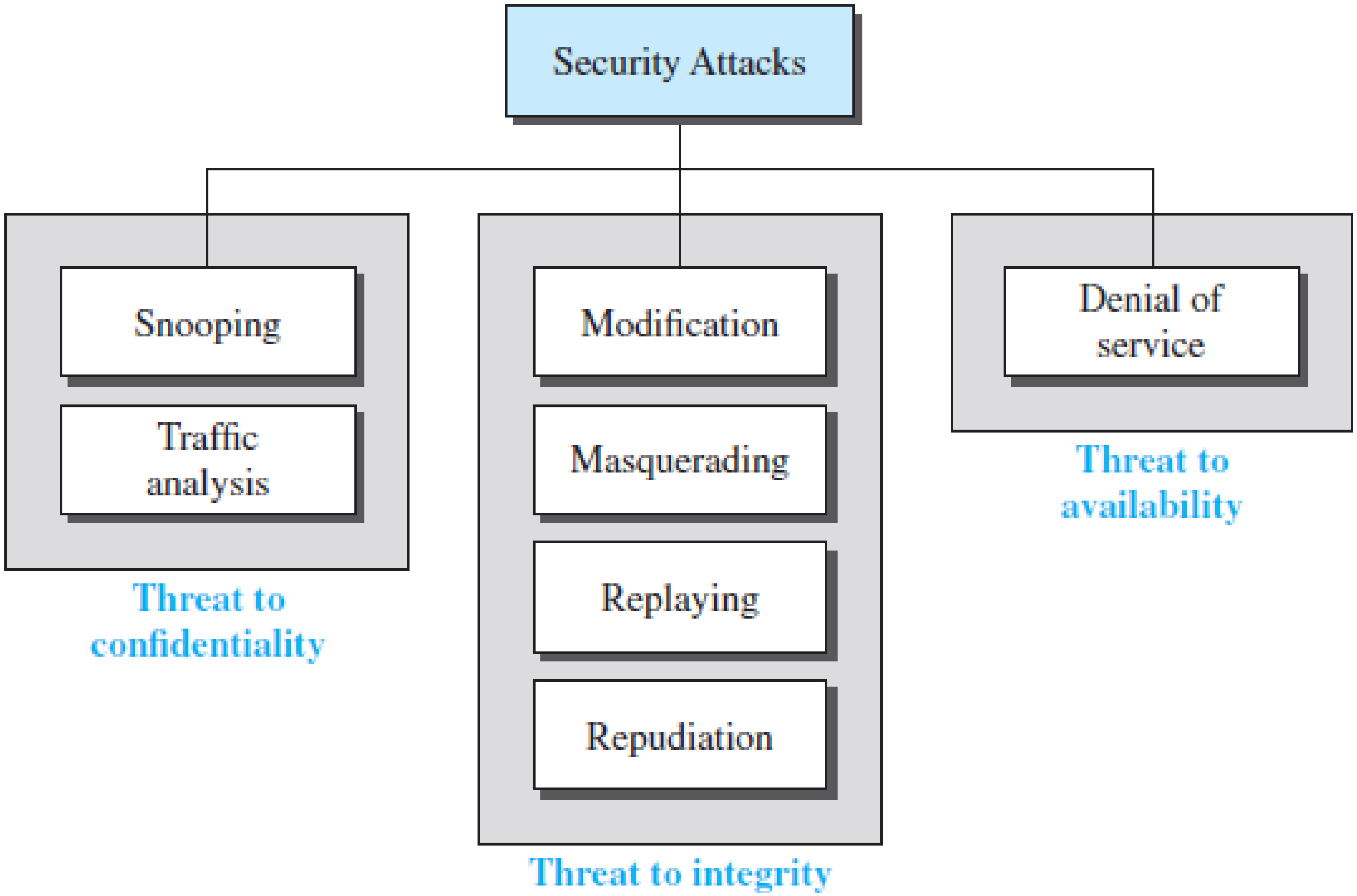
Cryptography and Network Security

SECURITY GOALS

- **Confidentiality**
  - Hidden from unauthorized access
  - Applied in storage and transmission
- **Integrity**
  - protect from unauthorized change
- **Availability**
  - Make available to authorized entity

SECURITY ATTACKS



Passive vs Active Attacks

Passive vs Active Attacks

SECURITY SERVICES AND TECHNIQUES

- **To achieve security goals and prevent attacks**
- **Two prevalent techniques**
  - Cryptography
  - Steganography
    - practice of concealing a file, message, image, or video within another file, message, image, or video

INTRODUCTION

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

<u>Topics discussed in this section:</u>

Definitions
Two Categories

Figure: *Cryptography components*

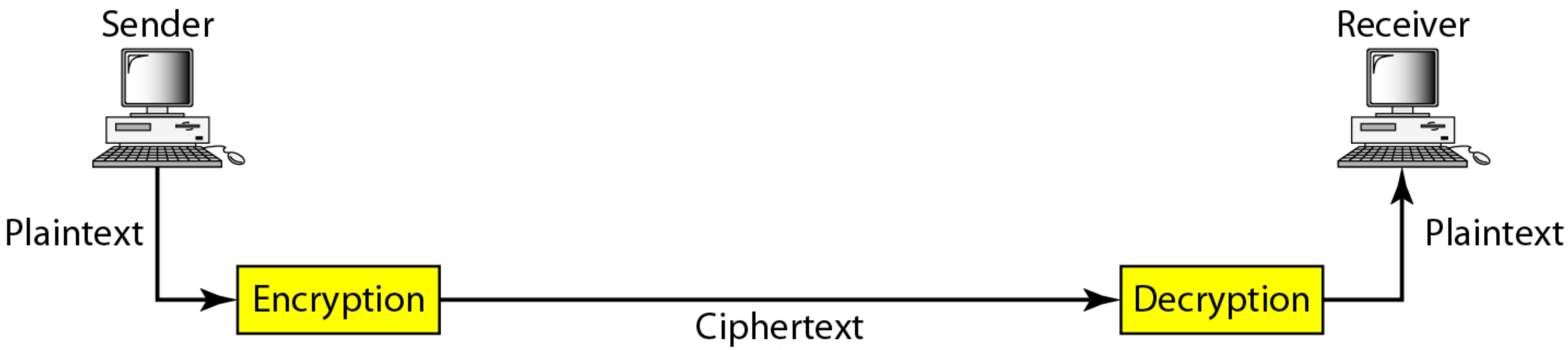


Figure: ***Categories of cryptography***

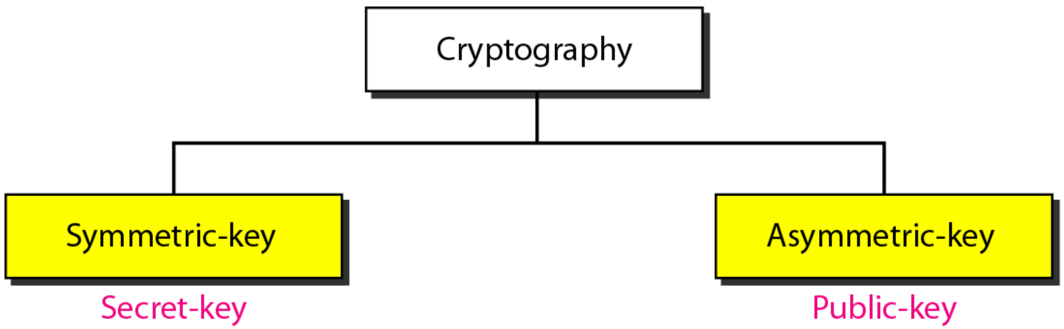
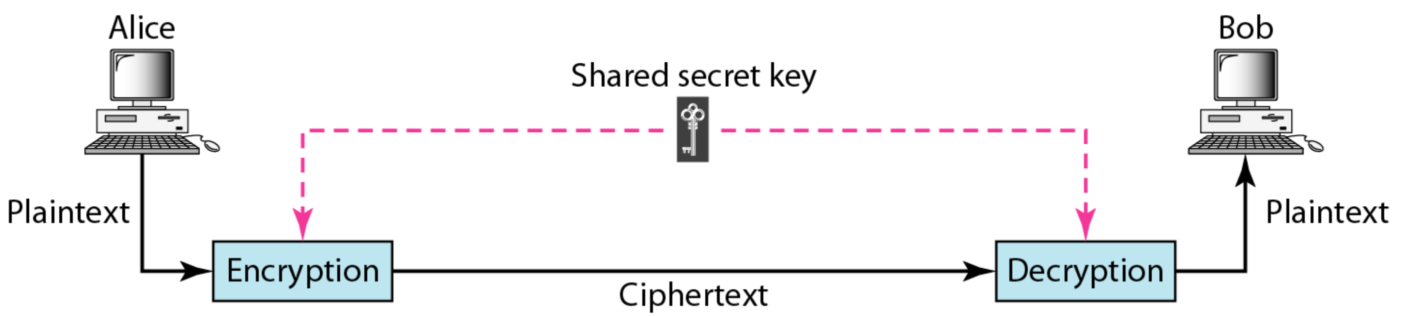


Figure: ***Symmetric-key cryptography***



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: *Asymmetric-key cryptography*

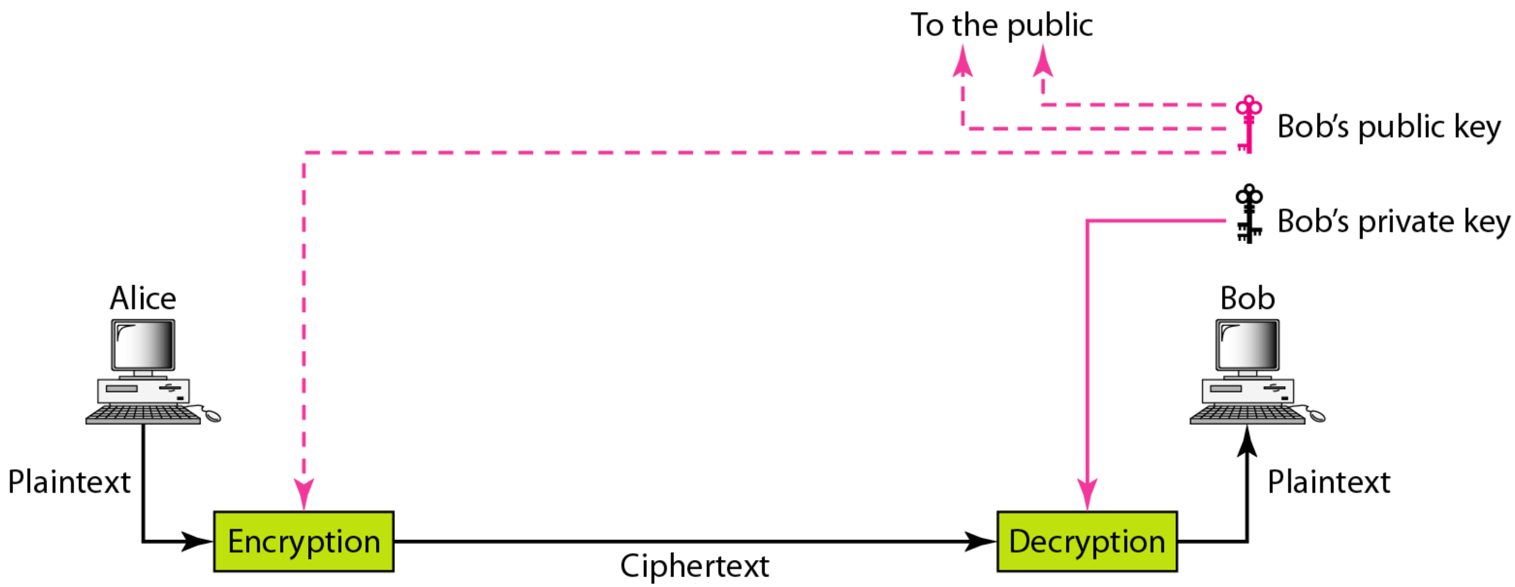


Figure: *Keys used in cryptography*

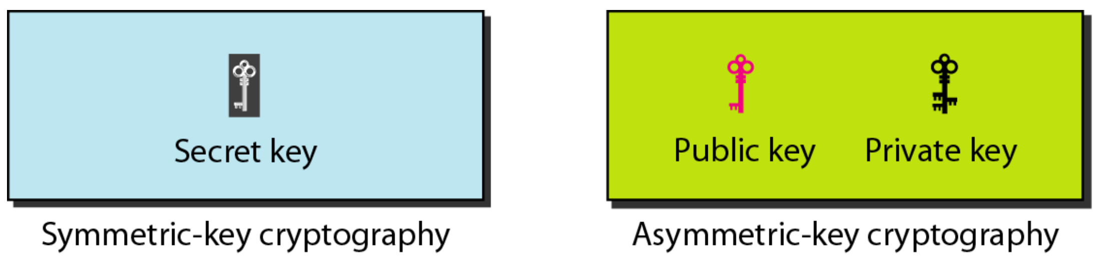
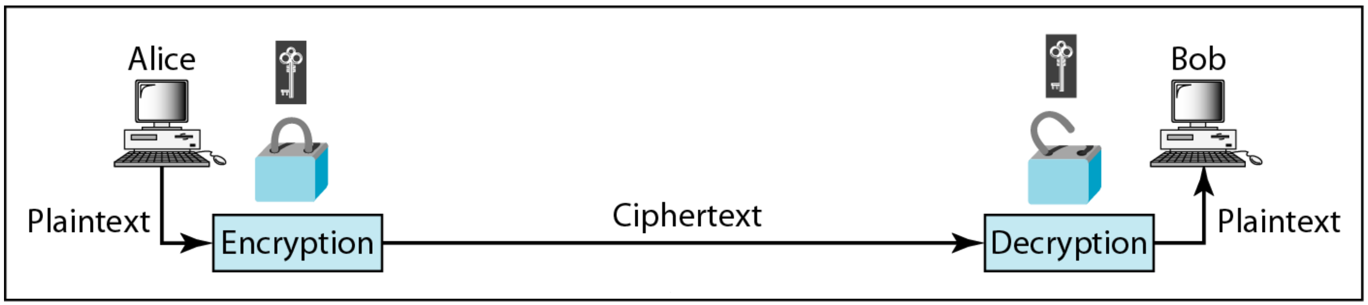
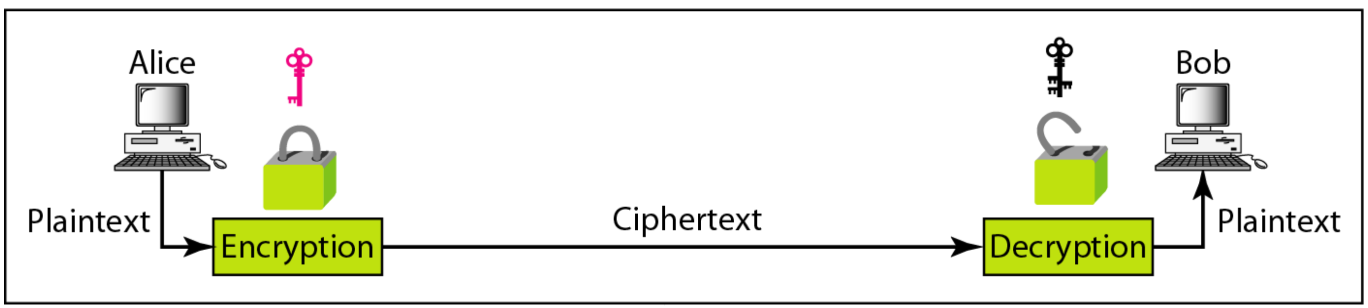


Figure: *Comparison between two categories of cryptography*



a. Symmetric-key cryptography



b. Asymmetric-key cryptography

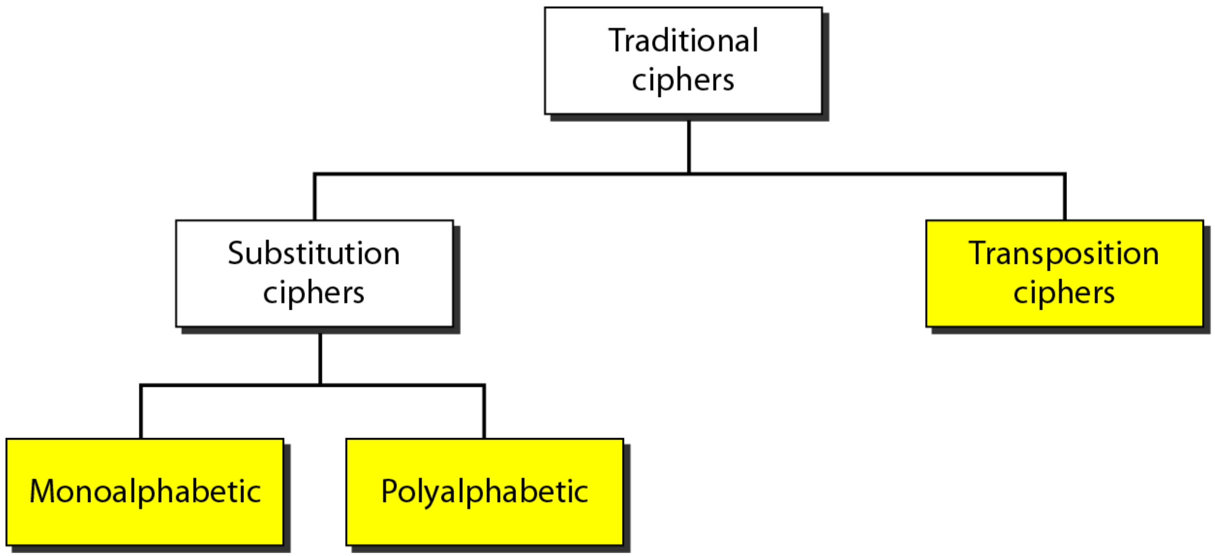
SYMMETRIC-KEY CRYPTOGRAPHY

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: *Traditional ciphers*



Note

A substitution cipher replaces one symbol with another.

Example

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

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.

- The simplest monoalphabetic cipher is the Additive cipher/Shift cipher/Caesar cipher
- **Plaintext**
  - The original message before transformed
- **Ciphertext**
  - After the message is transformed
- **Cipher**
  - encryption and decryption algorithms

Representation of plaintext and ciphertext characters in modulo 26

Plaintext →	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext →	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value →	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

and key are integers in modulo 26

Additive Cipher-Encryption

Use the additive cipher with key = 15 to encrypt the message "hello".

Solution

We apply the encryption algorithm to the plaintext, character by character:

Plaintext: h → 07	Encryption: (07 + 15) mod 26	Ciphertext: 22 → W
Plaintext: e → 04	Encryption: (04 + 15) mod 26	Ciphertext: 19 → T
Plaintext: l → 11	Encryption: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: l → 11	Encryption: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: o → 14	Encryption: (14 + 15) mod 26	Ciphertext: 03 → D

Additive Cipher-Decryption

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

Solution

We apply the decryption algorithm to the plaintext character by character:

Ciphertext: W $\rightarrow$ 22	Decryption: (22 – 15) mod 26	Plaintext: 07 $\rightarrow$ h
Ciphertext: T $\rightarrow$ 19	Decryption: (19 – 15) mod 26	Plaintext: 04 $\rightarrow$ e
Ciphertext: A $\rightarrow$ 00	Decryption: (00 – 15) mod 26	Plaintext: 11 $\rightarrow$ l
Ciphertext: A $\rightarrow$ 00	Decryption: (00 – 15) mod 26	Plaintext: 11 $\rightarrow$ l
Ciphertext: D $\rightarrow$ 03	Decryption: (03 – 15) mod 26	Plaintext: 14 $\rightarrow$ o

- A better solution is to create a mapping between each plaintext character and the corresponding ciphertext character

An example key for a monoalphabetic substitution cipher

- A better solution is to create a mapping between each plaintext character and the corresponding ciphertext character

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

Plaintext:	this message is easy to encrypt but hard to find the key
Ciphertext:	ICFVQRVVNERFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

Polyalphabetic Ciphers

- Autokey cipher with initial key value k1 = 12

$P = P_1P_2P_3 \ldots$ $C = C_1C_2C_3\ldots$ $k = (k_1, P_1, P_2, \ldots)$

Encryption: $C_i = (P_i + k_i) \bmod 26$ Decryption: $P_i = (C_i - k_i) \bmod 26$

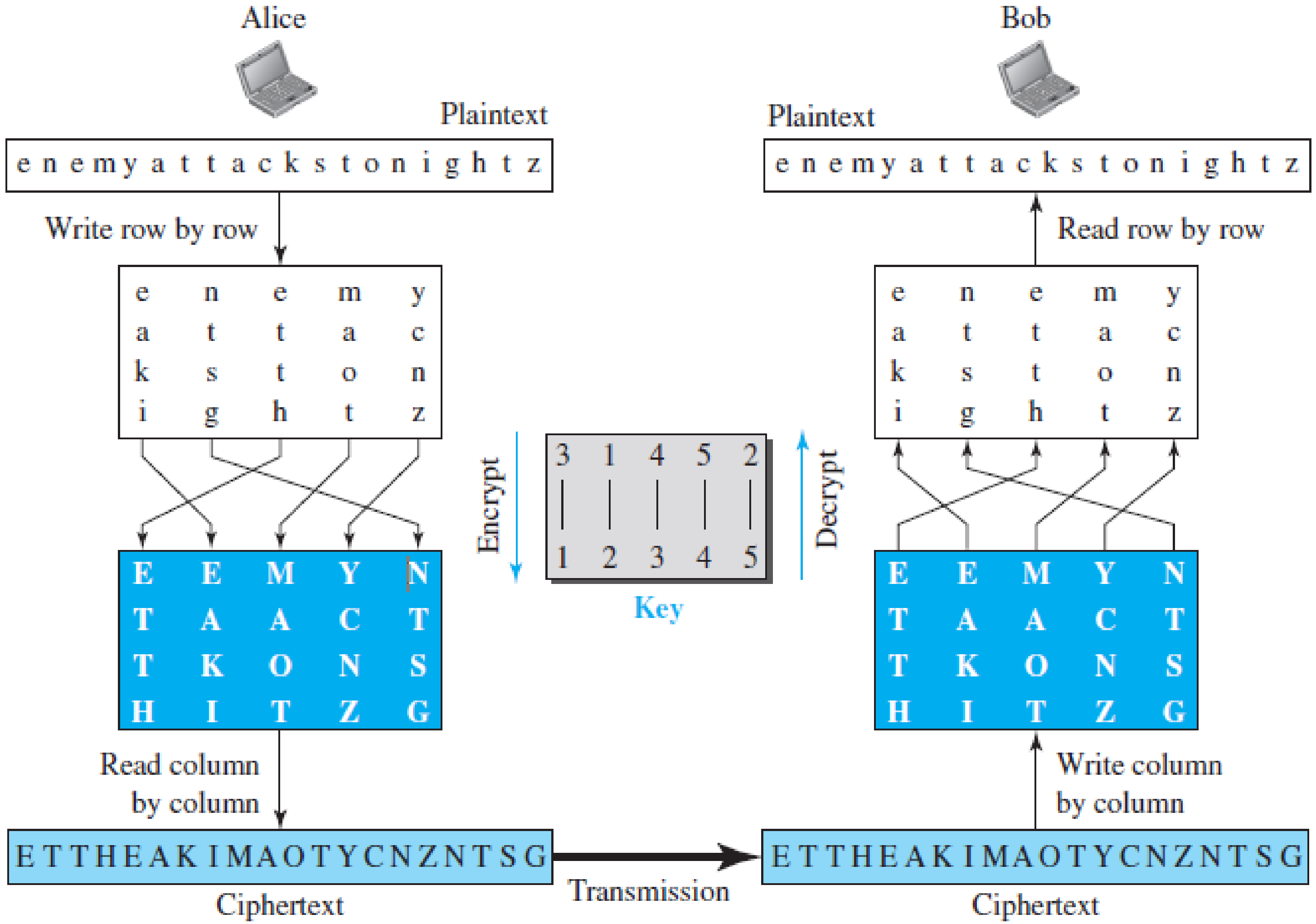
Plaintext:	a	t	t	a	c	k	i	s	t	o	d	a	y
P's Values:	00	19	19	00	02	10	08	18	19	14	03	00	24
Key stream:	*12*	*00*	*19*	*19*	*00*	*02*	*10*	*08*	*18*	*19*	*14*	*03*	*00*
C's Values:	12	19	12	19	02	12	18	00	11	7	17	03	24
Ciphertext:	M	T	M	T	C	M	S	A	L	H	R	D	Y

Note

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

Example-Transposition cipher

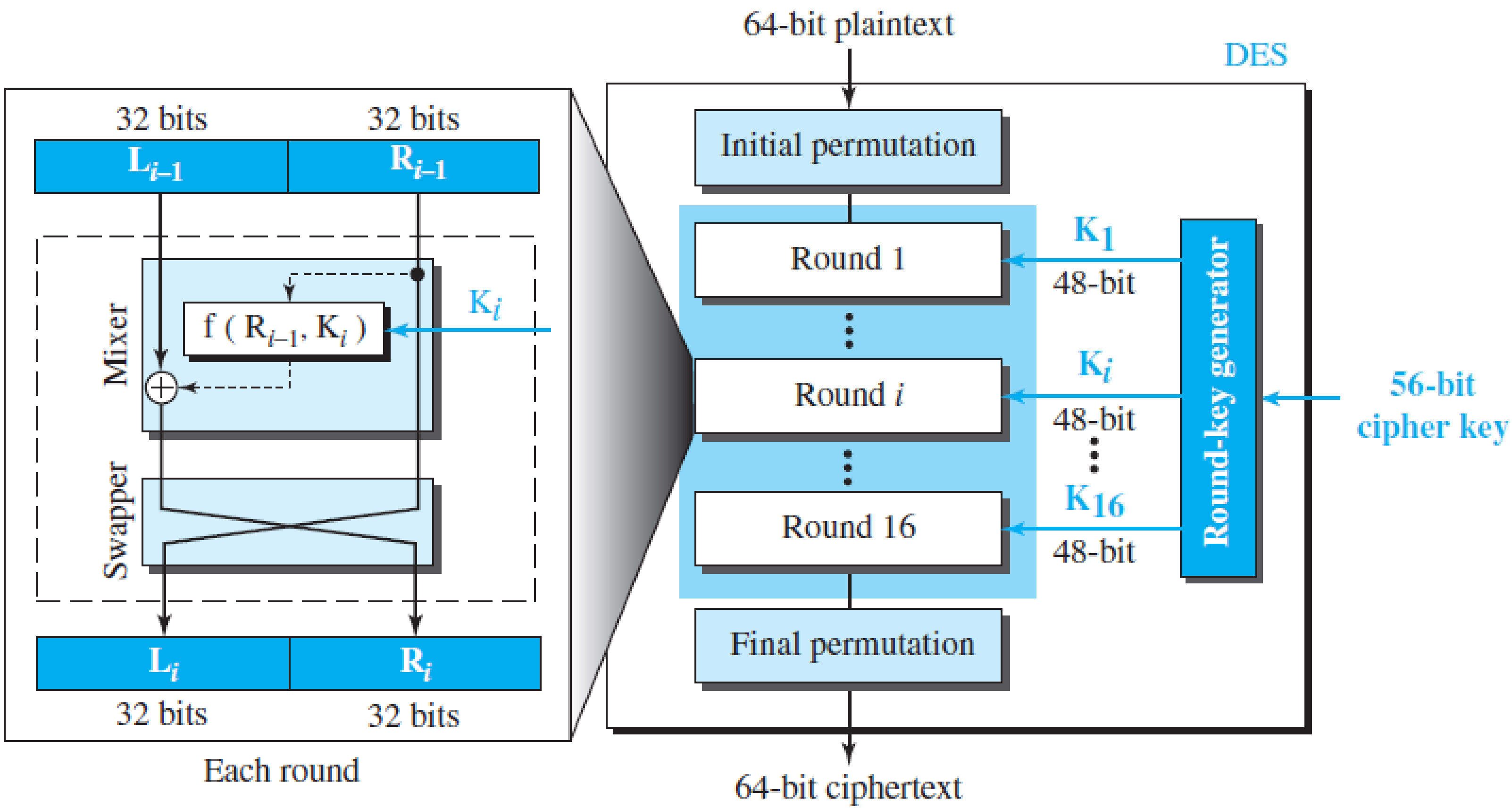
- z is added at the end to make the number of characters multiple of 5



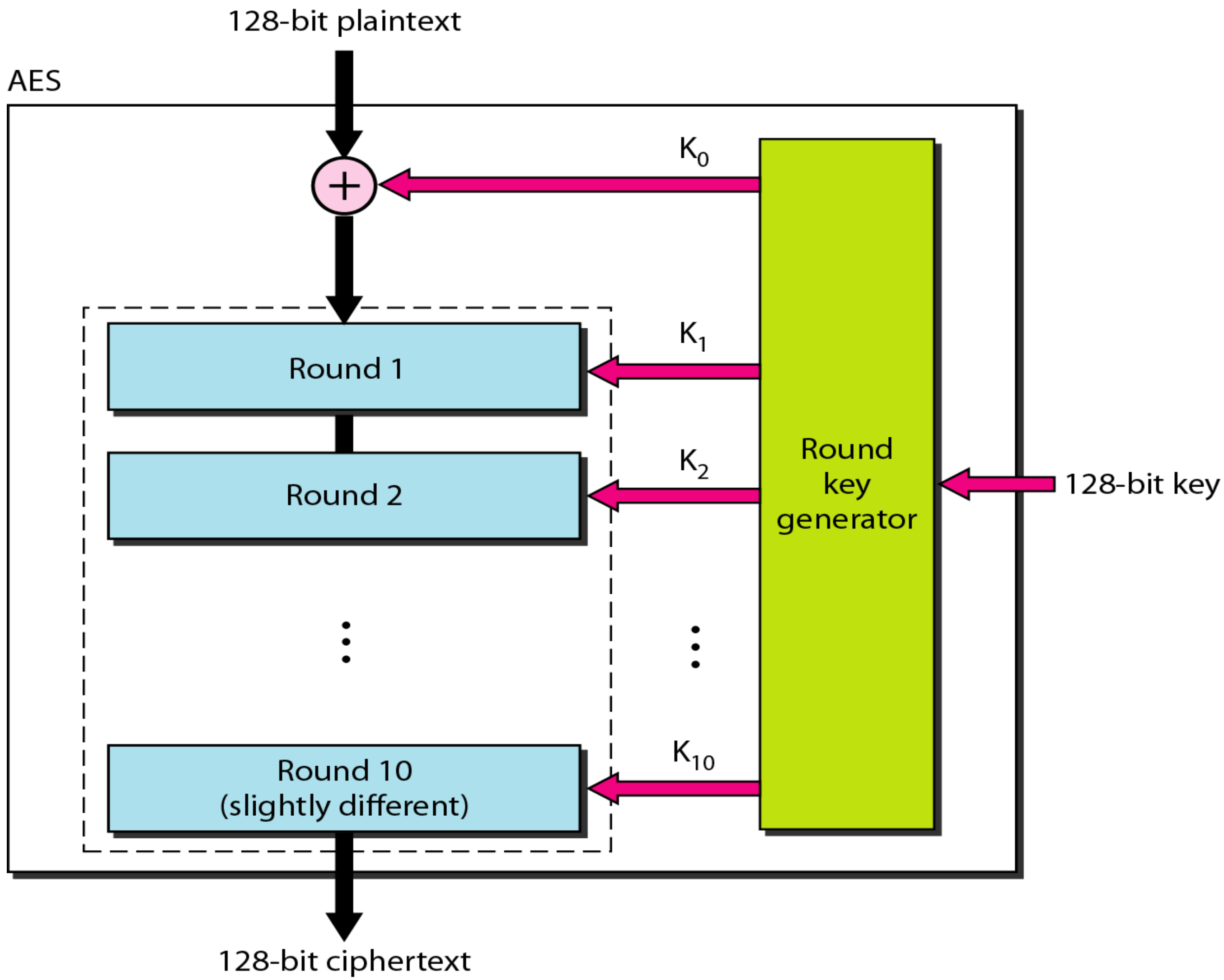
Modern Block Ciphers

- Lucifer / DES (Data Encryption Standard)
- IDEA (International Data Encryption Algorithm)
- RC5
- Rijndael / AES (Advanced Encryption Standard)
- Blowfish

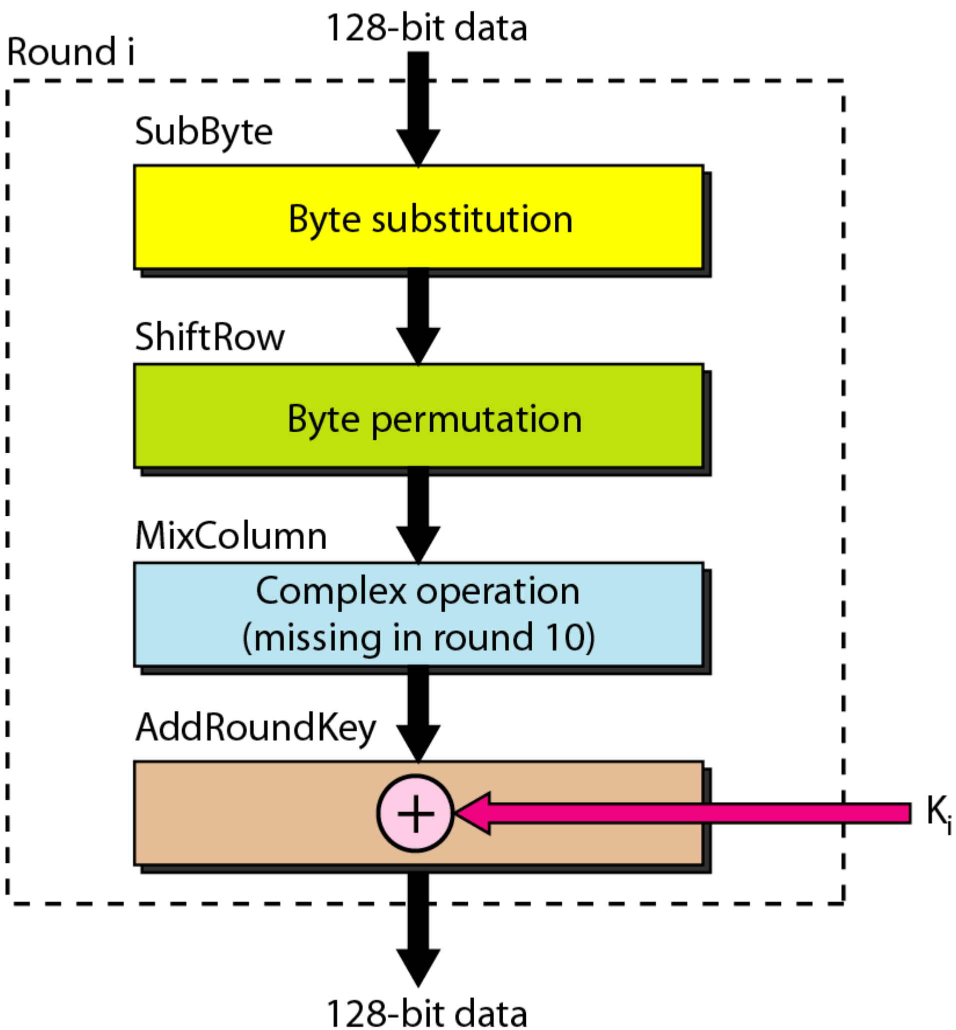
General Structure of DES



AES Structure



AES Structure of each round



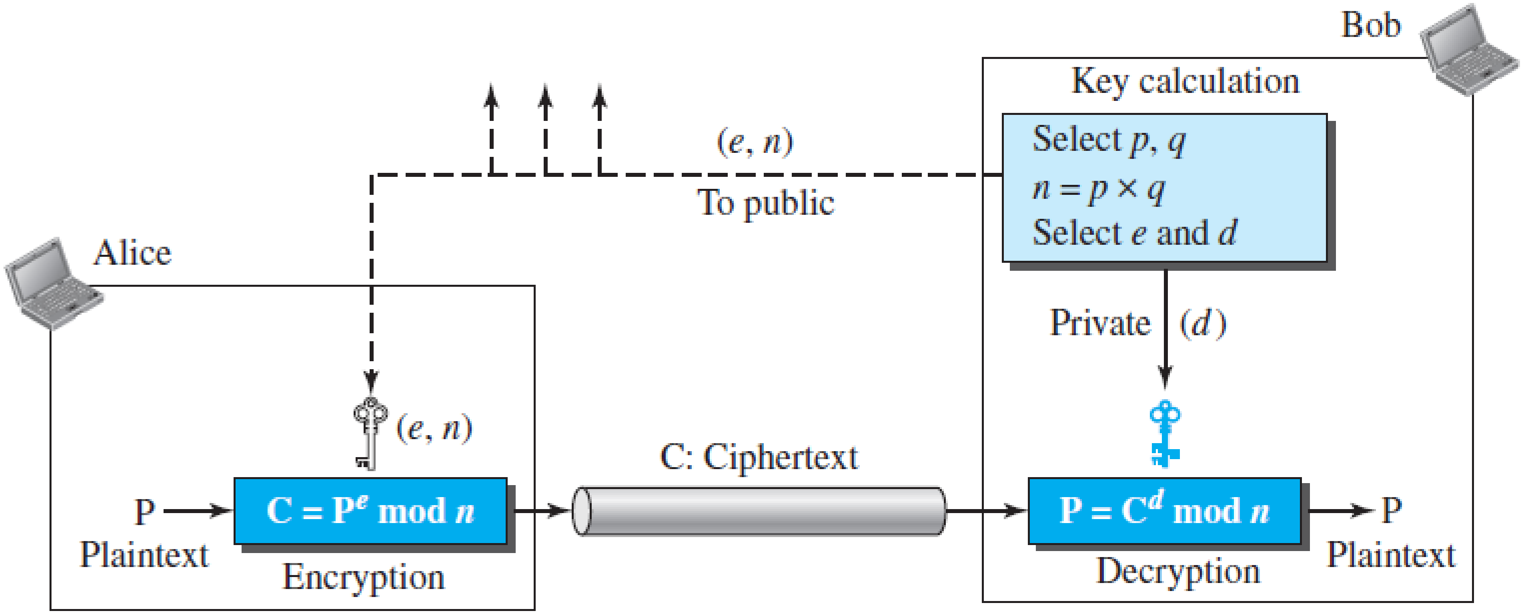
ASYMMETRIC-KEY CRYPTOGRAPHY

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

Asymmetric cryptography Algorithms

RSA (Rivest–Shamir–Adleman)
Diffie-Hellman
ECC (Elliptic-curve cryptography)
ElGamal
DSA (Digital Signature Algorithm)

Figure: *RSA*



Bob
Key calculation
Select p, q
$n = p \times q$
Select e and d
(e, n)
To public
Alice
Private
(d)
(e, n)
C: Ciphertext
P
Plaintext
$C = P^e \bmod n$
Encryption
$P = C^d \bmod n$
Decryption
P
Plaintext

Bob chooses two large numbers, p and q, and calculates $n = p \times q$ and $\phi = (p - 1) \times (q - 1)$. Bob then selects e and d such that $(e \times d) \bmod \phi = 1$. Bob advertises e and n to the community as the public key; Bob keeps d as the private key. Anyone, including Alice, can encrypt a message and send the ciphertext to Bob, using $C = (P^e) \bmod n$; only Bob can decrypt the message, using $P = (C^d) \bmod n$. An intruder such as Eve cannot decrypt the message if p and q are very large numbers (she does not know d).

For the sake of demonstration, let Bob choose 7 and 11 as p and q and calculate $n = 7 \times 11 = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$, or 60. If he chooses e to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$. Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5. This system is not safe because p and q are small.

Plaintext: 5

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

Ciphertext: 26

Ciphertext: 26

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

Plaintext: 5

Note

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

Example

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
3596600045617

Example (continued)

We calculate n. It has 309 digits:

n = 115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656772727460097082714127730434960500556347274566628060099924037102991424472292215772798531727033839381334692684137327622000966676671831831088373420823444370953

We calculate Φ. It has 309 digits:

ϕ = 115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656751054233608492916752034482627988117554787657013923444405716989581728196098226361075467211864612171359107358640614008885170265377277264467341066243857664128

Example (continued)

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

e = 35535

d = 58008302860037763936093661289677917594669062089650962180422866111380593852
822358731706286910030021710859044338402170729869087600611530620252495988448
4804756824096624708148581713046324064407770483313401085094738529564507193677406119732655742423721761767462077637164207600337085333288532144708859551
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 = 1907081826081826002619041819

Example (continued)

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

C = 4753091236462268272063655506105451809423717960704917165232392430544529
6061319932856661784341835911415119741125200568297979457173603610127821
8847892741566090480023507190715277185914975188465888632101148354103361
6578984679683867637337657774656250792805211481418440481418443081277305
9004692874248559166462108656

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

P = 1907081826081826002619041819

The recovered plaintext is THIS IS A TEST after decoding.

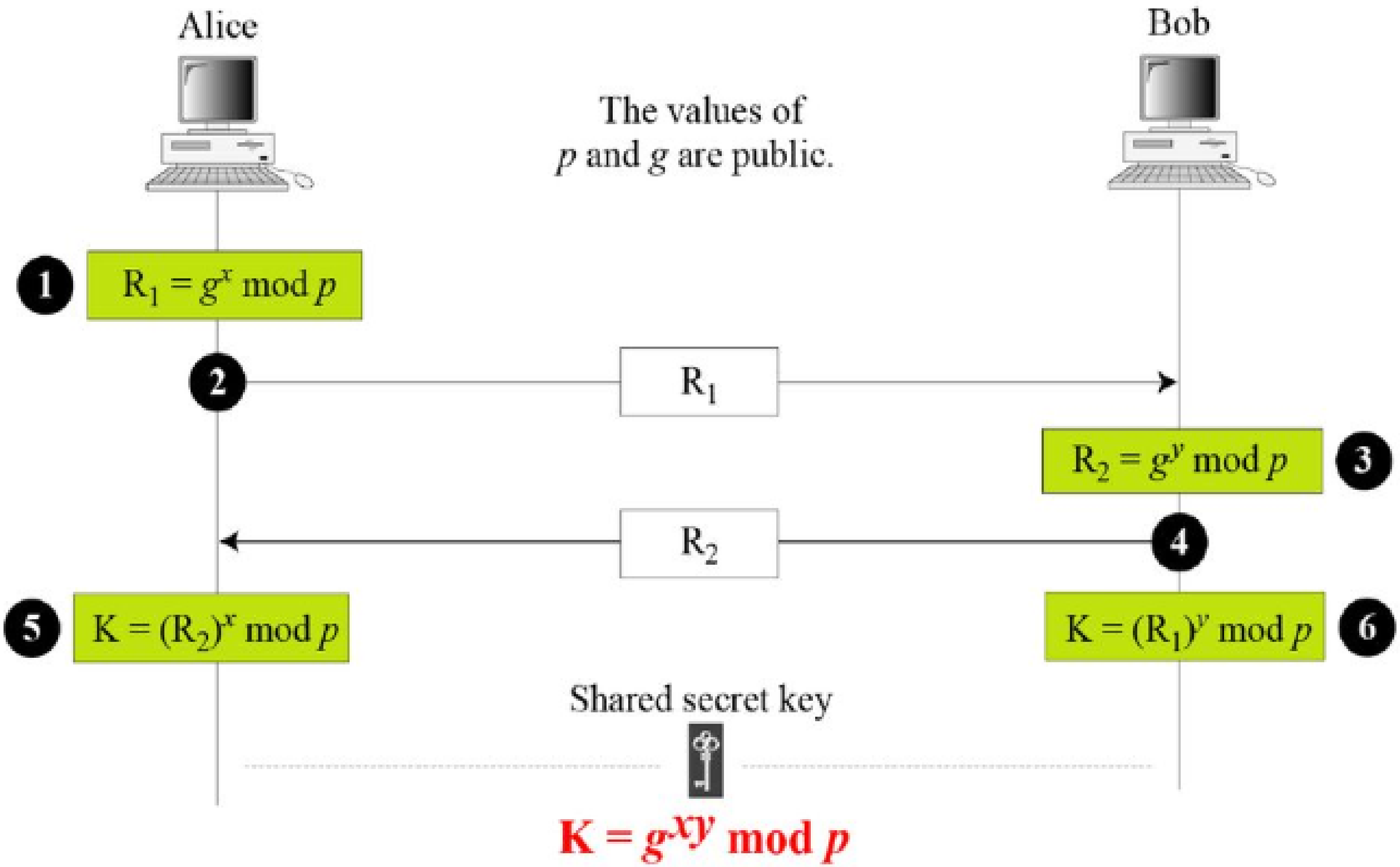
Note

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

p = large number

g = base

Diffie-Hellman Method



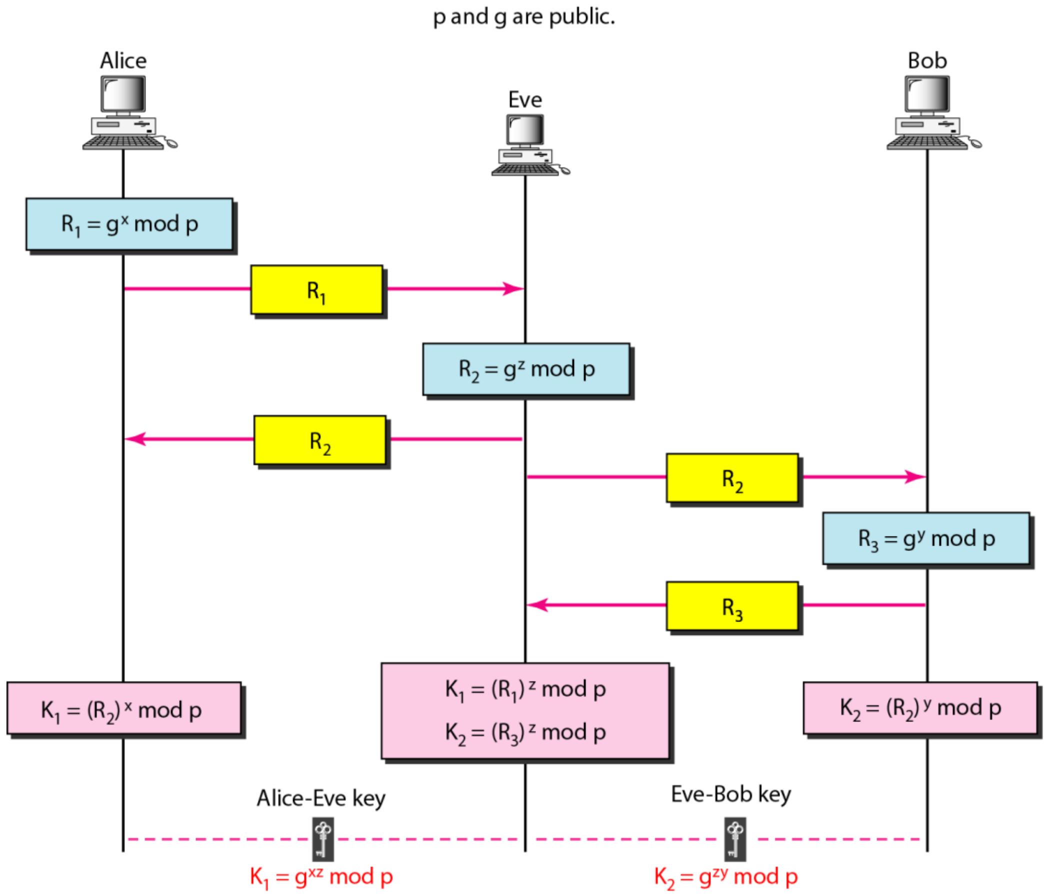
Example 30.10 (4^{th} Edition)

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: ***Man-in-the-middle attack***



Message Integrity

- Two pairs (document/fingerprint) and (message/message digest) are similar, with some differences
- The message digest needs to be safe from change. Digest is much shorter than the message

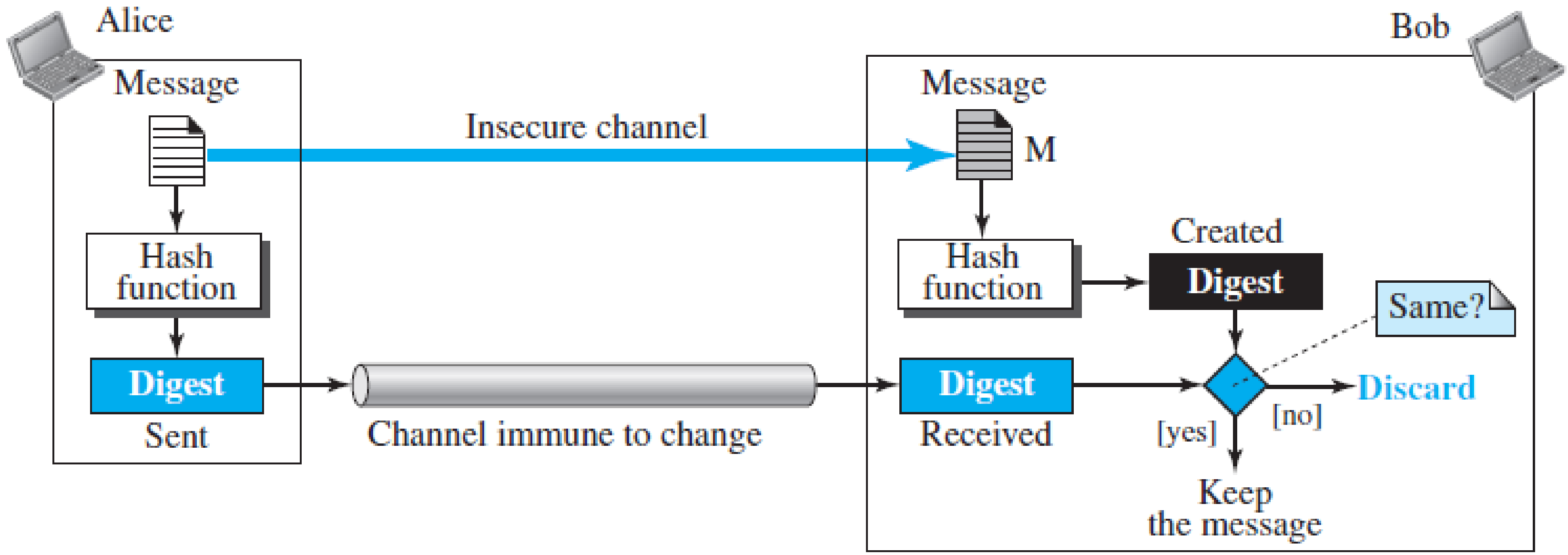


Figure: message and digest

Hash Functions

- MD2, MD4, MD5 where MD stands for Message Digest
- Secure Hash Algorithm (SHA)

Message Integrity

- A digest can be used to check the integrity of a message
- Who is the originator of the message?
- Need to create a **Message Authentication Code** (MAC)
- HMAC (hashed MAC)

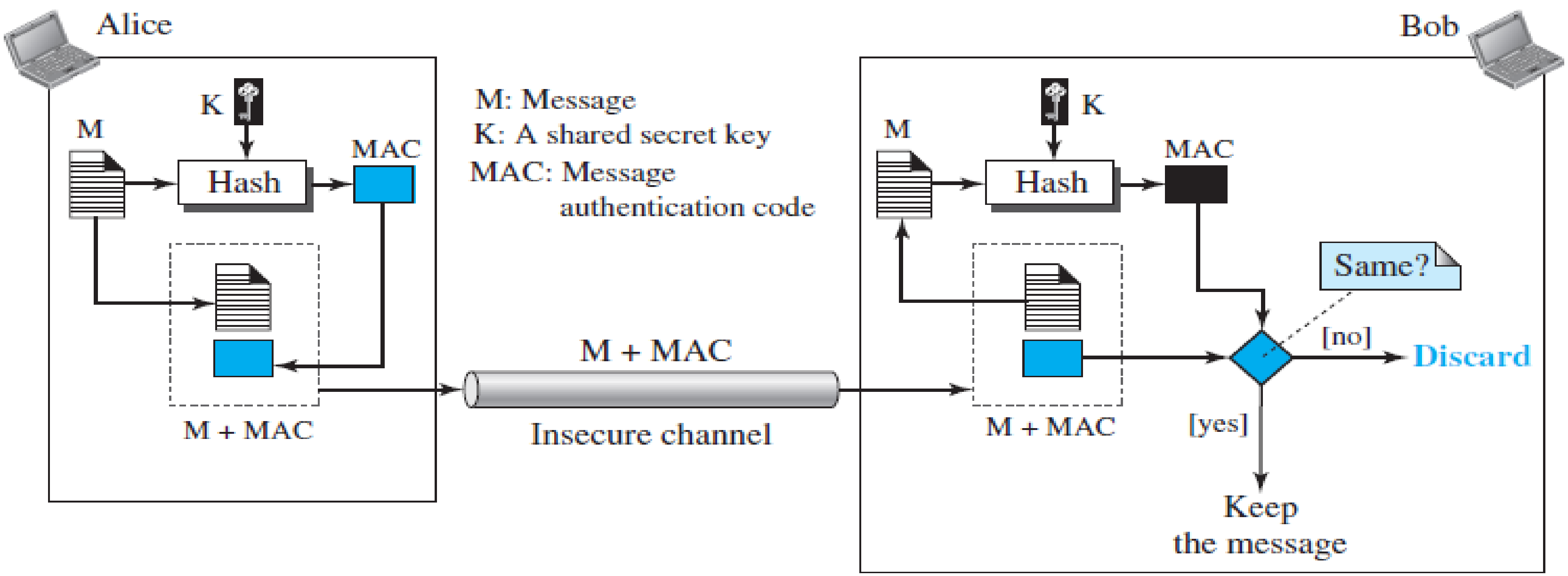


Fig.: Message authentication code

Digital Signature

- Uses the private and public keys of the sender
- Asymmetric-key cryptosystems are very inefficient when dealing with long messages. What is the solution? (See next slide)
- Services: Message authentication, Message Integrity, Nonrepudiation (using **trusted third party**)
- Does not provide confidential communication. If required then need to encrypt message and signature

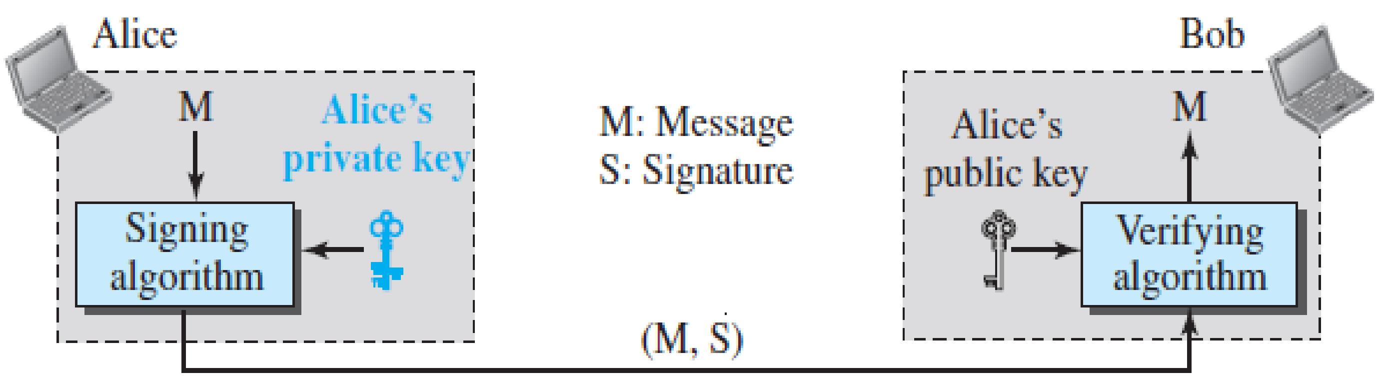
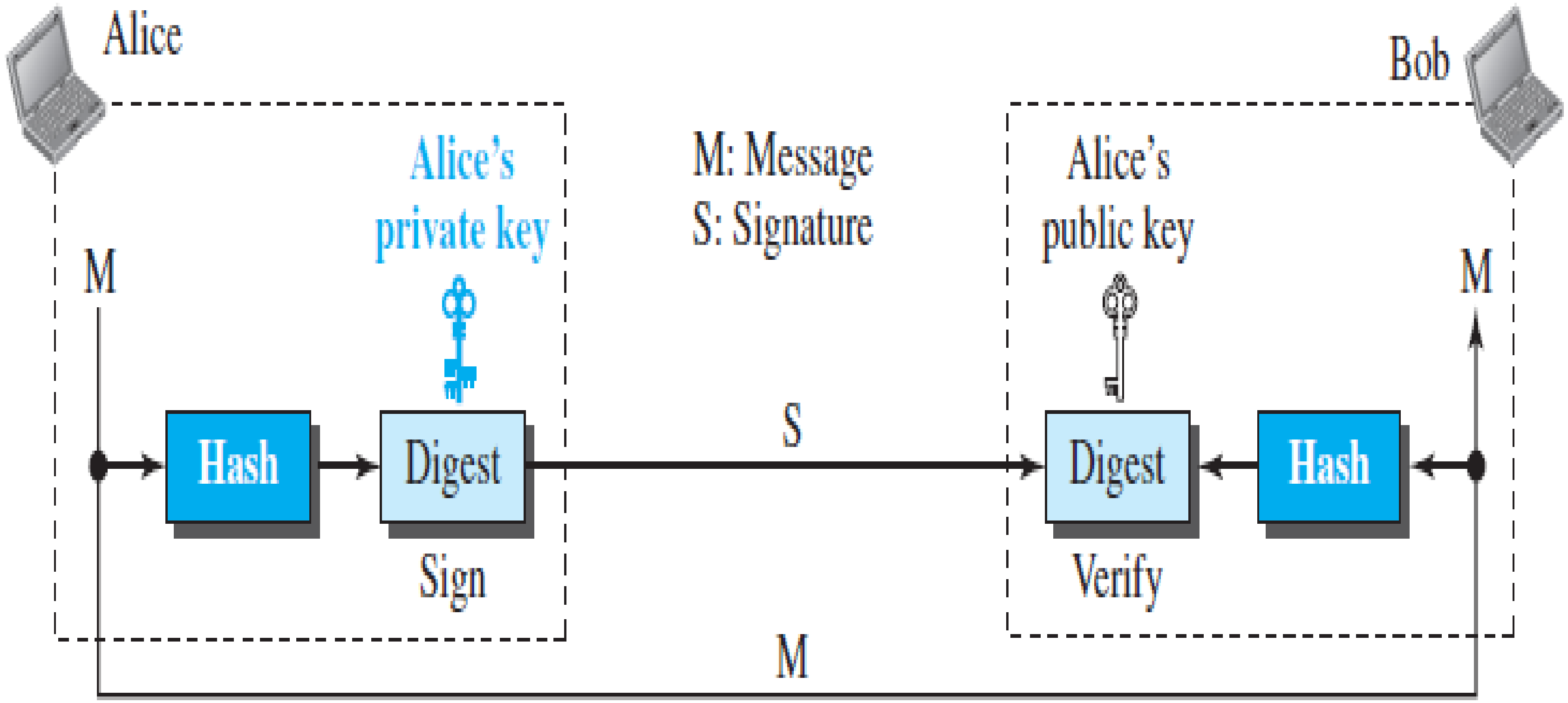


Fig.: Process of Digital Signature

Signing the digest



Encryption and Decryption

\# encrypt file.txt to file.enc using 256-bit AES in CBC mode
>**openssl enc -aes-256-cbc -in file.txt -out file.enc**

 # decrypt binary file.enc
>openssl enc -d -aes-256-cbc -in file.enc

 # see the list under the 'Cipher commands' heading
>openssl -h

Digital Signature

#Generate Public/Private key pair

```
>openssl genrsa -out mykey.pem
>openssl rsa -in mykey.pem -pubout >mypub.pem
```

#Create the signature

```
>openssl dgst -sha1 -sign mykey.pem -out mysign.sha1 file.txt
```

#Verify the signature

```
>openssl dgst -sha1 -verify mypub.pem –signature mysign.sha1 file.txt
```

See Next Slide: https://jumpnowtek.com/security/Code-signing-with-openssl.html

Digital Signature (Using EC Algorithm)

#Generate Private/Public key pair

>openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -out ec-private.pem

>openssl pkey -in ec-private.pem -pubout -out ec-public.pem

#Create the signature

>openssl dgst -sha3-512 -sign ec-private.pem -out data.sig file.txt

#Verify the signature

>openssl dgst -sha3-512 -verify ec-public.pem -signature data.sig data file.txt

Change file.txt or algorithm whether verification works or not