Chapter 3
Cryptography

Chapter 3

Objectives

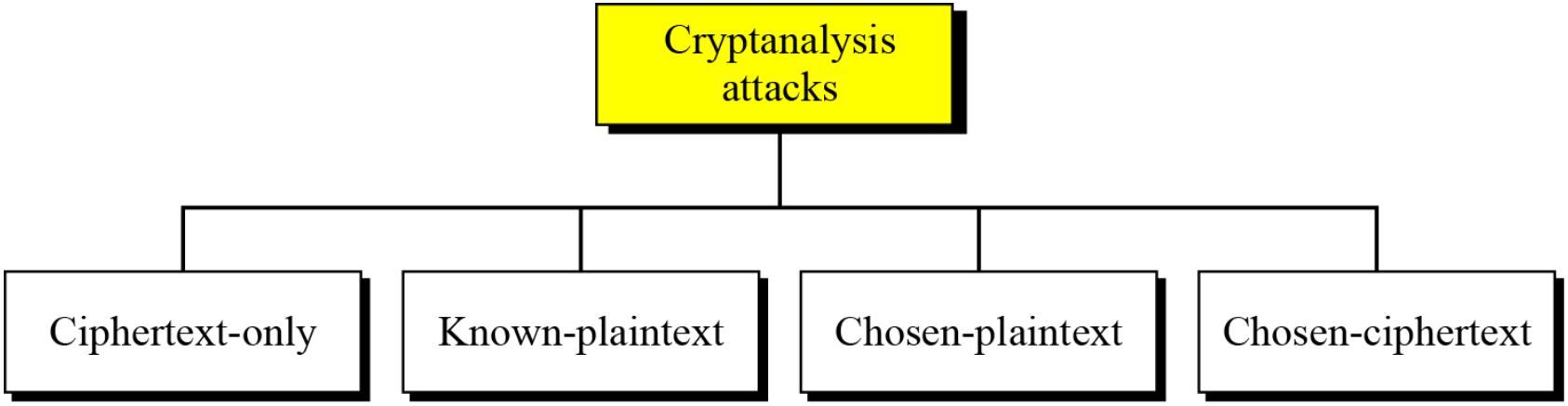
- Two categories of traditional ciphers: substitution and transposition ciphers.
- To describe the categories of cryptanalysis used to break the symmetric ciphers

SAMPLE ENCRYPTION AND DECRYPTION PROCESS

3.1.2 Cryptanalysis

As cryptography is the science and art of creating secret codes, cryptanalysis is the science and art of breaking those codes.

Figure 3.3 *Cryptanalysis attacks*



3-2 SUBSTITUTION CIPHERS

A substitution cipher replaces one symbol with another. Substitution ciphers can be categorized as either monoalphabetic ciphers or polyalphabetic ciphers.

Note

A substitution cipher replaces one symbol with another.

Topics discussed in this secton:

3.2.1 Monoalphabetic Ciphres
3.2.2 Polyalphabetic Ciphers

3.2.1 Monoalphabetic Ciphers

Note

In monoalphabetic substitution, the relationship between a symbol in the plaintext to a symbol in the ciphertext is always one-to-one.

3.2.1 Continued

Example 3.1

The following shows a plaintext and its corresponding ciphertext. The cipher is probably monoalphabetic because both *l*'s (els) are encrypted as *O*'s.

Plaintext: hello **Ciphertext:** KHOOR

Example 3.2

3.2.1 Continued

Additive Cipher

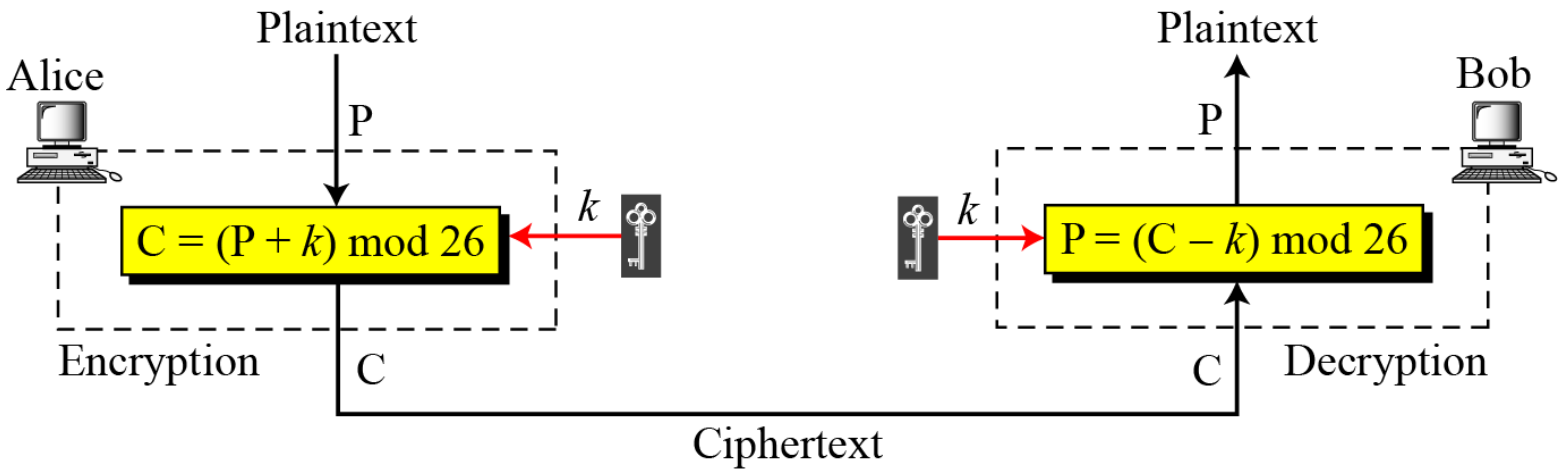
The simplest monoalphabetic cipher is the additive cipher. This cipher is sometimes called a shift cipher and sometimes a Caesar cipher, but the term additive cipher better reveals its mathematical nature.

Figure 3.8 *Plaintext and ciphertext in Z_{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

3.2.1 Continued

Figure 3.9 *Additive cipher*



Note

When the cipher is additive, the plaintext, ciphertext, and key are integers in Z_{26}.

Additive Modulo 26

Z26= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 21, 22, 23, 24, 25}

Multiplicative Modulo 26

Z26*: {1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25 }

Inverses mod 26												
x	1	3	5	7	9	11	15	17	19	21	23	25
x^{-1}	1	9	21	15	3	19	7	23	11	5	17	25

3.2.1 Continued

Example 3.3

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

3.2.1 Continued

Example 3.4

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

For negative number modulo:

To find –b mod N, just add N to –b until the number between 0 and N

3.2.1 Continued

Shift Cipher and Caesar Cipher

Historically, additive ciphers are called shift ciphers. Julius Caesar used an additive cipher to communicate with his officers. For this reason, additive ciphers are sometimes referred to as the Caesar cipher. Caesar used a key of 3 for his communications.

Note

Additive ciphers are sometimes referred to as shift ciphers or Caesar cipher.

3.2.1 Continued

Example 3.5

Eve has intercepted the ciphertext "UVACLYFZLJBYL". Show how she can use a brute-force attack to break the cipher.

Solution

Eve tries keys from 1 to 7. With a key of 7, the plaintext is "not very secure", which makes sense.

Ciphertext: UVACLYFZLJBYL

K = 1	→	**Plaintext:** tuzbkxeykiaxk
K = 2	→	**Plaintext:** styajwdxjhzwj
K = 3	→	**Plaintext:** rsxzivcwigyvi
K = 4	→	**Plaintext:** qrwyhubvhfxuh
K = 5	→	**Plaintext:** pqvxgtaugewtg
K = 6	→	**Plaintext:** opuwfsztfdvsf
K = 7	→	**Plaintext:** notverysecure

3.2.1 Continued

Table 3.1 Frequency of characters in English

Letter	*Frequency*	*Letter*	*Frequency*	*Letter*	*Frequency*	*Letter*	*Frequency*
E	12.7	H	6.1	W	2.3	K	0.08
T	9.1	R	6.0	F	2.2	J	0.02
A	8.2	D	4.3	G	2.0	Q	0.01
O	7.5	L	4.0	Y	2.0	X	0.01
I	7.0	C	2.8	P	1.9	Z	0.01
N	6.7	U	2.8	B	1.5		
S	6.3	M	2.4	V	1.0		

Table 3.2 Frequency of diagrams and trigrams

Digram	TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI, OF
Trigram	THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR, DTH

3.2.1 Continued

Example 3.6

Eve has intercepted the following ciphertext. Using a statistical attack, find the plaintext.

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILY-VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW

Solution

When Eve tabulates the frequency of letters in this ciphertext, she gets: I =14, V =13, S =12, and so on. The most common character is I with 14 occurrences. This means key = 4.

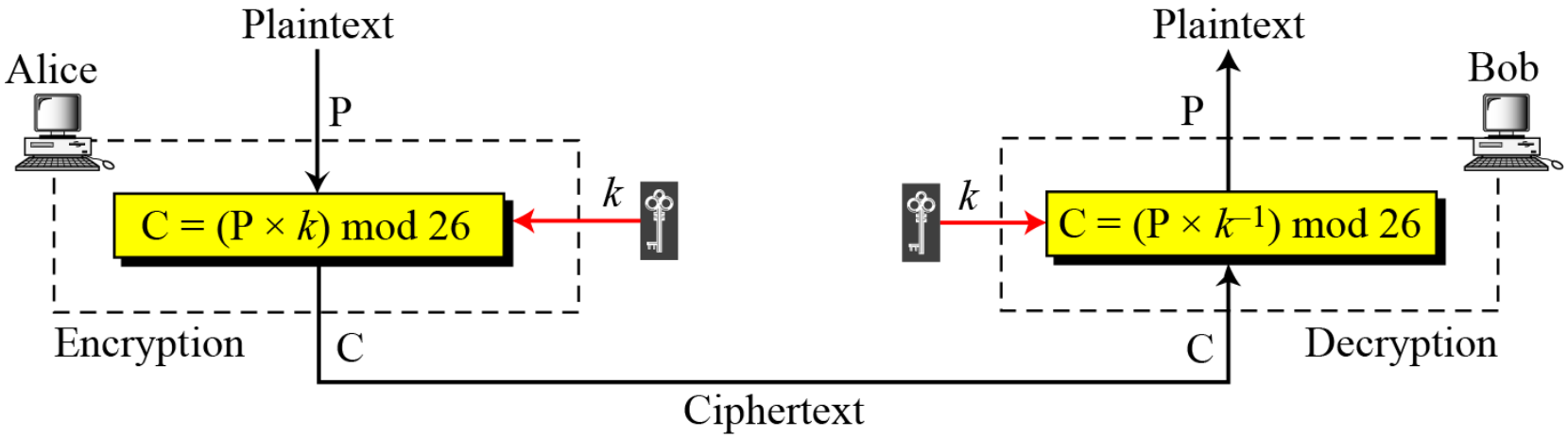
the house is now for sale for four million dollars it is worth more hurry before the seller receives more offers

Homework:

Use the additive cipher with key = 8 to encrypt the message “subject”. Perform decryption also.

Multiplicative Ciphers

Figure 3.10 *Multiplicative cipher*



Note

In a multiplicative cipher, the plaintext and ciphertext are integers in Z_{26}; the key is an integer in Z_{26}*.

3.2.1 Continued

Example 3.7

What is the key domain for any multiplicative cipher?

Solution

The key needs to be in Z_{26}^*. This set has only 12 members: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.

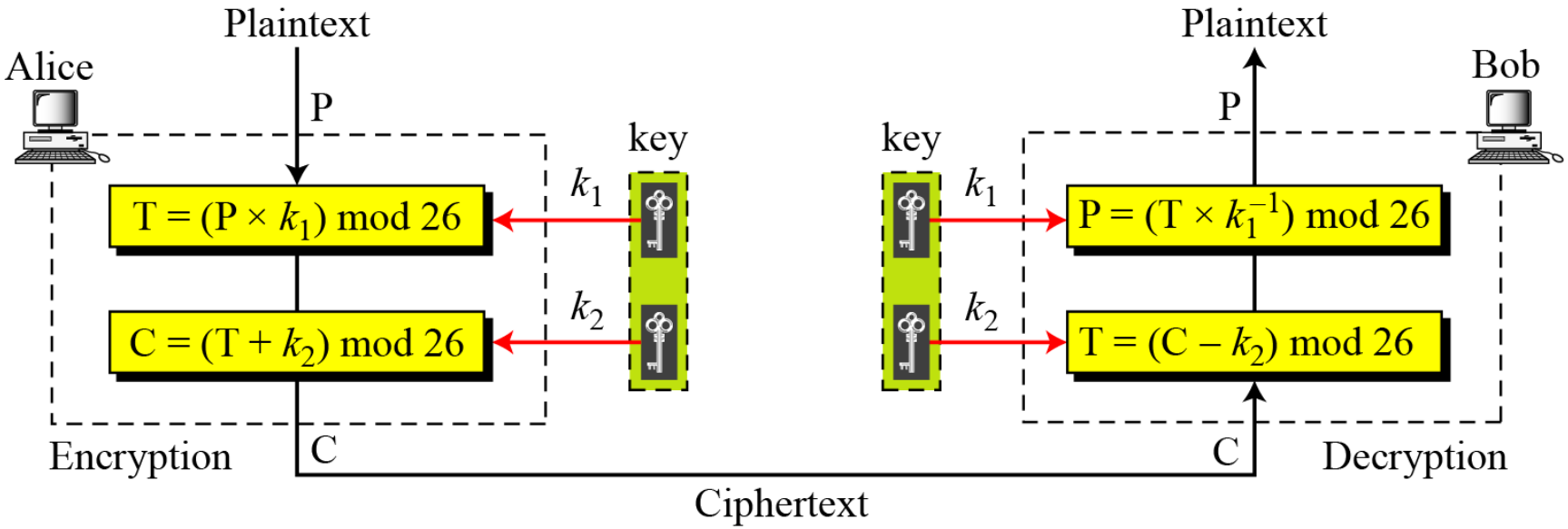
Example 3.8

We use a multiplicative cipher to encrypt the message "hello" with a key of 7. The ciphertext is "XCZZU".

Plaintext: h $\rightarrow$ 07	Encryption: $(07 \times 07) \bmod 26$	ciphertext: 23 $\rightarrow$ X
Plaintext: e $\rightarrow$ 04	Encryption: $(04 \times 07) \bmod 26$	ciphertext: 02 $\rightarrow$ C
Plaintext: l $\rightarrow$ 11	Encryption: $(11 \times 07) \bmod 26$	ciphertext: 25 $\rightarrow$ Z
Plaintext: l $\rightarrow$ 11	Encryption: $(11 \times 07) \bmod 26$	ciphertext: 25 $\rightarrow$ Z
Plaintext: o $\rightarrow$ 14	Encryption: $(14 \times 07) \bmod 26$	ciphertext: 20 $\rightarrow$ U

Affine Ciphers

Figure 3.11 *Affine cipher*



$$C = (P \times k_1 + k_2) \bmod 26 \qquad P = ((C - k_2) \times k_1^{-1}) \bmod 26$$

where k_1^{-1} is the multiplicative inverse of k_1 and $-k_2$ is the additive inverse of k_2

3.2.1 Continued

Example 3.09

The affine cipher uses a pair of keys in which the first key is from Z_{26}^* and the second is from Z_{26}. The size of the key domain is $26 \times 12 = 312$.

Example 3.10

Use an affine cipher to encrypt the message "hello" with the key pair (7, 2).

P: h $\rightarrow$ 07	Encryption: $(07 \times 7 + 2) \bmod 26$	C: 25 $\rightarrow$ Z
P: e $\rightarrow$ 04	Encryption: $(04 \times 7 + 2) \bmod 26$	C: 04 $\rightarrow$ E
P: l $\rightarrow$ 11	Encryption: $(11 \times 7 + 2) \bmod 26$	C: 01 $\rightarrow$ B
P: l $\rightarrow$ 11	Encryption: $(11 \times 7 + 2) \bmod 26$	C: 01 $\rightarrow$ B
P: o $\rightarrow$ 14	Encryption: $(14 \times 7 + 2) \bmod 26$	C: 22 $\rightarrow$ W

Example 3.11

Use the affine cipher to decrypt the message "ZEBBW" with the key pair (7, 2) in modulus 26.

Solution

C: Z → 25	Decryption: $((25 - 2) \times 7^{-1}) \bmod 26$	P:07 → h
C: E → 04	Decryption: $((04 - 2) \times 7^{-1}) \bmod 26$	P:04 → e
C: B → 01	Decryption: $((01 - 2) \times 7^{-1}) \bmod 26$	P:11 → l
C: B → 01	Decryption: $((01 - 2) \times 7^{-1}) \bmod 26$	P:11 → l
C: W → 22	Decryption: $((22 - 2) \times 7^{-1}) \bmod 26$	P:14 → o

Example 3.12

The additive cipher is a special case of an affine cipher in which $k_1 = 1$. The multiplicative cipher is a special case of affine cipher in which $k_2 = 0$.

Homework:

Use the affine cipher with key =(to encrypt the message “subject”. Perform decryption also.

Monoalphabetic Substitution Cipher - cryptanalysis

Because additive, multiplicative, and affine ciphers have small key domains, they are very vulnerable to brute-force attack.

A better solution is to create a mapping between each plaintext character and the corresponding ciphertext character. Alice and Bob can agree on a table showing the mapping for each character.

Figure 3.12 *An example key for monoalphabetic substitution cipher*

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

3.2.1 Continued

Monoalphabetic Substitution Cipher - cryptanalysis

Example 3.13

We can use the key in Figure 3.12 to encrypt the message

this message is easy to encrypt but hard to find the key

The ciphertext is

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

The monoalphabetic ciphers do not change the frequency of characters in the ciphertext, which makes the ciphers venerable to statistical attack

3.2.2 Polyalphabetic Ciphers

In polyalphabetic substitution, each occurrence of a character may have a different substitute. The relationship between a character in the plaintext to a character in the ciphertext is one-to-many.

Autokey Cipher

$$P = P_1 P_2 P_3 \ldots \qquad C = C_1 C_2 C_3 \ldots \qquad k = (k_1, P_1, P_2, \ldots)$$

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

3.2.2 Continued

Example 3.14

Assume that Alice and Bob agreed to use an autokey cipher with initial key value $k_1 = 12$. Now Alice wants to send Bob the message "Attack is today". Enciphering is done character by character.

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**

Cryptanalysis:

Autokey cipher definitely hides the single letter frequency statistics of the plaintext

It is still vulnerable to the brute force attack as an additive cipher

Playfair Cipher

Used by the British Army during World War I

An example of a secret key in the Playfair cipher is shown below:

- Key is of 5 X 5 Matrix
- Letters I and J are considered the same
- Different arrangements of letters in the matrix can create different secret keys

Secret Key =

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

Playfair Cipher

Before Encryption:

- Pair the letters of plaintext from left, if letters in pair are the same a bogus letter is inserted to separate them
- After inserting bogus letters, if the number of characters in the plaintext is odd, one extra bogus letter is add at the end to make the number of characters even

Encryption:

- If the two letters in a pair are **in same row of the secret key**, encrypted character for each letter **is the next letter to the right in the same row**.
- If the two letters in a pair are **in same column of the secret key**, encrypted character for each letter **is the letter beneath it in the same column.**
- If the two letters in a pair **are not in same row or column of the secret key**, encrypted character for each letter is **a letter that is in its own row but in the same column as the other one.**

Playfair Cipher

Example 3.15

Let us encrypt the plaintext "hello" using the above key

he → EC lx → QZ lo → BX

Plaintext: hello Ciphertext: ECQZBX

Cryptanalysis:

- **Brute force on a playfair cipher is very difficult. Because of the size of key domain is 25!.**
- **Encipherment hides the single letter frequency of the characters.**
- **However the frequencies of diagrams are preserved, so cryptanalyst can use a ciphertext only attack based on the diagram frequency test to find key.**

Homework: Write an algorithm for playfair cipher decryption and decrypt the ciphertext "ECQZBX" using the above key.

Vigenere Cipher

$P = P_1P_2P_3 \ldots$ $C = C_1C_2C_3 \ldots$ $K = [(k_1, k_2, \ldots, k_m), (k_1, k_2, \ldots, k_m), \ldots]$

Encryption: $C_i = P_i + k_i$ Decryption: $P_i = C_i - k_i$

Example 3.16

We can encrypt the message "She is listening" using the 6-character keyword "PASCAL".

Plaintext:	s	h	e	i	s	l	i	s	t	e	n	i	n	g
P's values:	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Key stream:	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***
C's values:	**07**	**07**	**22**	**10**	**18**	**22**	**23**	**18**	**11**	**6**	**13**	**19**	**02**	**06**
Ciphertext:	**H**	**H**	**W**	**K**	**S**	**W**	**X**	**S**	**L**	**G**	**N**	**T**	**C**	**G**

3.2.2 Continued

Example 3.16

Let us see how we can encrypt the message "She is listening" using the 6-character keyword "PASCAL". The initial key stream is (15, 0, 18, 2, 0, 11). The key stream is the repetition of this initial key stream (as many times as needed).

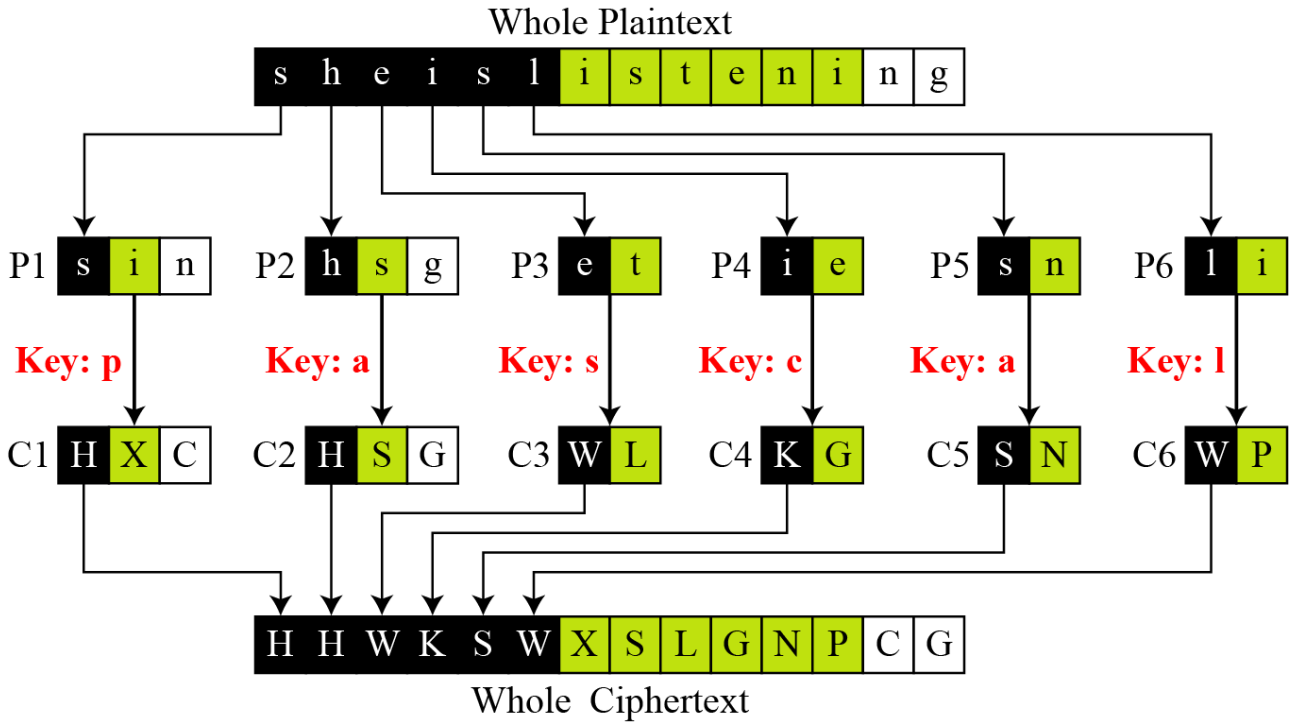
Plaintext:	s	h	e	i	s	l	i	s	t	e	n	i	n	g
P's values:	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Key stream:	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***
C's values:	**07**	**07**	**22**	**10**	**18**	**22**	**23**	**18**	**11**	**6**	**13**	**19**	**02**	**06**
Ciphertext:	**H**	**H**	**W**	**K**	**S**	**W**	**X**	**S**	**L**	**G**	**N**	**T**	**C**	**G**

3.2.2 Continued

Example 3.17

Vigenere cipher can be seen as combinations of m additive ciphers.

Figure 3.14 *A Vigenere cipher as a combination of m additive ciphers*



3.2.2 Continued

Example 3.18

Using Example 3.18, we can say that the additive cipher is a special case of Vigenere cipher in which $m = 1$.

Table 3.3

A Vigenere Tableau

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

3.2.2 Continued

Vigenere Cipher (Crypanalysis)

Example 3.19

Let us assume we have intercepted the following ciphertext:

LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEVLWPCZVGTH-
VTSGXQOVGCSVETQLTJSUMVWVEUVLXEWSLGFZMVVWLGYHCUSWXQH-
KVGSHEEVFLCFDGVSUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUEEHH-
VUCFVGOWICQLTJSUXGLW

The Kasiski test for repetition of three-character segments yields the results shown in Table 3.4.

String	*First Index*	*Second Index*	*Difference*
JSU	68	168	100
SUM	69	117	48
VWV	72	132	60
MPH	119	127	8

3.2.2 Continued

Example 3.19 (Continued)

The greatest common divisor of differences is 4, which means that the key length is multiple of 4. First try $m = 4$.

```
C1: LWGWCRAOKTEPGTQCTJVUEGVGUQGECVPRPVJGTJEUGCJG
P1: jueuapymircneroarhtsthihytrahcieixsthcarrehe
C2: IGGGQHGWGKVCTSOSQSWVWFVYSHSVFSHZHWWFSOHCOQSL
P2: usssctsiswhofeaeceihcetesoecatnpntherhctecex
C3: OFDHURWQZKLZHGVVLUVLSZWHWKHFDUKDHVIWHUHFWLUW
P3: lcaerotnwhiwedssirsiirhketehretltiideatrairt
C4: MEVHCWILEMWVVXGETMEXLMLCXVELGMIMBWXLGEVVITX
P4: iardysehaisrrtcapiafpwtethecarhaesfterectpt
```

In this case, the plaintext makes sense.

Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar cipher. It is an additive cipher with the key set to three. Each character in the plaintext is shifted three characters to create ciphertext.

Hill Cipher

Figure 3.15 *Key in the Hill cipher*

$$K = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1m} \\ k_{21} & k_{22} & \cdots & k_{2m} \\ \vdots & \vdots & & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{mm} \end{bmatrix}$$

$$C_1 = P_1 k_{11} + P_2 k_{21} + \cdots + P_m k_{m1}$$
$$C_2 = P_1 k_{12} + P_2 k_{22} + \cdots + P_m k_{m2}$$
$$\cdots$$
$$C_m = P_1 k_{1m} + P_2 k_{2m} + \cdots + P_m k_{mm}$$

Note

The key matrix in the Hill cipher needs to have a multiplicative inverse.

3.2.2 Continued

Example 3.20

For example, the plaintext "code is ready" can make a 3 × 4 matrix when adding extra bogus character "z" to the last block and removing the spaces. The ciphertext is "OHKNIHGLLISS".

Figure 3.16 *Example 3.20*

$$\underset{C}{\begin{bmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{bmatrix}} = \underset{P}{\begin{bmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{bmatrix}} \underset{K}{\begin{bmatrix} 09 & 07 & 11 & 13 \\ 04 & 07 & 05 & 06 \\ 02 & 21 & 14 & 09 \\ 03 & 23 & 21 & 08 \end{bmatrix}}$$

a. Encryption

$$\underset{P}{\begin{bmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{bmatrix}} \underset{K^{-1}}{\begin{bmatrix} 02 & 15 & 22 & 03 \\ 15 & 00 & 19 & 03 \\ 09 & 09 & 03 & 11 \\ 17 & 00 & 04 & 07 \end{bmatrix}}$$

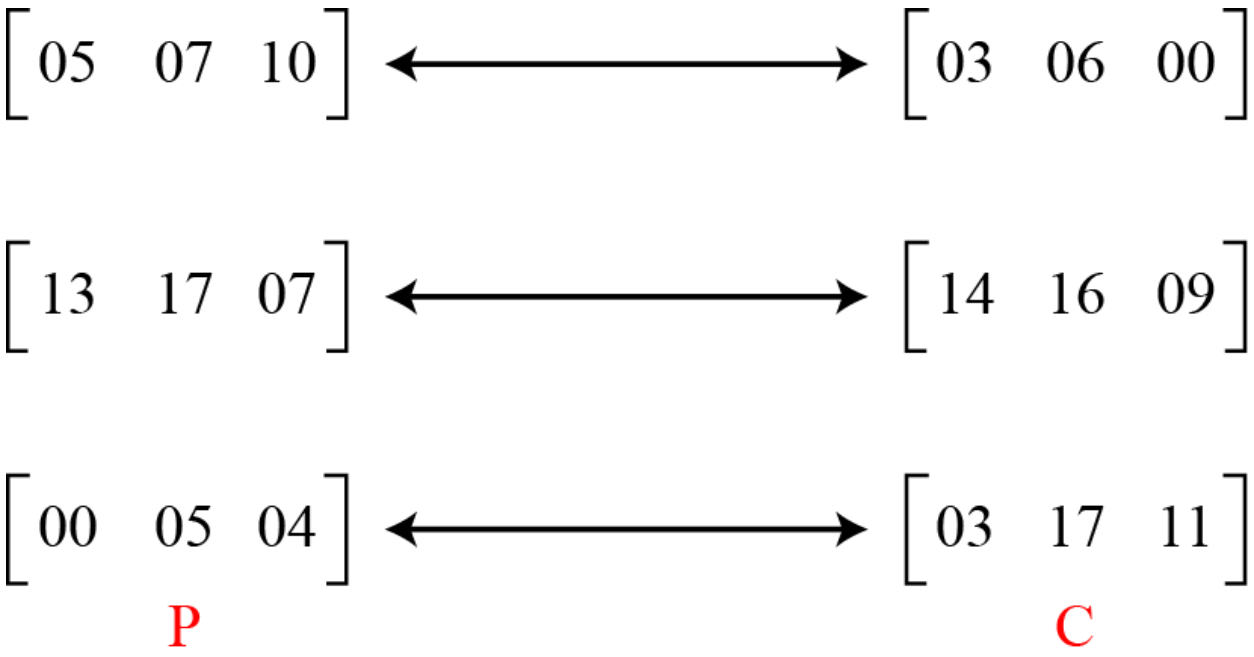
b. Decryption

3.2.2 Continued

Example 3.21

Assume that Eve knows that $m = 3$. She has intercepted three plaintext/ciphertext pair blocks (not necessarily from the same message) as shown in Figure 3.17.

Figure 3.17 *Example 3.21*



Example 3.21 (Continued)

She makes matrices P and C from these pairs. Because P is invertible, she inverts the P matrix and multiplies it by C to get the K matrix as shown in Figure 3.18.

Figure 3.18 *Example 3.21*

$$\begin{bmatrix} 02 & 03 & 07 \\ 05 & 07 & 09 \\ 01 & 02 & 11 \end{bmatrix} = \begin{bmatrix} 21 & 14 & 01 \\ 00 & 08 & 25 \\ 13 & 03 & 08 \end{bmatrix} \begin{bmatrix} 03 & 06 & 00 \\ 14 & 16 & 09 \\ 03 & 17 & 11 \end{bmatrix}$$

$$K \qquad P^{-1} \qquad C$$

Now she has the key and can break any ciphertext encrypted with that key.

3.2.2 Continued

Rotor Cipher

- **It uses the idea behind mono- alphabetic substitution but changes the mapping between the plaintext and the ciphertext characters for each plaintext character**
- **One example is shown below.**
- **It uses 6 letters, but actual rotors use 26 letters**

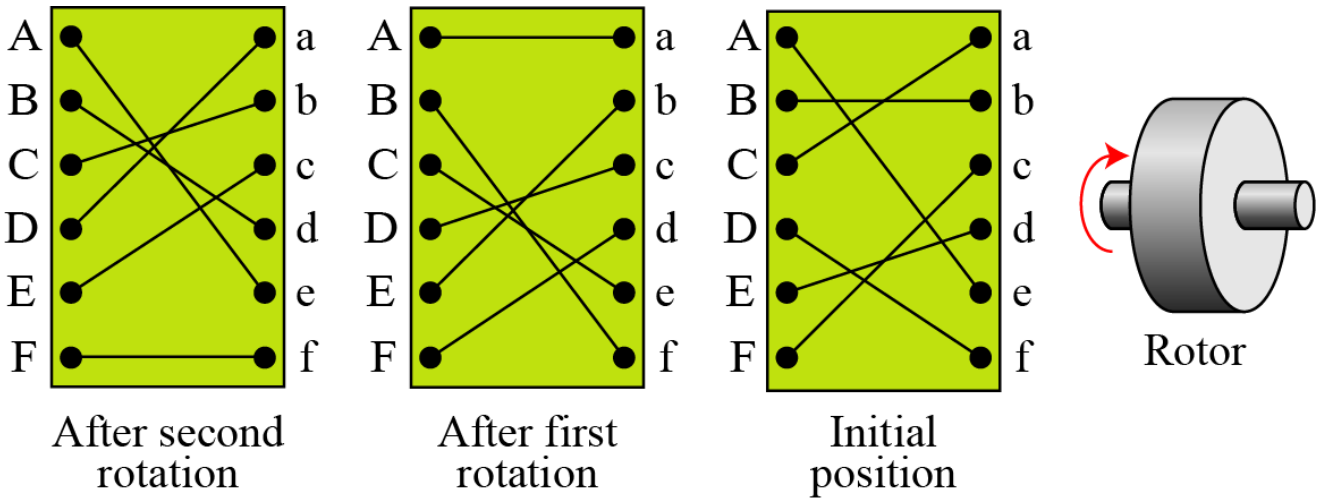


Figure 3.19 *A rotor cipher*

3.2.2 Continued

Rotor Cipher

- It uses the idea behind mono- alphabetic substitution but changes the mapping between the plaintext and the ciphertext characters for each plaintext character
- One example is shown below.
- It uses 6 letters, but actual rotors use 26 letters
- Plaintext = “bee”, Ciphertext = “BAA” , if rotor is stationary
- Plaintext = “bee”, Ciphertext = “BCA” , if rotor is rotating

3-3 TRANSPOSITION CIPHERS

A transposition cipher does not substitute one symbol for another, instead it changes the location of the symbols.

Note

A transposition cipher reorders symbols.

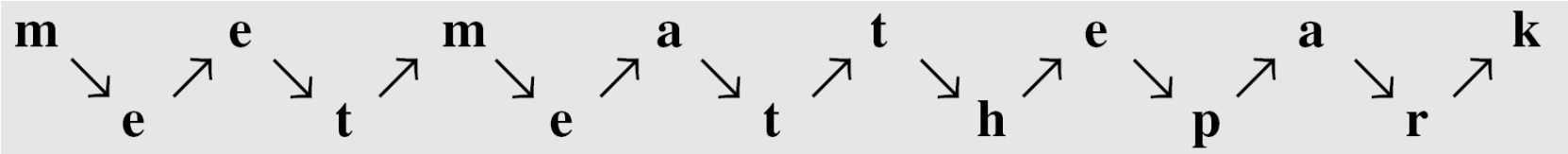
3.3.1 Keyless Transposition Ciphers
3.3.2 Keyed Transposition Ciphers
3.3.3 Combining Two Approaches

3.3.1 Keyless Transposition Ciphers

Simple transposition ciphers, which were used in the past, are keyless.

Example 3.22

A good example of a keyless cipher using the first method is the rail fence cipher. The ciphertext is created reading the pattern row by row. For example, to send the message "Meet me at the park" to Bob, Alice writes



She then creates the ciphertext "MEMATEAKETETHPR".

3.3.1 Continued

Example 3.23

Alice and Bob can agree on the number of columns and use the second method. Alice writes the same plaintext, row by row, in a table of four columns.

m	e	e	t
m	e	a	t
t	h	e	p
a	r	k	

She then creates the ciphertext "MMTAEEHREAEKTTP".

3.3.1 Continued

Example 3.24

The cipher in Example 3.23 is actually a transposition cipher. The following shows the permutation of each character in the plaintext into the ciphertext based on the positions.

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
01	05	09	13	02	06	10	13	03	07	11	15	04	08	12

The second character in the plaintext has moved to the fifth position in the ciphertext; the third character has moved to the ninth position; and so on. Although the characters are permuted, there is a pattern in the permutation: (01, 05, 09, 13), (02, 06, 10, 13), (03, 07, 11, 15), and (08, 12). In each section, the difference between the two adjacent numbers is 4.

3.3.2 Keyed Transposition Ciphers

The keyless ciphers permute the characters by using writing plaintext in one way and reading it in another way The permutation is done on the whole plaintext to create the whole ciphertext. Another method is to divide the plaintext into groups of predetermined size, called blocks, and then use a key to permute the characters in each block separately.

3.3.2 Continued

Example 3.25

Alice needs to send the message "Enemy attacks tonight" to Bob..

e n e m y a t t a c k s t o n i g h t z

The key used for encryption and decryption is a permutation key, which shows how the character are permuted.

Encryption ↓

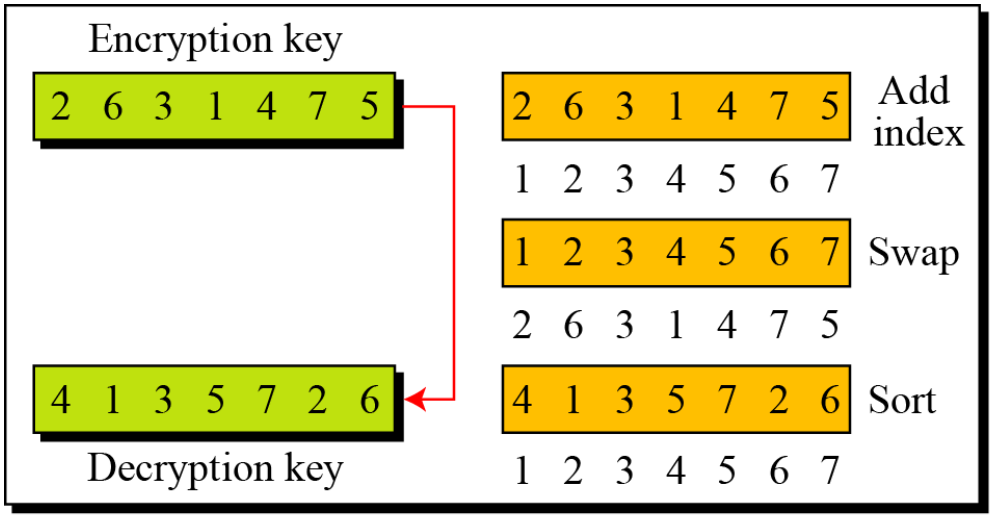
3	1	4	5	2
1	2	3	4	5

↑ Decryption

The permutation yields

E E M Y N T A A C T T K O N S H I T Z G

Figure 3.23 *Key inversion in a transposition cipher*



a. Manual process

```
Given: EncKey [index]
index ← 1
while (index ≤ Column)
  {
    DecKey[EncKey[index]] ← index
    index ← index + 1
  }
Return : DecKey [index]
```

b. Algorithm

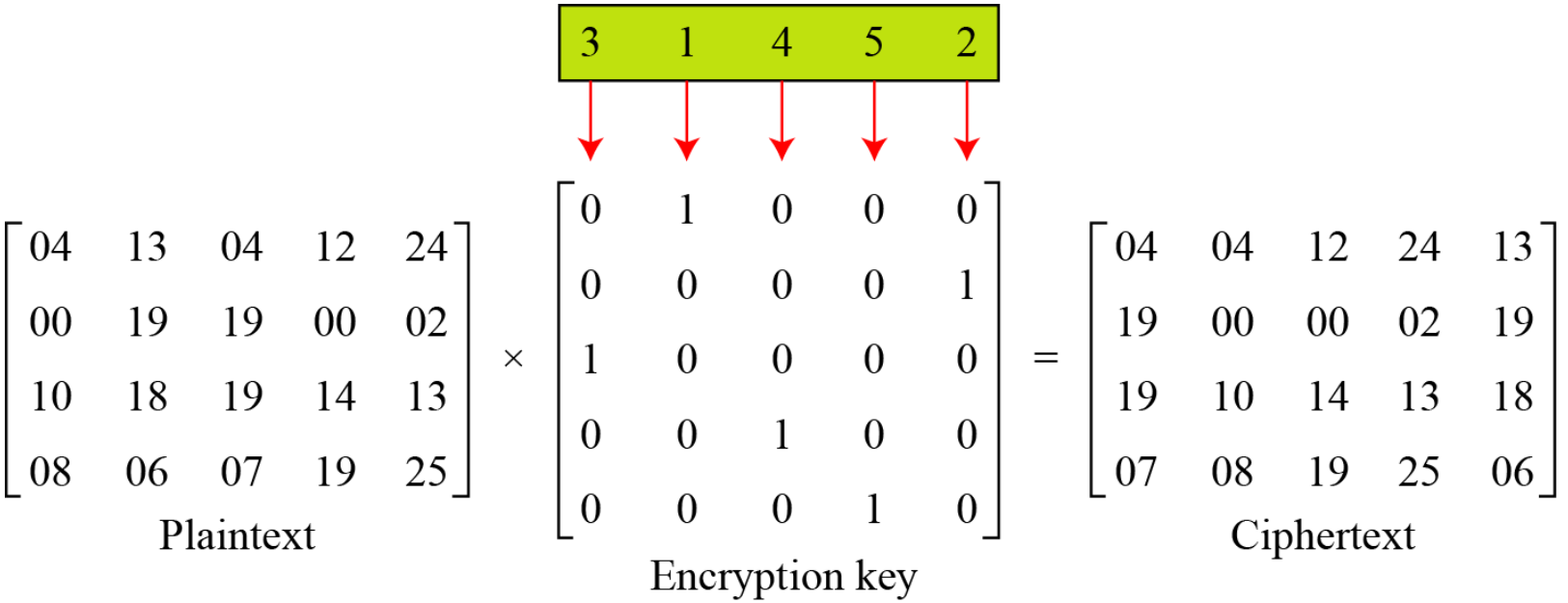
3.3.3 Continued

Using Matrices

We can use matrices to show the encryption/decryption process for a transposition cipher.

Example 3.27

Figure 3.24 ***Representation of the key as a matrix in the transposition cipher***



3.3.3 Continued

Example 3.27

Figure 3.24 shows the encryption process. Multiplying the 4 × 5 plaintext matrix by the 5 × 5 encryption key gives the 4 × 5 ciphertext matrix.

Figure 3.24 *Representation of the key as a matrix in the transposition cipher*

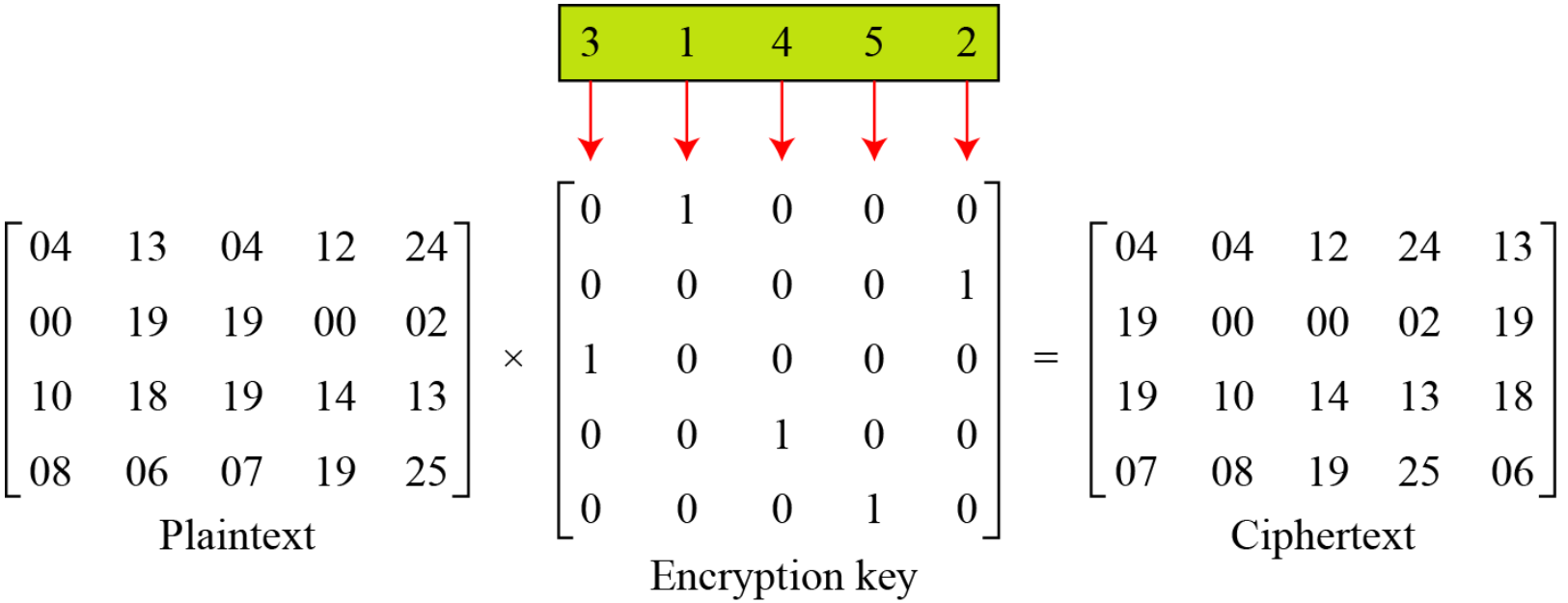
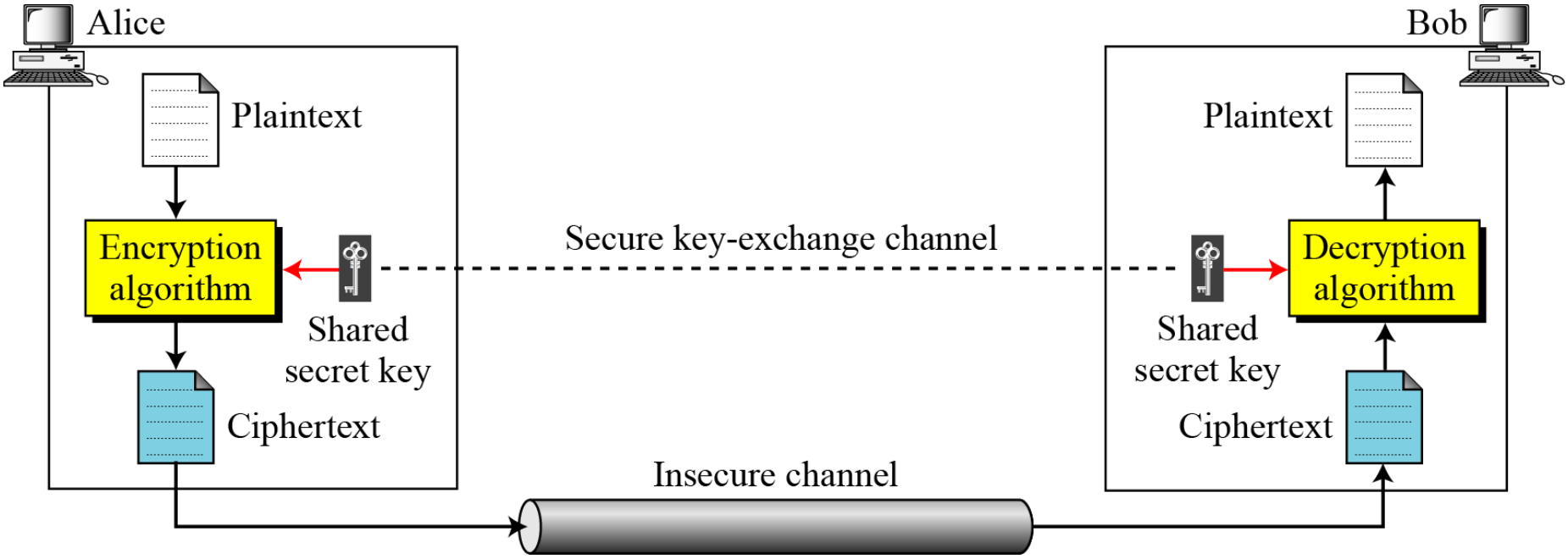


Figure 3.1 *General idea of symmetric-key cipher*



Symmetric-key cryptography

Advantages:

- Simple
- Faster

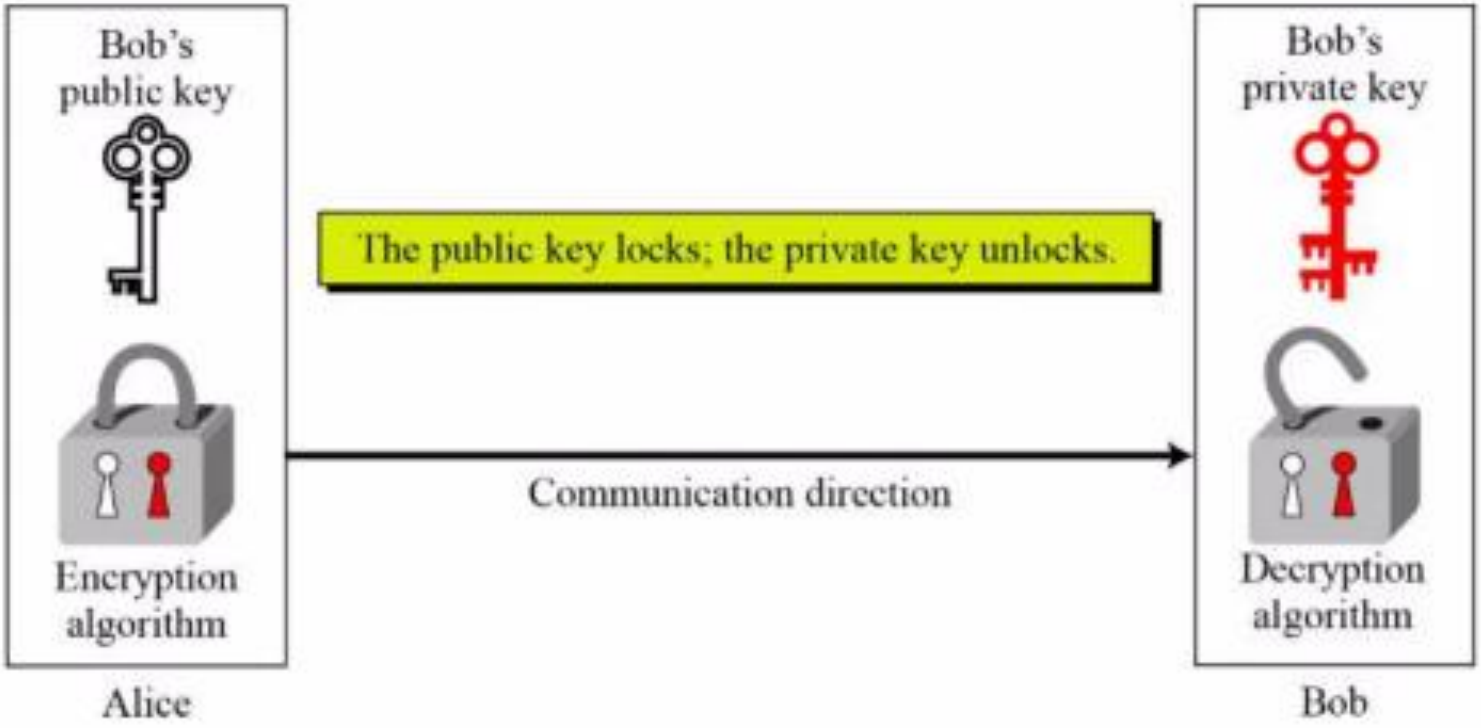
Disadvantages:

- Key must exchanges in secure way
- Easy for hacker to get a key as it is passed in unsecure way.

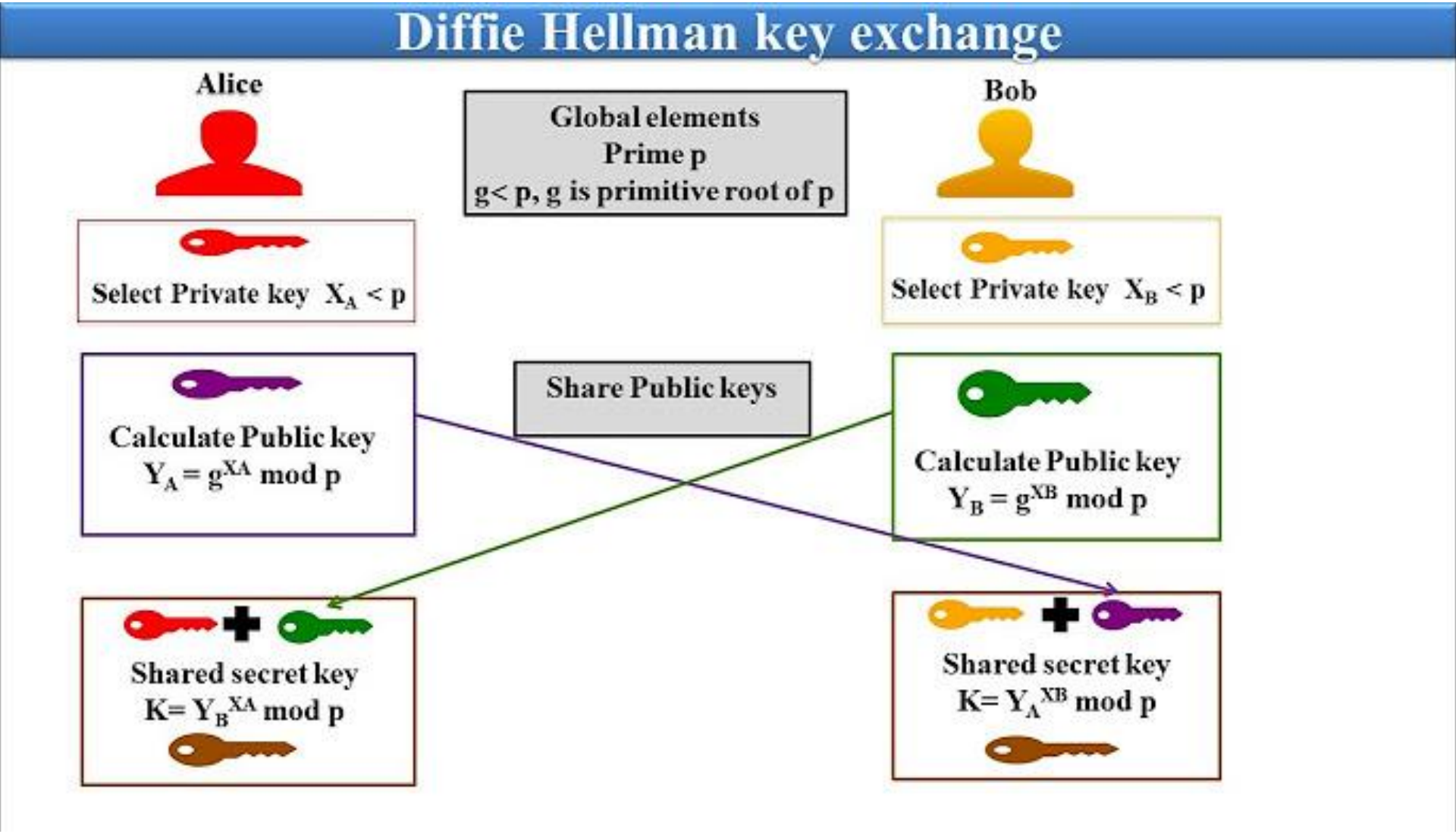
3.1 Asymmetric Cryptography

Asymmetric key cryptography, known as public key cryptography, uses two separate keys: one private and one public.

Figure 10.1 *Locking and unlocking in asymmetric-key cryptosystem*



3.1 Diffie-Hellman key exchange



3.3.3 Continued

Example 3.27

Alice and Bob agree to use the prime p = 17 and the primitive root g =3. Alice chooses the secret key Xa = 12 and Bob chooses the secret key Xb = 14 and computes public keys for both Alice and Bob and shared secret key at both sides.

Public keys : Alice-(3^12) mod 17=4, Bob- (3^14)mod 17=2
Shared secret key – Alice-(2^12)mod 17= 16, Bob -(4^14) mod 17=16
*Easy way –((3^12)^14)mod 17=(3^(12 *14))mod 17*

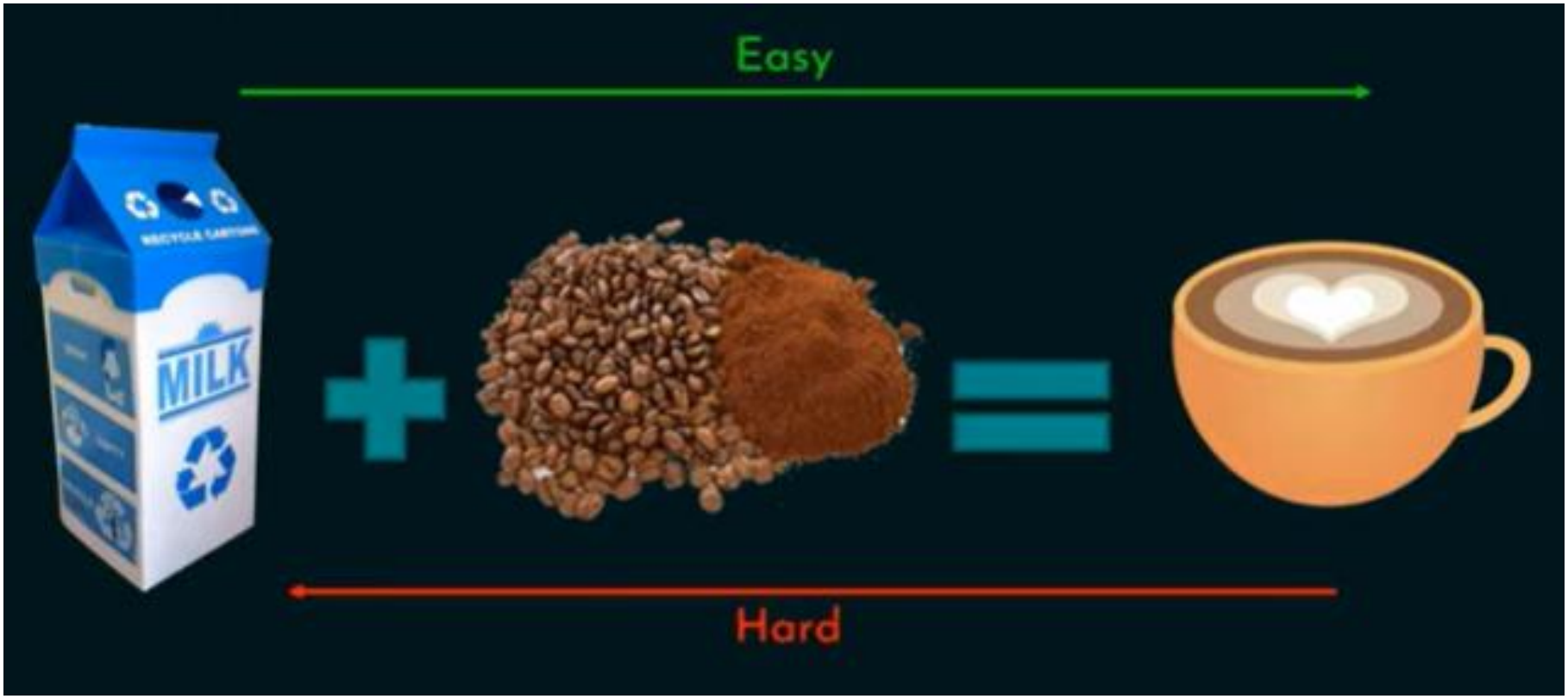
3.3.3 Continued

Example 3.27

Alice and Bob agree to use the prime p = 941 and the primitive root g = 627. Alice chooses the secret key a = 347 and Bob chooses the secret key b = 781 and computes public keys for both Alice and Bob and shared secret key at both sides.

Public keys : Alice-390, Bob-691
Shared secret key - 470

3.1 Discrete Logarithm problem



3.3.3 Continued

Example 3.27

Understanding the DLP

$5^1 \bmod 17 = 5$	$5^9 \bmod 17 = 12$
$5^2 \bmod 17 = 8$	$5^{10} \bmod 17 = 9$
$5^3 \bmod 17 = 6$	$5^{11} \bmod 17 = 11$
$5^4 \bmod 17 = 13$	$5^{12} \bmod 17 = 4$
$5^5 \bmod 17 = 14$	$5^{13} \bmod 17 = 3$
$5^6 \bmod 17 = 2$	$5^{14} \bmod 17 = 15$
$5^7 \bmod 17 = 10$	$5^{15} \bmod 17 = 7$
$5^8 \bmod 17 = 16$	$5^{16} \bmod 17 = 1$

3.3.3 Continued

3.3.3 Continued

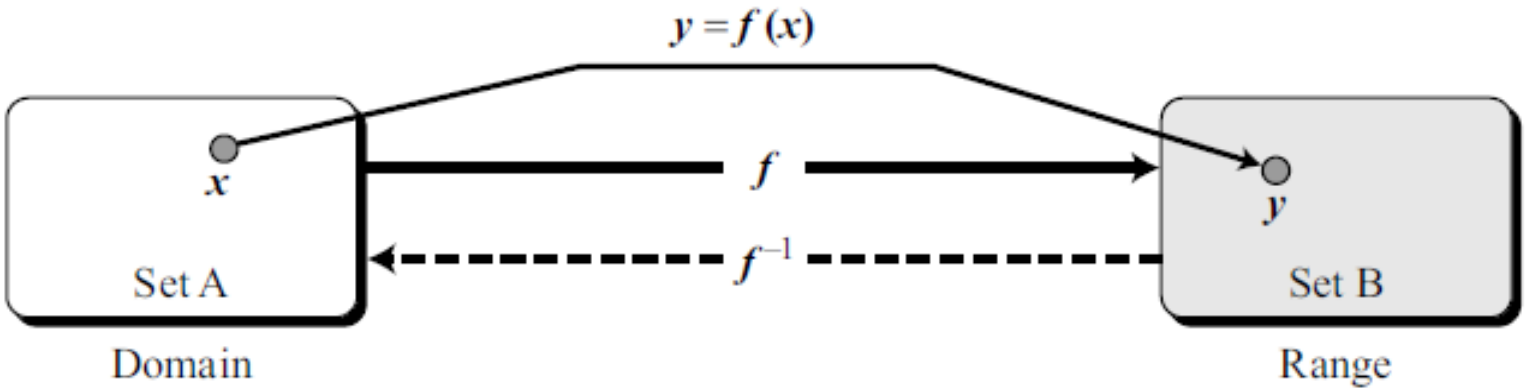
The Discrete Logarithm Problem

- g^X mod p
- $2^X \bmod 7 = 4$; X = 2, 5, etc.,
- For smaller value of 'p' is may be easy to find x.
- If 'p' is very large, then finding 'X' is hard.
- If 'p' is large, then the time and effort to find 'X' is very hard.
- The strength of one-way function is depending on how much time it takes to break it.

3.3.3 Functions

Example 3.27

A function is a rule that associates (maps) one element in set A, called the domain, to one element in set B, called the range.



3.3.3 One way Function

Example 3.27

10.1.4 Continued

One-Way Function (OWF)

1. f is easy to compute.
2. f^{-1} is difficult to compute.

Trapdoor One-Way Function (TOWF)

3. Given y and a trapdoor, x can be computed easily.

10.1.4 Continued

Example 10. 1

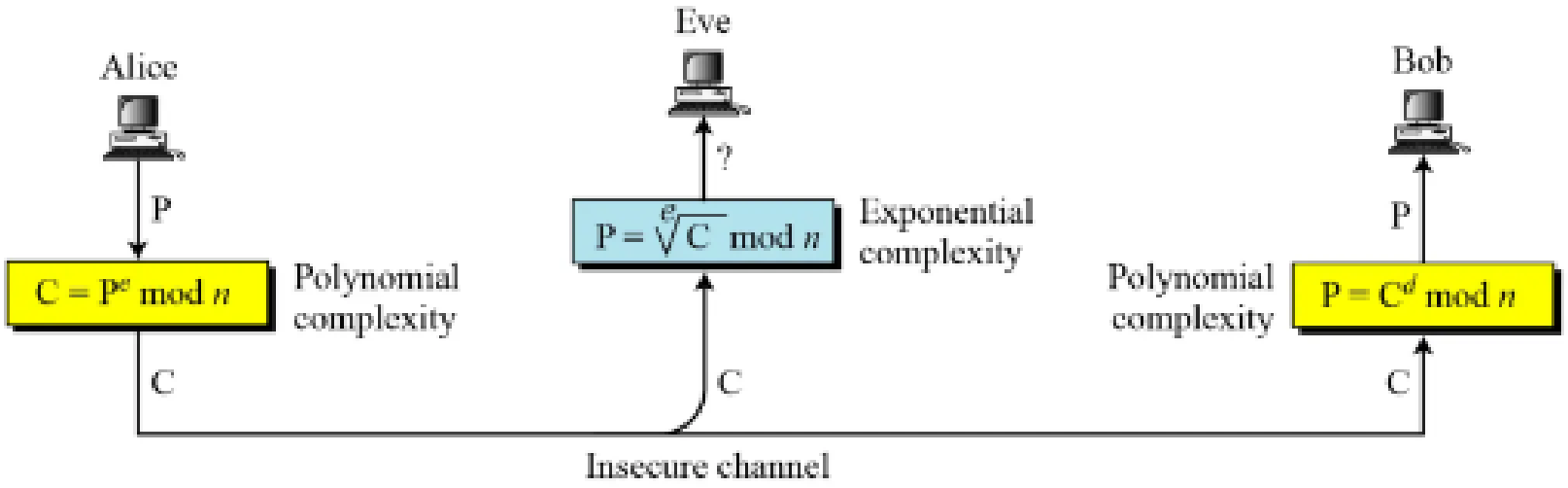
When n is large, $n = p \times q$ is a one-way function. Given p and q , it is always easy to calculate n ; given n, it is very difficult to compute p and q. This is the factorization problem.

Example 10. 2

When n is large, the function $y = x^k \bmod n$ is a trapdoor one-way function. Given x, k, and n, it is easy to calculate y. Given y, k, and n, it is very difficult to calculate x. This is the discrete logarithm problem. However, if we know the trapdoor, k′ such that $k \times k' = 1 \bmod \phi(n)$, we can use $x = y^{k'} \bmod n$ to find x.

3.1 RSA Cryptosystem

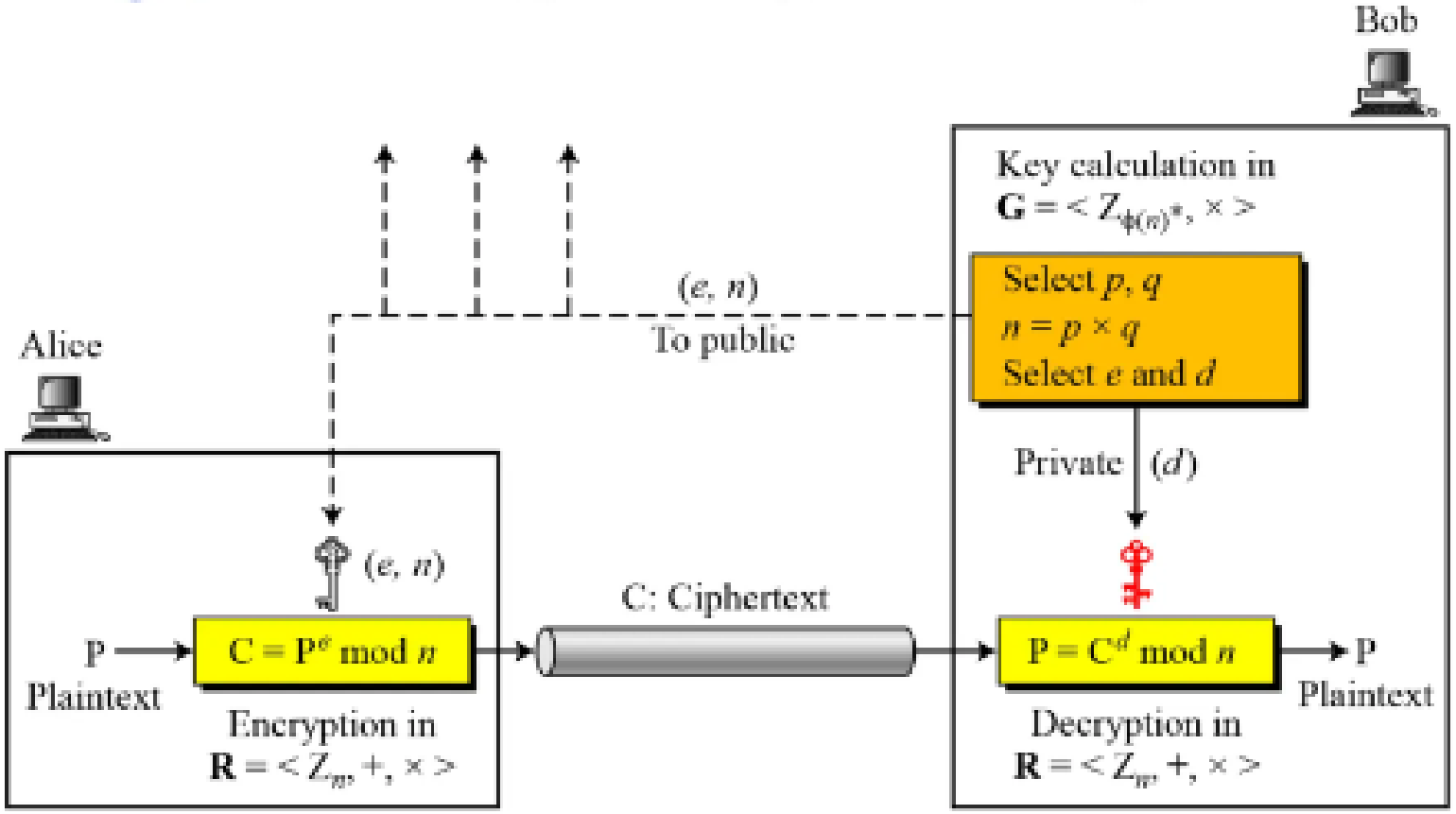
Figure 10.5 *Complexity of operations in RSA*



RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $\sqrt[e]{C} \bmod n$.

10.2.2 Procedure

Figure 10.6 *Encryption, decryption, and key generation in RSA*



Key Generation

Select p, q	p and q both prime
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

RSA Algorithm Example :

1. Let $p = 3, q = 11$
2. $n = p * q = 3 * 11 = 33$
3. $\Phi(n) = (p-1)*(q-1) = 20$
4. Let $e = 7$ such that $1 < 7 < 20$, $GCD(7, 20) = 1$
5. $ed = 1 \bmod \Phi(n)$

   $7 * d = 1 \bmod 20$

   So $7 * d \bmod 20 = 1$

   - $7 * 3 \bmod 20 = 1$
   - Here $d = 3$

Encryption :

Plaintext, $M < e$

$C = M^e \bmod n$

Let $M = 31$

Then , $C = 3^{17} \bmod 33$

We get $C = 4$

Decryption :

Ciphertext, C

$M = C^d \bmod n$

$M = 4^3 \bmod 33$

$M = 31$

Advantages :

- Very fast, very simple encryption and verification.
- Easy to implement than Elliptical Curve Cryptography.
- Easier to Understand
- Widely deployed, better industry support.

Disadvantages :

- Very slow key Generation.
- Slow decryption, which is slightly tricky to implement securely.

10.2.3 Some Trivial Examples

Example 10. 5

Bob chooses 7 and 11 as p and q and calculates $n = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, e and d, from Z_{60}^*. If he chooses e to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

Plaintext: 5 $\quad C = 5^{13} = 26 \bmod 77$ \quad Ciphertext: 26

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

Ciphertext: 26 $\quad P = 26^{37} = 5 \bmod 77$ \quad Plaintext: 5

10.2.3 Some Trivial Examples

Example 10. 6

Now assume that another person, John, wants to send a message to Bob. John can use the same public key announced by Bob (probably on his website), 13; John's plaintext is 63. John calculates the following:

Plaintext: 63	$C = 63^{13} = 28 \bmod 77$	Ciphertext: 28

Bob receives the ciphertext 28 and uses his private key 37 to decipher the ciphertext:

Ciphertext: 28	$P = 28^{37} = 63 \bmod 77$	Plaintext: 63

3.1 Digital Signature

Figure shows the digital signature process. The sender uses a signing algorithm to sign the message. The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the verifying algorithm to the combination. If the result is true, the message is accepted; otherwise, it is rejected.

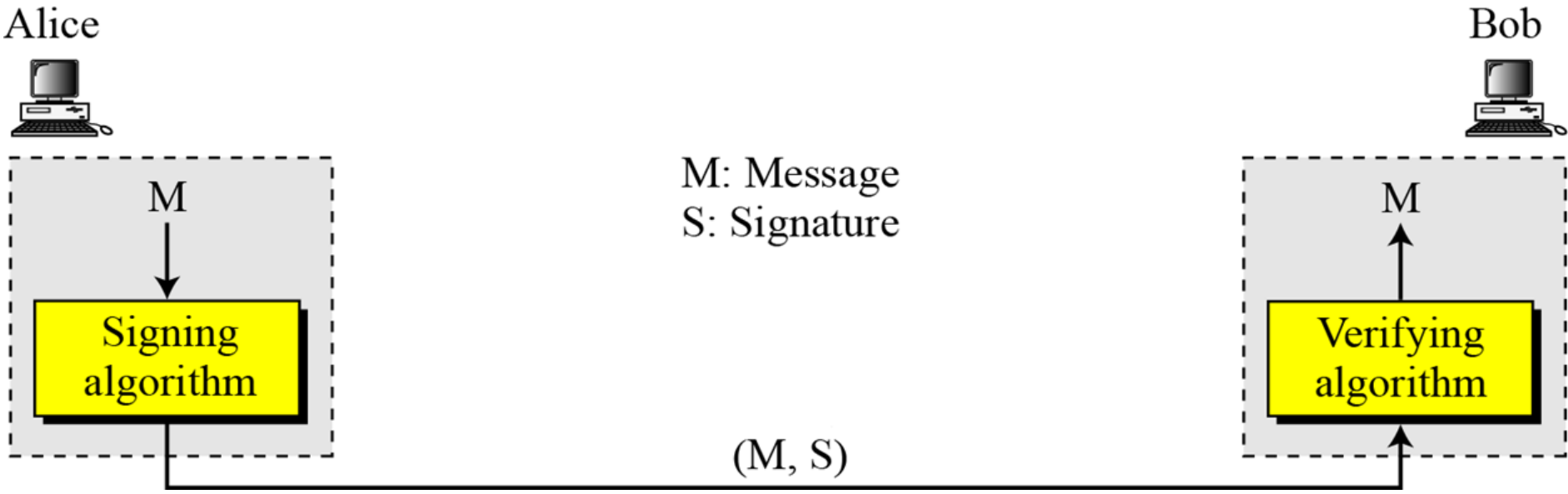
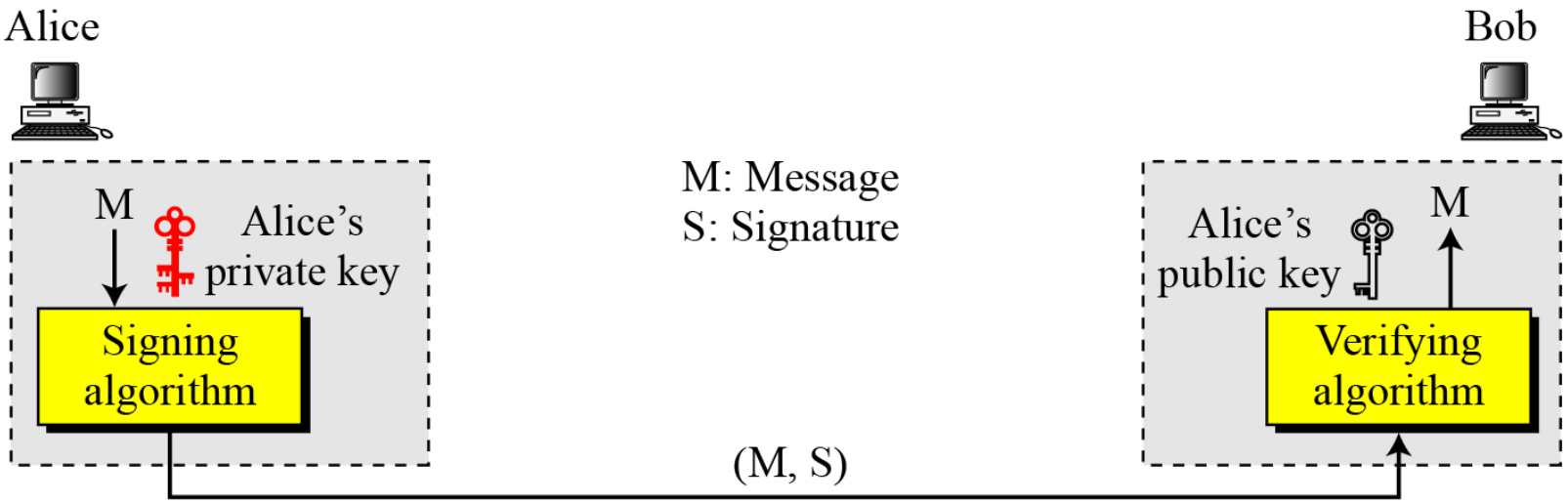


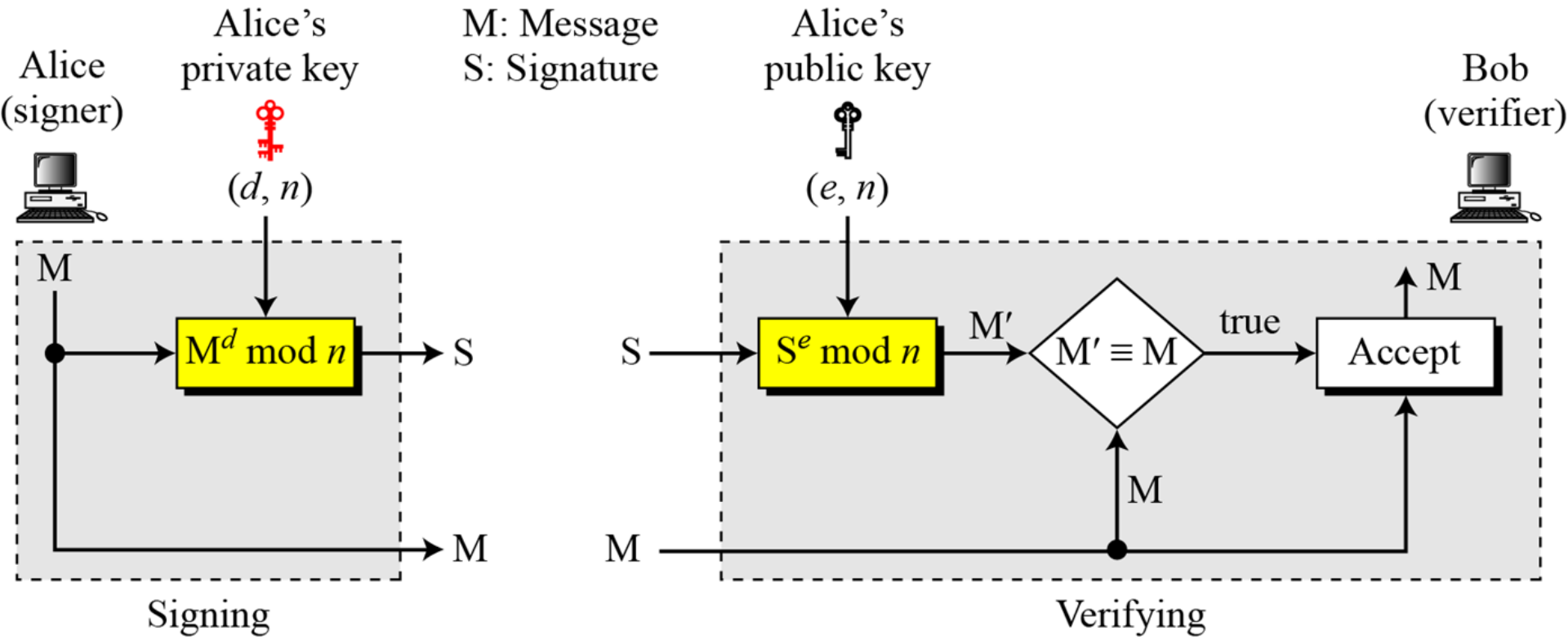
Figure 13.2



Note

A digital signature needs a public-key system. The signer signs with her private key; the verifier verifies with the signer's public key.

3.1 RSA signature schemes



3.3.3 Continued

Example

For the security of the signature, the value of p and q must be very large. As a trivial example, suppose that Alice chooses p = 823 and q = 953, and calculates n = 784319. The value of φ(n) is 782544. Now she chooses e = 313 and calculates d = 160009. At this point key generation is complete.

Now imagine that Alice wants to send a message with the value of M = 19070 to Bob. She uses her private exponent, 160009, to sign the message:

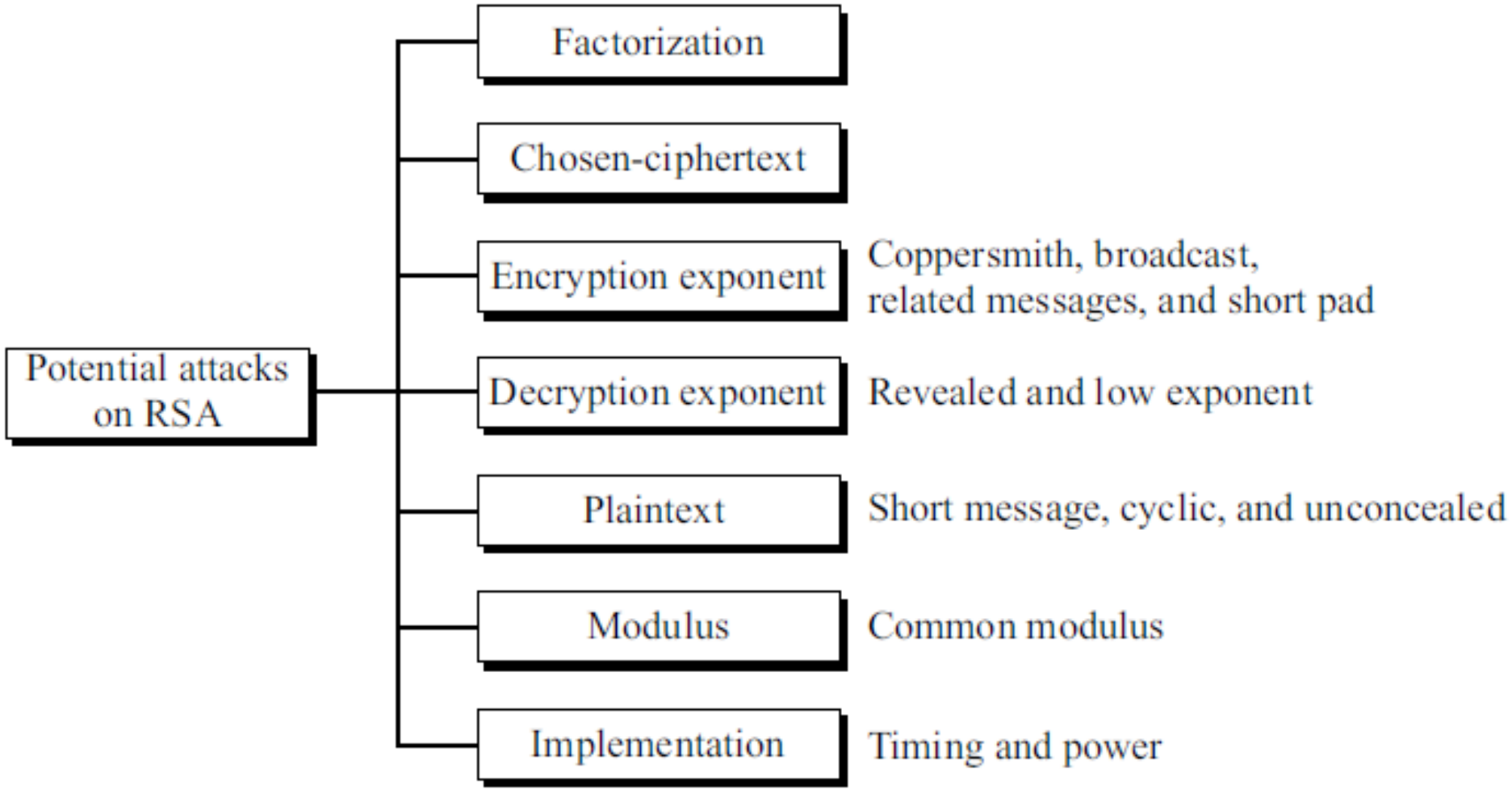
$$\text{M: } 19070 \quad \rightarrow \quad S = (19070^{160009}) \bmod 784319 = 210625 \bmod 784319$$

Alice sends the message and the signature to Bob. Bob receives the message and the signature.

$$M' = 210625^{313} \bmod 784319 = 19070 \bmod 784319 \quad \rightarrow \quad M \equiv M' \bmod n$$

Bob accepts the message because he has verified Alice's signature

3.1 Attacks on RSA



3.3.3 Continued

Factorization Attack

The security of RSA is based on the idea that the modulus is so large that it is infeasible to factor it in a reasonable time. Bob selects p and q and calculates $n = p \times q$.

Although n is public, p and q are secret. If Eve can factor n and obtain p and q, she can calculate $\varphi(n) = (p - 1)(q - 1)$.

Eve then can calculate $d = e^{-1} \bmod \varphi(n)$ because e is public.

To be secure, RSA presently requires that n should be more than 300 decimal digits, which means that the modulus must be at least 1024 bits. Even using the largest and fastest computer available today, factoring an integer of this size would take an infeasibly long period of time.

This means that RSA is secure as long as an efficient algorithm for factorization has not been found.

3.3.3 Continued

Chosen-Ciphertext Attack

A potential attack on RSA is based on the multiplicative property of RSA.

Assume that Alice creates the ciphertext C = Pe mod n and sends C to Bob.

a. Eve chooses a random integer X in Z_n^*.

b. Eve calculates $Y = C \times X^e \bmod n$.

c. Eve sends Y to Bob for decryption and get $Z = Y^d \bmod n$; This step is an instance of a chosen-ciphertext attack.

d. Eve can easily find P because

$$Z = Y^d \bmod n = (C \times X^e)^d \bmod n = (C^d \times X^{ed}) \bmod n = (C^d \times X) \bmod n = (P \times X) \bmod n$$

$$Z = (P \times X) \bmod n \quad \rightarrow \quad P = Z \times X^{-1} \bmod n$$

Still used in SSL/TLS protocol, generaly used for sharing symmetric key sharing, may use padding scheme or hybrid encryption.

3.3.3 Continued

Attacks on the Encryption Exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. The common value for e is e = 3 (the second prime). However, there are some potential attacks on low encryption exponent that we briefly discuss here. These attacks do not generally result in a breakdown of the system, but they still need to be prevented.

In RSA, if d is comprised, then p, q, n, e, and d must be regenerated.

3.3.3 Continued

Attacks on the Modulus

- **The common modulus attack can be launched if a community uses a common modulus, n.**
- **For example, people in a community might let a trusted party select p and q, calculate n and φ(n), and create a pair of exponents (e_i, d_i) for each entity.**
- **Now assume Alice needs to send a message to Bob. The ciphertext to Bob is $C = P^{eB} \bmod n$. Bob uses his private exponent, d_B, to decrypt his message, $P = C^{dB} \bmod n$.**
- **The problem is that if Eve is a member of the community and has been assigned a pair of exponents (eE and dE), Using her own exponents (eE and dE), Eve can launch a probabilistic attack to factor n and find Bob's dB.**
- **To thwart this type of attack, the modulus must not be shared. Each entity needs to calculate her or his own modulus.**

3.3.3 Continued

Timing Attack

- **Paul Kocher elegantly demonstrated a ciphertext-only attack, called the timing attack.**
- **The attack is based on the fast-exponential algorithm.**
- **The algorithm uses only squaring if the corresponding bit in the private exponent d is 0; it uses both squaring and multiplication if the corresponding bit is 1.**
- **Assume that Eve has intercepted a large number of ciphertexts, C1 to Cm.**
- **Also assume that Eve has observed how long it takes for Bob to decrypt each ciphertext, T1 to Tm. Eve, who knows how long it takes for the underlying hardware to calculate a multiplication operation, calculated t1 to tm, where ti is the time required to calculate the multiplication operation Result = Result × Ci mod n.**

RSA- Timing attack

$c = 3$

$d = (13)_{10} = (1101)_2$ ($d_3 d_2 d_1 d_0$)

$n = 17$

$p = 1$

For $d_3 = 1$ [Consider bits of d from left to right]

$p = p^2 = 1^2 = 1$

$p = p \times c = 1 \times 3 = 3$

For $d_2 = 1$

$p = p^2 = 3^2 = 9$

$p = p \times c = 9 \times 3 = 27$

$p = p \bmod 17 = 10$

For $d_1 = 0$

$p = p^2 = 10^2 = 100$ no multiplication.

$p = p \bmod 17 = 15$

For $d_0 = 1$

$p = p^2 = 15^2 = 225$

$p = p \bmod 17 = 4$

$p = p \times c = 4 \times 3 = 12$

$\therefore p = c^{13} \bmod 17 = 12$

Algorithm 10.5 *Timing attack on RSA*

```
RSA_Timing_Attack ([T_1 ... T_m])
{
    d_0 ← 1                                          // Because d is odd
    Calculate [t_1 ... t_m]
    [T_1 ... T_m] ← [T_1 ... T_m] − [t_1 ... t_m]    // Update T_i for the next bit
    for (j from 1 to k − 1)
    {
        Recalculate [t_1 ... t_m]                    // Recalculate t_i assuming the next bit is 1
        [D_1 ... D_m] ← [T_1 ... T_m] − [t_1 ... t_m]
        var ← variance ([D_1 ... D_m]) − variance ([T_1 ... T_m])
        if (var > 0)  d_j ← 1      else  d_j ← 0
        [T_1 ... T_m] ← [T_1 ... T_m] − d_j × [t_1 ... t_m]   // Update T_i for the next bit
    }
}
```

3.3.3 Continued

Timing Attack

There are two methods to thwart timing attack:

1. **Add random delays to the exponentiations** to make each exponentiation take the same amount of time.
2. Rivest recommended **blinding.** The idea is to multiply the ciphertext by a random number before decryption.

13-4 ATTACKS ON DIGITAL SIGNATURE

This section describes some attacks on digital signatures and defines the types of forgery

13.4.2 Forgery Types

Existential Forgery:
In an existential forgery, Eve may be able to create a valid message-signature pair, but not one that she can really use. In other words, a document has been forged, but the content is randomly calculated. This type of forgery is probable, but fortunately Eve cannot benefit from it very much. Her message could be syntactically or semantically unintelligible.

Selective Forgery:
In selective forgery, Eve may be able to forge Alice's signature on a message with the content selectively chosen by Eve. Although this is beneficial to Eve, and may be very detrimental to Alice, the probability of such forgery is low, but not negligible.

10.1.5 Knapsack Cryptosystem

Definition

$a = [a_1, a_2, \ldots, a_k]$ and $x = [x_1, x_2, \ldots, x_k]$.

$$s = knapsackSum\,(a, x) = x_1a_1 + x_2a_2 + \cdots + x_ka_k$$

Given a and x, it is easy to calculate s. However, given s and a it is difficult to find x.

Superincreasing Tuple

$$a_i \geq a_1 + a_2 + \ldots + a_{i-1}$$

10.1.5 Continued

Algorithm 10.1 *knapsacksum and inv_knapsackSum for a superincreasing k-tuple*

```
knapsackSum (x [1 ... k], a [1 ... k])
{
    s ← 0
    for (i = 1 to k)
    {
      s ← s + a_i × x_i
    }
    return s
}
```

```
inv_knapsackSum (s, a [1 ... k])
{
    for (i = k down to 1)
    {
        if s ≥ a_i
        {
            x_i ← 1
            s ← s − a_i
        }
        else x_i ← 0
    }
    return x [1 ... k]
}
```

10.1.5 Continued

Example 10. 3

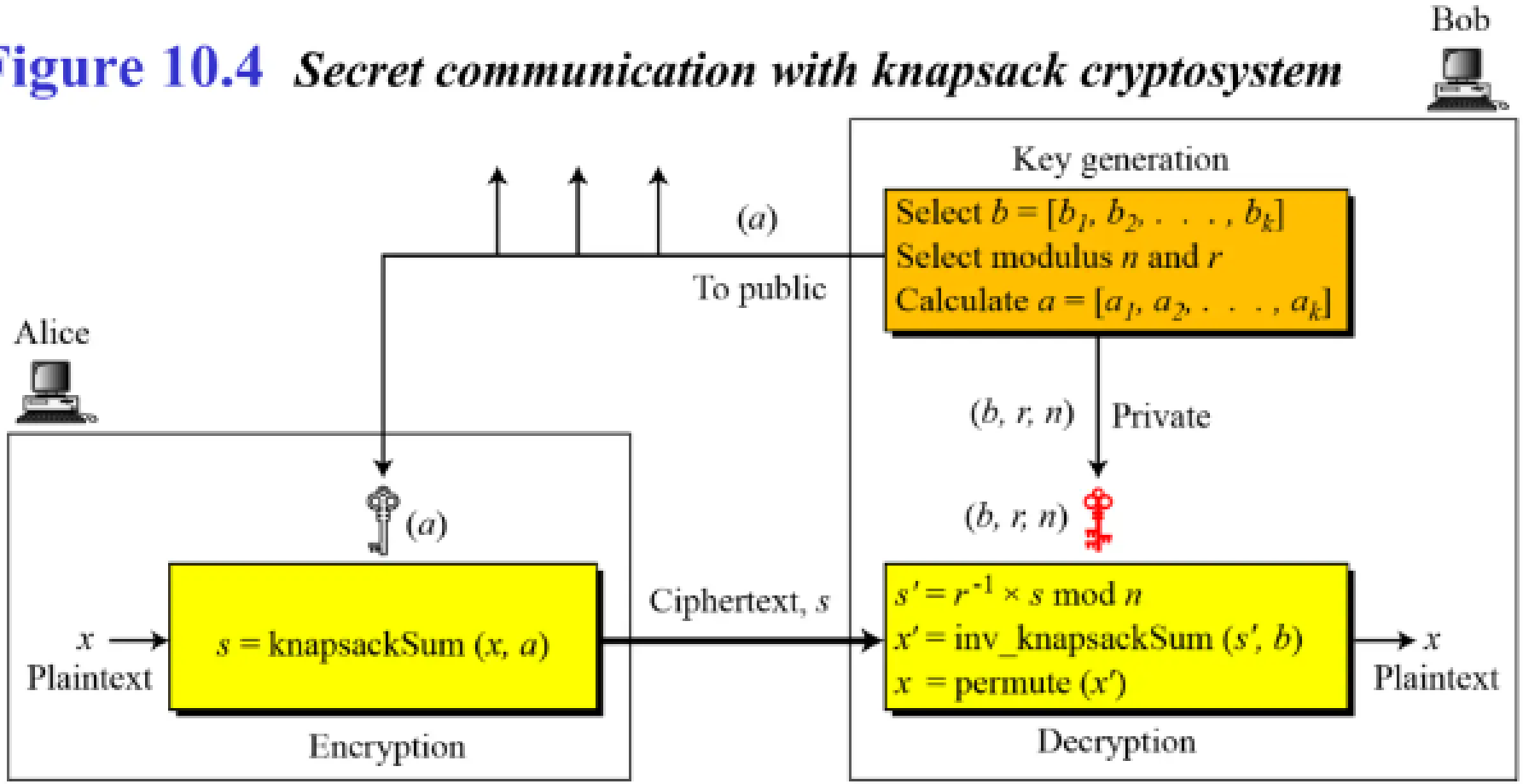
As a very trivial example, assume that a = [17, 25, 46, 94, 201,400] and s = 272 are given. Table 10.1 shows how the tuple x is found using inv_knapsackSum routine in Algorithm 10.1. In this case x = [0, 1, 1, 0, 1, 0], which means that 25, 46, and 201 are in the knapsack.

Table 10.1 *Values of i, a_i, s, and x_i in Example 10.3*

i	a_i	s	$s \geq a_i$	x_i	$s \leftarrow s - a_i \times x_i$
6	400	272	false	$x_6 = 0$	272
5	201	272	true	$x_5 = 1$	71
4	94	71	false	$x_4 = 0$	71
3	46	71	true	$x_3 = 1$	25
2	25	25	true	$x_2 = 1$	0
1	17	0	false	$x_1 = 0$	0

Secret Communication with Knapsacks.

Figure 10.4 *Secret communication with knapsack cryptosystem*



Key Generation

a. Create a superincreasing k-tuple $b = [b_1, b_2, \ldots, b_k]$

b. Choose a modulus n, such that $n > b_1 + b_2 + \cdots + b_k$

c. Select a random integer r that is relatively prime with n and $1 \leq r \leq n - 1$.

d. Create a temporary k-tuple $t = [t_1, t_2, \ldots, t_k]$ in which $t_i = r \times b_i \bmod n$.

e. Select a permutation of k objects and find a new tuple $a = permute\ (t)$.

f. The public key is the k-tuple a. The private key is n, r, and the k-tuple b.

Encryption

Suppose Alice needs to send a message to Bob.

a. Alice converts her message to a k-tuple $x = [x_1, x_2, \ldots, x_k]$ in which x_i is either 0 or 1. The tuple x is the plaintext.
b. Alice uses the *knapsackSum* routine to calculate s. She then sends the value of s as the ciphertext.

Decryption

Bob receives the ciphertext s.

a. Bob calculates $s' = r^{-1} \times s \bmod n$.
b. Bob uses *inv_knapsackSum* to create x'.
c. Bob permutes x' to find x. The tuple x is the recovered plaintext.

Example 10.4

This is a trivial (very insecure) example just to show the procedure.

1. Key generation:
   a. Bob creates the superincreasing tuple $b = [7, 11, 19, 39, 79, 157, 313]$.
   b. Bob chooses the modulus $n = 900$ and $r = 37$, and [4 2 5 3 1 7 6] as permutation table.
   c. Bob now calculates the tuple $t = [259, 407, 703, 543, 223, 409, 781]$.
   d. Bob calculates the tuple a = permute $(t) = [543, 407, 223, 703, 259, 781, 409]$.
   e. Bob publicly announces a; he keeps n, r, and b secret.

2. Suppose Alice wants to send a single character "g" to Bob.
   a. She uses the 7-bit ASCII representation of "g", $(1100111)_2$, and creates the tuple $x = [1, 1, 0, 0, 1, 1, 1]$. This is the plaintext.
   b. Alice calculates $s = knapsackSum\ (a, x) = $ ~~2165~~ 2399. This is the ciphertext sent to Bob.
3. Bob can decrypt the ciphertext, $s =$ ~~2165~~ 2399
   a. Bob calculates $s' = s \times r^{-1} \bmod n =$ ~~2165~~ $\times\ 37^{-1} \bmod 900 = 527$.
   b. Bob calculates $x' = Inv\_knapsackSum\ (s', b) = [1, 1, 0, 1, 0, 1, 1]$.
   c. Bob calculates $x = permute\ (x') = [1, 1, 0, 0, 1, 1, 1]$. He interprets the string $(1100111)_2$ as the character "g".

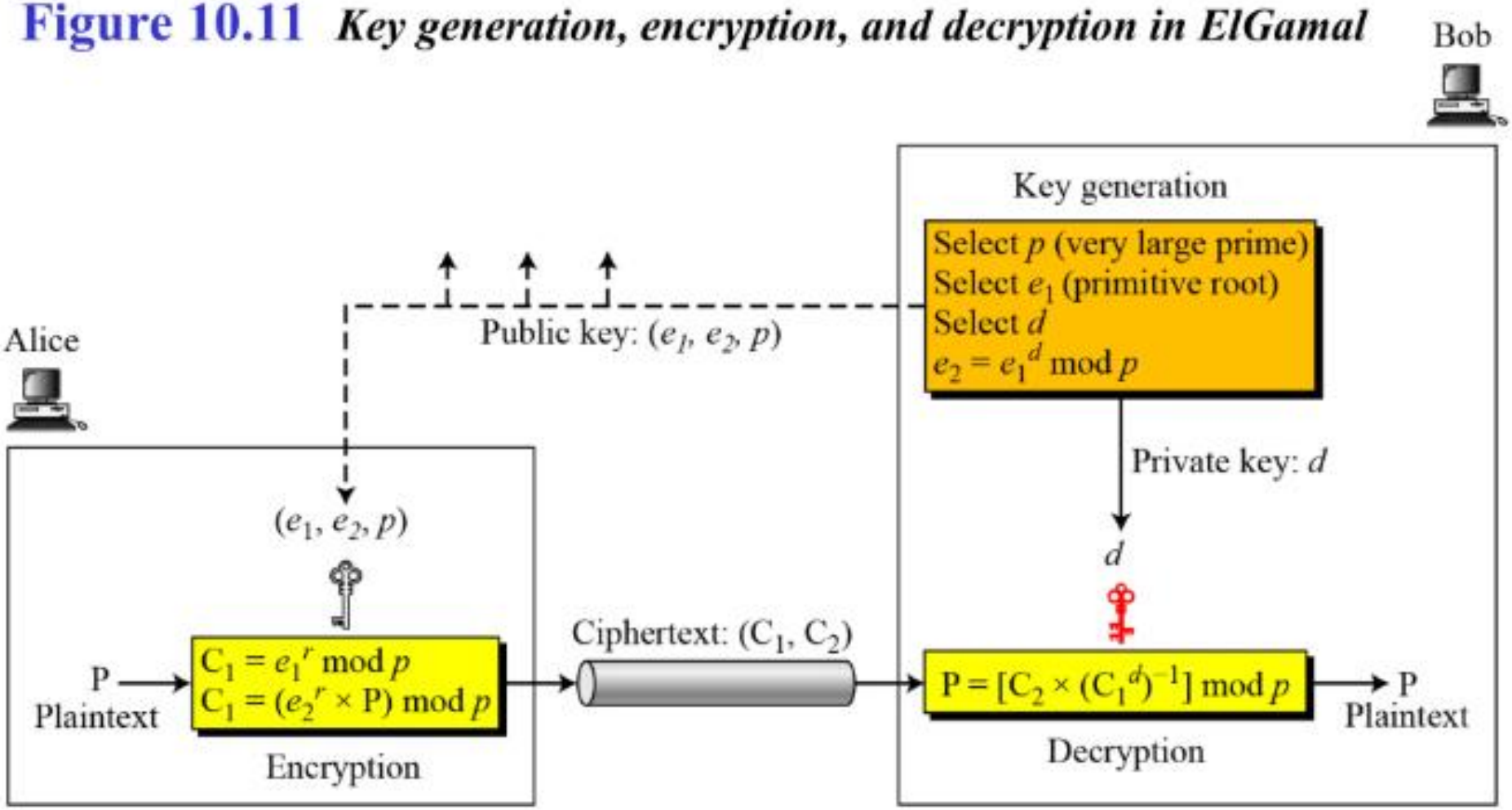
10-4 ELGAMAL CRYPTOSYSTEM

Besides RSA and Rabin, another public-key cryptosystem is ElGamal. ElGamal is based on the discrete logarithm problem discussed in Chapter 9.

Topics discussed in this section:

10.4.2 Procedure

Figure 10.11 *Key generation, encryption, and decryption in ElGamal*



Key Generation

Algorithm 10.9 *ElGamal key generation*

ElGamal_Key_Generation

{

Select a large prime p

Select d to be a member of the group $\mathbf{G} = < \mathbf{Z}_p^*, \times >$ such that $1 \leq d \leq p - 2$

Select e_1 to be a primitive root in the group $\mathbf{G} = < \mathbf{Z}_p^*, \times >$

$e_2 \leftarrow e_1^{\,d} \bmod p$

Public_key $\leftarrow (e_1, e_2, p)$ // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

10.4.2 Continued

Algorithm 10.10 *ElGamal encryption*

```
ElGamal_Encryption (e1, e2, p, P)          // P is the plaintext
{
    Select a random integer r in the group G = < Zp*, × >
    C1 ← e1^r mod p
    C2 ← (P × e2^r) mod p                  // C1 and C2 are the ciphertexts
    return C1 and C2
}
```

10.4.2 Continued

Algorithm 10.11 *ElGamal decryption*

```
ElGamal_Decryption (d, p, C1, C2)        // C1 and C2 are the ciphertexts
{
    P ← [C2 (C1^d)^-1] mod p             // P is the plaintext
    return P
}
```

Proof

The ElGamal decryption expression $C_2 \times (C_1^{\,d})^{-1}$ can be verified to be P through substitution:

$$[C_2 \times (C_1^{\,d})^{-1}] \bmod p = [(e_2^{\,r} \times P) \times (e_1^{\,nd})^{-1}] \bmod p = (e_1^{\,dr}) \times P \times (e_1^{\,rd})^{-1} = P$$

The bit-operation complexity of encryption or decryption in ElGamal cryptosystem is polynomial.

Example 10. 10

Here is a trivial example. Bob chooses p = 11 and $e_1 = 2$. and d = 3 $e_2 = e_1^d = 8$. So the public keys are (2, 8, 11) and the private key is 3. Alice chooses r = 4 and calculates C1 and C2 for the plaintext 7.

Plaintext: 7

$C_1 = e_1{}^r \bmod 11 = 16 \bmod 11 = 5 \bmod 11$

$C_2 = (P \times e_2{}^r) \bmod 11 = (7 \times 4096) \bmod 11 = 6 \bmod 11$

Ciphertext: (5, 6)

Bob receives the ciphertexts (5 and 6) and calculates the plaintext.

$$[C_2 \times (C_1{}^d)^{-1}] \bmod 11 = 6 \times (5^3)^{-1} \bmod 11 = 6 \times 3 \bmod 11 = 7 \bmod 11$$

Plaintext: 7

10.4.3 Continued

Example 10. 11

Instead of using $P = [C_2 \times (C_1^d)^{-1}] \bmod p$ for decryption, we can avoid the calculation of multiplicative inverse and use $P = [C_2 \times C_1^{p-1-d}] \bmod p$ (see Fermat's little theorem in Chapter 9). In Example 10.10, we can calculate $P = [6 \times 5^{11-1-3}] \bmod 11 = 7 \bmod 11$.

Note

For the ElGamal cryptosystem, *p* must be at least 300 digits and *r* must be new for each encipherment.

10.4.3 Continued

Example 10. 12

Bob uses a random integer of 512 bits. The integer p is a 155-digit number (the ideal is 300 digits). Bob then chooses e_1, d, and calculates e_2, as shown below:

$p =$	11534899272561676244925313717014331740490094532609834959814346921905689869862264593212975473787189514436889176526473093615929993728061165964347353440008577
$e_1 =$	2
$d =$	1007
$e_2 =$	97886413043009189508766856938097739043880062887337687610022062233255450707415618921231831770461014167336015088413294085724853770315820660100725587074 55

10.4.3 Continued

Example 10. 10

Alice has the plaintext P = 3200 to send to Bob. She chooses r = 545131, calculates C1 and C2, and sends them to Bob.

P =	3200
***r* =**	545131
C_1 =	887297069383528471022570471492275663120260067256562125018818835142941722359971268111410536366170517305158153318916540097373635508029573678856906061915288 1
C_2 =	708454333048929944577016012380794999567436021836192446961774506921244696155165800779455593080345889614402408599525919579209721628879681350582779566430295 0

Bob calculates the plaintext $P = C_2 \times ((C_1)^d)^{-1} \bmod p = 3200 \bmod p$.

P =	3200

Strong Prime (Gordon Method)

A strong prime number (p) is
when r, s and t satisfy the following:

- ***(p−1) has a large prime factor of r.***
- ***(p+1) has a large prime factor of s.***
- ***(r−1) has a large prime factor of t.***

Gordon method of generating strong prime

First we generate two random prime numbers (s and t).
Next we select an integer (i_0) and find the first prime number for:

$$r = 2it + 1$$

and where $i = i_0, i_0 + 1$, etc, until we find a prime number for r. Next we calculate:

$$p_0 = 2(s^{r-2} \pmod{r})s - 1$$

We then select j_0 and find the prime:

$$p = p_0 + 2jrs$$

and where $j = j_0, j_0 + 1$, etc, until we find a prime number for p. The results in a strong prime (p).

```
...
s=149
t=233
r=46601

p_0=2702859
p=474864191

Factor for (p-1) - should be r: 46601
Factor for (p+1) - should be s: 149
Factor for (r-1) - should be t: 233
```

Strong prime vs Safe Prime (Notes)