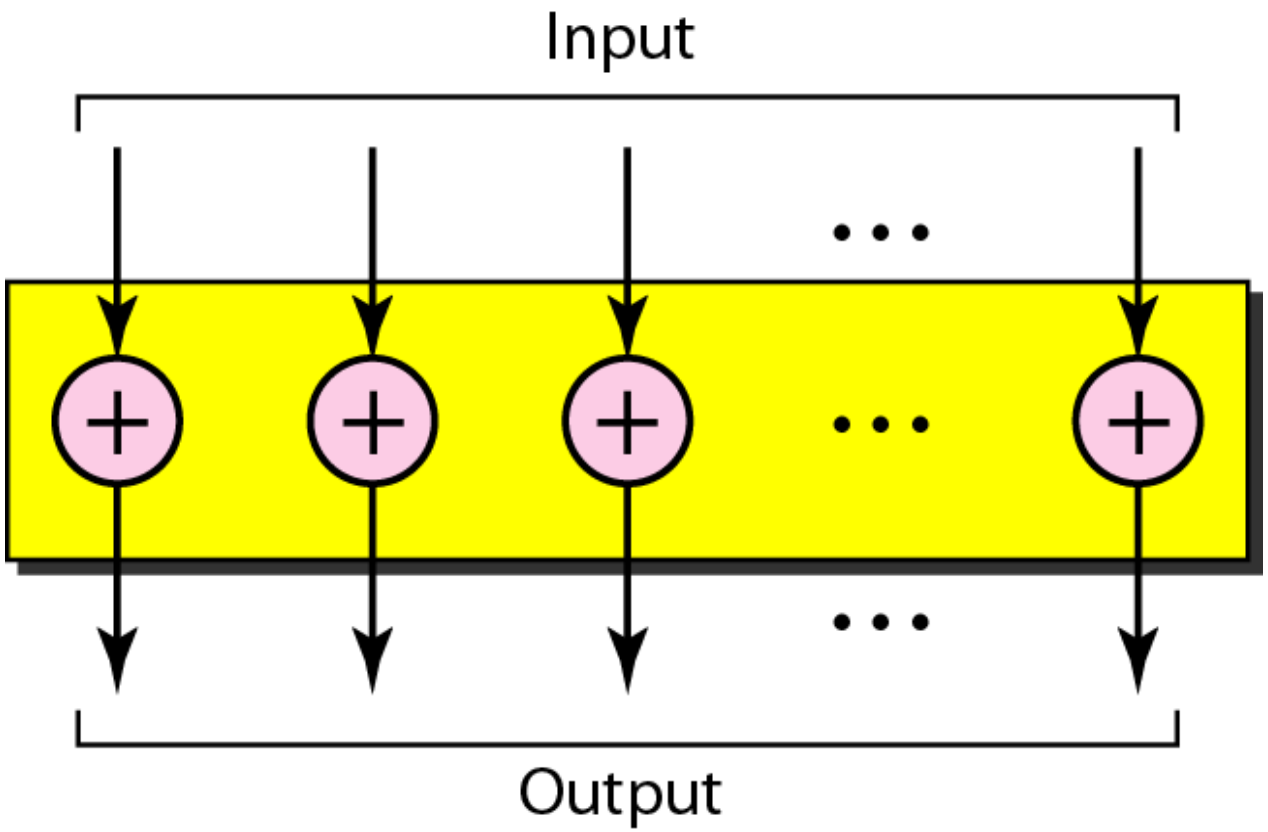
# Example 30.6

*Using Example 30.5, decrypt the message "ELHLMDOYAZER".*

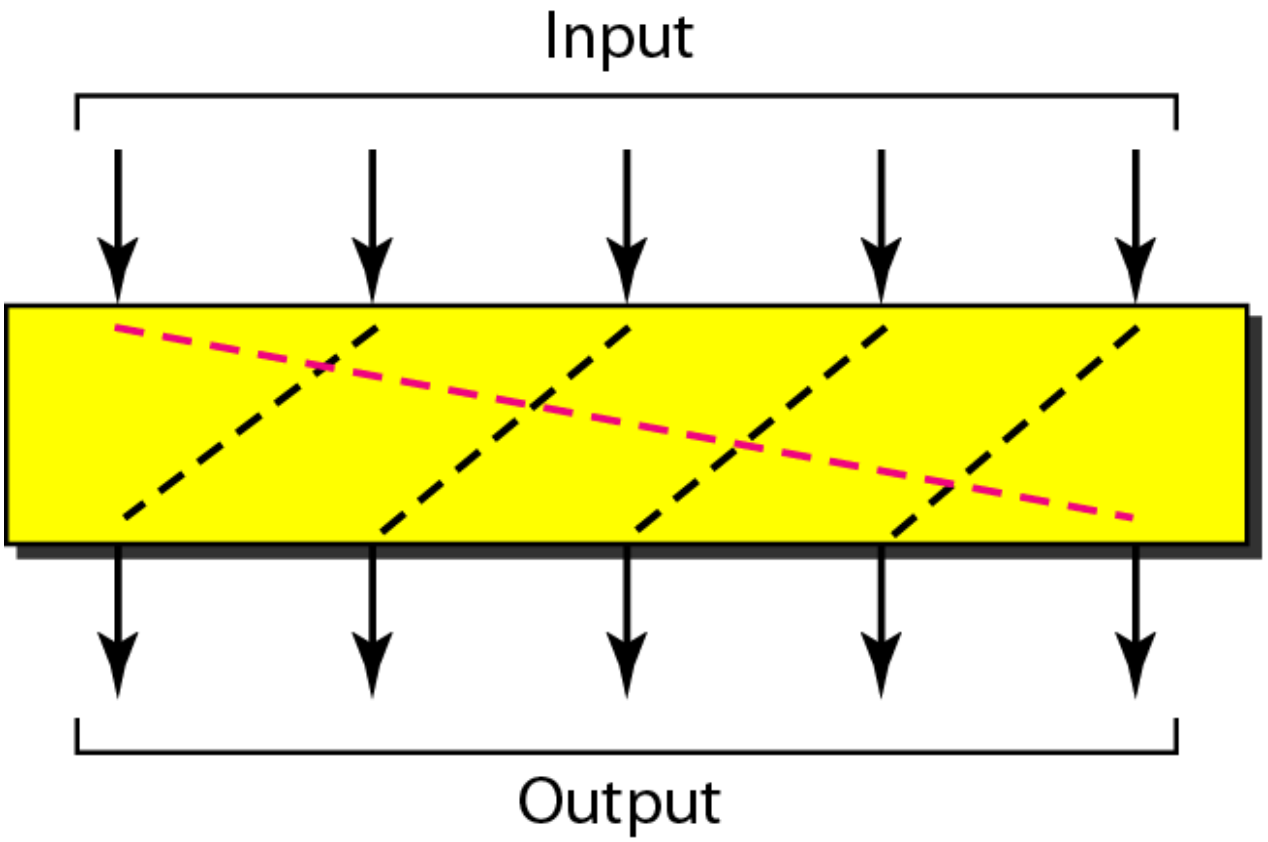
**Solution**

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

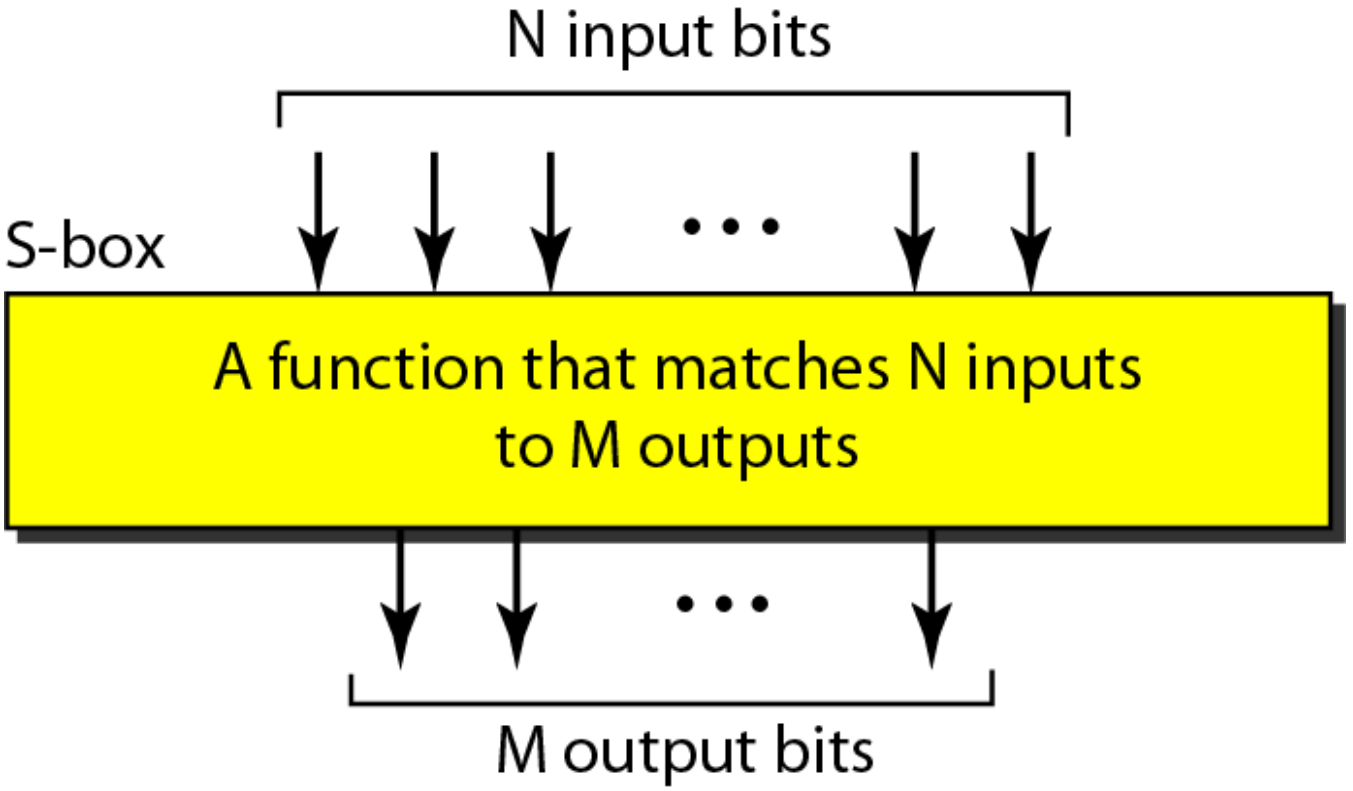
## Figure 30.9 *XOR cipher*



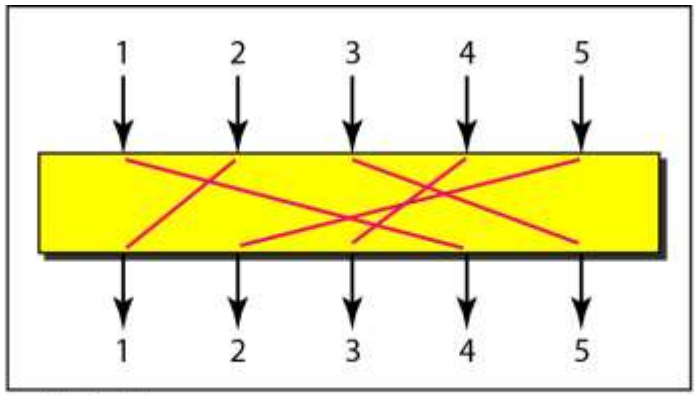
## Figure 30.10 *Rotation cipher*



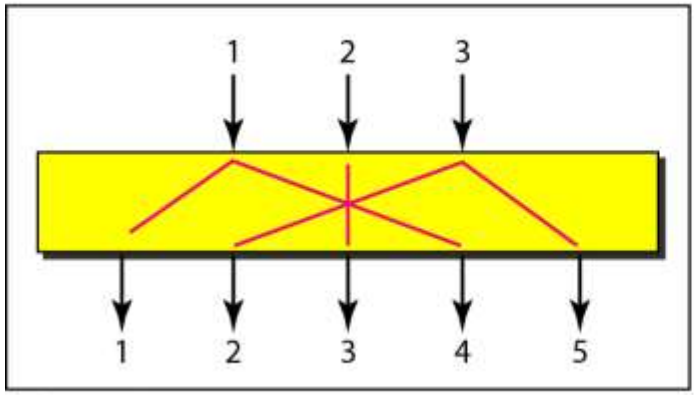
## Figure 30.11 *S-box*



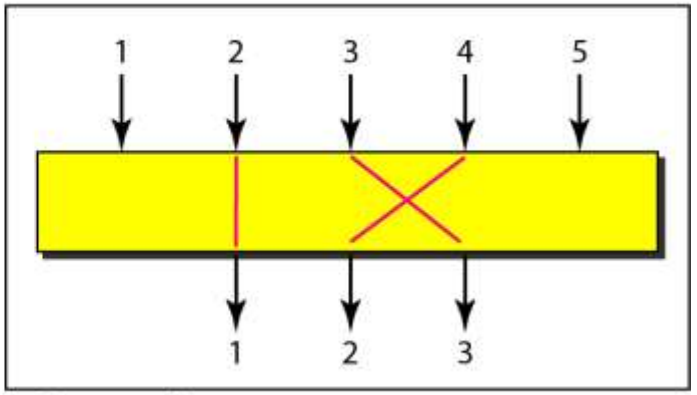
## Figure 30.12 *P-boxes: straight, expansion, and compression*




a. Straight

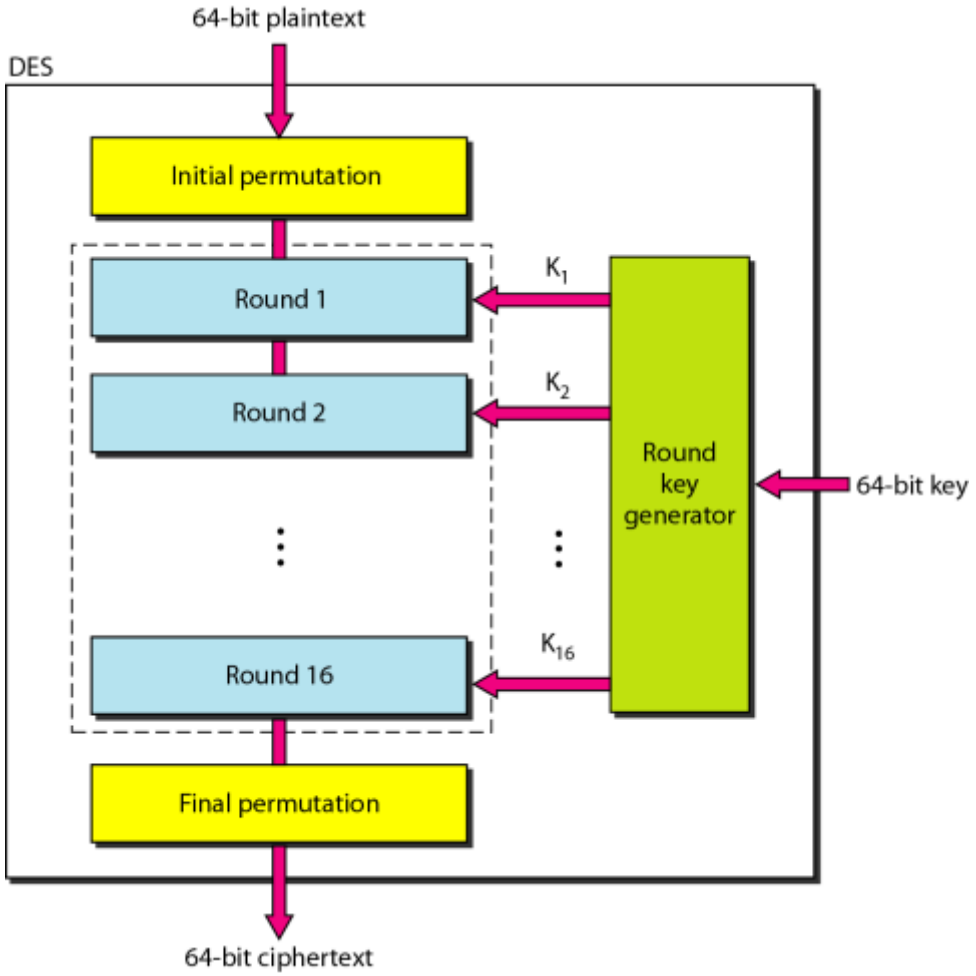



b. Expansion

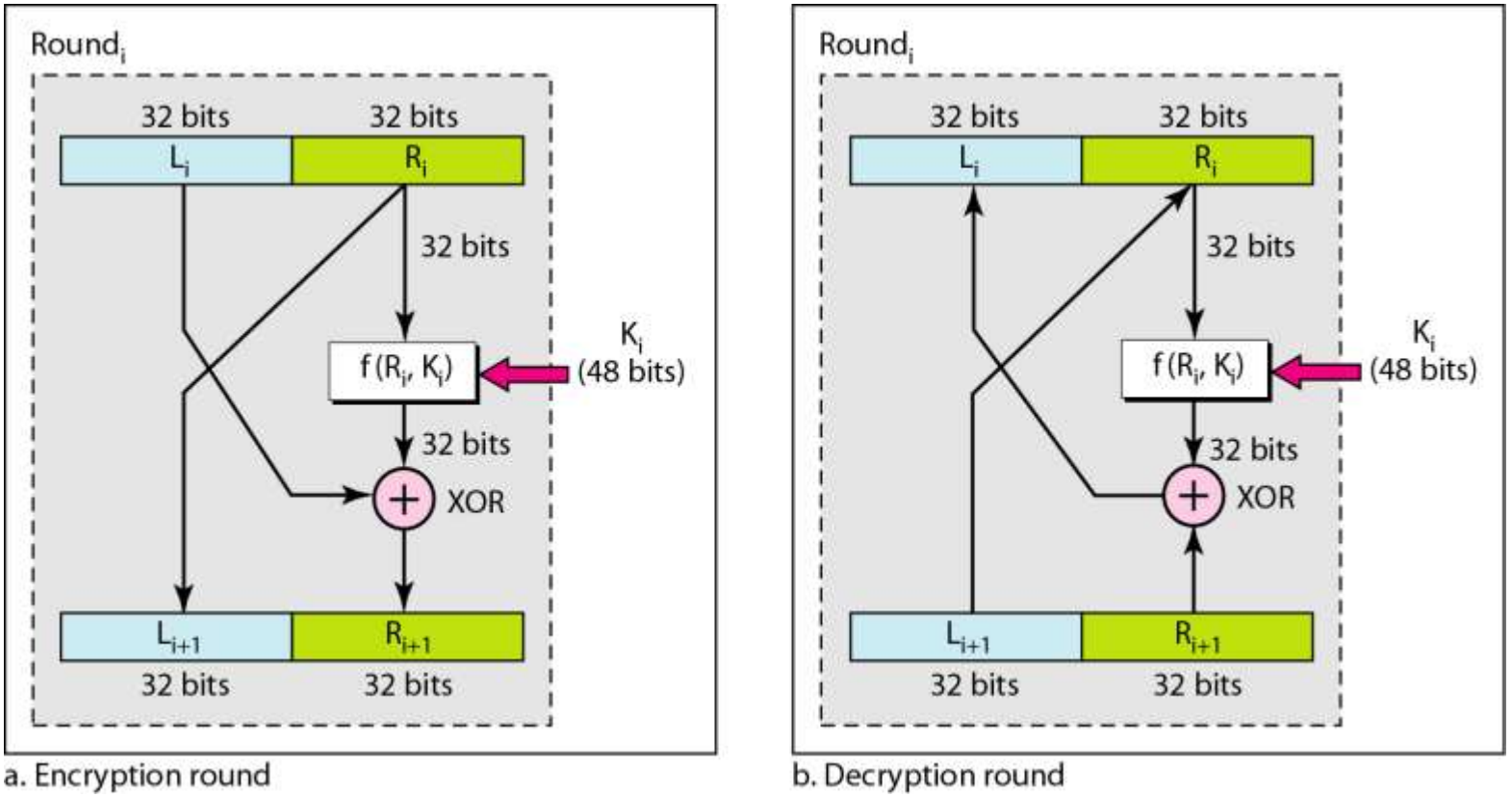



c. Compression

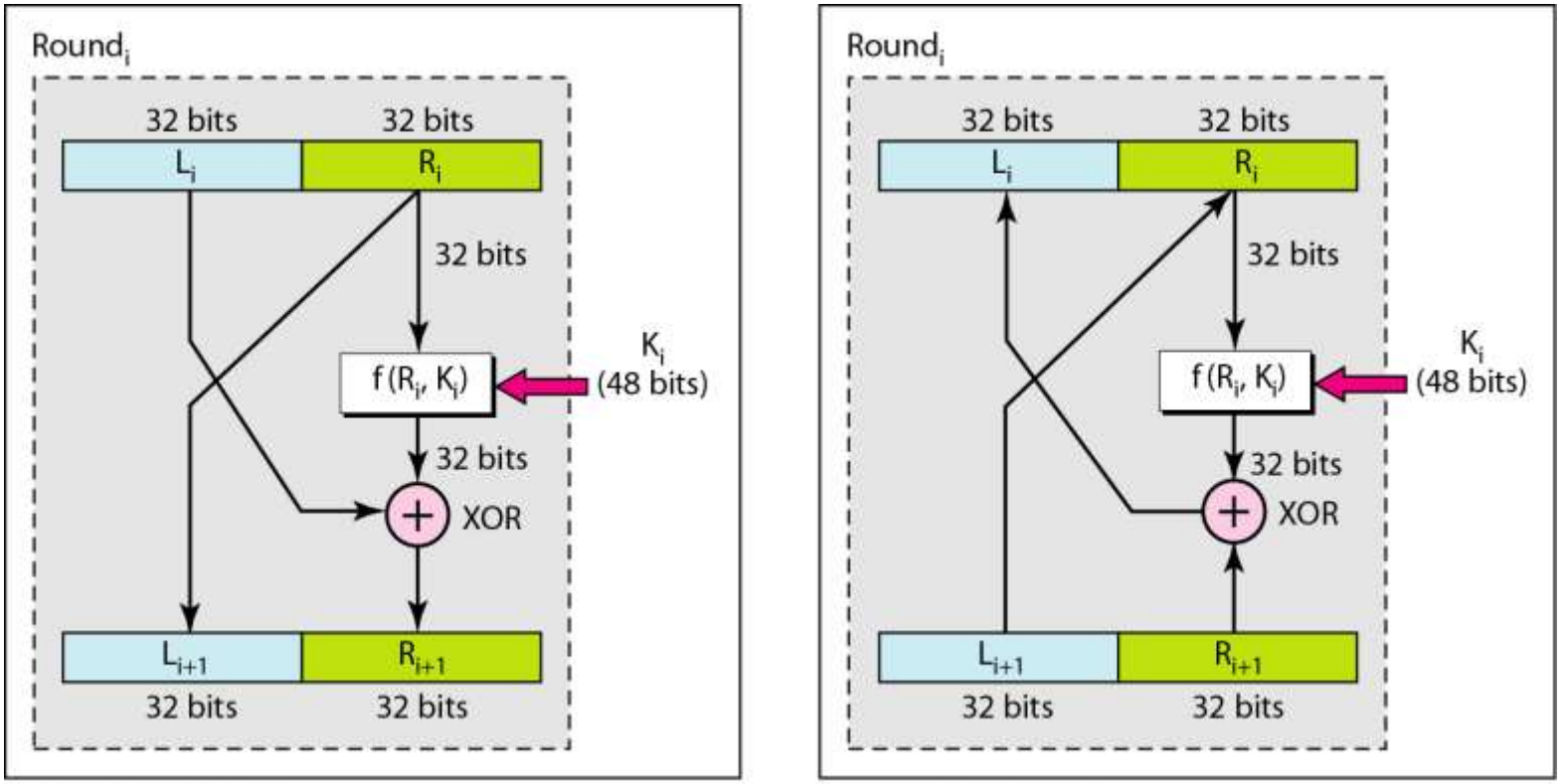
## Figure 30.13 *DES*



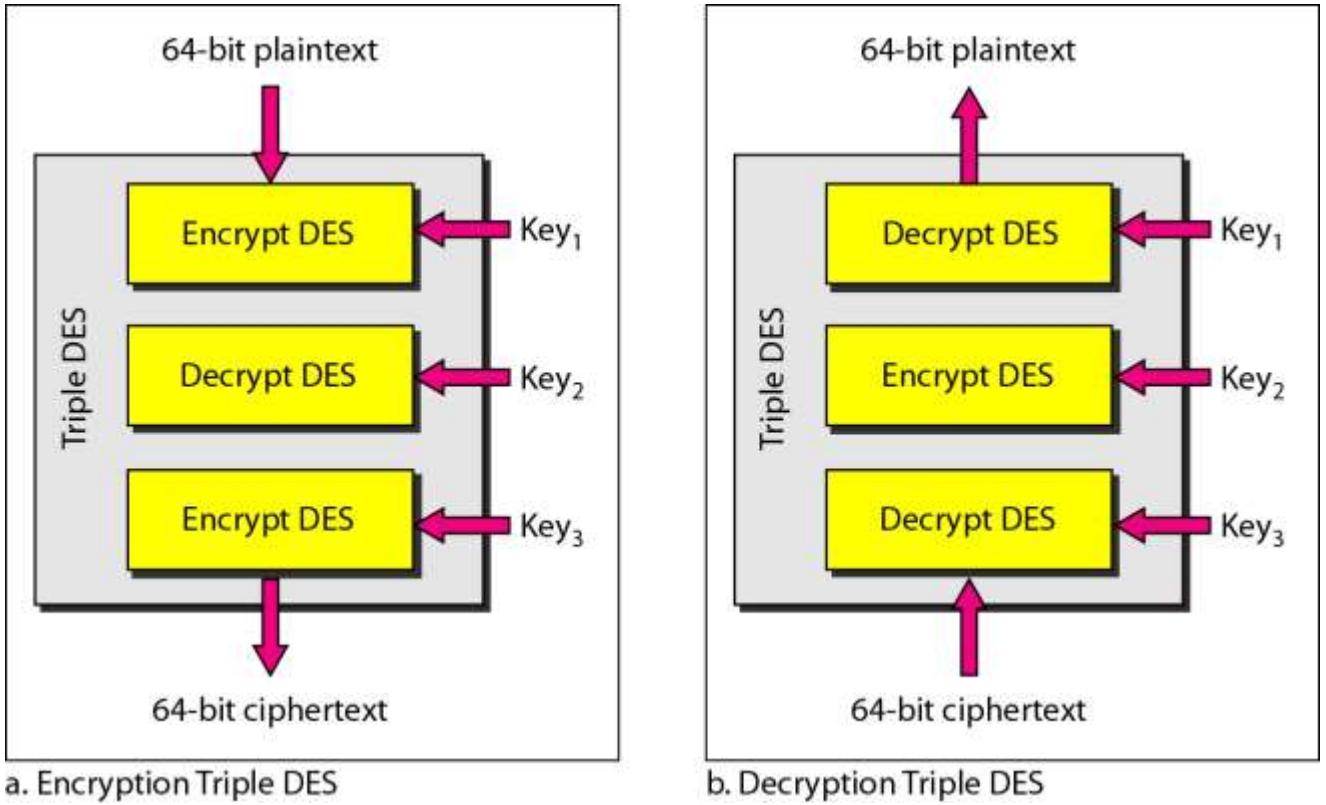
## Figure 30.14 *One round in DES ciphers*



## Figure 30.15 *DES function*



## Figure 30.16 *Triple DES*



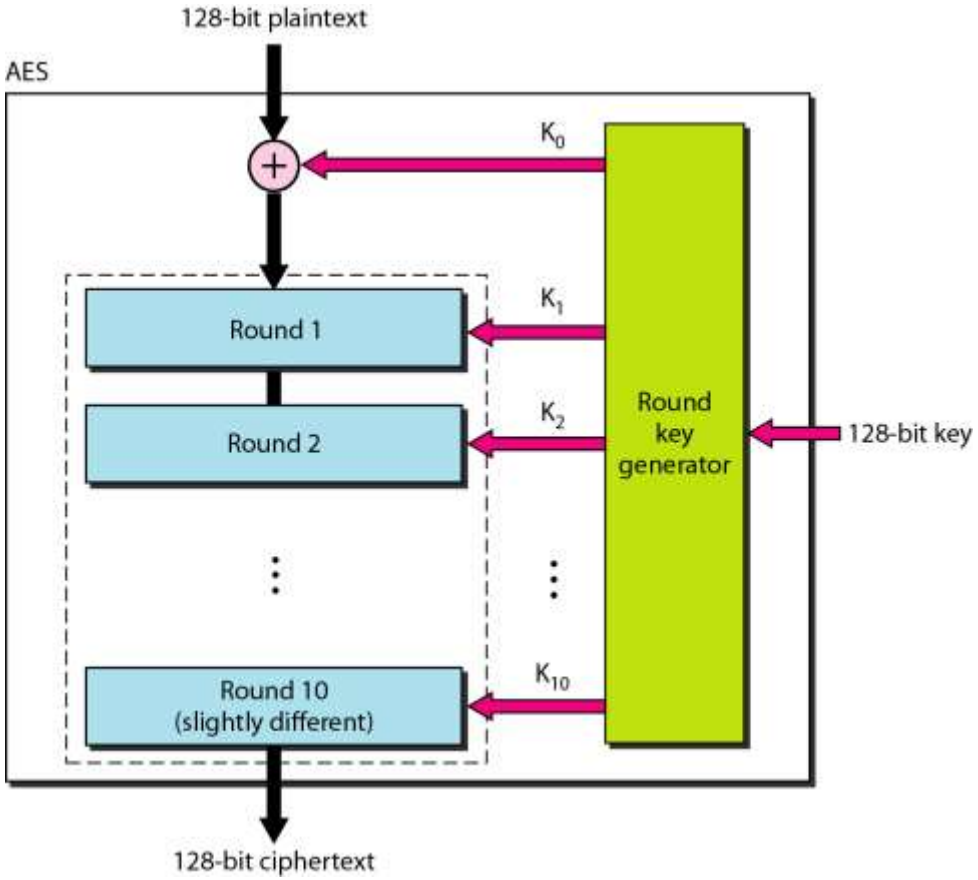
## Table 30.1 *AES configuration*

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

*Note*

**AES has three different configurations with respect to the number of rounds and key size.**

## Figure 30.17 *AES*



## Figure 30.18 *Structure of each round*

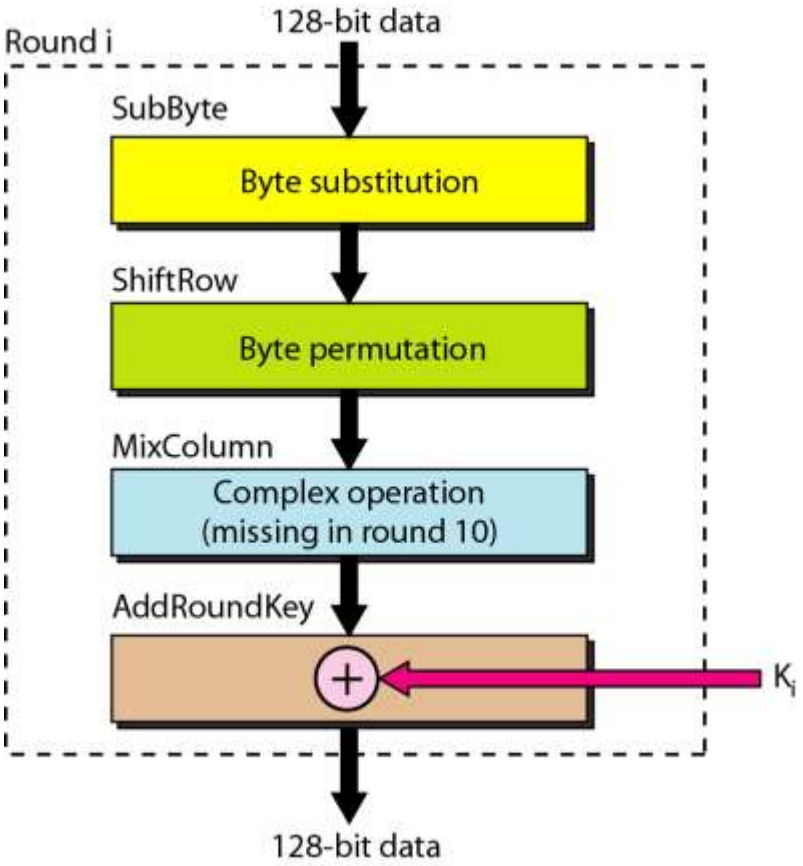
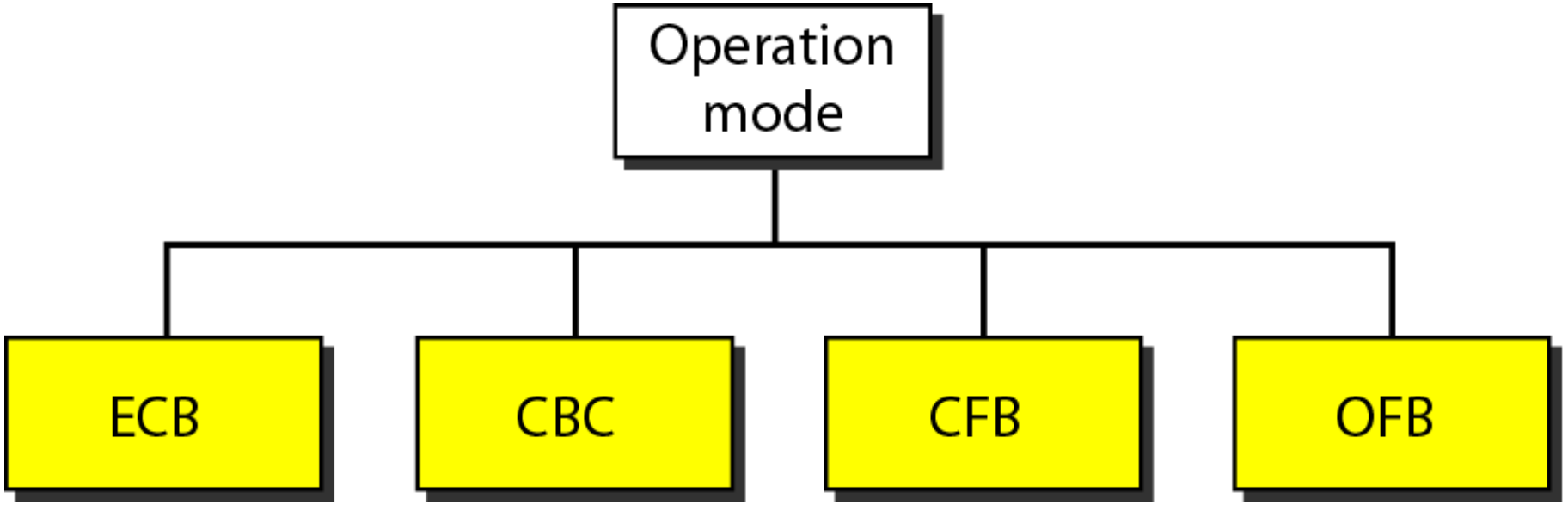
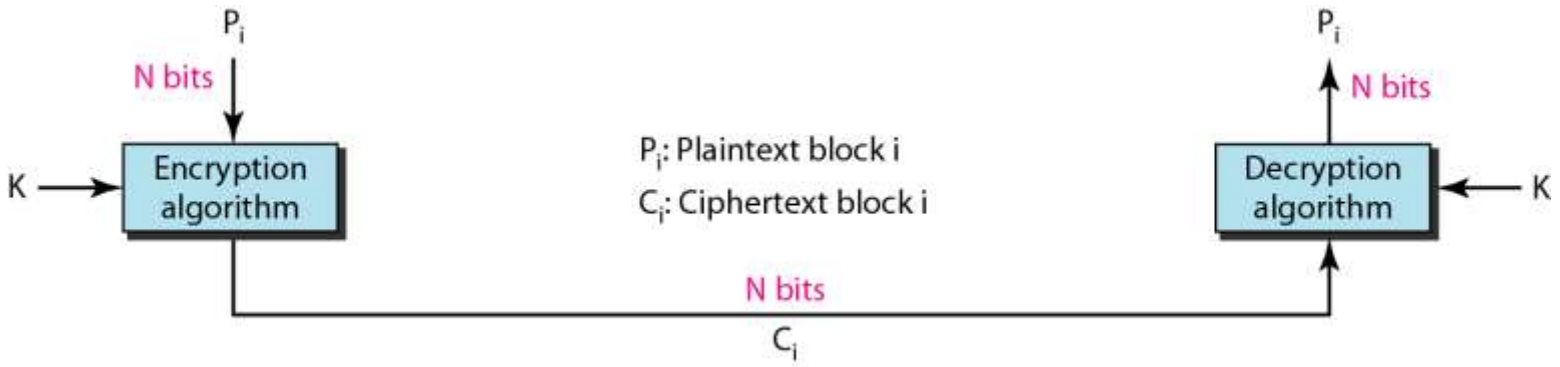


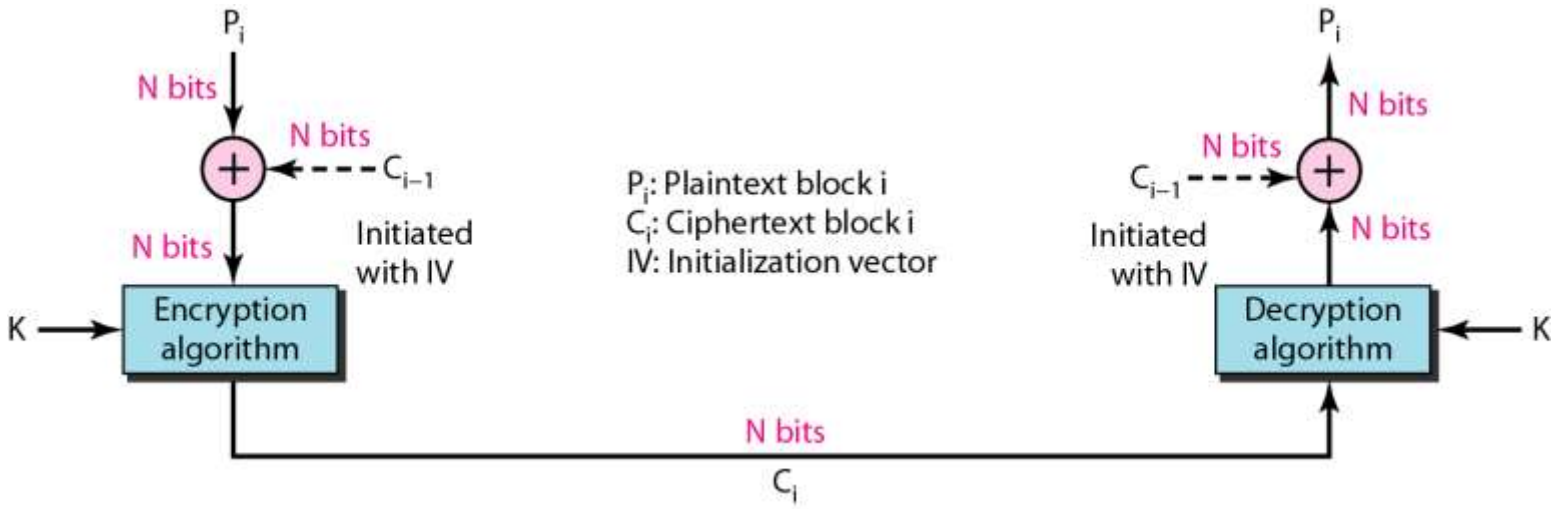
**Figure 30.19** *Modes of operation for block ciphers*



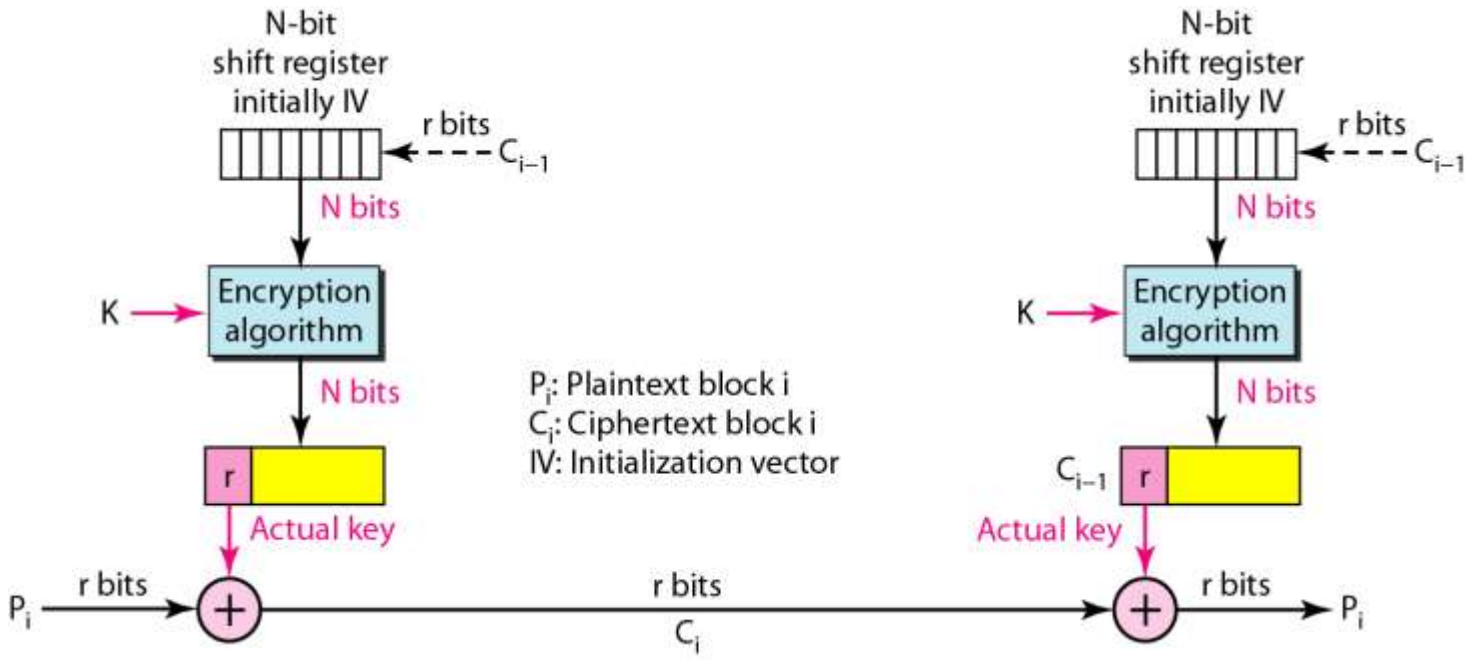
## Figure 30.20 *ECB mode*



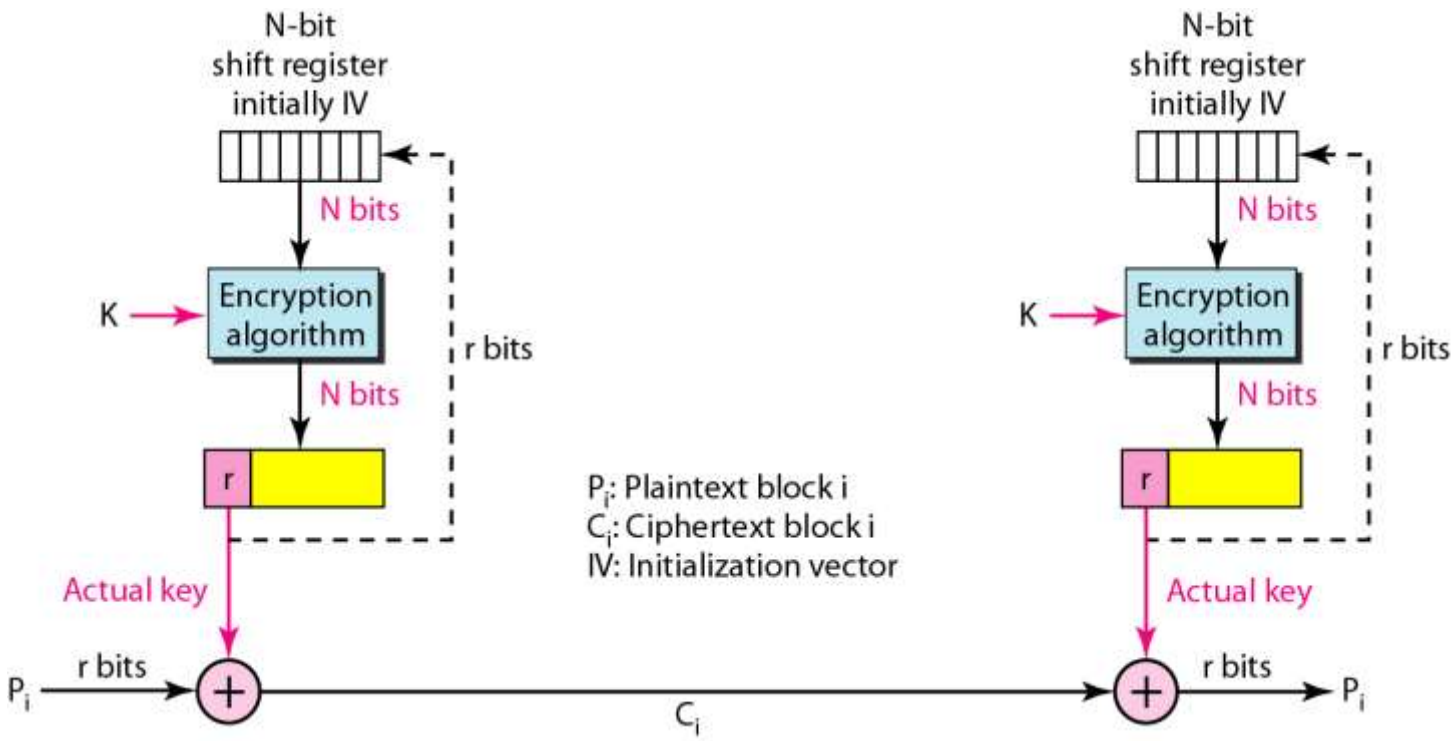
## Figure 30.21 *CBC mode*



## Figure 30.22 *CFB mode*



## Figure 30.23 *OFB mode*



# 30-3 ASYMMETRIC-KEY CRYPTOGRAPHY

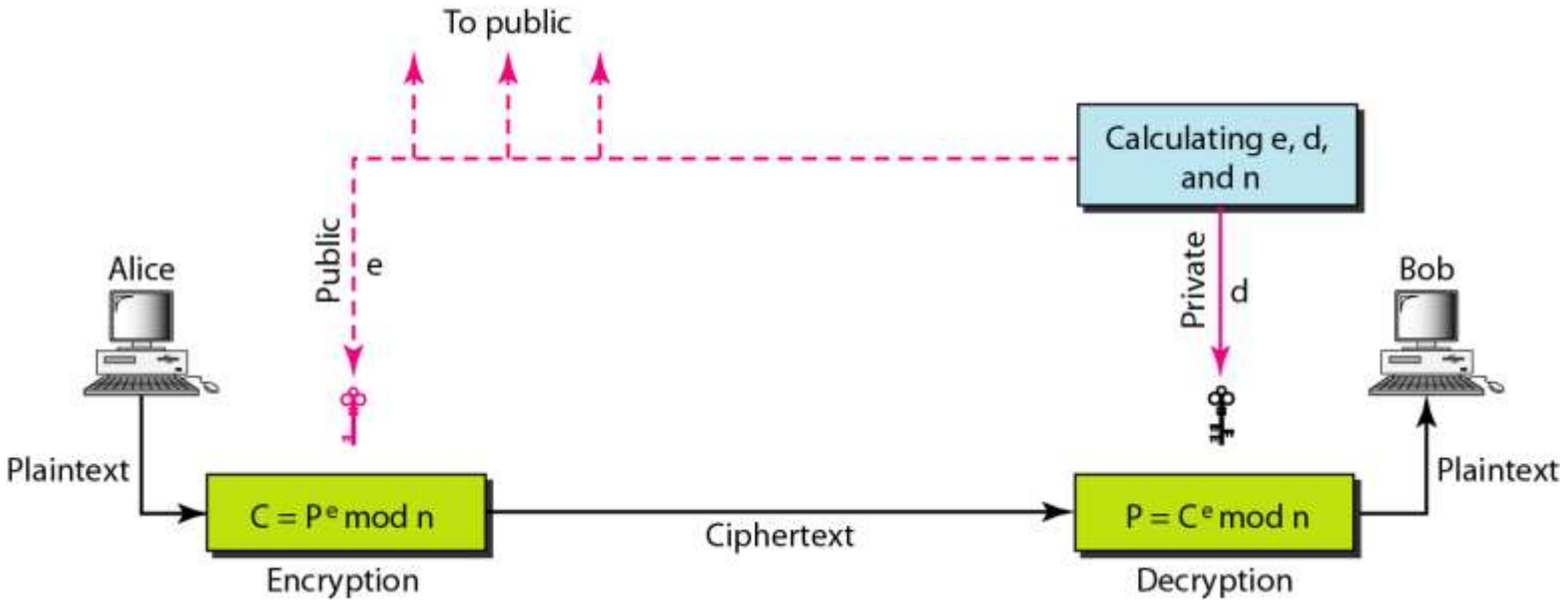
*An asymmetric-key (or public-key) cipher uses two keys: one private and one public. We discuss two algorithms: RSA and Diffie-Hellman.*

***Topics discussed in this section:***

**RSA**

**Diffie-Hellman**

## Figure 30.24 *RSA*



*Note*

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

# Example 30.7

***Bob chooses 7 and 11 as p and q and calculates $n = 7 \cdot 11 = 77$. The value of $\Phi = (7 - 1)\ (11 - 1)$ or 60. Now he chooses two keys, e and d. If he chooses e to be 13, then d is 37. Now imagine Alice sends the plaintext 5 to Bob. She uses the public key 13 to encrypt 5.***

Plaintext: 5

$C = 5^{13} = 26 \bmod 77$

Ciphertext: 26

# Example 30.7 (continued)

***Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:***

Ciphertext: 26

$P = 26^{37} = 5 \bmod 77$

Plaintext: 5

***The plaintext 5 sent by Alice is received as plaintext 5 by Bob.***

# Example 30.8

*Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 159{,}197$ and $\Phi = 396 \cdot 400 = 158{,}400$. She then chooses $e = 343$ and $d = 12{,}007$. Show how Ted can send a message to Jennifer if he knows e and n.*

# Example 30.8 (continuted)

## Solution

*Suppose Ted wants to send the message "NO" to Jennifer. He changes each character to a number (from 00 to 25) with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Ted then uses e and n to encrypt the message. The ciphertext is* $1314^{343} = 33{,}677 \bmod 159{,}197$. *Jennifer receives the message 33,677 and uses the decryption key d to decipher it as* $33{,}677^{12{,}007} = 1314 \bmod 159{,}197$. *Jennifer then decodes 1314 as the message "NO". Figure 30.25 shows the process.*

## Figure 30.25 *Example 30.8*

# Example 30.9

*Let us give a realistic example. We randomly chose an integer of 512 bits. The integer p is a 159-digit number.*

**p =** 96130345313583504574191581280615427909309845594996215822583150879647940
455056470638491257160180347503120986666064924201918087806674210960633542
19926661209

*The integer q is a 160-digit number.*

**q =** 12060191957231446918276794204450896001555925054637033936061798321731482
148483764659215389453209175225273226830107120695604602513887145524969000
359660045617

# Example 30.9 (continued)

*We calculate n. It has 309 digits:*

$$n = 115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656772727460097082714127730434960500556347274566628060099924037102991424472292215772798531727033839381334692684137327622000966676671831831088373420823444370953$$

*We calculate Φ. It has 309 digits:*

$$\phi = 115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656751054233608492916752034482627988117554787657013923444405716989581728196098226361075467211864612171359107358640614008885170265377277264467341066243857664128$$

# Example 30.9 (continued)

*We choose e = 35,535. We then find d.*

**e =** 35535

**d =** 58008302860037763936093661289677917594669062089650962180422866111380593852
82235873170628691003002171085904433840217072986908760061153062025249598844
48047568240966247081485817130463240644077704833134010850947385295645071936
77406119732655742423721761767462077637164207600337085333288532144708859551
36670294831

*Alice wants to send the message "THIS IS A TEST" which can be changed to a numeric value by using the 00–26 encoding scheme (26 is the space character).*

**P =** 19070818260818260026190418 19

# Example 30.9 (continued)

*The ciphertext calculated by Alice is $C = P^e$, which is.*

$$C = 4753091236462268272063655506105451809423717960704917165232392430544529$$
$$6061319932856661784341835911415119741125200568297979457173603610127821$$
$$8847892741566090480023507190715277185914975188465886321011483541033616$$
$$5789846796838676373376577746562507928052114814184404814184430812773059$$
$$00469287424855916646210865 6$$

*Bob can recover the plaintext from the ciphertext by using $P = C^d$, which is*

$$P = 190708182608182600261904 1819$$

*The recovered plaintext is THIS IS A TEST after decoding.*

*Note*

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

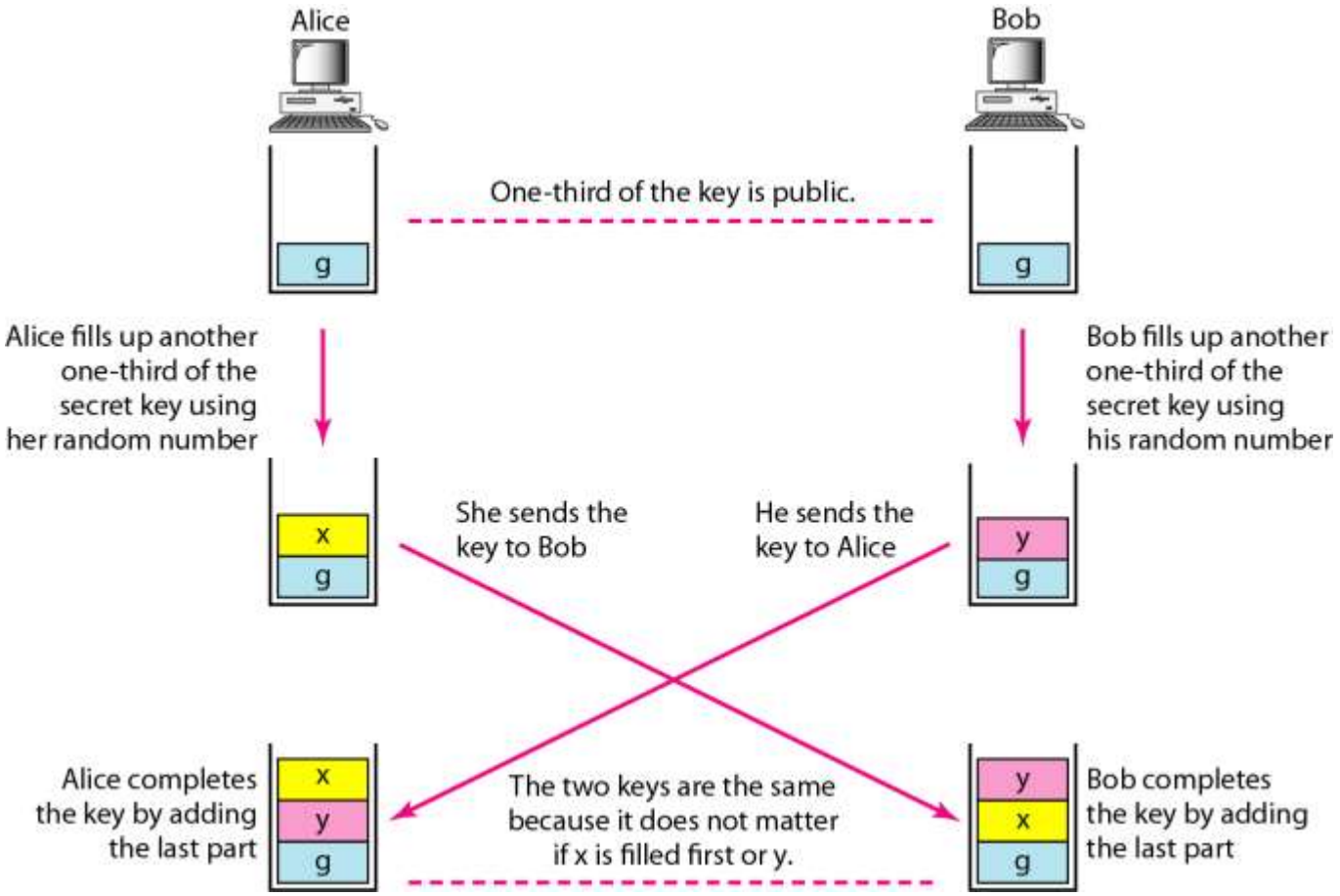
# Example 30.10

*Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume g = 7 and p = 23. The steps are as follows:*

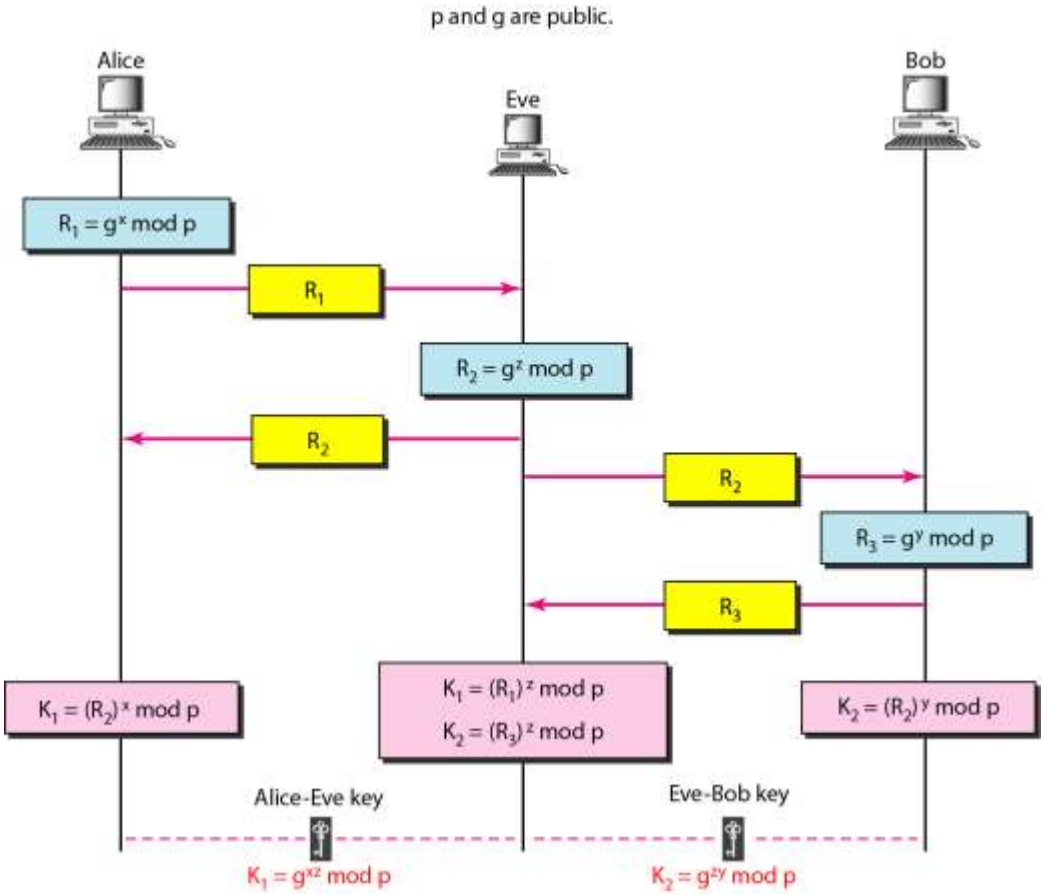
1. *Alice chooses x = 3 and calculates* $R_1 = 7^3 \bmod 23 = 21$.
2. *Bob chooses y = 6 and calculates* $R_2 = 7^6 \bmod 23 = 4$.
3. *Alice sends the number 21 to Bob.*
4. *Bob sends the number 4 to Alice.*
5. *Alice calculates the symmetric key* $K = 4^3 \bmod 23 = 18$.
6. *Bob calculates the symmetric key* $K = 21^6 \bmod 23 = 18$.

*The value of K is the same for both Alice and Bob;* $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$.

## Figure 30.27 *Diffie-Hellman idea*



## Figure 30.28 *Man-in-the-middle attack*



Data Communications and Networking Fourth Edition

Forouzan

# Chapter 31

# Network Security

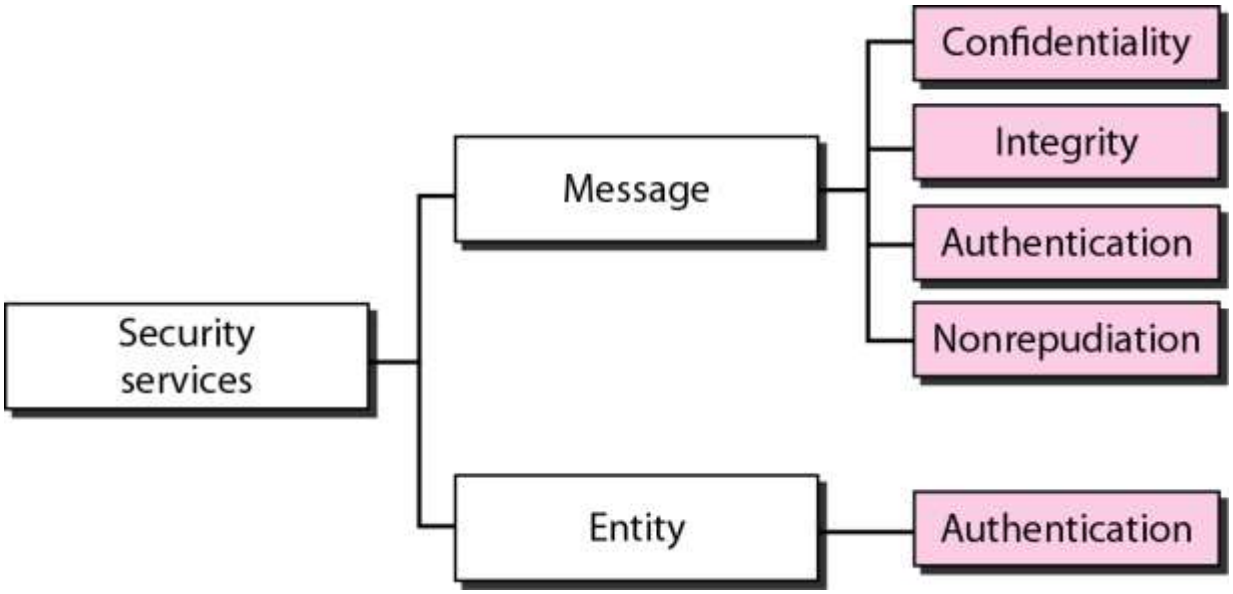
# 31-1 SECURITY SERVICES

***Network security can provide five services. Four of these services are related to the message exchanged using the network. The fifth service provides entity authentication or identification.***

***Topics discussed in this section:***

**Message Confidentiality**
**Message Integrity**
**Message Authentication**
**Message Nonrepudiation**
**Entity Authentication**

**Figure 31.1** *Security services related to the message or entity*



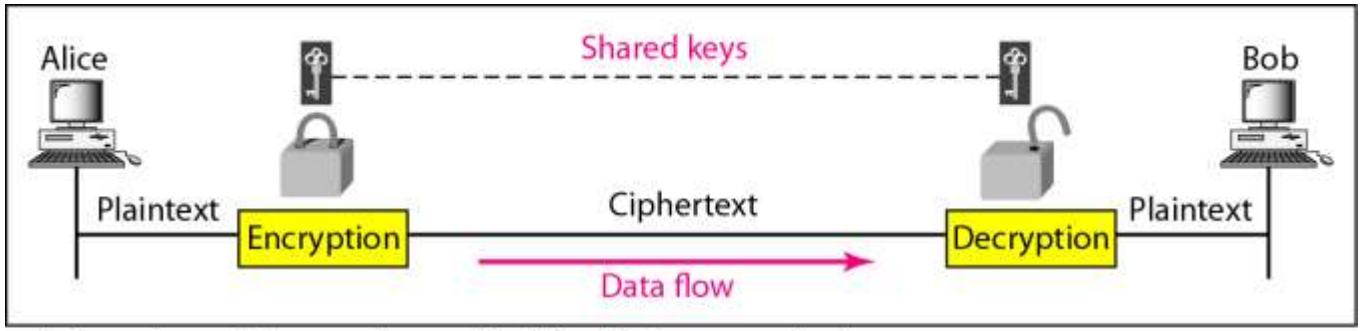
# 31-2 MESSAGE CONFIDENTIALITY

***The concept of how to achieve message confidentiality or privacy has not changed for thousands of years. The message must be encrypted at the sender site and decrypted at the receiver site. This can be done using either symmetric-key cryptography or asymmetric-key cryptography.***

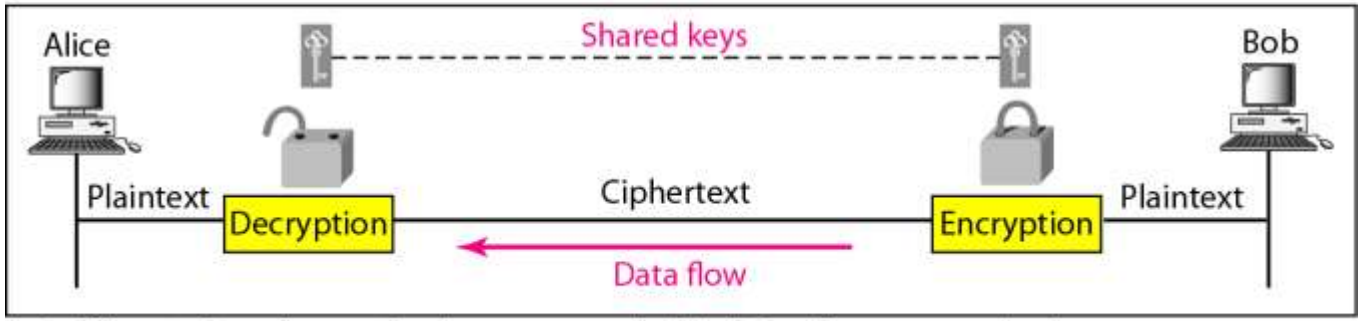
***Topics discussed in this section:***

**Confidentiality with Symmetric-Key Cryptography**
**Confidentiality with Asymmetric-Key Cryptography**

## Figure 31.2 *Message confidentiality using symmetric keys in two directions*

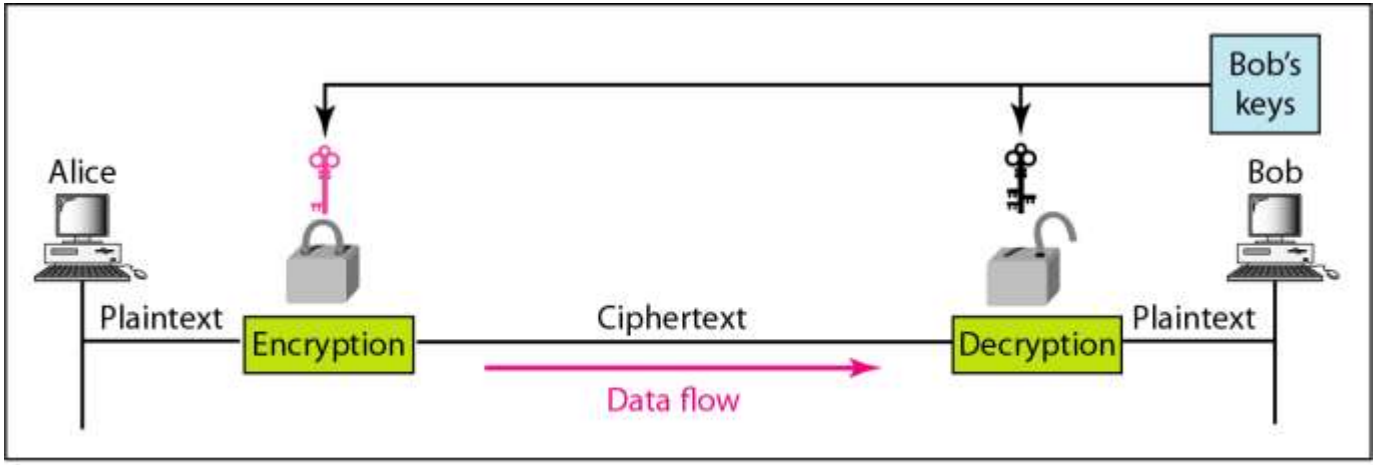



a. A shared secret key can be used in Alice-Bob communication

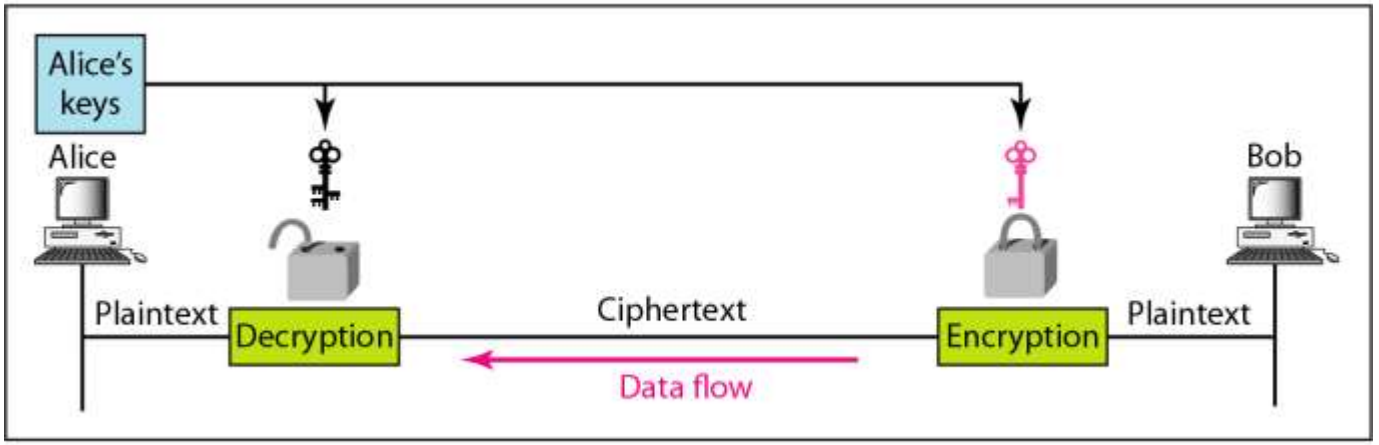



b. A different shared secret key is recommended in Bob-Alice communication

## Figure 31.3 *Message confidentiality using asymmetric keys*




a. Bob's keys are used in Alice-Bob communication




b. Alice's keys are used in Bob-Alice communication

# 31-3 MESSAGE INTEGRITY

***Encryption and decryption provide secrecy, or confidentiality, but not integrity. However, on occasion we may not even need secrecy, but instead must have integrity.***

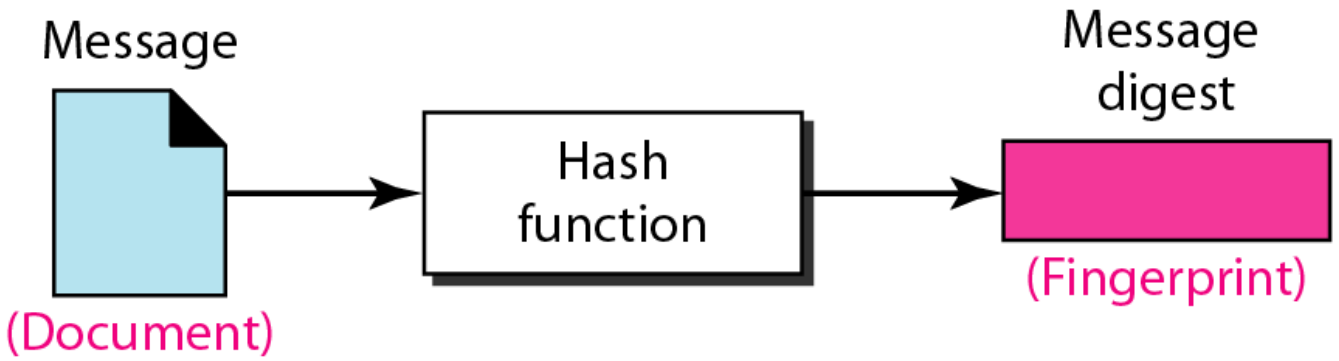
***Topics discussed in this section:***

**Document and Fingerprint**
**Message and Message Digest**
**Creating and Checking the Digest**
**Hash Function Criteria**
**Hash Algorithms: SHA-1**

*Note*

**To preserve the integrity of a document, both the document and the fingerprint are needed.**

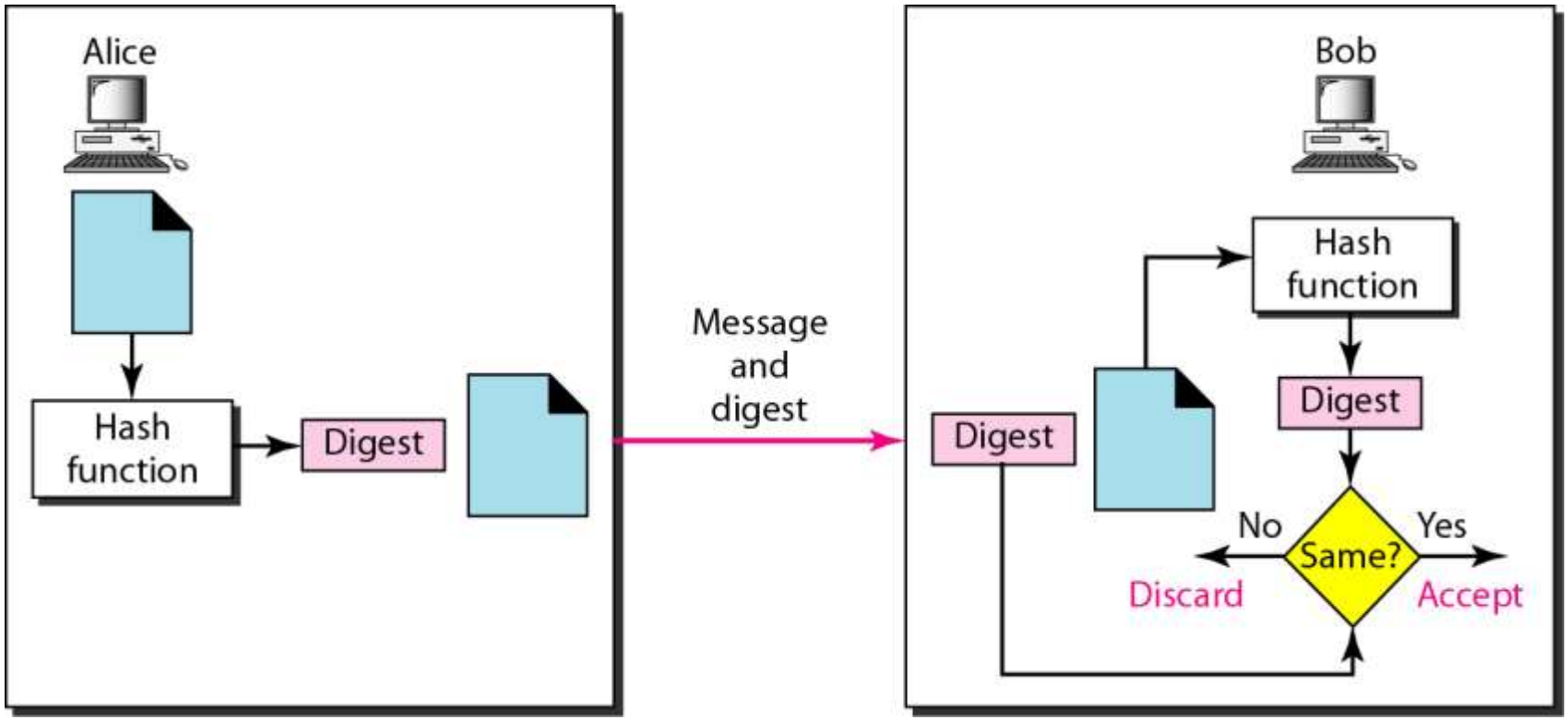
## Figure 31.4 *Message and message digest*



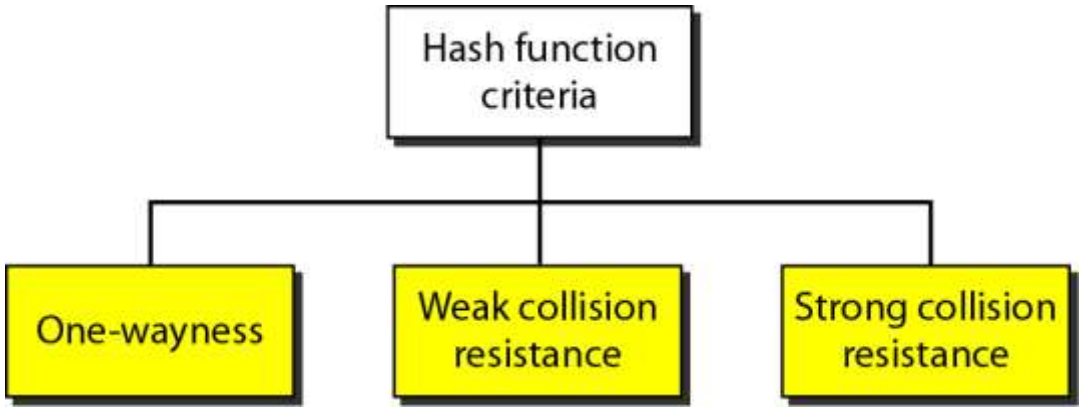
*Note*

**The message digest needs to be kept secret.**

## Figure 31.5 *Checking integrity*



## Figure 31.6 *Criteria of a hash function*



# Example 31.1

***Can we use a conventional lossless compression method as a hashing function?***

***Solution***

***We cannot. A lossless compression method creates a compressed message that is reversible. You can uncompress the compressed message to get the original one.***

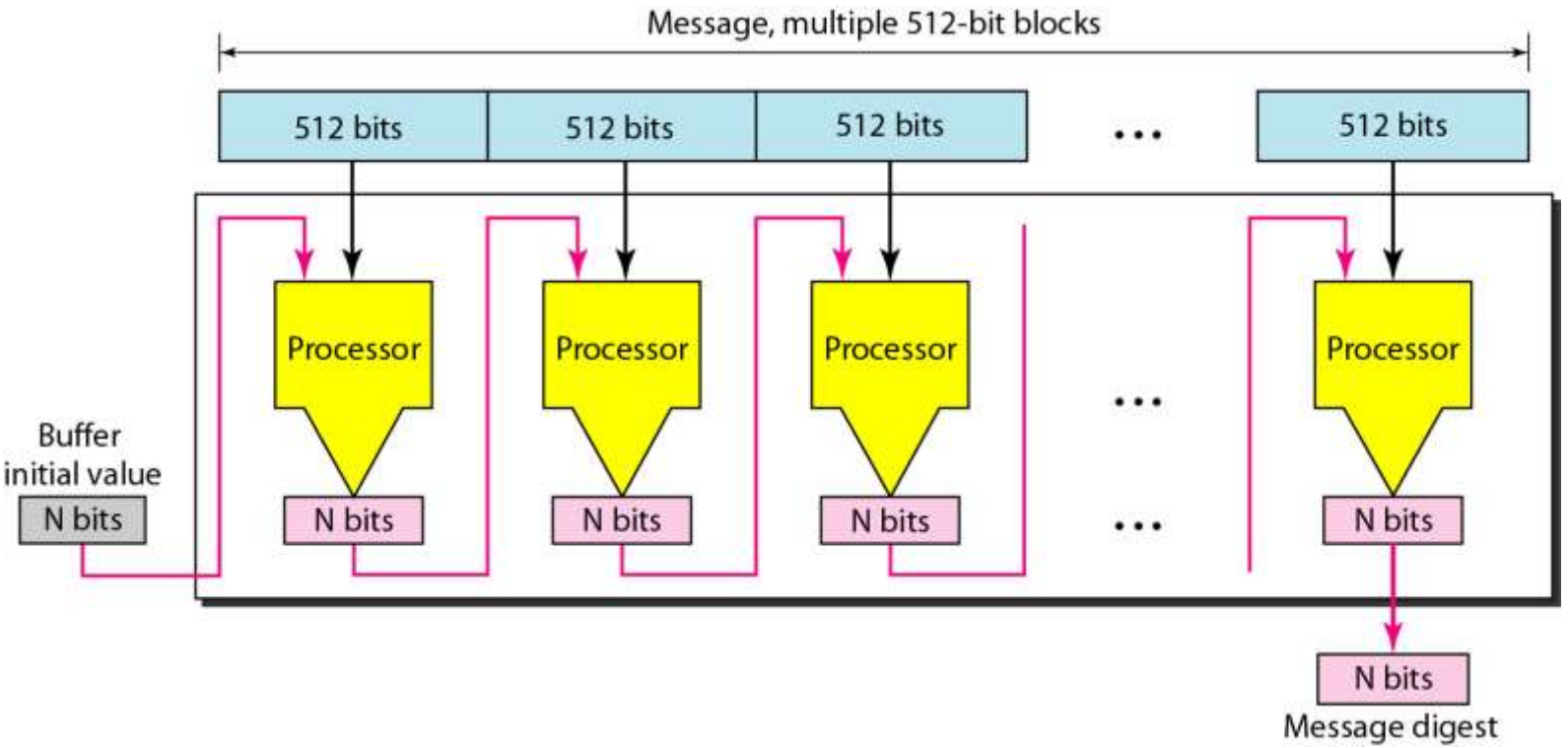
# Example 31.2

***Can we use a checksum method as a hashing function?***

***Solution***

***We can. A checksum function is not reversible; it meets the first criterion. However, it does not meet the other criteria.***

## Figure 31.7 *Message digest creation*

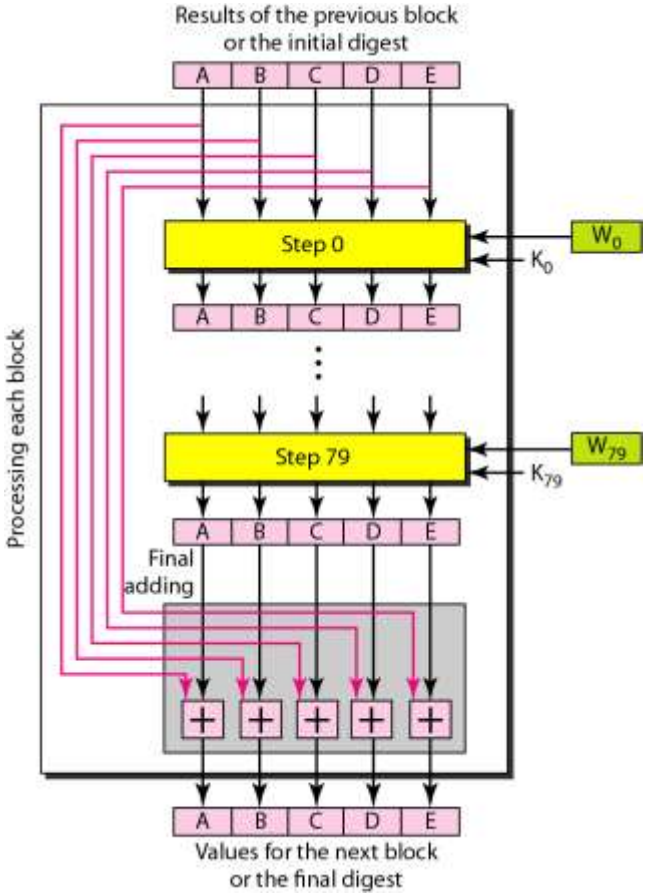


*Note*

**SHA-1 hash algorithms create an N-bit message digest out of a message of 512-bit blocks.**

**SHA-1 has a message digest of 160 bits (5 words of 32 bits).**

## Figure 31.8 *Processing of one block in SHA-1*



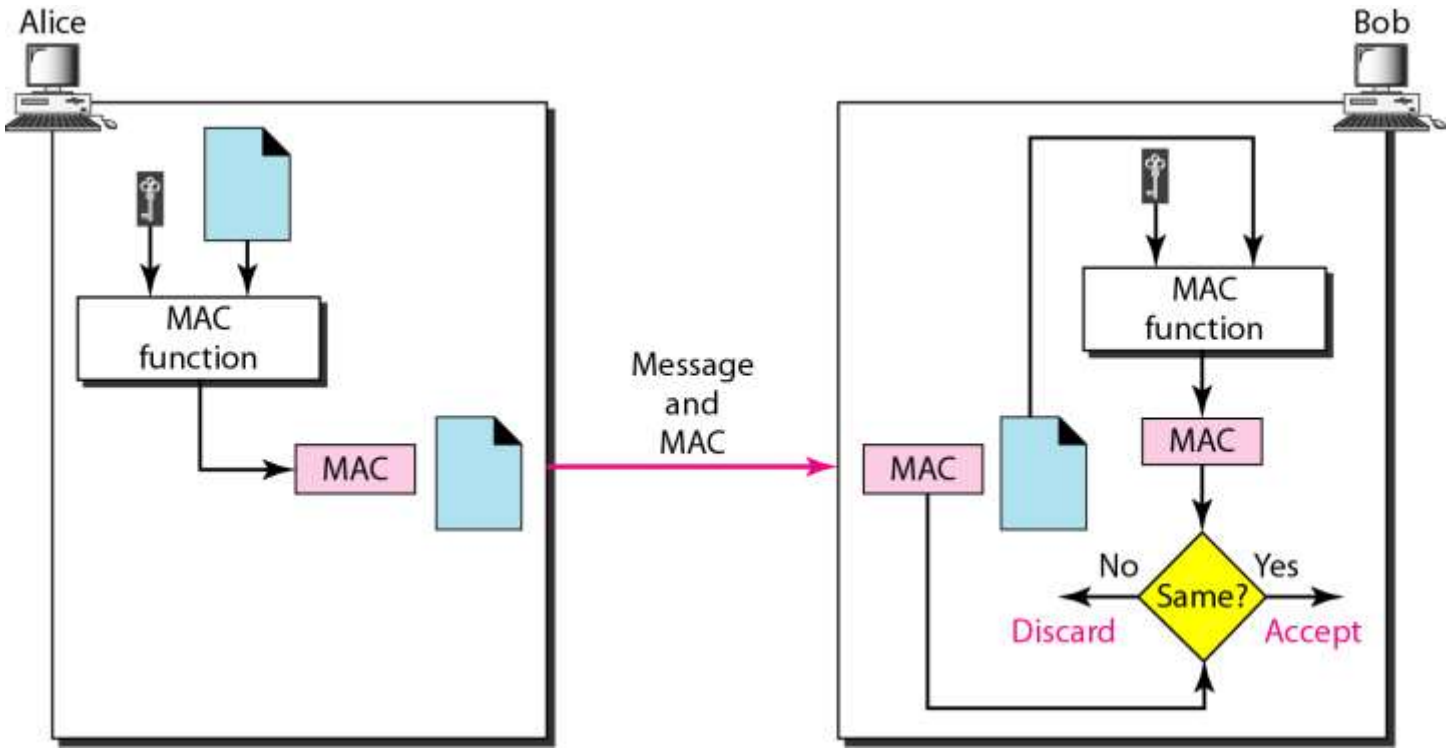
# 31-4 MESSAGE AUTHENTICATION

***A hash function per se cannot provide authentication. The digest created by a hash function can detect any modification in the message, but not authentication.***

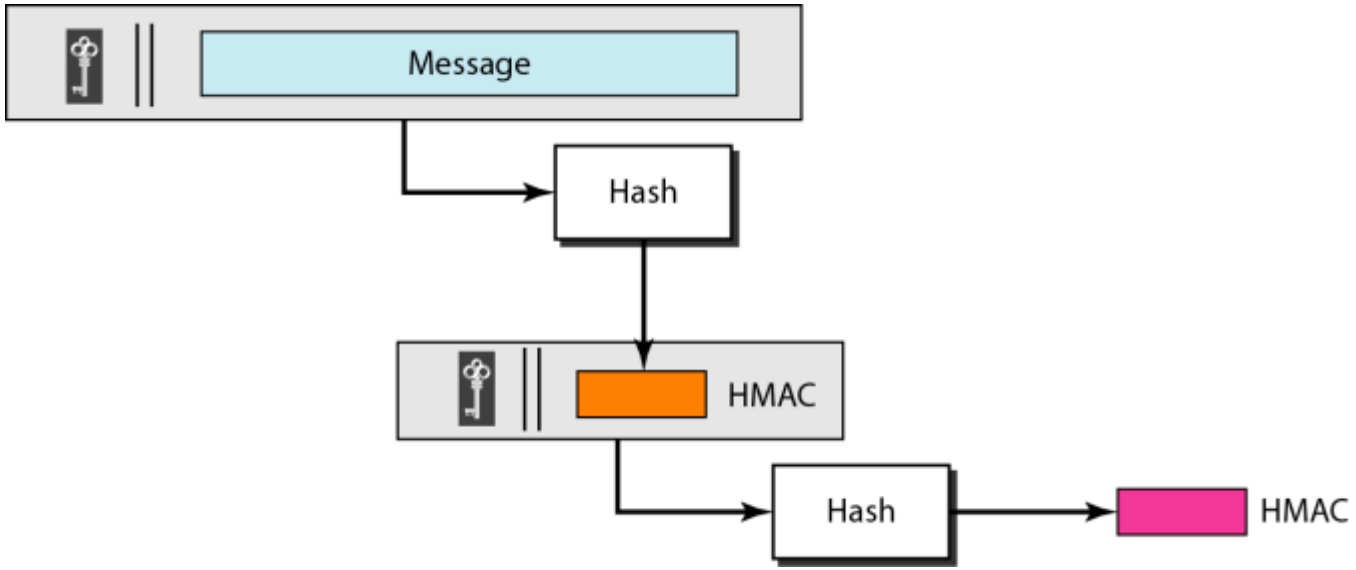
***Topics discussed in this section:***

**MAC**

## Figure 31.9 *MAC, created by Alice and checked by Bob*



## Figure 31.10 *HMAC*



# 31-5 DIGITAL SIGNATURE

*When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a digital signature.*

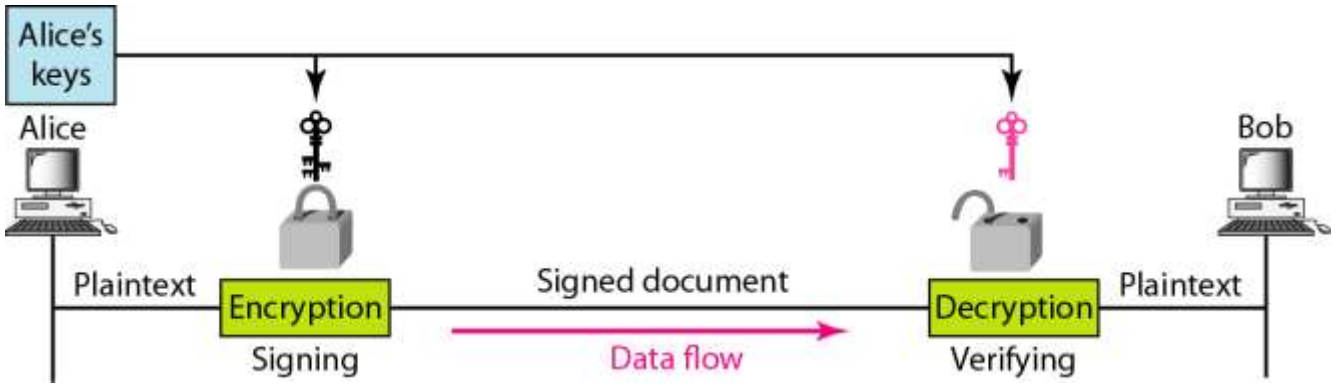
***Topics discussed in this section:***

**Comparison**
**Need for Keys**
**Process**

*Note*

**A digital signature needs a public-key system.**

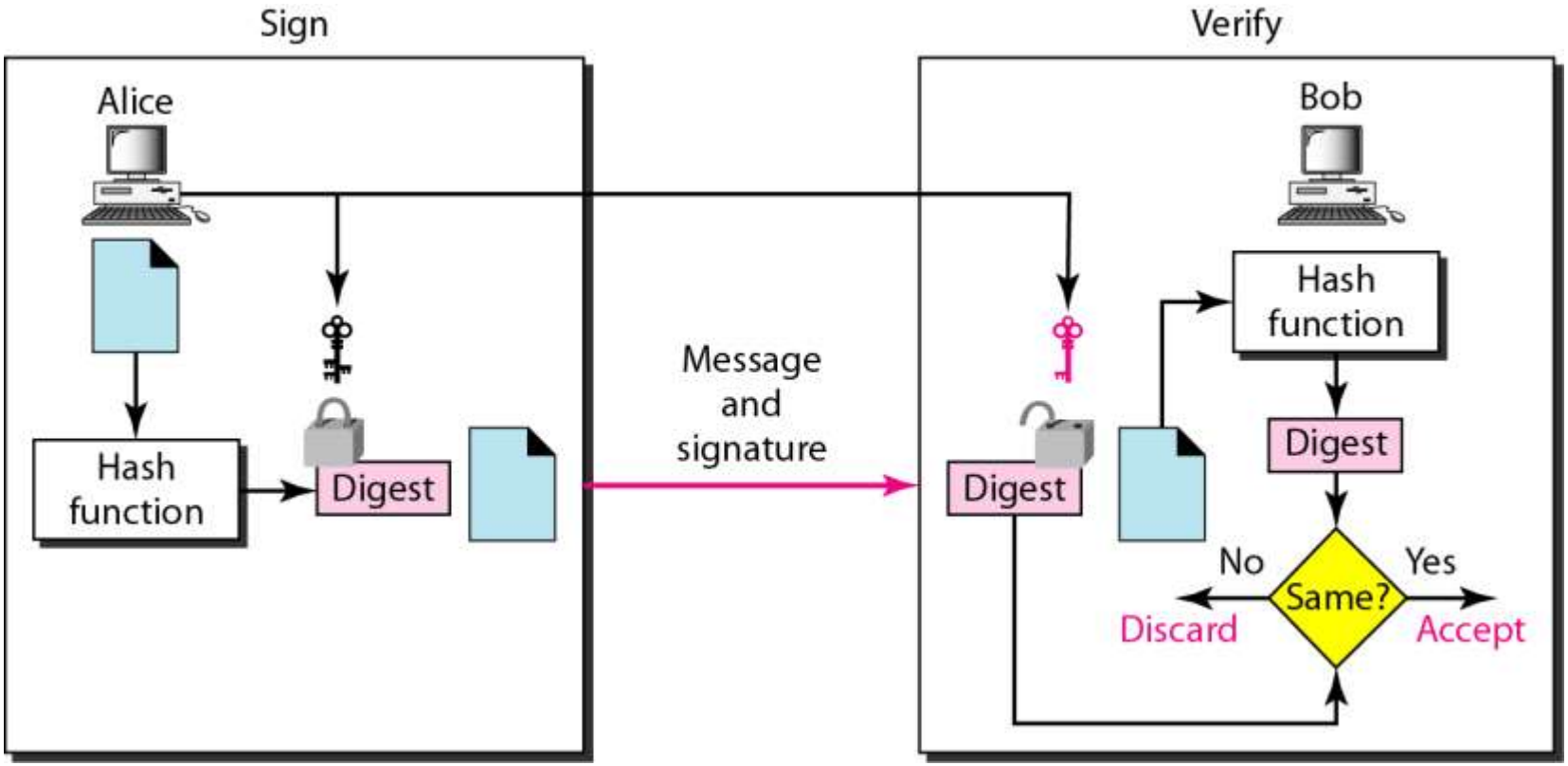
## Figure 31.11 *Signing the message itself in digital signature*



*Note*

**In a cryptosystem, we use the private and public keys of the receiver; in digital signature, we use the private and public keys of the sender.**

## Figure 31.12 *Signing the digest in a digital signature*



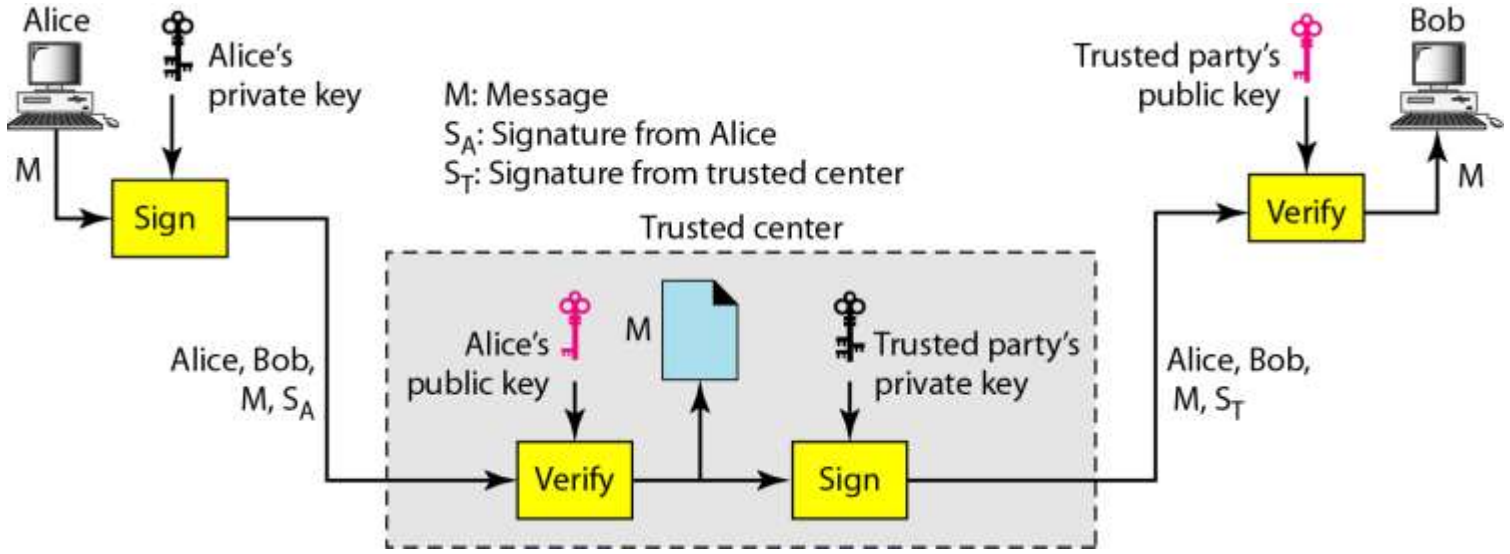
*Note*

**A digital signature today provides message integrity.**

*Note*

**Digital signature provides message authentication.**

## Figure 31.13 *Using a trusted center for nonrepudiation*



*Note*

**Nonrepudiation can be provided using a trusted party.**

# 31-6 ENTITY AUTHENTICATION

*Entity authentication is a technique designed to let one party prove the identity of another party. An entity can be a person, a process, a client, or a server. The entity whose identity needs to be proved is called the claimant; the party that tries to prove the identity of the claimant is called the verifier.*

***Topics discussed in this section:***

**Passwords**

**Challenge-Response**

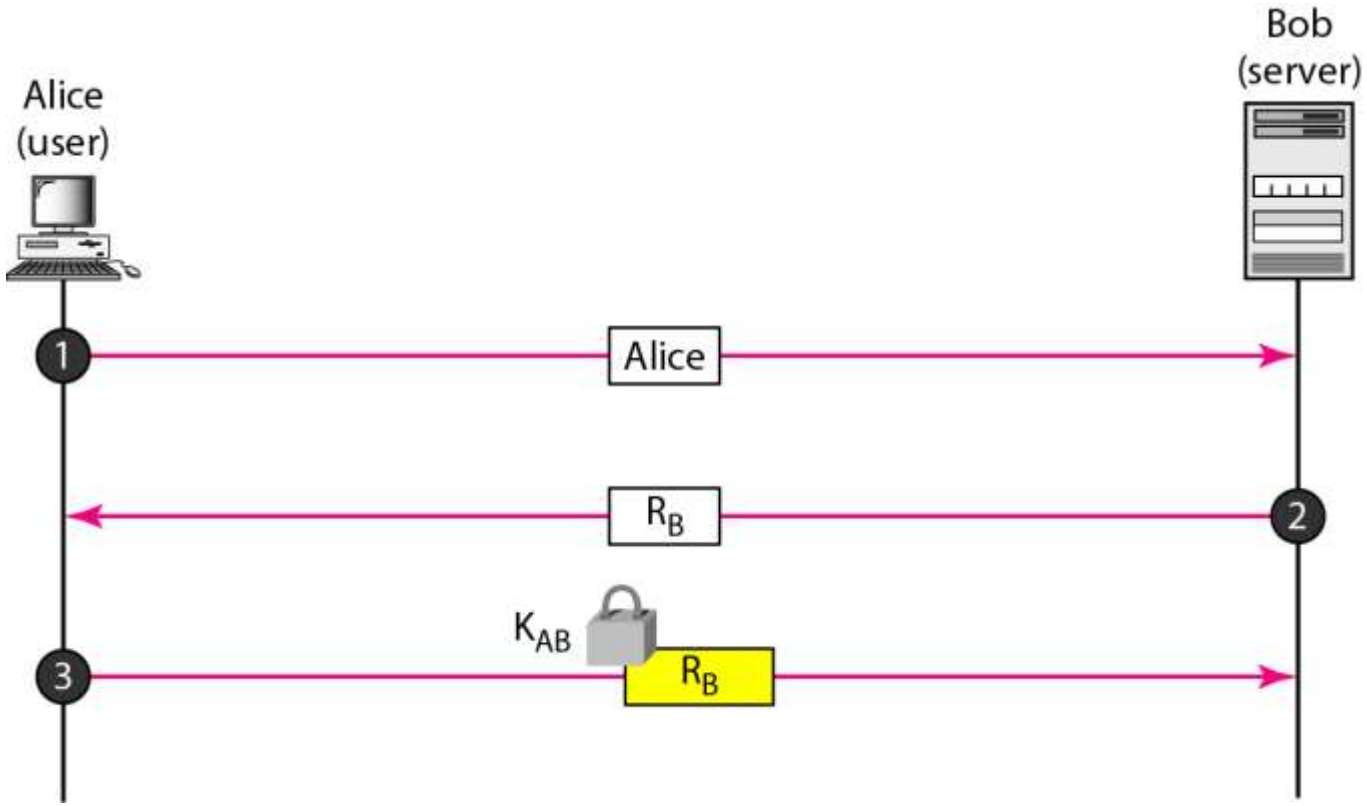
*Note*

**In challenge-response authentication, the claimant proves that she knows a secret without revealing it.**

*Note*

**The challenge is a time-varying value sent by the verifier;
the response is the result of a function applied on the challenge.**

## Figure 31.14 *Challenge/response authentication using a nonce*



## Figure 31.15 *Challenge-response authentication using a timestamp*

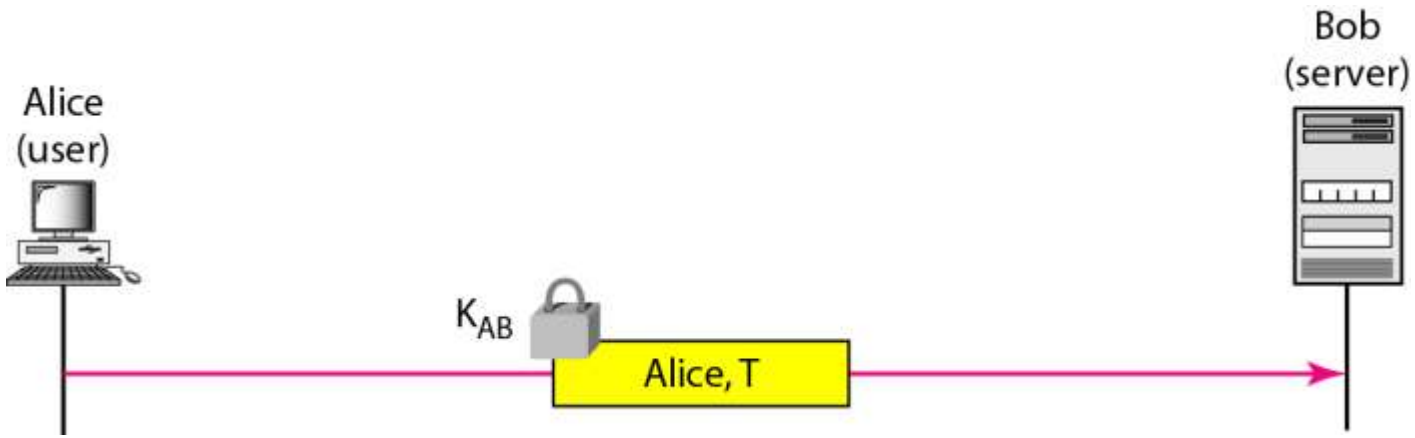
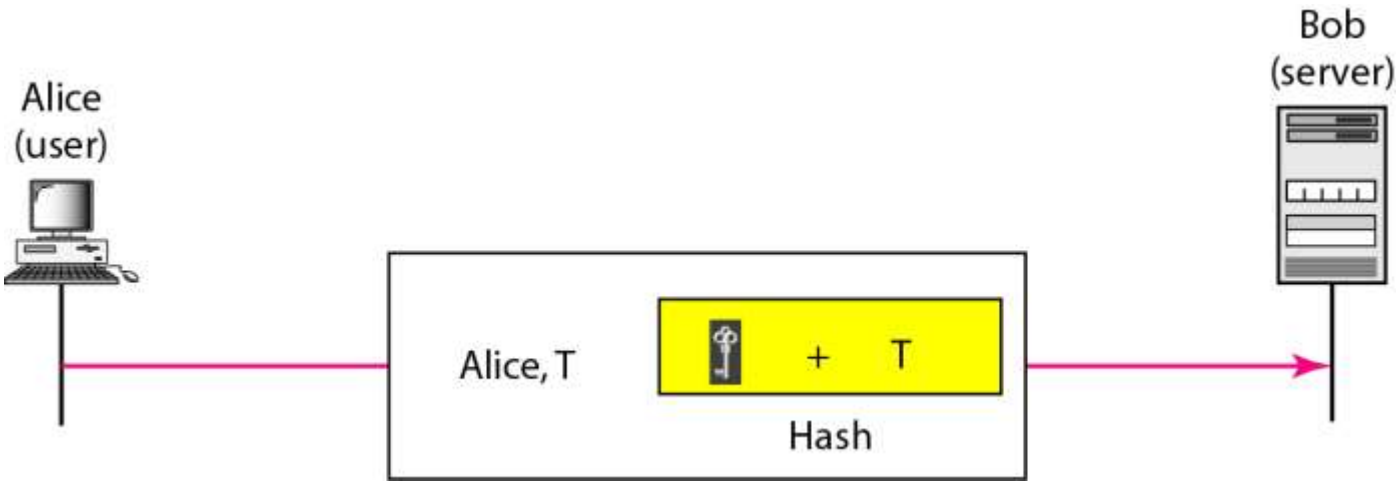
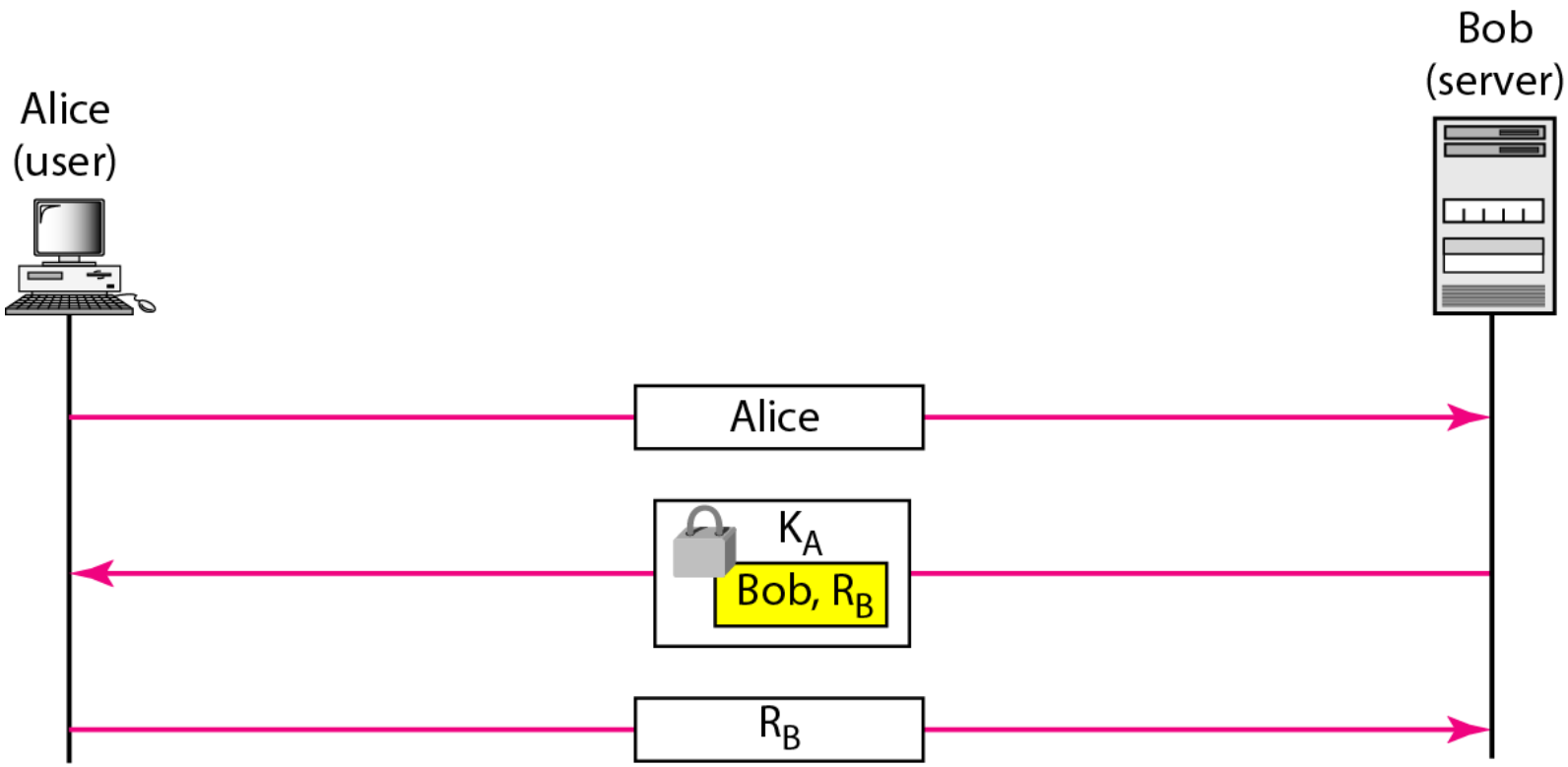


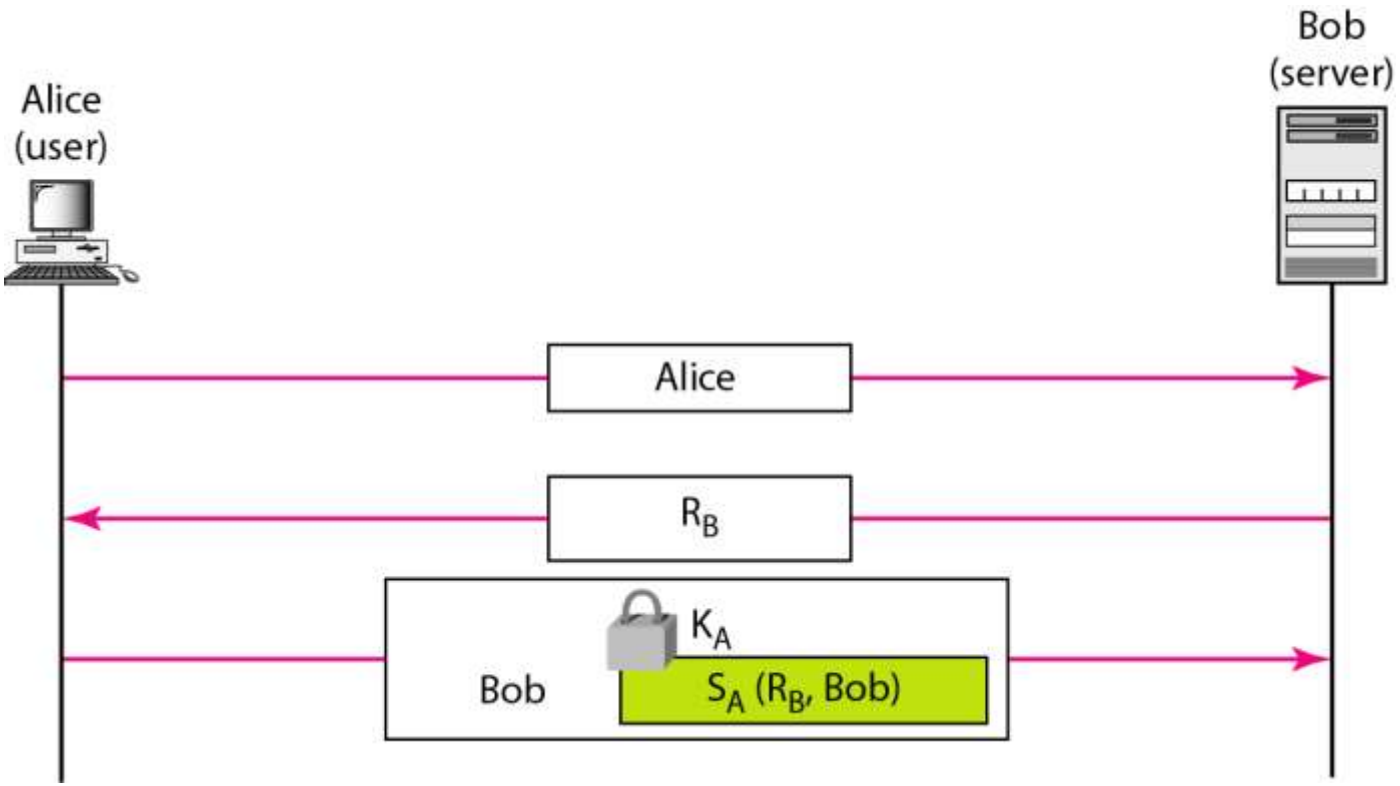
**Figure 31.16** *Challenge-response authentication using a keyed-hash function*



## Figure 31.17 *Authentication, asymmetric-key*



## Figure 31.18 *Authentication, using digital signature*



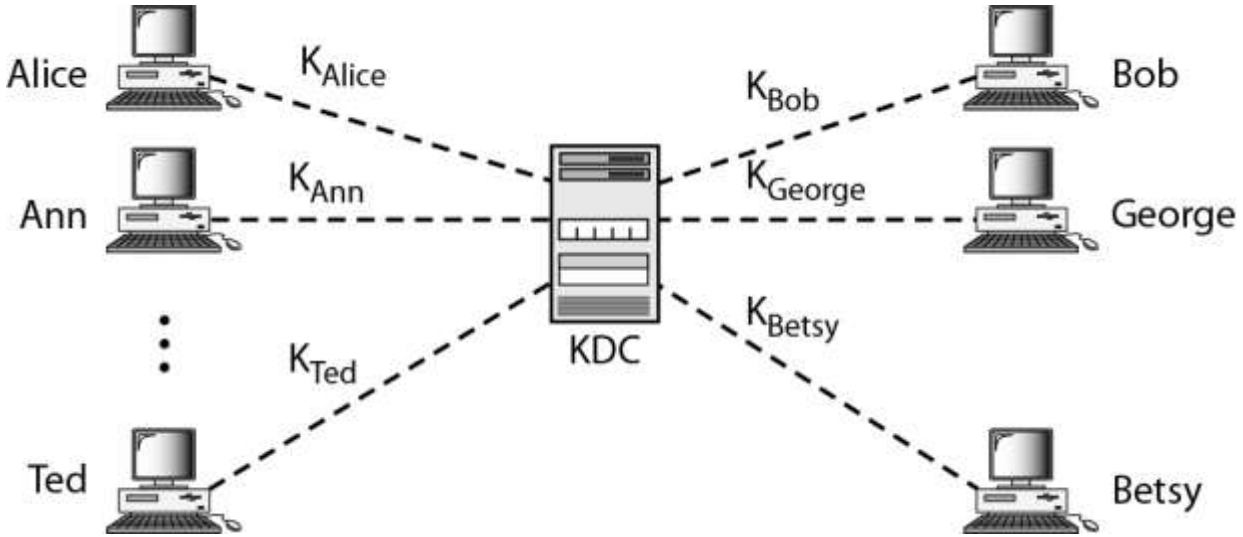
# 31-7 KEY MANAGEMENT

*We never discussed how secret keys in symmetric-key cryptography and how public keys in asymmetric-key cryptography are distributed and maintained. In this section, we touch on these two issues. We first discuss the distribution of symmetric keys; we then discuss the distribution of asymmetric keys.*

***Topics discussed in this section:***

**Symmetric-Key Distribution**
**Public-Key Distribution**

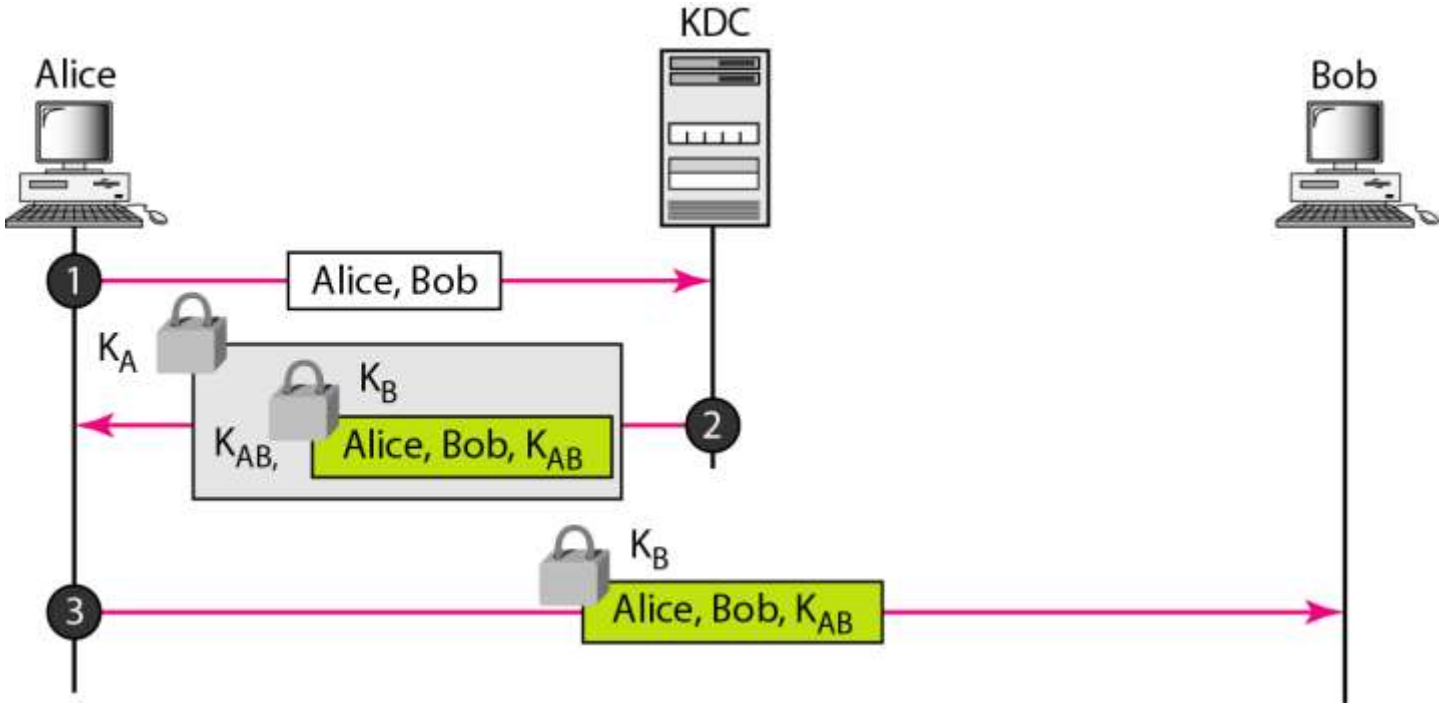
## Figure 31.19 *KDC*



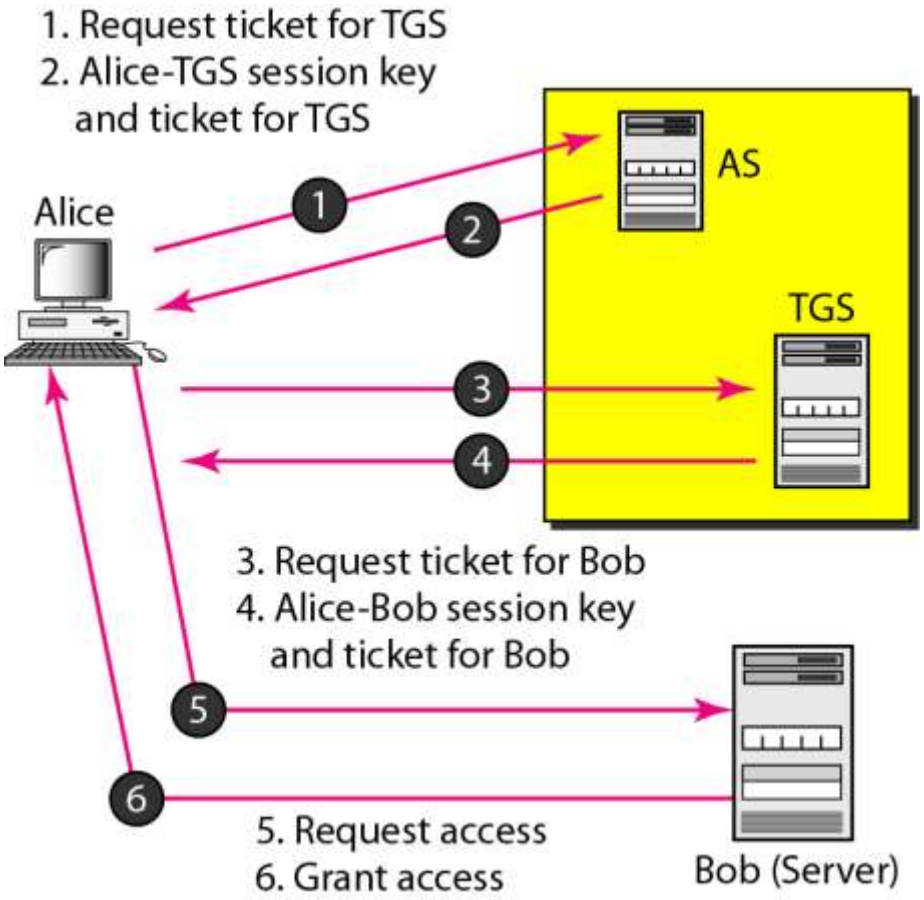
*Note*

**A session symmetric key between two parties is used only once.**

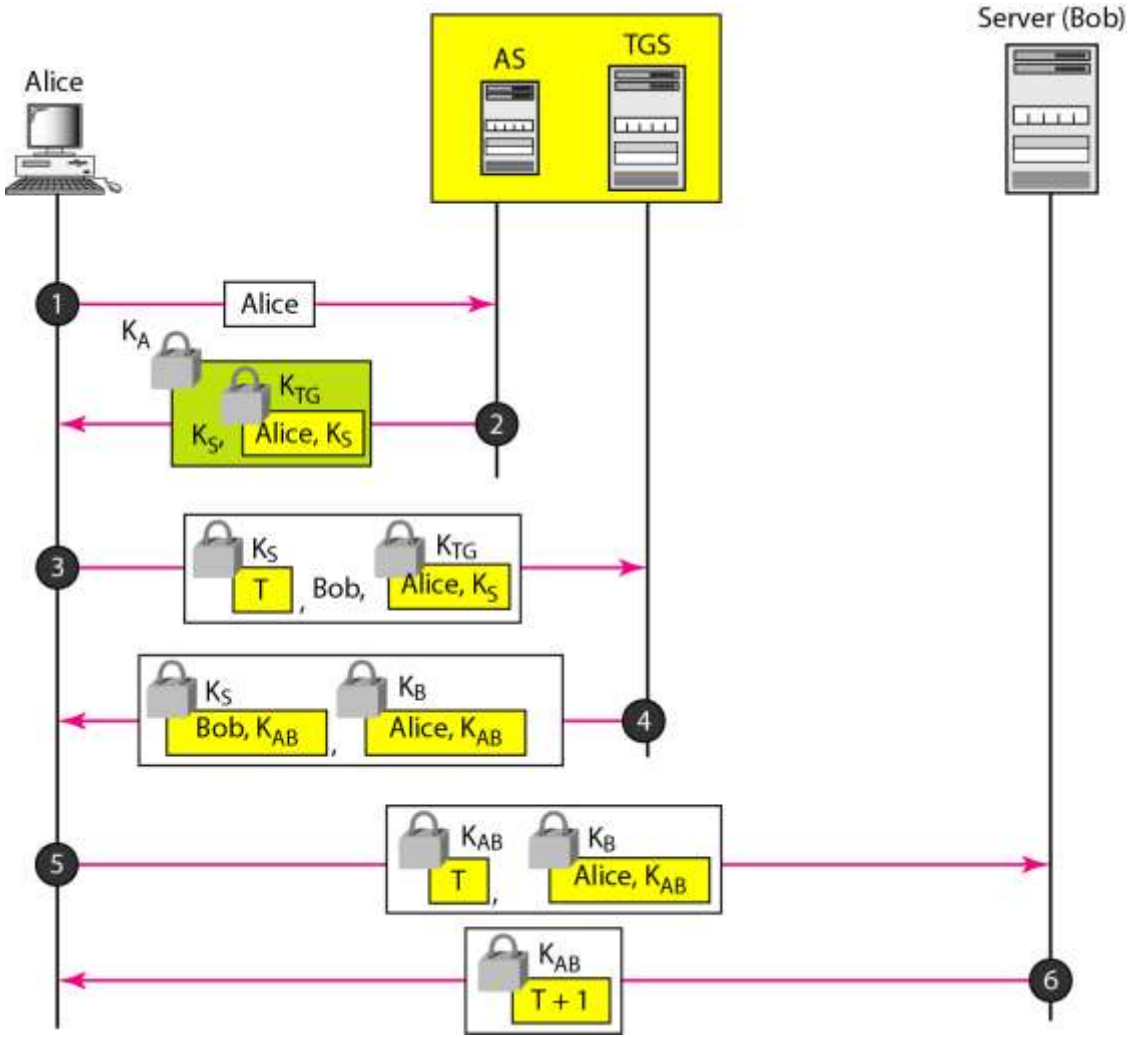
**Figure 31.30** *Creating a session key between Alice and Bob using KDC*



## Figure 31.21 *Kerberos servers*



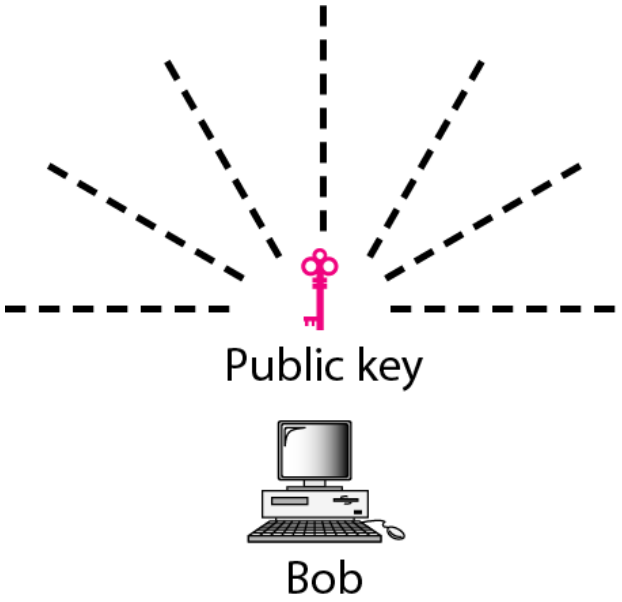
## Figure 31.22 *Kerberos example*



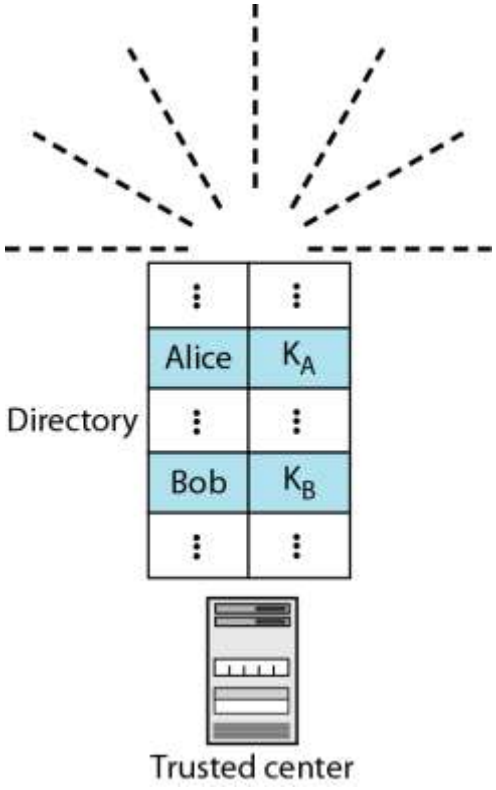
*Note*

**In public-key cryptography, everyone has access to everyone's public key; public keys are available to the public.**

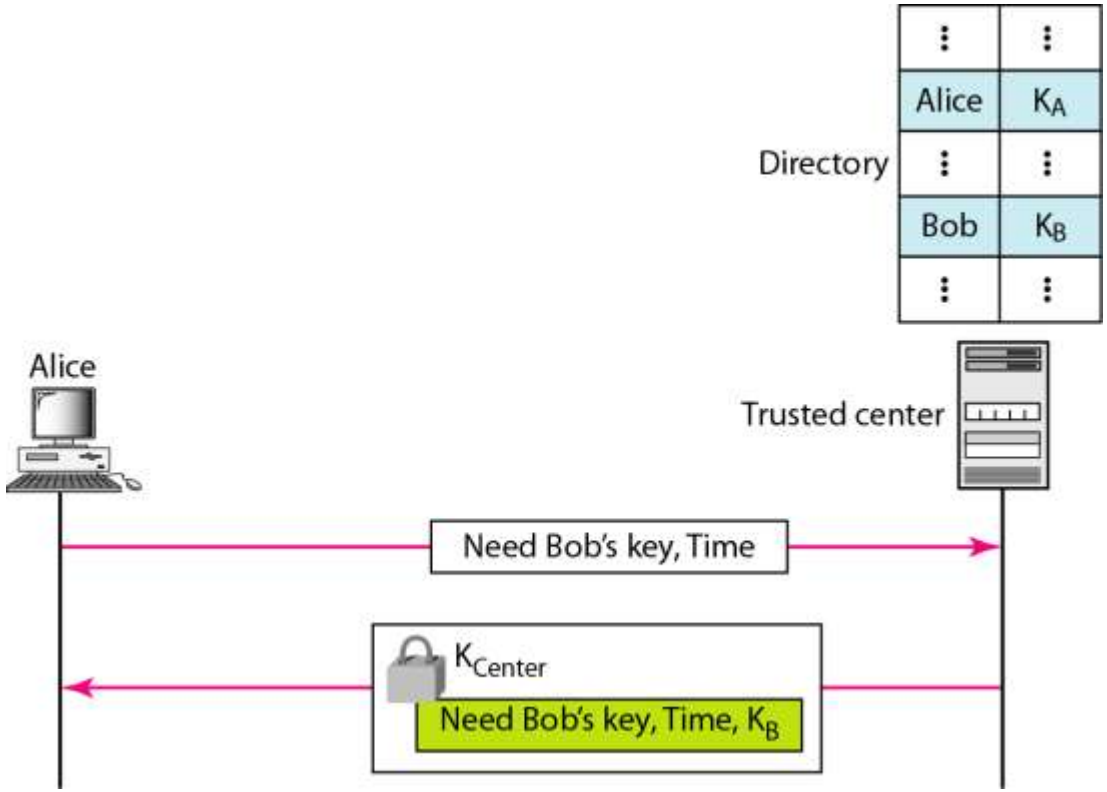
## Figure 31.23 *Announcing a public key*



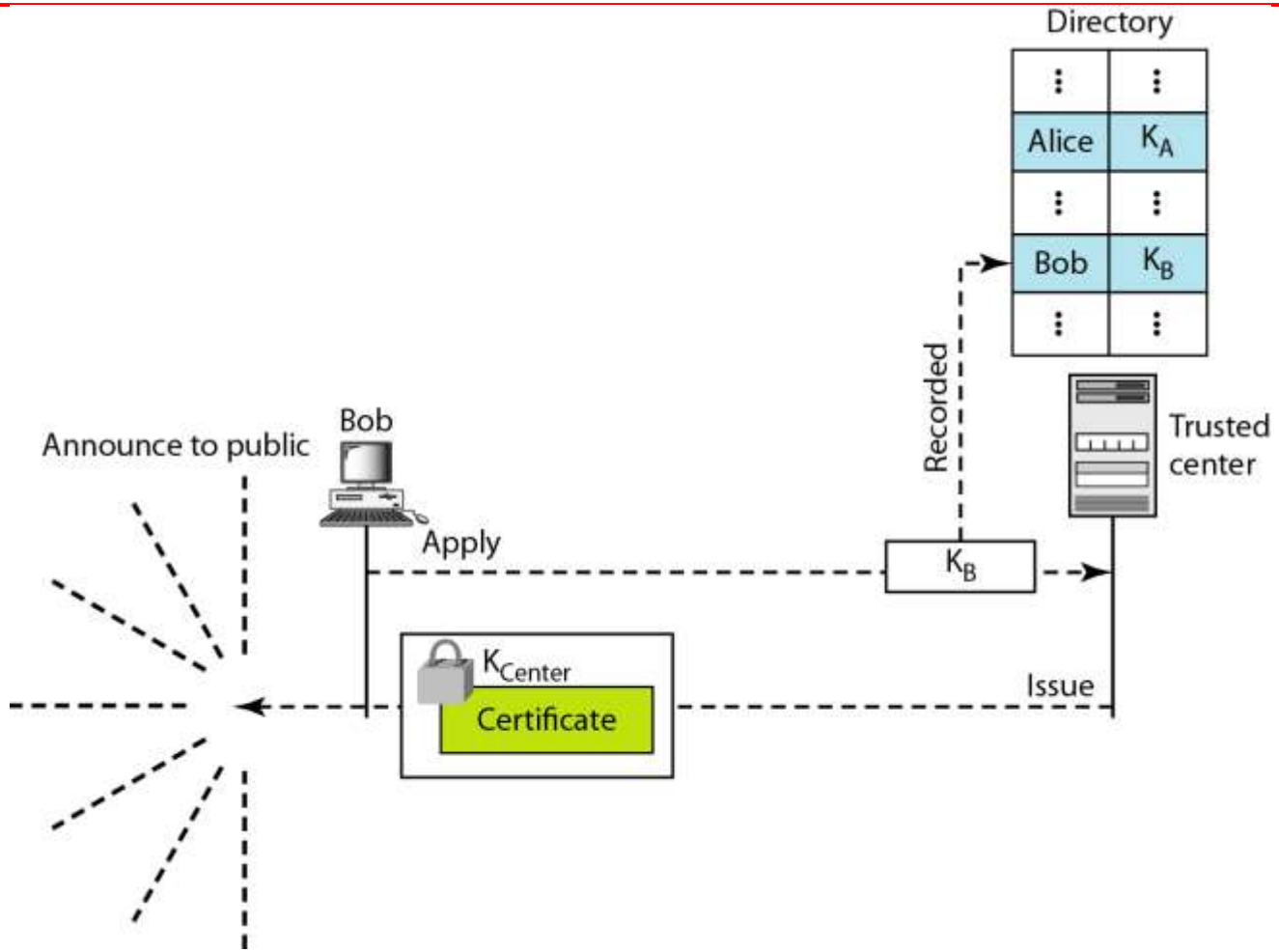
## Figure 31.24 *Trusted center*



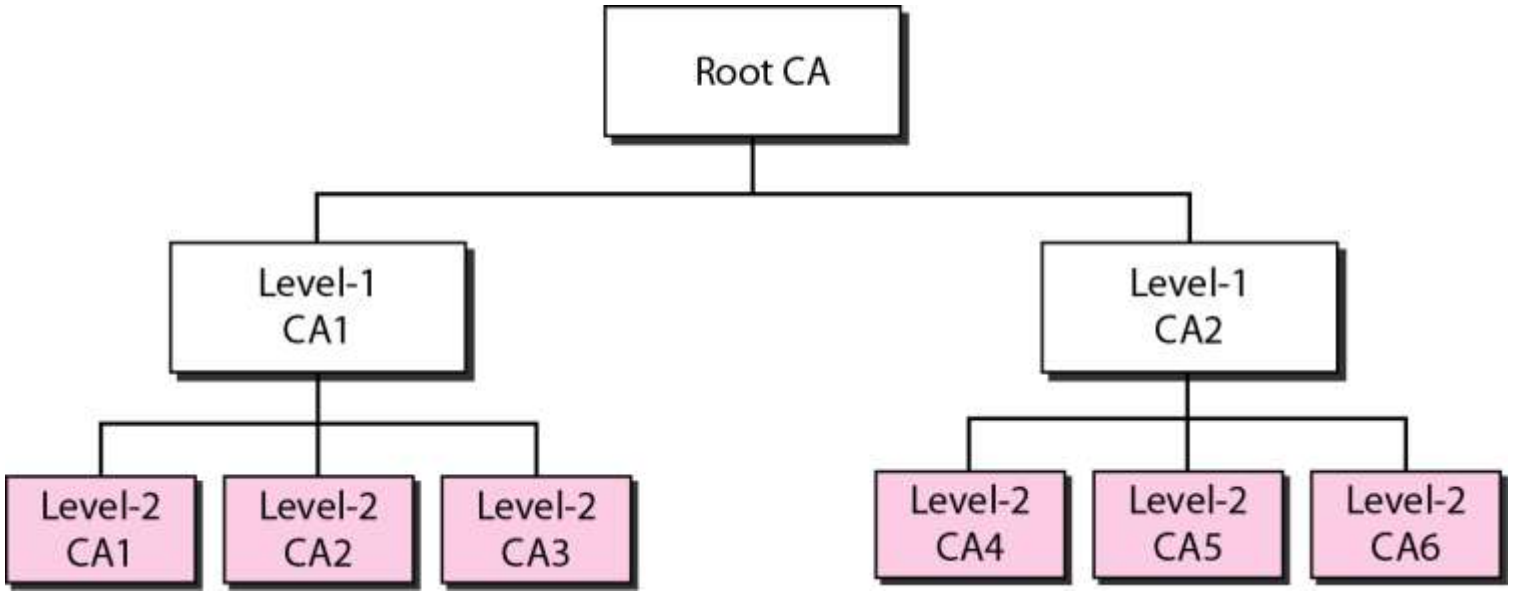
## Figure 31.25 *Controlled trusted center*



## Figure 31.26 *Certification authority*



## Figure 31.27 *PKI hierarchy*



Data Communications and Networking Fourth Edition

Forouzan

# Chapter 23

# Process-to-Process Delivery: UDP, TCP, and SCTP

# 23-1 PROCESS-TO-PROCESS DELIVERY

***The transport layer is responsible for process-to-process delivery—the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship, as we will see later.***

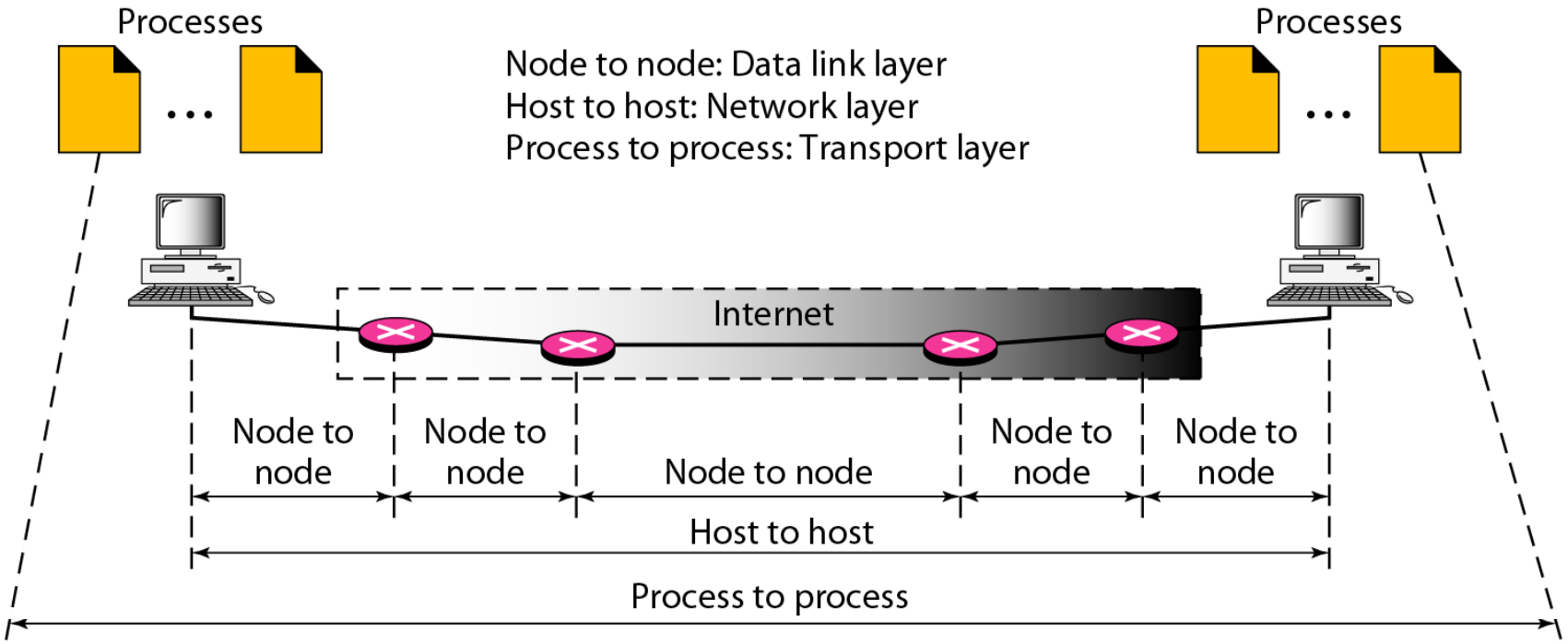
***Topics discussed in this section:***

**Client/Server Paradigm**
**Multiplexing and Demultiplexing**
**Connectionless Versus Connection-Oriented Service**
**Reliable Versus Unreliable**
**Three Protocols**

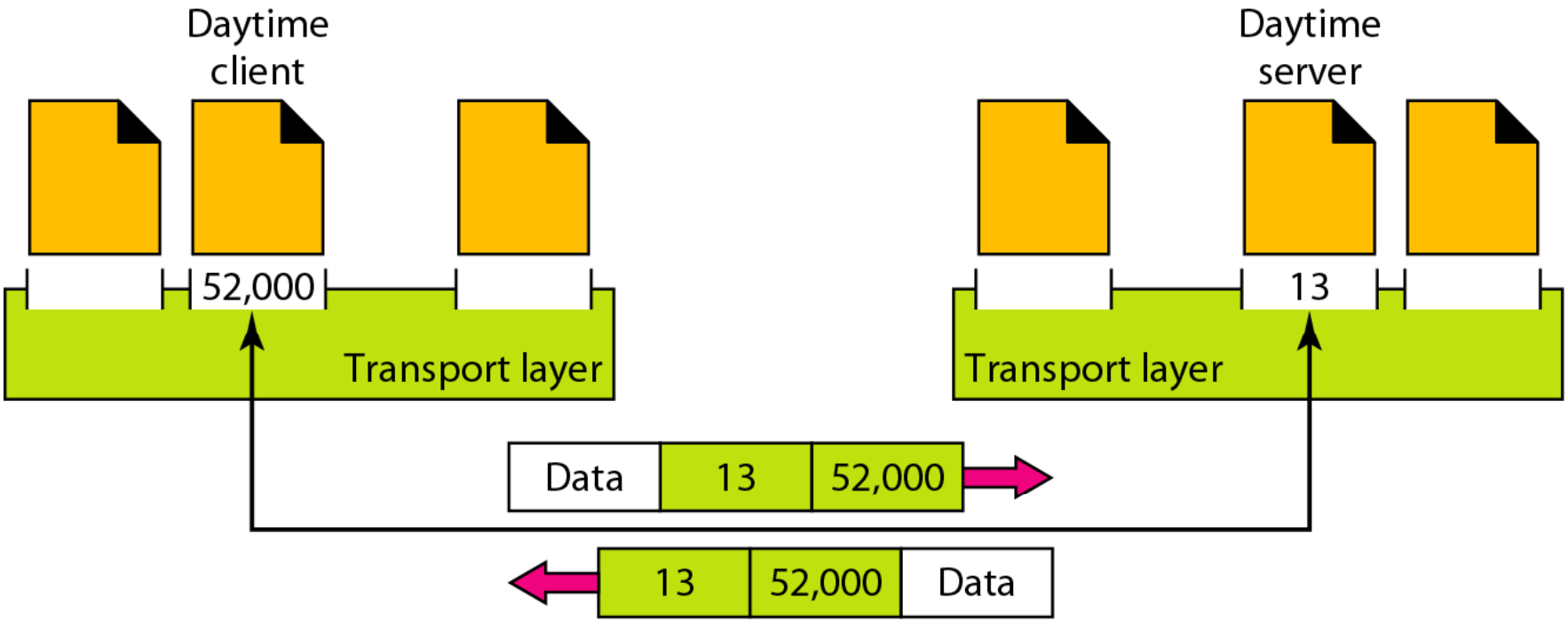
*Note*

**The transport layer is responsible for process-to-process delivery.**

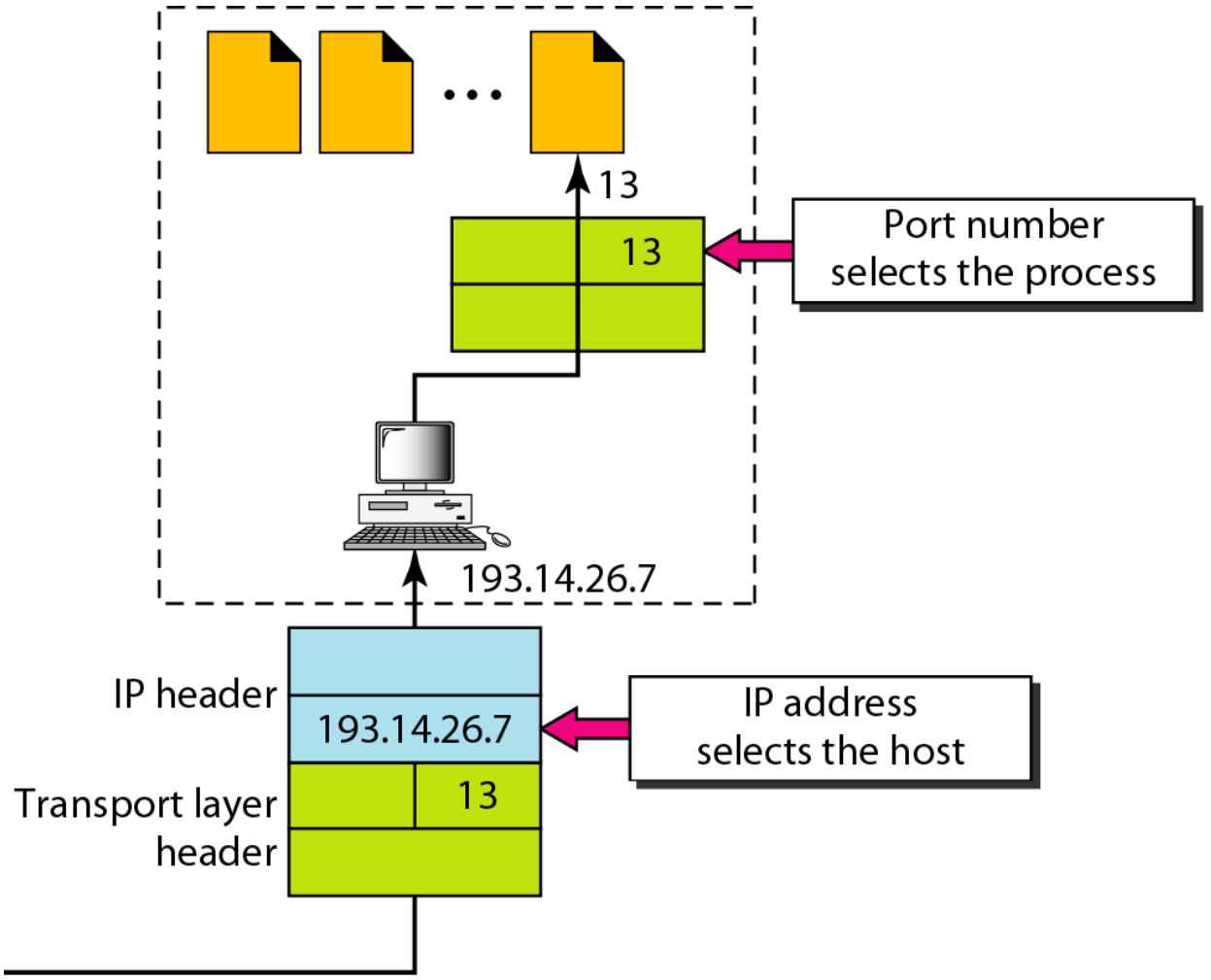
## Figure 23.1 *Types of data deliveries*



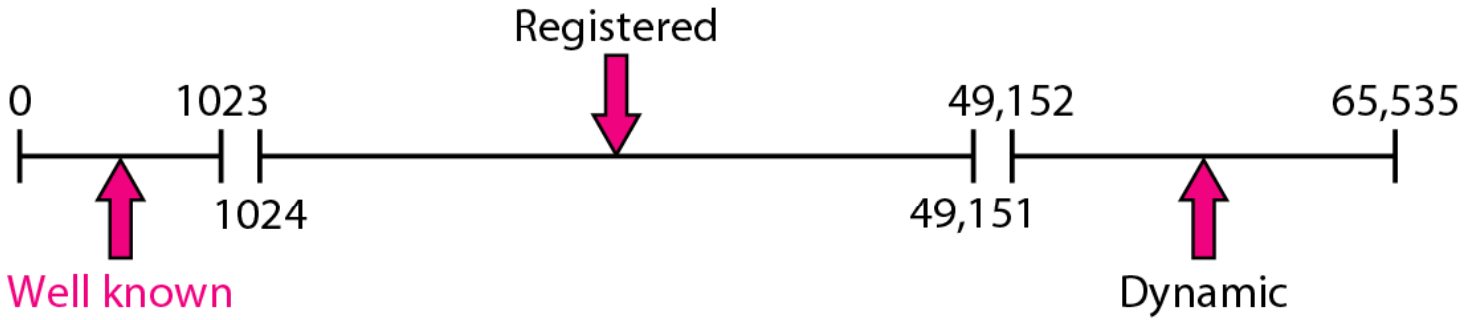
## Figure 23.2 *Port numbers*



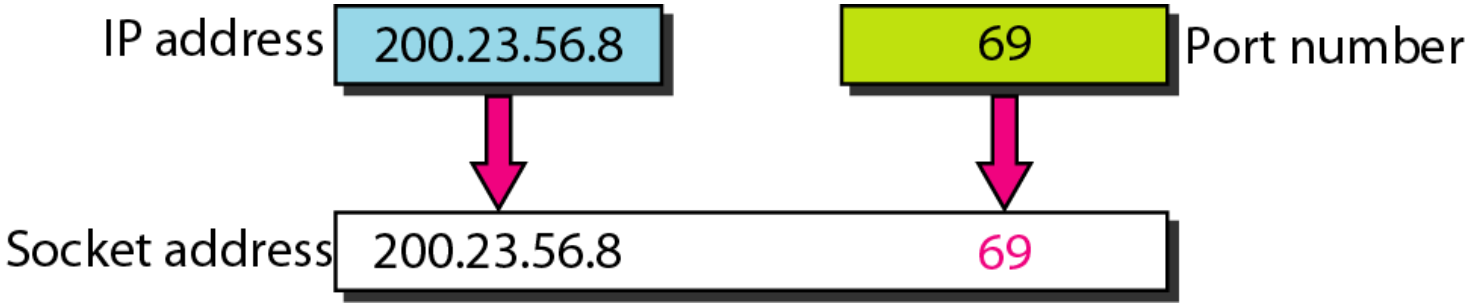
## Figure 23.3 *IP addresses versus port numbers*



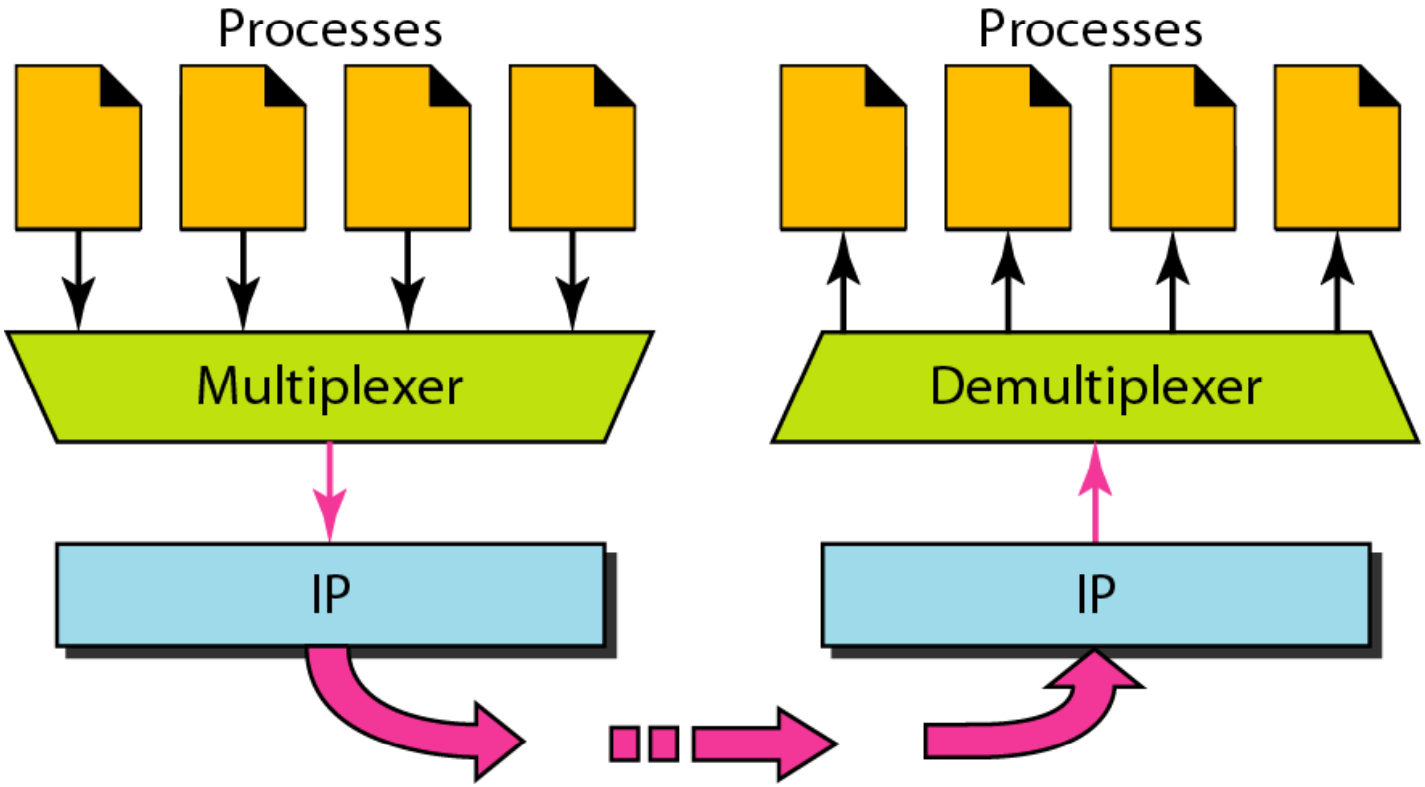
## Figure 23.4 *IANA ranges*



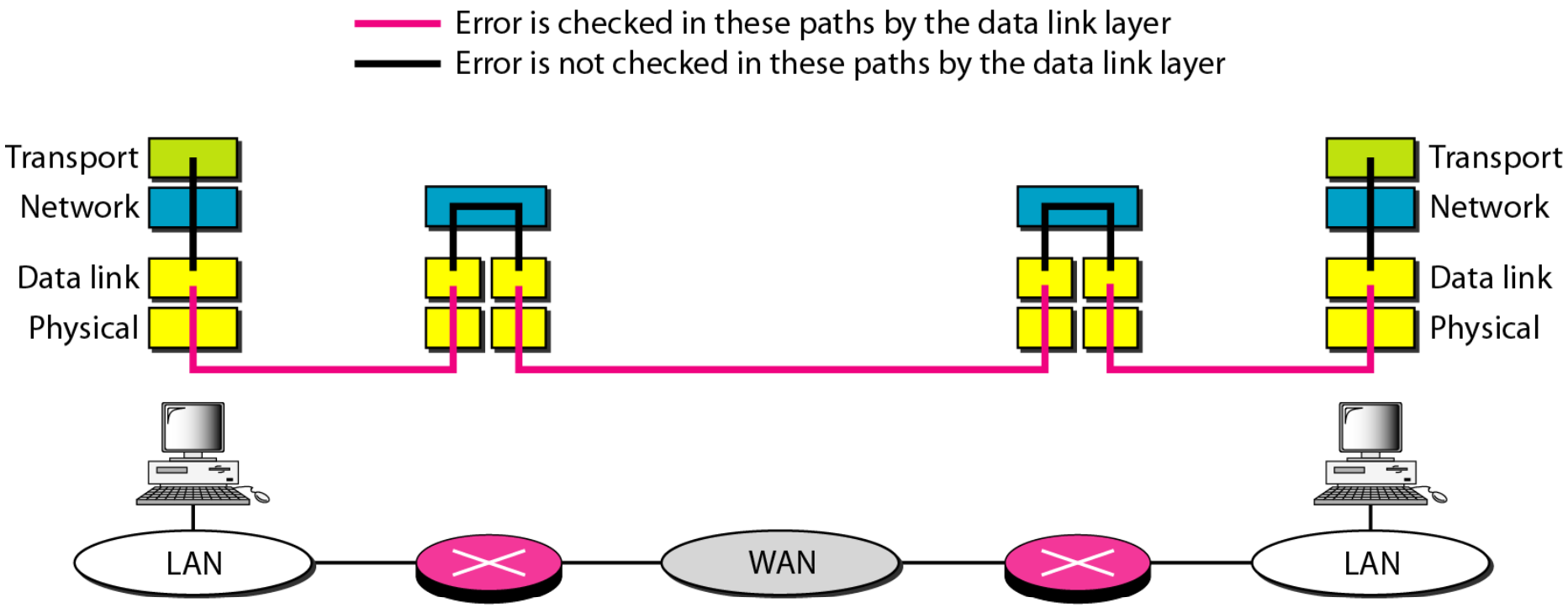
## Figure 23.5 *Socket address*



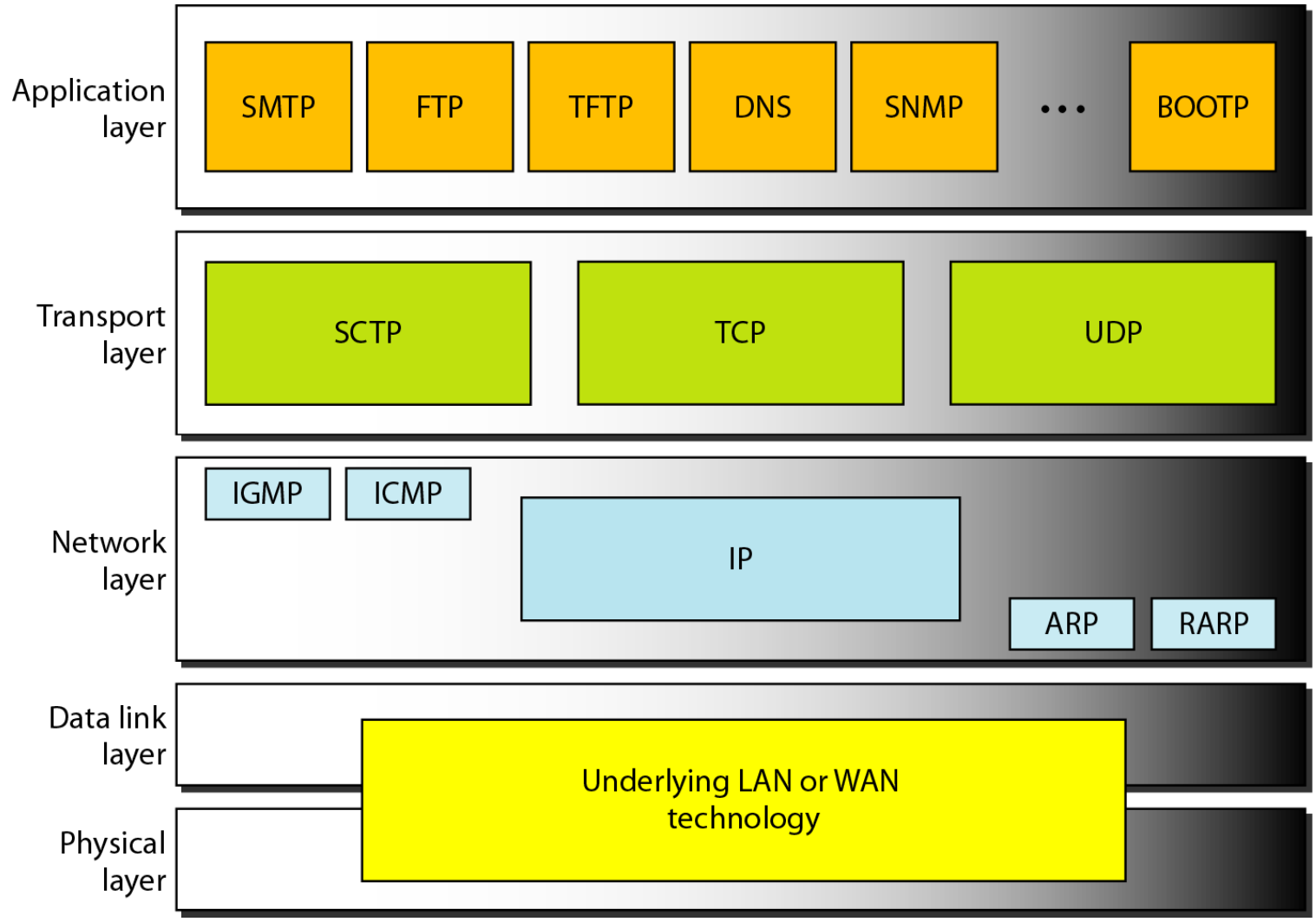
## Figure 23.6 *Multiplexing and demultiplexing*



## Figure 23.7 *Error control*



## Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



# 23-2 USER DATAGRAM PROTOCOL (UDP)

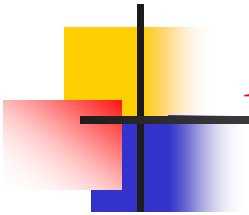
***The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.***

***Topics discussed in this section:***

**Well-Known Ports for UDP**
**User Datagram**
**Checksum**
**UDP Operation**
**Use of UDP**

## Table 23.1 *Well-known ports used with UDP*

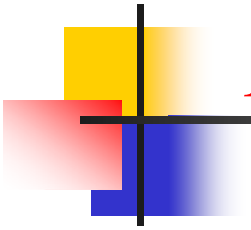
| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 53 | Nameserver | Domain Name Service |
| 67 | BOOTPs | Server port to download bootstrap information |
| 68 | BOOTPc | Client port to download bootstrap information |
| 69 | TFTP | Trivial File Transfer Protocol |
| 111 | RPC | Remote Procedure Call |
| 123 | NTP | Network Time Protocol |
| 161 | SNMP | Simple Network Management Protocol |
| 162 | SNMP | Simple Network Management Protocol (trap) |



# *Example 23.1*

***In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the port for FTP. Note that FTP can use port 21 with either UDP or TCP.***

```
$ grep    ftp   /etc/services
ftp          21/tcp
ftp          21/udp
```

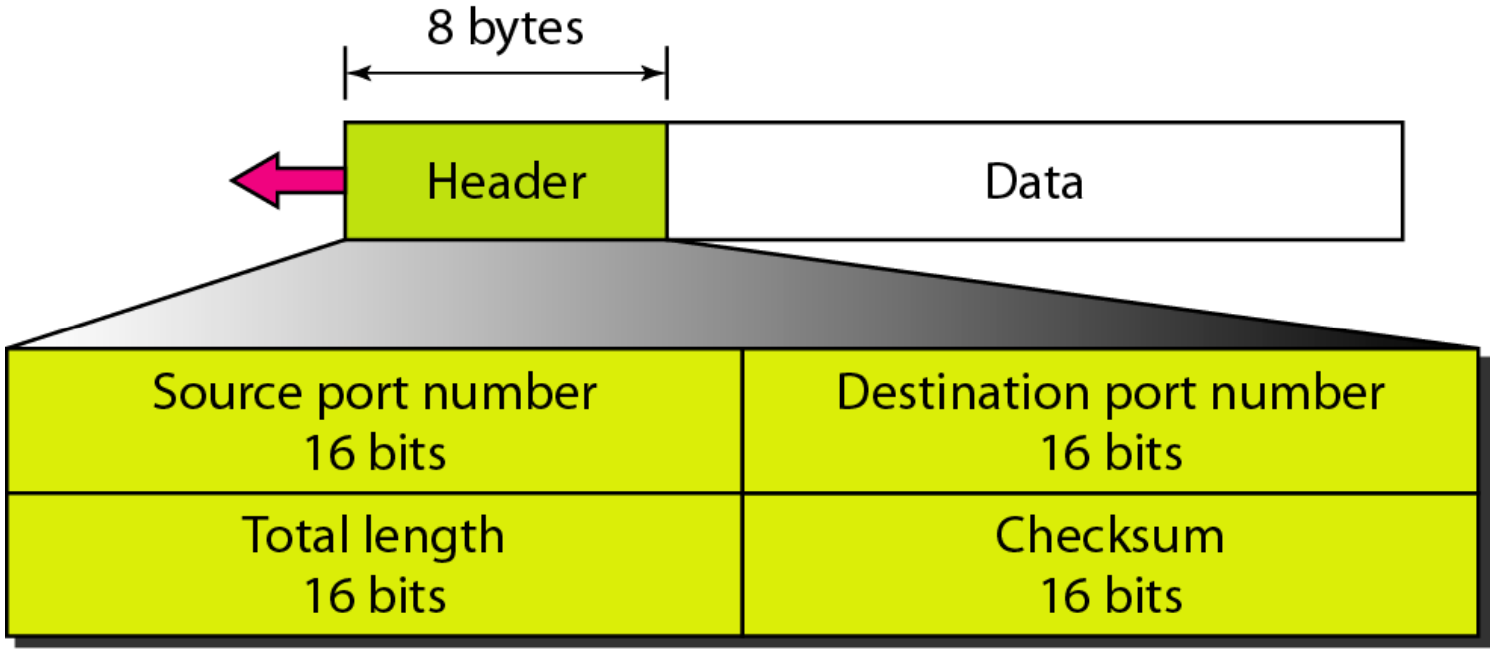


# *Example 23.1 (continued)*

***SNMP uses two port numbers (161 and 162), each for a different purpose, as we will see in Chapter 28.***

```
$ grep      snmp /etc/services
snmp            161/tcp         #Simple Net  Mgmt Proto
snmp            161/udp         #Simple Net  Mgmt Proto
snmptrap        162/udp         #Traps for SNMP
```

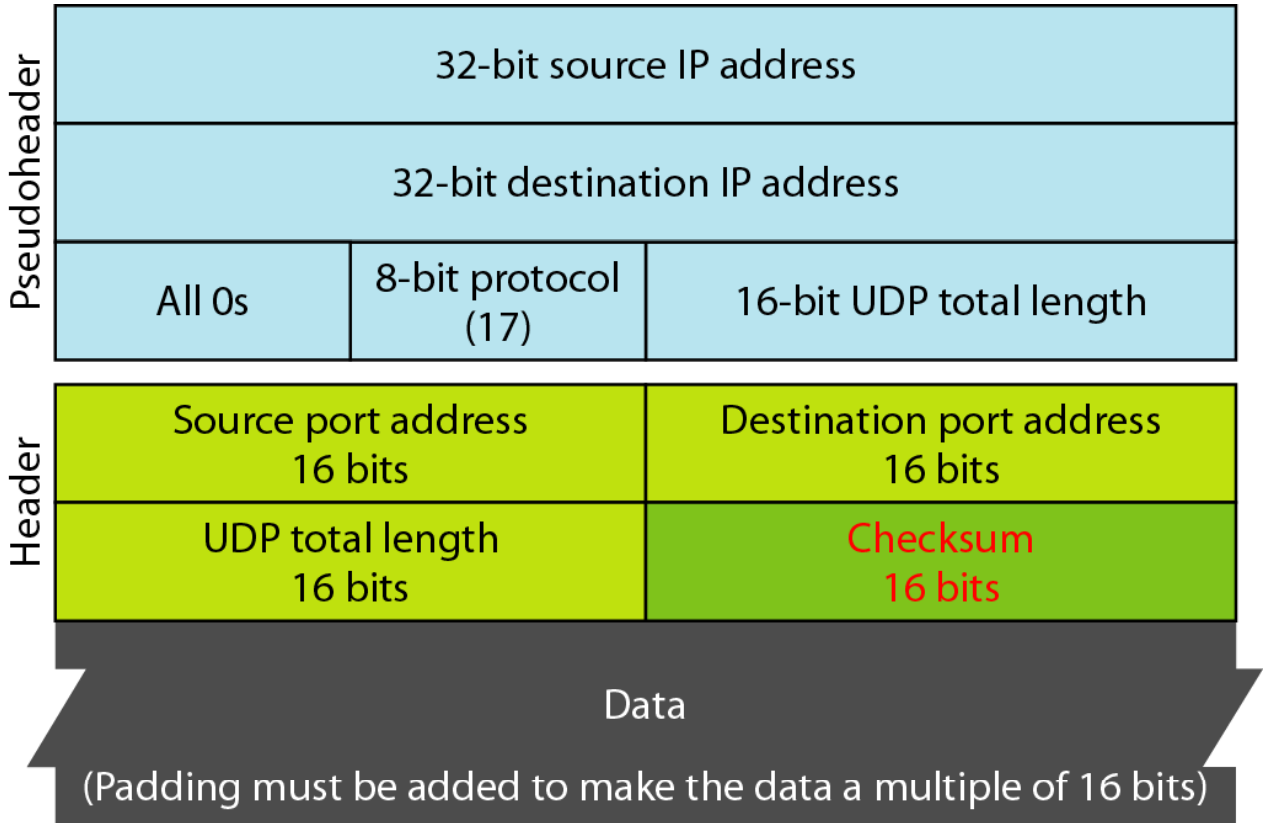
## Figure 23.9 *User datagram format*

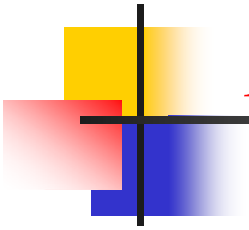


*Note*

**UDP length**
**= IP length – IP header's length**

## Figure 23.10 *Pseudoheader for checksum calculation*

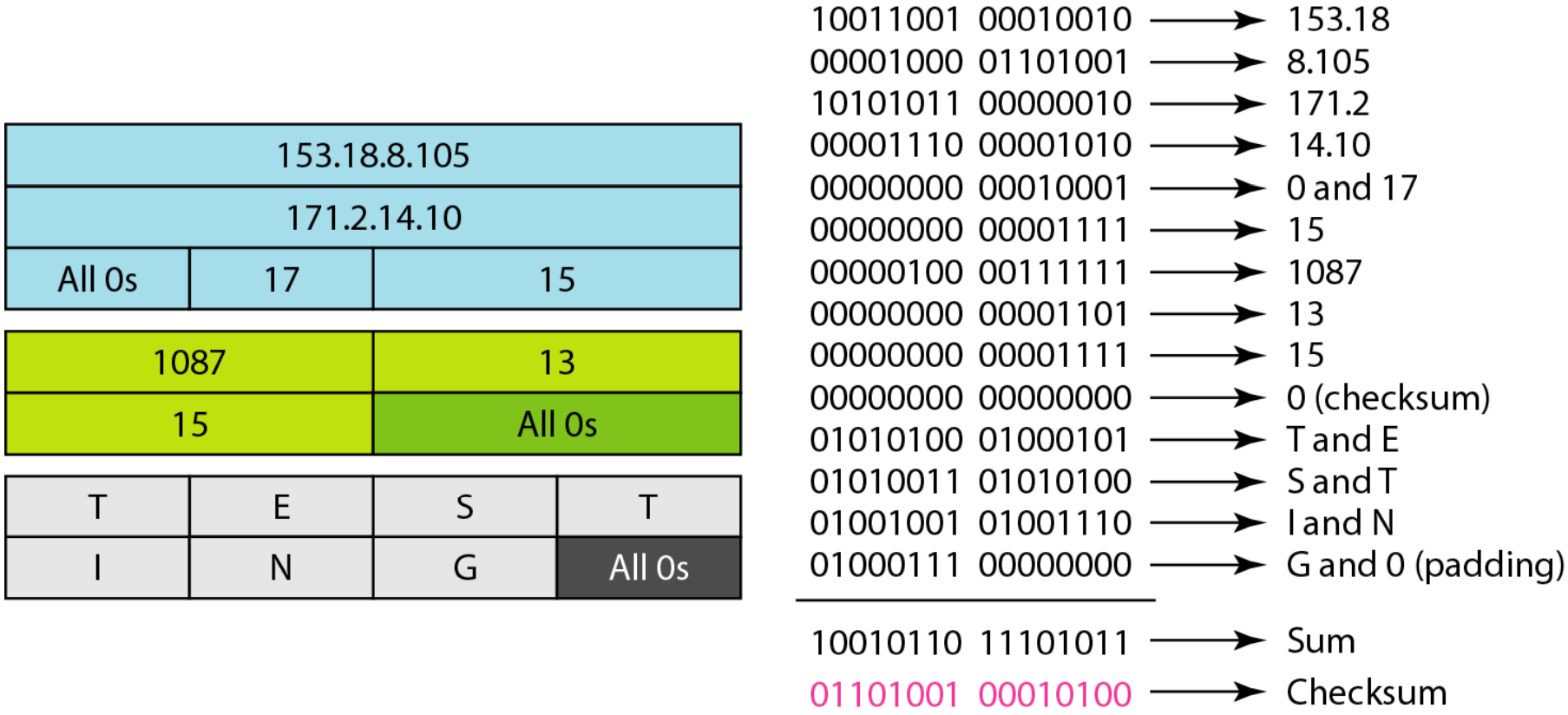




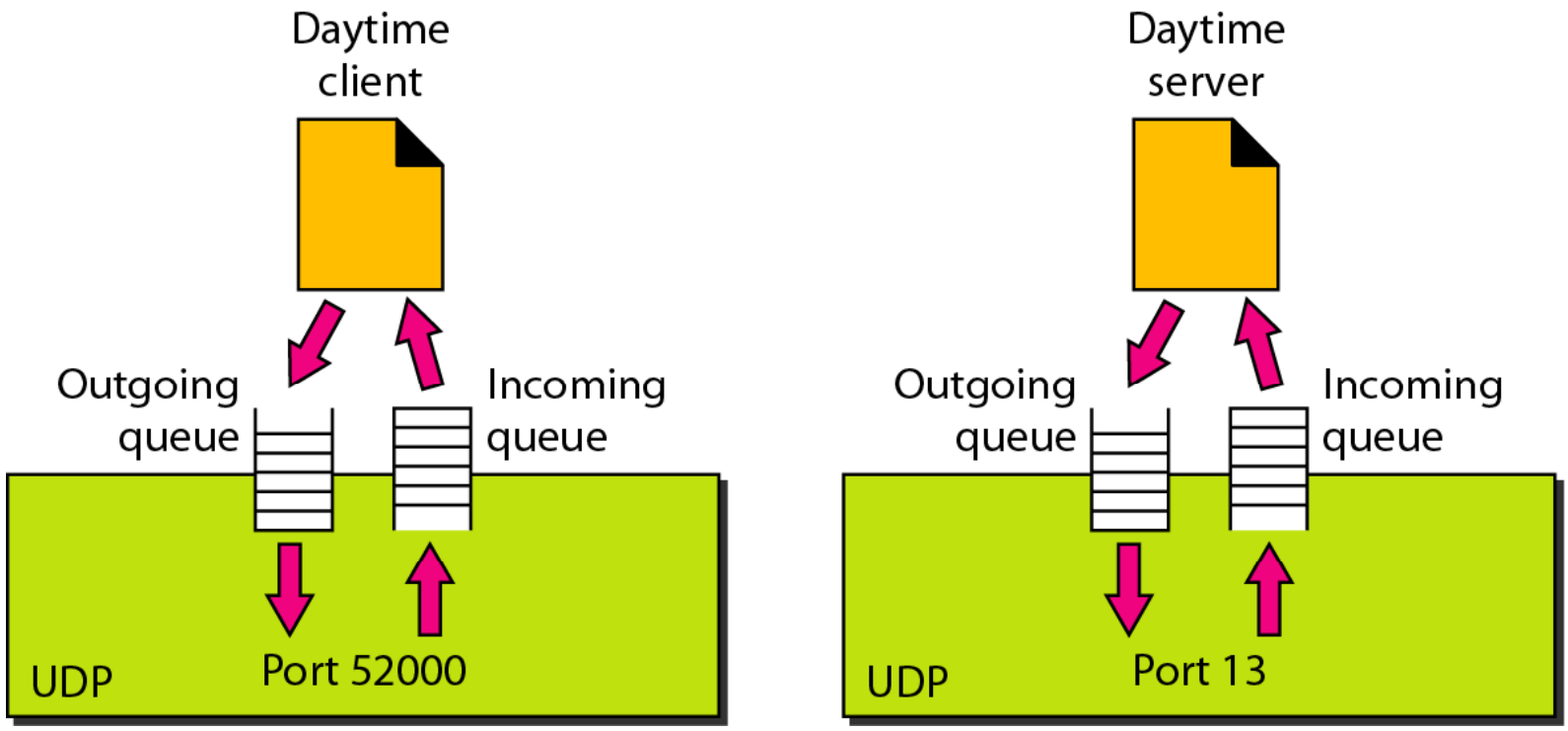
# *Example 23.2*

***Figure 23.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP.***

## Figure 23.11 *Checksum calculation of a simple UDP user datagram*



## Figure 23.12 *Queues in UDP*



# 23-3 TCP

***TCP is a connection-oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.***

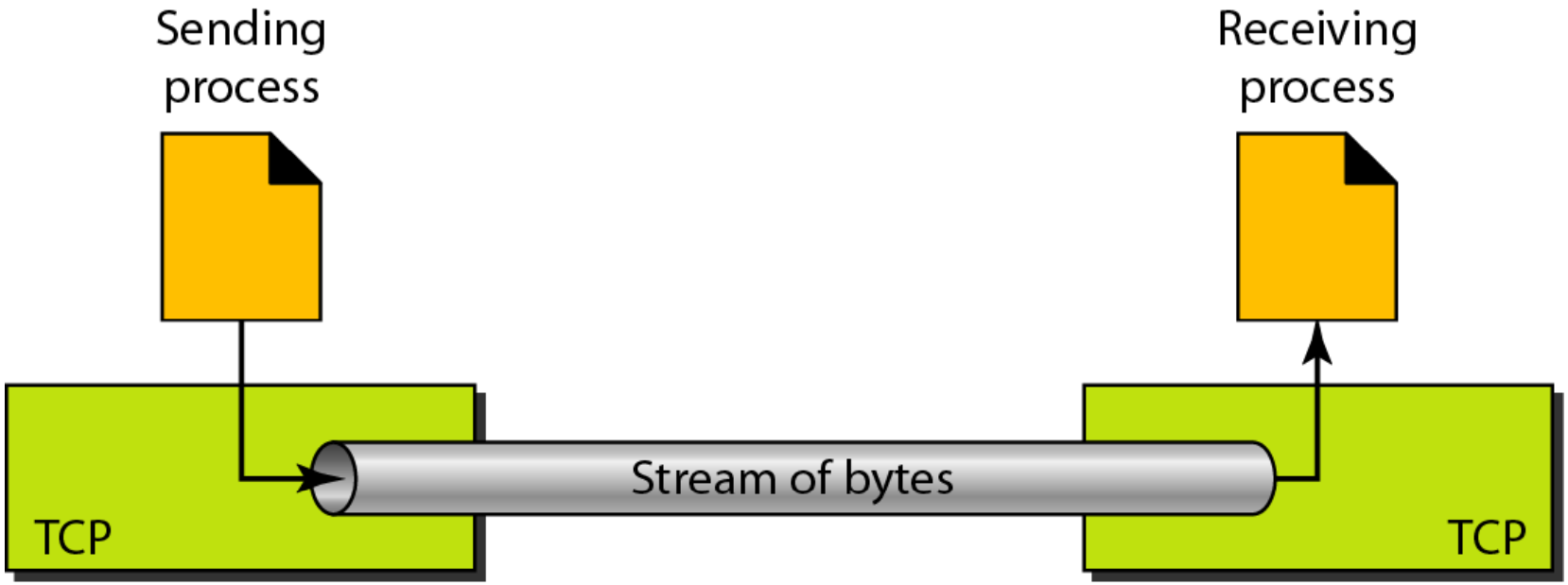
***Topics discussed in this section:***

**TCP Services**
**TCP Features**
**Segment**
**A TCP Connection**
**Flow Control**
**Error Control**

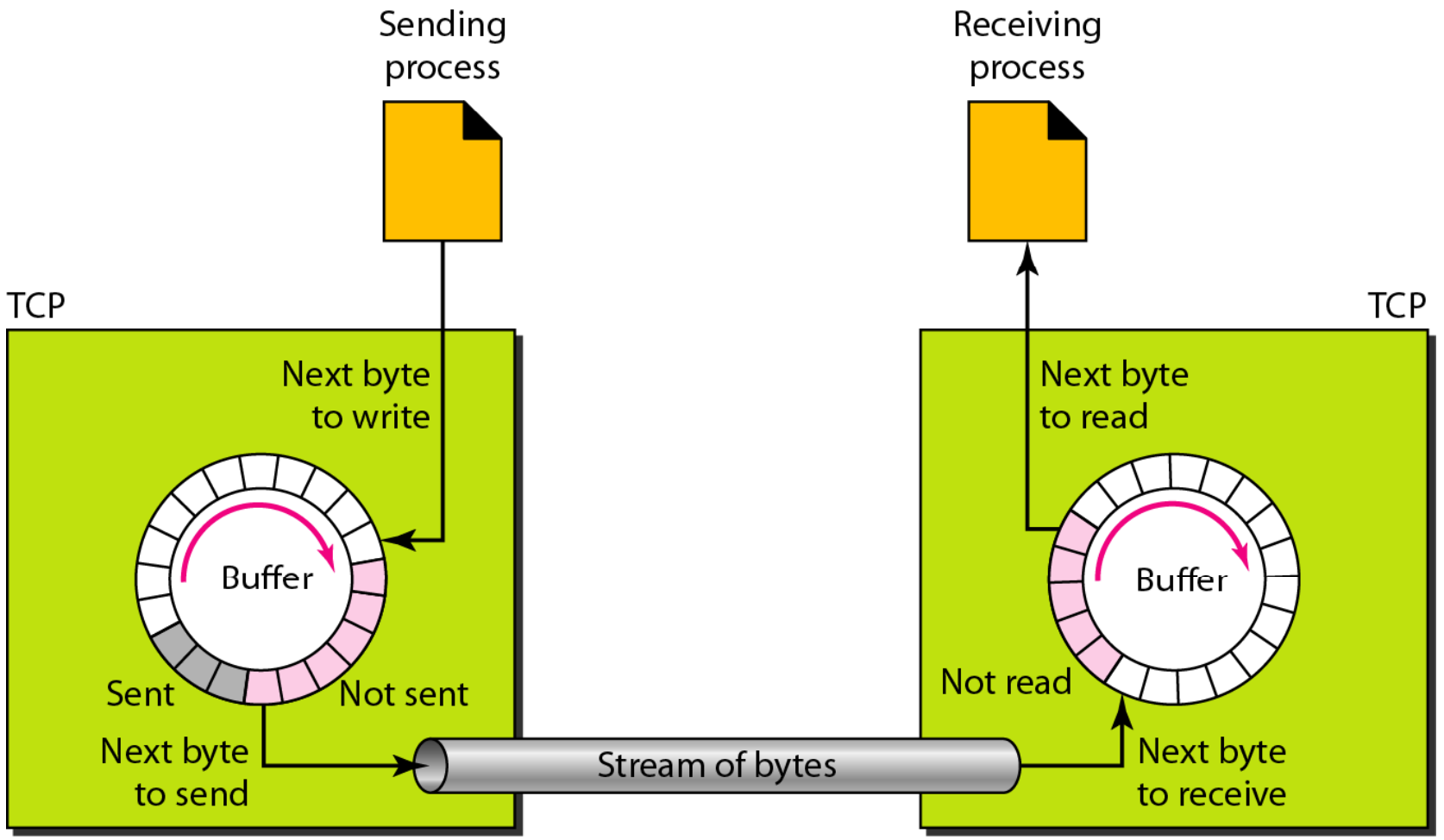
## Table 23.2 *Well-known ports used by TCP*

| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 20 | FTP, Data | File Transfer Protocol (data connection) |
| 21 | FTP, Control | File Transfer Protocol (control connection) |
| 23 | TELNET | Terminal Network |
| 25 | SMTP | Simple Mail Transfer Protocol |
| 53 | DNS | Domain Name Server |
| 67 | BOOTP | Bootstrap Protocol |
| 79 | Finger | Finger |
| 80 | HTTP | Hypertext Transfer Protocol |
| 111 | RPC | Remote Procedure Call |

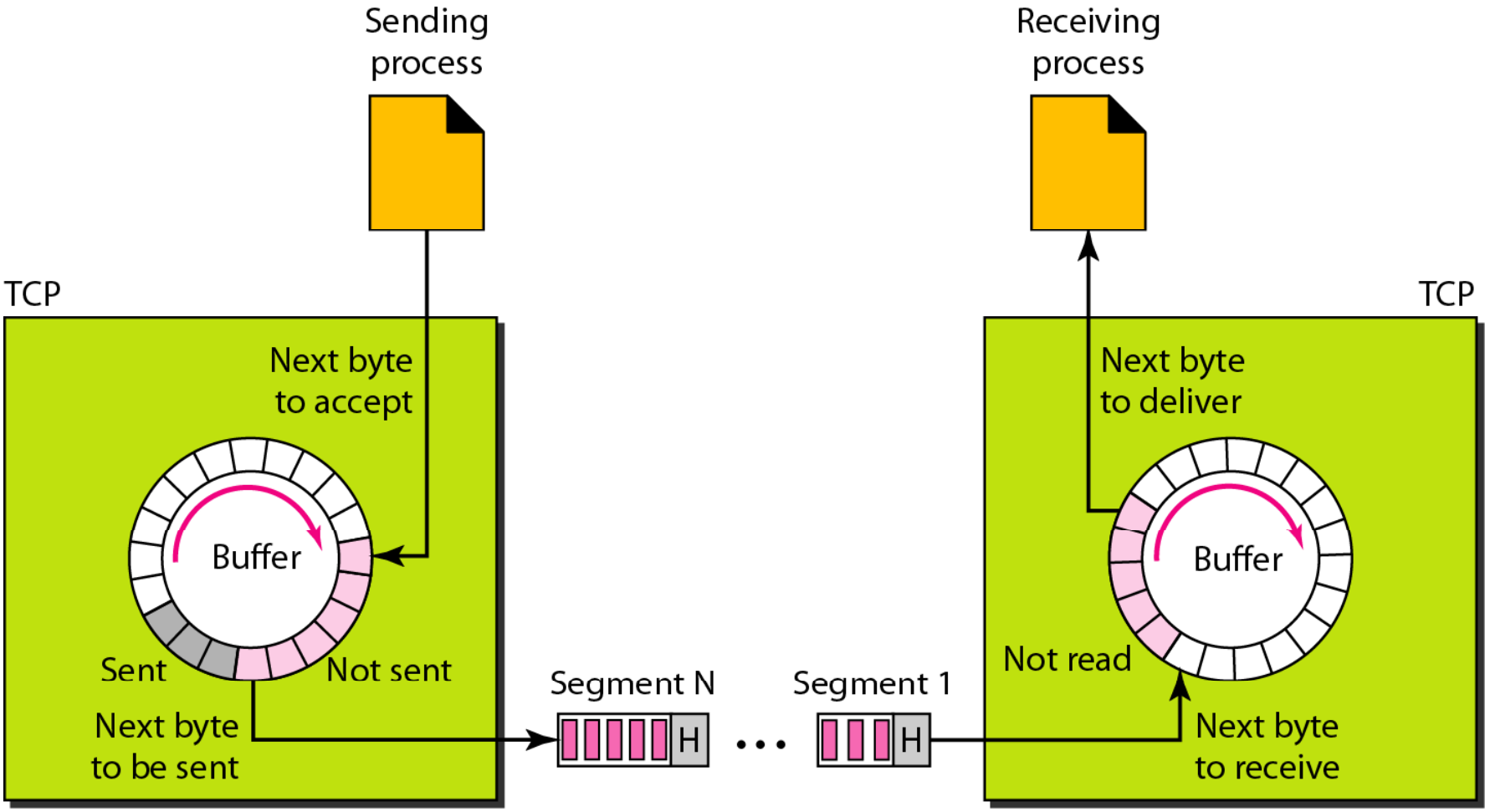
## Figure 23.13 *Stream delivery*



## Figure 23.14 *Sending and receiving buffers*



## Figure 23.15 *TCP segments*



*Note*

**The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.**

# Example 23.3

*The following shows the sequence number for each segment:*

| | | |
|---|---|---|
| Segment 1 | → | Sequence Number: 10,001 (range: 10,001 to 11,000) |
| Segment 2 | → | Sequence Number: 11,001 (range: 11,001 to 12,000) |
| Segment 3 | → | Sequence Number: 12,001 (range: 12,001 to 13,000) |
| Segment 4 | → | Sequence Number: 13,001 (range: 13,001 to 14,000) |
| Segment 5 | → | Sequence Number: 14,001 (range: 14,001 to 15,000) |

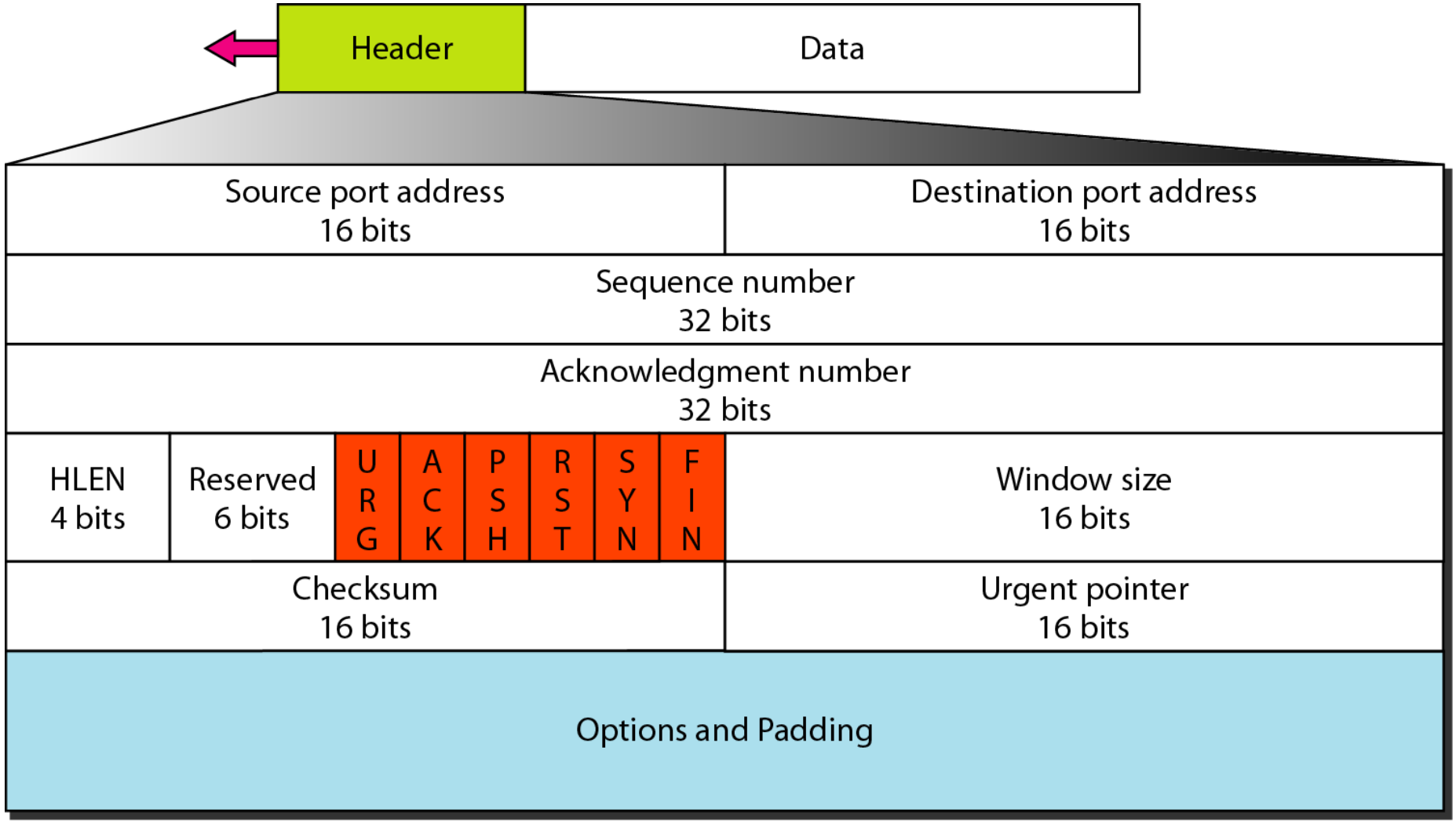
*Note*

**The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.**

*Note*

**The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.**

## Figure 23.16 *TCP segment format*



## Figure 23.17 *Control field*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push

RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

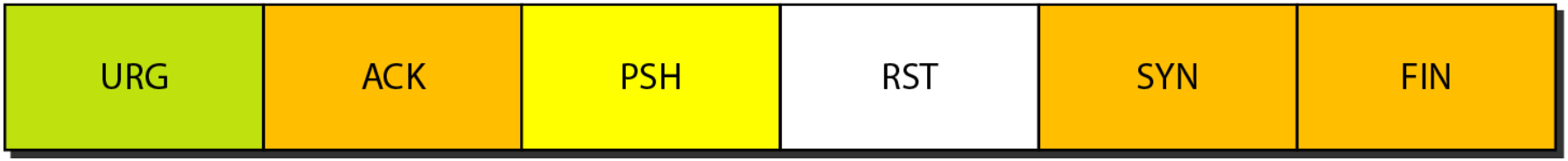
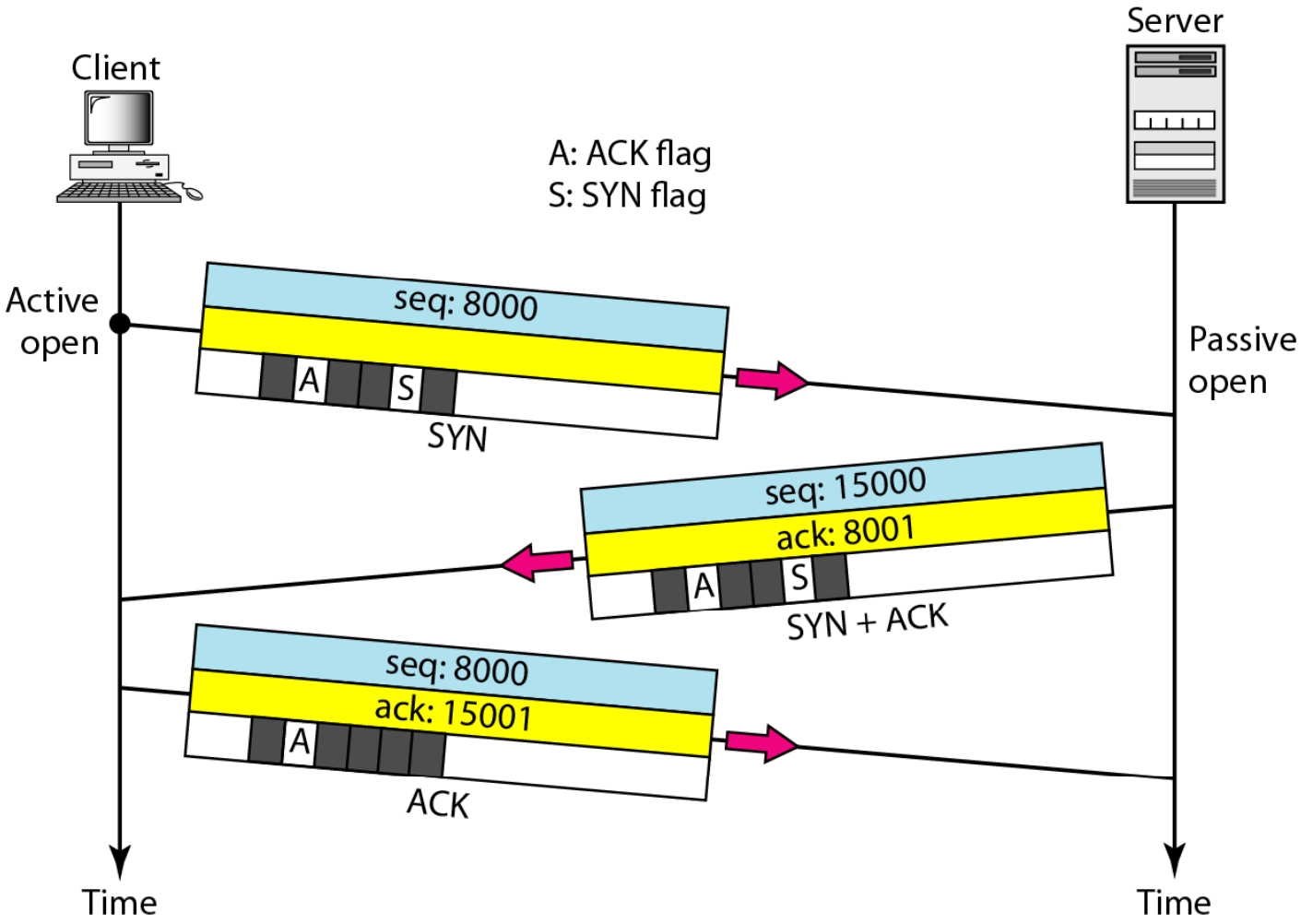


**Table 23.3** ***Description of flags in the control field***

| *Flag* | *Description* |
|---|---|
| URG | The value of the urgent pointer field is valid. |
| ACK | The value of the acknowledgment field is valid. |
| PSH | Push the data. |
| RST | Reset the connection. |
| SYN | Synchronize sequence numbers during connection. |
| FIN | Terminate the connection. |

## Figure 23.18 *Connection establishment using three-way handshaking*



*Note*

**A SYN segment cannot carry data, but it consumes one sequence number.**

*Note*

**A SYN + ACK segment cannot carry data, but does consume one sequence number.**

*Note*

**An ACK segment, if carrying no data, consumes no sequence number.**

## Figure 23.19 *Data transfer*

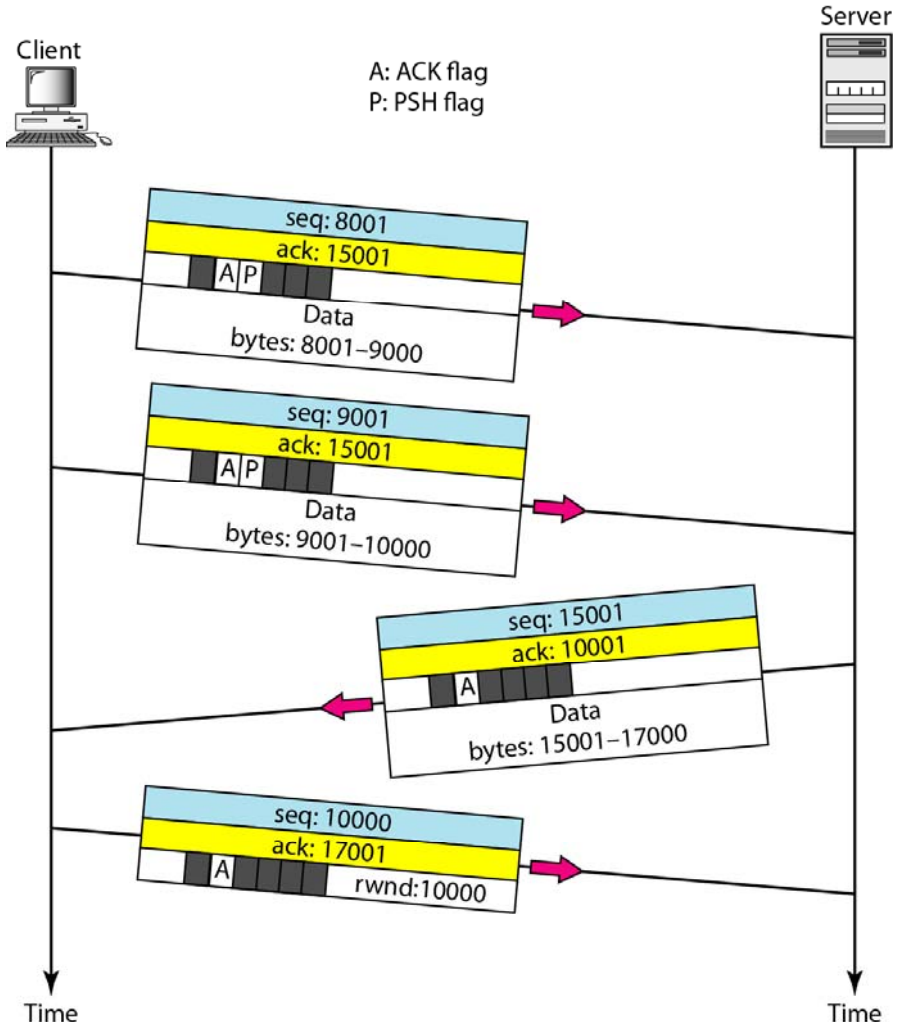
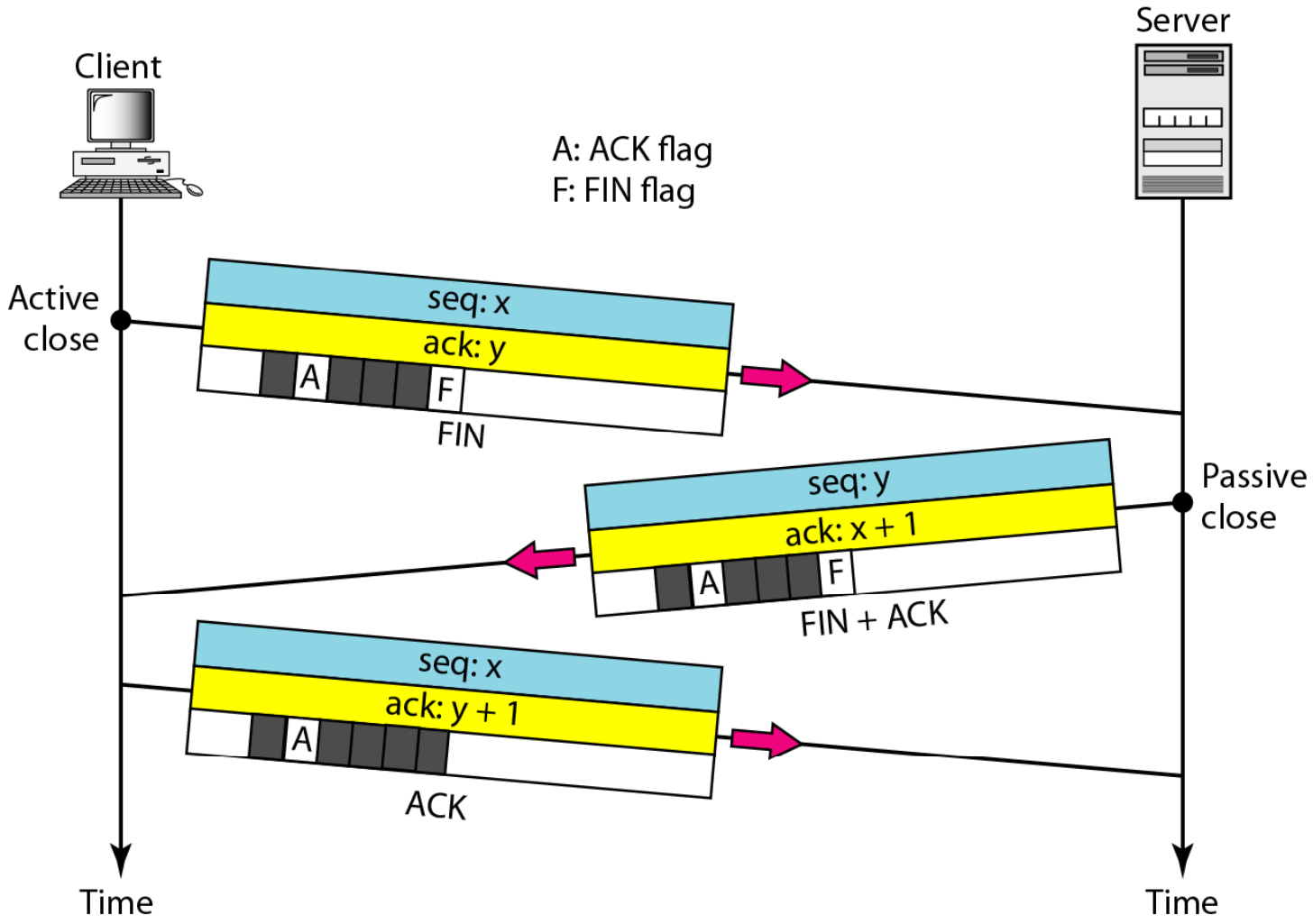


**Figure 23.20** *Connection termination using three-way handshaking*



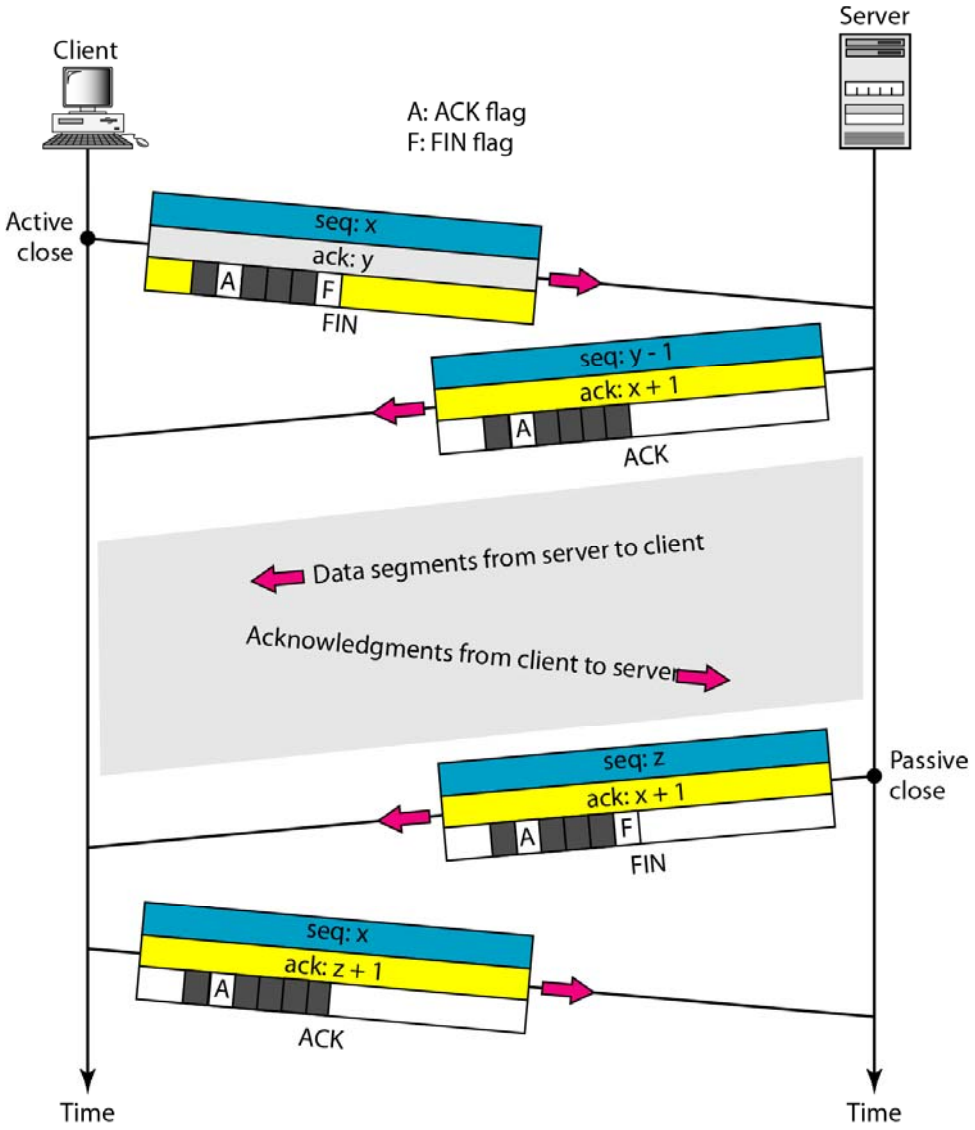
*Note*

**The FIN segment consumes one sequence number if it does not carry data.**

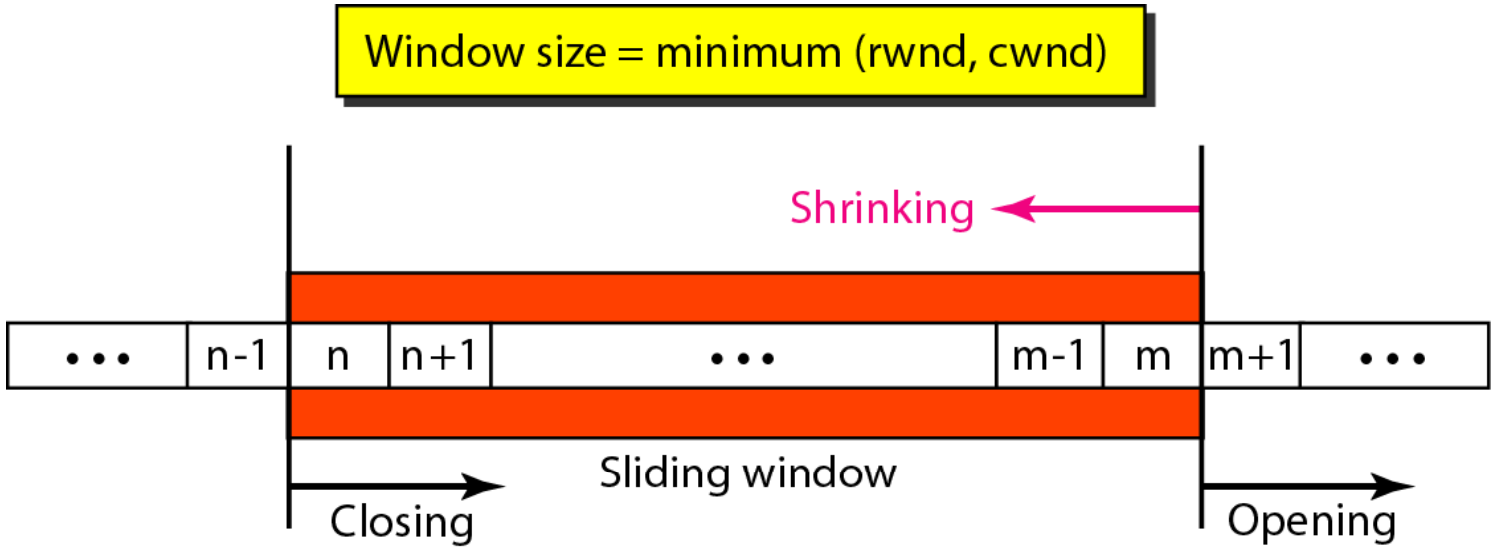
*Note*

**The FIN + ACK segment consumes one sequence number if it does not carry data.**

# Figure 23.21 *Half-close*

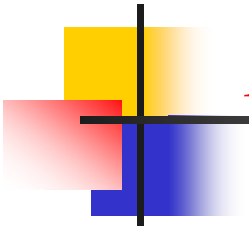


## Figure 23.22 *Sliding window*



Note

**A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.**
**TCP sliding windows are byte-oriented.**

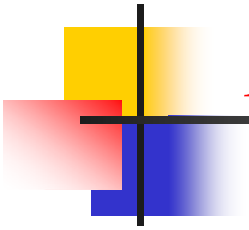


# Example 23.4

*What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?*

## Solution

*The value of rwnd = 5000 − 1000 = 4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.*

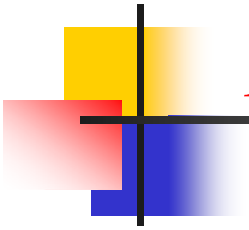


# Example 23.5

*What is the size of the window for host A if the value of rwnd is 3000 bytes and the value of cwnd is 3500 bytes?*

## Solution

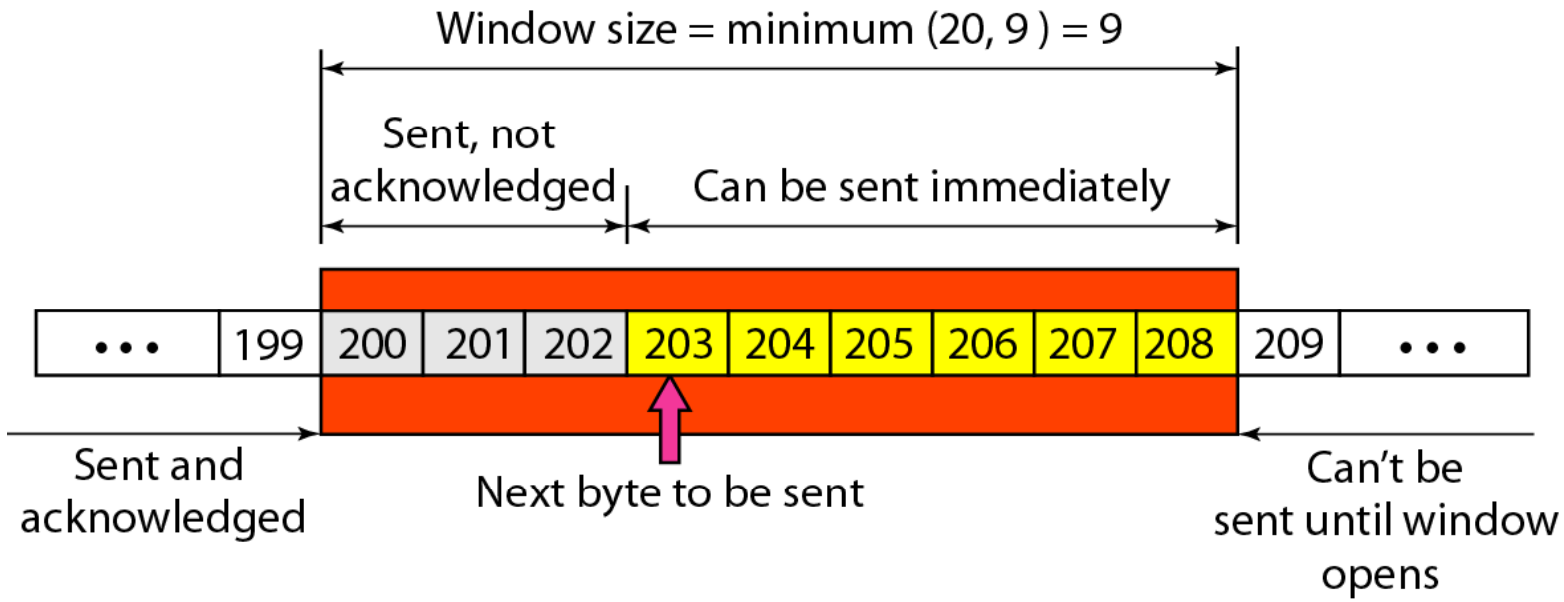
*The size of the window is the smaller of rwnd and cwnd, which is 3000 bytes.*



# *Example 23.6*

***Figure 23.23 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd, or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.***

## Figure 23.23 *Example 23.6*



Note

## Some points about TCP sliding windows:

- The size of the window is the lesser of rwnd and cwnd.
- The source does not have to send a full window's worth of data.
- The window can be opened or closed by the receiver, but should not be shrunk.
- The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

*Note*

**ACK segments do not consume sequence numbers and are not acknowledged.**

*Note*

**In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.**

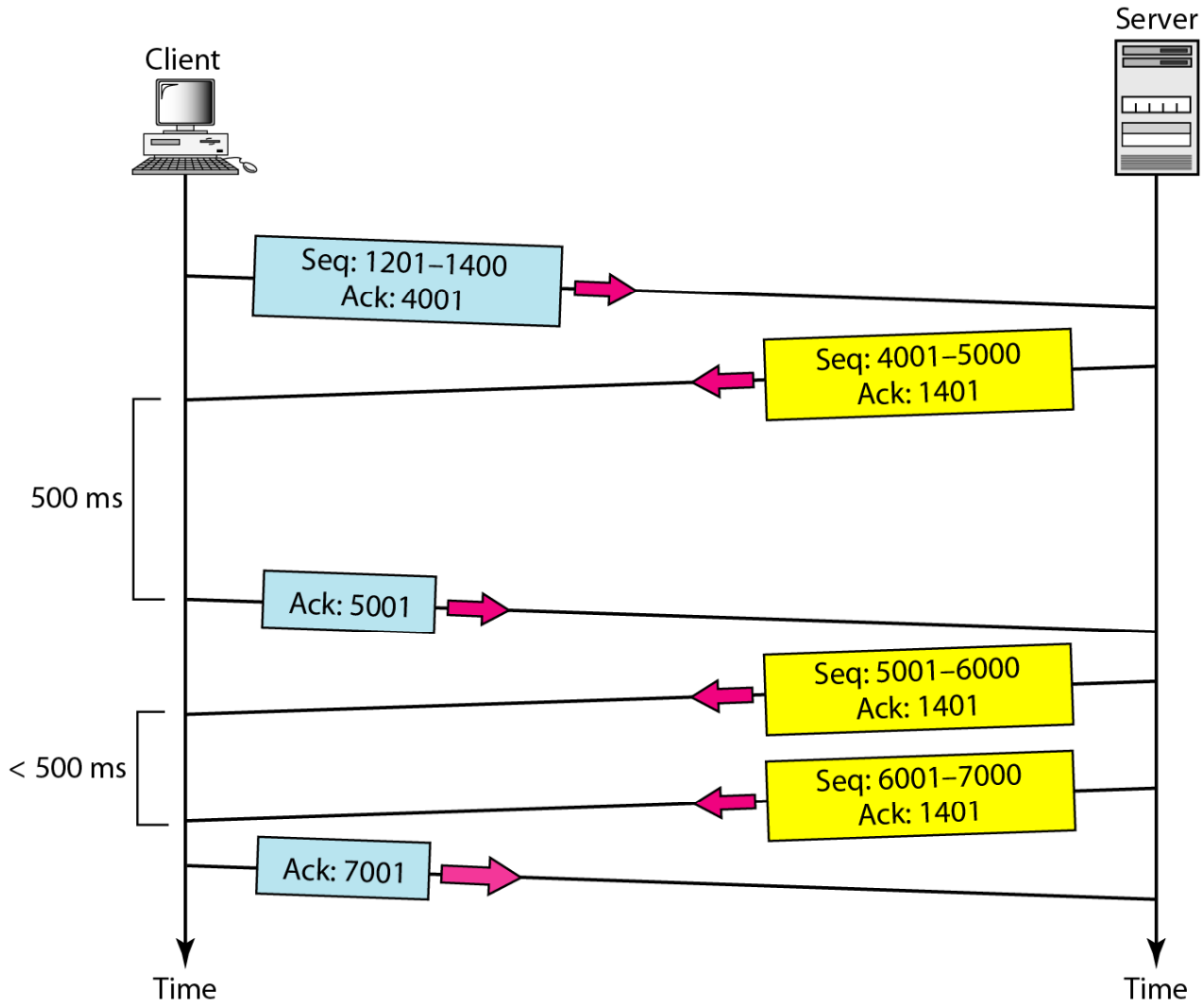
*Note*

**No retransmission timer is set for an ACK segment.**

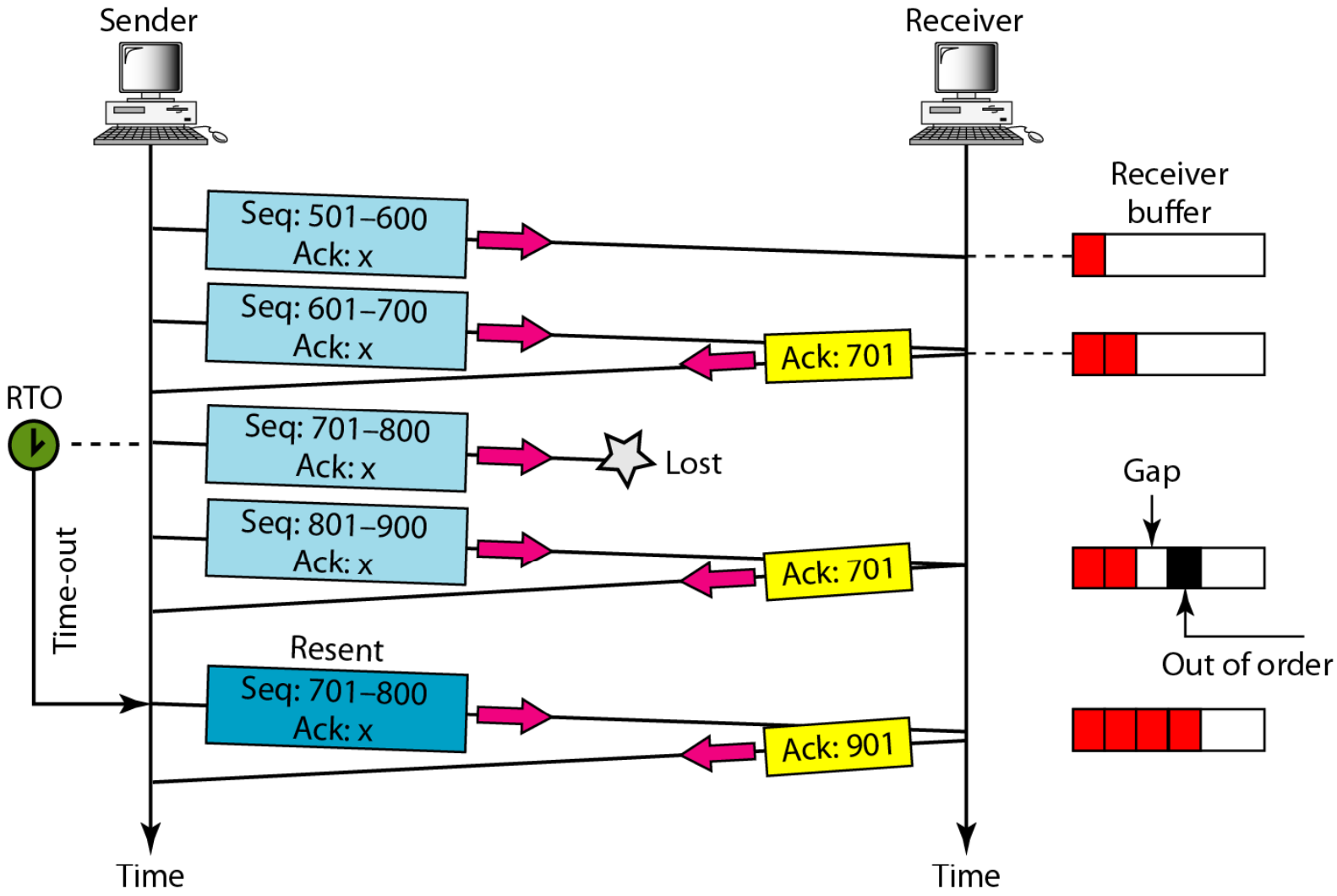
*Note*

**Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process.**

## Figure 23.24 *Normal operation*



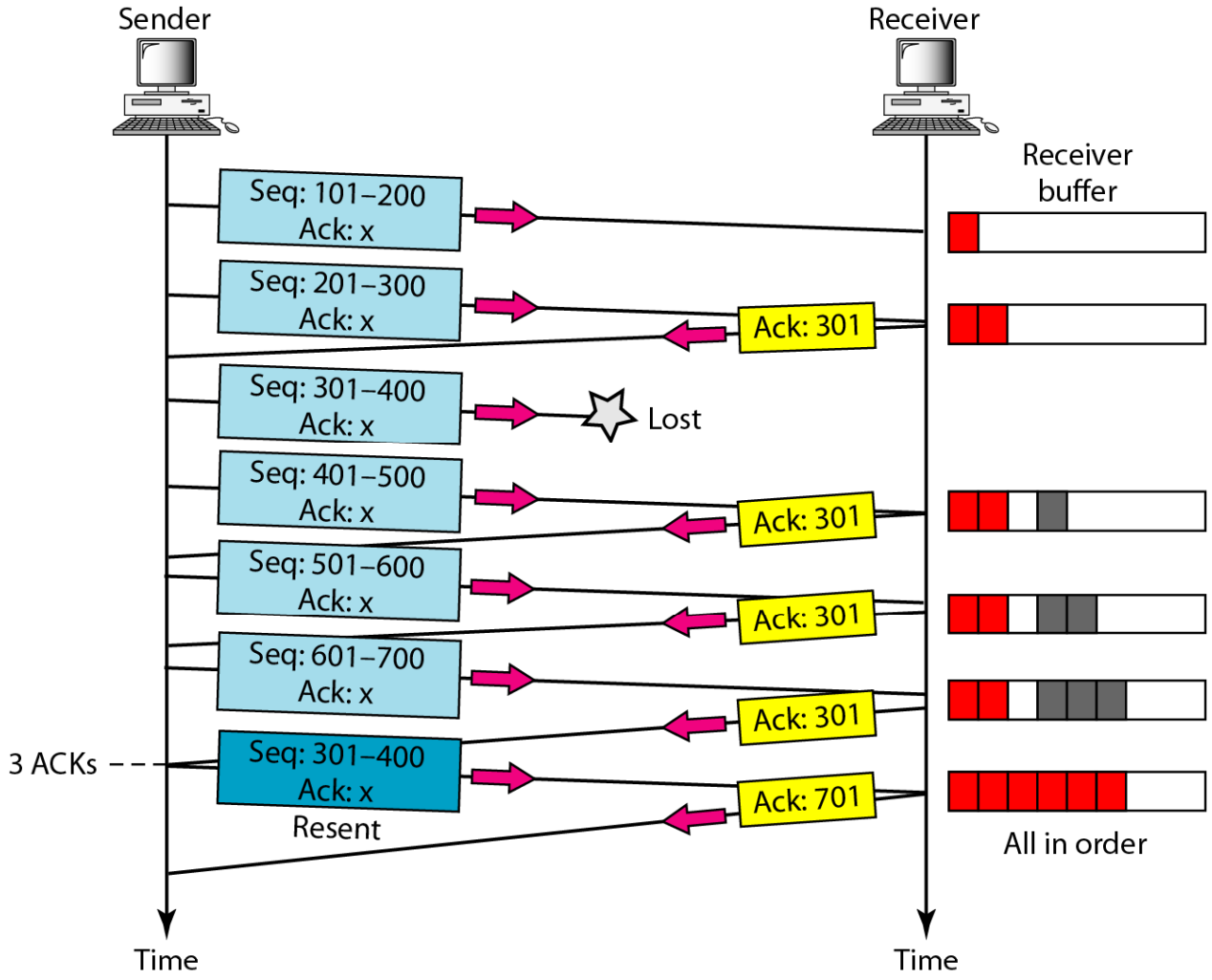
## Figure 23.25 *Lost segment*



*Note*

**The receiver TCP delivers only ordered data to the process.**

## Figure 23.26 *Fast retransmission*



# 23-4 SCTP

***Stream Control Transmission Protocol (SCTP) is a new reliable, message-oriented transport layer protocol. SCTP, however, is mostly designed for Internet applications that have recently been introduced. These new applications need a more sophisticated service than TCP can provide.***

***Topics discussed in this section:***

**SCTP Services and Features**
**Packet Format**
**An SCTP Association**
**Flow Control and Error Control**

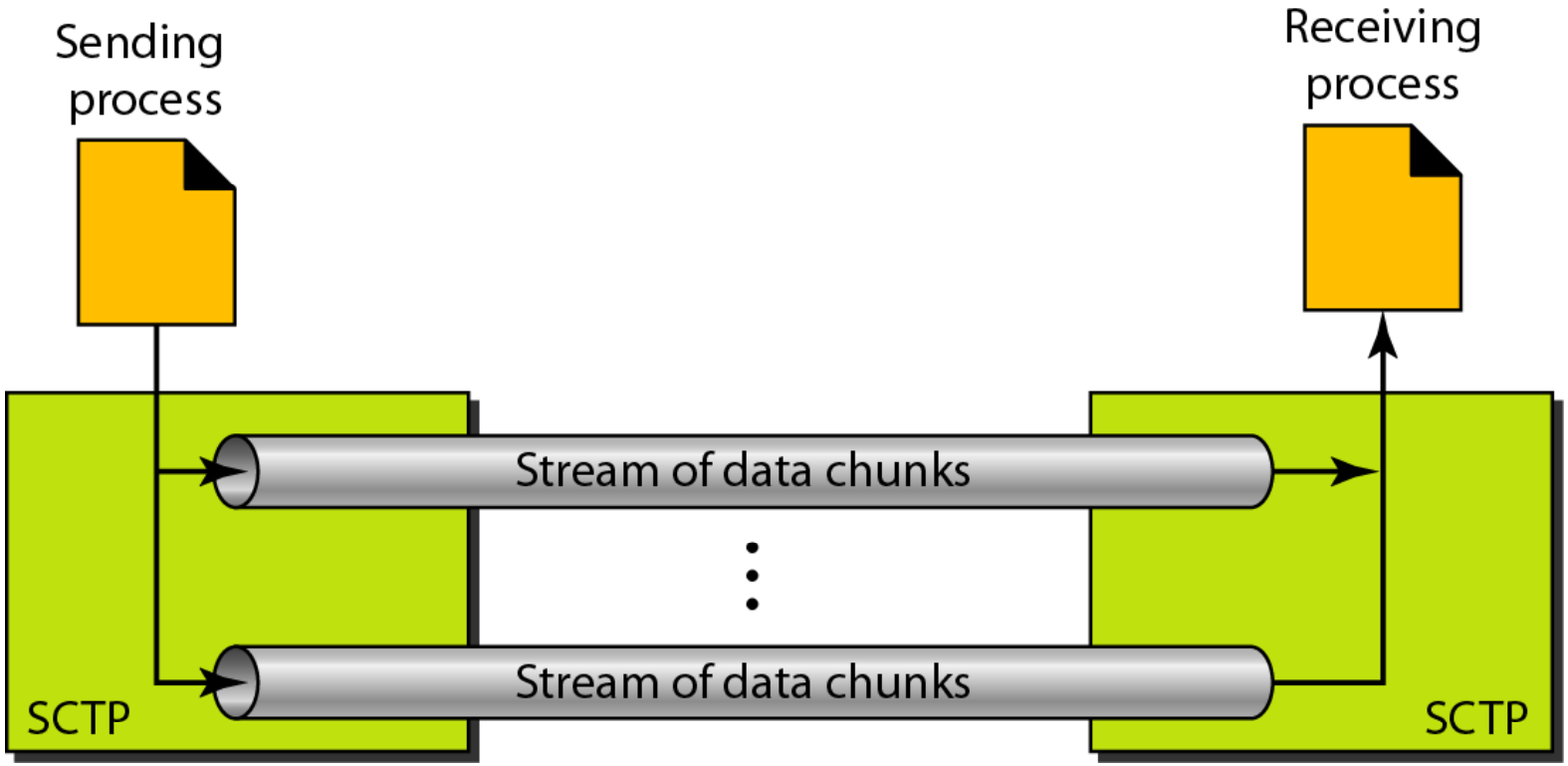
*Note*

**SCTP is a message-oriented, reliable protocol that combines the best features of UDP and TCP.**

## Table 23.4 *Some SCTP applications*

| *Protocol* | *Port Number* | *Description* |
|---|---|---|
| IUA | 9990 | ISDN over IP |
| M2UA | 2904 | SS7 telephony signaling |
| M3UA | 2905 | SS7 telephony signaling |
| H.248 | 2945 | Media gateway control |
| H.323 | 1718, 1719, 1720, 11720 | IP telephony |
| SIP | 5060 | IP telephony |

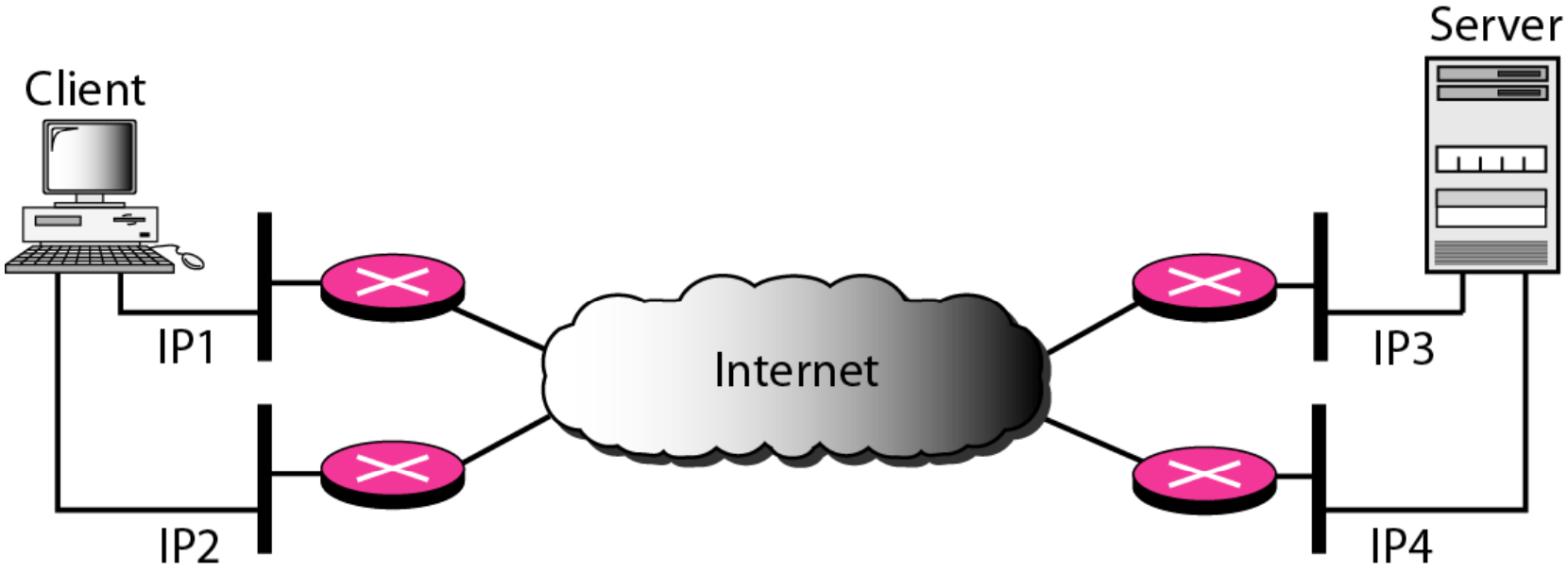
## Figure 23.27 *Multiple-stream concept*



*Note*

**An association in SCTP can involve multiple streams.**

## Figure 23.28 *Multihoming concept*



*Note*

**SCTP association allows multiple IP addresses for each end.**

*Note*

**In SCTP, a data chunk is numbered using a TSN.**

*Note*

**To distinguish between different streams, SCTP uses an SI.**

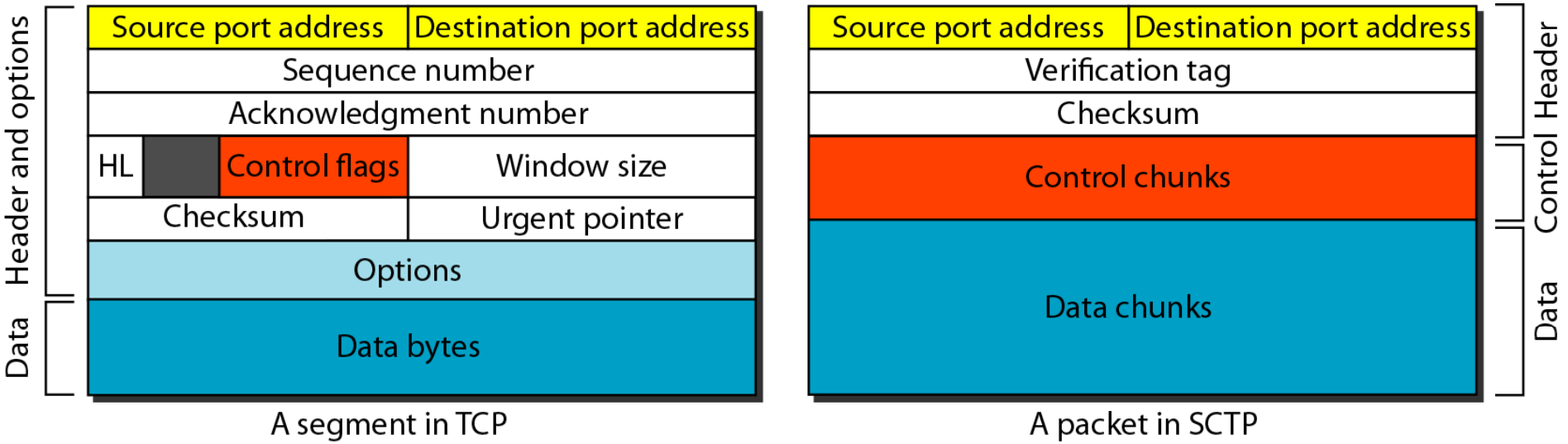
*Note*

**To distinguish between different data chunks belonging to the same stream, SCTP uses SSNs.**

*Note*

**TCP has segments; SCTP has packets.**

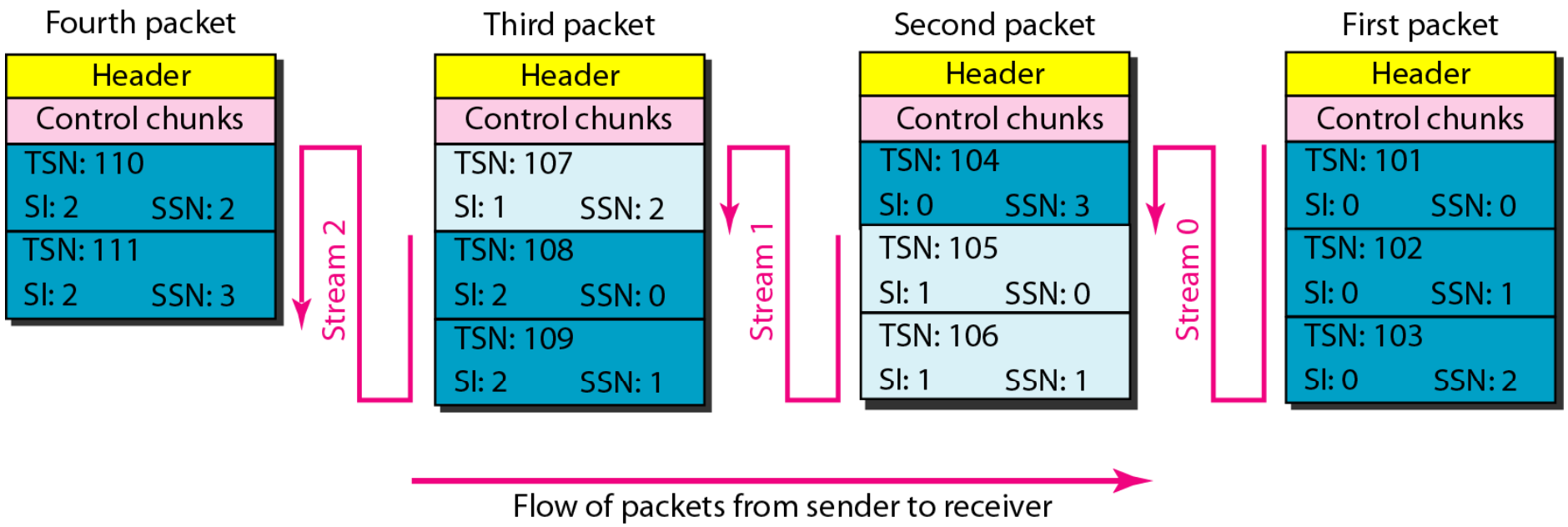
## Figure 23.29 *Comparison between a TCP segment and an SCTP packet*



*Note*

**In SCTP, control information and data information are carried in separate chunks.**

## Figure 23.30 *Packet, data chunks, and streams*



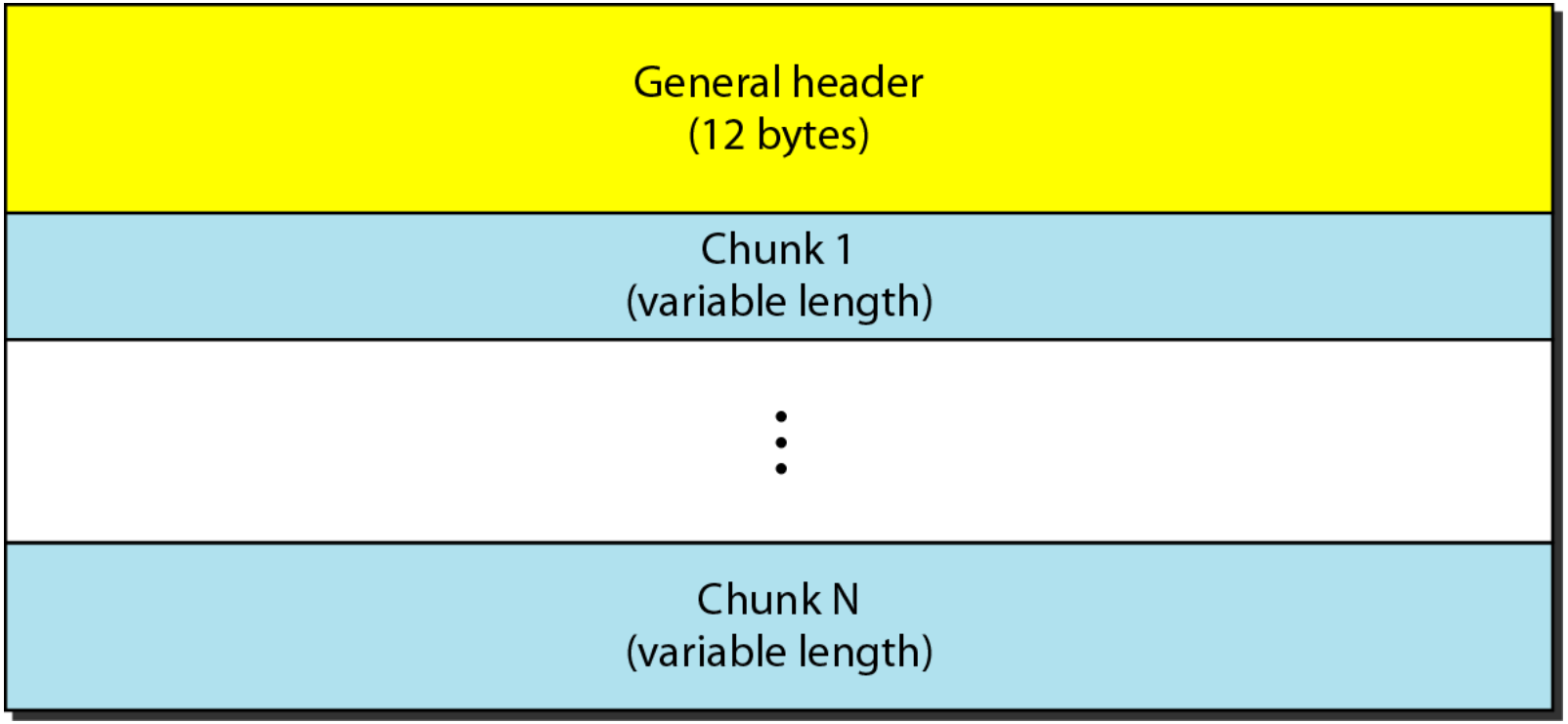
*Note*

**Data chunks are identified by three items: TSN, SI, and SSN. TSN is a cumulative number identifying the association; SI defines the stream; SSN defines the chunk in a stream.**

*Note*

**In SCTP, acknowledgment numbers are used to acknowledge only data chunks; control chunks are acknowledged by other control chunks if necessary.**

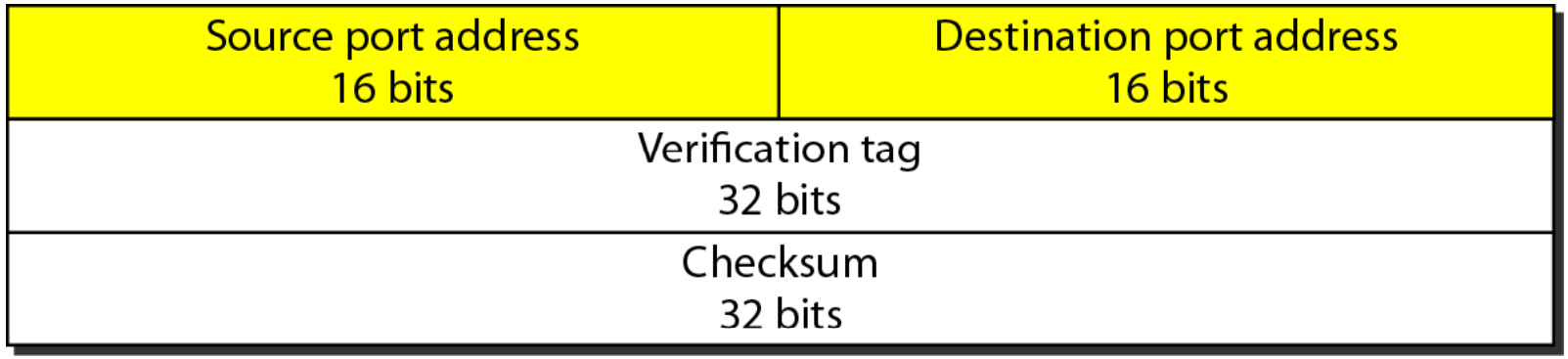
## Figure 23.31 *SCTP packet format*



*Note*

**In an SCTP packet, control chunks come before data chunks.**

## Figure 23.32 *General header*



## Table 23.5 *Chunks*

| *Type* | *Chunk* | *Description* |
|---|---|---|
| **0** | **DATA** | User data |
| **1** | **INIT** | Sets up an association |
| **2** | **INIT ACK** | Acknowledges INIT chunk |
| **3** | **SACK** | Selective acknowledgment |
| **4** | **HEARTBEAT** | Probes the peer for liveliness |
| **5** | **HEARTBEAT ACK** | Acknowledges HEARTBEAT chunk |
| **6** | **ABORT** | Aborts an association |
| **7** | **SHUTDOWN** | Terminates an association |
| **8** | **SHUTDOWN ACK** | Acknowledges SHUTDOWN chunk |
| **9** | **ERROR** | Reports errors without shutting down |
| **10** | **COOKIE ECHO** | Third packet in association establishment |
| **11** | **COOKIE ACK** | Acknowledges COOKIE ECHO chunk |
| **14** | **SHUTDOWN COMPLETE** | Third packet in association termination |
| **192** | **FORWARD TSN** | For adjusting cumulative TSN |

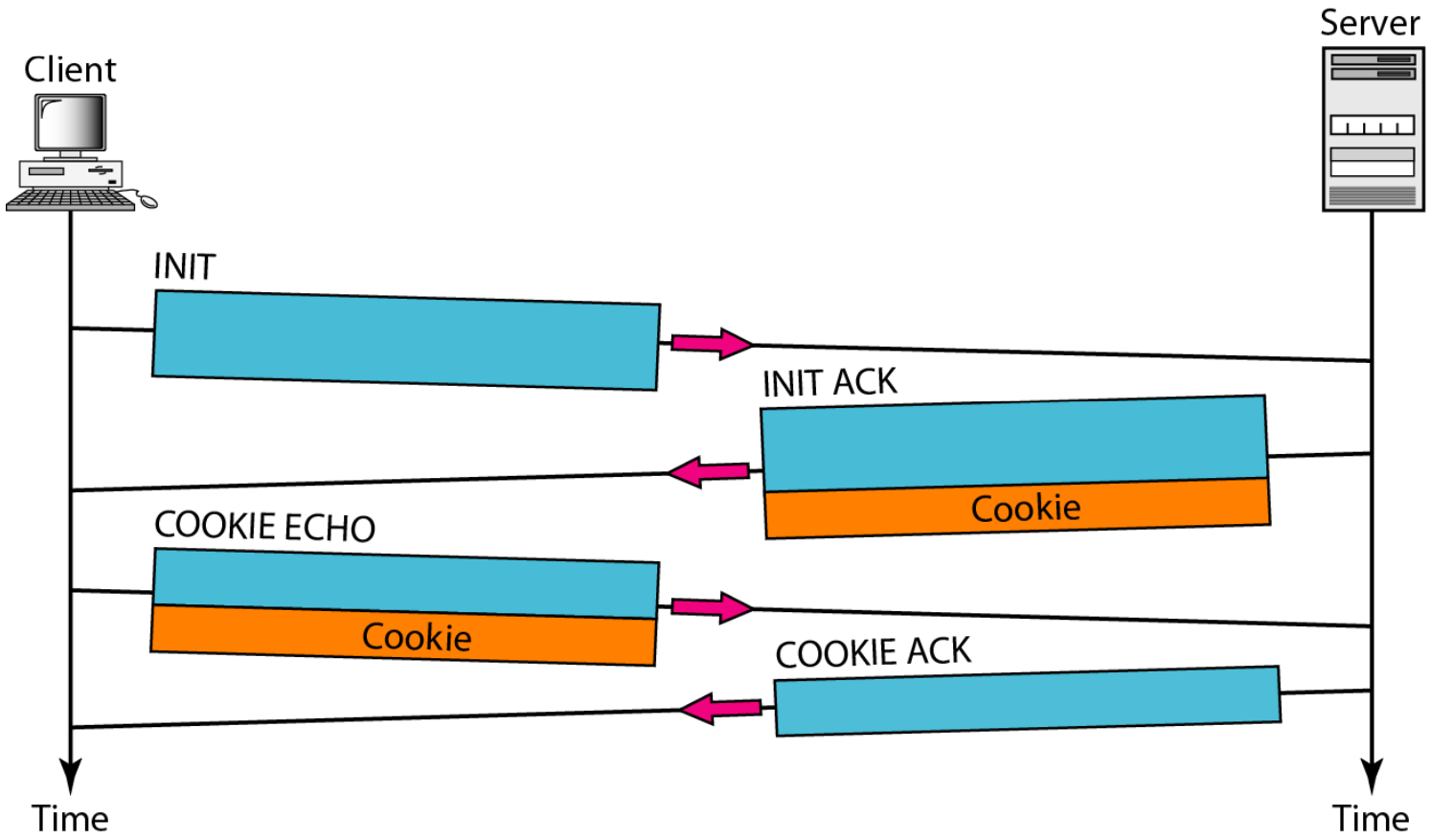
*Note*

**A connection in SCTP is called an association.**

*Note*

**No other chunk is allowed in a packet carrying an INIT or INIT ACK chunk. A COOKIE ECHO or a COOKIE ACK chunk can carry data chunks.**

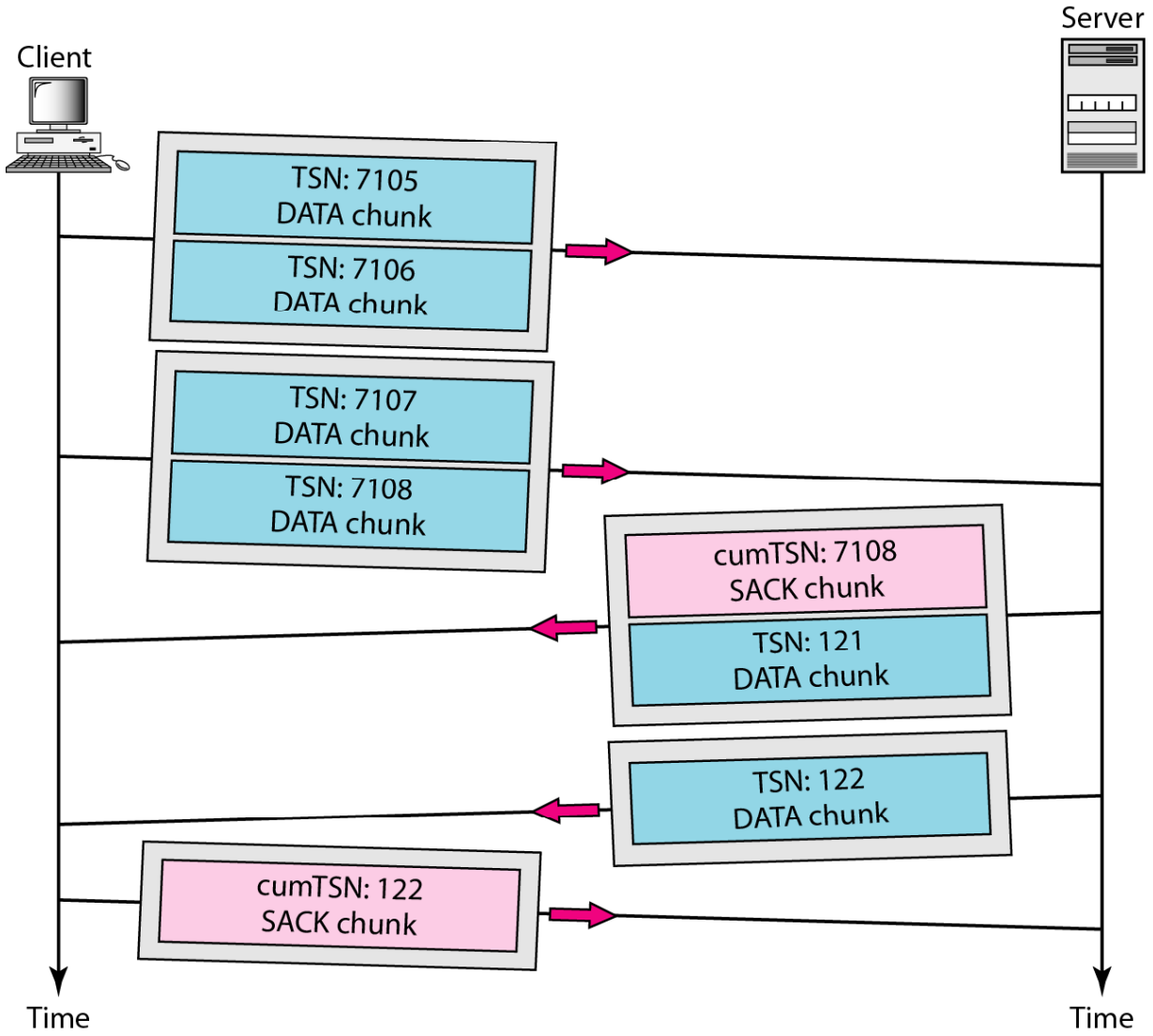
## Figure 23.33 *Four-way handshaking*



*Note*

**In SCTP, only DATA chunks consume TSNs; DATA chunks are the only chunks that are acknowledged.**

## Figure 23.34 *Simple data transfer*



*Note*

**The acknowledgment in SCTP defines the cumulative TSN, the TSN of the last data chunk received in order.**

## Figure 23.35 *Association termination*

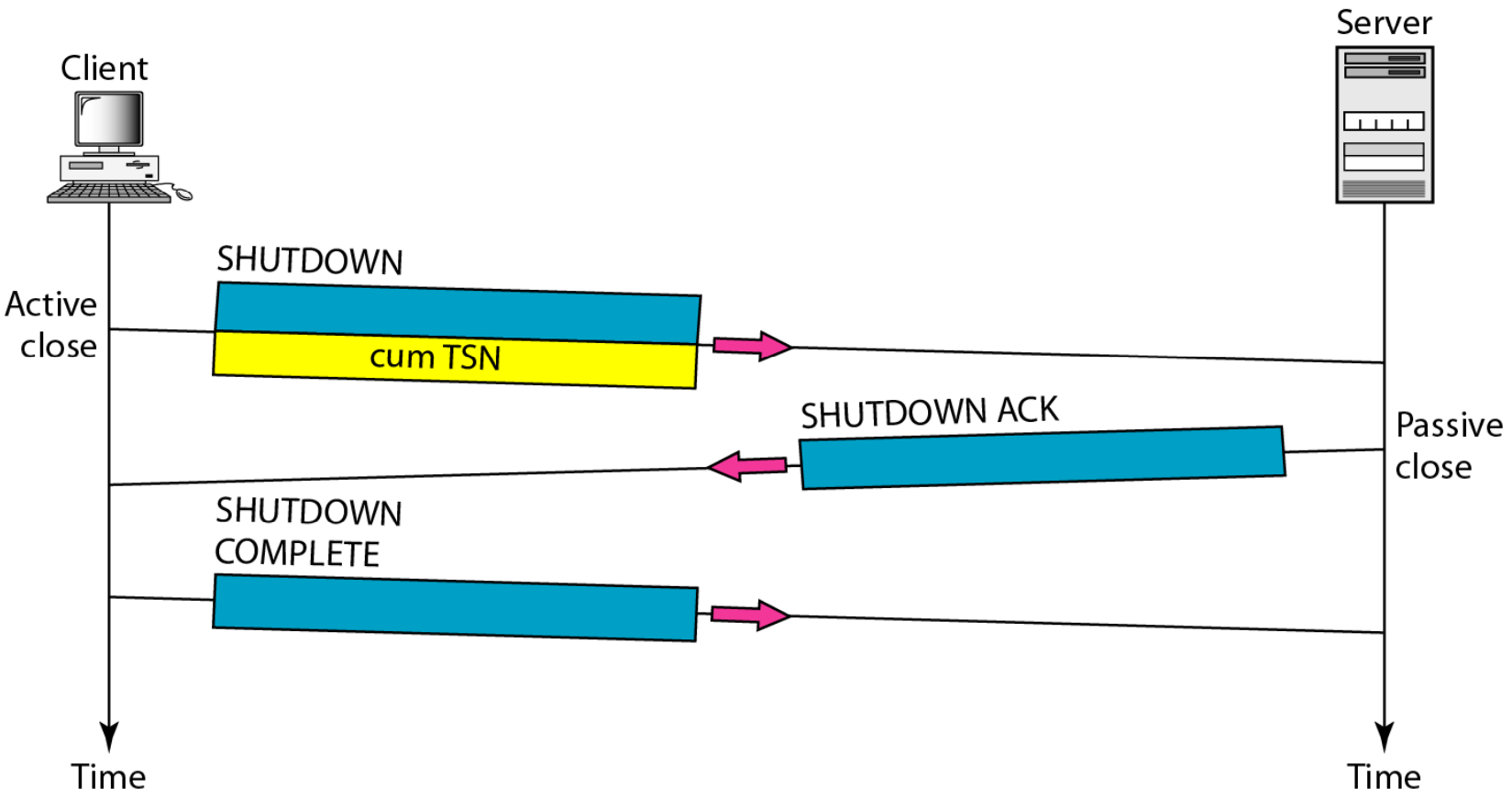
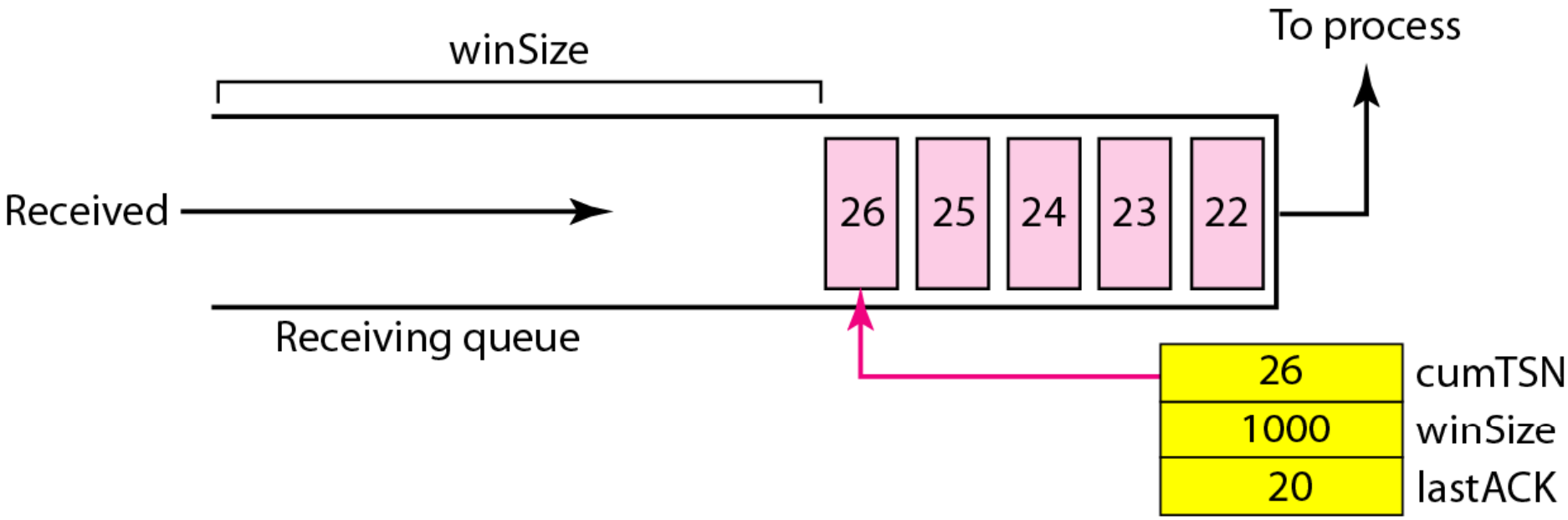
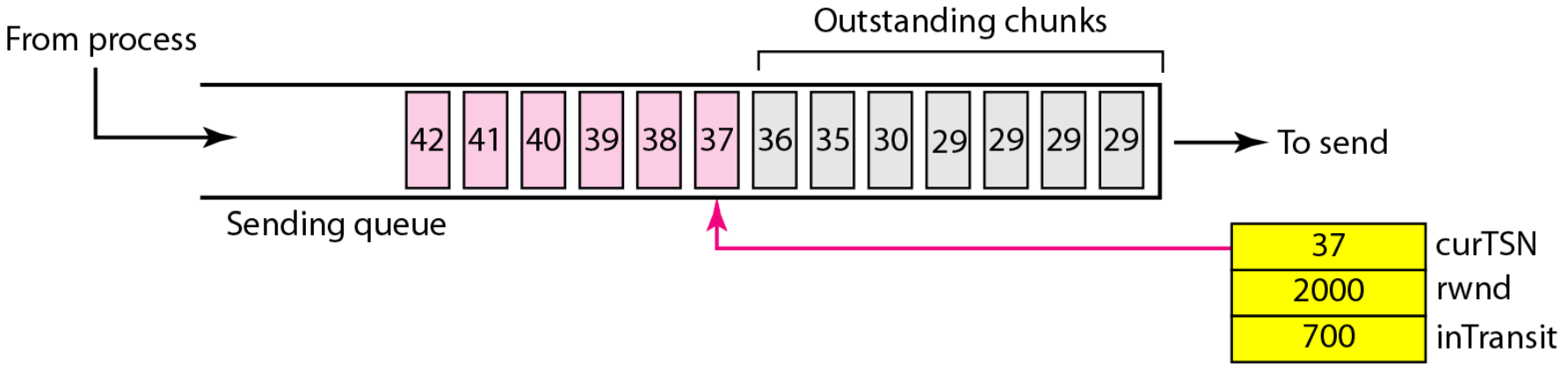


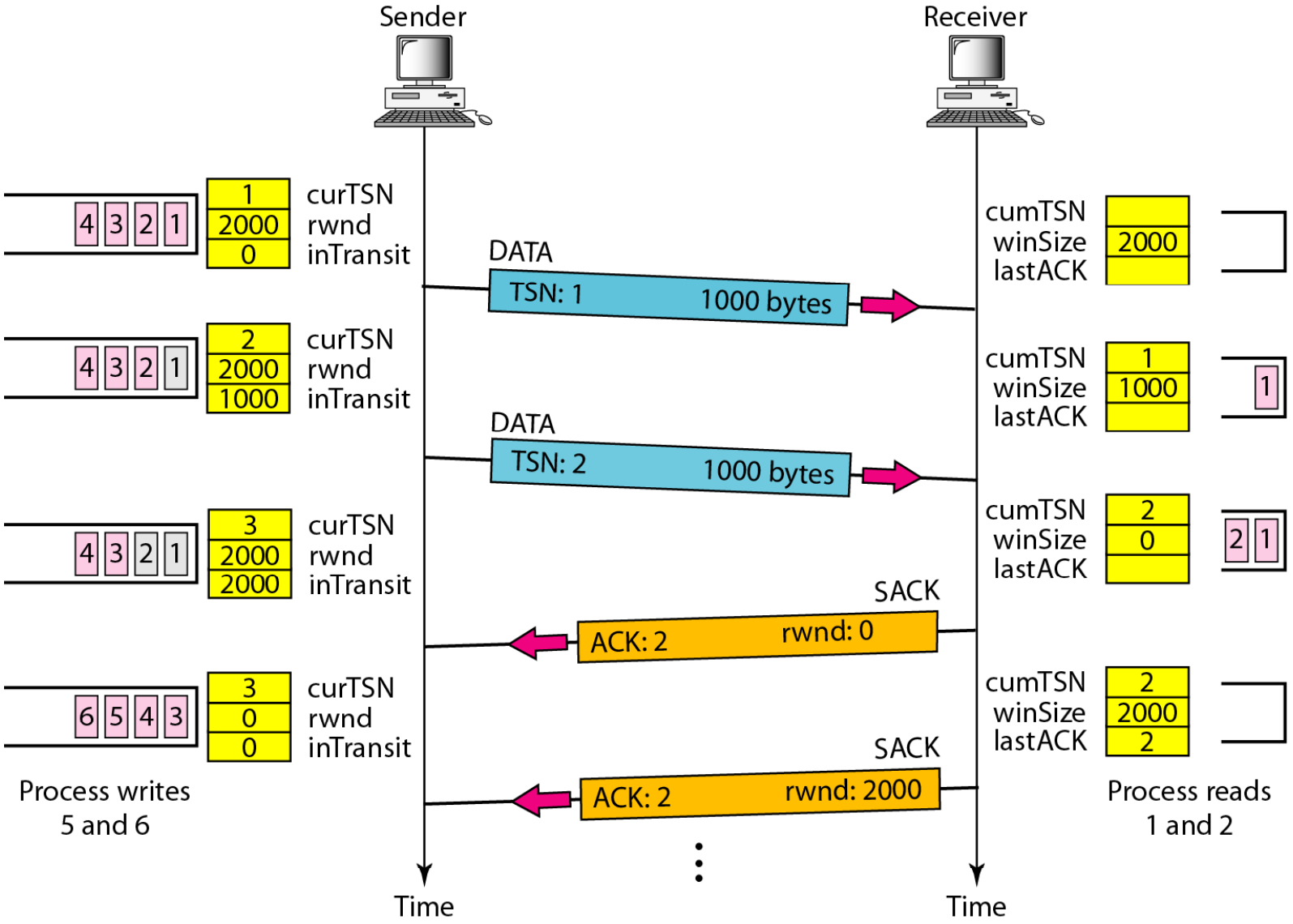
**Figure 23.36** *Flow control, receiver site*



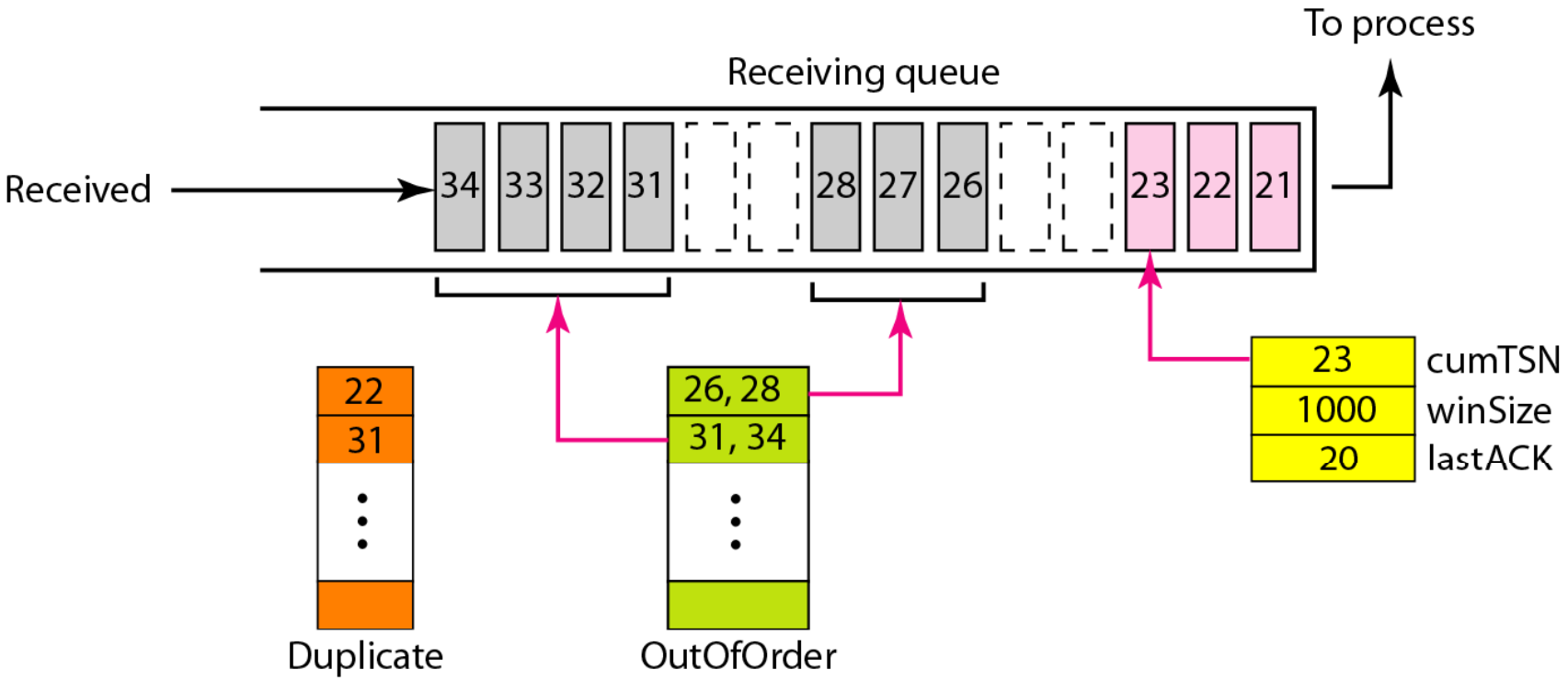
## Figure 23.37 *Flow control, sender site*



## Figure 23.38 *Flow control scenario*



## Figure 23.39 *Error control, receiver site*



## Figure 23.40 *Error control, sender site*