CHAPTER 1

Introduction

(Solution to Practice Set)

Review Questions

1. The three security goals are *confidentiality*, *integrity*, and *availability*.

- ❑ *Confidentiality* means protecting confidential information.
- ❑ *Integrity* means that changes to the information need to be done only by authorized entities.
- ❑ *Availability* means that information needs to be available to authorized entities.

2.

- ❑ In a *passive attack*, the attacker's goal is just to obtain information. This means that the attack does not modify data or harm the system. Examples of passive attacks are snooping and traffic analysis.
- ❑ An *active attack* may change the data or harm the system. Attacks that threaten the integrity and availability are active attacks. Examples of active attacks are modification, masquerading, replaying, repudiation, and denial of service.

3. We mentioned five security services: *data confidentiality*, *data integrity*, *authentication*, *nonrepudiation*, and *access control*.

- ❑ *Data confidentiality* is to protect data from disclosure attack.
- ❑ *Data integrity* is to protect data from modification, insertion, deletion, and replaying.
- ❑ *Authentication* means to identify and authenticate the party at the other end of the line.
- ❑ *Nonrepudiation* protects against repudiation by either the sender or the receiver of the data.
- ❑ *Access control* provides protection against unauthorized access to data.

4. Eight security mechanisms were discussed in this chapter. *encipherment*, *data integrity*, *digital signature*, *authentication exchange*, *traffic padding*, *routing control*, *notarization*, and *access control*.

   - *Encipherment* provides confidentiality.
   - The *data integrity* mechanism appends a short checkvalue to the data. The checkvalue is created by a specific process from the data itself.
   - A *digital signature* is a means by which the sender can electronically sign the data and the receiver can electronically verify the signature.
   - In *authentication exchange*, two entities exchange some messages to prove their identity to each other.
   - *Traffic padding* means inserting some bogus data into the data traffic to thwart the adversary's attempt to use the traffic analysis.
   - *Routing control* means selecting and continuously changing different available routes between the sender and the receiver to prevent the opponent from eavesdropping on a particular route.
   - *Notarization* means selecting a third trusted party to control the communication between two entities.
   - *Access control* uses methods to prove that a user has access right to the data or resources owned by a system.

5.

   - *Cryptography*, a word with origin in Greek, means "secret writing." We used the term to refer to the science and art of transforming messages to make them secure and immune to attacks.
   - *Steganography*, a word with origin in Greek, means "covered writing." Steganography refers to concealing the message itself by covering it with something else.

Exercises

6.

   a. A *regular mail* guarantees no security services. It is the *best-effort* delivery service. The mail can be lost, altered in the mail, opened by somebody other than the intended recipient.

   b. A *regular mail with delivery confirmation* can only show that the mail has been delivered. This can only give peace of mind to the sender that the packet is not lost. However, since there is no signature from the recipient, it does not guarantee any of the security services.

   c. A *regular mail with delivery confirmation and the recipient signature* can provide nonrepudiation service only at the mail level, not the contents level. In other words, the recipient of the mail cannot deny that she has not received the mail, but she can deny that the mail contained some specific information. For example, if Alice sends a mail with $100 cash inside to Bob via this type of mail, Bob cannot deny that he has received the mail, but he can deny that the mail contained some cash inside. In some cases, the sender is an authority and it is enough that Bob accepts he has received the mail. In this case, if there is a dispute, the court accept the testimony of the sender about the contents.

d. A *certified mail* is actually the same as the *regular mail with delivery confirmation and the recipient signature.*

e. A mail can be insured. However, this is not security in the sense we are talking in this chapter. Secured mail can only provide compensation if the mail is lost.

f. A *registered mail* is different from all of the previous delivery methods. A registered mail is carried by the post office under the tight security. This means that the confidentiality and integrity of the mail is guaranteed. Since a registered mail normally includes the signed receipt, the nonrepudiation is also guaranteed. However, nonrepudiation is only at the mail level, not the content level. The recipient of the registered mail cannot deny that the mail has been delivered, but it can deny that it contained a special message or an item of some value.

7.

a. This is *snooping* (attack to the confidentiality of stored data). Although the contents of the test is not confidential on the day of the test, it is confidential before the test day.

b. This is modification (attack to the integrity of data). The value of the check is changed (from $10 to $100).

c. This is *denial of service* (attack to availability). Sending so many e-mails may crash the server and the service may be interrupted.

8.

a. This provides *access control* mechanism. The process is to prove that the student has right to access the school resources.

b. This can provide *routing control*. The school may be doing this to prevent a student from eavesdropping on a particular route.

c. This can be *authentication exchange* mechanism. The professor needs to authenticate the student before sending the grade. The preassigned identification is a secret between the student and the professor.

d. The mechanism is similar to *digital signature*. It can be used for two purposes. If the signature of the customer is checked against a signature on the file, it can provide authentication. The signature on the withdrawal document definitely is served as the nonrepudiation. The customer cannot later denies that she has not received the cash.

9.

a. This is *steganography*. The answers to the test has not been changed; they have been only hidden.

b. This is *cryptography*. The characters in the message are not hidden; they are replaced by another characters.

c. This is *steganography*. The special ink hides the actual writing on the check.

d. This is *steganography*. The water marks hides the actual contents of the thesis.

10. A signature on a document is like a *digital signature* on a message. It protects the integrity of the document, it provides authentication, and it protects non-repudiation.

CHAPTER 2

Mathematics of Cryptography Part I

(Solution to Practice Set)

Review Questions

1. The set of integers is **Z**. It contains all integral numbers from negative infinity to positive infinity. The set of residues modulo n is $\mathbf{Z}_n$. It contains integers from 0 to $n - 1$. The set **Z** has non-negative (positive and zero) and negative integers; the set $\mathbf{Z}_n$ has only non-negative integers. To map a nonnegative integer from **Z** to $\mathbf{Z}_n$, we need to divide the integer by n and use the remainder; to map a negative integer from **Z** to $\mathbf{Z}_n$, we need to repeatedly add n to the integer to move it to the range 0 to $n - 1$.

2. We mentioned four properties:

- **Property 1:** if $a \mid 1$, then $a = \pm 1$.
- **Property 2:** if $a \mid b$ and $b \mid a$, then $a = \pm b$.
- **Property 3:** if $a \mid b$ and $b \mid c$, then $a \mid c$.
- **Property 4:** if $a \mid b$ and $a \mid c$, then $a \mid (m \times b + n \times c)$, where m and n are arbitrary integers.

3. The number 1 is an integer with only one divisor, itself. A prime has only two divisors: 1 and itself. For example, the prime 7 has only two divisor 7 and 1. A composite has more than two divisors. For example, the composite 42 has several divisors: 1, 2, 3, 6, 7, 14, 21, and 42.

4. The greatest common divisor of two positive integers, **gcd (*a*, *b*)**, is the largest positive integer that divides both a and b. The ***Euclidean algorithm*** can find the greatest common divisor of two positive integers.

5. A linear Diophantine equation of two variables is of the form $ax + by = c$. We need to find integer values for x and y that satisfy the equation. This type of equation has either no solution or an infinite number of solutions. Let $d = \gcd(a, b)$. If d does not divide c then the equation have no solitons. If d divides c, then we have an infinite number of solutions. One of them is called the particular solution; the rest, are called the general solutions.

6. The modulo operator takes an integer a from the set **Z** and a positive modulus n. The operator creates a nonnegative residue, which is the remainder of dividing a by n. We mentioned three properties for the modulo operator:

   - **First:** $(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$
   - **Second:** $(a - b) \bmod n = [(a \bmod n) - (b \bmod n)] \bmod n$
   - **Third:** $(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$

7. A residue class $[a]$ is the set of integers congruent modulo n. It is the set of all integers such that $x = a \pmod{n}$. In each set, there is one element called the least (nonnegative) residue. The set of all of these least residues is $\mathbf{Z}_n$.

8. The set $\mathbf{Z}_n$ is the set of all positive integer between 0 and $n - 1$. The set $\mathbf{Z}_n*$ is the set of all integers between 0 and $n - 1$ that are relatively prime to n. Each element in $\mathbf{Z}_n$ has an additive inverse; each element in $\mathbf{Z}_n*$ has a multiplicative inverse. The extended Euclidean algorithm is used to find the multiplicative inverses in $\mathbf{Z}_n*$.

9. A matrix is a rectangular array of $l \times m$ elements, in which l is the number of rows and m is the number of columns. If a matrix has only one row ($l = 1$), it is called a row matrix; if it has only one column ($m = 1$), it is called a column matrix. A square matrix is a matrix with the same number of rows and columns ($l = m$). The determinant of a square matrix **A** is a scalar defined in linear algebra. The multiplicative inverse of a square matrix exists only if its determinant has a multiplicative inverse in the corresponding set.

10. A linear equation is an equation in which the power of each variable is 1. A linear congruence equation is a linear equation in which calculations are done modulo n. An equation of type $ax = b \pmod{n}$ can be solved by finding the multiplicative inverse of a. A set of linear equations can be solved by finding the multiplicative inverse of a matrix.

Exercises

11.

   a. It is false because $26 = 2 \times 13$.

   b. It is true because $123 = 3 \times 41$.

   c. It is true because 127 is a prime.

   d. It is true because $21 = 3 \times 7$.

   e. It is false because $96 = 2^5 \times 3$.

   f. It is false because 8 is greater than 5.

12.

a. gcd (88, 220) = 44, as shown in the following table:

q	r_1	r_2	r
0	88	220	88
2	220	88	44
2	88	44	0
	44	**0**	

b. gcd (300, 42) = 6, as shown in the following table:

q	r_1	r_2	r
7	300	42	6
7	42	6	0
	6	0	

c. gcd (24, 320) = 8, as shown in the following table:

q	r_1	r_2	r
0	24	320	24
13	320	24	8
3	24	8	0
	8	**0**	

d. gcd (401, 700) = 1 (coprime), as shown in the following table:

q	r_1	r_2	r
0	401	700	401
1	700	401	299
1	401	299	102
2	299	102	95
1	102	95	7
13	95	7	4
1	7	4	3
1	4	3	1
3	3	1	0
	1	0	

13.

a. gcd (a, b, 16) = gcd (gcd (a, b), 16) = gcd (24, 16) = 8

b. gcd (a, b, c, 16) = gcd (gcd (a, b, c), 16) = gcd (12, 16) = 4

c. gcd (200, 180, 450) = gcd (gcd (200, 180), 450) = gcd (20, 450) = 10

d. gcd (200, 180, 450, 600) = gcd (gcd (200, 180, 450), 600) = gcd (10, 600) = 10

14.

a. gcd $(2n + 1, n)$ = gcd $(n, 1) = 1$

b.

gcd (201, 100) = gcd $(2 \times 100 + 1, 100) = 1$

gcd (81, 40) = gcd $(2 \times 40 + 1, 40) = 1$

gcd (501, 250) = gcd $(2 \times 250 + 1, 250) = 1$

15.

a. gcd $(3n + 1, 2n + 1)$ = gcd $(2n + 1, n) = 1$

b.

gcd (301, 201) = gcd $(3 \times 100 + 1, 2 \times 100 + 1) = 1$
gcd (121, 81) = gcd $(3 \times 40 + 1, 2 \times 40 + 1) = 1$

16.

a. We use the following table:

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
0	4	7	4	1	0	1	0	1	0
1	7	4	3	0	1	−1	1	0	1
1	4	3	1	1	−1	2	0	1	−1
3	3	1	0	−1	2	−7	1	−1	4
	1	0		2	−7		−1	4	
	↑ gcd			↑ s			↑ t		

gcd (4, 7) = 1 $\rightarrow$ (4)(2) + (7)(−1) = 1

b. We use the following table:

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
6	291	42	39	1	0	1	0	1	−6
1	42	39	3	0	1	−1	1	−6	7
13	39	3	0	1	−1	14	−6	7	−97
	3	0		−1	14		7	−97	
	↑ gcd			↑ s			↑ t		

gcd (291, 42) = 3 $\rightarrow$ (291)(−1) + (42)(7) = 3

c. We use the following table:

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
0	84	320	84	1	0	1	0	1	0
3	320	84	68	0	1	−3	1	0	1
1	84	68	16	1	−3	−4	0	1	−1
4	68	16	4	−3	4	−19	1	−1	5
4	16	4	0	4	−19	80	−1	5	−21
	4	0		−19	80		5	−21	
	↑			↑			↑		
	gcd			*s*			t		

gcd (84, 320) = 4 $\rightarrow$ (84)(−19) + (320)(5) = 4

d. We use the following table:

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
6	400	60	40	1	0	1	0	1	−6
1	60	40	20	0	1	−1	1	−6	7
2	40	20	0	1	−1	3	−6	7	−20
	20	0		−1	4		7	−20	
	↑			↑			↑		
	gcd			*s*			t		

gcd (400, 60) = 20 $\rightarrow$ (400)(−1) + (60)(7) = 20

17.

a. 22 mod 7 = 1

b. 291 mod 42 = 39

c. 84 mod 320 = 84

d. 400 mod 60 = 40

18.

a. (273 + 147) mod 10 = (273 mod 10 + 147 mod 10) mod 10 = (3 + 7) mod 10 = 0 mod 10

b. (4223 + 17323) mod 10 = (4223 mod 10 + 17323 mod 10) mod 10 = (3 + 3) mod 10 = 6 mod 10

c. (148 + 14432) mod 12 = (148 mod 12 + 14432 mod 12) mod 12 = (4 + 8) mod 12 = 0 mod 12

d. (2467 + 461) mod 12 = (2467 mod 12 + 461 mod 12) mod 12 = (7 + 5) mod 12 = 0 mod 12

19.

a. $(125 \times 45) \bmod 10 = (125 \bmod 10 \times 45 \bmod 10) \bmod 10 = (5 \times 5) \bmod 10$
$= 5 \bmod 10$

b. $(424 \times 32) \bmod 10 = (424 \bmod 10 \times 32 \bmod 10) \bmod 10 = (4 \times 2) \bmod 10$
$= 8 \bmod 10$

c. $(144 \times 34) \bmod 10 = (144 \bmod 10 \times 34 \bmod 10) \bmod 10 = (4 \times 4) \bmod 10$
$= 6 \bmod 10$

d. $(221 \times 23) \bmod 10 = (221 \bmod 10 \times 23 \bmod 10) \bmod 10 = (1 \times 3) \bmod 10$
$= 3 \bmod 10$

20.

a. $a \bmod 10 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 10$
$= [(a_n \times 10^n) \bmod 10 + \ldots + (a_1 \times 10^1) \bmod 10 + a_0 \bmod 10] \bmod 10$
$= [0 + \ldots + 0 + a_0 \bmod 10] = \boldsymbol{a_0 \bmod 10}$

b. $a \bmod 100 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 10$
$= [(a_n \times 10^n) \bmod 100 + \ldots + (a_1 \times 10^1) \bmod 100 + a_0 \bmod 10] \bmod 10$
$= [0 + \ldots + 0 + (a_1 \times 10^1) \bmod 100 + a_0 \bmod 100]$
$= (a_1 \times 10^1) \bmod 100 + a_0 \bmod 100 = \boldsymbol{[a_1 \times 10^1 + a_0] \bmod 100}$.

c. Similarly $a \bmod 1000 = \boldsymbol{[a_2 \times 10^2 + a_1 \times 10^1 + a_0] \bmod 1000}$.

21. $a \bmod 5 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 5$
$= [(a_n \times 10^n) \bmod 5 + \ldots + (a_1 \times 10^1) \bmod 5 + a_0 \bmod 5] \bmod 5$
$= [0 + \ldots + 0 + a_0 \bmod 5] = \boldsymbol{a_0 \bmod 5}$

22. $a \bmod 2 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 2$
$= [(a_n \times 10^n) \bmod 2 + \ldots + (a_1 \times 10^1) \bmod 2 + a_0 \bmod 2] \bmod 2$
$= [0 + \ldots + 0 + a_0 \bmod 2] = \boldsymbol{a_0 \bmod 2}$

23. $a \bmod 4 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 4$
$= [(a_n \times 10^n) \bmod 4 + \ldots + (a_1 \times 10^1) \bmod 4 + a_0 \bmod 4] \bmod 4$
$= [0 + \ldots + 0 + (a_1 \times 10^1) \bmod 4 + a_0 \bmod 4] = \boldsymbol{(a_1 \times 10^1 + a_0) \bmod 4}$

24. $a \bmod 8 = (a_n \times 10^n + \ldots + a_2 \times 10^2 + a_1 \times 10^1 + a_0) \bmod 8$
$= [(a_n \times 10^n) \bmod 8 + \ldots + (a_2 \times 10^2) \bmod 8 + (a_1 \times 10^1) \bmod 8$
$+ a_0 \bmod 8] \bmod 8$
$= [0 + \ldots + 0 + (a_1 \times 10^2) \bmod 8 + (a_1 \times 10^1) \bmod 8 + a_0 \bmod 8]$
$= \boldsymbol{(a_2 \times 10^2 + a_1 \times 10^1 + a_0) \bmod 4}$

25. $a \bmod 9 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 9$
$= [(a_n \times 10^n) \bmod 9 + \ldots + (a_1 \times 10^1) \bmod 9 + a_0 \bmod 9] \bmod 9$
$= \boldsymbol{(a_n + \ldots + a_1 + a_0\ ) \bmod 9}$

26. $a \bmod 7 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 7$
$= [(a_n \times 10^n) \bmod 7 + \ldots + (a_1 \times 10^1) \bmod 7 + a_0 \bmod 7] \bmod 7$
$= \ldots + \boldsymbol{a_5} \times (-2) + \boldsymbol{a_4} \times (-3) + \boldsymbol{a_3} \times (-1) + \boldsymbol{a_2} \times (2) + \boldsymbol{a_1} \times (3) + \boldsymbol{a_0} \times (1)$**] mod 7**

For example, $631453672 \bmod 13 =$ **[**$(-1)6 + (2)3 + (1)1 + (-2)4 + (-3)5 + (-1)3$
$+ (2)6 + (3)7 + (1)2$**] mod 7** $= 3 \bmod 7$

27. $a \bmod 11 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 11$
$= [(a_n \times 10^n) \bmod 11 + \ldots + (a_1 \times 10^1) \bmod 11 + a_0 \bmod 11] \bmod 11$
$= \ldots + \boldsymbol{a_3} \times (-1) + \boldsymbol{a_2} \times (1) + \boldsymbol{a_1} \times (-1) + \boldsymbol{a_0} \times (1)]$ **mod 11**

For example, 631453672 mod 11 = [(1)6 + (−1)3 + (1)1 + (−1)4 + (1)5 + (−1)3 + (1)6 + (−1)7 + (1)2] mod 11 = −8 mod 11 = 5 mod 11

28. $a \bmod 13 = (a_n \times 10^n + \ldots + a_1 \times 10^1 + a_0) \bmod 13$
$= [(a_n \times 10^n) \bmod 13 + \ldots + (a_1 \times 10^1) \bmod 13 + a_0 \bmod 13] \bmod 13$
$= \ldots + \boldsymbol{a_5} \times (4) + \boldsymbol{a_4} \times (3) + \boldsymbol{a_3} \times (-1) + \boldsymbol{a_2} \times (-4) + \boldsymbol{a_1} \times (-3) + \boldsymbol{a_0} \times (1)]$ **mod 13**

For example, 631453672 mod 13 = [(−4)6 + (−3)3 + (1)1 + (4)4 + (3)5 + (−1)3 + (−4)6 + (−3)7 + (1)2] mod 13 = 3 mod 13

29.

a. (A + N) mod 26 = (0 + 13) mod 26 = 13 mod 26 = **N**

b. (A + 6) mod 26 = (0 + 6) mod 26 = 6 mod 26 = **G**

c. (Y − 5) mod 26 = (24 −5) mod 26 = 19 mod 26 = **T**

d. (C − 10) mod 26 = (2 −10) mod 26 = −8 mod 26 = 18 mod 26 = **S**

30. (0, 0), (1, 19), (2, 18), (3, 17), (4, 16), (5, 15), (6, 14), (7, 13), (8, 12), (9, 11), (10, 10)

31. (1, 1), (3, 7), (9, 9), (11, 11), (13, 17), (19, 19)

32.

a. We use the following table:

q	*r₁*	*r₂*	*r*	*t₁*	*t₂*	*t*
4	180	38	28	0	1	−4
1	38	28	10	1	−4	5
2	28	10	8	−4	5	−14
1	10	8	2	5	−14	19
4	8	2	0	−14	19	90
	2	0		19		
	gcd			***t***		

gcd (180, 38) = 2 ≠ 1 → 38 has no inverse in $\mathbf{Z}_{180}$.

b. We use the following table:

q	r_1	r_2	r	t_1	t_2	t
25	180	7	5	0	1	
1	7	5	2	1	–25	
2	5	2	1	–25	26	
2	2	1	0	26	–77	
	1	0		–77	180	
	gcd			***t***		

gcd (180, 7) = 1 → 7^{-1} mod 180 = –77 mod 180 = 103 mod 180.

c. We use the following table:

q	r_1	r_2	r	t_1	t_2	t
1	180	132	48	0	1	–1
2	132	48	36	1	–1	3
1	48	36	12	–1	3	–4
3	36	12	0	3	–4	15
	12	0		–4	15	
	gcd			***t***		

gcd (180, 132) = 12 ≠ 1 → 132 has no inverse in $\mathbf{Z}_{180}$.

d. We use the following table:

q	r_1	r_2	r	t_1	t_2	t
7	180	24	12	0	1	–7
2	24	12	0	1	–7	15
	12	0		–7	15	
	gcd			***t***		

e. gcd (180, 24) = 12 ≠ 1 → 24 has no inverse in $\mathbf{Z}_{180}$.

33.

a. We have $a = 25$, $b = 10$ and $c = 15$. Since $d = \gcd(a, b) = 5$ divides c, there is an infinite number of solutions. The reduced equation is $5x + 2y = 3$. We solve the equation $5s + 2t = 1$ using the extended Euclidean algorithm to get $s = 1$ and $t = -2$. The particular and general solutions are

Particular: $x_0 = (c/d) \times s = 3$ $y_0 = (c/d) \times t = -6$
General: $x = 3 + 2 \times k$ $y = -6 - 5 \times k$ (*k* is an integer)

b. We have $a = 19$, $b = 13$ and $c = 20$. Since $d = \gcd(a, b) = 1$ and divides c, there is an infinite number of solutions. The reduced equation is $19x + 13y = 20$. We solve the equation $19s + 13t = 1$ to get $s = -2$ and $t = 3$. The particular and general solutions are

Particular:	$x_0 = (c/d) \times s = -40$	$y_0 = (c/d) \times t = 60$
General:	$x = -40 + 13 \times k$	$y = 60 - 19 \times k$ (k is an integer)

c. We have $a = 14$, $b = 21$ and $c = 77$. Since $d = \gcd(a, b) = 7$ divides c, there is an infinite number of solutions. The reduced equation is $2x + 3y = 11$. We solve the equation $2s + 3t = 1$ to get $s = -1$ and $t = 1$. The particular and general solutions are

Particular:	$x_0 = (c/d) \times s = -11$	$y_0 = (c/d) \times t = 11$
General:	$x = -11 + 3 \times k$	$y = 11 - 2 \times k$ (k is an integer)

d. We have $a = 40$, $b = 16$ and $c = 88$. Since $d = \gcd(a, b) = 8$ divides c, there is an infinite number of solutions. The reduced equation is $5x + 2y = 11$. We solve the equation $5s + 2t = 1$ to get $s = 1$ and $t = -2$. The particular and general solutions are

Particular:	$x_0 = (c/d) \times s = 11$	$y_0 = (c/d) \times t = -22$
General:	$x = 11 + 2 \times k$	$y = -22 - 5 \times k$ (k is an integer)

34.

a. Since gcd (15, 12) = 3 and 3 does not divide 13, there is no solution.

b. Since gcd (18, 30) = 6 and 6 does not divide 20, there is no solution.

c. Since gcd (15, 25) = 5 and 5 does not divide 69, there is no solution.

d. Since gcd (40, 30) = 10 and 10 does not divide 98, there is no solution.

35. We have the equation $39x + 15y = 270$. We have $a = 39$, $b = 15$ and $c = 270$. Since $d = \gcd(a, b) = 3$ divides c, there is an infinite number of solutions. The reduced equation is $13x + 5y = 90$. We solve the equation $13s + 5t = 1$: $s = 2$ and $t = -5$. The particular and general solutions are

Particular:	$x_0 = (c/d) \times s = 180$	$y_0 = (c/d) \times t = -450$
General:	$x = 180 + 5 \times k$	$y = -450 - 13 \times k$

To find an acceptable solution (nonnegative values) for x and y, we need to start with negative values for k. Two acceptable solutions are

$$k = -35 \rightarrow x = 5 \text{ and } y = 5 \qquad k = -36 \rightarrow x = 0 \text{ and } y = 18$$

36. In each case, we follow three steps discussed in Section 2.4 of the textbook.

a.

Step 1: $a = 3, b = 4, n = 5 \quad \rightarrow \quad d = \gcd(a, n) = 1$

Since d divides b, there is only one solution.

Step 2: Reduction: $3x \equiv 4 \pmod 5$

Step 3: $x_0 = (3^{-1} \times 4) \pmod 5 = 2$

b.

Step 1: $a = 4, b = 4, n = 6 \quad \rightarrow \quad d = \gcd(a, n) = 2$

Since d divides b, there are two solutions.

Step 2: Reduction: $2x \equiv 2 \pmod 3$

Step 3: $x_0 = (2^{-1} \times 2) \pmod 3 = 1 \qquad x_1 = 1 + 6 / 2 = 4$

c.

Step 1: $a = 9, b = 12, n = 7 \quad \rightarrow \quad d = \gcd(a, n) = 1$

Since d divides b, there is only one solution.

Step 2: Reduction: $9x \equiv 12 \pmod 7$

Step 3: $x_0 = (9^{-1} \times 12) \pmod 7 = (2^{-1} \times 5) \pmod 7 = 4$

d.

Step 1: $a = 256, b = 442, n = 60 \quad \rightarrow \quad d = \gcd(a, n) = 4$

Since d does not divide b, there is no solution.

37.

a.

$3x + 5 \equiv 4 \pmod 5 \rightarrow 3x \equiv (-5 + 4) \pmod 5 \rightarrow 3x \equiv 4 \pmod 5$

$a = 3, b = 4, n = 5 \quad \rightarrow \quad d = \gcd(a, n) = 1$

Since d divides b, there is only one solution.

Reduction: $3x \equiv 4 \pmod 5$

$x_0 = (3^{-1} \times 4) \pmod 5 = 2$

b.

$4x + 6 \equiv 4 \pmod 6 \rightarrow 4x \equiv (-6 + 4) \pmod 6 \rightarrow 4x \equiv 4 \pmod 6$

$a = 4, b = 4, n = 6 \rightarrow d = \gcd(a, n) = 2$

Since d divides b, there are two solutions.

Reduction: $2x \equiv 2 \pmod 3$

$x_0 = (2^{-1} \times 2) \pmod 3 = 1$

$x_1 = 1 + 6 / 2 = 4$

c.

$9x + 4 \equiv 12 \pmod 7 \rightarrow 9x \equiv (-4 + 12) \pmod 7 \rightarrow 9x \equiv 1 \pmod 7$

$a = 9, b = 1, n = 7 \rightarrow d = \gcd(a, n) = 1$

Since d divides b, there is only one solution.

Reduction: $9x \equiv 1 \pmod 7$

$x_0 = (9^{-1} \times 1) \pmod 7 = 4$

d.

$232x + 42 \equiv 248 \pmod{50} \rightarrow 232x \equiv 206 \pmod{50}$

$a = 232, b = 206, n = 50 \rightarrow d = \gcd(a, n) = 2$

Since d divides b, there are two solutions.

Reduction: $116x \equiv 103 \pmod{25} \rightarrow 16x \equiv 3 \pmod{25}$

$x_0 = (16^{-1} \times 3) \pmod{25} = 8$

$x_1 = 8 + 50/2 = 33$

38.

a. The result of multiplying the first two matrices is a 1 × 1 matrix, as shown below:

$$\begin{bmatrix} 3 & 7 & 10 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} (3 \times 2 + 7 \times 4 + 10 \times 12) \bmod 16 \end{bmatrix} = \begin{bmatrix} 10 \end{bmatrix}$$

b. The result of multiplying the second two matrices is a 3 × 3 matrix, as shown below:

$$\begin{bmatrix} 3 & 4 & 6 \\ 1 & 1 & 8 \\ 5 & 8 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & 0 \\ 5 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 0 & 11 \\ 11 & 1 & 1 \\ 1 & 14 & 1 \end{bmatrix}$$

39.

a. The determinant and the inverse of matrix A are shown below:

$$A = \begin{bmatrix} 3 & 0 \\ 1 & 1 \end{bmatrix} \longrightarrow \det(A) = 3 \bmod 10 \longrightarrow (\det(A))^{-1} = 7 \bmod 10$$

$$A^{-1} = 7 \times \underbrace{\begin{bmatrix} 1 & 0 \\ 9 & 3 \end{bmatrix}}_{\text{adj}(A)} \longrightarrow A^{-1} = \begin{bmatrix} 7 & 0 \\ 3 & 1 \end{bmatrix}$$

b. Matrix B has no inverse because det(B) = (4 × 1 − 2 × 1) mod = 2 mod 10, which has no inverse in $\mathbf{Z}_{10}$.

c. The determinant and the inverse of matrix C are shown below:

$$C = \begin{bmatrix} 3 & 4 & 6 \\ 1 & 1 & 8 \\ 5 & 8 & 3 \end{bmatrix} \longrightarrow \det(C) = 3 \bmod 10 \longrightarrow (\det(C))^{-1} = 7 \bmod 10$$

$$C^{-1} = \begin{bmatrix} 3 & 2 & 2 \\ 9 & 3 & 4 \\ 1 & 2 & 3 \end{bmatrix}$$

In this case, det(C) = 3 mod 10; its inverse in $\mathbf{Z}_{10}$ is 7 mod 10. It can proved that $C \times C^{-1} = \mathbf{I}$ (identity matrix).

40. Although we give the general method for every case using matrix multiplication, in cases *a* and *c*, there is no need for matrix multiplication because the coefficient of *y* (in *a*) or *x* (in *c*) is actually 0 in these two cases. These cases can be solved much easier.

a. In this particular case, the answer can be found easier because the coefficient of y is 0 in the first equation.The solution is shown below:

$$\begin{bmatrix} 3 & 5 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 2 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \rightarrow \boxed{\begin{matrix} x = 3 \bmod 5 \\ y = 2 \bmod 5 \end{matrix}}$$

b. The solution is shown below:

$$\begin{bmatrix} 3 & 2 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 4 & 6 \end{bmatrix}^{-1} \times \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \rightarrow \boxed{\begin{matrix} x = 5 \bmod 7 \\ y = 2 \bmod 7 \end{matrix}}$$

c. The solution is shown below:

$$\begin{bmatrix} 7 & 3 \\ 4 & 2 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 & 3 \\ 4 & 2 \end{bmatrix}^{-1} \times \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 5 & 0 \end{bmatrix} \times \begin{bmatrix} 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \end{bmatrix} \rightarrow \boxed{\begin{matrix} x = 6 \bmod 7 \\ y = 1 \bmod 7 \end{matrix}}$$

d. The solution is shown below:

$$\begin{bmatrix} 2 & 3 \\ 1 & 6 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & 6 \end{bmatrix}^{-1} \times \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 & 5 \\ 7 & 2 \end{bmatrix} \times \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix} \rightarrow \boxed{\begin{matrix} x = 5 \bmod 8 \\ y = 1 \bmod 8 \end{matrix}}$$

CHAPTER 3

Traditional Symmetric-Key Ciphers

(Solution to Practice Set)

Review Questions

1. Symmetric-key encipherment uses a single key for both encryption and decryption. In addition, the encryption and decryption algorithms are inverse of each other.

2. Traditional symmetric-key ciphers can be divided into two broad categories: substitution ciphers and transposition ciphers. A substitution cipher replaces one character with another character. A transposition cipher reorders the symbols.

3. Substitution ciphers can be divided into two broad categories: monoalphabetic ciphers and polyalphabetic ciphers. In monoalphabetic substitution, the relationship between a character in the plaintext and the characters in the ciphertext is one-to-one. In polyalphabetic substitution, the relationship between a character in the plaintext and the characters in the ciphertext is one-to-many.

4. Symmetric-key ciphers can also be divided into two broad categories: stream ciphers and block ciphers. In a stream cipher, encryption and decryption are done one symbol at a time. In a block cipher, symbols in a block are encrypted together.

5. A stream cipher is a monoalphabetic cipher if the value of k_i does not depend on the position of the plaintext character in the plaintext stream; otherwise, the cipher is polyalphabetic.

6. In a block cipher, each character in a ciphertext block depends on all characters in the corresponding plaintext block. The cipher, therefore, is a polyalphabetic.

7. The additive ciphers, multiplicative ciphers, affine ciphers, and monoalphabetic substitution cipher are some examples of monoalphabetic ciphers.

8. The autokey cipher, Playfair cipher, Vigenere cipher, Hill cipher, rotor cipher, one-time pad, and Enigma machine are some examples of polyalphabetic ciphers.

9. The rail fence cipher and double transposition cipher are examples of transposition ciphers.

10. Brute-force attack, statistical attack, pattern attack, and Kasiski test are examples of attacks on traditional ciphers.

Exercises

11.

a. The number of keys = $n \times (n - 1) / 2 = (100 \times 99) / 2 = 4950$.

b. Only 100 keys are needed: There should be one secret key between the president and each member.

c. Only 100 keys are needed: There should be one secret key between the president and each member to create the session secret key. After the session key is established, the two members can use the one-time session key.

12. The sentence in the tablet and its translation serves as a known plaintext/ciphertext pair. This is a *known-plaintext* attack.

13. Double encryption here does not help. Encryption with k_1 followed by encryption with k_2 is the same as encryption with $k = (k_1 + k_2) \bmod 26$.

$$C = [(P + k_1) \bmod 26) + k_2] \bmod 26 = (P \bmod 26 + k_1 \bmod 26 + k_2) \bmod 26$$
$$C = (P \bmod 26) \bmod 26 + (k_1 \bmod 26) \bmod 26 \ + k_2 \bmod 26$$
$$C = P \bmod 26 + k_1 \bmod 26 \ + k_2 \bmod 26 = (P + k_1 + k_2) \bmod 26$$
$$C = (P + k) \bmod 26, \text{ where } k = (k_1 + k_2) \bmod 26$$

14.

a. The compression in general creates a text which is not in the source language (English for example). This means that the compressed plaintext does not preserve the frequency of characters. It helps Alice in this case.

b. Compression before encryption can be more efficient because encryption is normally time consuming if the message is very long.

15.

a. The size of the key domain is 26 + 10 = 36. The modulus is also 36. Alice needs to use the set $\mathbf{Z}_{36}$.

b. The size of the key domain is 12; the domain is (1, 5, 7, 11, 13, 17, 19, 23, 25, and 29). The modulus is 36. Alice needs to use the set $\mathbf{Z}_{36}*$.

c. The key domain is $36 \times 12 = 432$. The modulus is still 36. However, Alice needs to use $\mathbf{Z}_{36}$ for addition and $\mathbf{Z}_{36}*$ for multiplication.

16.

a. The size of the key domain is 29.

b. The size of the key domain is 28 because 29 is a prime number.

c. The size of the key domain is $29 \times 28 = 812$.

17.

a. Random switching between substitution and transposition is as difficult for Alice and Bob as it is for Eve to discover which method is being used. If Alice and Bod do not have a secure channel to inform each other which method they are using (which is normally the case), they need to toss a coin. Eve can do the same. However, Alice and Bob can use a pattern (for example, three substitutions and two transpositions), but using any pattern is a kind of weakness in secrecy.

b. The same argument used in part *a* can be used here. However, if Eve knows that cipher is a substitution, she can use more tools to find out the which type. For example, if she can easily break the code using brute-force attack on the key, she knows that they are using either additive or multiplicative cipher.

c. The same argument used in part *a* can be used here. However, if Eve knows that the cipher is transposition, she can use the pattern attack to find the size of the section.

18.

a. In additive cipher, $C_i = (P_i + k) \bmod 26$. This means that if one character in the plaintext is changed only one character in the ciphertext is changed. C_i depends only P_i.

b. In multiplicative cipher, $C_i = (P_i \times k) \bmod 26$. This means that if one character in the plaintext is changed only one character in the ciphertext is changed. C_i depends only P_i.

c. In affine cipher, $C_i = (P_i \times k_1 + k_2) \bmod 26$. This means that if one character in the plaintext is changed only the corresponding character in the ciphertext is changed. C_i depends only P_i.

d. In Vigenere cipher, $C_i = (P_i + k_i) \bmod 26$. Although, the value of k_i may change, but the change does not depend on the previous or next characters; the change depends only on position of the plaintext character. If only one character in the plaintext is changed, only one character in the ciphertext will be changed.

e. In autokey cipher, $C_i = (P_i + k_i) \bmod 26 = (P_i + P_{i-1}) \bmod 26$. This means each ciphertext character (except the first) depends on the corresponding plaintext character and the previous plaintext character. Therefore, changing one single character in the plaintext will change all characters in the ciphertext which comes after that character. In other words, if we change character 21 in the plaintext, characters 22, 23, 24, … will be changed.

f. In one-time pad, the key stream is used only once. Each ciphertext character, however, depends only the corresponding plaintext ciphertext. If only one character in the plaintext is changed, only one character in the ciphertext will be changed.

g. The rotor cipher is a kind of monoalphabetic substitution cipher, in which the mapping table will be changed from one character to the next. Each ciphertext character, however, depends only the corresponding plaintext character. If only one character in the ciphertext is changed, only one character in the plaintext will be changed.

h. The Enigma machine is based on the rotor cipher. Therefore, If only one character in the plaintext is changed, only one character in the ciphertext will be changed.

19.

a. Single transposition only reorders the characters. If one character is changed in the plaintext, it affects only one character in the ciphertext.

b. Double transposition only reorders the characters. If one character is changed in the plaintext, it affects only one character in the ciphertext.

c. In the Playfair cipher, encryption is two characters at a time. If one character in the plaintext is changed, it normally changes one or two characters in the ciphertext. However, if the changed character is the same as the previous or next character, we need to add one bogus character in the plaintext that may change several characters in the ciphertext.

20.

a. The Playfair is a block cipher; encryption is done two character at a time.

b. The autokey cipher is a stream cipher; encryption is done one character at a time.

c. The one-time pad cipher is a stream cipher; encryption is done one character at a time.

d. The rotor cipher is a stream cipher; encryption is done one character at a time.

e. The Enigma machine is a is a stream cipher; encryption is done one character at a time.

21.

Cipher	Plaintext	Ciphertext
Additive, key = 20	**This is an exercise**	**NBCMCMUHYRYLWCMY**
Multiplicative, key = 15	**This is an exercise**	**ZBQKQKANIHIVEQKI**
Affine, key = (15, 20)	**This is an exercise**	**TVKEKEUHCBCPYKEC**

22.

Cipher	Plaintext	Ciphertext
Vigenere	**The house is being sold tonight**	**WVPSOLKHWDMEZFJGZWDKGQWRST**
Autokey	**The house is being sold tonight**	**AALLVIMWMATFMVTYGZOWHBVONA**
Playfair	**The house is being sold tonight**	**WECEXIOHNOEIFIHKXBBWSIRBEW**

23.

Plaintext	Ciphertext
Life is full of surprise	**SMFPBZMYLWHMZYPPKPZI**

24. We use the following key:

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

We need to add three bogus character, x's, to separate two d's and two o's and one at the end to make the number of characters even. The transformed plaintext is actually "th ek ey is hi dx de nu nd er th ed ox or pa dx".

Plaintext	Ciphertext
The key is hidden under the door pad	**POKLBXDRLGIYIBCGBGLXPOBILZTLGTIY**

25. We add the bogus character, "z" to the end of the plaintext to make the number of characters multiple of 2. The plaintext matrix, the key matrix, and ciphertext matrix are shown below:

$$\underset{\mathbf{C}}{\begin{bmatrix} 8 & 20 \\ 21 & 0 \\ 5 & 18 \\ 11 & 3 \\ 13 & 13 \\ 11 & 3 \\ 22 & 12 \\ 2 & 14 \\ 19 & 10 \\ 6 & 12 \\ 2 & 7 \\ 4 & 25 \end{bmatrix}} = \underset{\mathbf{P}}{\begin{bmatrix} 22 & 4 \\ 11 & 8 \\ 21 & 4 \\ 8 & 13 \\ 0 & 13 \\ 8 & 13 \\ 18 & 4 \\ 2 & 20 \\ 17 & 4 \\ 22 & 14 \\ 17 & 11 \\ 3 & 25 \end{bmatrix}} \times \underset{\mathbf{K}}{\begin{bmatrix} 3 & 2 \\ 5 & 7 \end{bmatrix}}$$

The ciphertext is then "IUVAFSLDNNLDWMCOTKGMCHEZ", in which the last character is a bogus character.

26. This is known-plaintext attack. Johns knows a plaintext/ciphertext pair (yes → CIW). He knows that the algorithm is shift cipher and the key = 4. When he gets access to the ciphertext "XVIEWYVT", he decrypts it as "treasure".

27.

a. Eve is launching a chosen-plaintext attack.

b. The length of the message is 10. Since 10 = 2 × 5, The number of columns can be 1, 2, 5 or 10. The first or the last guess is unlikely, so the number of columns is either 2 or 5.

28. We can try the keys 13, 12, 14, 11, 15 which are close to Alice's birthday. When we use the key = 11, the plaintext makes sense.

Key = 11

Ciphertext: NCJAEZRCLASJLYODEPRLYZRCLASJLCPEHZDTOPDZQLNZTY

Plaintext: cryptography and steganography are two sides of a coin

29. We know that "ab" $\rightarrow$ "GL". This means that

$$00 \rightarrow 06 \quad \text{and} \quad 01 \rightarrow 11$$

We can construct two equations from these two pieces of information:

$$00 \times k_1 + k_2 \equiv 06 \pmod{26} \qquad 01 \times k_1 + k_2 \equiv 11 \pmod{26}$$

Solving these two equations give us $k_1 = 5$ and $k_2 = 6$. This means,

$$\text{P} = ((\text{C} - k_2) \times {k_1}^{-1}) \bmod 26 = ((\text{C} + 20) \times 21) \bmod 26$$

Ciphertext: XPALASXYFGFUKPXUSOGEUTKCDGFXANMGNVS

Plaintext: the best of a fight is making up afterwards

30. We can find the single-character frequency of letters in this ciphertext as shown below:

Character	O	H	B	G	W	N	V	E	J	C	U	L	R
Frequency	4	4	4	4	3	2	2	2	1	1	1	1	1

This statistic, however, is not very useful because of several reasons:

a. The length of the ciphertext is very short for this analysis (only 30 characters).

b. Only 10 out of 26 characters in the alphabet is used. When monoalphabetic substitution is used, we need to have all or most of the characters to be present in the ciphertext.

c. The frequencies are not distributed. We have only four 4's, one 3, three 2's, and five 1's.

With some more guessing, we can find the mapping as:

O	**N**	**H**	**O**	**V**	**E**	**J**	**H**	**W**	**O**	**B**	**E**	**V**	**G**	**W**	**O**	**C**	**B**	**W**	**H**	**N**	**U**	**G**	**B**	**L**	**H**	**G**	**B**	**G**	**R**
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
o	**n**	**e**	**o**	**f**	**t**	**h**	**e**	**m**	**o**	**s**	**t**	**f**	**a**	**m**	**o**	**u**	**s**	**m**	**e**	**n**	**w**	**a**	**s**	**c**	**e**	**a**	**s**	**a**	**r**

This gives us the plaintext that makes sense:

Plaintext: One of the most famous men was Ceasar.

31.

a. Since each entry can be one of the 29 characters, the total number of potential keys are $29^4 = 707,281$.

b. Out of these 707,281 keys, only 682,080 of them are usable.

32. We can find the single-character frequency of letters in this ciphertext as shown below:

Character	B	Q	M	A	V	L	Z	I	T	W	P
Frequency	18	15	14	12	11	10	10	9	9	9	7
Character	E	G	N	C	O	U	X	J	S	K	D
Frequency	6	6	5	5	4	4	3	2	1	1	1

Although "B" and "Q" have the highest frequencies, if we assume that "B" or "Q" are the decryption of "e", the plaintext does not make sense. We assume that "M" is the decryption of "e". In this case 12 is decrypted as 04 so the key = 8. Using this key the plaintext is

Plaintext:
Glow of youth is tarnished, by rolling time's heavy rust
Winter draped spring, in deep snow and dark frost,
Bird of delight named youth, swiftly left the nest
Alas, couldn't tame it, while it was in my trust.

It is the English translation of a poem by Khayyam, the Persian poet, philosopher, and mathematician of the twelve century.

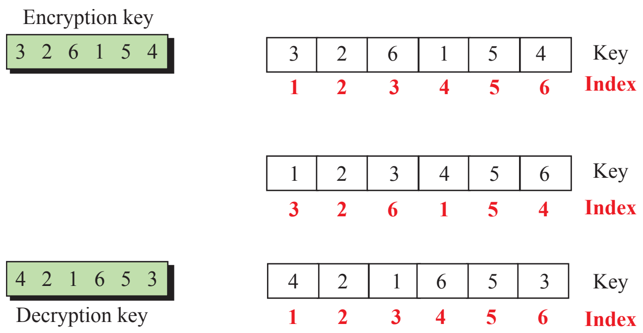
33. This is an example that Kasiski test cannot help us. The reason is that the plaintext has been encrypted line by line. The ciphertext is also created line by line, but each line is too short for Kasiski test. The encryption algorithm uses a 5-character word "*a poet*" to encrypt the text without adding any padding at the end of the last segment.

```
maken oment ionof rough times forge tabou teven tspas t
MPYIG OBSRM IDBSY RDIKA TXAIL FDFKX TPPSN TTJIG THDEL T
timey ettoc omeis unrev ealed andbe ingre veale dwill notla st
TXAIR EIHSV OBSML UCFIO EPZIW ACRFX ICUVX VTOPX DLWPE NDHPT SI
dontd wello ndays ofyou rnorb uildo nhere after
DDBXW WTZPH NSOCL OUMSN RCCVU UXZHH NWSVX AUHIK
lifet otoda yandt histo owill whisk awaya sblas t
LXTIM OICHT YPBHM HXGXH OLWPE WWWWD ALOCT SQZEL T
```

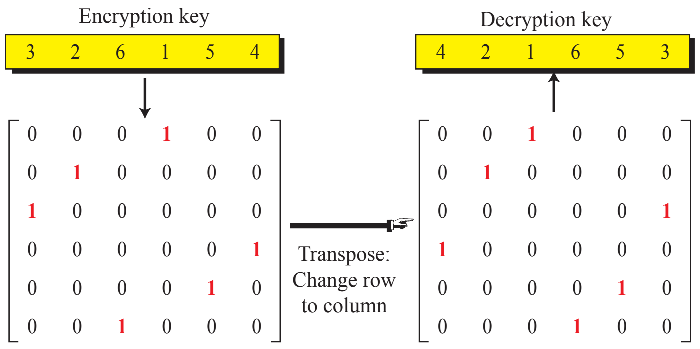
After adding the punctuation, we get another English translation of a poem by Khayyam, the Persian poet, philosopher, and mathematician of the twelve century.

Make no mention of rough times, forget about events past.
Time yet to come is unrevealed, and being revealed will not last,
Don’t dwell on days of your, nor build on here after,
Life to today and this too will whisk away as blast.

34. We follow the process shown in Figure 3.23 of the text.



35. We follow the idea shown in Figure 3.24 of the text.



36. The length of the message is 12. The size of the block needs to divide 12. This means that the size of the block can be 1, 2, 3, 4, 6, 12. Since the first and the last size is trivial, we can ignore them. We need to try the block sizes 2, 3, 4, and 6.

However, the size of message (12) does not allow us to test for the matrices of size 4 or 16. If the size of the key matrix is 4, we need at least the plaintext/ciphertext of size 16. If the size of the key matrix is 6, we need to have a plaintext/ciphertext of size 36. Our only choices are to test for the key matrices of size 2 and 3.

a. Let us first try the block size 2. Following Example 3.21 in the text, we can make a plaintext/ciphertext pair of the first four of the given plaintext and given ciphertext (letu → HBCD). In this case the plaintext matrix and ciphertext are

$$P = \begin{bmatrix} 11 & 04 \\ 19 & 08 \end{bmatrix} \qquad C = \begin{bmatrix} 07 & 01 \\ 02 & 03 \end{bmatrix}$$

Since P is not invertible in modulo 26 arithmetic, we can not proceed.

b. Let us now try the block size 3. Following Example 3.21 in the text, we can make a plaintext/ciphertext pair of the first nine character of the given plaintext and given ciphertext (let usm eet → HBCDFNOPI). In this case the plaintext and ciphertext matrices are

$$P = \begin{bmatrix} 11 & 04 & 19 \\ 20 & 18 & 12 \\ 04 & 04 & 19 \end{bmatrix} \qquad C = \begin{bmatrix} 07 & 01 & 02 \\ 03 & 05 & 13 \\ 14 & 15 & 08 \end{bmatrix}$$

Since P is not invertible in modulo 26 arithmetic, we can not proceed. This means that we cannot solve the problem with the information given.

37. We can use a row matrix of size m for addition matrix, but we need to use a square matrix for multiplication (to be reversible).

a. For the additive cipher we have

$$\underset{\text{C}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}} = \underset{\text{P}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}} + \underset{\text{k}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}}$$

$$\underset{\text{P}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}} = \underset{\text{C}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}} - \underset{\text{k}}{\begin{bmatrix} \quad 1 \times m \quad \end{bmatrix}}$$

b. For the affine cipher we have

$$\underset{C}{\begin{bmatrix} 1\times m \end{bmatrix}} = \underset{P}{\begin{bmatrix} 1\times m \end{bmatrix}} \times \underset{k_1}{\begin{bmatrix} m\times m \end{bmatrix}} + \underset{k_2}{\begin{bmatrix} 1\times m \end{bmatrix}}$$

$$\underset{P}{\begin{bmatrix} 1\times m \end{bmatrix}} = \left(\underset{C}{\begin{bmatrix} 1\times m \end{bmatrix}} - \underset{k_2}{\begin{bmatrix} 1\times m \end{bmatrix}}\right) \times \underset{k_1}{\begin{bmatrix} m\times m \end{bmatrix}}^{-1}$$

38. The following shows the encryption process. The position and the value of each character in the plaintext (P. Text) is found. The multiplication and addition keys (k_1 and k_2) are listed. The numeric result of encryption for each character is calculated (P × k_1 + k_2), and finally the ciphertext characters (C. Text) are determined and listed.

P. Text	**c**	**r**	**y**	**p**	**t**	**o**	**g**	**r**	**a**	**p**	**h**	**i**	**s**	**f**	**u**	**n**
Position	0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3

Values	**2**	**17**	**24**	**15**	**19**	**14**	**6**	**17**	**0**	**15**	**7**	**8**	**18**	**5**	**20**	**13**
k_1	**1**	**3**	**5**	**7**	**9**	**11**	**15**	**17**	**19**	**21**	**23**	**25**	**1**	**3**	**5**	**7**
k_2	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

results	**2**	**0**	**18**	**4**	**19**	**3**	**18**	**10**	**8**	**12**	**15**	**3**	**4**	**2**	**10**	**4**
C. Text	**C**	**A**	**S**	**E**	**T**	**D**	**S**	**K**	**I**	**M**	**P**	**D**	**E**	**C**	**K**	**E**

39. In the Hill cipher, **C** = **P** × **K**. If the plaintext is the identity matrix **I**, then we have **C** = **K**. This means that if can access to the Alice computer and launch a chosen plaintext attack using the trivial identity matrix, Eve can find the key. This shows that the Hill cipher is very vulnerable to the chosen-plaintext attack.

40. The relationship between the plaintext and ciphertext is P + C = 25 or C = (25 − P) mod 26.

Plaintext	Ciphtertext
an exercise	**ZMVCVIXRHV**

41. We use the following key:

	1	**2**	**3**	**4**	**5**
1	z	q	p	f	e
2	y	r	o	g	d
3	x	s	n	h	c
4	w	t	m	i / j	b
5	v	u	l	k	a

The plaintext and ciphertext are shown below:

Plaintext	Ciphtertext
an exercise	(5, 5), (3, 3), (1, 5), (3, 1), (1, 5), (2, 2), (3, 5), (4, 4), (3, 2), (1, 5)

CHAPTER 4

Mathematics of Cryptography Part II: Algebraic Structures

(Solution to Practice Set)

Review Questions

1. The combination of the set and the operations that are applied to the elements of the set is called an *algebraic structure*. We have defined three common algebraic structures: *groups*, *rings*, and *fields*.

2. A group is a set of elements with a binary operation that satisfies four properties: closure, associativity, existence of identity, and existence of inverse. A commutative group, also called an abelian group, is a group in which the operator satisfies the four properties for groups plus an extra property, commutativity.

3. A ring is an algebraic structure with two operations. The first operation must satisfy all five properties required for an abelian group. The second operation must satisfy only the first two. In addition, the second operation must be distributed over the first. A commutative ring is a ring in which the commutative property is also satisfied for the second the operation.

4. A field is a commutative ring in which the second operation satisfies all five properties defined for the first operation except that the identity of the first operation (sometimes called the zero element) has no inverse. A finite field is a field with a finite number of elements. An infinite field is a field with an infinite number of elements.

5. A Galois field, GF(p^n), is a finite field with p^n elements. If $n = 1$, the field is sometimes referred to as GF(p).

6. An example of a group is $\mathbf{G} = <\mathbf{Z}_n, +>$. The identity element is 0; the inverse of an element a is $-a$.

7. An example of a ring is $\mathbf{R} = <\mathbf{Z}, +, \times>$. For the first operation, the identity element is 0; the inverse of an element a is $-a$. Neither the identify element nor the inverse of an element is defined for the second operation.

8. An example of a field is $\mathbf{F} = <\mathbf{Z}_p, +, \times>$. For the first operation, the identity element of is 0; the inverse of an element a is $-a$. For the second operation, the identity element is 1 and the inverse of an element *a* is a^{-1}. The identity element of the first operation, however, has no inverse with regard to the second operation.

9. A polynomial of degree $n - 1$ with coefficient in GF(2) can represent an *n*-bit word with power of each term defining the position of the bit and the coefficients of the terms defining the value of the bits.

10. In GF(2^n), *a reducible polynomial* of degree *n* is a polynomial with coefficient in GF(2) that cannot be factored into a polynomial with degree of less than *n*. A reducible polynomial is sometimes referred to as a prime polynomial.

Exercises

11. The group **G** = <**Z**$_4$, +> has only four members: 0, 1, 2, and 3.

a. For all *a*'s and *b*'s members of G, we need to prove that $a + b = b + a$. The following shows the proof (all operations are modulo 4).

$$(0 + 1) = (1 + 0) \qquad (0 + 2) = (2 + 0) \qquad (0 + 3) = (3 + 0)$$
$$(1 + 2) = (2 + 1) \qquad (1 + 3) = (3 + 1)$$
$$(2 + 3) = (3 + 2)$$

b.

$$(3 + 2) \bmod 4 = 1 \bmod 4 \qquad (3 - 2) \bmod 4 = -1 \bmod 4 = 3 \bmod 4$$

12. The group **G** = <**Z**$_6$*, ×> has only two members: 1 and 5.

a. For all *a*'s and *b*'s members of G, we need to prove that $a \times b = b \times a$. Since this group has only two members, we can see that $(1 \times 5) \bmod 6 = (5 \times 1) \bmod 6$.

b.

$$(5 \times 1) \bmod 6 = 5 \bmod 6$$
$$(1 \div 5) \bmod 6 = (1 \times 5^{-1}) \bmod 6 = (1 \times 5) \bmod 6 = 5 \bmod 6$$

c. There is no need to worry about division by zero in this group, because 0 is not the member of this group.

13. Assume that the operation is (♦). We can say that $(x ♦ y)$ is the same as $x \bullet (-y)$, in which $(-y)$ is the inverse of *y* with respect to operation (•). Using Table 4.1, we can create the following table:

♦	***a***	***b***	***c***	***d***
a	*a*	*d*	*c*	*b*
b	*b*	*a*	*d*	*c*
c	*c*	*b*	*a*	*d*
d	*d*	*c*	*b*	*a*

Another way to solve the problem is to think about similarity between the group represented in Table 4.1 and the group **G** = <**Z**$_4$, +>. Make the table for subtraction operation in the group **G** = <**Z**$_4$, +> and replace 0 with *a*, 1 with *b*, 2 with *c*, and 3 with *d*.

14. It is enough to prove one single case ($a \circ b \neq b \circ a$):

$$[1\ 3\ 2] \circ [2\ 1\ 3] = [3\ 1\ 2] \quad \text{but} \quad [2\ 1\ 3] \circ [1\ 3\ 2] = [2\ 3\ 1]$$

15. We use only two cases:

a. We first prove that

$$([1\ 3\ 2] \circ [2\ 1\ 3]) \circ [3\ 1\ 2] = [1\ 3\ 2] \circ ([2\ 1\ 3] \circ [3\ 1\ 2])$$

$$([1\ 3\ 2] \circ [2\ 1\ 3]) \circ [3\ 1\ 2] = [3\ 1\ 2] \circ [3\ 1\ 2] = [2\ 3\ 1]$$
$$[1\ 3\ 2] \circ ([2\ 1\ 3] \circ [3\ 1\ 2]) = [1\ 3\ 2] \circ [3\ 2\ 1] = [2\ 3\ 1]$$

b. We then prove that

$$([1\ 2\ 3] \circ [2\ 1\ 3]) \circ [3\ 1\ 2] = [1\ 2\ 3] \circ ([2\ 1\ 3] \circ [3\ 1\ 2])$$

$$([1\ 2\ 3] \circ [2\ 1\ 3]) \circ [3\ 1\ 2] = [2\ 1\ 3] \circ [3\ 1\ 2] = [3\ 2\ 1]$$
$$[1\ 2\ 3] \circ ([2\ 1\ 3] \circ [3\ 1\ 2]) = [1\ 2\ 3] \circ [3\ 2\ 1] = [3\ 2\ 1]$$

16. The following shows the composition with identify elements.

∘	[1 2]	[2 1]
[1 2]	[1 2]	[2 1]
[2 1]	[2 1]	[1 2]

17. The result of ([1 3 2] ∘ [3 2 1]) ∘ [2 1 3] = **[3 2 1]**. Bob can use the permutation [3 2 1] to reverse the operation. This proves that double or multiple permutation does not help; Alice could have used one single permutation.

18.

a. In this case, |**G**| = 16, which means the order of each subgroup should divide 16. We can have subgroups with orders 1, 2, 4, 8, 16. The following shows some of these subgroups.

|H| = 1	→	H = <{1}, ×>		
|H| = 2	→	H = <{1, 7}, ×>	H = <{1, 9}, ×>	H = <{1, 15}, ×>
|H| = 4	→	H = <{1 , 3, 9 , 11}, ×>	H = <{1, 7, 9 , 15}, ×>	
|H| = 8	→	H = G		

b. In this case, |**G**| = 23, which means the order of each subgroup should divide 23. We can have subgroups with orders 1 and 23. The following shows some of these subgroups.

|H| = 1	$\rightarrow$	**H = <{0}, +>**
|H| = 23	$\rightarrow$	**H = G**

c. In this case, |**G**| = 8, which means the order of each subgroup should divide 8. We can have subgroups with orders 1, 2, 4, 8. The following shows some of these subgroups.

|H| = 1	$\rightarrow$	**H = <{1}, ×>**		
|H| = 2	$\rightarrow$	**H = <{1, 7}, ×>**	**H = <{1, 9}, ×>**	**H = <{1, 15}, ×>**
|H| = 4	$\rightarrow$	**H = <{1 , 3, 9 , 11}, ×>**	**H = <{1, 7, 9 , 15}, ×>**	
|H| = 8	$\rightarrow$	**H = G**		

d. In this case, |**G**| = 16, which means the order of each subgroup should divide 16. We can have subgroups with orders 1, 2, 4, 8, 16. The following shows some of these subgroups.

|H| = 1	$\rightarrow$	**H = <{1}, ×>**
|H| = 2	$\rightarrow$	**H = <{1, 16}, ×>**
|H| = 4	$\rightarrow$	**H = <{1, 4, 13, 16}, ×>**
|H| = 16	$\rightarrow$	**H = G**

19.

a. The order of the group is |**G**| = 18. The order of potential subgroups should divide 18, which means |**H**| can be 1, 2, 3, 6, 9, and 18.

b. The order of the group is |**G**| = 29. The order of potential subgroups should divide 29, which means |**H**| can be 1 and 29.

c. The order of the group is |**G**| = 4. The order of potential subgroups should divide 4, which means |**H**| can be 1, 2, and 4.

d. The order of the group is |**G**| = 18. The order of potential subgroups should divide 18, which means |**H**| can be 1, 2, 3, 6, 9, and 18.

20.

a. For **G** = <$\mathbf{Z}_8$, +>, we have

ord (0) = 0	**ord (1) = 8**	**ord (2) = 4**	**ord (3) = 8**	**ord (4) = 2**
ord (5) = 8	**ord (6) = 4**	**ord (7) = 8**		

b. For **G** = <$\mathbf{Z}_7$, +>, we have

ord (0) = 0	ord (1) = 7	ord (2) = 7	ord (3) = 7	ord (4) = 7
ord (5) = 7	ord (6) = 7			

c. For **G** = <**Z**$_9$* ×>, we have

ord (1) = 0	ord (2) = 6	ord (4) = 3	ord (5) = 6	ord (7) = 3
ord (8) = 2				

d. For **G** = <**Z**$_7$* ×>, we have

ord (1) = 0	ord (2) = 3	ord (3) = 6	ord (4) = 6	ord (5) = 6
ord (6) = 2				

21. The elements 0, g^0, g^1, g^2, and g^3 can be easily be generated, because they are the 4-bit representations of 0, 1, x^2, and x^3. We use the relation $f(g) = g^4 + g^3 + 1 = 0$ to generate other powers. Using this relation, we have $g^4 = g^3 + 1$. We use this relation to find the value of all elements as 4-bit words:

$$\begin{array}{llllll}
0 & = 0 & = 0 & = 0 & \longrightarrow & 0 = (0000) \\
g^0 & = g^0 & = g^0 & = g^0 & \longrightarrow & g^0 = (0001) \\
g^1 & = g^1 & = g^1 & = g^1 & \longrightarrow & g^1 = (0010) \\
g^2 & = g^2 & = g^2 & = g^2 & \longrightarrow & g^2 = (0100) \\
g^3 & = g^3 & = g^3 & = g^3 & \longrightarrow & g^3 = (1000) \\
g^4 & = g^4 & = g^4 & = g^3 + 1 & \longrightarrow & g^4 = (1001) \\
g^5 & = g\,(g^4) & = g\,(g^3 + 1) & = g^3 + g + 1 & \longrightarrow & g^5 = (1011) \\
g^6 & = g\,(g^5) & = g\,(g^3 + g + 1) & = g^3 + g^2 + g + 1 & \longrightarrow & g^6 = (1111) \\
g^7 & = g\,(g^6) & = g\,(g^3 + g^2 + g + 1) & = g^2 + g + 1 & \longrightarrow & g^7 = (0111) \\
g^8 & = g\,(g^7) & = g\,(g^2 + g + 1) & = g^3 + g^2 + g & \longrightarrow & g^8 = (1110) \\
g^9 & = g\,(g^8) & = g\,(g^3 + g^2 + g) & = g^2 + 1 & \longrightarrow & g^9 = (0101) \\
g^{10} & = g\,(g^9) & = g\,(g^2 + 1) & = g^3 + g & \longrightarrow & g^{10} = (1010) \\
g^{11} & = g\,(g^{10}) & = g\,(g^3 + g) & = g^3 + g^2 + 1 & \longrightarrow & g^{11} = (1101) \\
g^{12} & = g\,(g^{11}) & = g\,(g^3 + g^2 + 1) & = g + 1 & \longrightarrow & g^{12} = (0011) \\
g^{13} & = g\,(g^{12}) & = g\,(g + 1) & = g^2 + g & \longrightarrow & g^{13} = (0110) \\
g^{14} & = g\,(g^{13}) & = g\,(g^2 + g) & = g^3 + g^2 & \longrightarrow & g^{14} = (1100)
\end{array}$$

22. The following shows a few examples of addition and subtraction. Note that modulus for the exponent is $2^n - 1$ or 15.

a. We show two examples of addition (using the results of Exercise 21):

$$\begin{array}{lcl}
g^3 + g^{12} = g^3 + (g + 1) = g^3 + g + 1 & \rightarrow & (1011) = (1000) + (0011) \\
g^{10} + g^{12} = (g^3 + g) + (g + 1) = g^3 + 1 & \rightarrow & (1001) = (1010) + (0011)
\end{array}$$

b. We show two examples of subtraction (using the results of Exercise 21):

$$\begin{array}{lcl}
g^3 - g^9 = g^3 - (g^2 + 1) = g^3 + g^2 + 1 & \rightarrow & (1101) = (1000) - (0101) \\
g^{10} - g^4 = (g^3 + g) - (g^3 + 1) = g + 1 & \rightarrow & (0011) = (1010) - (1001)
\end{array}$$

23.

a. We show two examples of multiplication (using the results of Exercise 21):

$$g^3 \times g^{12} = g^{15 \bmod 15} = g^0 = 1 \quad \rightarrow \quad (0001) = (1000) \times (0011)$$
$$g^{10} \times g^{12} = g^{22 \bmod 15} = g^7 = g^2 + g + 1 \quad \rightarrow \quad (0111) = (1010) + (0011)$$

b. We show two examples of division (using the results of Exercise 21):

$$g^3 \div g^9 = g^{-6 \bmod 15} = g^9 = g^2 + 1 \quad \rightarrow \quad (0101) = (1000) \div (0101)$$
$$g^{10} \div g^4 = g^{6 \bmod 15} = g^6 = g^3 + g^2 + g + 1 \quad \rightarrow \quad (1111) = (1010) - (1001)$$

24.

a. GF(12) is ***not*** a Galois field, because 12 cannot be written in the form p^n.

b. GF(13) is a Galois field; 13 can be written in the form p^n (p =13 and n = 1).

c. GF(16) is a Galois field; 16 can be written in the form p^n (p = 2 and n = 4).

d. GF(17) is a Galois field; 17 can be written in the form p^n (p =17 and n = 1).

25.

a. $x^4 + x$

b. x

c. $x^5 + 1$

d. $x + 1$

26.

a. 0101

b. 00101

c. 011

d. 10000000

27.

a. 5 + 3 = 8 mod 7 = 1 mod 7

b. 5 − 4 = 1 mod 7

c. 5 × 3 = 15 mod 7 = 1 mod 7

d. $5 \div 3 = 5 \times (3^{-1}) = 5 \times 5 = 25 \bmod 7 = 4 \bmod 7$

28. A polynomial $f(x)$ of degree n is irreducible if $f(x) = g(x) \times h(x)$, where g and h are two polynomials, each with the degree greater than zero. According to this definition we have **degree** (f) = **degree** (g) + **degree** (h). Based on this, we can prove that every polynomial of degree 1 is a reducible polynomial because the integer 1 cannot be sum of two integers greater than 0. Since the only two polynomials of degree 1 are $f_1(x) = x$ and $f_2(x) = x + 1$, these two polynomials are irreducible.

29. A polynomial $f(x)$ of degree n is irreducible if $f(x) = g(x) \times h(x)$, where g and h are two polynomials, each with the degree greater than zero. According to this definition we have **degree** (f) = **degree** (g) + **degree** (h). Based on this, a reducible polynomial of degree 2 can be factored only as two polynomials of degree 1 ($2 = 1 + 1$). In other words, a factors of a reducible polynomial of degree 2 can be only x or ($x + 1$) (the only two polynomials of degree 1). We can check all polynomials of degree 2 to see which one can be factored as such.

$(x^2) = (x) \times (x)$	$\rightarrow$	(x^2) is reducible
$(x^2 + 1) = (x + 1) \times (x + 1)$	$\rightarrow$	$(x^2 + 1)$ is reducible
$(x^2 + x) = (x) \times (x + 1)$	$\rightarrow$	$(x^2 + x)$ is reducible
$(x^2 + x + 1)$ cannot be factored.	$\rightarrow$	$(x^2 + x + 1)$ is irreducible

It can also be proved that $f(x) = x^2 + x + 1$ cannot be evenly divided by x or $x + 1$ because this implies that $x = 0$ or $x = -1$ must be the root of the $f(x)$, which are not ($f(0) = 1$ and $f(-1) = 1$.

30. A polynomial $f(x)$ of degree n is irreducible if $f(x) = g(x) \times h(x)$, where g and h are two polynomials, each with the degree greater than zero. According to this definition we have **degree** (f) = **degree** (g) + **degree** (h). Based on this, a reducible polynomial of degree 3 can be factored only as two polynomials of degree 1 and 2 ($3 = 1 + 2$). In other words, one of the factors of a reducible polynomial of degree 3 must be x or ($x + 1$) (the only two polynomials of degree 1). We can check all polynomials of degree 3 to see which one can be factored as such.

$(x^3) = (x) \times (x^2)$	$\rightarrow$	(x^3) is reducible
$(x^3 + 1) = (x + 1) \times (x^2 + x + 1)$	$\rightarrow$	$(x^3 + 1)$ is reducible
$(x^3 + x) = (x) \times (x^2 + 1)$	$\rightarrow$	$(x^3 + x)$ is reducible
$(x^3 + x + 1)$ cannot be factored	$\rightarrow$	$(x^3 + x + 1)$ is irreducible
$(x^3 + x^2) = (x^2) \times (x + 1)$	$\rightarrow$	$(x^3 + x^2)$ is reducible
$(x^3 + x^2 + 1)$ cannot be factored	$\rightarrow$	$(x^3 + x^2 + 1)$ is irreducible
$(x^3 + x^2 + x) = (x) \times (x^2 + x + 1)$	$\rightarrow$	$(x^3 + x^2 + x)$ is reducible
$(x^3 + x^2 + x + 1) = (x + 1) \times (x^2 + x + 1)$	$\rightarrow$	$(x^3 + x^2 + x + 1)$ is reducible

It is clear that $f_1(x) = x^3 + x + 1$ can not be factored as (x) or ($x + 1$) because neither 0 nor -1 are the root for this polynomial. This is true for $f_2(x) = x^3 + x^2 + 1$.

31. We first write each number as a polynomial with coefficient in GF(2). We then multiply the polynomials. Finally, we convert the result to the binary pattern.

a. $(x + 1) \times (x + 1) \rightarrow (x^2 + x + x + 1) \rightarrow (x^2 + 1) \rightarrow 101$

b. $(x^3 + x) \times (x^3) \rightarrow (x^6 + x^4) \rightarrow 1010000$

c. $(x^4 + x^3 + x^2) \times (x^4) \rightarrow (x^{16} + x^7 + x^6) \rightarrow 10000000011000000$

32. The only irreducible polynomial of degree 2 is $x^2 + x + 1$. We can guess that the inverse of 1 is 1 and the other two polynomials are inverses of each other, but we use the extended Euclidean algorithm to find them.

a. The inverse of 1 is 1, as shown in the following table.

q	r_1	r_2	r	t_1	t_2	t
$x^2 + x + 1$	$x^2 + x + 1$	1	0	0	1	1
	1	0		$\mathbf{1}$	1	

b. The inverse of x is $x + 1$, as shown in the following table.

q	r_1	r_2	r	t_1	t_2	t
$x + 1$	$x^2 + x + 1$	x	1	0	1	$x + 1$
x	x	1	0	1	$x + 1$	1
	1	0		$\mathbf{x + 1}$	1	

c. The inverse of $x + 1$ is x, as shown in the following table.

q	r_1	r_2	r	t_1	t_2	t
x	$x^2 + x + 1$	$x + 1$	1	0	1	x
$x + 1$	$x + 1$	1	0	1	x	1
	1	0		$\mathbf{x}$	1	

33. The inverse is $\mathbf{x^3 + x}$, as shown below (using the extended Euclidean algorithm).

q	r_1	r_2	r	t_1	t_2	t
$x + 1$	$x^5 + x^2 + 1$	$x^4 + x^3 + 1$	$x^3 + x^2 + x$	0	1	$x + 1$
x	$x^4 + x^3 + 1$	$x^3 + x^2 + x$	$x^2 + 1$	1	$x + 1$	$x^2 + x + 1$
$x + 1$	$x^3 + x^2 + x$	$x^2 + 1$	1	$x + 1$	$x^2 + x + 1$	$x^3 + x$
$x^2 + 1$	$x^2 + 1$	1	0	$x^2 + x + 1$	$x^3 + x$	1
	1	0		$\mathbf{x^3 + x}$	1	

34. We use the irreducible polynomial $\mathbf{(x^4 + x^3 + 1)}$.

a. The following shows the addition table. We have used hexadecimal values to make the table shorter.

⊕	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	D	C
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	D	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

b. The following shows the multiplication table. We have used hexadecimal values to make the table shorter.

⊗	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	9	B	D	F	1	3	5	7
3	0	3	6	5	C	F	A	9	1	2	7	4	D	E	B	8
4	0	4	8	C	9	D	1	5	B	F	3	7	2	6	A	E
5	0	5	A	F	D	8	7	2	3	6	9	C	E	D	4	1
6	0	6	C	A	1	7	D	B	2	4	E	8	3	5	F	9
7	0	7	E	9	5	2	B	C	A	D	4	3	F	8	1	6
8	0	8	9	1	B	3	2	A	F	7	6	E	4	C	D	5
9	0	9	B	2	F	6	4	D	7	E	C	5	8	1	3	A
A	0	A	D	7	3	9	E	4	6	C	B	1	5	F	8	2
B	0	B	F	4	7	C	8	3	E	5	1	A	9	2	6	D
C	0	C	1	D	2	E	3	F	4	8	5	9	6	A	7	B
D	0	D	3	E	6	B	5	8	C	1	F	2	A	7	9	4
E	0	E	5	B	A	4	F	1	D	3	8	6	7	9	2	C
F	0	F	7	8	E	1	9	6	5	A	2	D	B	4	C	3

35. We use Table 4.10 to find the multiplicative inverse of the second word. We then use the same table to multiply the first word with the inverse of the second word.

a. $(100) \div (010) = (100) \times (010)^{-1} = (100) \times \mathbf{(110)} = (010)$

b. $(100) \div (000) \rightarrow$ This operation is impossible because (000) has no inverse.

c. $(101) \div (011) = (101) \times (011)^{-1} = (101) \times \mathbf{(100)} = (011)$

d. $(000) \div (111) = (000) \times (111)^{-1} = (000) \times \mathbf{(101)} = (111)$

36. We let $P_1 = (x^2 + 1)$, $P_2 = (x^3 + x^2 + x + 1)$, and modulus $= (x^4 + x^3 + 1)$. The following table shows the process. Note that we only need to add the modulus with the new result (if the degree of the result is greater than the degree of the modulus); no division is needed.

Powers	***Operation***	***New Result***	***Reduction***
$\mathbf{x^0 \otimes P_2}$		$\mathbf{x^3 + x^2 + x + 1}$	**No**
$x^1 \otimes P_2$	$x \otimes (x^3 + x^2 + x + 1)$	$x^2 + x + 1$	Yes
$\mathbf{x^2 \otimes P_2}$	$\mathbf{x \otimes (x^2 + x + 1)}$	$\mathbf{x^3 + x^2 + x}$	**No**
$\mathbf{P_1 \otimes P_2 = (x^0 \otimes P_2) \oplus (x^2 \otimes P_2) = (x^3 + x^2 + x + 1) \oplus (x^3 + x^2 + x) = (1)}$			

The result is **(1)**, which can be proved using multiplication and division by the modulus.

37. We let $P_1 = (10000)$, $P_2 = (10101)$, and modulus $= (100101)$. The following table shows the process:

Powers	***Shift-Let Operation***	***Exclusive-Or***
$x^0 \otimes P_2$		10101
$x^1 \otimes P_2$	01010	$01010 \oplus 00101 = 01111$
$x^2 \otimes P_2$	11110	11110
$x^3 \otimes P_2$	11100	$11100 \oplus 00101 = 11001$
$\mathbf{x^4 \otimes P_2}$	**10010**	$\mathbf{10010 \oplus 00101 = 10111}$
$\mathbf{P_1 \otimes P_2 = (x^4 \otimes P_2) = 10111}$		

The result is **(10111)** or $\mathbf{(x^4 + x^2 + x + 1)}$, which can be proved using multiplication and division by the modulus.

CHAPTER 5

Introduction to Modern Symmetric-Key Ciphers

(Solution to Practice Set)

Review Questions

1. The traditional symmetric-key ciphers are character-oriented ciphers. The modern symmetric-key ciphers are bit-oriented ciphers.

2. To be resistant to exhaustive-search attack, a modern block cipher needs to be designed as a substitution cipher because in a transposition cipher we have the same number of 1s in the plaintext and ciphertext, which makes exhaustive-search attack simpler.

3. A transposition is definitely a permutation of bits. A substitution of bits can be thought of a permutation if we add decoding and encoding to the operation.

4. We mentioned several components of a modern block ciphers in this chapter: P-boxes, S-boxes, the exclusive-or operation, the swap operation, the circular shift operation, the split operation, and the combine operation.

5. A P-box (permutation box) transposes bits. We have three types of P-boxes in modern block ciphers: straight P-boxes, expansion P-boxes, and compression P-boxes. A straight P-box is invertible; the other two are not.

6. An S-box is an $m \times n$ substitution unit, where m and n are not necessarily the same. The necessary condition for invertibility is that m should be equal to n.

7. A product cipher is a complex cipher combining substitution, permutation, and other components discussed in this chapter. We discussed two classes of product ciphers: Feistel and non-Feistel ciphers.

8. Diffusion hides the relationship between the ciphertext and the plaintext. Confusion hides the relationship between the ciphertext and the key.

9. A Feistel block cipher uses both invertible and noninvertible components. A non-Feistel block cipher uses only invertible components.

10. A differential cryptanalysis is based on a nonuniform differential distribution table of the S-boxes in a block cipher; it is a chosen-plaintext attack. A linear cryptanal-

ysis is based on a linear approximation of an S-box; it is a known-plaintext attack.

11. In a synchronous stream cipher the key stream is independent of the plaintext or ciphertext. In a nonsynchronous stream cipher the key stream is somehow dependent on the plaintext or ciphertext.

12. A feedback shift register is made of a shift register and a feedback function. We discussed two variations: linear feedback shift register (LFSR) and nonlinear feedback shift register (NLFSR).

Exercises

13. The order of the group is 10! = 3,626,800. The key size is $\lceil \log_2(10!) \rceil = 22$ bits. Note that a key of 22 bits can select 2^{22} = 4,194,304 different permutations, but only 3,626,800 of them are used here.

14. The order of the group is $(2^{10})! = 1024!$ The key size is $\lceil \log_2 (1024!) \rceil = 8770$ bits. Note that a key of 8770 bits can select 2^{8770} different permutations, but only (1024!) of them are used here.

15.

a. $\text{CircularLeftShift}_3(\mathbf{100}11011) \rightarrow 11011\mathbf{100}$

b. $\text{CircularRightShift}_3(11011\mathbf{100}) \rightarrow \mathbf{100}11011$

c. The original word in Part *a* and the result of Part *b* are the same, which shows that circular left shift and circular right shift operations are inverses of each other.

16.

a. *Swap* (**10011011**) → **10111001**

b. *Swap* (**10111001**) → **10011011**

c. The original word in Part *a* and the result of Part *b* are the same, which shows that swapping is a self-invertible operation.

17. We show the complement of A with $\overline{A}$.

a. $(01001101) \oplus (01001101) = (00000000) \rightarrow (A \oplus A = \text{All 0s})$

b. $(01001101) \oplus (10110010) = (11111111) \rightarrow (A \oplus \overline{A} = \text{All 1s})$

c. $(01001101) \oplus (00000000) = (01001101) \rightarrow (A \oplus \text{All 0s} = A)$

d. $(01001101) \oplus (11111111) = (10110010) \rightarrow (A \oplus \text{All 1s} = \overline{A})$

18.

a. The value of $010_2 = 2$, which can be decoded as the 8-bit word (0000**1**00**)**.

b. The 8-bit word (00**1**00000) has a 1 in position 5, which can be encoded as $(101)_2$.

19. Using eight bits for each character, |M| = 8 × 2000 = 16,000 bits. Therefore, we have

$$\mathbf{|M| + |Pad| = 0 \bmod 64 \rightarrow |Pad| = -|M| \bmod 64 \rightarrow |Pad| = -16{,}000 \bmod 64 = 0}$$

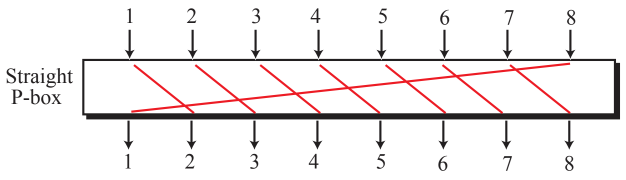
This means no padding is needed. The message is divided into 250 blocks.

20. The permutation table is [2 5 4 1 3].

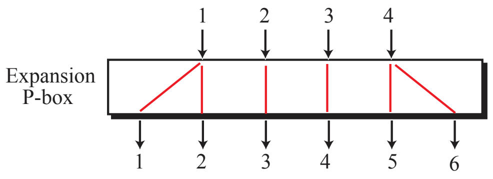
21. The permutation table is [1 3 5].

22. The permutation table is [1 3 3 1 2].

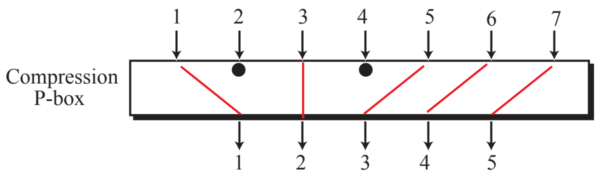
23. See the figure below:



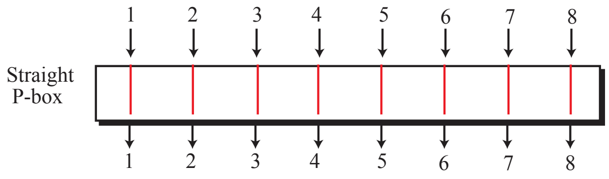
24. It is an expansion P-box with 4 inputs and 6 outputs as shown below:



25. It is a compression P-box with 7 inputs and 5 outputs as shown below:



26. It is a straight P-box with 6 inputs and 6 outputs as shown below:



27. We use the following procedure:

a. We first find the input/output relation based on the given S-box:

Input:	00	01	10	11
Output:	01	11	10	00

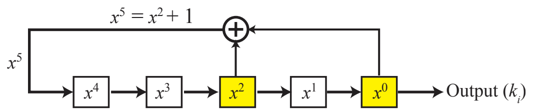
b. We then find the inverse input/output relation (sorted on input):

Input:	00	01	10	11
Output:	11	00	10	01

c. Now we create the table for the inverse S-box (the left row defines the first input bit, the first column defines the second input bit, and the entries define the output):

	0	1
0	11	00
1	10	01

28. The characteristic polynomial is $x^5 + x^2 + 1 = 0$. The feedback function can be found as $x^5 = x^2 + 1$. The following figure shows the LFSR.



The characteristic polynomial $x^5 + x^2 + 1$ or $(25)_{16}$ is both a primitive polynomial (see Appendix G). However, the corresponding number of cells is 5, which is odd. This means that period is less than 31 ($2^m - 1$).

29. The characteristic polynomial is $x^4 + x^3 + x^2 + 1$ or $(11101)_2$ or $(1D)_{16}$. The polynomial is not primitive (see Appendix G). The maximum period is then less than $2^4 - 1$ or less than 15.

30. The following table shows the initial state (seed) and the next twenty states.

States	b_4	b_3	b_2	b_1	b_0	k_i
Initial	**1**	1	1	1	0	
01	**0**	1	1	1	1	**0**
02	**0**	0	1	1	1	**1**
03	**0**	0	0	1	1	**1**
04	**1**	0	0	0	1	**1**
05	**0**	1	0	0	0	**1**
06	**0**	0	1	0	0	**0**
07	**1**	0	0	1	0	**0**
08	**1**	1	0	0	1	**0**
09	**0**	1	1	0	0	**1**
10	**1**	0	1	1	0	**0**
11	**0**	1	0	1	1	**0**
12	**1**	0	1	0	1	**1**
13	**1**	1	0	1	0	**1**
14	**1**	1	1	0	1	**0**
15	**1**	1	1	1	0	**1**
16	**0**	1	1	1	1	**0**
17	**0**	0	1	1	1	**1**
18	**0**	0	0	1	1	**1**
19	**1**	0	0	0	1	**1**
20	**0**	1	0	0	0	**1**

The initial state in the above table is the same as the state 11 in Table 5.6. The next 9 states are also the same; state 09 in the above table is the same as the state 20 in Table 5.6. The rest of the states are different in the two tables.

31. To have a maximum period of 32, the characteristic polynomial should be of degree 6 because $2^5 - 1 = 31 < 32$. However, if the characteristic polynomial is primitive, the maximum period is $2^6 - 1 = 63$. But the problem says that the maximum period is 32, therefore, the characteristic polynomial is not primitive. In other words, we have an LFSR of degree 6, with 6 cells (6-bit register) whose characteristic polynomial is not primitive.

32. We assume that bits are numbered b_0 to b_5 from right to left. Other assumptions yield different results.

Input: **110010** $\rightarrow$ Left bit $= \mathbf{1} \oplus \mathbf{0} \oplus \mathbf{1} = \mathbf{0}$ Right bit $= \mathbf{1} \oplus \mathbf{0} \oplus \mathbf{0} = \mathbf{1}$ $\rightarrow$ Output: 01

Input: **101101** $\rightarrow$ Left bit $= \mathbf{1} \oplus \mathbf{1} \oplus \mathbf{0} = \mathbf{0}$ Right bit $= \mathbf{0} \oplus \mathbf{1} \oplus \mathbf{1} = \mathbf{0}$ $\rightarrow$ Output: 00

33.

Input: **1**011 $\rightarrow$ Left rotate $\rightarrow$ Output: 110

Input: **0**110 $\rightarrow$ Right rotate $\rightarrow$ Output: 011

34.

```
split (word [0 … n], n)
{
        $i \leftarrow 1$
        while ($i \leq n / 2$)
        {
                rightWord[$i$] $\leftarrow$ word[$i$]
                leftWord[$i$] $\leftarrow$ word[$i + n / 2$]
                $i \leftarrow i + 1$
        }
        return (rightWord, leftWord)
}
```

35.

```
combine (rightWord [1 … m], leftWord [1 … m], m)
{
        $i \leftarrow 1$
        while ($i \leq m$)
        {
                word[$i$] $\leftarrow$ rigthWord[$i$]
                word[$i + m$] $\leftarrow$ leftWord[$i$]
                $i \leftarrow i + 1$
        }
        return (word[1 … n])                          // $n = 2m$
}
```

36.

```
swap (word [1 … n], n)
{
        $i \leftarrow 1$
        while ($i \leq n / 2$)
        {
                temp[$i + n / 2$] $\leftarrow$ word[$i$]
                temp[$i$] $\leftarrow$ word[$i + n / 2$]
                $i \leftarrow i + 1$
        }
        return (temp[0 … n])
}
```

37.

```
circularShift (shift, word [1 … n], n, k)
{
        i ← 1
        while (i ≤ k)
        {
                if (shift = left)
                        word ← circularShiftLeft(word, n)
                else
                        word ← circularShiftRight(word, n)
                i ← i + 1
        }
        return (word[1 … n])
}
circularShiftLeft (word [1 … n], n)
{
        temp ← word[n]
        j ← n − 1
        while (j ≥ 0)
        {
                word[j +1] ← word[j]
                j ← j − 1
        }
        word[1] ← temp
        return (word[0 … n])
}
circularShiftRight (word [0 … n], n)
{
        temp ← word[1]
        j ← 1
        while (j ≤ n)
        {
                word [j −1] ← word[j]
                j ← j + 1
        }
        word[n] ← temp
        return (word[0 … n])
}
```

38.

```
P-box (inputBits [1 … n], Table [1 … m], n, m)
{
        $i \leftarrow 1$
        while ($i \le m$)
        {
                outputBits[i] ← inputBits[Table[i]]
                $i \leftarrow i + 1$
        }
        return (outputBits[1 … n])
}
```

39. The table can be designed in many different ways. We assume, we have a linear table of *n* cells, in which each cell contains a value of *m* bits. The input defines the cell number, the contents of the cell defines the output. With this configuration, the routine looks like the one shown below:

```
S-box (inputBits [1 … n], Table [1 … n], n, m)
{
        index ← binaryToDecimal (inputBits)
        value ← Table [index]
        outputBits ← decimalToBinary (value)
        return (outputBits [0 … m])
}
```

40. We assume that the key mixer apply the XOR operation on the input and the round key. The following shows the general idea.

```
Non_FeistelRound (inputBits [1 … n], roundKey[1 … n], n, PermuteTable)
{
        temp ← inputBits ⊕ roundKey
        temp ← substitute(temp, SubstituteTables)
        outputBits ← permute(temp, PermuteTable)
        return (outputBits [1 … n])
}
```

41.

```
FeistelRound (inputBits [1 … n], roundKey[1 … n], n)
{
        (tempLeft, tempRight) ← split (word, n)
        tempLeft ← tempLeft ⊕ function (tempRight, roundKey)
        (tempLeft, tempRight) ← swap (tempLeft, tempRight)
        outputBits ← combine(tempLeft, tempRight)
        return (outputBits [1 … n])
}
```

42.

```
LFSR (initialState [0 … n −1], n)
{
        b[n] ← f (initialState)
        i ← n
        while (i > 0)
        {
                b[i − 1] ← b[i]
                i ← i − 1
        }
        return (b[0])
}
```

CHAPTER 6

Data Encryption Standard (DES)

(Solution to Practice Set)

Review Questions

1. The block size in DES is 64 bits. The cipher key size is 56 bits. The round key size is 48 bits.
2. DES uses 16 rounds.
3. In the first approach, DES uses 16 mixers and 15 swappers in encryption or decryption algorithm; in the second (alternative approach), DES use 16 mixers and 16 swappers in encryption or decryption algorithm.
4. In DES, encryption or decryption uses $16 \times 2 + 2 = 34$ permutations, because each mixer uses two permutations and there are two permutations before and after the rounds. The round-key generator uses 17 permutation operations: one parity drop and 16 compression permutation operations for each round.
5. The total number of exclusive-or operations is $16 \times 2 = 32$, because each round uses two exclusive-or operations (one inside the function and one outside of the function).
6. The input to the function is a 32-bit word, but the round-key is a 48-bit word. The expansion permutation is needed to increase the number of bits in the input word to 48.
7. The cipher key that is used for DES include the parity bits. To remove the parity bits and create a 56-bit cipher key, a parity drop permutation is needed. Not only does the parity-drop permutation drop the parity bits, it also permutes the rest of the bits.
8. A weak key is the one that, after parity drop operation, consists either of all 0s, all 1s, or half 0s and half 1s. Each weak key is the inverse of itself: $E_k(E_k(P)) = P$. A semi-weak key creates only two different round keys and each of them is repeated eight times. The semi-weak round keys come in pairs, where a key in the pair is the inverse of the other key in the pair: $E_{k1}(E_{k2}(P)) = P$. A possible weak key is a key that creates only four distinct round keys; in other words, the sixteen round keys are divided into four groups and each group is made of four equal round keys.

9. Double DES uses two instances of DES ciphers for encryption and two instances of reverse ciphers for decryption. Each instance uses a different key, which means that the size of the key is 112 bits. However, double DES is vulnerable to meet-in-the-middle attack.

10. Triple DES uses three stages of DES for encryption and decryption. Two versions of triple DES are in use today: triple DES with two keys and triple DES with three keys. In triple DES with two keys, there are only two keys: K_1 and K_2. The first and the third stages use K_1; the second stage uses K_2. In triple DES with three keys, there are three keys: K_1, K_2, and K_3.

Exercises

11.

a.

Input: 1 1011 1 → **3, 11** → **Output: 03 (0011)**

b.

Input: 0 0110 0 → **0, 6** → **Output: 09 (1001)**

c.

Input: 0 0000 0 → **0, 0** → **Output: 04 (0100)**

d.

Input: 1 1111 1 → **3, 15** → **Output: 09 (1001)**

12. The following table shows the output from all boxes. No pattern can be found:

Input	*Box 1*	*Box 2*	*Box 3*	*Box 4*	*Box 5*	*Box 6*	*Box 7*	*Box 8*
000000	**1110**	**1111**	**1010**	**0111**	**0010**	**1100**	**0100**	**1101**

13. The following table shows the output from all boxes. No pattern can be found:

Input	*Box 1*	*Box 2*	*Box 3*	*Box 4*	*Box 5*	*Box 6*	*Box 7*	*Box 8*
111111	**1101**	**1001**	**1100**	**1110**	**0011**	**0011**	**1101**	**1011**

14.

a. The following shows that 3 bits will be changed in the output.

Input: 0 0000 0	→	**00, 00**	→	**Output: 10 (1010)**
Input: 0 0000 1	→	**01, 00**	→	**Output: 13 (1101)**

b. The following shows that 2 bits will be changed in the output.

Input: 1 1111 1	→	**03, 15**	→	**Output: 09 (1001)**
Input: 1 1101 1	→	**03, 14**	→	**Output: 14 (1100)**

15.

a. The following shows that 2 bits will be changed in the output.

Input: 0 0110 0	→	**00, 06**	→	**Output: 03 (0011)**
Input: 0 0000 0	→	**00, 00**	→	**Output: 15 (1111)**

b. The following shows that 2 bits will be changed in the output.

Input: 1 1001 1	→	**03, 09**	→	**Output: 06 (0110)**
Input: 1 1111 1	→	**03, 15**	→	**Output: 09 (1001)**

16.

a. The following shows that two outputs are different.

Input: 0 0110 0	→	**00, 06**	→	**Output: 09 (1001)**
Input: 1 1000 0	→	**02, 08**	→	**Output: 15 (1111)**

b. The following shows that two outputs are different.

Input: 1 1001 1	→	**03, 09**	→	**Output: 04 (0100)**
Input: 0 0111 1	→	**01, 07**	→	**Output: 03 (0011)**

17. The following table shows 32 input pairs and 32 output pairs. The last column is the difference between the outputs.

Input pairs		*Output pairs*		*d*
000000	000001	0010	1110	**1100**
000010	000011	1100	1011	**0111**
000100	000101	0100	0010	**0110**
000110	000111	0001	1100	**1111**
001000	001001	0111	0100	**0011**
001010	001011	1010	0111	**1001**
001100	001101	1011	1101	**0110**
001110	001111	0110	0001	**0111**
010000	010001	1000	0101	**1101**
010010	010011	0101	0000	**0101**
010100	010101	0011	1111	**1100**
010110	010111	1111	1010	**0101**
011000	011001	1101	0011	**1110**
011010	011011	0000	1001	**1001**
011100	011101	1110	1000	**0110**
011110	011111	1001	0110	**1111**
100000	100001	0110	1011	**1101**
100010	100011	0010	1000	**1010**
100100	100101	0001	1100	**1101**
100110	100111	1011	0111	**1100**
101000	101001	1010	0001	**1011**
101010	101011	1101	1110	**0011**
101100	101101	0111	0010	**0101**
101110	101111	1000	1101	**0101**
110000	110001	1111	0110	**1001**
110010	110011	1001	1111	**0110**
110100	110101	1100	0000	**1100**
110110	110111	0101	1001	**1100**
111000	111001	0110	1010	**1100**
111010	111011	0011	0100	**0111**
111100	111101	0000	0101	**0101**
111110	111111	1110	0011	**1101**

If we sort the table on last column (output differences), we get the following: two (0011)'s, five (0101)'s, four (0110)'s, three (0111)'s, three (0111)'s, one (1010), one (1011), one (1100), five (1100)'s, four (1101)'s, one (1110), and two (1111)'s. **None of the group has a size larger than eight.**

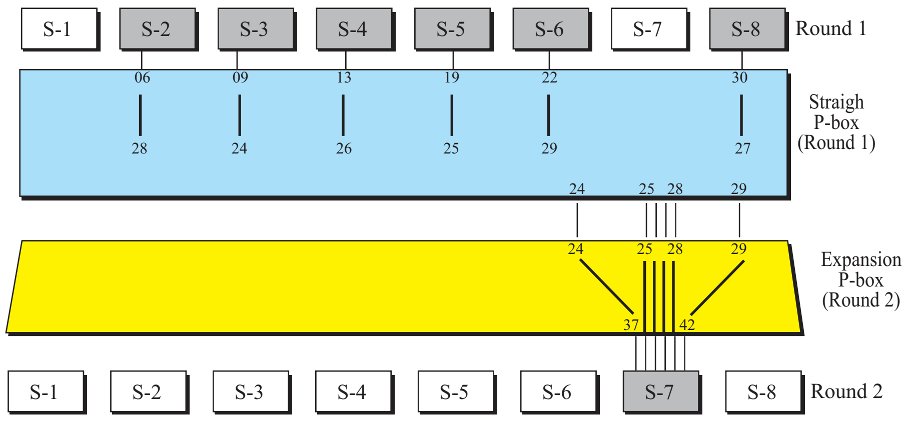
18. For S-Box 7, we set the first (leftmost) input bit to 0. We change the other five input bits. We then observe one of the output bits (we have chosen the third bit from the let) and count how many 0's and how may 1's we obtain for this bit. These two count must be close to each other. The following table shows inputs and outputs.

Input	*Output*	*Third bit*
0 0000 0	0 1 **0** 0	0
0 0000 1	1 1 **0** 1	0
0 0001 0	1 0 **1** 1	1
0 0001 1	0 0 **0** 0	0
0 0010 0	0 0 **1** 0	1
0 0010 1	1 0 **1** 1	1
0 0011 0	1 1 **1** 0	1
0 0011 1	0 1 **1** 1	1
0 0100 0	1 1 **1** 1	1
0 0100 1	0 1 **0** 0	0
0 0101 0	0 0 **0** 0	0
0 0101 1	1 0 **0** 0	0
0 0110 0	1 0 **0** 0	0
0 0110 1	0 0 **0** 1	0
0 0111 0	1 1 **0** 1	0
0 0111 1	1 0 **1** 0	1
0 1000 0	0 0 **1** 1	1
0 1000 1	1 1 **1** 0	1
0 1001 0	1 1 **0** 0	0
0 1001 1	0 0 **1** 1	1
0 1010 0	1 0 **0** 1	0
0 1010 1	0 1 **0** 1	0
0 1011 0	0 1 **1** 1	1
0 1011 1	1 1 **0** 0	0
0 1100 0	0 1 **0** 1	0
0 1100 1	0 0 **1** 0	1
0 1101 0	1 0 **1** 0	1
0 1101 1	1 1 **1** 1	1
0 1110 0	0 0 **0** 0	0
0 1110 1	1 0 **0** 0	0
0 1111 0	0 0 **0** 1	0
0 1111 1	0 1 **1** 0	1

As the table shows we have **17 0's** and **15 1's**; close enough.

19. Figure S6.19 shows the situation. The inputs to S-box 7 in round 2 comes from six different S-boxes in round 1.

Figure S6.19 *Solution to Exercise 19*



a. The six inputs to S-box 7 in round 2 come from six outputs (37, 38, 39, 40, 41, 42) of expansion permutation box.

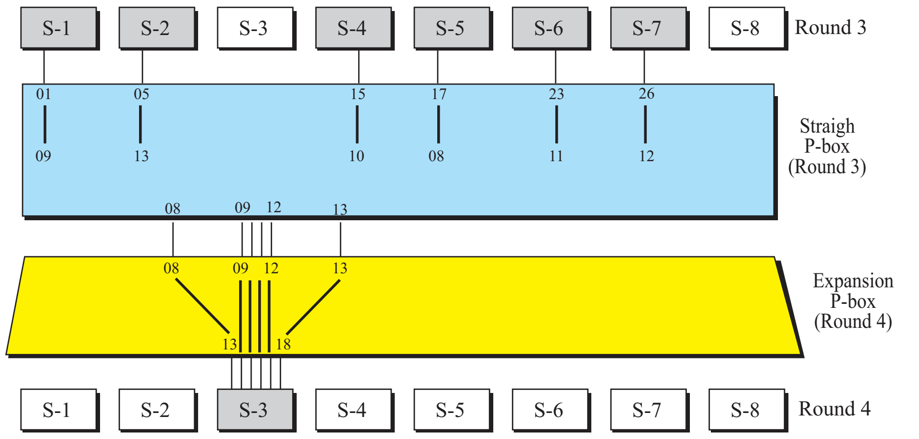
b. The above six outputs correspond to the six inputs (24, 25, 26, 27, 28, 29) in the expansion permutation box (See Table 6.2 in the textbook).

c. The above six inputs correspond to the six inputs (09, 19, 13, 30, 06, 22) inputs in the straight permutation box (See Table 6.11 in the textbook).

d. The above six inputs correspond to the outputs of six different S-Boxes (S-3, S-5, S-4, S-8, S-2, and S-6) in round 1.

20. Figure S6.20 shows the situation. The inputs to S-box 7 in round 2 comes from six S-boxes in round 1.

Figure S6.20 *Solution to Exercise 20*



a. The six inputs to S-box 3 in round 4 come from six outputs (13, 14, 15, 16, 17, 18) of expansion permutation box in round 4.

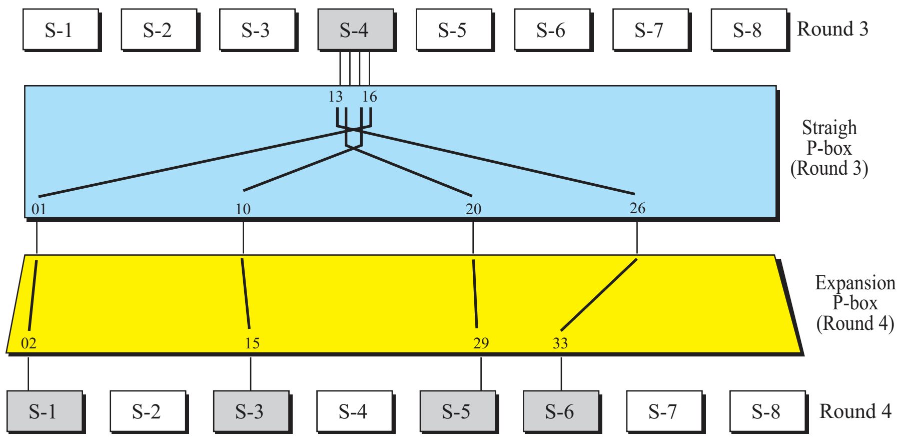
b. The above six outputs corresponds to the six inputs (08, 09, 10, 11, 12, 13) in the expansion permutation box in round 4 (See Table 6.2 in the textbook).

c. The above six inputs correspond to the six inputs (17, 01, 15, 23, 26, 05) inputs in the straight permutation box (See Table 6.11 in the textbook).

d. The above six inputs correspond to the outputs of six different S-Boxes (S-5, S-1, S-4, S-6, S-7, and S-2) in round 3. None of them come from S-3.

21. Figure S6.21 shows the situation. The outputs from S-box 4 in round 3 go to four S-boxes in round 4.

Figure S6.21 *Solution to Exercise 21*



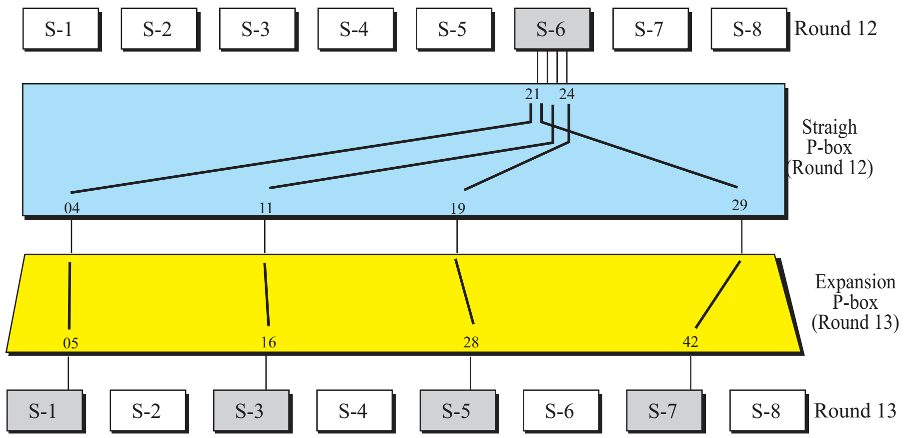
a. The four outputs from S-box 3 in round 4 go to six outputs (26, 20, 10, 01) of straight permutation box (See Table 6.1).

b. The above four outputs correspond to four outputs (33, 29, 15, 02) in the expansion permutation box (See Table 6.11).

c. The above four inputs corresponds to the inputs of four different S-Boxes (S-6, S-5, S-3, and S-1) in round 4.

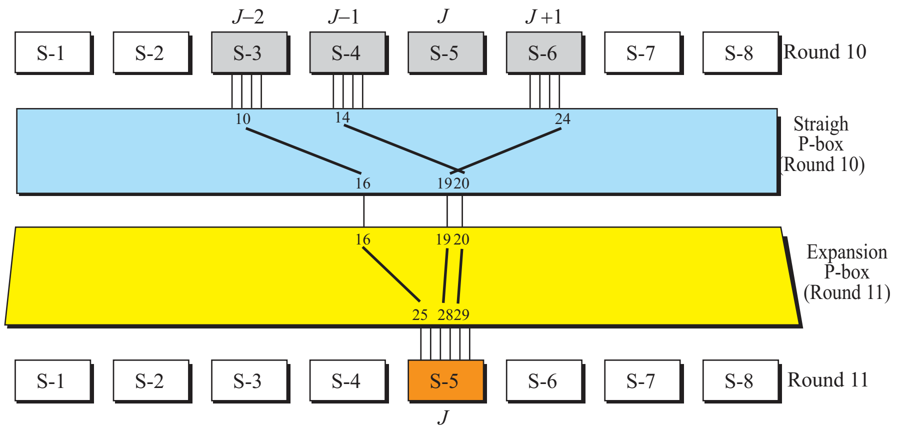
22. Figure S6.22 shows the situation. The outputs from S-box 6 in round 12 goes to four different S-boxes in round 13. None of them go to S-box 6. So the criterion cannot actually be tested by these exercise, but it does not violate the criterion either.

Figure S6.22 *Solution to Exercise 22*



23. Figure S6.23 shows the situation. We assume *j* = 5.

Figure S6.23 *Solution to Exercise 23*



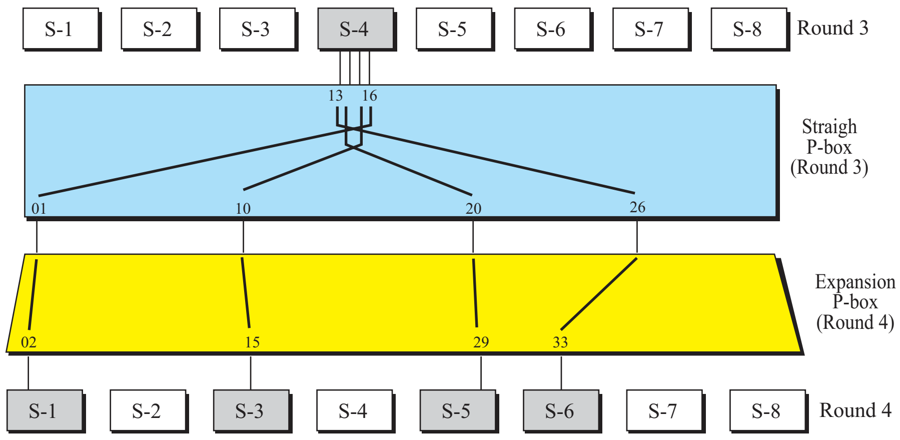
a. One output from S-box 3 (output 10) goes to the first input of S-box 5 (input 25).

b. An output from S-box 4 (output 14) goes one the last two inputs of S-box 6 (input 29).

c. An output from S-box 6 (output 24) goes to one of the middle input of S-box 5 (input 19).

24. The solution can be found in Figure S6.24. The outputs of S-4 is distributed between S-1, S-3, S-5, and S-7 in the next round.

Figure S6.24 *Solution to Exercise 24*



a. The output 16 goes to bit 2 (one of the first two bits of S-box 1).

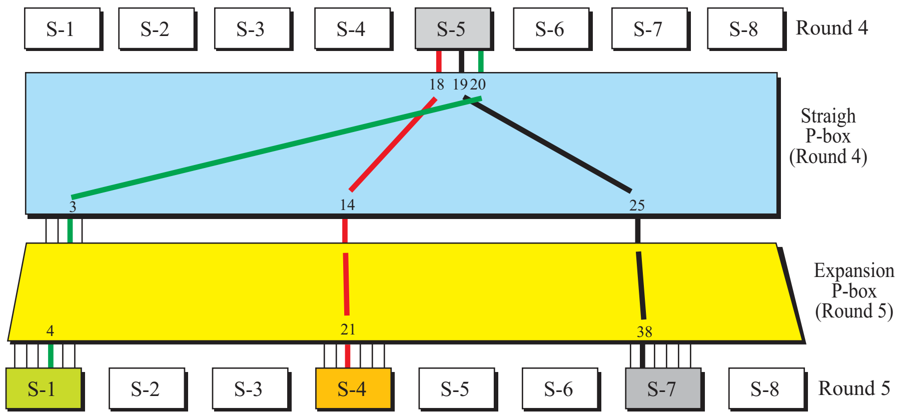
b. The output 14 goes to bit 29 (the last bit of S-box 5).

c. The output 15 goes to bit 15 (one of middle bit of S-box 3).

d. The output 13 goes to bit 33 (one of the middle bit of S-6).

25. Figure S6.25 will help us in this problem.

Figure S6.25 *Solution to Exercise 25*



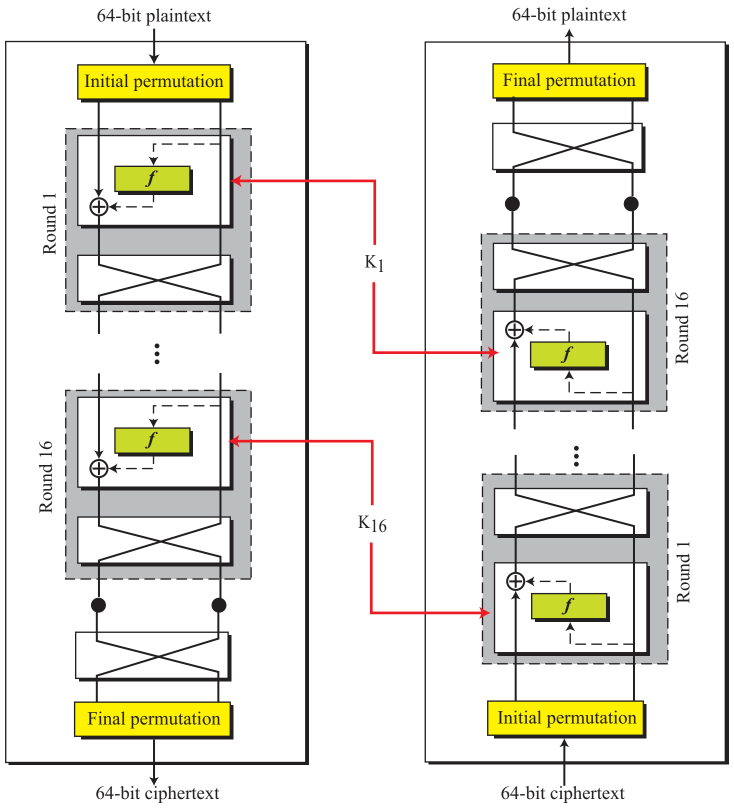
a. Only output 19 form S-box 5 (in round 4) goes to S-box 7 (in round 5). However, the input 38 is not a middle input, so the criterion does not apply. This answers the question about this exercise, but we do some more investigations.

b. Output 18 from S-box 5 goes a middle input (21) in S-box 4 in the next round. To check the criteria, we need to see if any input from S-box 4 goes to a middle input in next round. Looking at Figure S6.24, we can see that this is not the case. The criterion applies here.

c. We also observe that output 19 from S-box 5 goes a middle input in S-box 1 (input 4). However, none of the inputs from S-box 1 in round 4 goes to a middle input of S-box 5 in the next round. Only one output from S-box 1 (output 2 goes to S-box 5, input 26, but it is not a middle input). The criterion applies here.

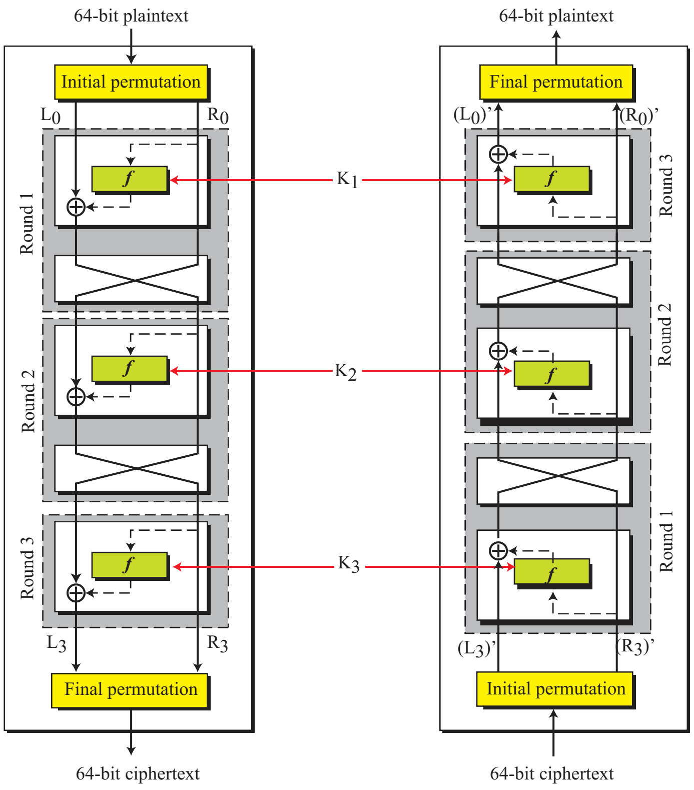
26. Figure S6.26 shows the alternative approach.

Figure S6.26 *Solution to Exercise 26*



27. Figure S6.27 shows a three-round cipher. We prove the equalities between the L's and R's from bottom to top. We have labeled each left section in the encryption L_i and in the decryption (L_i)'; we have labeled each right sections R_i in the encrypting and (R_i)' in decryption.

Figure S6.27 *Solution to Exercise 27*



a. Since final permutation and initial permutations are inverse of each other (if there is no corruption during transmission), we have

$\mathbf{(L_3)' = (L_3)}$	$\mathbf{(R_3)' = (R_3)}$

b. Using the previous equalities and the relations in the mixers, we have

$(L_2)' = (R_3)' = R_3 = R_2$	$(R_2)' = (L_3)' \oplus \boldsymbol{f}[(R_3)', K_3]$
	$(R_2)' = L_3 \oplus \boldsymbol{f}[R_3, K_3]$
	$(R_2)' = L_2 \oplus \boldsymbol{f}[R_2, K_3] \oplus \boldsymbol{f}[R_2, K_3]$
$\mathbf{(L_2)' = R_2}$	$\mathbf{(R_2)' = L_2}$

c. Using the previous equalities and the relations in the mixers, we have

$(L_1)' = (R_2)' = L_2 = R_1$

$(L_1)' = R_1$

$(R_1)' = R_2 \oplus f[L_2, K_2]$
$(R_1)' = R_2 \oplus f[R_1, K_2]$
$(R_1)' = L_1 \oplus f[R_1, K_1] \oplus f[R_1, K_1]$
$(R_1)' = L_1$

d. Using the previous equalities and the relations in the mixers, we have

$(L_0)' = (L_1)' \oplus f[(R_1)', K_1]$
$(L_0)' = R_1 \oplus f[L_1, K_1]$
$(L_0)' = L_0 \oplus f[L_0, K_1] \oplus f[L_0, K_1]$
$(L_1)' = L_0$

$(R_0)' = (R_1)' = L_1 = R_0$

$(R_0)' = R_0$

We have proved that $(L_1)' = L_0$ and $(R_0)' = R_0$. Since the final permutation and initial permutation are inverse of each other, the plaintext created at the destination is the same as the plaintext started at the source.

28. If we create a table of input and output, we can answers the three questions.

In	Out	In	Out
14	01	23	13
17	02	19	14
11	03	12	15
24	04	04	16
01	05	26	17
05	06	08	18
03	07	16	19
28	08	07	20
15	09	27	21
06	10	20	22
21	11	13	23
10	12	02	24

In	Out	In	Out
41	25	44	37
52	26	49	38
31	27	39	39
37	28	56	40
47	29	34	41
55	30	53	42
30	31	46	43
40	32	42	44
51	33	50	45
45	34	36	46
33	35	29	47
48	36	32	48

a. The missing inputs are 09, 18, 22, 25, 35, 38, 43 and 54.

b. From above table, we can see that the left 24 bits come from the left 28 bits (except bits 09, 18, 22, and 25, which are blocked).

c. From the above table, we can see the right 24 bits come from the right 28 bits (except bits 35, 38, 43, and 54, which are blocked).

29. The following shows the result:

$$(1066\ 0099\ 0088\ 0088)_{16}$$

30. The following shows the result:

$$(\text{0F55 AAFF 0F55 AAFF})_{16}$$

31. The first round key is

$$(\text{1437 4013 3784})_{16}$$

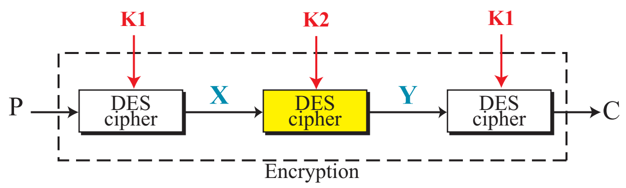
32. The following shows the effect:

	Originals				Complements			
Plaintext:	0000	0000	0000	0000	FFFF	FFFF	FFFF	FFFF
Key:	0000	0000	0000	0000	FFFF	FFFF	FFFF	FFFF
Ciphertext:	0808	02AA	AA02	A8AA	F7F7	FD55	55FD	5755

When we complement the plaintext and the key, the ciphertext is complemented.

33. Figure S6.33 shows the encryption using 3DES with two keys, in which X or Y are the intermediate texts.

Figure S6.33 *Solution to Exercise 33*



Van Oorschot and Wiener have devised a meet-in-the-middle attack on the above configuration. The attack is basically a known-plaintext attack. It follows the steps shown below:

a. Eve intercepts *n* plaintext/ciphertext pairs and stores them in a table which is sorted on values of P as shown below:

Plaintext	*Ciphertext*
P_1	C_1
P_2	C_2
…	…
P_n	C_n

Table 1: *n* P/C pairs

b. Eve now chooses a value for X (see Figure S6.33) and uses the decryption algorithm, and all 2^{56} possible K1's values to create 2^{56} different P values as shown below:

$$P_1 = D(K1_1, X) \quad P_2 = D(K1_2, X) \quad \ldots \quad P_m = D(K1_m, X) \qquad \text{where } m = 2^{56}$$

c. If a value of P created in step *b* matches one of the value of P in Table 1, Eve uses the corresponding value for K1 (from the list in step b) and the value of C from Table 1 and calculates a value for second intermediate text Y = D (K1, C). Now Eve creates a second table, Table 2, which is sorted on the value of Y. Eve has now *r* possible candidate for K1 keys.

Y	K1
Y_1	$K1_1$
Y_2	$K1_2$
…	…
Y_r	$K1_r$

Table 2: *r* Y/K1 pairs

d. Eve now searches for K2. For each 2^{56} possible values of K2, Eve uses the decryption algorithm and the value of X chosen in step *b* to create 2^{56} different values for Y's.

$$Y_1 = D(K2_1, X) \quad Y_2 = D(K2_2, X) \quad \ldots \quad Y_m = D(K2_m, X) \qquad \text{where } m = 2^{56}$$

If a value of Y_i created in this step matches one of the value in Table 2, Eve have found a pair of keys: K1 is extracted from Table 2 and K2 is extracted from the decryption algorithm that matches the value of Y in Table 2.

e. Now Eve tests pairs of K1/K2 values on more intercepted plaintext/ciphertext. If there is matching, Eve has found the keys; if there is no match, Eve needs to repeat step *b* to *e* using a different value of X.

34.

```
permute (n, m, inBlock[1 … n], outBlock[1 … m], permutationTable[1 … m])
{
        i ← 1
        while (i ≤ m)
        {
                outBlock[i] ← inBlock[permutationTable[i]]
                i ← i + 1
        }
        return
}
```

35.

```
split ($n$, $m$, inBlock[1 … $n$], leftBlock[1 … $m$], rightBlock[1 … $m$])
{
    $i \leftarrow 1$
    while ($i \le m$)
    {
        leftBlock[$i$] $\leftarrow$ inBlock[$i$]
        rightBlock[$i$] $\leftarrow$ inBlock[$i + m$]
        $i \leftarrow i + 1$
    }
    return
}
```

36.

```
combine ($n$, $m$, leftBlock[1 … $n$], rightBlock[1 … $n$], outBlock[1 … $m$])
{
    $i \leftarrow 1$
    while ($i \le m$)
    {
        outBlock[$i$] $\leftarrow$ leftBlock[$i$]
        outBlock[$i + m$] $\leftarrow$ rightBlock[$i$]
        $i \leftarrow i + 1$
    }
    return
}
```

37.

```
exclusiveOr ($n$, firstBlock[1 … $n$], secondBlock[1 … $n$], outBlock[1 … $n$])
{
    $i \leftarrow 1$
    while ($i \le n$)
    {
        outBlock[$i$] $\leftarrow$ firstBlock[$i$] $\oplus$ secondBlock[$i$]
        $i \leftarrow i + 1$
    }
    return
}
```

38.

```
cipher (plainBlock[1 … 64], RoundKeys[1 … 16][1 … 48], cipherBlock[1 … m64])
{
        permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
        split (64, 32, inBlock, rightBlock, leftBlock)
        for (round = 1 to 16)
        {
                mixer (leftBlock, rightBlock, RoundKey[round])
                swapper (leftBlock, rightBlock)
        }
        swapper (leftBlock, rightBlock)
        combine (32, 64, leftBlock, rightBlock, outBlock)
        permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
```

39. We have added one extra parameter ED (encrypt/decrypt). If ED = E, we do encryption; if ED = D, we do decryption.

```
cipher (ED, plainBlock[1…64], RoundKeys[1…16][1…48], cipherBlock[1 … m64])
{
        permute (64, 64, plainBlock, inBlock, InitialPermutationTable)
        split (64, 32, inBlock, rightBlock, leftBlock)
        for (round = 1 to 16)
        {
                if (ED  = E)
                        mixer (leftBlock, rightBlock, RoundKey[round]
                if (ED  = D)
                        mixer (leftBlock, rightBlock, RoundKey[16 − round]
                if (round!= 16)
                        swapper (leftBlock, rightBlock)
        }
        combine (32, 64, leftBlock, rightBlock, outBlock)
        permute (64, 64, outBlock, cipherBlock, FinalPermutationTable)
}
```

CHAPTER 7

AES

(Solution to Practice Set)

Review Questions

1. The criteria defined by NIST for selecting AES fall into three areas: *security*, *cost*, and *implementation*.

2. The following table lists the parameters:

Version	*Block size*	*Key size*	*Round-key size*	*Number of rounds*
AES-128	128	128	128	10
AES-192	128	192	128	12
AES-256	128	256	128	14

3. The number of round keys and the number of transformation depend on the number of rounds. The following table list the information. Note that there is one transformation for pre-round. Also note that the last round uses only three transformations.

Version	*Number of rounds*	*Number of round keys*	*Number of Transformations*
AES-128	10	11	$1 + 9 \times 4 + 3 = 40$
AES-192	12	13	$1 + 11 \times 4 + 3 = 48$
AES-256	14	15	$1 + 13 \times 4 + 3 = 56$

4. DES is bit-oriented; AES is byte-oriented. In DES, an S-box takes 6 bits and transforms it to 4 bits; in AES, SubBytes transformation takes a byte and change it to another byte. In DES, P-boxes transpose bits; in AES the ShiftRows transformation transposes bytes. To provide mixing of bits inside a byte, AES uses the MixColumns transformation.

5. Before and after each stage, the data block is referred to as a state. States, like blocks, are made of 16 bytes, but they are normally treated as matrices of 4 × 4 bytes. The following table shows the number of states used in each version. We have used two extra states: one before the pre-round and one after the last round. If you do not consider this, the number of states is reduced by two.

Version	*Number of rounds*	*Number of States*
AES-128	10	**(1)** + 1 + 9 × 4 + 3 + **(1)** = 42
AES-192	12	**(1)** + 1 + 11 × 4 + 3 + **(1)** = 50
AES-256	14	**(1)** + 1 + 13 × 4 + 3 + **(1)** = 58

6. The SubBytes, MixColumns, and AddRoundKey transformation change the contents of bytes; the ShiftRows transformation does not.

7. Substitution in DES is done by S-boxes. Each box substitutes a 6-bit value with a 4-bit value. We need eight S-boxes to create a 32-bit half block. Substitution in AES is done through SubBytes transformation that transforms a whole state to another state. However, we can say that SubBytes actually substitutes 16 bytes with new 16 bytes.

8. Permutation in DES is applied in two steps: before substitution and after subsitution. The first is an expansion permutation to create a 48-bit half block out of a 32-bit half block. This is needed because the round key is 48 bits long. Permutation in AES is straight permutation of bytes. Since the size of the block and the size of the round key are the same, there is no need for an expansion permutation.

9. In DES, the size of the block is 64 bits, but the size of the round key is 48 bits. In AES the size of the block and the round key are both 128 bits (for all versions).

10. In DES, permutation is bit-oriented; in AES, permutation is byte oriented. To permute the bits inside a byte, AES uses the MixColumns transformation.

Exercises

11. The disadvantage of using keyed S-boxes is that it makes the design of the cipher more difficult. In particular, it is more difficult to create S-boxes that are inverse of each other in the encryption and decryption cipher. The advantage of using keyed S-boxes is that a keyed S-box is normally non-linear, which protect the cipher against linear cryptanalysis.

12. A larger block size creates more mixing of bits in each block. In a cipher with a block size of 64, each bit, in average, is dependent on only 32 other bits; in a block size of 128 bits, each bit depends, on average, on 64 bits. This is definitely an advantage. One disadvantage is that the large block size creates an overhead of adding more padding to the last block. In particular, if the size of the message is very small and less than the size of a block, most of the plaintext is made of padding.

13. Having different number of rounds has the advantage that new versions of cipher with more number of rounds can be used, without changing the structure of cipher, if the cipher is attacked by differential and linear cryptanalysis (or other attacks that depend on the number of rounds). For example, we can use AES with 10 rounds as long as it is not secured. If it is attacked, we can move to the version with 12 or 14 rounds without changing the structure of the cipher.

14. Having different key size has the advantage that new versions of cipher with larger key size can be used, without changing the structure of cipher, if the cipher is attacked by brute-force cryptanalysis. For example, we can use AES with a 128-bit cipher key as long as it is not attacked. If it is attacked, we can move to the version with 192-bit or 256-bit cipher key.

15. A cipher in which the size of the round is the same as the size of the round key is easier to design because we do not have to use expansion (or compression) permutation to match the size of the block to the size of the round key. This enable us to make the non-Feistel ciphers.

16.

a. The state has 16 bytes which are permuted by ShiftRows transformation using the following permutation table:

01	02	03	04	06	07	08	05	11	12	09	10	16	13	14	15

b. The state has 16 bytes which are permuted by InvShiftRows transformation using the following permutation table:

01	02	03	04	08	05	06	07	11	12	09	10	14	15	16	13

c. We can prove that two transformation are inverse of each other by applying the the process we learned in Chapter 3. We add indexes, we swap indexes and contents, and then sort the table according to the indexes. We start with the ShiftRows table to get InvShiftRows table. After adding the index, we have

01	02	03	04	06	07	08	05	11	12	09	10	16	13	14	15
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

After swapping the contents and the indexes, we have

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
01	02	03	04	06	07	08	05	11	12	09	10	16	13	14	15

After sorting the table according to the index, we have the following table which is the same as the InvShiftRows table found in part *b.*

01	02	03	04	08	05	06	07	11	12	09	10	14	15	16	13
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

17. We use two plaintexts that differ only in the first bit:

P_1: $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

P_2: $(8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

a. Applying the SubBytes transformation to P_1 and P_2, we get:

T_1: $(6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363)_{16}$

T_2: $(CD63\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363\ 6363)_{16}$

T_2 and T_1 differ only in 5 bits.

b. Applying the ShiftRows transformation to P_1 and P_2, we get:

T_1: $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2: $(8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2 and T_1 differ only in 1 bit.
Applying the MixColumn transformation to P_1 and P_2, we get:

T_1: $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2: $(1B80\ 809B\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2 and T_1 differ in 9 bits.

c. Applying the AddRoundKey of 128 0-bit transformation to P_1 and P_2, we get:

T_1: $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2: $(8000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_{16}$

T_2 and T_1 differ only in 1 bit. Note that we have used a round key with 128 0's, but the result is the same if we use any key.

18. We have

SubBytes (0x57 ⊕ 0xA2) = SubBytes (0xF5) = **0xE6**

SubBytes (0x57) ⊕ SubBytes (0xA2) = 0x5B ⊕ 0x3A = **0x61**

Definitely, the two are different; the SubByte transformation is non-linear

19.

a. The SubBytes transformation is repeated in each round. So we have N_r of this transformation.

b. The ShiftRows transformation is repeated in each round. So we have N_r of this transformation.

c. The MixColumns transformation is repeated in each round except the last round. So we have $(N_r - 1)$ of this transformation.

d. The AddRoundKey transformation is repeated in each round. In addition, we have one of this transformation in the pre-round section. So we have $(N_r + 1)$ of this transformation.

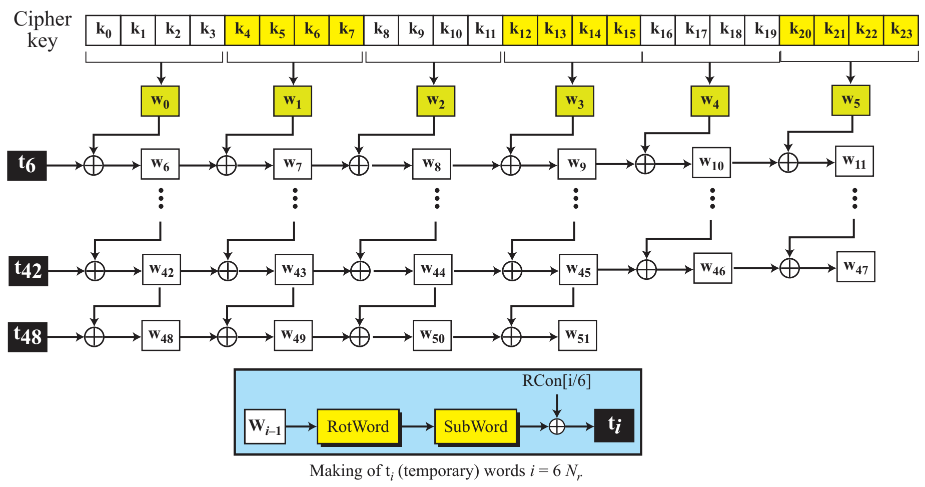
e. The total number of transformation is

$$\text{Total number of transformation} = N_r + N_r + (N_r - 1) + (N_r + 1) = 4 \times N_r$$

20.

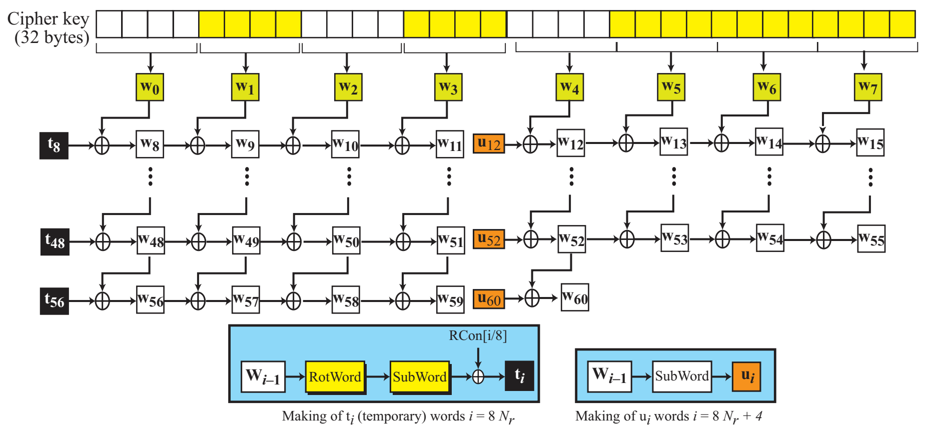
a. Figure S7.20a shows the key expansion in AES-192.

Figure S7.20a *Solution to Exercise 20.a*


Cipher key
k0 k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 k13 k14 k15 k16 k17 k18 k19 k20 k21 k22 k23
w0 w1 w2 w3 w4 w5
t6 w6 w7 w8 w9 w10 w11
t42 w42 w43 w44 w45 w46 w47
t48 w48 w49 w50 w51
RCon[i/6]
Wi–1 RotWord SubWord ti
Making of ti (temporary) words i = 6 Nr

b. Figure S7.20b shows the key expansion in AES-256

Figure S7.20b *Solution to Exercise 20.b*


Cipher key (32 bytes)
w0 w1 w2 w3 w4 w5 w6 w7
t8 w8 w9 w10 w11 u12 w12 w13 w14 w15
t48 w48 w49 w50 w51 u52 w52 w53 w54 w55
t56 w56 w57 w58 w59 u60 w60
RCon[i/8]
Wi–1 RotWord SubWord ti
Wi–1 SubWord ui
Making of ti (temporary) words i = 8 Nr
Making of ui words i = 8 Nr + 4

21.

a. We can use (x^{11-1} mod prime) and (x^{12-1} mod prime), in which the prime is the the irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$), to find the first terms of RCon[11] and RCon[12]. The following shows the constants for AES-192.

Round	*(RCon)*	*Round*	*(RCon)*	*Round*	*(RCon)*
1	$(\underline{\textbf{01}}\ 00\ 00\ 00)_{16}$	5	$(\underline{\textbf{10}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\textbf{1B}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\textbf{02}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\textbf{20}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\textbf{36}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\textbf{04}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\textbf{40}}\ 00\ 00\ 00)_{16}$	11	$(\underline{\textbf{6C}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\textbf{08}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\textbf{80}}\ 00\ 00\ 00)_{16}$	12	$(\underline{\textbf{D8}}\ 00\ 00\ 00)_{16}$

b. We can use (x^{13-1} mod prime) and (x^{14-1} mod prime), in which the prime is the the irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$), to find the first terms of RCon[13] and RCon[14]. The following shows the constants for AES-256.

Round	*(RCon)*	*Round*	*(RCon)*	*Round*	*(RCon)*
1	$(\underline{\textbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\textbf{20}}\ 00\ 00\ 00)_{16}$	11	$(\underline{\textbf{6C}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\textbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\textbf{40}}\ 00\ 00\ 00)_{16}$	12	$(\underline{\textbf{D8}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\textbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\textbf{80}}\ 00\ 00\ 00)_{16}$	13	$(\underline{\textbf{AB}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\textbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\textbf{1B}}\ 00\ 00\ 00)_{16}$	14	$(\underline{\textbf{4D}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\textbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\textbf{36}}\ 00\ 00\ 00)_{16}$		

22.

a. The pre-round operation needs a four-word round key. The cipher key in AES-192 is six words; only the first four words of it is used in the pre-round operation.

b. The pre-round operation needs a four-word round key. The cipher key in AES-256 is eight words; only the first four words of it is used in the pre-round operation.

23. The result is an identity matrix as shown below. Note that the addition and multiplication of elements are in GF(2).

$$\underbrace{\begin{bmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&0 \end{bmatrix}}_{\text{X}} \times \underbrace{\begin{bmatrix} 0&0&1&0&0&1&0&1\\ 1&0&0&1&0&0&1&0\\ 0&1&0&0&1&0&0&1\\ 1&0&1&0&0&1&0&0\\ 0&1&0&1&0&0&1&0\\ 0&0&1&0&1&0&0&1\\ 1&0&0&1&0&1&0&0\\ 0&1&0&0&1&0&1&0 \end{bmatrix}}_{\text{X}^{-1}} = \underbrace{\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}}_{\text{X} \times \text{X}^{-1}}$$

24. The result is an identity matrix as shown below. Note that the addition and multiplication of coefficients are in GF(2).

$$\underbrace{\begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix}}_{C} \times \underbrace{\begin{bmatrix} x^3+x^2+x & x^3+x+1 & x^3+x^2+1 & x^3+1 \\ x^3+1 & x^3+x^2+x & x^3+x+1 & x^3+x^2+1 \\ x^3+x^2+1 & x^3+1 & x^3+x^2+x & x^3+x+1 \\ x^3+x+1 & x^3+x^2+1 & x^3+1 & x^3+x^2+x \end{bmatrix}}_{C^{-1}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{C \times C^{-1}}$$

The identity matrix can be easy found. For example, if we multiply the first row of C by the first column of C^{-1}, we get

$$(x)(x^3+x^2+x) + (x+1)(x^3+1) + (1)(x^3+x^2+1) + (1)(x^3+x+1) = 1$$

But if we multiply the second row of C by the first column of C^{-1}, we get

$$(1)(x^3+x^2+x) + (x)(x^3+1) + (x+1)(x^3+x^2+1) + (1)(x^3+x+1) = 0$$

25. Most of the code matches, line by line, with the steps in the process. The only section that needs some explanation is the loop. The iterations of the loop gives us

$$\begin{array}{lcl}
\mathbf{c}_0 = \mathbf{b}_0 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 & \rightarrow & \mathbf{c}_0 = \mathbf{b}_0 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_1 = \mathbf{b}_1 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 & \rightarrow & \mathbf{c}_1 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_2 = \mathbf{b}_2 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 & \rightarrow & \mathbf{c}_2 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7 \\
\mathbf{c}_3 = \mathbf{b}_3 \oplus \mathbf{b}_7 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 & \rightarrow & \mathbf{c}_3 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_7 \\
\mathbf{c}_4 = \mathbf{b}_4 \oplus \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 & \rightarrow & \mathbf{c}_4 = \mathbf{b}_0 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \\
\mathbf{c}_5 = \mathbf{b}_5 \oplus \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 & \rightarrow & \mathbf{c}_5 = \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \\
\mathbf{c}_6 = \mathbf{b}_6 \oplus \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 & \rightarrow & \mathbf{c}_6 = \mathbf{b}_2 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \\
\mathbf{c}_7 = \mathbf{b}_7 \oplus \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 & \rightarrow & \mathbf{c}_7 = \mathbf{b}_3 \oplus \mathbf{b}_4 \oplus \mathbf{b}_5 \oplus \mathbf{b}_6 \oplus \mathbf{b}_7
\end{array}$$

The rearranged code is the result of matrix multiplication $\mathbf{c} = \mathbf{X} \times \mathbf{b}$ if we ignore the zero terms. After each line is created $\mathbf{d} = \mathbf{c} + \mathbf{y}$ is made.

26.

a. The **invByte** routine calls two other routine: **multiply** and **quotation**. The first routine multiplies two bytes; the second routine finds the quotient of dividing the first byte by the second. The **quotation** routine calls the **degree** routine which finds the degree of a byte (as a polynomial)

```
invByte (byte)
{
    if (byte = 0)
        return byte
    $r_1 \leftarrow (11B)_{16}$                              // modulus
    $r_2 \leftarrow$ byte
    $t_1 \leftarrow (00)_{16}$
    $t_2 \leftarrow (01)_{16}$
    while ($r_2 > (00)_{16}$)
    {
        $q \leftarrow$ quotation($r_1, r_2$)
        $r \leftarrow r_1 \oplus$ multiply ($q, r_2$)
        $t \leftarrow t_1 \oplus$ multiply ($q, t_2$)
        $r_1 \leftarrow r_2$    $r_2 \leftarrow r$    $t_1 \leftarrow t_2$    $t_2 \leftarrow t$
    }
    return $t_1$
}
```

```
multiply (byte$_1$, byte$_2$)
{
    $i \leftarrow (00)_{16}$
    $res \leftarrow (00)_{16}$
    while ($byte_2 \neq (00)_{16}$)
    {
        if [($byte_2$ AND $(01)_{16}) = (01)_{16}$]
            $res \leftarrow res \oplus$ shiftLeft($i$, byte$_1$)
        byte$_2 \leftarrow$ shiftRight(1, byte$_2$)
    }
    return $res$
}
```

```
Quotation (byte$_1$, byte$_2$)
{
    degreeDiff $\leftarrow$ degree (byte$_1$) $-$ degree (byte$_2$)
    while (degreeDiff $\geq 0$)
    {
        temp $\leftarrow$ shiftLeft(degreeDiff, 1)
```

```
                byte$_1$ ← byte$_1$ ⊕ **shiftLeft**(degreeDiff, byte$_2$)
                res ← res OR temp
        }
        return *res*
}
```

```
**degree** (byte)
{
        deg ← −1
        while (byte > 0)
        {
                deg ← deg + 1
                byte ← **shiftLeft**(1, byte)
        }
        return deg
}
```

b.

```
**ByteToMatrix** (byte, **matrix**)
{
        for ($r$ = 7 downto 0)
        {
                **matrix**[r] ← byte mod 2
                byte ← byte / 2                         // integer division
        }
}
```

c.

```
**MatrixToByte** (**matrix**, byte)
{
        byte ← 0
        for ($r$ = 7 downto 0)
        {
                byte ← byte × 2 + **matrix**[r]
        }
}
```

27.

```
InvSubBytes (S[0 … 3][0 … 3])
{
    for ($r$ = 0 to 3)
        for (c = 0 to 3)
            S[$r$][$c$] ← invsubbyte (S[$r$][$c$])
    return (S)
}
```

```
invsubbyte (byte)
{
    d ← ByteToMatrix (byte)
    c ← d ⊕ ByteToMatrix (0x63)
    b ← $X^{-1}$ × c
    a ← MatrixToByte (b)
    return ($a^{-1}$)
}
```

28. The **ShiftRows** algorithm calls the **shiftrow** routine three times (the first row is not shifted) to shift each row. The following shows how shifting is done:

First call $n = 1$	c = 0	$row_3 \leftarrow t_0$
	c = 1	$row_0 \leftarrow t_1$
	c = 2	$row_1 \leftarrow t_2$
	c = 3	$row_2 \leftarrow t_3$
Second call $n = 2$	c = 0	$row_2 \leftarrow t_0$
	c = 1	$row_3 \leftarrow t_1$
	c = 2	$row_0 \leftarrow t_2$
	c = 3	$row_1 \leftarrow t_3$
Third call $n = 3$	c = 0	$row_3 \leftarrow t_0$
	c = 1	$row_2 \leftarrow t_1$
	c = 2	$row_0 \leftarrow t_2$
	c = 3	$row_1 \leftarrow t_3$

29.

```
CopyRow (row[0 … 3], t[0 … 3])
{
    for ($c$ = 0 to 3)
        t[$c$] ← row [$c$]
}
```

30.

```
InvShiftRows(S[0 … 3][0 … 3]
{
        for ($r$ = 0 to 3)
                invshiftrow (S[$r$], $r$)
}
```

```
invshiftrow (row[0 … 3], r)
{
        CopyRow (row, t)
        for ($c$ = 0 to 3)
                        row[(c + n) mod 4] ← t[c]
}
```

31. The **MixColumns** algorithm calls the **mixcolum** routine four times, once for each column of the old state to create the correspond column of the new state. The **mixcolumn** routine actually performs the following matrix multiplication **col** = **C** × **t**, in which **col** is the new column matrix, **t** is the old column matrix, and the **C** is the square constant matrix. We can write the code in the **mixcolumn** routine as

$$\begin{array}{lllllllll}
\mathbf{col}_0 & \leftarrow & C_{00} \bullet t_0 & \oplus & C_{01} \bullet t_1 & \oplus & C_{02} \bullet t_2 & \oplus & C_{03} \bullet t_3 \\
\mathbf{col}_1 & \leftarrow & C_{10} \bullet t_0 & \oplus & C_{11} \bullet t_1 & \oplus & C_{12} \bullet t_2 & \oplus & C_{13} \bullet t_3 \\
\mathbf{col}_2 & \leftarrow & C_{20} \bullet t_0 & \oplus & C_{21} \bullet t_1 & \oplus & C_{22} \bullet t_2 & \oplus & C_{23} \bullet t_3 \\
\mathbf{col}_3 & \leftarrow & C_{30} \bullet t_0 & \oplus & C_{31} \bullet t_1 & \oplus & C_{32} \bullet t_2 & \oplus & C_{33} \bullet t_3
\end{array}$$

32.

```
CopyColumn (column[0 … 3], t[0 … 3])
{
        for ($r$ = 0 to 3)
                t[$r$] ← column [$r$]
}
```

33.

```
MixColumns (S[0 … 3][0 … 3]
{
        for ($c$ = 0 to 3)
                mixcolumn (S[c])
}
```

```
mixcolumn (col [0 … 3])
{
    CopyColumn (col, t)
    col[0] ← MultField (0x02, t[0]) ⊕ MultField (0x03, t[1]) ⊕ t[2] ⊕ t[3]
    col[1] ← t[0] ⊕ MultField (0x02, t[1]) ⊕ MultField (0x03, t[2]) ⊕ t[3]
    col[2] ← t[0] ⊕ t[1] ⊕ MultField (0x02, t[2]) ⊕ MultField (0x03, t[3])
    col[3] ← MultField (0x03, t[0]) ⊕ t[1] ⊕ t[2] ⊕ MultField (0x02, t[3])
}
```

34.

```
InvMixColumns (S[0 … 3][0 … 3]
{
    for (c = 0 to 3)
        invmixcolumn (S[c])
}
```

```
invmixcolumn (col [0 … 3])
{
    CopyColumn (col, t)
    col[0] ← 0x0E • t[0]) ⊕ (0x0B • t[1]) ⊕ (0x0D • t[2]) ⊕ (0x09 • t[3])
    col[1] ← 0x09 • t[0]) ⊕ (0x0E • t[1]) ⊕ (0x0B • t[2]) ⊕ (0x0D • t[3])
    col[2] ← 0x0D • t[0]) ⊕ (0x09 • t[1]) ⊕ (0x0E • t[2]) ⊕ (0x0B • t[3])
    col[3] ← 0x0B • t[0]) ⊕ (0x0D • t[1]) ⊕ (0x09 • t[2]) ⊕ (0x0E • t[3])
}
```

35. The algorithm uses a loop that iterates four times, one for each column of the current state. In each iteration, a column of the old state is exclusive-ored with a key word to create a new column.

$$\mathbf{S}[c] \leftarrow \mathbf{S}[c] \oplus \mathbf{W}[4 \times \text{round} + c]$$

For example, in round 5, we have

$\mathbf{S}[0]$	$\leftarrow$	$\mathbf{S}[0] \oplus \mathbf{W}[20]$
$\mathbf{S}[1]$	$\leftarrow$	$\mathbf{S}[1] \oplus \mathbf{W}[21]$
$\mathbf{S}[2]$	$\leftarrow$	$\mathbf{S}[2] \oplus \mathbf{W}[22]$
$\mathbf{S}[3]$	$\leftarrow$	$\mathbf{S}[3] \oplus \mathbf{W}[23]$

36.

a. The subbyte routine called by the SubWord is the one defined in the text.

```
SubWord (W[0 … 3])
{
        for ($i$ = 0 to 3)
                W[$i$] ← subbyte (W[$i$])
        return (W)
}
```

b.

```
RotWord (W[0 … 3])
{
        CopyRow (W, t)
        for ($i$ = 0 to 3)
                        W[($i$ + 3) mod 4] ← t[$i$]
        return (W)
}
```

37.

a.

```
KeyExpansion (Key[0 … 23], W[0 … 51]
{
        for ($i$ = 0 to 5)
                W[$i$] ← Key[4$i$] | Key[4$i$ +1] | Key[4$i$ +2] | Key[4$i$ +3]
        for ($i$ = 6 to 51)
                if ($i$ mod 6 = 0)
                {
                        t ← subword(rotWord (W[$i$ – 1]) ⊕ Rcon[$i$ /6]
                        W[$i$] ← t ⊕ W[$i$ – 6]
                }
                else
                        W[$i$] ← W[$i$ – 1] ⊕ W[$i$ – 6]
}
```

b.

```
KeyExpansion (Key[0 … 31], W[0 … 59])
{
        for ($i$ = 0 to 7)
                W[$i$] ← Key[4$i$] | Key[4$i$ +1] | Key[4$i$ +2] | Key[4$i$ +3]
        for ($i$ = 8 to 59)
```

```
                if ($i$ mod 8 = 0)
                {
                        t ← subword(rotWord (W[$i$ – 1]) ⊕ Rcon[$i$ /8]
                        W[$i$] ← t ⊕ W[$i$ – 8]
                }
                if ($i$ mod 8 = 4)
                {
                        t ← subword(W[$i$ – 1])
                        W[$i$] ← t ⊕ W[$i$ – 8]
                }
                else
                        W[$i$] ← W[$i$ – 1] ⊕ W[$i$ – 8]
}
```

38. The following algorithm modifies the keys created for the original design to create the round keys for the alternate design.

```
KeyExpansionAlternative (Key[0 … 15])
{
      KeyExpansion(Key[0 … 15], W[0 … 43])
      for ($rnd$ = 1 to 9)
      {
            for ($r$ = 0 to 3)
                  t[$r$] ← W[$rnd$ × 4 + $r$]
            W[$rnd$ × 4 + 0] ← 0x0E • t[0] ⊕ 0x0B • t[1] ⊕ 0x0D • t[2] ⊕ 0x09 • t[3]
            W[$rnd$ × 4 + 1] ← 0x09 • t[0] ⊕ 0x0E • t[1] ⊕ 0x0B • t[2] ⊕ 0x0D • t[3]
            W[$rnd$ × 4 + 2] ← 0x0D • t[0] ⊕ 0x09 • t[1] ⊕ 0x0E • t[2] ⊕ 0x0B • t[3]
            W[$rnd$ × 4 + 3] ← 0x0B • t[0] ⊕ 0x0D • t[1] ⊕ 0x09 • t[2] ⊕ 0x0E • t[3]
      }
      return (W[0 … 43])
}
```

39.

```
InvCipher (InBlock[0 … 16], outBlock[0 … 16], W[0 … 43]
{
        BlockToState (Inblock, S)
        S ← AddRoundKey (S, W[40 … 43])
        for ($r$ = 1 to 10)                          // $r$ defines the round
        {
                S ← InvShiftRows (S)
                S ← InvSubBytes (S)
                S ← AddRoundKey (S, W[$(10 - r) \times 4$ … $(10 - r) \times 4 + 3$)
                if ($r \neq 10$)
                        S ← InvMixColumns (S)
        }
        StateToBlock (S, outBlock)
}
```

40.

```
InvCipherAlternate (InBlock[0 … 16], outBlock[0 … 16], W[0 … 43]
{
        BlockToState (Inblock, S)
        S ← AddRoundKey (S, W[40 … 43])
        for ($r = 1$ to 9)                                  // r defines the round
        {
                S ← InvSubBytes (S)
                S ← InvShiftRows (S)
                S ← InvMixColumns (S)
                S ← InvAddRoundKey (S, W[$(10 - r) \times 4$ … $(10 - r) \times 4 + 3$)
        }
        S ← InvSubBytes (S)
        S ← InvShiftRows (S)
        S ← AddRoundKey (S, W[0 … 3])
        StateToBlock (S, outBlock)
}
```

CHAPTER 8

Encipherment Using Modern Block Ciphers

(Solution to Practice Set)

Review Questions

1. Modern block ciphers encrypt and decrypt small blocks. DES uses a block size of 8 bytes (characters) and AES uses a block size of 16 bytes (characters). In real life, we need to encrypt or decrypt larger units of data. For example, a one-page document is normally more than 1000 characters. Mode of operations are designed to allow us to repeatedly use a modern block cipher for encryption and decryption.

2. We discussed electronic codebook (ECB), cipher block chaining (CBC), cipher feedback (CFB), output feedback (OFB), and counter (CTR) modes.

3. In the electronic codebook (ECB) mode, the plaintext is divided into N blocks. Each block is n bits. The same key is used to encrypt and decrypt each block.

- **Advantages**. This mode has two obvious advantages. First, it is simple. Second, transmission error is not propagated from one block to the other.
- **Disadvantages**. This mode has some security problems. First, patterns at the block level are preserved. Second, block independency creates opportunities for Eve to substitutes some cipher blocks with some cipher blocks of her own.

4. In the cipher block chaining (CBC) mode, each plaintext block is exclusive-ored with the previous ciphertext block before being encrypted. A phony block called the initial vector (IV) is used to serve as C_0. The same key is used to encrypt and decrypt each block.

- **Advantages**. This mode has one obvious advantage. Each ciphertext block depends on previous ciphertext blocks. Patterns at the block level are not preserved. Eve cannot reorder the ciphertext blocks.
- **Disadvantages**. This mode has some security problems. First, if two messages are equal, their encipherment is the same if they use the same IV. Second, Eve can add some ciphertext blocks at the end of the message. The mode also has some error-propagation problem: a single bit error in one ciphertext block may creates errors in the corresponding plaintext block and the next plaintext block.

5. In the cipher feedback (CFB) mode, the size of the plaintext or ciphertext block is r, where $r \leq n$. The idea is to use DES or AES, not for encrypting the plaintext or decrypting the ciphertext, but to encrypt or decrypt the contents of a shift register, S, of size n. Data encryption is done by exclusive-oring an r-bit plaintext block with r bits of the shift register. Data decryption is done by exclusive-oring an r-bit ciphertext block with r bits of the shift register. For each block, the shift register S_i is made by shifting the shift register S_{i-1}(previous shift register) r bits to the left and filling the rightmost r bits with C_{i-1}.

   - ❑ **Advantages**. One advantage of CFB is that no padding is required because the size of the blocks, r, is normally chosen to fit the data unit to be encrypted. Another advantage is that the system does not have to wait until it has received a large block of data before starting the encryption.

   - ❑ **Disadvantages.** One disadvantage of CFB is that it is less efficient than CBC or ECB, because it needs to apply the encryption function of underlying block cipher for each small block of size r. Another disadvantage is that Eve can add some ciphertext block to the end of the stream.

6. The output feedback (OFB) mode is very similar to CFB mode, with one difference: each bit in the ciphertext is independent of the previous bit or bits. This avoids error propagation. If an error occurs in transmission, it does not affect the bits that follow. Like CFB, both the sender and the receiver use the encryption algorithm.

   - ❑ **Advantages**. One advantage of OFB is that no padding is required because the size of the blocks, r, is normally chosen to fit the data unit to be encrypted. Another advantage is that the system does not have to wait until it has received a large block of data before starting the encryption.

   - ❑ **Disadvantages.** One disadvantage of OFB is that it is less efficient than CBC or ECB, because it needs to apply the encryption function of underlying block cipher for each small block of size r. Another disadvantage is that Eve can add some ciphertext block to the end of the stream.

7. In the counter (CTR) mode, there is no feedback. The pseudorandomness in the key stream is achieved using a counter. An n-bit counter is initialized to a predetermined value (IV) and incremented based on a predefined rule (mod 2^n). To provide a better randomness, the increment value can depend on the block number to be incremented. The plaintext and ciphertext block have the same block size as the underlying cipher (e.g., DEA or AES). Plaintext blocks of size n are encrypted to create ciphertext blocks of size n.

   - ❑ **Advantages**. One advantage of CTR is that it can be used to create random access file as long as the value of the counter is related to the record number in the file.

   - ❑ **Disadvantages.** One disadvantage of CTR is that the size of block is the same as the size of the underlying cipher (DES or AES). Encryption or decryption needs to be done n-bit at a time; the process needs to wait until n bit of data is accumulated.

8.

First Group: ECB and CBC	**Second Group:** CFB, OFB, and CTR

9.

First Group: ECB and CBC	**Second Group:** CFB, OFB, and CTR

10.

First Group: ECB and CBC	**Second Group:** CFB, OFB, and CTR

11. RC4 uses a state of bytes for key generation. Each key in the key stream is a permutation of a key state. A5/1 uses the result of three LFSR's to create a key bit. We can say that key generation in RC4 is byte-oriented, but the key generation in A5/1 is bit-oriented. The other main difference is encryption and decryption. RC4 encrypts and decrypts a character at a time; A5/1 encrypts and decrypts a frame at a time.

12. The size of data in RC4 is normally 8 bits; data is normally encrypted or decrypted a byte at a time. The size of data in A5/1 is 228 bits; data is encrypted or decrypted a frame at a time.

13. ECB can be used for parallel processing because each block is encrypted or decrypted independently.

14. ECB can be used for encipherment of blocks in a random-access file because the encryption and decryption of each block is independent from the rest of the blocks.

Exercises

15. In CFB mode, the key generator for block i uses C_{i-1}, the ciphertext created in block $(i - 1)$. According to our definition in Chapter 5, this is a *nonsynchronous stream cipher*. In OFB mode, the key generator for block i uses part of the key from the previous block, but it is independent from the plaintext and ciphertext in the previous block. According to our definition in Chapter 5, this is a *synchronous stream cipher*.

16. In CFB, ciphertext is transmitted in blocks of r bits. A single bit error in a ciphertext block affects the corresponding bit plaintext block. However, the corruption does not stop here. The corrupted ciphertext block in the receiver site enters the shift register and corrupts the subsequent plaintext blocks until it falls off the shift register. In general, a single bit error in ciphertext, may corrupt $(n/r + 1)$ blocks of the plaintext. To see the relations, assume the underlying cipher is DES $(n = 64)$ and encryption/decryption is done a byte at a time $(r = 8)$. Imagine the second bit in ciphertext block 5 is corrupted during transmission. When this block is decrypted, only the second bit in plaintext block 5 is in error. However, this ciphertext block enters the shift register and remains in there until the next 8 blocks arrives. This results in the possible corruption of all bits in the next 8 plaintext blocks. In other words, 9 blocks may have been corrupted.

17. In ECB, each block is decrypted independently. However, since the decryption is done a block at a time (DES or AES), the corrupted bit 17 in ciphertext block 8 may affect all *n* bits in plaintext block 8.

18. In CBC, if bits 17 and 18 in ciphertext block 9 are corrupted, all *n* bits in plaintext block 9 may be corrupted (decryption is one block at a time). However, bits 17 and 18 in ciphertext block 9 are also exclusive-ored with bits 17 and 18 of the next block (block 10). Therefore, $n + 2$ bits may be corrupted; *n* bits in block 9 and 2 bits in block 10. None of the bits of other blocks (block 11 …) are affected. The system recovers itself after block 11.

19. As discussed in the solution to Exercise 16, a corrupted ciphertext block in CFB affects the corresponding plaintext block and the following n/r plaintext blocks. If we assume n = 64, the following plaintext blocks may be corrupted: block 11 (bits 3 to 6) and block 12 to 19 (all bits).

20. In CRT mode, there is no feedback. If ciphertext blocks 3 and 4 are corrupted, only plaintext blocks 3 and 4 will be corrupted.

21. In OFB mode, the feedback is only in the key-generation system. If ciphertext block 11 is corrupted, only plaintext block 11 may be corrupted.

22. In CFB mode, it is obvious that if k_i used at the Bob's site is the same as k_i used at the Alice's site, then $(\mathrm{P}_i)'$ created by Bob is the same as P_i sent by Alice (assuming no corruption in transmission):

$$(\mathrm{P}_i)' = (\mathrm{C}_i)' \otimes k_i = \mathrm{P}_i \otimes k_i \otimes k_i = \mathrm{P}_i \otimes 0 = \mathrm{P}_i$$

However, we also need to prove that k_i used by Alice and Bob are also the same. If there is no corruption in transmission and the IV's used by Alice and Bob are the same, then $(\mathrm{S}_i)' = \mathrm{S}_i$. Now, we can prove

$$(k_i)' = \text{ExtractLeft}_r\ [\mathrm{E_K}(\mathrm{S}_i)'] = \text{ExtractLeft}_r\ [\mathrm{E_K}(\mathrm{S}_i)] = k_i$$

23. In OFB mode, it is obvious that if k_i used at the Bob's site is the same as k_i used at the Alice's site, then $(\mathrm{P}_i)'$ created by Bob is the same as P_i sent by Alice (assuming no corruption in transmission):

$$(\mathrm{P}_i)' = (\mathrm{C}_i)' \otimes k_i = \mathrm{P}_i \otimes k_i \otimes k_i = \mathrm{P}_i \otimes 0 = \mathrm{P}_i$$

The k_i used by Bob is definitely the same as k_i used by Alice if both Alice and Bob use the same IV's.

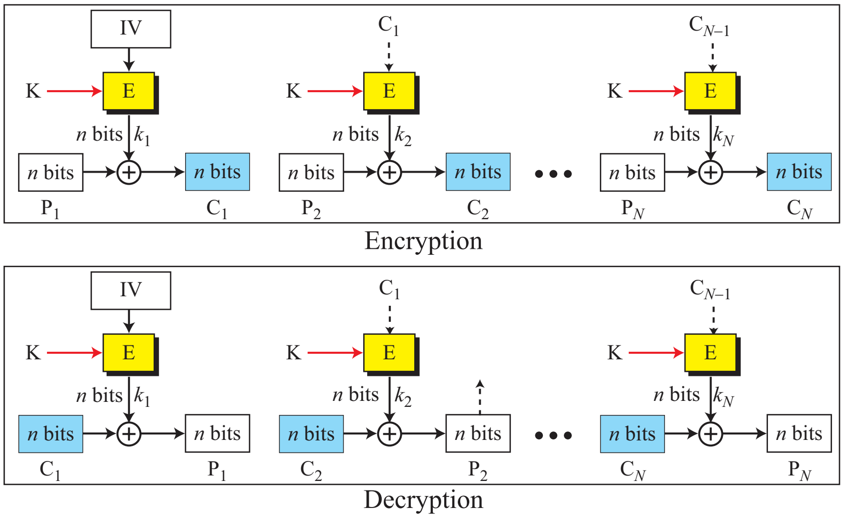
24. In CRT mode, it is obvious that if k_i used at the Bob's site is the same as k_i used at the Alice's site, then $(\mathrm{P}_i)'$ created by Bob is the same as P_i sent by Alice (assuming no corruption in transmission):

$$(\mathrm{P}_i)' = (\mathrm{C}_i)' \otimes k_i = \mathrm{P}_i \otimes k_i \otimes k_i = \mathrm{P}_i \otimes 0 = \mathrm{P}_i$$

The k_i used by Bob is definitely the same as k_i used by Alice if both Alice and Bob use the same IV's for the counter.

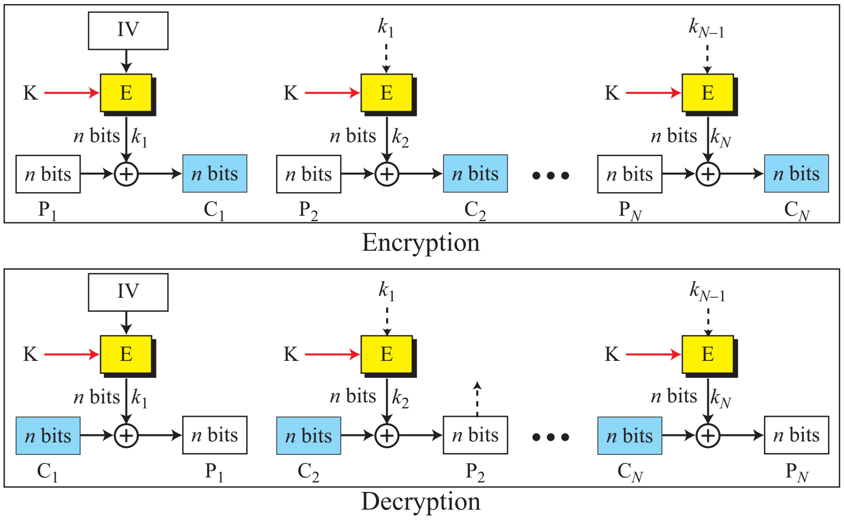
25. Figure S8.25 shows both the encryption and decryption. It is possible to cancel the underlying encryption for the first block and exclusive-or the IV directly with P_1.

Figure S8.25 *Solution to Exercise 25*



26. Figure S8.26 shows both the encryption and decryption. It is possible to cancel the underlying encryption for the first block and exclusive-or the IV directly with P_1.

Figure S8.26 *Solution to Exercise 26*

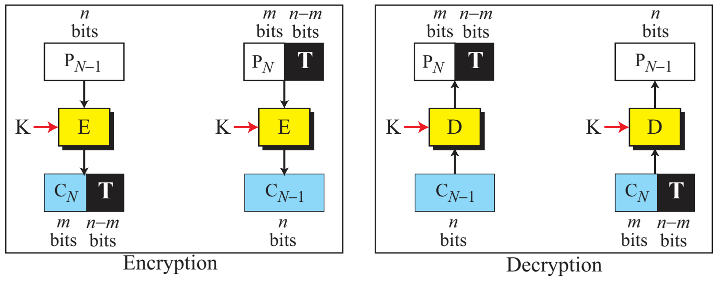


27. The following shows the process:

$X = D_K(C_{N-1})$	$\rightarrow$	$P_N = \mathbf{head}_m(X)$
$Y = C_N \mid \mathbf{tail}_{n-m}(X)$	$\rightarrow$	$P_{N-1} = D_K(Y)$

28. Figure S8.28 shows the diagram for encryption and decryption.

Figure S8.28 *Solution to Exercise 28*

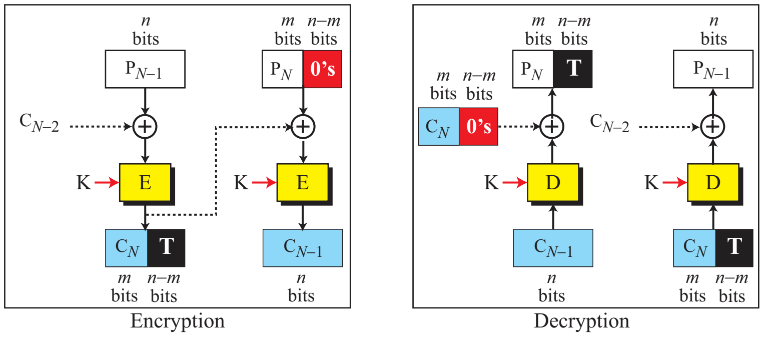


29. The following shows the process:

$U = D_K(C_{N-1})$	$X = U \oplus [C_N \mid \mathbf{Pad}_{n-m}(0)]$	$\rightarrow$	$P_N = \mathbf{head}_m(X)$
$Y = DK\,[C_N \mid \mathbf{tail}_{n-m}(X)]$		$\rightarrow$	$P_{N-1} = C_{N-2} \oplus Y$

30. Figure S8.30 shows the diagram for encryption and decryption.

Figure S8.30 *Solution to Exercise 30*



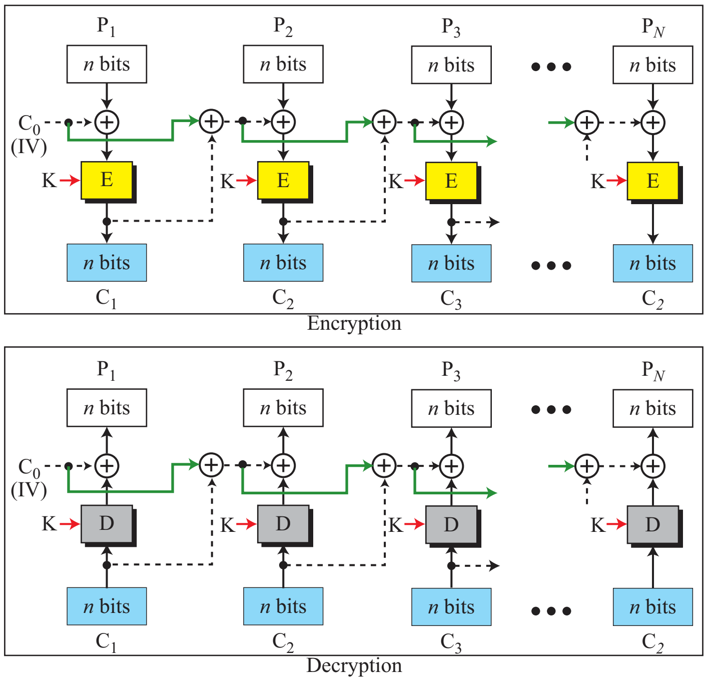
31. CFB, OFB, and CRT are stream ciphers, in which the size of the block is usually fixed (one character). There is no need for padding. This means there is no need for cipher stealing technique.

32. In this case, the error propagation is the same as the case where the CTS technique is not used except for the last two blocks. If C_{N-1} is corrupted, it corrupts both P_{N-1} and P_N. If C_N is corrupted, it corrupts only P_{N-1}.

33. In this case, the error propagation is the same as the case where the CTS technique is not used except for the last two blocks. If C_{N-1} is corrupted, it corrupts both P_{N-1} and P_N. If C_N is corrupted, it corrupts only P_{N-1}.

34. Figure S8.34 shows the BC mode.

Figure S8.34 *Solution to Exercise 34*



35. Figure S8.35 shows the PCBC mode. In PCBC an error in a ciphertext block corrupts all plaintext block that follows. This mode was used in Kerberos (See Chapter 15) to create both encryption and integrity of blocks. If a block was corrupted, all previous blocks were discarded.

36. Figure S8.36 shows the CBCC mode. In CBCC an error in a plaintext block corrupts all ciphertext block that follows.

Figure S8.35 *Solution to Exercise 35*

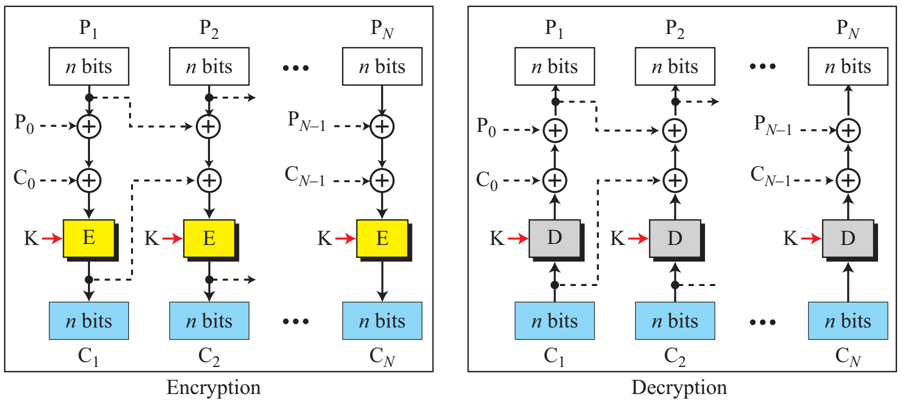
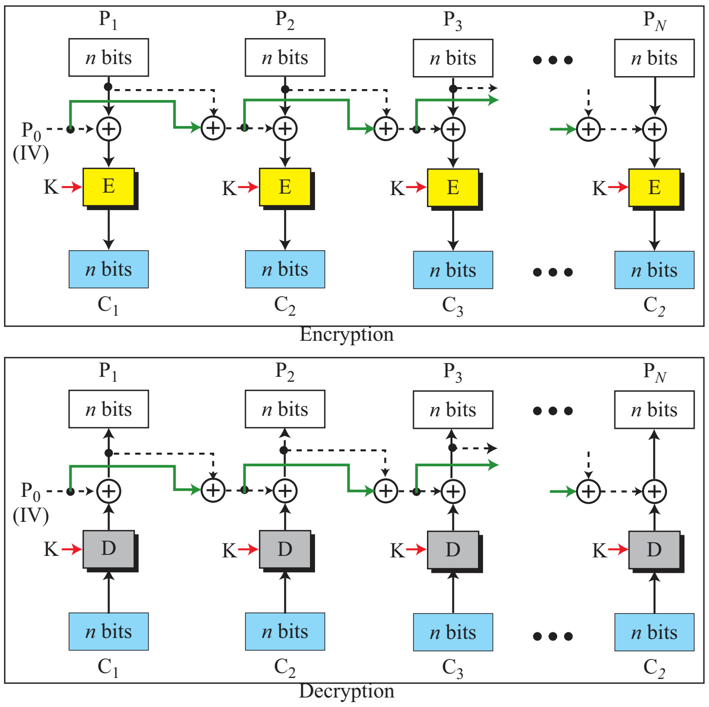


Figure S8.36 *Solution to Exercise 36*



37. We have written a program based on Algorithm 8.6 of the textbook. The result is

41 63 2 212 127 55 201 182 51 242 175 82 133 254 180 107 230 32 241 57

38. This situation is impossible because

- **a.** If the state must remain the same after the second step, it is needed that S[i] has the same value for each $i = 0$ to 255 (for example all 0's, all 1's, all 2's, …) because only in these cases the permutation in the following steps cannot change the values.
- **b.** But the above condition can never happen because, before the second step, we have S[0] = 0, S[1] = 1, …, S[255] = 255 and the swapping can never make them the same.

39. This problem can be solved using the classical third birthday problem that we review in Appendix E. We have a sample set of k values, in which each sample can take one of the N values. What is the minimum size of k such that it is probable (with probability $P \geq 1/2$) such that at least two samples are the same. The answer is $k = 1.18\ N^{1/2}$. In this case $N = 2^{128}$ possible keys, which means $k = 1.18 \times 2^{64}$.

40. According to the literature, all of the three characteristic polynomials are irreducible and the number of taps are even, so the maximum period can be found using the formula $2^m - 1$.

- **a.** For LFSR1, we have $2^{19} - 1$.
- **b.** For LFSR2, we have $2^{22} - 1$.
- **c.** For LFSR3, we have $2^{23} - 1$.

41.

- **a.** Majority (1, 0, 0) = 0. Therefore, $LFSR_2$ and $LFSR_3$ whose clocking bits matches with the value of the Majority function are clocked.
- **b.** Majority (0, 1, 1) = 1. Therefore, $LFSR_2$ and $LFSR_3$ whose clocking bits matches with the value of the Majority function are clocked.
- **c.** Majority (0, 0, 0) = 0. Therefore all three LFSR's are clocked.
- **d.** Majority (1, 1, 1) = 1. Therefore all three LFSR's are clocked.

42. If the three bits in the Majority functions are called a, b, and c respectively, then

$$\text{Majority}\ (a, b, c) = (a \text{ AND } b) \oplus (b \text{ AND } c) \oplus (c \text{ AND } a)$$

43.

```
ECB_Decryption (K, C[1 … N])
{
        for (i = 1 to N)
        {
                P[i] ← D_K (C[i])
        }
        return (P[1 … N])
}
```

44.

```
CBC_Decryption (IV, K, C[1 … N])
{
        C[0] ← IV
        for (i = 1 to N)
        {
                temp ← $D_K$ (C[i])
                temp ← temp ⊕ C[i −1]
        }
        return (P[1 … N])
}
```

45. We use a variable Pre_C to hold the previous cipher block. In this way, we do not have to keep an array of ciphertext blocks.

```
CFB_Decryption (IV, K, r)
{
        i ← 1
        while (more blocks to decrypt)
        {
                input (C)
                if (i = 1)
                        S ← IV
                else
                {
                        Temp ← shiftLeft(r , S)
                        S ← concatenate(Temp, Pre_C)
                }
                T ← $E_K$(S)
                k ← selectLeft(r, T)
                P ← C ⊕ k
                output (P)
                Pre_C ← C
                i ← i + 1
        }
}
```

46. We use a variable Pre_*k* to hold the previous key. In this way, we do not have to keep an array of *k*'s.

```
OFB_Decryption (IV, K, r)
{
        $i \leftarrow 1$
        while (more blocks to decrypt)
        {
                input (C)
                if ($i = 1$)
                        $S \leftarrow$ IV
                else
                {
                        Temp $\leftarrow$ shiftLeft($r$, S)
                        S $\leftarrow$ concatenate(Temp, Pre_$k$)
                }
                T $\leftarrow$ $E_K$(S)
                $k \leftarrow$ selectLeft($r$, T)
                P $\leftarrow$ C $\oplus$ $k$
                output (P)
                Pre_$k \leftarrow k$
                $i \leftarrow i + 1$
        }
}
```

47. We have written the decryption algorithm assuming all *N* blocks are ready for decryption to match the encryption algorithm in the text. However, as described in the text, the algorithm can be written if ciphertext blocks are ready only one at a time.

```
CTR_Decryption (IV, K, C[1 … N])
{
        Counter $\leftarrow$ IV
        for ($i = 1$ to $N$)
        {
                Counter $\leftarrow$ (Counter $+ i - 1$) mod $2^N$
                $k_i \leftarrow E_K$ (Counter)
                P[$i$] $\leftarrow$ C[$i$] $\oplus$ $k$[$i$]
        }
        return (P[1 … N])
}
```

48.

```
shiftLeft ($r$, S[1 … $n$])
{
        for ($i = n$ downto $r$)
                S[$i$] $\leftarrow$ S[$i - $r]
        for ($i = 0$ to $r$)
                S[$i$] $\leftarrow$ 0
        return S
}
```

49.

```
concatenate (X[1 … $n$], Y[1 … $r$])
{
        for ($i = 1$ to $r$)
                X[$n - r + 1$] $\leftarrow$ Y[i]
        return X
}
```

50.

```
selectLeft ($r$, X[1 … $n$])
{
        for ($i = 1$ to $r$)
                Y[$i$] $\leftarrow$ X[i]
        return Y
}
```

CHAPTER 9

Mathematics of Cryptography: Part 3

(Solution to Practice Set)

Review Questions

1. A positive integer is a *prime* if and only if it is exactly divisible by two integers, 1 and itself. A *composite* is a positive integer with more than two divisors.

2. Two positive integers, a and b, are said to be *relatively prime,* or *coprime*, if gcd (a, b) = 1.

3.

a. The function $\pi(n)$ finds the number of primes smaller than or equal to n.

b. Euler's phi-function, $\phi(n)$, which is sometimes called the Euler's totient function, finds the number of integers that are both smaller than n and relatively prime to n.

4. The Sieve of Eratosthenes is a method to find all primes less than n. We write down all the integers between 2 and n. We cross out all integers divisible by 2 (except 2 itself). We cross out all integers divisible by 3 (except 3 itself). And so on. When we have crossed out all integers divisible by all primes less than $\sqrt{n}$, the remaining integers are primes.

5. We discussed two versions of Fermat's little theorem. The first version says that if p is a prime and a is an integer such that p does not divide a, then we have $a^{p-1} \equiv 1 (\text{mod } p)$. The second Version removes the condition on a. It says that if p is a prime and a is an integer, then $a^p \equiv a \ (\text{mod } p)$. Two immediate applications of this theorem is to find solutions to exponentiation and multiplicative inverses when the modulus is a prime.

6. Euler's Theorem is a generalization of Fermat's little theorem. The modulus in the Fermat theorem is a prime, the modulus in Euler's theorem is an integer. We introduce two versions of this theorem. The first version says that if a and n are coprime, then $a^{\phi(n)} \equiv 1 \ (\text{mod } n)$. The second version says if $n = p \times q$, $a < n$, and k *is* an integer, then $a^{k \times \phi(n) + 1} \equiv a$ (mod n). Two immediate applications of this theorem is to find solutions to exponentiation and multiplicative inverses when the modulus is a composite but we can easily find the value of ϕ(n).

7.

- **a.** Mersenne defined the formula $\mathbf{M_p = 2^p - 1}$ that was supposed to enumerate all primes. However, not all Mersenne numbers are primes.
- **b.** Fermat defined the formula $\mathbf{F_n = 2^{2^n} + 1}$ that was supposed to enumerate all primes. However, not all Fermat's numbers are primes.

8. A deterministic algorithm always gives a correct answer; a probabilistic algorithm gives an answer that is correct most of the time, but not all of the time. We mention two deterministic algorithms for primality testing: divisibility and AKS. We discussed three probability algorithms for primality testing: Fermat, square-root, and Miller-Rabin.

9. We mentioned the trial-division, Fermat, Polard $p - 1$, Polard rho, quadratic sieve, and number field sieve.

10. The Chinese remainder theorem (CRT) solves a set of congruent equations with one variable but different moduli, which are relatively prime. The Chinese remainder theorem has several applications in cryptography. One is to solve quadratic congruence. The other is to represent a very large integer in terms of a list of small integers.

11. A quadratic congruence is an equations of the form $a_2x^2 + a_1x + a_0 \equiv 0 \pmod{n}$. In this text, we have limited our discussion to equations of the form $x^2 \equiv a \pmod{n}$. In this equation a is called a quadratic residue (QR) if the equation has two solutions; a is called quadratic nonresidue (QNR) if the equation has no solutions.

12. The group $\mathbf{G} = <\mathbf{Z}_p{}^*, \times>$ has primitive roots. The primitive roots can be thought as the base of logarithm. Given $x = \log_g y$ for any element y in the set, there is another element x that is the log of y in base g. This type of logarithm is called discrete logarithm.

Exercises

13.

- **a.** The number of primes between 100,000 and 200,000 can be found as $\pi(200{,}000) - \pi(100{,}000)$. Using the upper and lower limits devised by Gauss and Lagrange, we have

$$16385 < \pi(\mathbf{200{,}000}) < \mathbf{17985} \quad \rightarrow \quad \pi(\mathbf{200{,}000}) \approx 17200$$
$$8688 < \pi(\mathbf{100{,}000}) < \mathbf{9587} \quad \rightarrow \quad \pi(\mathbf{100{,}000}) \approx 9138$$
$$\pi(\mathbf{200{,}000}) - \pi(\mathbf{100{,}000}) \approx \mathbf{17200 - 9138} \approx \mathbf{8062}$$

- **b.** The number of composites between 100,000 and 200,000 is

$$\mathbf{100{,}000 - 8062 \approx 91938}$$

- **c.** The ratio of primes to composites in the above range is 8062/91938 or approximately 8.77 percent. This ratio for numbers between 1 to 10 (without considering 1 and 10) is 4/4 or 100 percent.

14.

a. All of these numbers have the same largest prime factor, because

100	=	$(2 \times 5)^2$	=	$2^2 \times 5^2$	→	**Largest factor is 5**
1000	=	$(2 \times 5)^3$	=	$2^3 \times 5^3$	→	**Largest factor is 5**
10,000	=	$(2 \times 5)^4$	=	$2^4 \times 5^4$	→	**Largest factor is 5**
100,000	=	$(2 \times 5)^5$	=	$2^5 \times 5^5$	→	**Largest factor is 5**
1,000,000	=	$(2 \times 5)^6$	=	$2^6 \times 5^6$	→	**Largest factor is 5**

b. However, when we add 1 to each integer, the largest prime factor changes unpredictably.

101	=	1×101	→	**Largest factor is 101**
1001	=	$7 \times 11 \times 13$	→	**Largest factor is 13**
10,001	=	73×137	→	**Largest factor is 137**
100,000	=	11×9091	→	**Largest factor is 9091**
1,000,000	=	101×9901	→	**Largest factor is 9901**

15. When an integer is divided by 4, the remainder is either 0, 1, 2, or 3. This means that an integer can be written as $(4k + 0)$, $(4k + 1)$, $(4k + 2)$, or $(4k + 3)$, in which k is the quotient. An integer in the form $(4k + 0)$ or $4k$ is not a prime because it is divisible by 4. An integer in the form $(4k + 2)$ can be a prime only if $k = 0$ (the integer is 2 and it is the first prime). The other two forms, $(4k + 1)$ and $(4k + 3)$, can represent a prime or a composite. This means that any prime can be either in the form of $(4k + 1)$ or $(4k + 3)$. However, this does not mean that any integer in one of these forms is a prime.

16. We show some primes for each form:

a. $p = 5k + 1$:

$k = 2 \rightarrow p = 11$	$k = 6 \rightarrow p = 31$	$k = 8 \rightarrow p = 41$	$k = 10 \rightarrow p = 51$

b. $p = 5k + 2$:

$k = 0 \rightarrow p = 2$	$k = 1 \rightarrow p = 7$	$k = 3 \rightarrow p = 17$	$k = 7 \rightarrow p = 37$

c. $p = 5k + 3$:

$k = 0 \rightarrow p = 3$	$k = 2 \rightarrow p = 13$	$k = 4 \rightarrow p = 23$	$k = 8 \rightarrow p = 43$

d. $p = 5k + 4$:

$k = 3 \rightarrow p = 19$	$k = 5 \rightarrow p = 29$	$k = 9 \rightarrow p = 49$	$k = 11 \rightarrow p = 59$

17.

a. $\phi(29) = 29 - 1 = 28$ **(29 is a prime)**

b. $\phi(32) = \phi(2^5) = 2^5 - 2^4 = 16$ **(2 is a prime)**

c. $\phi(80) = \phi(2^4 \times 5^1) = (2^4 - 2^3) \times (5^1 - 5^0) = 8 \times 4 = 32$ **(2 and 5 are primes)**

d. $\phi(100) = \phi(2^2 \times 5^2) = (2^2 - 2^1) \times (5^2 - 5^1) = 2 \times 20 = 40$ **(2 and 5 are primes)**

e. $\phi(101) = 101 - 1 = 100$ **(101 is a primes)**

18.

a. $(2^{24} - 1) = (2^{12} - 1) \times (2^{12} + 1)$. Because none of the factors is 1, $(2^{24} - 1)$ is a composite.

b. $(2^{16} - 1) = (2^8 - 1) \times (2^8 + 1)$. Because none of the factors is 1, $(2^{16} - 1)$ is a composite.

c. In general, if a positive integer can be written in the form $(2^n - 1)$ and n is an even integer greater than 2, then $(2^n - 1) = (2^{n/2} - 1) \times (2^{n/2} + 1)$. If $n = 2$, then we have $(2^n - 1) = 3$, a prime.

19. We have $(10 = 3 + 7)$, $(24 = 11 + 13)$, $(28 = 11 + 17)$, and $(100 = 11 + 89)$.

20. Some of these primes are shown below. We can always check the primeness using Appendix H.

$n = 1$	$\rightarrow$	$(n^2 + 1) = 2$	$n = 2$	$\rightarrow$	$(n^2 + 1) = 5$
$n = 4$	$\rightarrow$	$(n^2 + 1) = 17$	$n = 6$	$\rightarrow$	$(n^2 + 1) = 37$
$n = 10$	$\rightarrow$	$(n^2 + 1) = 101$	$n = 14$	$\rightarrow$	$(n^2 + 1) = 197$
$n = 16$	$\rightarrow$	$(n^2 + 1) = 257$	$n = 20$	$\rightarrow$	$(n^2 + 1) = 401$
$n = 24$	$\rightarrow$	$(n^2 + 1) = 577$	$n = 26$	$\rightarrow$	$(n^2 + 1) = 677$

21.

a.

$$(5^{15} \bmod 13) = [(5^2 \bmod 13) \times (5^{13} \bmod 13)] \bmod 13$$
$$= [(-1 \bmod 13) \times (5 \bmod 13)] \bmod 13 = -5 \bmod 13 = 8 \bmod 13$$

b.

$$(15^{18} \bmod 17) = [(15 \bmod 17) \times (15^{17} \bmod 17)] \bmod 17$$
$$= [(-2 \bmod 17) \times (-2 \bmod 17)] \bmod 17 = 4 \bmod 17$$

c.

$$(456^{17} \bmod 17) = (456 \bmod 17) = 14 \bmod 17$$

d.

$$(145^{102} \bmod 101) = [(145^{101} \bmod 101) \times (145 \bmod 101)] \bmod 101$$
$$= [145 \times 145] \bmod 101 = [44 \times 44] \bmod 101 = 17 \bmod 101$$

22. We know that if p is a prime, $\boldsymbol{x^{-1} \bmod p = x^{p-2} \bmod p}$.

a.

$$5^{-1} \bmod 13 = 5^{13-2} \bmod 13 = 5^{11} \bmod 13 = 8 \bmod 13$$

b.

$$15^{-1} \bmod 17 = 15^{17-2} \bmod 17 = 15^{15} \bmod 17 = 8 \bmod 17$$

c.

$$27^{-1} \bmod 41 = 27^{41-2} \bmod 41 = 27^{39} \bmod 41 = 38 \bmod 41$$

d.

$$70^{-1} \bmod 101 = 70^{101-2} \bmod 101 = 70^{99} \bmod 101 = 13 \bmod 101$$

23. We know that if n is an integer, $x^{-1} \bmod n = x^{\phi(n)-1} \bmod n$.

a.

$$12^{-1} \bmod 77 = 12^{\phi(77)-1} \bmod 77 = 12^{59} \bmod 77 = 45 \bmod 77$$

b.

$$16^{-1} \bmod 323 = 16^{\phi(323)-1} \bmod 323 = 16^{287} \bmod 323 = 101 \bmod 323$$

c.

$$20^{-1} \bmod 403 = 20^{\phi(403)-1} \bmod 403 = 20^{359} \bmod 403 = 262 \bmod 403$$

d.

$$44^{-1} \bmod 667 = 44^{\phi(667)-1} \bmod 667 = 44^{615} \bmod 667 = 379 \bmod 667$$

24. For each Mersenne number, $M_p = 2^p - 1$, we try some integers in the form $2kp + 1$ to find a possible divisor.

a. $M_{23} = 2^{23} - 1 = 8{,}388{,}607$ is not a prime. When $k = 2$, $2kp + 1 = 47$ divides the number: $8{,}388{,}607 = 47 \times 178{,}481$. **$M_{23}$ is a composite**

b. $M_{29} = 2^{29} - 1 = 536{,}870{,}911$ is not a prime. When $k = 4$, $2kp + 1 = 233$ divides the number: $536{,}870{,}911 = 233 \times 1{,}103 \times 2{,}089$. **$M_{29}$ is a composite**

c. $M_{31} = 2^{31} - 1 = 2{,}147{,}483{,}647$ is a prime. If this number is a composite, it must have a divisor less than 46,341 (the square root of M_{31}), which means that k in $2kp + 1$ must be less than 748. We tried all $k = 0$ to 747 with no success. **M_{31} is a prime**.

25. It can be checked that if $2^n - 1$ is a prime, then n is a prime. However, there are some values of n for which $2^n - 1$ is a composite, but n is a prime ($n = 11$, for example). In other words, if n is a prime, $2^n - 1$ may or may not be a prime; if $2^n - 1$ is a prime, then n is a prime. This is to say that not all Mersenne numbers are primes. Since Mersenne's idea cannot be used for primality test, the fact stated in this problem cannot be used for primality test.

$2^2 - 1 = 3$ is a prime	$\rightarrow$	$n = 2$ is a prime
$2^3 - 1 = 7$ is a prime	$\rightarrow$	$n = 3$ is a prime
$2^4 - 1 = 15 = 3 \times 5$ is a composite	$\rightarrow$	$n = 4$ is a composite
$2^5 - 1 = 31$ is a prime	$\rightarrow$	$n = 5$ is a prime

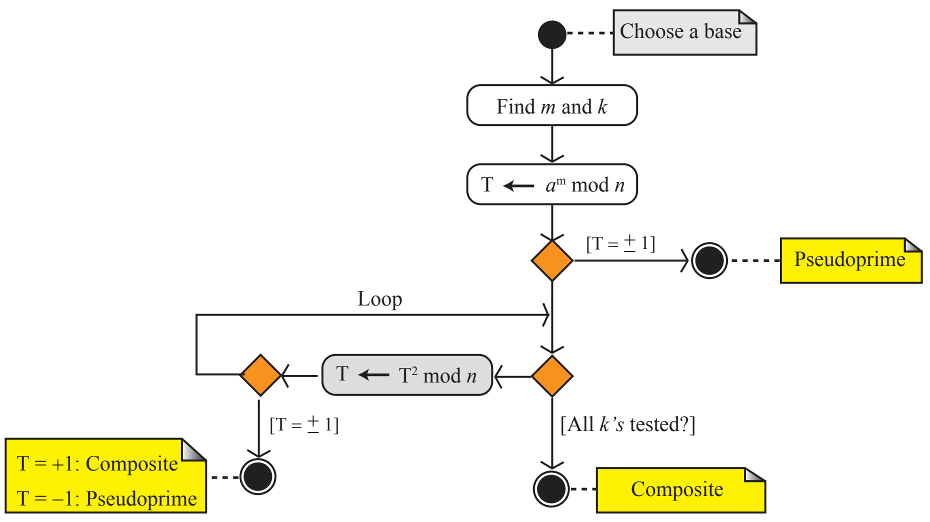
$2^6 - 1 = 63 = 3 \times 5$ is a composite	$\rightarrow$	$n = 6$ is a composite
$2^7 - 1 = 127$ is a prime	$\rightarrow$	$n = 7$ is a prime
$2^8 - 1 = 255 = 5 \times 51$ is a composite	$\rightarrow$	$n = 8$ is a composite
$2^9 - 1 = 511 = 7 \times 73$ is a composite	$\rightarrow$	$n = 9$ is a composite
$2^{10} - 1 = 1023 = 3 \times 341$ is a composite	$\rightarrow$	$n = 10$ is a composite
$2^{11} - 1 = 2047 = 23 \times 89$ is a composite	$\rightarrow$	$n = 11$ is a prime

26. The following shows the results of the Fermat test on this group of integers. Note that if the integer does not pass the test, it is definitely a composite; if it passes the test, it may be a prime. The last two integers pass the test, but we know that they are composite.

$n = 100$	$\rightarrow$	$2^{n-1} \bmod n = 88$ (not passed)	$\rightarrow$	composite
$n = 110$	$\rightarrow$	$2^{n-1} \bmod n = 72$ (not passed)	$\rightarrow$	composite
$n = 130$	$\rightarrow$	$2^{n-1} \bmod n = 88$ (not passed)	$\rightarrow$	composite
$n = 150$	$\rightarrow$	$2^{n-1} \bmod n = 88$ (not passed)	$\rightarrow$	composite
$n = 200$	$\rightarrow$	$2^{n-1} \bmod n = 88$ (not passed)	$\rightarrow$	composite
$n = 250$	$\rightarrow$	$2^{n-1} \bmod n = 62$ (not passed)	$\rightarrow$	composite
$n = 271$	$\rightarrow$	$2^{n-1} \bmod n = 1$ (passed)	$\rightarrow$	prime
$n = 341$	$\rightarrow$	$2^{n-1} \bmod n = 1$ (passed)	$\rightarrow$	composite
$n = 561$	$\rightarrow$	$2^{n-1} \bmod n = 1$ (passed)	$\rightarrow$	composite

27. The flow in the Miller-Rabin algorithm may be better understood using the chart in Figure S9.27. We follow the chart in each case.

Figure S9.27 *Solution to Exercise 27*



a. $n = 100 \rightarrow 100 - 1 = 99 \times 2^0 \rightarrow m = 99$ and $k = 0$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{99} \bmod 100 = 88}$

Loop is by-passed because $k = 0 \rightarrow$ Composite (100 is an even integer)

b. $n = 109 \rightarrow 109 - 1 = 27 \times 2^2 \rightarrow m = 27$ and $k = 2$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{27} \bmod 109 = 33}$
$k = 1$	$\rightarrow$	$\mathbf{T = T^2 \bmod 109 = 108 \bmod 109 = (-1) \bmod 109}$

Loop is broken because T = –1 $\rightarrow$ Pseudoprime (109 is actually a prime)

c. $n = 201 \rightarrow 201 - 1 = 25 \times 2^3 \rightarrow m = 27$ and $k = 3$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{25} \bmod 201 = 95}$
$k = 1$	$\rightarrow$	$\mathbf{T = T^2 \bmod 201 = 181 \bmod 201}$
$k = 2$	$\rightarrow$	$\mathbf{T = T^2 \bmod 201 = 199 \bmod 201}$

Loop is terminated $\rightarrow$ Composite ($201 = 3 \times 67$)

d. $n = 271 \rightarrow 271 - 1 = 135 \times 2^1 \rightarrow m = 135$ and $k = 1$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{135} \bmod 271 = 1 \bmod 271}$

T = +1 in the initialization step $\rightarrow$ Pseudoprime (271 is actually a prime)

e. $n = 341 \rightarrow 341 - 1 = 85 \times 2^2 \rightarrow m = 85$ and $k = 2$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{85} \bmod 341 = 32}$
$k = 1$	$\rightarrow$	$\mathbf{T = T^2 \bmod 341 = 1 \bmod 341}$

Loop is broken because T = +1 $\rightarrow$ Composite ($341 = 11 \times 31$)

f. $n = 349 \rightarrow 349 - 1 = 87 \times 2^2 \rightarrow m = 87$ and $k = 2$.

Pre-loop	$\rightarrow$	$\mathbf{T = 2^{87} \bmod 349 = 213}$
$k = 1$	$\rightarrow$	$\mathbf{T = T^2 \bmod 349 = 348 \bmod 349 = (-1) \bmod 349}$

Loop is broken because T = –1 $\rightarrow$ Pseudoprime (349 is actually a prime)

g. $n = 2047 \rightarrow 2047 - 1 = 1023 \times 2^1 \rightarrow m = 1023$ and $k = 1$.

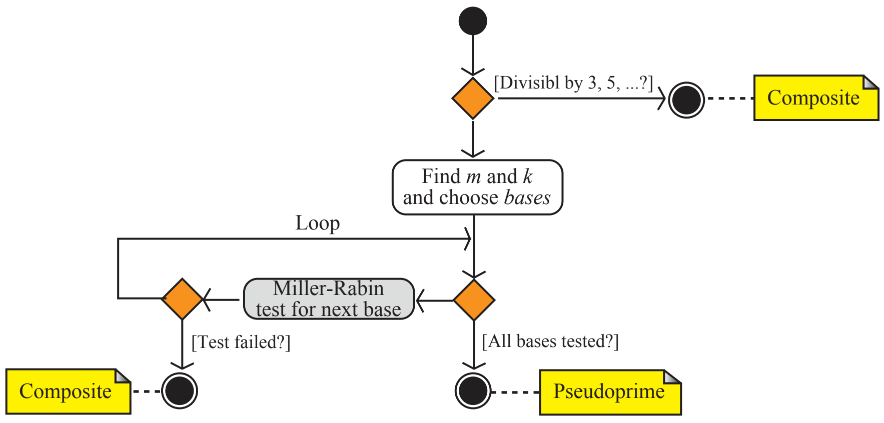
Pre-loop	$\rightarrow$	$\mathbf{T = 2^{1023} \bmod 2047 = 1 \bmod 2047}$

T = +1 in the initialization step $\rightarrow$ Pseudoprime (but $2047 = 23 \times 89$)

In this case, the test declares the integer 2047 as a pseudoprime, which is actually a composite.

28. We use the chart in Figure S9.28 to follow the recommended primality test that include several instances of Miller-Rabin test, each with a different base. For each number we use the base set (2, 3, 4).

Figure S9.28 *Solution to Exercise 28*



a. $n = 271$ is not divisible by 3, 5, 7, 11, 13, 17, 19, 23. We start Miller-Rabin tests. The tests tells us that 271 is a pseudoprime; it is in fact a prime.

Initialization: $271 - 1 = 135 \times 2^1 \rightarrow m = 135$ and $k = 1$		
$a = 2$	$\rightarrow$	$T = a^{135} \bmod 271 = 1$
$a = 3$	$\rightarrow$	$T = a^{135} \bmod 271 = -1$
$a = 4$	$\rightarrow$	$T = a^{135} \bmod 271 = 1$

The number passes three Miller-Rabin tests $\rightarrow$ Pseudoprime

b. $n = 3149$ is not divisible by 3, 5, 7, 11, 13, 17, 19, 23.We start Miller-Rabin tests. This integer fails the test; it is definitely a composite ($3149 = 47 \times 67$).

Initialization: $3149 - 1 = 787 \times 2^2 \rightarrow m = 787$ and $k = 2$			
$a = 2$	$\rightarrow$	$T = a^{787} \bmod 3149 = 2523$	$T = T^2 \bmod 3149 = 140$

The number fails the Miller-Rabin test for $a = 2$ $\rightarrow$ Composite

c. $n = 9673$ is not divisible by 3, 5, 7, 11, 13, but is divisible by 17. It is **composite** and no Miller-Rabin tests are needed ($9673 = 17 \times 569$).

29.

a. We test the claim using $(3 - 2)^p \bmod p = (3^p - 2) \bmod p$ with $x = 3$, $a = 2$, and some small primes.

$p = 2$	$(3 - 2)^2 \bmod 2 = 1$	$(3^2 - 2) \bmod 2 = 1$
$p = 3$	$(3 - 2)^3 \bmod 3 = 1$	$(3^3 - 2) \bmod 2 = 1$
$p = 7$	$(3 - 2)^5 \bmod 7 = 1$	$(3^7 - 2) \bmod 7 = 1$

b. We also test the claim using $x = 7$, $a = 3$, and some primes.

$p = 2$	$(7 - 3)^2 \bmod 2 = 0$	$(7^2 - 3) \bmod 2 = 0$
$p = 5$	$(7 - 3)^5 \bmod 5 = 4$	$(7^5 - 3) \bmod 5 = 4$
$p = 17$	$(7 - 3)^{17} \bmod 17 = 4$	$(7^{17} - 3) \bmod 17 = 4$

30. Generally, $p_n > n \times \ln n$. The following shows some primes and their approximations. As n becomes larger, the ratio of error becomes smaller.

$n = 1$	$p_n = 2$	$n \times \ln n = 0$
$n = 25$	$p_n = 97$	$n \times \ln n = 80.47$
$n = 168$	$p_n = 997$	$n \times \ln n = 860.80$
$n = 668$	$p_n = 4999$	$n \times \ln n = 4344.86$
$n = 10{,}000$	$p_n = 104{,}729$	$n \times \ln n = 92103{,}40$

31.

a.

$a_1 = 2 \quad m_1 = 7 \qquad a_2 = 3 \quad m_2 = 9 \qquad \rightarrow \qquad M = 63$

$M_1 = 9 \quad M_1^{-1} = 9^{-1} \bmod 7 = 4 \qquad ; \qquad M_2 = 7 \quad M_2^{-1} = 7^{-1} \bmod 9 = 4$

$$x = (2 \times 9 \times 4 + 3 \times 7 \times 4) \bmod 63 = 30$$

b.

$a_1 = 4 \quad m_1 = 5 \qquad a_2 = 10 \; m_2 = 11 \qquad \rightarrow \qquad M = 55$

$M_1 = 11 \quad M_1^{-1} = 11^{-1} \bmod 5 = 1 \qquad ; \qquad M_2 = 5 \quad M_2^{-1} = 5^{-1} \bmod 11 = 9$

$$x = (4 \times 11 \times 1 + 10 \times 5 \times 9) \bmod 55 = 54$$

c.

$a_1 = 7 \quad m_1 = 13 \qquad a_2 = 11 \; m_2 = 12 \qquad \rightarrow \qquad M = 156 \qquad M_1 = 12 \quad M_2 = 13$

$M_1 = 12 \quad M_1^{-1} = 12^{-1} \bmod 13 = 12 \qquad ; \qquad M_2 = 13 \quad M_2^{-1} = 13^{-1} \bmod 12 = 1$

$$x = (7 \times 12 \times 12 + 11 \times 13 \times 1) \bmod 156 = 59$$

32. The following show QRs and QNRs in each case.

a. QRs and QNRs in $\mathbf{Z}_{13}*$

QRs: 1, 3, 4, 9, 10, 12	**QNRs: 2, 5, 6, 7, 8, 11**

b. QRs and QNRs in $\mathbf{Z}_{17}*$

QRs: 1, 2, 4, 8, 9, 13, 15, 16	**QNRs: 3, 5, 6, 7, 10, 11, 12, 14**

c. QRs and QNRs in $\mathbf{Z}_{23}*$

QRs: 1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18	**QNRs: 5, 7, 10, 11, 14, 15, 17, 19, 20, 21, 22**

33.

a. The integer 4 is a QR in $\mathbf{Z}_7*$. Since $7 = 4 \times k + 3$ with $k = 1$, we can use the following expressions to find the solutions.

x: $4^{(7+1)/4}$ mod 7 = 2	**x: $-4^{(7+1)/4}$ mod 7 = −2**

b. The integer 5 is a QR in $\mathbf{Z}_{11}$*. Since 11 = $4 \times k + 3$ with $k = 2$, we can use the following expressions to find the two solutions:

x: $5^{(11+1)/4}$ mod 11 = 4	**x: $-5^{(11+1)/4}$ mod 11 = −4**

c. The integer 7 is not a QR in $\mathbf{Z}_{13}$* (see Exercise 32). **This equation has no solutions.**

d. The integer 12 is not a QR in $\mathbf{Z}_{17}$* (see Exercise 32). **This equation has no solutions.**

34.

a. $n = 14 = 2 \times 7 \rightarrow p_1 = 2$ and $p_2 = 7$. The four new equations are

$x^2 \equiv 4 \pmod 2 \rightarrow x \equiv \pm 0 \pmod 2$	$x^2 \equiv 4 \pmod 7 \rightarrow x \equiv \pm 2 \pmod 7$

We have four sets of equations: The solution gives us $x = 2$ and $x = 12$.

Set 1:	$x \equiv +0 \pmod 2$	$x \equiv +2 \pmod 7$
Set 2:	$x \equiv +0 \pmod 2$	$x \equiv -2 \pmod 7$
Set 3:	$x \equiv -0 \pmod 2$	$x \equiv +2 \pmod 7$
Set 4:	$x \equiv -0 \pmod 2$	$x \equiv -2 \pmod 7$

b. $n = 10 = 2 \times 5 \rightarrow p_1 = 2$ and $p_2 = 5$. The four new equations are

$x^2 \equiv 1 \pmod 2 \rightarrow x \equiv \pm 1 \pmod 2$	$x^2 \equiv 0 \pmod 5 \rightarrow x \equiv \pm 0 \pmod 5$

We have four sets of equations: The solution gives us $x = 5$.

Set 1:	$x \equiv +1 \pmod 2$	$x \equiv +0 \pmod 5$
Set 2:	$x \equiv +1 \pmod 2$	$x \equiv -0 \pmod 5$
Set 3:	$x \equiv -1 \pmod 2$	$x \equiv +0 \pmod 5$
Set 4:	$x \equiv -1 \pmod 2$	$x \equiv -0 \pmod 5$

c. $n = 33 = 3 \times 11 \rightarrow p_1 = 3$ and $p_2 = 11$.

$x^2 \equiv 1 \pmod 3 \rightarrow x \equiv \pm 1 \pmod 3$	$x^2 \equiv 7 \pmod{11} \rightarrow$ **no solutions**

Since the second equation has no solution, the equation $x^2 = 7$ mod 33 has no solution.

d. $n = 34 = 2 \times 17 \rightarrow p_1 = 2$ and $p_2 = 17$. The four new equations are

$x^2 \equiv 12 \pmod 2 \rightarrow x \equiv \pm 0 \pmod 2$	$x^2 \equiv 12 \pmod{17} \rightarrow$ **No solutions**

Since the second equation has no solution, the equation $x^2 = 12$ mod 34 has no solution.

35. We use tables based on Figure 9.7. We first calculate all powers of a's.

a. $y = 21^{24} \bmod 8 \rightarrow a = 21, x = 24 = 11000_2$. Shaded areas represent no moutiplications. All calculations are in modulo 8. The answer is $\boldsymbol{y = 1}$.

***a*'s**	x_i	$y = \mathbf{1} \bmod 8$
$a^1 = 5 \bmod 8$	0	$y = 1 \textbf{ mod 8}$
$a^2 = 1 \bmod 8$	0	$y = 1 \textbf{ mod 8}$
$a^4 = 1 \bmod 8$	0	$y = 1 \textbf{ mod 8}$
$a^8 = 1 \bmod 8$	1	$y = 1 \times 1 \bmod 8 = 1 \bmod 8$
$a^{16} = 1 \bmod 8$	1	$y = 1 \times 1 \bmod 8 = \mathbf{1} \bmod 8$

The table shows that we can stop whenever a^{x_i} becomes 1.

b. $y = 320^{23} \bmod 461 \rightarrow a = 320, x = 23 = 10111_2$. Shaded areas represent no multiplications. All calculations are in modulo 461. The answer is $\boldsymbol{y = 373}$.

***a*'s**	x_i	$y = \mathbf{1} \bmod 461$
$a^1 = 320 \bmod 461$	1	$y = 1 \times 320 \bmod 461 = 320 \bmod 461$
$a^2 = 58 \bmod 461$	1	$y = 320 \times 58 \bmod 461 = 120 \bmod 461$
$a^4 = 137 \bmod 461$	1	$y = 120 \times 137 \bmod 461 = 305 \bmod 461$
$a^8 = 329 \bmod 461$	0	$y = 305 \textbf{ mod 461}$
$a^{16} = 367 \bmod 461$	1	$y = 305 \times 367 \bmod 461 = \mathbf{373} \bmod 461$

c. $y = 1776^{41} \bmod 2134 \rightarrow a = 1776, x = 41 = 101001_2$. Shaded areas represent no multiplications. All calculations are in modulo 2134. The answer is $\boldsymbol{y = 698}$.

***a*'s**	x_i	$y = \mathbf{1} \bmod 2134$
$a^1 = 1776 \bmod 2134$	1	$y = 1 \times 1776 \bmod 2134 = 1776 \bmod 2134$
$a^2 = 124 \bmod 2134$	0	$y = \textbf{1776 mod 2134}$
$a^4 = 438 \bmod 2134$	0	$y = \textbf{1776 mod 2134}$
$a^8 = 1918 \bmod 2134$	1	$y = 1776 \times 1918 \bmod 2134 = 504 \bmod 2134$
$a^{16} = 1842 \bmod 2134$	0	$y = 504 \bmod 2134$
$a^{32} = 2038 \bmod 2134$	1	$y = 504 \times 2038 \bmod 2134 = \mathbf{698} \bmod 2134$

d. $y = 2001^{35} \bmod 2000 \to a = 2001$, $x = 35 = 100011_2$. Shaded areas represent no multiplications. All calculations are in modulo 2001. The answer is $\boldsymbol{y = 1}$.

***a*'s**	$\boldsymbol{x_i}$	$\boldsymbol{y} = \mathbf{1} \bmod 2000$
$a^1 = 1 \bmod 2000$	1	$y = 1 \times 1 \bmod 2000 = 1 \bmod 2000$
$a^2 = 1 \bmod 2000$	1	$y = 1 \times 1 \bmod 2000 = 1 \bmod 2000$
$a^4 = 1 \bmod 2000$	0	$y = 1 \bmod 2000$
$a^8 = 1 \bmod 2000$	0	$y = 1 \bmod 2000$
$a^{16} = 1 \bmod 2000$	0	$y = 1 \bmod 2000$
$a^{32} = 1 \bmod 2000$	1	$y = 1 \times 1 \bmod 2000 = \mathbf{1} \bmod 2000$

The table shows that we can stop whenever a^{x_i} becomes 1.

36.

a. The order of the group is $\phi(19) = 18$.

b. The following shows the order of each element:

ord (1) = 1	**ord(2) = 18**	**ord (3) = 18**	**ord (4) = 9**	**ord (5) = 9**	**ord (6) = 9**
ord (7) = 3	**ord(8) = 6**	**ord (9) = 9**	**ord (10) = 18**	**ord (11) = 3**	**ord (12) = 6**
ord (13) = 18	**ord(14) = 18**	**ord (15) = 18**	**ord (16) = 9**	**ord (17) = 9**	**ord (18) = 2**

c. The number of primitive roots is $\phi(\phi(19)) = \phi(18) = 6$.

d. The primitive roots are those element with order 18. They are **2**, **3**, **10**, **13**, **14**, and **15**.

e. We know that the group is cyclic because 19 is a prime. Any of the primitive root can generate all elements of the group. For example, if we choose $g = 2$ as the generator, we can generate all elements as shown below (all calculations are in modulo 19).

$\boldsymbol{g^1 \to 2}$	$\boldsymbol{g^2 \to 4}$	$\boldsymbol{g^3 \to 8}$	$\boldsymbol{g^4 \to 16}$	$\boldsymbol{g^5 \to 13}$	$\boldsymbol{g^6 \to 7}$
$\boldsymbol{g^7 \to 14}$	$\boldsymbol{g^8 \to 9}$	$\boldsymbol{g^9 \to 18}$	$\boldsymbol{g^{10} \to 17}$	$\boldsymbol{g^{11} \to 15}$	$\boldsymbol{g^{12} \to 11}$
$\boldsymbol{g^{13} \to 3}$	$\boldsymbol{g^{14} \to 6}$	$\boldsymbol{g^{15} \to 12}$	$\boldsymbol{g^{16} \to 5}$	$\boldsymbol{g^{17} \to 10}$	$\boldsymbol{g^{18} \to 1}$

f. We can create a table of discrete logarithms for bases **2**, **3**, **10**, **13**, **14**, and **15.**

$\boldsymbol{x}$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
$L_2(x)$	**18**	**1**	**13**	**2**	**16**	**14**	**6**	**3**	**8**	**17**	**12**	**15**	**5**	**7**	**11**	**4**	**10**	**9**
$L_3(x)$	**18**	**7**	**1**	**14**	**4**	**8**	**6**	**3**	**2**	**11**	**12**	**15**	**17**	**13**	**5**	**10**	**16**	**9**
$L_{10}(x)$	**18**	**17**	**5**	**16**	**2**	**4**	**12**	**15**	**10**	**1**	**6**	**3**	**13**	**11**	**7**	**14**	**8**	**9**
$L_{13}(x)$	**18**	**11**	**17**	**4**	**14**	**10**	**12**	**15**	**16**	**7**	**6**	**3**	**1**	**5**	**13**	**8**	**2**	**9**
$L_{14}(x)$	**18**	**13**	**7**	**8**	**10**	**2**	**6**	**3**	**14**	**5**	**12**	**15**	**11**	**1**	**17**	**16**	**4**	**9**
$L_{15}(x)$	**18**	**5**	**11**	**10**	**8**	**16**	**12**	**15**	**4**	**13**	**6**	**3**	**7**	**17**	**1**	**2**	**14**	**9**

37.

a. To solve the equation $x^5 \equiv 11 \pmod{17}$, we need to find a primitive root in the group $\mathbf{G} = <\mathbf{Z}_{17}*, \times>$ and the discrete logarithm table for that root. The first primitive root in this group is 3 (primitive roots are 3, 5, 6, 7, 10, 11, 12, and 14). The discrete logarithm table for this root (base) can be found as

x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
$L_3(x)$	**16**	**14**	**1**	**12**	**5**	**15**	**11**	**10**	**2**	**3**	**7**	**13**	**4**	**9**	**6**	**8**

We then apply the function L_3 to both sides of the congruence. Note that the working modulus is $\phi(17) = 16$ and $L_3(11) = 7$ (from the table).

$$L_3(x^5) \equiv L_3(11) \pmod{16} \rightarrow \quad 5 \times L_3(x) \equiv 7 \pmod{16} \quad \rightarrow 5 \times L_3(x) \equiv 7 \pmod{16}$$

Now we need to solve the congruence equation $5 \times L_3(x) \equiv 7 \pmod{16}$. Recall from Chapter 2 that this equation has only one solution because gcd (5, 16) = 1.

$$L_3(x) \equiv 5^{-1} \times 7 \pmod{16} \equiv \ 11 \pmod{16}$$

Now we can use the table to find x if $L_3(x) = 11$; the answer is $\boldsymbol{x = 7}$, which can be checked as $7^5 \equiv 11 \pmod{17}$. We can also write a program to test all of values of x from 1 to 17 to see if any of this values satisfies the equation. We did so; the only value is $\boldsymbol{x = 7}$.

b. To solve the equation $2x^{11} \equiv 22 \pmod{19}$ or $2x^{11} \equiv 3 \pmod{19}$, we need to find a primitive root in the group $\mathbf{G} = <\mathbf{Z}_{19}*, \times>$ and the discrete logarithm table for that root. The first primitive root in this group is 2 (see Exercise 36). The discrete logarithm table for this root (base) can be found as

x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
$L_2 x$	**18**	**1**	**13**	**2**	**16**	**14**	**6**	**3**	**8**	**17**	**12**	**15**	**5**	**7**	**11**	**4**	**10**	**9**

We then apply the function L_2 to both sides of the congruence. Note that the working modulus is $\phi(19) = 18$, $L_2(2) = 1$ and $L_2(3) = 13$.

$$L_2(2x^{11}) \equiv L_2(3) \pmod{18} \quad \rightarrow \quad L_2(2) + 11 \times L_2(x) \equiv L_2(3) \pmod{18}$$
$$\rightarrow \ 1 + 11 \times L_2(x) \equiv 13 \pmod{18} \ \rightarrow \ 11 \times L_2(x) \equiv 12 \pmod{18}$$

Now we need to solve the congruence equation $11 \times y \equiv 12 \pmod{18}$. Recall from Chapter 2 that this equation has only one solution because gcd (11, 18) = 1.

$$11 \times y \equiv 12 \pmod{18} \ \rightarrow \ y \equiv 11^{-1} \times 12 \pmod{18} \equiv 5 \ \times 12 \pmod{18} \equiv 6 \pmod{18}$$

Now we can use the table to find x if $L_2(x) = 6$; the answer is $\boldsymbol{x = 7}$, which can be checked as $2 \times 7^{11} \equiv 3 \pmod{19}$. We can also write a program to test all of values of x from 1 to 19 to see if any of this values satisfies the equation. We did so; the only value is $\boldsymbol{x = 7}$.

c. The equation $5x^{12} + 6x \equiv 8 \pmod{23}$ cannot be solved using the discrete logarithm discussed in this chapter because there is no property of discrete logarithm to allows us extract $L(x)$ from $L(5x^{12} + 6x)$. However, we can write a program to test all of values of x from 1 to 22 to see if any of this values satisfies the equation. We did so, but find no value of x satisfying the congruence; the congruence has no solution.

38. One million operations per second means 3,600,000,000 operations per hour.

a. If we assume that the complexity of divisibility test is 2^{n_b} (which is very unrealistic), then

$$2^{nb} = 3{,}600{,}000{,}000 \;\rightarrow\; 2^{nb} \approx 2^{32} \quad n_b \approx 32$$

This means $\boldsymbol{n < 2^{32}}$

b. The complexity of ASK is $(\log_2 n_b)^{12}$.

$$(\log_2 n_b)^{12} = 3{,}600{,}000{,}000 \;\rightarrow \log_2 n_b = 6.25 \;\rightarrow n_b \approx 2^{6.25} \;\rightarrow n_b \approx 76$$

This means $\boldsymbol{n < 2^{76}}$

c. The complexity of Fermat is n_b if we use Fast Exponentiation algorithm.

$$n_b = 3{,}600{,}000{,}000$$

This means $\boldsymbol{n < 2^{3{,}600{,}000{,}000}}$

However, as we know the algorithm is probabilistic. We are not sure if the result is correct.

d. The complexity of square root test is n or 2^{nb}

$$2^{n_b} = 3{,}600{,}000{,}000 \;\rightarrow\; 2^{n_b} \approx 2^{32} \quad n_b \approx 32$$

This means $\boldsymbol{n < 2^{32}}$

However the test is not deterministic. We are not sure if the result is correct.

e. The main operation in Miller-Rabin test is Fermat test, so the complexity is somehow more than Fermat. However, the test is not deterministic.

39. One million operations per second means 3,600,000,000 operations per hour.

a. The complexity of trial division method is exponential (2^{n_b}).

$$2^{n_b} = 3{,}600{,}000{,}000 \;\rightarrow\; 2^{n_b} \approx 2^{32} \quad n_b \approx 32$$

This means $\boldsymbol{n < 2^{32}}$

b. The complexity of Fermat method is subexponential or $2^{p(\log_2 n_b)}$. For simplicity, we assume the complexity to be $2^{(\log_2 n_b)2}$.

$$2^{(\log_2 n_b)2} = 3{,}600{,}000{,}000 \rightarrow 2^{(\log_2 n_b)2} \approx 2^{32} \rightarrow (\log_2 n_b)^2 \approx 32$$
$$\rightarrow (\log_2 n_b) \approx 5.7 \rightarrow n_b \approx 2^{5.7} \rightarrow 52$$

This means $\boldsymbol{n} < \mathbf{2^{52}}$

c. The complexity of Polard rho method is exponential or ($2^{n_b/4}$).

$$2^{n_b/4} = 3{,}600{,}000{,}000 \rightarrow 2^{n_b/4} \approx 2^{32} \quad n_b \approx 128$$

This means $\boldsymbol{n} < \mathbf{2^{128}}$

d. The complexity here is e^C where $C = (\ln n \ \ln\ln n)^{1/2}$. Since it is very difficult to calculate n in this case, we assume that $C = \ln n)^{1/2}$ or $C = (\ln n)$.

$$e^{\ln n} = 3{,}600{,}000{,}000 \rightarrow \ln n = 3{,}600{,}000{,}000$$

This means $\boldsymbol{n} < \mathbf{e^{3,600,000,000}}$

e. The complexity here is e^C where $C = 2(\ln n)^{1/3}(\ln\ln n)^{2/3}$. Since it is very difficult to calculate n in this case, we assume that $C = 2(\ln n)^{1/3}(\ln n)^{2/3} = 2(\ln n)$

$$e^{2\ln n} = 3{,}600{,}000{,}000 \rightarrow \ln n = 1{,}800{,}000{,}000$$

This means $\boldsymbol{n} < \mathbf{e^{1,800,000,000}}$

40.

```
Square_and_Multiply (a, x, n)
{
    y ← 1
    for (i = 0 to n_b − 1)
    {
        if (a = 1)
            return y                        // no need to continue
        if (x_i = 1)
            y ← y × a mod n
        a ← a^2 mod n
    }
    return y
}
```

41.

```
Square_and_Multiply (a, x, n)
{
    y ← 1
    for (i = n_b − 1 downto 0)
    {
        a ← a^2 mod n
        if (x_i = 1)
            y ← y × a mod n
    }
    return $y$
}
```

42.

```
Square_and_Multiply ($a, x, n$)
{
    $y \leftarrow 1$
    while  ($x > 0$)
    {
        if  ($x \bmod 2 = 1$)
            $y \leftarrow y \times a \bmod n$          // not needed in the last round
        $a \leftarrow a^2 \bmod n$
        $x \leftarrow x / 2$                           // integer division
    }
    return $y$
}
```

43.

```
FermatPrimalityTest ($a, n$)                            // We can use different bases
{
    $y \leftarrow$ Square_and_Multiply ($a, n - 1, n$)   // See Algorithm 9.7 in the text
    if ($y = 1$)
        return ($n$ is probably a prime)
    return ($n$ is a composite)
}
```

44.

```
SquareRootPrimalityTest ($n$)
{
    for  ($a = 2$  to  $n/2 + 1$)                      // $n$ is normally odd
    {
        $x \leftarrow a^2 \bmod n$
        $y \leftarrow (-a)^2 \bmod n$
        if ($x = 1 \bmod n$) or ($y = 1 \bmod n$)
            return ($n$ is a composite)
    }
    return ($n$ is probably a prime)
}
```

45.

```
ChineseRemainderTheorem($k$, a[1 … $k$], m[1 … $k$])
{
        M ← 1
        for  ($i$ = 1  to  $k$)
                M ←  M × m[$i$]
        for  ($i$ = 1  to  $k$)
        {
                M[$i$] ←  M / m[$i$]
                InvM[$i$] ← M[$i$]$^{-1}$ mod m[$i$]
        }
        $x$ ← 0
        for  ($i$ = 1  to  $k$)
                $x$ ←  [$x$ + (a[$i$] × M[$i$] × InvM[$i$])] mod M
        return $x$
}
```

46.

```
FindQuadraticResidues ($p$)                          // $p$ is a prime
{
        for  ($a$ = 2  to  $n/2+1$)                  // $n$ is normally odd
        {
                $x$ ← Square_and_Multiply ($a$, ($p$ − 1)/2, $p$)
                if ($x$  = 1 mod $p$)
                        insert $a$ in QRList
                else
                        insert $a$ in QNRList
        }
        return QRList and QNRList
}
```

47.

```
FindFirstPrimitiveRoot ($p$)                         // $p$ is a prime
{
        for  ($a$ = 2  to  $p-1$)
        {
                $i$ ← 1
                while ($a^i$ mod $p \neq 1$)
                {
                        $i$ ← $i$ + 1
                }
                if ($i = p - 1$)                     // order $a = \phi(p)$
                        return $a$
        }
}
```

48.

```
FindAllPrimitiveRoots ($p$)                    // $p$ is a prime
{
    for ($a = 2$ to $p - 1$)
    {
        $i \leftarrow 1$
        while ($a^i \bmod p \neq 1$)
        {
            $i \leftarrow i + 1$
        }
        if ($i = p - 1$)                       // order $a = \phi(p)$
            Insert $a$ to PrimitiveRootList
    }
    return PrimitiveRootList
}
```

49.

```
FindAllDiscreteLogs ($p$)                      // $p$ is a prime
{
    PrimitiveRootList $\leftarrow$ FindAllPrimitiveRoots($p$)     // See Exercise 48
    Create a DiscreteLogTable sorted on $y$
    for (each $g$ in PrimitiveRootList)
    {
        for ($x = 1$ to $p - 1$)
        {
            $y \leftarrow g^x \bmod p$
            insert $x$ to $L_g$ row of DiscreteLogTable under $y$ column
        }
    }
    return DiscreteLogTable
}
```

CHAPTER 10

Symmetric-Key Cryptography

(Solution to Practice Set)

Review Questions

1. Symmetric-key cryptography is based on sharing secrecy; asymmetric-key cryptography is based on personal secrecy.

2. In asymmetric-key cryptography, each entity has a pair of public/private key. The public key is universal; the private key is personal. In symmetric-key cryptography a shared secret key is used for secret communication between two entities.

3. In cryptography, a trapdoor is a secret with which Bob can use a feasible algorithm to decrypt the ciphertext. If Eve does not know the trapdoor, she needs to use an algorithm which is normally infeasible.

4. In the knapsack cryptosystem, if we are told which elements, from a predefined set of numbers, are in a knapsack, we can easily calculate the sum of the numbers; if we are told the sum, it is difficult to say which elements are in the knapsack.

a. The one-way function is the multiplication of the row matrix $\boldsymbol{x}$ by the column matrix $\boldsymbol{a}$ to get $s = \boldsymbol{x} \times \boldsymbol{a}$. Given $\boldsymbol{x}$ and $\boldsymbol{a}$, it is easy to calculate $\boldsymbol{s}$; given s and $\boldsymbol{a}$, it is difficult to calculate $\boldsymbol{x}$.

b. The trapdoor in this system is the value of r and n that allow Bob to calculate the value of $s' = r^{-1} \times s$. In the matrix multiplication $\boldsymbol{s}' = \boldsymbol{x} \times \boldsymbol{b}$, the matrix $\boldsymbol{b}$ can be calculated if it is a superincreasing column matrix (tuple).

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

d. The security of the system depends on the size of the two matrices $\boldsymbol{x}$ and $\mathbf{a}$. If they are very large, it is difficult to find $\boldsymbol{x}$ given s and $\boldsymbol{a}$. However, today, knapsack cryptosystem is not considered secure; it is broken.

5. RSA uses two exponents, e and d, where e is public and d is private. Alice calculates $C = P^e \bmod n$ to create ciphertext C from plaintext P; Bob uses $P = C^d \bmod n$ to retrieve the plaintext sent by Alice.

a. The one-way function is the $C = P^e \bmod n$. Given P and e, it is easy to calculate C; given C and e, it difficult to calculate P if n is very large.

b. The trapdoor in this system is the value of d, which enables Bob to use $P = C^d$ mod n.

c. The public key is the tuple (e, n). The private key is d.

d. The security of RSA mainly depend on the factorization of n. If n is very large, and the value of e and d are chosen properly, the system is secure.

6. The Rabin cryptosystem is a variation of the RSA cryptosystem. RSA is based on the exponentiation congruence; Rabin is based on quadratic congruence. The Rabin cryptosystem can be thought of as an RSA cryptosystem in which $e = 2$ and $d = 1/2$.

a. The one way function is $C = P^2$ mod n. Given P, it is easy to calculate C; given C, it difficult to calculate P if n is very large.

b. The trapdoor is the factorization of $n = p \times q$. Bob knows the value of p and q, but Eve does not. Bob uses the Chinese remainder theorem and the values of p and q, to find P.

c. The public key is n. The private key is the tuple (p and q).

d. Security of Rabin cryptosystem is basically depends on how it difficult to factor n.

7. ElGamal is based on discrete logarithm problem. The plaintext is masked with e_1^{rd} to create the ciphertext. Part of the mask is created by Bob and become public; the other part is created by Alice.

a. The one-way function is C = mask (P). Given P and the mask, it is easy to calculate the C; given C is difficult to unmask P.

b. The trapdoor is the value of d that enables Bob to unmask C.

c. The public key is (e_1, e_2 and n). The private key is d.

d. The security of ElGamal depends on two points; p should be very large and Alice needs to select a new r for each encryption.

8. ECC is based on elliptic curves. We can defined a group GF(p) or GF(2^n) in which the elements are points on an elliptic curve. ECC simulates the idea of ElGamal using the above-mentioned groups.

a. The one-way function in ECC is the idea of multiplying an integer by a point to get a new point on the curve. If the original point is given, the new point can be calculated with polynomial complexity. If the new point is given, it is very hard to calculate the original point without knowing the trapdoor.

b. The trapdoor is the value of d that enables Bob to calculate the original point on the curve using an algorithm with polynomial complexity.

c. The public key is (e_1, e_2, and E) in which e_1 and e_2 are two points on the curve E. The private key is d.

d. The security of ECC is based on the difficulty of solving elliptic curve logarithms.

9. An elliptic curve is an equation in two variables similar to the equations used to calculate the length of a curve in the circumference of an ellipse. Elliptic curves with points belonging to the groups GF(p) or GF(2^n) are used in cryptography.
10. Operations are basically adding two points on the curve to get a new point on the curve.

Exercises

11. We use the 7-bit representation of character "a" as the plaintext.

Key Generation:

$t = [287, 451, 943, 762, 564, 86, 623] \rightarrow a = [623, 86, 564, 287, 451, 943, 762]$

Encryption:

Plaintext: $x = [1, 1, 0, 0, 0, 0, 1] \quad \rightarrow \quad$ Ciphertext: $s = 1471$

Decryption:

$$\mathbf{s' = (41^{-1}, 1471) \bmod 1001 = 293 \times 1471 \bmod 1001 = 573}$$

$$\mathbf{x' = [0, 0, 0, 1, 0, 1, 1] \rightarrow \text{Plaintext } x = [1, 1, 0, 0, 0, 0, 1]}$$

12.

a. $\phi(n) = \phi(13 \times 17) = 192 \rightarrow d = e^{-1} \bmod \phi(n) = 5^{-1} \bmod 192 =$ **77**

b. $\phi(n) = \phi(31 \times 127) = 3780 \rightarrow d = e^{-1} \bmod \phi(n) = 17^{-1} \bmod 3780 =$ **3113**

c. $n = p \times q = 437 \rightarrow \phi(n) = 396$. **Since gcd($n$, e) $\neq$ 1, we cannot calculate d for this problem**. This shows that we need to choose the value of e carefully. In this case $\phi(n) = 396 = 2^2 \times 3^2 \times 1$, which means e cannot be 2, 3, or 11. We can have $e = 5$, which means $d = 5^{-1} \bmod 396 = 317$.

13. $n = 187 = 17 \times 11 \rightarrow \phi(n) = 17 \times 11 = 160 \rightarrow d = e^{-1} \bmod \phi(n) = 113$. This proves that the value of n in RSA must be very large.We could find d because we could factor n. The modulus must be large enough to make the factorization infeasible.

14. We can create two equations of two unknowns (p and q):

$$\mathbf{p \times q = n}$$
$$\mathbf{(p - 1) \times (q - 1) = \phi(n)}$$

If we let A = $(n - \phi(n) + 1)$, then $p^2 - \text{A} \times p + n = 0$. This means

$$\mathbf{p = [A + (A^2 - 4 \times n)^{1/2}] / 2 \qquad \text{and} \qquad q = [A - (A^2 - 4 \times n)^{1/2}] / 2}$$

For example, assume $n = 209$ and $\phi(n) = 180$. Then A = 209 − 180 + 1 = 30. We have $p = 19$ and $q = 11$.

15. In a real situation, the value of n is so large that it is impossible for Alice to check if n and e are chosen properly. In this toy problem, it is easy to see that n is not properly selected because $n = 100$ cannot be factored into two primes ($n = p \times q$). Although we can still encrypt the message using $e = 13$ and $n = 100$, the encrypted message cannot be decrypted. The problem prove that Bob needs to first select p and q and be sure that they are primes before calculating $n = p \times q$. After the correct selection of n, then e needs to be selected in such a way that $\phi(n)$ and e be coprimes. **This problem has no solution.**

16. Although in real life Alice cannot check to see if n and e are properly selected by Bob, in this toy problem she can. The value of n is proper because $n = 107 \times 113$ (two primes). The value of e is also proper because e, which is 13 and $\phi(n)$, which is 11872, are coprime.

a. The plaintext is: **19070818260818261914200607**

b. For encryption, we create 4-digit blocks so that the value of each block be less than n.

$P_1 = 1907$	$\rightarrow$	$C_1 = 1907^{13} \bmod 12091$	$= 10614$
$P_2 = 0818$	$\rightarrow$	$C_2 = 0818^{13} \bmod 12091$	$= 7787$
$P_3 = 2608$	$\rightarrow$	$C_3 = 2608^{13} \bmod 12091$	$= 1618$
$P_4 = 1826$	$\rightarrow$	$C_4 = 1826^{13} \bmod 12091$	$= 10717$
$P_5 = 1914$	$\rightarrow$	$C_5 = 1914^{13} \bmod 12091$	$= 4084$
$P_6 = 2006$	$\rightarrow$	$C_6 = 2006^{13} \bmod 12091$	$= 6558$
$P_7 = 07$	$\rightarrow$	$C_7 = 07^{13} \bmod 12091$	$= 6651$

c. In this toy problem, we can easily find $d = e^{-1} \bmod \phi(n) = 3653$. Each block can be decrypted as

$C_1 = 10614$	$\rightarrow$	$P_1 = 10614^{3653} \bmod 12091$	$= 1907$
$C_2 = 7787$	$\rightarrow$	$P_2 = 7787^{3653} \bmod 12091$	$= 0818$
$C_3 = 1618$	$\rightarrow$	$P_3 = 1618^{3653} \bmod 12091$	$= 2608$
$C_4 = 10717$	$\rightarrow$	$P_4 = 10717^{3653} \bmod 12091$	$= 1826$
$C_5 = 4084$	$\rightarrow$	$P_5 = 4084^{3653} \bmod 12091$	$= 1914$
$C_6 = 6558$	$\rightarrow$	$P_6 = 6558^{3653} \bmod 12091$	$= 2006$
$C_7 = 6651$	$\rightarrow$	$P_7 = 6651^{3653} \bmod 12091$	$= 07$

The plaintext is:19070818**26**0818**26**1914200607 or “THIS IS TOUGH”.

17.

a. If $e = 1$, there is no encryption: C = P. If Eve intercepts the ciphertext, she has the plaintext.

b. If $e = 2$, the cipher is actually Rabin, not RSA.

18. Eve can use a type of plaintext attack called short-message attack if she knows that each character is encrypted separately. Eve can write a simple program using e and n to find all plaintext/ciphertext character as shown in the following table.

P = 0	P = 1	P = 2	P = 3	P = 4	P = 5	P = 6
C = 0	C = 1	C = 13958	C = 2459	**C = 6625**	C = 7340	C = 8320
P = 7	P = 8	P = 9	P = 10	P = 11	P = 12	P = 13
C = 8911	C = 10247	C = 15310	C = 16008	C = 18021	C = 12029	C = 2232
P = 14	P = 15	P = 16	P = 17	P = 18	P = 19	P = 20
C = 4670	C = 13504	C = 11913	C = 10706	**C = 2968**	C = 8539	C = 5671
P = 21	P = 22	P = 23	P = 24	P = 25		
C = 11831	C = 15284	C = 4718	**C = 17863**	C = 1160		

Eve knows that the plaintext is made of P = 4, P = 0, P = 18, and P = 24. The plaintext is "easy". Eve can break the code because the set of plaintext values is small. Eve can try all of them using a program. The set should be very large. Even if Alice has short pieces of messages to send, she need to pad them to make the set large (see OAEP encryption/decryption).

19. Eve has intercepted C = 57

a. Eve chooses X = 17 (which is in $\mathbf{Z}_{143}$*).

b. Eve calculates $Y = C \times 17^7 \bmod 143 = 57 \times 17^7 \bmod 143 = 137$

c. Eve sends 137 to Bob and ask to decrypt it. The response is Z = 136 (We know how to calculate this because we can easily find $d = 103$, but we assume that Eve cannot; she needs to send the ciphertext to Bob and receive the plaintext.

d. $P = (Z \times X^{-1}) \bmod 143 = (136 \times 17^{-1}) \bmod 143 = 8 \bmod 143$. Which is the plaintext. This shows that RSA is very vulnerable to chosen-ciphertext attack.

20.

Intercepted ciphertext is C = 22.
$C_1 = C^e \bmod n = 22^3 \bmod 35 = 8$.
$C_2 = C1^e \bmod n = 8^3 \bmod 35 = 22$.
Since $C_2 = C = 22$, $P = C_1 = 8$.

21. One solution is to use different padding with each plaintext. As we discussed in the text, using OAEP encryption, each time with different random r, removes the relationships between different plaintexts that are sent by Alice to Bob.

22. We first find $n = p \times q = 47 \times 11 = 517$.

a. $C = P^2 \bmod 517 = 17^2 \bmod 517 = 289$.

b.

$a_1 = +\, C^{(p+1)/4} \bmod p = +\, 289^{12} \bmod 47 = +\, 17$ and $a_2 = -17$.
$b_1 = +\, C^{(q+1)/4} \bmod q = +\, 289^3 \bmod 11 = +\, 5$ and $b_2 = -\, 5$.

We have the following four sets, each of two equations to solve:

$\mathbf{P_1 \equiv +17 \pmod{47}}$	$\mathbf{P_1 \equiv +5 \pmod{11}}$	$\rightarrow$	$P_1 = 346$
$\mathbf{P_2 \equiv +17 \pmod{47}}$	$\mathbf{P_2 \equiv -5 \pmod{11}}$	$\rightarrow$	$P_2 = 17$
$\mathbf{P_3 \equiv -17 \pmod{47}}$	$\mathbf{P_3 \equiv +5 \pmod{11}}$	$\rightarrow$	$P_3 = 500$
$\mathbf{P_4 \equiv -17 \pmod{47}}$	$\mathbf{P_4 \equiv -5 \pmod{11}}$	$\rightarrow$	$P_4 = 171$

The only acceptable answer is $\mathbf{P_2 = 17}$.

23.

a. We choose $e_1 = 3$ (a primitive root of $p = 31$) and $d = 10$. Then we have $e_2 = 3^{10}$ mod 31= 25.

b. The common factor for calculation of C_2's is $e_2{}^7$ mod 31 = 25^7 mod 31 = 25.

P = "H" = 07	$C_1 = 3^7$ mod 31 = 17	$C_2 = 07 \times 25$ mod 31 = 20	$\rightarrow$	C = (17, 20)
P = "E" = 04	$C_1 = 3^7$ mod 31 = 17	$C_2 = 04 \times 25$ mod 31 = 07	$\rightarrow$	C = (17, 07)
P = "L" = 11	$C_1 = 3^7$ mod 31 = 17	$C_2 = 11 \times 25$ mod 31 = 27	$\rightarrow$	C = (17, 27)
P = "L" = 11	$C_1 = 3^7$ mod 31 = 17	$C_2 = 11 \times 25$ mod 31 = 27	$\rightarrow$	C = (17, 27)
P = "O" = 14	$C_1 = 3^7$ mod 31 = 17	$C_2 = 14 \times 25$ mod 31 = 09	$\rightarrow$	C = (17, 09)

c.

C = (17, 20)	$\rightarrow$	$P = 20 \times (17^{10})^{-1}$ mod 31 = 07	$\rightarrow$	"H"
C = (17, 07)	$\rightarrow$	$P = 07 \times (17^{10})^{-1}$ mod 31 = 04	$\rightarrow$	"E"
C = (17, 27)	$\rightarrow$	$P = 27 \times (17^{10})^{-1}$ mod 31 = 11	$\rightarrow$	"L"
C = (17, 27)	$\rightarrow$	$P = 27 \times (17^{10})^{-1}$ mod 31 = 11	$\rightarrow$	"L"
C = (17, 09)	$\rightarrow$	$P = 09 \times (17^{10})^{-1}$ mod 31 = 14	$\rightarrow$	"O"

24. In general, the numeric value of C_1 and C_2 are different in a ciphertext. If these two values are swapped during transmission, Bob cannot correctly decrypt the ciphertext to get the plaintext because

$$C_2 \times (C_1{}^d)^{-1} \neq C_1 \times (C_2{}^d)^{-1}$$

25. Although the value of p (the modulus) and d are not given, we can assume a modulus which is greater that 17 and 37 and its primitive root is 2. We have chosen p = 53 and d = 3. In this case, we have p = 53, d = 3, e_1 = 2, and $e_2 = e_1{}^3$ mod 53 = 8.

a. Alice uses r = 9 to encrypt two messages, 17 and 37. The values of ciphertexts are: $C_1 = 35$, $C_2 = 19$, $C_1' = 35$ and $C_2' = 32$.

b. Eve intercepts $C_1 = 35$, $C_2 = 19$, $C_1' = 35$ and $C_2' = 32$ and she know P = 17. Eve can use the known-plaintext attack to find P′.

$$P' = C_2' \times (e_2^r)^{-1} \bmod p = C_2' \times (C_2 \times P^{-1})^{-1} \bmod p = C_2' \times C_2^{-1} \times P \bmod p$$
$$P' = 32 \times 19^{-1} \times 17 \mod 53 = 32 \times 14 \times 17 \bmod 53 = 37$$

26.

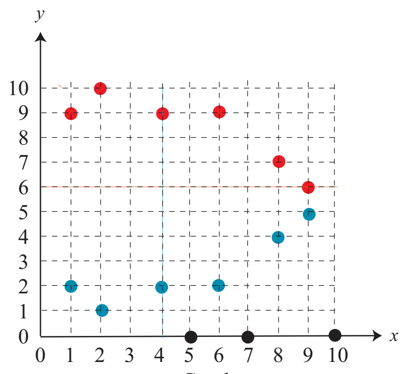
a. E(1, 2) means that $a = 1$ and $b = 2$ in the equation $y^2 = x^3 + ax + b$. The equation of the curve is then $y^2 = x^3 + x + 2$.

b. Figure S10.26 shows the points on the curve.

Figure S10.26 *Part of solution to Exercise 26*

P	–P
(1, 2)	(1, 9)
(2, 1)	(2, 10)
(4, 2)	(4, 9)
(5, 0)	**(5, 0)**
(6, 2)	(6, 9)
(7, 0)	**(7, 0)**
(8, 4)	(8, 7)
(9, 5)	(9, 6)
(10, 0)	**(10, 0)**

Points



Graph

c. Bob chooses $e_1 = (2, 1)$ and $d = 3$, then $e_2 = 3 \times (2, 1) = (4, 9)$. Public key is $E_{11}(1, 2)$, e_1, and e_2. The private key is d.

d. Assume Alice has the plaintext P = (4, 2) to send to Bob.

e. Alice choose $r = 6$ and calculate the two points of the ciphertext

$$C_1 = r \times e_1 = 6 \times (2, 1) = (8, 7)$$
$$C_2 = (4, 2) + 6 \times (4, 9) = (4, 2) + (8, 4) = (2, 10)$$

f. Bob receives C_1 and C_2. Bob calculates the plaintext as to get the P.

$$P = C_2 - (d \times C_1) = (2, 10) - 3 \times (8, 7)$$
$$= (2, 10) - (8, 4) = (2, 10) + (8, 7) = (4, 2)$$

27.

a. E(g^4, 1) means that $a = g^4$ and $b = 1$. The equation of the curve is

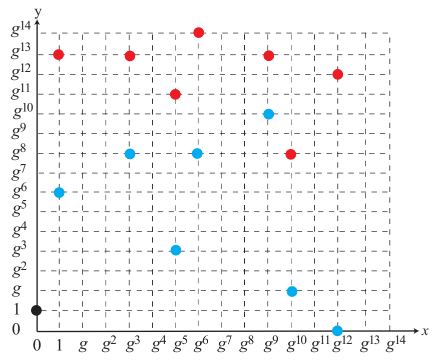
$$y^2 + xy = x^3 + g^4x^2 + 1$$

b. Assume that the irreducible polynomial is $x^4 + x + 1$ (See Appendix G). We use the process in Example 10.16 in the textbook Page 326 to find the points on the curve as shown in Figure S10.27.

Figure S10.27 *Part of the solution to Exercise 27*

P	–P
(0, 1)	(0, 1)
$(1, g^6)$	$(1, g^{13})$
(g^3, g^8)	(g^3, g^{13})
(g^5, g^3)	(g^5, g^{11})
(g^6, g^8)	(g^6, g^{14})
(g^9, g^{10})	(g^9, g^{13})
(g^{10}, g)	(g^{10}, g^8)
$(g^{12}, 0)$	(g^{12}, g^{12})

Points



Graph

c. Bob chooses $e_1 = (g^3, g^8)$ and $d = 2$. Then $e_2 = d \times e_1 = (g^5, g^3)$. The public key is the combination of e_1, e_2, and $E(g^4, 1)$. The private key is d.

d. Alice chooses $P = (g^{10}, g)$ and $r = 3$.

e. Alice calculates $C_1 = r \times e_1 = (g^9, g^{10})$ and $C_2 = P + r \times e_2 = (1, g^6)$.

f. Bob decrypt the message $P = C_2 - (d \times C_1) = (g^{10}, g) = P$

28.

a. The following shows the encrypting algorithm. It calls the **KnapsackSum** algorithm defined in the text (Algorithm 10.1).

```
KnapsackEncrypt (P, a)
{
        C = KnapSum (P, a)
        return C
}
```

b. The following shows the decrypting algorithm. It calls the **inv_KnapsackSum** algorithm defined in the text (Algorithm 10.1).

```
KnapsackDecrypt (C, b, r, n)
{
        C' = $r^{-1}$ × C mod n
        P' = inv_KnapsackSum (C', b)
        P = permute (P')
        return C
}
```

29.

a. The following shows the encrypting algorithm. Alice uses the original message (*m*), a random number (*r*) and two functions (G and H).

```
RSA_OAEP_Encrypt ($m, e, n, r$)
{
        $M \leftarrow$ pad ($m$)
        $P_1 \leftarrow M \oplus G(r)$
        $P_2 \leftarrow H(P_1) \oplus r$
        $C \leftarrow$ Fast_Exponentiation ($P_1 | P_2, e, n$)
        return C
}
```

b. The following shows the decrypting algorithm. Bob uses the ciphertext (C), the private key (*d*), modulus (*n*), and two functions G and H.

```
RSA_OAEP_Decrypt (C, $d, n$)
{
        $P \leftarrow$ Fast_Exponentiation ($C, d, n$)
        P1, $P_2 \leftarrow \text{Split}_{m,k}$ (P)
        $r \leftarrow H(P_1) \oplus P_2$
        $M \leftarrow P_1 \oplus G(r)$
        $m \leftarrow$ Extract (M)
        return $m$
}
```

30.

```
CyclingAttack (C, $e, n$)
{
        $T_1 \leftarrow C$
        $T_2 \leftarrow T_1^e \bmod n$
        while ($T_2 \neq C$)
        {
                $T_1 \leftarrow T_2$
                $T_2 \leftarrow T_1^e \bmod n$
        }
        return $T_1$
}
```

31. The following shows the AddPoint algorithm. P(x) and P(y) are x and y components of P. Calculations are done in GF(p) .

```
AddPoint ($p$, $a$, $b$, $P_1$, $P_2$)
{
        if ($P_1(x) \neq P_2(x)$)
                $\lambda \leftarrow ( (P_2(y) - P_1(y) ) / (P_2(x) - P_1(x) ) )$ mod $p$
        else
                $\lambda \leftarrow ( (3 \times (P_1(x))^2 + a ) / (2 \times P_1(y) ) )$ mod $p$
        $P_3(x) \leftarrow (\lambda^2 - P_1(x) - P_2(x))$ mod $p$
        $P_3(y) \leftarrow (\lambda \times (P_1(x) - P_3(x)) - P_1(y))$ mod $p$
        return $P_3$
}
```

32. The following shows the AddPoint algorithm. P(x) and P(y) are x and y components of P. Calculation are done in GF(2^n) with the selected irreducible polynomial.

```
AddPoint ($n$, $a$, $b$, $P_1$, $P_2$)
{
        if ($P_1(x) \neq P_2(x)$)
        {
                $\lambda \leftarrow (P_2(y) + P_1(y) ) / (P_2(x) + P_1(x) )$
                $P_3(x) \leftarrow \lambda^2 + \lambda + P_1(x) + P_2(x) + a$
                $P_3(y) \leftarrow \lambda (P_1(x) + P_3(x)) + P_3(x) + P_1(y)$
        }
        else
        {
                $\lambda \leftarrow P_1(x) + P_1(y) / P_1(x)$
                $P_3(x) \leftarrow \lambda^2 + \lambda + a$
                $P_3(y) \leftarrow (P_1(x))^2 + (\lambda + 1) P_3(x)$
        }
        return $P_3$
}
```

CHAPTER 11

Message Integrity and Message Authentication

(Solution to Practice Set)

Review Questions

1. Message integrity guarantees that the message has not been changed. Message authentication guarantees that the sender of the message is authentic.
2. The first criteria is *preimage resistant*, which ensures that Eve cannot find any message whose hash is the same as the one intercepted.
3. The *second preimage resistance* ensures that a message cannot easily be forged. In other words, given a specific message and its digest, it is impossible to create another message with the same digest.
4. The third criteria is *collision resistance,* which ensures that Eve cannot find two messages that hash to the same digest.
5. The Random Oracle Model is an ideal mathematical model for a hash function. A function based on this model behaves completely randomly. The digest can be thought of as a random variable uniformly distributed between 0 and $N - 1$ in which $N = 2^n$. Attacks on hash functions can be analyzed working with this random variable.
6. The *pigeonhole principle* says that if n pigeonholes are occupied by $n + 1$ pigeons, then at least one pigeonhole is occupied by two pigeons. Because the digest is shorter than the message, according to the pigeonhole principle there can be collisions. In other words, there are some digests that correspond to more than one message; the relationship between the possible messages and possible digests is many-to-one.
7. The following briefly states the four birthday problems:
   - ❑ **Problem 1:** What is the minimum number, k, of students in a classroom such that it is *likely* that at least one student has a predefined birthday?
   - ❑ **Problem 2:** What is the minimum number, k, of students in a classroom such that it is *likely* that at least one student has the same birthday as the student selected by the professor? This problem can be generalized as follows.

- **Problem 3:** What is the minimum number, k, of students in a classroom such that it is *likely* that at least two students have the same birthday?
- **Problem 4:** We have two classes, each with k students. What is the minimum value of k so that it is *likely* that at least one student from the first classroom has the same birthday as a student from the second classroom?

8. The following table shows the association:

Birthday Problem	***Attack on hash function***
1	*Preimage*
2	*Second preimage*
3	*Collision*
4	*Alternate collision*

9. A *modification detection code* (*MDC*) is a message digest that can prove the integrity of the message. A *message authentication code* (*MAC*) ensures the integrity of the message and the data origin authentication. The difference between an MDC and a MAC is that the second includes a secret between Alice and Bob.

10. HMAC is a nested MAC that uses two instances of a hash function. CMAC is a MAC that uses a symmetric-key cipher in the CBC mode.

Exercises

11. Imagine Alice uses the oracle. She give her message M to send to the oracle and obtain the digest d. Alice then sends the message and digest to Bob. To verify that the message has come from Alice, Bob needs to give the received message M to the same oracle and obtain the digest d_1. If d_1 is not equal (all the time with d), Bob cannot verify that the message has been sent by Alice. This means that the oracle needs to record the message obtained from Alice and the digest given to her to be able to create the same digest when Bob ask for calculation of the digest.

12. The key which is concatenated with the message in a MAC should be a secret between Alice and Bob; no one else should know this secret. When Alice sends a MAC to Bob, the key cannot be the public key of Bob or the private key of Alice for the following reasons:

a. If the key is the public key of Bob, every one (including Eve) knows this key and can use it to create a MAC to pretend that she is Alice.

b. If the key is the private key of Alice, no one (including Bob) knows the key. So Bob cannot verify that a MAC has truly created by Alice.

13. This the first birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 30$". We have

$$k \approx 0.69 \times N \approx 21$$

14. This the third birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 30$". We have

$$k \approx 1.18 \times N^{1/2} \approx 7$$

15. This the first birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 2007 - 1950 = 57$". We have

$$k \approx 0.69 \times N \approx 40.$$

16. This the third birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 2007 - 1950 = 57$". We have

$$k \approx 1.18 \times N^{1/2} \approx 9.$$

17. We solve each case separately.

a. This the third birthday problem with N = 365 and $k = 6$. We let the probability of two family member having the same birthday be P and no family member having the same birthday be Q. According to Table 11.3 in the text, we have

$$\mathrm{P} \approx 1 - e^{-k(k-1)/2N} \approx 0.04 \qquad \mathrm{Q} = 1 - \mathrm{P} \approx 0.96$$

b. This the third birthday problem with N = 30 and $k = 6$. We let the probability of two family member born on the same day of the month be P and no family member born on the same day of the month be Q. According to Table 11.3 in the text, we have

$$\mathrm{P} \approx 1 - e^{-k(k-1)/2N} \approx 0.04 \qquad \mathrm{Q} = 1 - \mathrm{P} \approx 0.96$$

c. This the first birthday problem with N = 30 and $k = 6$. We can solve this problem directly or using the approximation given in Table 11.3 in the text. In the direct solution, the probability of one particular member is born on the first day of the month is 1/30; the probability that any member of the family is born at the first day of the month is 6/30.

$$\mathrm{P} = 6 \times (1/30) = 0.20 \qquad \mathrm{P} \approx 1 - e^{-k/N} \approx 0.19$$

d. This the same as the third birthday problem because "three" can be thought of "at least two". Using the approximation in the third birthday problem we can have

$$\mathrm{P} \approx 1 - e^{-k(k-1)/2N} \approx 0.35$$

18. This is similar to the fourth birthday problem, but the size of the two classes are different. Using the same strategy we used in Appendix E, we can find that the probability that no student in the first class has the same birthday as a student in the second class is $Q = (1 - 1/N)^{k \times l}$. The probability of collision is then $P = 1 - Q = 1 - (1 - 1/N)^{k \times l}$.

19. This the third birthday problem with N = 9999 and k = 100. We let the probability of two students having the same four digits in their social security number be P. According to Table 11.3 in the text, we have

$$P \approx 1 - e^{-k(k-1)/2N} \approx 0.74$$

20. This is the pigeonhole problem. The grades (A, B, C, D, or F) are similar to the pigeonholes; the students that obtain one of these grades are playing the roles of pigeons. So we have $kn + 1 = 100$ pigeons and $n = 5$ pigeonholes. If we solve for k, we found $k = 99 / 5 > 19$. So at least 20 pigeons needs to go be in one pigeonhole, which means at least 20 students need to have one of the grades.

21. The pigeons can be distributed in the pigeonholes in any order. The principle does not define any order.

22. The probability of failure, Q, with a list of digest $k = 0.69 \times 2^n$, is 50 percent. This means that if Eve needs to be sure that she is successful, she needs to repeat the algorithm m times so that $Q = (1/2)^m = 0$. Solving this equation, gives us m = infinity, which means that Eve can never be sure that she can find the answer. However, if Eve repeats the algorithm many time, the probability of failure reduces tremendously. For example, if she repeat the algorithm 1000 times, $Q = 1 / 2^{1024}$, which is a very small number.

23. The probability of failure, Q, with a list of digest $k = 1.18 \times 2^{n/2}$, is 50 percent. This means that if Eve needs to be sure that she is successful, she needs to repeat the algorithm m times so that $Q = (1/2)^m = 0$. Solving this equation, gives us m = infinity, which means that Eve can never be hundred present sure that she can find the answer. However, if Eve repeats the algorithm many time, the probability of failure reduces tremendously. For example, if she repeat the algorithm 1000 times, $Q = 1 / 2^{1024}$, which is a very small number.

24. The following shows the value of the digest after hashing each character of the text. The method is not secure because the digest is always between 0 and 25. The total number of possible digest is N = 26.

Initial value of the digest:	d = 00
After hashing the first character (H: 07):	d = (00 + 07) mod 26 = 07
After hashing the first character (E: 04):	d = (07 + 04) mod 26 = 11
After hashing the first character (L: 11):	d = (11 + 11) mod 26 = 22
After hashing the first character (L: 11):	d = (22 + 11) mod 26 = 07
After hashing the first character (O: 14):	d = (07 + 14) mod 26 = 21

25. Although the problem does not mention the substitution process, we substitute the number x with the $(x + 1)$th prime number (the first prime number is 2). The following shows how the digest is calculated (using the table of prime numbers in Appendix H. The method is very insecure because the possible number of digests N = 100.

Initial value of the digest:			d = 00
Character: H (07)	$\rightarrow$	8th prime: 19	d = (00 + 19) mod 100 = 19
Character: E (04)	$\rightarrow$	5th prime: 11	d = (19 + 11) mod 100 = 30
Character: H (11)	$\rightarrow$	12th prime: 37	d = (30 + 37) mod 100 = 67
Character: H (11)	$\rightarrow$	12th prime: 37	d = (67 + 37) mod 100 = 04
Character: H (14)	$\rightarrow$	15th prime: 47	d = (04 + 47) mod 100 = 51

26. We use the 8-bit ASCII representation of the text (with the leftmost bit set to 0). We choose $H_0 = (11110000)_2 = 240$ and the $p = 137$. We choose $m = 8$, so we do not need padding. We have N = 5 blocks. We can then calculate the digests as follows:

H_0	$\rightarrow$	**240**				
H_1	$\rightarrow$	$(H_0 + \text{"H"}) \bmod p$	=	(240 + 72) mod 137	=	**38**
H_2	$\rightarrow$	$(H_1 + \text{"H"}) \bmod p$	=	(38 + 69) mod 137	=	**107**
H_3	$\rightarrow$	$(H_2 + \text{"H"}) \bmod p$	=	(107 + 76) mod 137	=	**46**
H_4	$\rightarrow$	$(H_3 + \text{"H"}) \bmod p$	=	(46 + 76) mod 137	=	**122**
H_5	$\rightarrow$	$(H_4 + \text{"H"}) \bmod p$	=	(122 + 79) mod 137	=	**67**

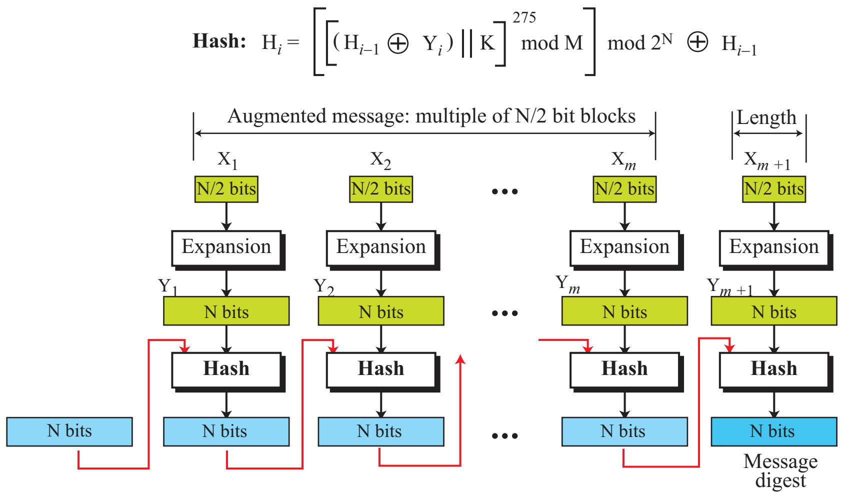
The digest is then $H_5 = 67 = (43)_{16} = (01000011)_2$. Even if we let m to be large, still the algorithm is insecure because the algorithm is trivial. A cryptographical hash algorithm needs to be more complex, with many rounds, to throughly mix the bits of the message (See Chapter 12).

27. Figure S11.27 shows the diagram for this hash algorithm. We assume that the message is already augmented and the last block (length block) is already added. We have $m + 1$ blocks, each of N/2 bits. Note that << means shift left and >> means shift right. The || operator means the OR operation.

```
MASH ($x[0 \ldots m + 1]$, K, $H_{init}$, $p$, $q$)
{
        $H[0] \leftarrow H_{init}$
        $M \leftarrow p \times q$
        for ($i = 1$ to $m + 1$)
        {
                if ($i < m + 1$)
                        $Y[i] \leftarrow$ Expand ($x[i]$, $(1111)_2$)
                else
```

```
                    $Y[i] \leftarrow$ **Expand** $(x[i], (1010)_2)$
                    $T[i] \leftarrow ((H[i-1] \oplus Y[i]) \parallel K)^{275} \bmod M$
                    $G[i] \leftarrow T[i] \bmod 2^N$
                    $H[i] \leftarrow H[i-1] \oplus G[i]$
          }
          return $H[m+1]$
}
Expand $(x, const)$
{
          $u \leftarrow (const << 4)$                        // u is const with four 0's
          $y \leftarrow (000\ldots 0)$                        // y is made of N zeros
          bytecount $\leftarrow (N/2)/4$
          for $(i = 1$ to bytecount$)$
          {
                    $t \leftarrow x \parallel (F)_{16}$               // extract the four rightmost bits
                    $u \leftarrow u \parallel t$
                    $y \leftarrow y \parallel u << (i-1) \times 8$
                    $x \leftarrow x >> 4$
          }
          return $y$
}
```

Figure S11.27 *Diagram for Exercise 27*



Hash: $H_i = \left[\left[(H_{i-1} \oplus Y_i) \parallel K\right]^{275} \bmod M\right] \bmod 2^N \oplus H_{i-1}$
Augmented message: multiple of N/2 bit blocks
Length
X_1
X_2
X_m
X_{m+1}
N/2 bits
Expansion
Y_1
Y_2
Y_m
Y_{m+1}
N bits
Hash
Message digest

28. Given the value of P (less than 1), the following algorithm finds the value of *k* for the first general birthday problem:

```
**BirthdayFirst** (P, $N$)
{
        $k \leftarrow 1$
        while ($true$)
        {
                $Q \leftarrow (1 - 1 / N)^{k}$
                if ((1 – Q) = P)
                        return $k$
                $k \leftarrow k + 1$
        }
}
```

29. Given the value of P (less than 1), the following algorithm finds the value of *k* for the second general birthday problem:

```
BirthdaySecond (P, $N$)
{
        $k \leftarrow 1$
        while ($true$)
        {
                $Q \leftarrow (1 - 1 / N)^{k-1}$
                if ((1 – Q) = P)
                        return $k$
                $k \leftarrow k + 1$
        }
}
```

30. Given the value of P (less than 1) and *N*, the following algorithm finds the value of *k* for the third general birthday problem:

```
BirthdayThrid (P, $N$)
{
        $Q \leftarrow 1$
        $k \leftarrow 1$
        while ($true$)
        {
                $Q \leftarrow Q \times (1 - (k - 1) / N)$
                if ((1 – Q) = P)
                        return $k$
                $k \leftarrow k + 1$
        }
}
```

31. Given the value of P (less than 1) and N, the following algorithm finds the value of k for the fourth general birthday problem:

```
BirthdayFourth(P, N)
{
        $k \leftarrow 1$
        while (true)
        {
                $Q \leftarrow (1 - 1 / N)^{k \times k}$
                if ((1 – Q) = P)
                        return k
                $k \leftarrow k + 1$
        }
}
```

32. Given the value of b, n, M (message), and K (key), the following algorithm finds the hash HMAC of M.

```
HMAC (b, n, M, K)
{
        ipad $\leftarrow$ Concatenate (b/8, $36_{16}$)
        opad $\leftarrow$ Concatenate (b/8, $5C_{16}$)
        K $\leftarrow$ PadZero (b, K)
        $K_1 \leftarrow K \oplus$ ipad
        $h_{middle} \leftarrow$ hash ($K_1$ | M)
        $h_{middle} \leftarrow$ Padzero (b, h)
        $K_2 \leftarrow K \oplus$ opad
        h $\leftarrow$ hash ($K_2$ | $h_{middle}$)
        return h
}
PadZero (b, T)
{
        while (length (T) < b)
                T $\leftarrow$ 0 | T
        return Res
}
Concatenate (m, C)
{
        Res $\leftarrow$ C
        $i \leftarrow 1$
        while ($i < m$)
        {
                Res $\leftarrow$ Res | C
                $i \leftarrow i + 1$
        }
        return Res
}
```

33. Given the value of n, N, k, K (key), and a message of N blocks, the following algorithm calculates the CMAC. Note that we have not shown how to calculate the value of k, but it is not difficult to write another algorithm to do so.

```
CMAC ($n, N, k$, K, M[1 … $N$])
{
        $C[1] \leftarrow E_K(M[1])$
        $i \leftarrow 2$
        while ($i < N$)
        {
                $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
                $i \leftarrow i + 1$
        }
        $C[N] \leftarrow E_K(C[N-1] \oplus M[N] \oplus k)$
        h ← SelectLeft ($n$, C[$N$])
        return h
}
```

CHAPTER 12

Cryptographic Hash Functions

(Solution to Practice Set)

Review Questions

1. A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length.

2. An iterated cryptographic hash function uses a compression function with fixed-size input and repeat the function a necessary number of times until the whole message is hashed.

3. The Merkle-Damgard scheme is an iterated hash function that is collision resistant if the compression function is collision resistant. If we use this scheme, we need only to make the compression function collision resistant.

4. A category of cryptographic hash functions uses compression functions that are made from scratch. We can mention several groups of hash functions in this category: MD's (MD2, MD4, and MD5), SHA (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512), RIPMED, and HAVAL.

5. An iterated cryptographic hash function can use a symmetric-key block cipher as a compression function. We mentioned Rabin, Davies-Meyer, Matyas-Meyer-Oseas, and Miyaguchi-Preneel schemes.

6. SHA-512 is an iterated cryptographic hash function based on Merkle-Damgard scheme in which the compression function is made from scratch. The following table shows some characteristics of SHA-512.

Characteristics	***Values***
Minimum message size	$2^{128} - 1$ bits
Block size	1024 bits
Message digest size	512 bits
Number of rounds	80
Word size	64 bits

7. Whirlpool is an iterated cryptographic hash function, based on the Miyaguchi-Preneel scheme, that uses a symmetric-key block cipher in place of the compression function. The block cipher is a modified AES cipher that has been tailored for this purpose. The following table shows some characteristics of Whirlpool.

Characteristics	***Values***
Minimum message size	$2 <^{256}$ bits
Block size	512 bits
Message digest size	512 bits
Number of rounds	10

8. SHA-512 is a hash function made of scratch; Whirlpool is a hash function made of a encryption cipher. The following table shows a comparison between the two:

Characteristics	***SHA-512***	***Whirlpool***
Scheme	Merkle-Damgard	Miyaguchi-Preneel
Minimum message size	$2^{128} - 1$ bits	$2 <^{256}$ bits
Block size	1024 bits	512 bits
Message digest size	512 bits	512 bits
Number of rounds	80	10

Exercises

9. The size of the length field is 128 bit or 32 hexadecimal digits.

a.	0000	0000	0000	0000	0000	0000	0000	03**EB**
b.	0000	0000	0000	0000	0000	0000	0000	**2710**
c.	0000	0000	0000	0000	0000	0000	000**F**	**4240**

10. The size of the length field is 256 bit or 64 hexadecimal digit. We show only part of each value.

a.	0000	(52 extra hexadecimal 0-digits)	0000	03**EB**
b.	0000	(52 extra hexadecimal 0-digits)	0000	**2710**
c.	0000	(52 extra hexadecimal 0-digits)	000**F**	**4240**

11. We need to have |M| + |P| + 128) mod 1024 = 0 or |P| = (−|M| − 128) mod 1024.

a. |P| = (− |M| − 128) mod 1024 = (−5120 − 128) mod 1024 = 896

b. |P| = (−||M| − 128) mod 1024 = (−5121 − 128) mod 1024 = 895

c. |P| = (−||M| − 128) mod 1024 = (−6143 − 128) mod 1024 = 897

12. We need to have $|M| + |P| = (2k + 1) \times 256$ in which $k \geq 0$. In other words, we need to find $|P| = (2k + 1) \times 256 - |M|$.

a.	$\lvert P\rvert = (2k+1) \times 256 - 5120$	$\rightarrow$	$k = 10$	$\rightarrow$	$P = 256$
b.	$\lvert P\rvert = (2k+1) \times 256 - 5121$	$\rightarrow$	$k = 10$	$\rightarrow$	$P = 255$
c.	$\lvert P\rvert = (2k+1) \times 256 - 6143$	$\rightarrow$	$k = 12$	$\rightarrow$	$P = 257$

13.

a. In SHA-512, the last block, which is 1024 bits, consists of

X | padding | 128-bit message length

In which, X is the rightmost part of the message of (|M| mod 1024) bits. If two messages are the same, then X is the same, the padding section is the same, the message length value is the same. This means the last block is the same.

b. In Whirlpool, the last block, which is 512 bits, consists of

X | padding | 256-bit message length

In which, X is the rightmost part of the message of (|M| mod 256) bits. If two messages are the same, then X is the same, the padding section is the same, the message length value is the same. This means the last block is the same.

14. We have $(17)^{1/2} = 4.12310562562$. If we convert this number to binary, and keep only the first 64 bits in the fraction part, we have

$100.$ **0001 1111 1000** $\ldots$ **0110 10100** $\rightarrow (4.1F83D9ABFB41BD6B)_{16}$

The fractional part is $G_0 = (1F83D9ABFB41BD6B)_{16}$.

15. The compression function of SHA-512 can be compared to a Feistel cipher (or encryption cipher) of 80 rounds:

a. The initial digest in the compression function can be thought as the plaintext in the Feistel cipher.

b. The final digest in the compression function can be thought as the ciphertext in the Feistel cipher.

c. Each word in the compression function can be thought as the corresponding round key in the Feistel cipher.

16. We can say that there is a similarity between SHA-512 with Davies-Meyer scheme if we let

a. the input digest to each compression function to be thought of as the plaintext to the imaginary cipher.

b. the output digest from each compression function to be thought of the ciphertext from the imaginary cipher.

c. the message block to be thought of as the cipher key to each cipher.

d. the *final adding operation* to be thought of as the XOR operation.

17. If the *final adding* operation is removed from the SHA-512 compression function, then its structure is similar to Rabin scheme, which is subject to the meet-in-the-middle attack as discussed in the textbook.

18. The following table shows the comparison between two ciphers;

Characteristic	*AES* (*with 10 rounds*)	*Whirlpool*
Block size	128	512 bits
State	4×4 bytes	8×8 bytes
Cipher key size	128	512 bits
Number of rounds	10 plus a pre-round	10 plus a pre-round
Key expansion	A complex process	Using the cipher itself
Round key	$(10 + 1) \times 128$ bit	$(10 + 1) \times 512$ bit
Substitution	SubBytes	SubBytes
Permutation	ShiftRows	ShiftColumns
Mixing	MixColumns	MixRows
Adding round key	Column-wise	Byte-wise

19. In AES, the need for removing the third operation is to make the encryption and decryption inverse of each other. In Whirlpool, we use a cryptographic cipher to simulate a hash function. The cipher is used only as an encryption algorithm without the decryption algorithm. The hash function can have any structure without worrying about the inverse structure.

20. $\text{RotR}_{12}(x)$ means the right rotation of the argument by 12 bits or 3 hexadecimal digits.

Before Rotation:	**1234 5678 ABCD 2345 3456 5678 ABCD 2468**
After Rotation:	**4681 2345 678A BCD2 3453 4565 678A BCD2**

21. $\text{ShL}_{12}(x)$ means the left shifting of the argument by 12 bits or 3 hexadecimal digit.

Before Shifting:	**1234 5678 ABCD 2345 3456 5678 ABCD 2468**
After Shifting:	**4567 8ABC D234 5345 6567 8ABC D246 8000**

22. $\text{Rotate}(x) = \text{RotR}_{28}(x) \oplus \text{RotR}_{34}(x) \oplus \text{RotR}_{39}(x)$. We first rotate the block three times and then apply the exclusive-or operation.

$\text{RotR}_{28}(x)$:	`BCD2 4681 2345 678A BCD2 3453 4565 678A`
$\text{RotR}_{34}(x)$:	`2AF3 491A 048D 159E 2AF3 48D1 4D15 959E`
$\text{RotR}_{39}(x)$:	`F157 9A48 D024 68AC F157 9A46 8A68 ACAC`
Rotate(x):	`6776 95D3 F7EC 1AB8 6776 E6C4 8218 5EB8`

23. The three blocks differ only in the first digit (leftmost digit). For the first digits, Conditional (0001, 0010, 0011) = $(0011)_2 = 2_{16}$. For the rest of the block, it means applying the Conditional function to three digits of equal values, which results in the common digit. Therefore, we have

Result: `2234 5678 ABCD 2345 3456 5678 ABCD 2468`

24. The three blocks differ only in the first digit (leftmost digit). For the first digits, Majority (0001, 0010, 0011) = $(0010)_2 = 3_{16}$. For the rest of the block, it means applying the Majority function to three digits of equal values, which results in the common digit. Therefore, we have

Result: `3234 5678 ABCD 2345 3456 5678 ABCD 2468`

25. Although this operation is available in most high-level languages, we write a routine for that. The following routine calls another routine, RotR(x), which rotates right only one bit. We assume that the word is stored in an array of 64 bits with the leftmost bit as the first element and the rightmost bit as the last element.

```
RotR (x, i)
{
        count ← 1
        while (count < i)
        {
                RotR (x)
                count ← count + 1
        }
        return x
}
RotR(x)
{
        temp ← x[64]
        j ← 63
        while (j > 1)
        {
                x[j + 1] ← x[j]
                j ← j − 1
        }
        x[1]← temp
        return x
}
```

26. Although this operation is available in most high-level languages, we write a routine for that. The following routine calls another routine, ShL(x), which shifts left only one bit. We assume that the word is stored in an array of 64 bits with the leftmost bit as the first element and the rightmost bit as the last element.

```
ShL_i (x, i)
{
        count ← 1
        while (count < i)
        {
                ShL (x)
                count ← count + 1
        }
        return x
}
ShL (x)
{
        j ← 2
        while (j ≤ 64)
        {
                x[j − 1] ← x[j]
                j ← j + 1
        }
        x[64] ← 0
        return x
}
```

27. We assume that words x, y, z are represented as arrays of 64 elements. The following routine shows how to find the result. We have use the if-else statement to show the conditional nature of the operation. The code can be shorter if we use the logical operators (AND, OR, and NOT).

```
Conditional (x, y, z)
{
        i ← 1
        while (i ≤ 64)
        {
                if (x[i] = 1)
                        result[i] ← y[i]
                else
                        result[i] ← z[i]
                i ← i + 1
        }
        return result
}
```

28. We assume that words x, y, z are represented as arrays of 64 elements. The following routine shows how to find the result.

```
Conditional (x, y, z)
{
        $i \leftarrow 1$
        while ($i \leq 64$)
        {
                result[i] $\leftarrow (x[i] \text{ AND } y[i]) \oplus (y[i] \text{ AND } z[i]) \oplus (z[i] \text{ AND } x[i])$
                $i \leftarrow i + 1$
        }
        return result
}
```

29. We call the RotR(x, i) function we used in Exercise 25.

```
Rotate (x)
{
        $x_1 \leftarrow x$    $x_2 \leftarrow x$    $x_3 \leftarrow x$
        result $\leftarrow$ RotR($x_1$, 28) $\oplus$ RotR($x_2$, 34) $\oplus$ RotR($x_3$, 39)
        return result
}
```

30. We represent the initial digests as a two-dimensional array Digests[8][64] and the first eight primes as Primes[8]. Note that, after taking the square root of the corresponding prime, we subtract $(i + 1) / 2$ from the result. This extracts the fractional part of the square root because the integral part is 1, 2, 3, or 4. The routine **Convert**$_{63}$ converts the fractional part to a 64-bit binary pattern.

```
CalcInitialDigest ()
{
        Primes [8] = {2, 3, 5, 7, 11, 13, 17, 19}
        $i \leftarrow 1$
        while ($i \leq 8$)
        {
                temp $\leftarrow$ ExtractFraction (Primes $[i]^{1/2}$)
                Digests [i] $\leftarrow$ Convert$_{64}$ (temp)
                $i \leftarrow i + 1$
        }
        return Digests
}
ExtractFraction(x)
{
        while ($x > 1.0$)
        {
                $x \leftarrow x - 1.0$
        }
        return x
}
Convert$_{64}$ ($x$)
{
        $j \leftarrow 1$
        while ($i \leq 64$)
        {
                result[$j$] $\leftarrow 0$
                $x \leftarrow x \times 2$
                if ($x \geq 1$)
                        {
                                result[i] $\leftarrow 1$
                                $x \leftarrow x - 1$

                        }
                $j \leftarrow j + 1$
        }
        return result
}
```

31. This is the same as the previous example, except that we need to use the first eighty primes and then take the cubic root of them. The routine uses the same routine, Convert64 defined in the solution to Exercise 30.

```
CalcConstants ()
{
        Primes [80] = {2, 3, ..., 401, 409}
        $i \leftarrow 1$
        while ($i \leq 80$)
        {
                $temp \leftarrow (\text{Primes }[i])^{1/3}$
                $temp \leftarrow$ ExtractFraction ($temp$)
                Constants [i] $\leftarrow$ Convert$_{64}$ ($temp$)
                $i \leftarrow i + 1$
        }
        return (Constants[1 ... 80])
}
```

32. The following shows the simple routine to create 80 words. The RotR and ShL routine have already been defined in the solution to Exercises 25 and 26.

```
**WordExpansion** (Key[0 … 15])
{
        $i \leftarrow 1$
        while ($i \leq 80$)
        {
                if ($i < 16$)
                        W[i] ← Key[i]
                else
                {
                        temp1 ← RotShift(W[$i$ −15], 1, 8, 7)
                        temp2 ← RotShift(W[$i$ −2], 19, 61, 6)
                        W[i] ← W[$i$ −16] ⊕ temp1 ⊕ W[$i$ −7] ⊕ temp2
                }
                $i \leftarrow i + 1$
        }
        return (W[0 … 79])
}
RotShift(W, $i$, $j$, $k$)
{
        temp1 ← RotR(W, $i$)
        temp2 ← RotR(W, $j$)
        temp3 ← ShL(W, $k$)
        result ← temp1 ⊕ temp2 ⊕ temp3
        return result
}
```

33.

```
CompressionFunction (H[1 … 8], W[0 …79], K[0 … 79])
{
        $i \leftarrow 1$
        while ($i \leq 8$)
        {
                Temp[$i$] ← H[$i$]
                $i \leftarrow i + 1$
        }
        $j \leftarrow 0$
        while ($j < 80$)
        {
                H[1 … 8] ← RoundFunction(H[1 … 8], W[$j$], K[$j$])
                $j \leftarrow j + 1$
        }
        $i \leftarrow 1$
        while ($i \le 8$)
        {
                H[$i$] $\leftarrow$ (H[$i$] + Temp[$i$]) mod $2^{64}$
                $i \leftarrow i + 1$
        }
        return (H[1 … 8])
}
RoundFunction (H[1 … 8], W, K)
{
        $i \leftarrow 1$
        while ($i \le 8$)
        {
                T[$i$] $\leftarrow$ H[$i$]
                $i \leftarrow i + 1$
        }
        H[2] $\leftarrow$ T[1]
        H[3] $\leftarrow$ T[2]
        H[4] $\leftarrow$ T[3]
        H[6] $\leftarrow$ T[5]
        H[7] $\leftarrow$ T[6]
        H[8] $\leftarrow$ T[7]
        Temp1 $\leftarrow$ (Majority (T[1], T[2], T[3]) + Rotate [T[1]) mod $2^{64}$
        Temp2 $\leftarrow$ (Conditional (T[5], T[6], T[7]) + Rotate [T[5] + W + K) mod $2^{64}$
        H[1] $\leftarrow$ (Temp1 + Temp2) mod $2^{64}$
        H[5] $\leftarrow$ (Temp2 + T[4]) mod $2^{64}$
        return (H[1 … 8])
}
```

34.

```
TransformBlockToState (b[0 … 63])
{
        $i \leftarrow 0$
        while ($i < 64$)
        {
                S[$i$/8][$i$ mod 8] $\leftarrow$ b[$i$]
                $i \leftarrow i + 1$
        }
        return (S[0 … 7][0 … 7])
}
```

35.

```
TransformStateToBlock (S[0 … 7][0 … 7])
{
        $i \leftarrow 0$
        $j \leftarrow 0$
        while ($i < 8$)
        {
                while ($j < 8$)
                {
                        $\mathbf{b}[i \times 8 + j] \leftarrow \mathbf{S}[i][j]$
                        $j \leftarrow j + 1$
                }
                $i \leftarrow i + 1$
        }
        return (b[0 … 63])
}
```

36.

```
SubByte (S[0 … 7][0 … 7])
{
        $i \leftarrow 0$
        $j \leftarrow 0$
        while ($i < 8$)
        {
                while ($j < 8$)
                {
                        $\mathbf{s}[i][j] \leftarrow$ ByteTrans ($\mathbf{s}[i][j]$)
                        $j \leftarrow j + 1$
                }
                $i \leftarrow i + 1$
        }
        return (S[0 … 7][0 … 7])
}
```

37.

```
ShiftColumns (S[0 … 7] [0 … 7])
{
        $c \leftarrow 1$
        while ($c \leq 7$)
        {
                shiftcolumn (S[c], c)
                $c \leftarrow c + 1$
        }
        return (S[0 … 7][0 … 7])
}
shiftcolumn (col[1 … 8], n)
{
        CopyColumn (col, temp)
        $r \leftarrow 0$
        while ($r \leq 7$)
        {
                col [(r − n) mod 8] ← temp [$r$]
                $r \leftarrow r + 1$
        }
        return col[1 … 8]
}
```

38.

```
MixRows (S[0 … 7] [0 … 7])
{
        $r \leftarrow 0$
        while ($r < 8$)
        {
                mixrow (S[r], $r$)
                $r \leftarrow r + 1$
        }
        return (S[0 … 7] [0 … 7])
}
shiftcolumn (row[0 … 7], $n$, Constant [0 … 7] [0 … 7])
{
        row ← Constant × row
        return row [0 … 7]
}
```

39.

```
AddRoundKey (S[0 … 7][0 … 7], k[0 … 7][0 … 7])
{
        $i \leftarrow 0$
        $j \leftarrow 0$
        while ($i < 8$)
        {
                while ($j < 8$)
                {
                        $\mathbf{s}[i][j] \leftarrow \mathbf{s}[i][j] \oplus \mathbf{k}[i][j]$
                        $j \leftarrow j + 1$
                }
                $i \leftarrow i + 1$
        }
        return (S[0 … 7][0 … 7])
}
```

40.

```
KeyExpansion (CipherKey, RC[1 … 10])
{
        K[0] ← CipherKey
        $x \leftarrow$ CipherKey
        $i \leftarrow 0$
        while ($i \le 10$)
        {
                $x \leftarrow$ SubBytes ($x$)
                $x \leftarrow$ ShiftColumn ($x$)
                $x \leftarrow$ MixRows ($x$)
                K[$i$] ← AddRoundKey ($x$, RC[$i$])
                $i \leftarrow i + 1$
        }
        return (K[0 … 10])
}
```

41.

```
RoundConstant ()
{
        $r \leftarrow 0$
        $i \leftarrow 0$
        $j \leftarrow 0$
        while ($r \le 10$)
        {
                while ($i < 8$)
                {
                        while ($j < 8$)
                        {
                                if ($i = 0$)
                                    RC[$r$][$i$][$j$] ← ByteTrans ($8 \times (r - 1) + j$)
                                else
                                    RC[$r$][$i$][$j$] ← 0
                                $j \leftarrow j + 1$
                        }
                        $i \leftarrow i + 1$
                }
                $r \leftarrow r + 1$
        }
        return (RC[0 … 10][0 … 7] [0 … 7])
}
```

42.

```
WhirlpoolCipher (P, CipherKey)
{
        K [0 … 10] ← KeyExpansion (CipherKey, RC[1 … 10])
        X ← AddRoundKey (P, K[0])
        $i \leftarrow 0$
        while ($i \leq 10$)
        {
                X = SubBytes (X)
                X = ShiftColumn (X)
                X = MixRows (X)
                X = AddRoundKey (X, K[$i$])
                $i \leftarrow i + 1$
        }
        C ← X
        return C
}
```

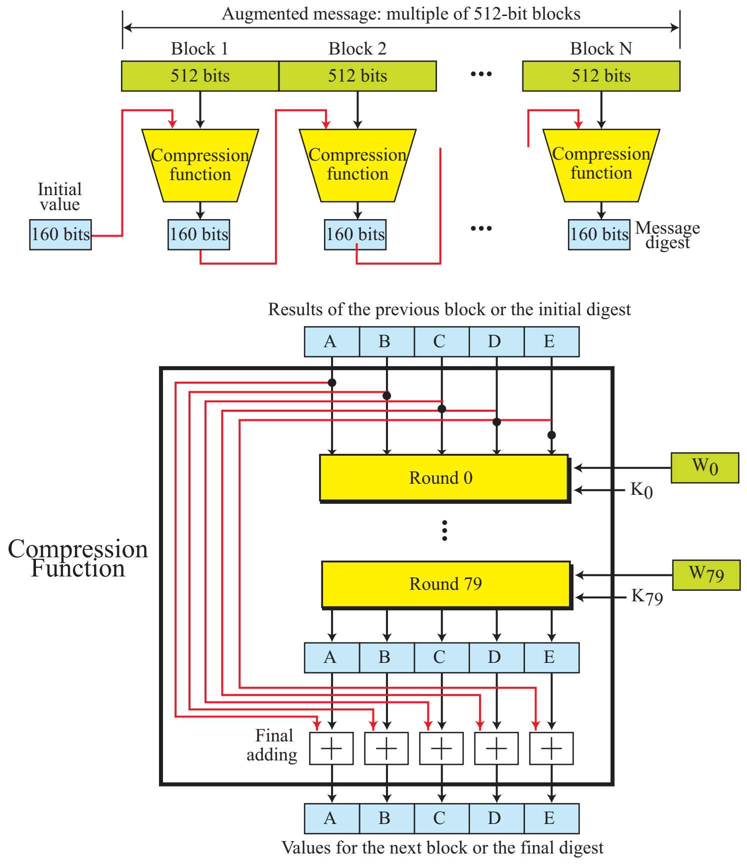
43.

```
WhirlpoolHashFunction (M[1 … N], N)
{
        H ← (00 … 0)                              // 520 of 0's
        $i \leftarrow 0$
        while ($i \leq N$)
        {
                H ← WhirlpoolCipher (M[$i$], H) ⊕ H ⊕ M[$i$]
                $i \leftarrow i + 1$
        }

        return H
}
```

44. Figure S12.44 shows the outline of SHA-1. The maximum message size in SHA is $2^{64} - 1$ (compared with ($2^{128} - 1$ in SHA-512). The block size is 512 bits and the digest size is 160 bits. SHA-1 uses only five words (each of 32 bits instead of 8 words, each of 64 bits in SHA-512). The number of rounds is 80 (same as SHA-512). The structure of round in SHA-1, however, is somehow different from SHA-512. There are 80 constants of size 32 bits. The word expansion in SHA-1 is also simpler than SHA-512; eighty 32-bit words are made from sixteen 32-bit block.

Figure S12.44 *Solution to Exercise 44.*



45. All of these three hash functions are either similar to SHA-1 or SHA-512. The differences are in the block size, digest size, word size, and the number of rounds.

a. SHA-224 is very similar to SHA-1 (See Figure S12.45a) except that the digest size is 224 bits (7 words, each of 32 bits). The number of rounds is 64.

b. SHA-256 is very similar to SHA-1 (See Figure S12.45b) except that the digest size is 256 bits (8 words, each of 32 bits). The number of rounds is 64.

Figure S12.45a *Solution to Exercise 45 part a*

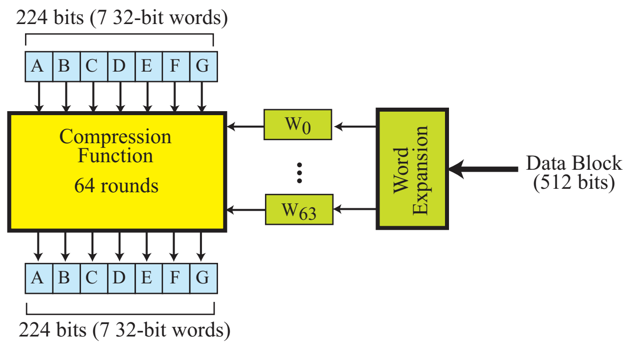
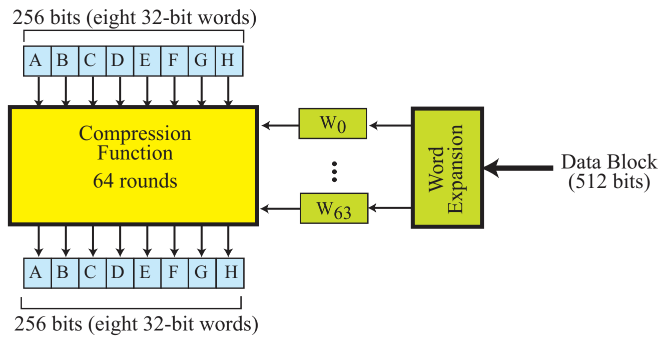
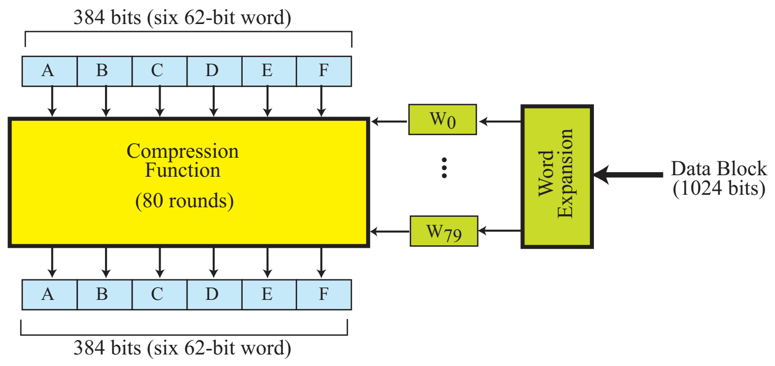


Figure S12.45b *Solution to Exercise 45 part b*



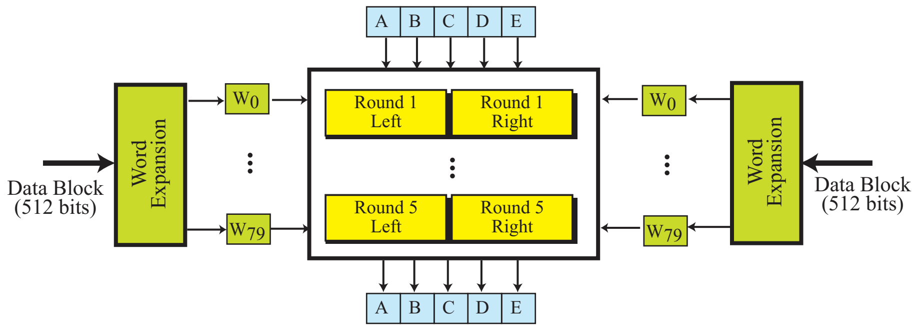
c. SHA-256 is very similar to SHA-512 (See Figure S12.45c) except that the digest size is 384 bits (6 words, each of 64 bits). The number of rounds is 80.

Figure S12.45c *Solution to Exercise 45 part c*



46. We discuss RIPEMD-160 which is similar to SHA-1. RIPEMD-160 uses blocks of 512 bits and create digest of 160 bits as SHA-1 does. The difference is the structure of the compression function and the function used. Figure S12.46 shows the general layout of RIPEMD-160. It uses 5 rounds, but each round is made of 16 iteration, which is the same as 80 rounds in SHA-1. RIPEMD, however, uses two left and round sections in each round. The block is fed to two-word expansion process that creates two sets of 80 words (each set uses a different combination of the original 16 words). The initialized digests or digests from the previous compression function are equally fed into left and right rounds.

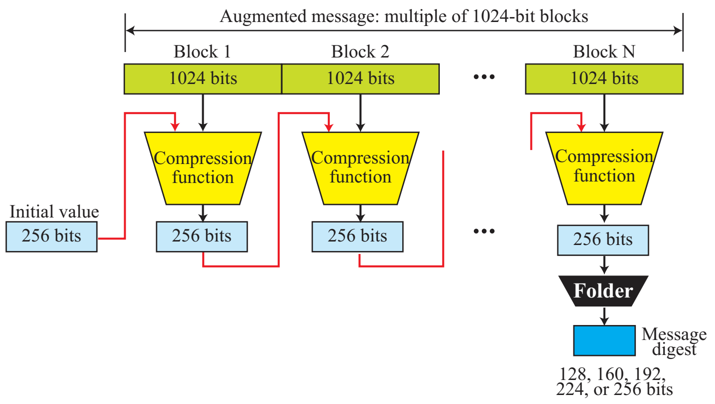
Figure S12.46 *Solution to Exercise 46*



47.

a. HAVAL is a hashing algorithm of variable-size digest designed by Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry in 1992 as shown in Figure S12.47a.

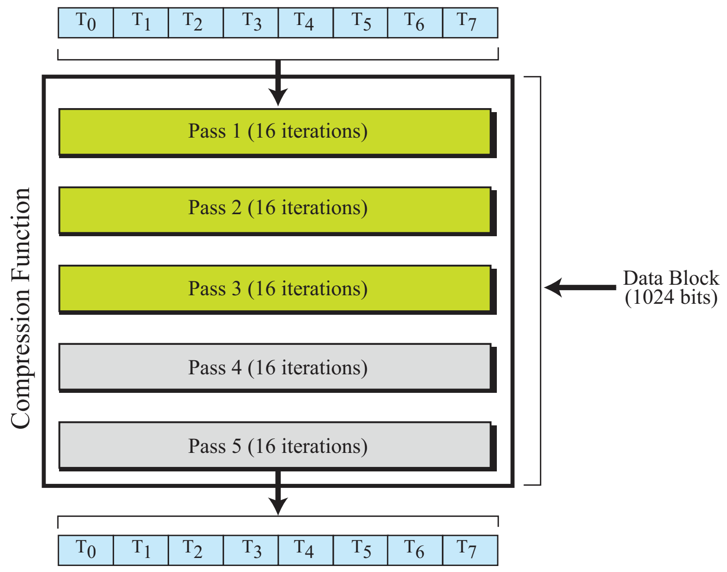
Figure S12.47a *Solution to Exercise 47 Part a*



The digest can be 128, 160, 192, 224, or 256 bits. The block size is 1024 bits. The algorithm actually creates a digest of 256 bits, but a folding algorithm matches the resulting 256 bits to one of the desired sizes.

b. The compression function uses 3, 4, and 5 passes in which each pass uses 16 iteration of different complex functions. The three-pass version is the fastest, but least secure; the five-pass version is the slowest, but the most secure. Figure S12.47b shows the structure of the compression function in HAVAL.

Figure S12.47b *Solution to Exercise 47 part b*



CHAPTER 13

Digital Signature

(Solution to Practice Set)

Review Questions

1. We mentioned four areas in which there is a differences between a conventional and a digital signature: inclusion, verification, document-signature relation, and duplicity.

   a. **Inclusion**: a conventional signature is included in the document; a digital signature is a separate document.

   b. **Verification**: A conventional signature is verified by comparing with the signature on file. The verifier of a digital signature needs to create a new signature.

   c. **Relation:** A document and a conventional signature has a one-to-many relation; a message and a digital signature has one-to-one relation.

   d. **Duplicity:** In conventional signature, a copy of the signed document can be distinguished from the original one on file. In digital signature, there is no such distinction unless there is a factor of time (such as a timestamp) on the document.

2. A digital signature can provide three security services: *message authentication*, *message integrity*, and *nonrepudiation*. It does not provide *confidentiality.*

3. The following table shows the relationship between attacks on a cryptosystem and attacks on a digital signature.

Cryptosystem attacks	***Digital signature Attacks***
Ciphertext-only	Key-only
Known-plaintext	Known-message
Chosen-plaintext	Chosen-message
Chosen-ciphertext	▨

4. In an *existential forgery*, Eve may be able to create a valid message-signature pair, but not the one that she can really use. In *selective forgery*, Eve may be able to

forge Alice's signature on a message with the content selectively chosen by Eve. Existential forgery is easy; selective forgery is difficult.

5. The idea behind the RSA digital signature scheme is the same as the RSA cryptosystem, but the roles of the private and public keys are changed. First, the private and public keys of the sender, not the receiver, are used. Second, the sender uses her own private key to sign the document; the receiver uses the sender's public key to verify it.

6. The ElGamal digital signature scheme uses the same keys as the ElGamal cryptosystem, but the algorithm is different. RSA digital signature scheme creates one signature out of the message; ElGamal digital signature scheme creates two signatures.

7. The Schnorr digital signature scheme is similar the ElGamal digital signature but the size of the signatures are smaller.

8. The Digital Signature Standard (DSS) was adopted by the National Institute of Standards and Technology (NIST) in 1994. It combines the advantages of the ElGamal scheme with some ideas from the Schnorr scheme. DSS has been criticized from the time it was published. It is less complex than Schnorr scheme, but the sizes of the signatures are smaller than ones in ElGamal scheme.

9. The elliptic curve digital signature scheme is based on DSA, but uses elliptic curves. The scheme is similar to the elliptic curve cryptosystem in which the signer and verifiers manipulate points on an elliptic curve.

10. We mentioned three variations of digital signature: *timestamped digital signatures*, *blind digital signatures*, and *undeniable digital signature*. A timestamped digital signature prevents the signature from being replayed by an adversary. A blind digit signature allows an entity to let another entity sign a document without revealing the contents of the document to the signer. An undeniable digital signature allows an entity to sign a message that cannot be denied, at the same time, cannot be forged.

Exercise

11. We have $n = 809 \times 751 = 607559$ $\phi(n) = (809 - 1) \times (751 - 1) = 606000$. Since d = 23, we have $e = d^{-1} \bmod \phi(n) = 158087$.

   a. We have

$$S_1 = M_1^d \bmod n = 100^{23} \bmod 607559 = 223388$$
$$M_1 = S_1^e \bmod n = 223388^{158087} \bmod 607559 = 100$$

   b. We have

$$S_2 = M_2^d \bmod n = 50^{23} \bmod 607559 = 5627$$
$$M_2 = S_2^e \bmod n = 5627^{158087} \bmod 607559 = 50$$

   c. If $M = M_1 \times M_2 = 5000$, we have

$S = M^d \bmod n = 5000^{23} \bmod 607559 = \mathbf{572264}$
$S = (S_1 \times S_2) \bmod n = (223388 \times 5627) \bmod 607559 = \mathbf{572264}$

12. We have $p = 881$ $d = 700$. We choose $e_1 = 3$. Then $e_2 = e_1{}^d \bmod p = 471$.

$S_1 = e_1{}^r \bmod p = 3^{17} \bmod 881 = \mathbf{540}$
$S_2 = (M - d \times S_1)\, r^{-1} \bmod (p - 1) = (400 - 700 \times 540)\, 17^{-1} \bmod 880 = \mathbf{720}$

We can verify the signature because V_1 is congruent to V_2.

$V_1 = e_1{}^M \bmod p = 3^{400} \bmod 881 = \mathbf{186}$
$V_2 = e_2{}^{S_1} \times S_1{}^{S_2} \bmod p = 471^{540} \times 540^{720} \bmod 881 = \mathbf{186}$

13. We have $q = 83$, $p = 997$, and $d = 23$. We choose $e_0 = 7$. Then $e_1 = e_0{}^{(p-1)/q} \bmod p = 9$ $e_2 = e_1{}^d \bmod p = 521$. We calculate S_1 and S_2 in mod q. We let h(40067) = 81 (The actual value does not matter here).

$S_1 = h(M \mid e_1{}^r \bmod p) = h(400 \mid 9^{11} \bmod p) = h(400 \mid 67) = h(40067) = \mathbf{81}$

$S_2 = r + ds_1 \bmod q = 11 + 23 \times 81 \bmod 83 = \mathbf{48}$

We can verify the signature assuming that h(40067) = 81.

$V = h(M \mid e_1{}^{S_2}\, e_2{}^{-S_1} \bmod p) = h(400 \mid 9^{48}\, 521^{-81} \bmod 997) =$

$V = h(400 \mid 9^{48}\, 521^{2} \bmod 997) = h(400 \mid 877 \times 257 \bmod 997) =$

$V = h(400 \mid 67) = h(40067) = \mathbf{81}$

14. We have $q = 59$, $p = 709$, $d = 14$, $r = 13$ and $h(M) = 100$. We choose $e_0 = 2$. Then we have $e_1 = e_0{}^{(p-1)/q} \bmod p = 551$ $e_2 = e_1{}^d \bmod p = 399$. The following shows the calculation for S_1, S_1 at the sender site and the calculation of V at the receiver site. Since $V = S_1$, the signature is verified.

$S_1 = (e_1{}^r \bmod p)\ q = \mathbf{48}$
$S_2 = (h(M) + d \times S_1)\, r^{-1} \bmod q = \mathbf{14}$
$S_2{}^{-1} \bmod q = 38 \qquad \rightarrow V = (e_1{}^{h(M) \times 38}\, e_2{}^{S_1 \times 38} \bmod p) \bmod q = \mathbf{48}$

15.

a. In RSA scheme $S = M^d \bmod n$. This means that the value of S can be as large as $(n - 1)$. In other words the size of $|S| \approx |n| \approx 1024$ bits.

b. In ElGamal scheme $S_1 = (\ldots) \bmod p$ and $S_2 = (\ldots) \bmod (p - 1)$. This means that the value of S_1 can be as large as $(p - 1)$ and the value of S_2 can be as large as $(p - 2)$ In other words the size of $|S_1| \approx |p| \approx 1024$ bits and the size of $|S_2| \approx |p| \approx 1024$ bits. This means the sign of the signature is 2048 bits.

c. In Schnorr scheme $S_1 = h$ (...) and $S_2 = (\ldots) \bmod (q)$. This means that the value of S_1 is exactly equals h (...) and the value of S_2 can be as large as $(q - 1)$. Since q is required to be the same size as q. The size of $|S_1| \approx |q| \approx 160$ bits and the size of $|S_2| \approx |q| \approx 160$ bits. This means the sign of the signature is 320 bits. The signature in Schnorr is much smaller than signature in ElGamal.

d. In DSS scheme $S_1 = (\ldots) \bmod q \quad S_2 = (\ldots) \bmod q$. This means that the value of S_1 and S_2 can be as large as $(q - 1)$. The size of $|S_1| = |S_2| \approx |q| \approx 160$ bits and the This means the sign of the signature is 320 bits. The signature in DSS is the same size as the signature in the Schnorr scheme.

16. If $S_2 = 0$, $S_2^{-1} \bmod q$ does not exist and the verifier cannot calculate the value of V to verify the signature.

17. In all of these schemes, Eve can calculate the value of d if she intercept a message and its signature. She can then forge a message from Alice to Bob. Each case is described separately in Exercises 23, 24, and 25.

18.

a. In the ElGamal scheme, if Alice uses the same value for r to sign two messages, the value of S_1 is the same for both signature. This is a weakness in the security of the signature.

b. In the Schnorr scheme, since the value of S_1 depends on the value of the message, the two values of S_1 are different. The weakness described in part *a* does not apply here.

c. In the DSS scheme, the situation is the same as in the ElGamal scheme. The two values of S_1 are the same for two signature, which is considered a weakness in the security of DSS.

19. If $p = 19$ and $q = 3$, $n = 57$. Eve can easily calculate $\phi(n) = \phi(57) = 36$. Since e is public, Eve can find $d = e^{-1} \bmod n$. Eve can now choose a message of her own M, calculate $S = M^d \bmod n$. Eve then sends M and S to Bob and pretends that they are coming from Alice.

20. If $p = 19$, then the value of d is between 2 and 17. Since $e_2 = e_1^d \bmod p$ and the values of e_1, e_2, and p are public, Eve can find the value of d using exhaustive search. Eve can now choose a message of her own M, calculates S_1 and S_2. Eve then sends M, S_1, and S_2 to Bob and pretends that they are coming from Alice.

21. If $p = 29$ and $q = 7$, then the value of d is between 2 and 7 (it should be less than $q - 1$). Since $e_2 = e_1^d \bmod p$ and the values of e_1, e_2, p, and q are public, Eve can find the value of d using exhaustive search. Eve can now choose a message of her own M, calculates S_1 and S_2. Eve then sends M, S_1, and S_2 to Bob and pretends that they are coming from Alice.

22. If $p = 29$ and $q = 7$, then the value of d is between 2 and 7 (it should be less than $q - 1$). Since $e_2 = e_1^d \bmod p$ and the values of e_1, e_2, p, and q are public, Eve can find the value of d using exhaustive search. Eve can now choose a message of her own M, calculates S_1 and S_2. Eve then sends M, S_1, and S_2 to Bob and pretends that they are coming from Alice.

23. In ElGamal scheme, if Eve can somehow finds out what value of r is used by Alice to calculate the signature for a particular message, the whole system is broken. Eve knows the value of M, S_1, S_2 and r. She can calculate the value of d as shown below:

$$\boldsymbol{d = (M - rS_2)S_1^{-1} \bmod (p - 1)}$$

This is possible if **gcd**$(S_1, p - 1) = 1$, which is very probable. When d is found, Eve can choose a message of her own (selective forgery), calculate the signature and send them to Bob fooling him that the message is coming from Alice.

24. In Schnorr scheme, if Eve can somehow finds out what value of r is used by Alice to calculate the signature for a particular message, the whole system is broken. Eve knows the value of M, S_1, S_2 and r. She can calculate the value of d as shown below:

$$\boldsymbol{d = (S_2 - r)S_1^{-1} \bmod q}$$

This is possible if **gcd**$(S_1, q) = 1$, which is very probable. When d is found, Eve can choose a message of her own (selective forgery), calculate the signature, and send them to Bob fooling him that the message is coming from Alice.

25. In DSS scheme, if the value of r revealed, the whole system is broken. Eve knows the value of M, S_1, S_2 and r. She can calculate the value of d as shown below:

$$\boldsymbol{d = (rS_2 - h(M))S_1^{-1} \bmod q}$$

This is possible if **gcd**$(S_1, q) = 1$, which is very probable. When d is found, Eve can choose a message of her own (selective forgery), calculate the signature and send them to Bob fooling him that the message is coming from Alice.

26.

a. Although the value of S_1 is calculated differently in each scheme, both values are in the range 0 to q if the size of the digest in Schnorr is the same as the size of the q (which is normally the case).

b. Although the value of S_2 is calculated differently in each scheme, both values are in the range 0 to q since calculation is done modulo q in both cases.

27. This is done to make the calculation possible because if $a^x \equiv a^y \bmod p$, then $x \equiv y \bmod (p - 1)$.

28. In the Schnorr scheme, we need to make both S_1 and S_2 smaller than q. Since we apply a hashing function to S_1 in which the size of the digest is q, the result is automatically smaller than q. Since no hashing is applied to S2, we need to do the calculation in modulo q to make the result smaller than q.

29. In the DSS scheme, we need to make both S_1 and S_2 smaller than q. However, to make it more difficult for Eve to find the value of r, we first do exponentiation in modulo p (which is much larger than q), but we apply another modulo operation to

reduce the size of S_1. In case of S_2, since there is no exponentiation and the size of the digest is smaller than q, we need to apply only a modulo q operation to make the size of S_2 smaller than q.

30. We start with V and show that it is congruent to S_1.

$$V = h(M \mid e_1^{S_2} e_2^{-S_1} \bmod p)$$

$$V = h(M \mid e_1^{S_2} (e_1^{d} \bmod p)^{-S_1} \bmod p) \qquad \text{// Since } e_2 = e_1^{d} \bmod p$$

$$V = h(M \mid e_1^{S_2} (e_1^{-dS_1} \bmod p) = h(M \mid e_1^{r + dS_1} e_1^{-dS_1} \bmod p)$$

$$V = h(M \mid e_1^{r} \bmod p) = S_1$$

31. We start with V and show that it is congruent to S_1. Let $h(M) = x$.

$$V = (e_1^{xS_2-1} e_2^{S_1 S_2-1} \bmod p \bmod q = (e_1^{x} \; e_2^{S_1})^{S_2-1} \bmod p \bmod q$$

$$V = (e_1^{x} \; e_1^{dS_1})^{S_2-1} \bmod p \bmod q \qquad \text{// Since } e_2 = e_1^{d} \bmod p$$

$$V = (e_1^{x + dS_1})^{S_2-1} \bmod p \bmod q$$

$$V = (e_1^{rS_2})^{S_2-1} \bmod p \bmod q \qquad \text{// Since } S_2 = (x + dS)\, r^{-1} \bmod q$$

$$V = (e_1^{r})^{S_2 S_2-1} \bmod p \bmod q = (e_1^{r}) \bmod p \bmod q = S_1$$

32. We prove that the point T(…, …) reached by Bob on the curve is the same point re_1(…, …) reached by Alice on the curve (all calculation are in mod q).

$$T(\ldots,\ldots) = Ae_1(\ldots,\ldots) + Be_2(\ldots,\ldots) = Ae_1(\ldots,\ldots) + Bde_1(\ldots,\ldots)$$

$$T(\ldots,\ldots) = (A + dB)\, e_1(\ldots,\ldots) = (h(M)S_2^{-1} + dS_2^{-1}S_1)\, e_1(\ldots,\ldots)$$

$$T(\ldots,\ldots) = S_2^{-1}\,(h(M) + dS_1)\, e_1(\ldots,\ldots) = S_2^{-1}\,(rS_2)\, e_1(\ldots,\ldots) = r\, e_1(\ldots,\ldots)$$

If T(…, …) and re_1(…, …) represent the same point on the curve, then their first coordinates is the same. This proves that $V = S_1$.

33.

```
RSA_Signing (M, d, n)
{
        S ← $M^d$ mod n
        return (M, S)
}
```

```
RSA_Verifying (M, e, n, S)
{
        M′ ← S^e mod n
        if (M’ = M)
                Accept M
        else
                Reject M
}
```

34.

```
ElGamal_Signing (M, r, e_1, d, p)
{
        S_1 ← e_1^r mod p
        S_2 ← (M − d × S_1) r^{−1} mod (p − 1)
        return (M, S_1, S_2)
}
```

```
ElGamal_Verifying (M, e_1, e_2, p, S_1, S_2)
{
        V_1 ← e_1^M mod p
        V_2 ← e_2^{S_1} × S_1^{S_2} mod p
        if (V_1 = V_2)
                Accept M
        else
                Reject M
}
```

35.

```
Schnorr_Signing (M, r, e_1, d, p, q)
{
        S_1 ← h (M | e_1^r mod p)
        S_2 ← (r + d × S_1) mod q
        return (M, S_1, S_2)
}
```

```
Schnorr_Verifying (M, e_1, e_2, p, q, S_1, S_2)
{
        V ← h (M | e_1^{S_1} × e_2^{−S_2} mod p)
        if (S_1 = V)
                Accept M
        else
```

```
                Reject M
}
```

36.

```
DSS_Signing (M, $r$, $e_1$, $d$, $p$, $q$)
{
        $S_1 \leftarrow (e_1^{\,r} \bmod p) \bmod q$
        $S_2 \leftarrow (h(M) + d \times S_1)\, r^{-1} \bmod q$
        return (M, $S_1$, $S_2$)
}
```

```
DSS_Verifying (M, $e_1$, $e_2$, $p$, $q$, $S_1$, $S_2$)
{
        $x \leftarrow h(M) \times S_1^{-1} \bmod q$
        $y \leftarrow S_2 \times S_1^{-1} \bmod q$
        $V \leftarrow (e_1^{\,x} \times e_2^{\,y} \bmod p) \bmod q$
        if ($S_1 = V$)
                Accept M
        else
                Reject M
}
```

37.

```
EllipticCurve_Signing (M, $a$, $b$, $r$, $e_1(\ldots, \ldots)$, $d$, $p$, $q$)
{
        $P(u, v) \leftarrow r \times e_1(\ldots, \ldots)$
        $S_1 \leftarrow u \bmod q$
        $S_2 \leftarrow (h(M) + d \times S_1)\, r^{-1} \bmod q$
        return (M, $S_1$, $S_2$)
}
```

```
EllipticCurve_Verifying (M, $a$, $b$, $e_1(\ldots, \ldots)$, $e_1(\ldots, \ldots)$, $p$, $q$, $S_1$, $S_2$)
{
        $A \leftarrow (h(M) \times S_2^{-1}) \bmod q$
        $B \leftarrow (S_1 \times S_2^{-1}) \bmod q$
        $T(x, y) \leftarrow A \times e_1(\ldots, \ldots) + B \times e_1(\ldots, \ldots)$
        $V \leftarrow x \bmod q$
        if ($S_1 = V$)
                Accept M
        else
                Reject M
}
```

CHAPTER 14

Entity Authentication

(Solution to Practice Set)

Review Questions

1. There are two differences between message authentication and entity authentication. First, message authentication might not happen in real time; entity authentication does. Second, message authentication simply authenticates one message; the process needs to be repeated for each new message. Entity authentication authenticates the claimant for the entire duration of a session.
2. Verification can be done with one of three kinds of witnesses: *something known, something possessed*, or *something inherent. Something know* is a secret known only by the claimant that can be checked by the verifier. *Something possessed* is something that can prove the claimant's identity. *Something inherent*. is an inherent characteristic of the claimant.
3. A fixed password is a password that is used over and over again for every access. A one-time password is a password that is used only once.
4. Long passwords has the advantage that they cannot be easily guessed by an intruder. They have the disadvantage that cannot be easily remembered by the user and needs to be recorded somewhere, which make them vulnerable to password stealing attack.
5. In challenge-response authentication, the claimant proves that she knows a secret without sending it to the verifier.
6. A nonce is a random number used only once. A nonce must be time-varying; every time it is created, it should be different.
7. In a dictionary attack, Eve is interested in finding one password, regardless of the user ID. Eve can create a list of numbers. She then applies the hash function to every number until she finds a match with a hashed password.
8. In challenge-response authentication, the claimant proves that she knows a secret without sending it to the verifier. In zero-knowledge authentication, the claimant proves that she knows a secret without revealing it.
9. Biometrics is the measurement of physiological or behavioral features that identify a person (authentication by something inherent). Biometrics measures features that

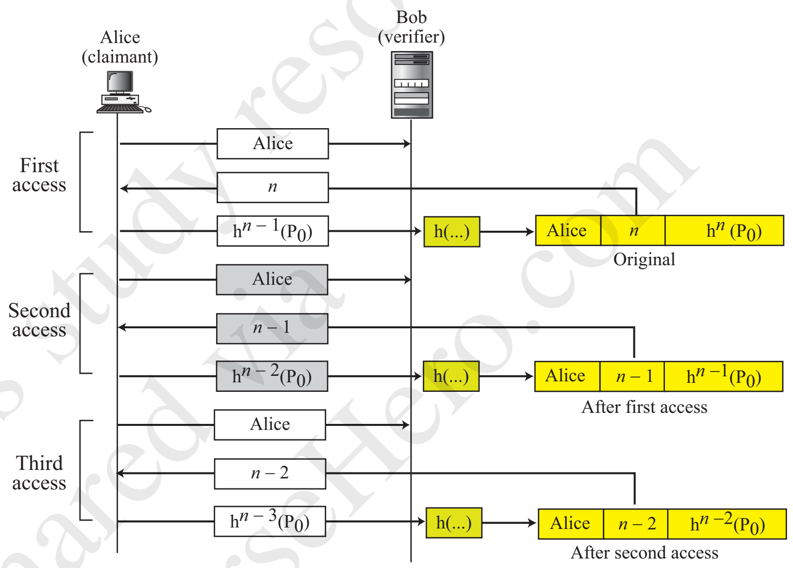
cannot be guessed, stolen, or shared. These techniques can be divided into two broad categories: *physiological* and *behavioral*.

10. Accuracy of biometric techniques is measured using two parameters: *false rejection rate (FRR)* and *false acceptance rate (FAR)*. FRR measures how often a person, who should be recognized, is not recognized by the system. FAR measures how often a person, who should not be recognized, is recognized by the system.

Exercises

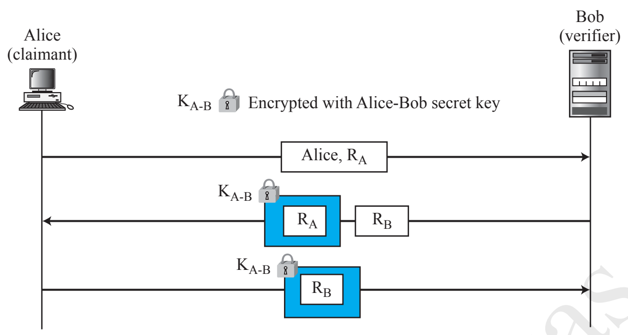
11. A system may require that users frequently change their passwords. In this case, a validity period is defined for each password. Near the end of the period, the system warns the user to create a new password and use the password-changing process to change the password.
12. Guessing attacks can be prevented if people use long passwords with no relation with their names, dates of birth, account numbers, and so on.
13. See Figure S14.13.

Figure S14.13 *Solution to Exercise 13*



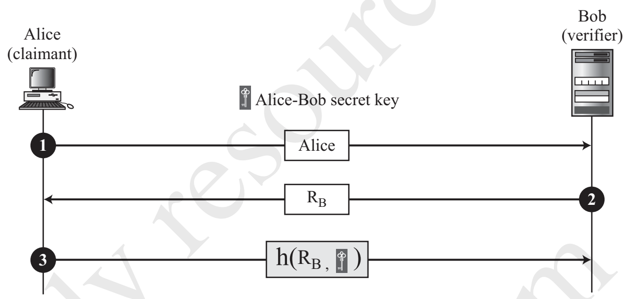
14. One problem with timestamp is the difficulty in synchronization. The computer of the claimant and the verifier needs to be synchronized for the timestamp to be effective.
15. It can be done. However, it is very inefficient. It can be done using three exchange as shown in Figure S14.15. But this efficient protocol is subject to an attack called reflection attack (See solution to Exercise 30).

Figure S14.15 *Solutions to Exercise 15*



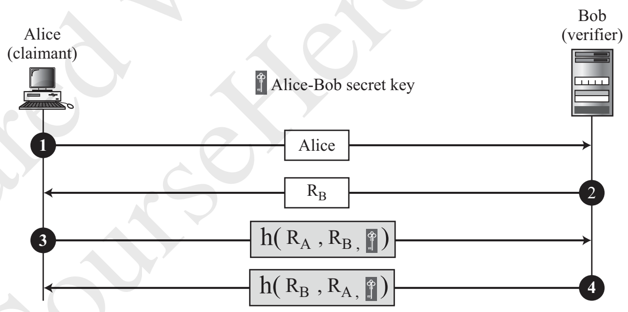
16. See Figure S14.16.

Figure S14.16 *Solution to Exercise 16*



17. See Figure S14.17.

Figure S14.17 *Solution to Exercise 17*



18. The following table shows the comparison.

Feature	*Protocol in Figure 14.5*	*Protocol in Figure 14.9*
Challenge	Bob's nonce is sent in plaintext.	Bob's nonce is encrypted with Alice's public key
Response	Alice should show that she have the secret by encrypting Bob's nonce and send the encrypted nonce.	Alice should show that she have the secret by decrypting Bob's nonce and send it in plaintext.

19. The following table shows the comparison.

Authentication	*Feature*	*Protocol in Figure 14.7*	*Protocol in Figure 14.10*
Authentication of Bob	Challenge	Bob's nonce is sent in plain-text (second exchange).	Alice challenges Bob by sending her nonce encrypted with Bob's public key (first exchange)
	Response	Alice should show that she have the secret by encrypting Bob's nonce and send the encrypted nonce (third exchange).	Bob shows that he possesses his private key by decrypting Alice's nonce and resending it in the second exchange.
Authentication of Alice	Challenge	Alice's nonce is sent in encrypted form (third exchange).	Bob challenges Alice by sending his nonce encrypted with Alice's public key (sec-ond exchange)
	Response	Bob should show that he have the secret by encrypting Alice's nonce and send the encrypted nonce (fourth exchange).	Alice shows that she pos-sesses her private key by decrypting Bob's nonce resending it in the third exchange

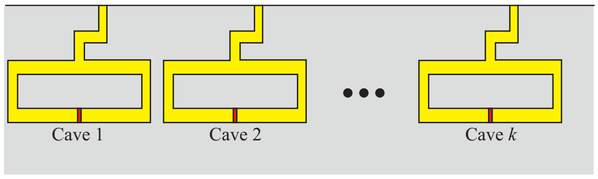
20. A timestamp can be used anywhere a nonce can be used. So we can use a timestamp T instead of R_B in Figure 14.9 (in the textbook) and two timestamps T_1 and T_2 instead of R_A and R_B in Figure 14.10 (in the textbook).

21. All three protocols use witness, challenge, and response. However, the value of these three items are different in different protocols as shown in the following table.

	Witness	*Challenge*	*response*
Figure 14.13	$x = r^2 \bmod n$	c: (0 or 1)	$y = rs^c \bmod n$
Figure 14.15	$x = r^2 \bmod n$	$(c_1, c_2, \ldots c_k)$	$y = rs_1^{c1} s_2^{c2} \ldots s_k^{ck} \bmod n$
Figure 14.16	$x = r^e \bmod n$	c: *(1 to e)*	$y = rs^c \bmod n$

22. One way to apply the cave analogy to Feige-Fiat-Shamir protocol is to have k caves. After Alice comes up from the one cave, she need to try the next one. She has passed the test if she come up from all of them. The door in each cave opens with different magic word. Figure S14.22 shows the idea.

Figure S14.22 *Solution to Exercise 22*



23. $p \leftarrow 569$ $q \leftarrow 683$ $n \leftarrow 388{,}267$ $s \leftarrow 157$ $v \leftarrow 24{,}649$

r	$x \leftarrow r^2 \bmod n$	c	$y \leftarrow rs^c \bmod n$	$y^2 \bmod n$	$xv^c \bmod n$
203,122	130663	0	203122	**130,663**	**130,663**
153,271	292,873	1	379,260	**366,513**	**366,513**
377,245	345,180	1	210,881	**247,049**	**247,049**

The values of the last two columns should be the same if Alice is honest or has pre-guessed the value of c.

24. $p \leftarrow 683$ $q \leftarrow 811$ $n \leftarrow 553{,}913$ ($s_1 \leftarrow 157$ $s_2 \leftarrow 43215$) ($v_1 \leftarrow 112068$ $v_2 \leftarrow 338402$).

r	$x \leftarrow r^2 \bmod n$	c's	$y \leftarrow rs^c \bmod n$	$y^2v^c \bmod n$	x
12,672	498,727	(0, 1)	354436	**498,727**	**498,727**
14,567	48,810	(1, 1)	491234	**48,810**	**48,810**
16,034	213386	(1, 0)	374,579	**213,386**	**213,386**

The values of the last two columns should be the same if Alice is honest or has pre-guessed the value of c's correctly.

25. $p \leftarrow 683$ $q \leftarrow 811$ $n \leftarrow 553{,}913$ $\phi(n) \leftarrow 552{,}402$ $s \leftarrow 157$ $e \leftarrow 7$ $v \leftarrow 444751$

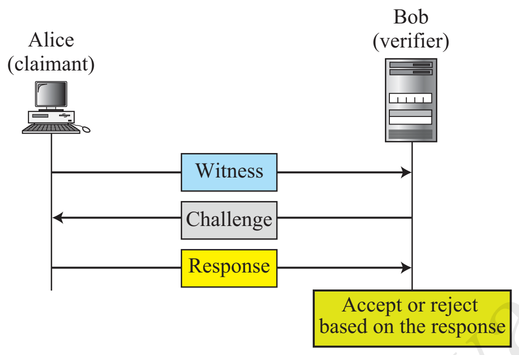
r	$x \leftarrow r^e \bmod n$	c	$y \leftarrow rs^c \bmod n$	$y^e v^c \bmod n$	x
15,024	519,635	1	153,116	**519,635**	**519,635**
7,235	135,522	3	35,444	**135,522**	**135,522**
423	200,972	4	18,109	**200,972**	**200,972**

The values of the last two columns should be the same if Alice is honest or has pre-guessed the value of c correctly.

26. All three protocols use three exchanges and some calculation. Figure S14.26 shows the general idea in these protocols.

27. In the Fiat-Shamir protocol, a dishonest claimant can correctly responds to a change with the probability of 1/2. The probability that a dishonest claimant

Figure S14.26 *Solution to Exercise 26*



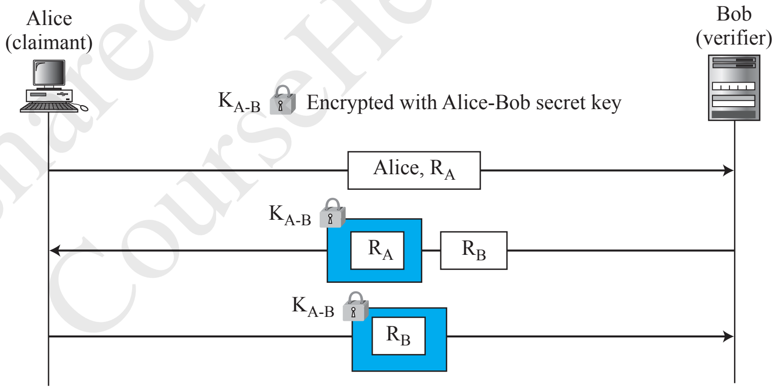
responds correctly 15 times is then P = $(1/2)^{15}$ = 1/32768 ≈ 0.0000305, which is very small.

28. In the Feige-Fiat-Shamir protocol, a dishonest claimant can correctly responds to a change with the probability of $(1/2)^k$ because there are k challenges. The probability that a dishonest claimant responds correctly 15 times is then P = $[(1/2)^k]^{15}$, which is extremely small when k is large.

29. In the Guillou-Quiquater protocol, a dishonest claimant can correctly responds to a change with the probability of $1/(e - 1)$ because the challenge value is between 1 and e. The probability that a dishonest claimant responds correctly 15 times is then P = $[1/(e - 1)]^{15}$, which is extremely small when e is large.

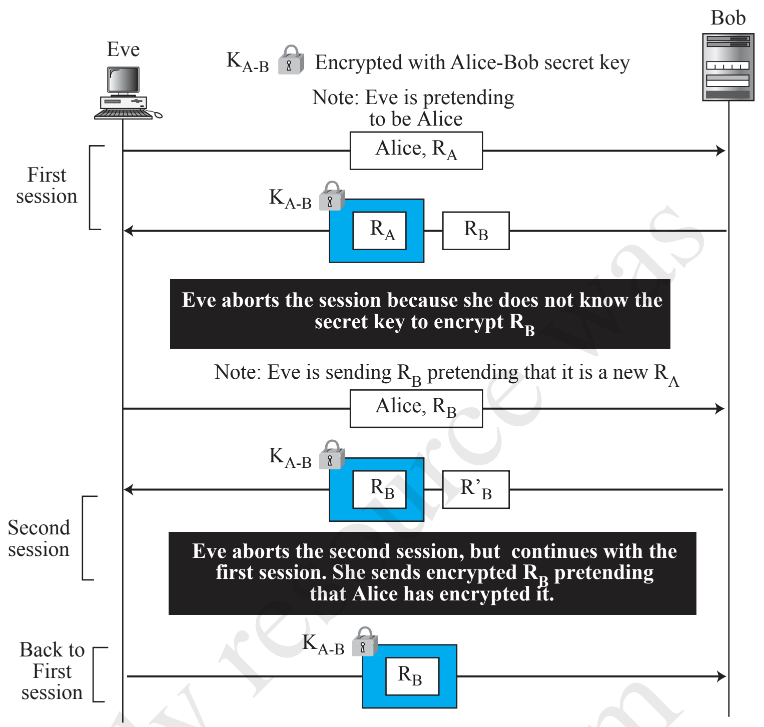
30. Reflection attack cannot happened in Figure 14-10 of the text. It may happened if bidirectional authentication is done using the secret key between Alice and Bob as shown in Figure S14.30a.

Figure S14.30a *Solution to Exercise 30*



In this version, Eve can use two sessions to fool Bob as shown in Figure S14.30b.

Figure S14.30b


Eve
Bob
$K_{A\text{-}B}$ Encrypted with Alice-Bob secret key
Note: Eve is pretending to be Alice
First session
Alice, R_A
$K_{A\text{-}B}$
R_A
R_B
Eve aborts the session because she does not know the secret key to encrypt R_B
Note: Eve is sending R_B pretending that it is a new R_A
Alice, R_B
R_B
R'_B
Second session
Eve aborts the second session, but continues with the first session. She sends encrypted R_B pretending that Alice has encrypted it.
Back to First session
R_B

In the first session, Eve pretends that she is Alice and sends R_A and Alice identity. She receives R_B from Bob, which she keeps. Eve now pretends that this session is aborted. After a while she sends R_B to bob with Alice ID as though she is starting a new session. Bob is fooled that R_B that he has sent is a new R_A from Alice (this can occur if Bob does not keep track of R's). Bob know encrypt R_B with the secret key and sends it to Eve. That is what Eve needs. She sends what she has received (she cannot decrypt it, but it does not matter) to Bob and pretends that the first session continues. Now Eve is falsely authenticated to Bob as Alice.

CHAPTER 15

Key Management

(Solution to Practice Set)

Review Questions

1. The following shows the main duties.
   - **a.** KDC establishes a shared secret key between itself and each newly joined member.
   - **b.** KDC accepts requests from members who wants to establish a session key between themselves and other members.
   - **c.** KDC checks the willingness of members for establishing session keys.
   - **d.** KDC creates a session key and sends it to two parties who want to use it.
2. A session key is a secret key that is used only once (during the session). After the session termination, the session key is no longer useful. The following shows one of the ways a session is established between Alice and Bob:
   - ❑ Alice sends a request to the KDC stating that she needs a session key.
   - ❑ KDC informs Bob about Alice's request.
   - ❑ If Bob agrees, a session key is created between the two.
3. Kerberos is a popular authentication protocol, and at the same time a KDC. Three servers are involved in the Kerberos protocol: an authentication server (AS), a ticket-granting server (TGS), and a real (data) server that provides services to others. AS is the KDC. TGS issues a ticket for the real server (Bob) and provides the session key. The real server provides services.
4. Diffie-Hellman protocol is a symmetric-key agreement protocol that allows Alice and Bob to create a session key between themselves without using a KDC.
5. Man-in-the-middle attack is an attack on the Diffie-Hellman protocol, in which Eve can fool Alice and Bob by creating two keys: one between herself and Alice, and another between herself and Bob.
6. The station-to-station protocol is a symmetric-key agreement protocol based on Diffie-Hellman. It uses digital signatures with public-key certificates to establish a session key between Alice and Bob.

7. A certification authority (CA), is a federal or state organization that binds a public key to an entity and issues a certificate.
8. The X.509 recommendation is a way to describe a certificate in a structured way using the ASN.1 protocol.
9. Public-Key Infrastructure (PKI), created by the Internet Engineering Task Force, is a model for creating, distributing, and revoking certificates based on the X.509.
10. A trust model defines the rules that specify how a user can verify a certificate received from a CA. We mentioned three variation of a trust model: hierarchical model, mesh model, and web of trust.

Exercises

11. In this case, Alice can access the ticket, change the session key, and send the altered session key to Bob. In other words, Alice can create the session key instead KDC; the role of KDC in authentication is totally deleted in this case.

12.
   a. R_A ensures that the session is totally fresh. In other words, R_A ensures Alice that the second message is from KDC, not replayed by Eve. Assume that there is no R_A. Alice starts and ends a session. However, Eve intercepts the second message and stores it in her computer. Later, when Alice starts a new session to communicate with Bob, Eve intercepts the request and discards it. Eve now replayed the old stored message. Alice is fooled that this second message is coming from KDC. Alice sends the third message to Bob. If Eve somehow knows Bob's secret key with KDC (K_B), she can intercepts the third message and decrypts it. Eve now knows the session key between Alice and Bob. She can pretend to be Bob and continue communication with Alice. The use of R_A prevents this type of replaying attack.
   b. R_B in step 4 and $R_B - 1$ in step 5 are to ensure Bob that Alice actually has K_{AB} (the session key). This nonce acts like a testing messages exchanged between Alice and Bob.

13.
   a. Alice is authenticated by KDC, because only Alice can decrypt the message sent in step 2.
   b. KDC is an authorized and well-know entity. The whole assumption is that Alice trusts KDC.
   c. KDC is an authorized and well-know entity. The whole assumption is that Bob also trusts KDC.
   d. Alice is authenticated to KDC. KDC is authenticated to Bob. Therefore, Alice is authenticated to Bob.
   e. Bob is authenticated to KDC. KDC is authenticated to Alice. Therefore, Bob is authenticated to Alice.

14. The Needham-Schroeder protocol involves the KDC from the beginning even if Bob is not ready or not willing to communicate with Bob. In the Otway-Rees protocol, on the other hand, Alice first contact Bob. If Bob is ready to communicate with Alice, then KDC is being involved.

15. There is one flaw in the Needham Schroeder protocol (discovered by Denning and Sacco), which is often referred to as *known-session-key attack*. Eve records the exchanges in a session between Alice and Bob. If is somehow successful to obtain the session key, K_{AB}, Eve now launches a new session starting with the third exchange; she resends the ticket to Bob. Bob responds by sending a new nonce, R_B. Eve can decrypt this message (she knows the session key) and obtain R_B. Eve now responds using R_B-1. A session has been created between Bob and Eve. The flaw in the protocol is that there is not a nonce that glues the five exchanges in the session. The first nonce, R_A, is active only for the first two messages; the second nonce, R_B, is active only for the last two messages. Eve can partially replay the second part of the these messages. In Otway-Rees protocol, a third nonce, R, is used to be active during all four exchanges. Eve cannot replay only part of the message.

16. The timestamp in Kerberos actually serves as nonce R in Otway-Rees protocol. Although we have not shown them in Figure 15.8 of the textbook, Kerberos can use R_A and R_B in the protocol (Version five actually uses these nonces).

17.

a. $K = g^{xy} \bmod p = 7^{3 \times 5} \bmod 23 = \mathbf{14}$

b. $R_1 = g^x \bmod p = 7^3 \bmod 23 = 21$ $R_2 = g^y \bmod p = 7^5 \bmod 23 = 17$. Note that $K = R_2^{\,x} \bmod 23 = 17^3 \bmod 23 = R_1^{\,y} \bmod 23 = 21^5 \bmod 23 = \mathbf{14}$.

18. Assume that both Alice and Bob choose x (instead of x and y). Then we have

$R_1 = g^x \bmod p$	$R_2 = g^x \bmod p$
$K = (g^x \bmod p)^x \bmod p = g^{x^2} \bmod p$	$K = (g^x \bmod p)^x \bmod p = g^{x^2} \bmod p$

Both R1 and R2 are the same and the K is the same. For example, if we use $g = 7$ and $p = 23$, and $x = 3$, we get $R_1 = R_2 = 21$ and $K = 15$.

19. Appendix J gives us the first primitive root for a prime less than 1000. According to this appendix, the first primitive root of 53 is $g = 2$. Note that the number of primitive roots are $\phi(\phi(53)) = 24$. We can also find other primitive roots for 53 using one of the procedures given in the literature. The fastest one is when we know the prime factors of $\phi(p) = p - 1$. In this case, g is a primitive root if $g^{(p-1)/q} \bmod p \neq 1$ for all q's where q is a prime factor of $p - 1$. In this case,

$$\phi(p) = \phi(53) = 52 = 2^2 \times 13 \quad \rightarrow \quad q = 2 \text{ and } q = 13$$

We need to check the powers of 52/13 = 4 and 52/2 = 26. We give the proof for the first three primitives.

a. The first primitive is $g = 2$ because $2^4 \bmod 53 = 16 \neq \mathbf{1}$ and $2^{26} \bmod 53 = 52 \neq \mathbf{1}$.

b. The second primitive is $g = 3$ because $3^4 \bmod 53 = 28 \neq \mathbf{1}$ and $3^{26} \bmod 53 = 52 \neq \mathbf{1}$.

c. The third primitive is $g = 5$ because $5^4 \bmod 53 = 42 \neq \mathbf{1}$ and $5^{26} \bmod 53 = 52 \neq \mathbf{1}$.

20. A simple view of this attack can be viewed as follows (see Figure 15.12 of the textbook):

a. Eve intercepts the message sent in step 2, discards the message, and sends a new message with the same value of R_1 to Bob.

b. Bob receives the message, and judging by the sender address of the packet, he believes that the message is originated from Eve.

c. Bob, calculates R_2 and K and sends them, in step 5, to Eve (the sender of the message in step 2).

d. Eve intercepts the message and resends it to Alice pretending that it is coming from Bob (false sender address).

e. Alice calculate the session key and sends the message in step 8 to Bob.

f. Bob receives the message in step 8 and he knows that it is coming from Alice (sender address). Bob discards this message because he does expect that this message come from Alice; he is waiting to receive a message from Eve, which never happens.

g. The result is that Alice believes that a session key is established between herself and Bob; Bob believes that the session with Eve is broken.

21. The root certificate offered by some browsers are not hundred percent trustworthy because we are not sure if the browser actually check the validity of this certificates. The browsers belongs to private companies that their certificates are not necessarily endorsed by governmental authorities.

CHAPTER 16

PGP and S/MIME

(Solution to Practice Set)

Review Questions

1. Alice needs to include the identifiers of the algorithms in the packets sent to Bob. Each packet type has a field that defines the identity of the algorithm being used.
2. Alice needs to include the identifiers of the algorithms in the header sent to Bob.
3. The secret key is encrypted with the public key and sent with the message.
4. The secret key is encrypted with the public key and send with the message.
5. PGP uses a web of trust; S-MIME uses certificates signed by CA's, but the user is responsible to keep a web of trust.
6.
   - ❑ Session-key packet
   - ❑ Signature packet
   - ❑ Private-key packet
   - ❑ Compressed-data packet
   - ❑ Data packet encrypted with a secret key
   - ❑ Literal data packet
   - ❑ User ID packet
7.
   - ❑ Encrypted message
   - ❑ Signed message
   - ❑ Certified message
8.
   - ❑ Data Content Type
   - ❑ Signed-Data Content Type
   - ❑ Enveloped-Data Content Type

- ❑ Digest-Data Content Type
- ❑ Encrypted-Data Content Type
- ❑ Authenticated Data Content Type

9. In PGP, everyone in the community needs two rings (one public and one private); in S/MIME, the public keys are distributed through X.509 certificates.

Exercises

10. Note that when we talk about the tag value, we actually refer to the rightmost 6-bit values (ignoring the leftmost two bits):

- **a.** According to Table 16.12, a type value of 8 defines a *compressed data packet.*
- **b.** According to Table 16.12, a type value of 9 defines a *data packet encrypted with a secret key.*
- **c.** According to Table 16.12, a type value of 2 defines a *signature packet.*

11. Alice can use two public-key algorithms and two public keys each sent separately in a public-key packet.

12.

- **a.** A packet of type 1 can carry only the session key, not another packet.
- **b.** A packet of type 6 can carry only a public key, not another packet.

13.

- **a.** For confidentiality, two packets need to be sent. A session key packet (type 1) and an encrypted data packet (type 9). However, the second packet contains either a compressed data packet (which contains a literal data packet) or simply a literal data packet.
- **b.** For message integrity, two packets need to be sent. A signature packet (type 2) and a literal data packet (type 11).
- **c.** The packets in part *b* also provide authentication.
- **d.** Nonrepudiation needs a third party. Since e-mail communication is only between two parties, it is not possible to provide this security service.
- **e.** To provide both confidentiality and message integrity, four packets are needed to be sent (type 1, type 9, type 2 and type 11).
- **f.** Same as part *e*.
- **g.** Same as part *e*.
- **h.** This is impossible because there is no third party to provide nonrepudiation.

14.

- **a.** (Enveloped-Data)
- **b.** (Digested-Data or Signed-Data)
- **c.** (Authenticated-Data)

d. Nonrepudiation needs a third party. Since e-mail communication is only between two parties, it is not possible to provide this security service.

e. (Enveloped-Data) + (Digested-Data or Signed-Data)

f. (Enveloped-Data) + (Authenticated-Data)

g. (Enveloped-Data) + (Digested-Data or Signed-Data) + (Authenticated-Data)

h. This is impossible because there is no third party to provide nonrepudiation.

15. The following table shows the comparison:

Algorithms	***PGP***	***S/MIME***
No Encryption	✓	▨
IDEA	✓	▨
Triple DES	✓	✓
CAST-128	✓	▨
Blowfish	✓	▨
SAFER-SK 128	✓	▨
DES/SK	✓	▨
AES-128	✓	✓
AES-192	✓	▨
AES-256	✓	▨
RC2/40		✓

16. The following table shows the comparison. Note that S/MIME does not support any public-key algorithm for encryption/decryption.

Algorithms	***PGP***	***S/MIME***
RSA	✓	▨
ElGaml	✓	▨
Elliptic curve	✓	▨

17. The following table shows the comparison:

Algorithms	***PGP***	***S/MIME***
MD2	✓	▨
MD5	✓	✓
SHA-1	✓	✓
double-width SHA	✓	▨
RIPEMED/160	✓	▨
TIGER/192	✓	▨
HAVAL	✓	▨

18. The following table shows the comparison:

Algorithms	***PGP***	***S/MIME***
RSA	✓	✓
DSS	✓	✓
ECDSA	✓	▨
ElGamal	✓	▨

19.

a. Although the message can be sent without encoding, we show how to send it using Radix-64.The text to be sent in English is "This is a test". We add a null character at the end to make the English text multiple of 3 characters. We show the space character with "_".

Text	**ASCII code**			**R-64 code**				**Text**
`Thi`	01000001	01101000	01101001	010000	010110	100001	101001	`QWhp`
`s_i`	01110011	00100000	01101001	011100	110010	000001	101001	`cyBp`
`s_a`	01110011	00100000	01100001	011100	110010	000001	100001	`eyBh`
`_te`	00100000	01111100	01100101	001000	000111	110001	100101	`IHxl`
`st`	01110011	01111100	00000000	011100	110111	110000	000000	`c3wA`

The text to be sent is "QWhpcyBpeyBhIHxlc3wA".

b. The message consists only of ASCII characters, so the English text and the quoted-printable text are the same. The text to be sent is "This is a test".

CHAPTER 17

SSL and TLS

(Solution to Practice Set)

Review Questions

1. Five services are provided by SSL or TLS: *fragmentation*, *compression*, *message integrity*, *confidentiality*, and *framing*.
2. In SSL, a 48-byte master secret is created from the pre-master secret by applying two hash functions (SHA-1 and MD5).
3. TLS uses the PRF function to create the master secret from the pre-master secret. The first parameter (*secret*) is the pre-master secret; the second parameter (*label*) is the string "master secret"; the third parameter (*seed*) is the concatenation of the client random number and server random
4. In SSL, the master secret is used to create variable-length key materials by applying the same set of hash functions used to create the master secret and prepending with different constants. The module is repeated until key materials of adequate size are created.
5. TLS uses the PRF function to create key materials from the master secret. The first parameter (*secret*) is the master secret, the second parameter (*label*) is the string "key expansion", and the third parameter (*seed*) is the concatenation of the server random number and the client random number.
6. In a session, one party has the role of a client and the other party has the role of a server; in a connection, both parties have equal roles, they are peers. For two entities to exchange data, the establishment of a session is necessary, but not sufficient; they need to create a connection between themselves. A connection between two parties can be terminated and re-established within the same session.
7. The following list the four protocols:
   a. The Handshake Protocol uses messages to negotiate the cipher suite, to authenticate the server to the client and the client to the server if needed, and to exchange information for building the cryptographic secrets.
   b. The ChangeCipherSpec Protocol defines the process of moving cryptographic parameters between the pending and active states.

c. The Alert Protocol is used to report errors and abnormal conditions.

d. The Record Protocol carries messages from the upper layer (Handshake Protocol, ChangeCipherSpec Protocol, Alert Protocol, or application layer).

8. The following gives the goal of each phase:

a. In Phase I, the client and the server announce their security capabilities and choose those that are convenient for both. After this phase, the client and server know the version of SSL, the algorithms for key exchange, message authentication, and encryption, the compression method, and the two random numbers for key generation.

b. After this phase, the server is authenticated to the client and the client knows the public key of the server if required.

c. After this phase, the client is authenticated for the server and both the client and the server know the pre-master secret.

d. In Phase IV, the client and server send messages to change cipher specification and to finish the handshaking protocol. After phase IV, the client and server are ready to exchange data.

9. TLS uses the Handshake Protocol defined for SSL with only two small changes in CertificateVerify and Finished messages:

a. In SSL, the hash used in the CertificateVerify message is the two-step hash of the handshake messages plus a pad and the master secret. in TLS the hash is only over the handshake messages.

b. In TLS, a pseudorandom function (PRF) is used to calculate two hashes used for the Finished message.

10. TLS uses the Record Protocol defined for SSL with only one small change: SSL uses a MAC to sign the message, but TLS uses an HMAC to do so.

Exercises

11. The following table shows the size of the key-material in each case

	Client Auth.	Server Auth.	Client Enc.	Server Enc.	Client IV	Server IV	**Total Size**
a.	**1024**	**1024**	**0**	**0**	**0**	**0**	**2048**
b.	**1024**	**1024**	**0**	**0**	**0**	**0**	**2048**
c.	**1024**	**1024**	**56**	**56**	**64**	**64**	**2288**
d.	**1024**	**1024**	**168**	**168**	**64**	**64**	**2512**
e.	**1024**	**1024**	**56**	**56**	**64**	**64**	**2288**
f.	**1024**	**1024**	**168**	**168**	**64**	**64**	**2512**

We assume that the size of key for RSA authentication is 1024 bits although it can be 512 bits.We also assume that single DES uses a 56-bit bits and triple DES uses a key of 168 bits.

12. The number of iterations are different when the protocol is SSL from when from when it is TLS.

a. **SSL:** 2048 / 128 → 16 iterations of MD5

b. **SSL:** 2048 / 128 → 16 iterations of MD5

c. **TLS:** 2228 / 128 → 18 iterations of MD5, but 2228 / 160 → 14 iterations of SHA

d. **TLS:** 2512 / 128 → 20 iterations of MD5, but 2512 / 160 → 16 iterations of SHA

e. **TLS:** 2228 / 128 → 18 iterations of MD5, but 2228 / 160 → 14 iterations of SHA

f. **TLS:** 2512 / 128 → 20 iterations of MD5, but 2512 / 160 → 16 iterations of SHA

13. At first glance, it looks that TLS uses the premaster secret only once to create the master secret, but if we look more carefully at the data expansion function and the PRF function, we see that this calculation in TLS is more complex than the corresponding calculation in SSL. We believe the calculation in TLS is less efficient than the one in SSL.

14.

a. We believe that the calculation in TLS is more complex than the one in SSL and probably less efficient.

b. Since the calculation in TLS looks more complex, it should be more secure. However, there is not enough evidence in this case.

15. Although TLS uses only one PRF function, the PRF function is made of two data expansion function and each expansion function is an iteration of two-stage HMAC calculation. Therefore, TLS also uses iteration to create variable-size key materials although it is not as explicit as the SSL in this issue.

16. Only six messages are needed to resume a session:

1. ClientHello	Client	→	Server
2. ServerHello	Client	←	Server
3. ChangeCipherSpec	Client	←	Server
4. Finished	Client	←	Server
5. ChangeCipherSpec	Client	→	Server
6. Finished	Client	→	Server

17. Authentication keys, encryption keys, and IV's need to be created. The premaster and master secret do not need to be created again.

18. The client needs to send the ChangeCipherSpec message first. The server does not send its ChangeCipherSpec message until the one from the client has arrived. So this situation never occurs.

19. The calculation in TLS is more consistent to other standards for MAC calculation (padding the secret and exclusive-oring them with ipad or opad). We believe the efficiency of both methods is the same.

20. The hash algorithm in TLS is more efficient and simpler than the one in SSL. The first has only level of hashing; the second has two levels of hashing.

21. It is difficult to say which one is more efficient, but the one used in TLS looks more secure because it creates two different digests from the handshake message using two different algorithms (MD5 and SHA-1).

22. The only reason that comes to mind is that the designer of TLS wanted this digest to be calculated faster.

23.

a.

Key Material = MD5 (M | SHA-1 ("A" | M | CR | SR)) |
MD5 (M | SHA-1 ("BB" | M | CR | SR)) |
MD5 (M | SHA-1 ("…" | M | CR | SR)) |
…

b.

MAC = Hash (WriteSecret | pad-2 | Hash (WriteSecret | Pad-1 |
Sequence number | Compressed type | Compressed length
|Compressed fragment))

c.

Hash Digest = Hash (M | pad-2 | Hash (Handshake message | M | **Pad-1))**

d.

Hash Digest = Hash (M | pad-2 | Hash (Handshake message | M | **Pad-1))**

e.

Expanded Secret = $\text{HMAC}_{\text{Secret}}$ ($\text{HMAC}_{\text{Secret}}$ (Seed) | Seed) |
$\text{HMAC}_{\text{Secret}}$ ($\text{HMAC}_{\text{Secret}}$ (X) | Seed) |
$\text{HMAC}_{\text{Secret}}$ ($\text{HMAC}_{\text{Secret}}$ (Y) | Seed) |
…

Where X = $\text{HMAC}_{\text{Secret}}$ **(Seed), Y =** $\text{HMAC}_{\text{Secret}}$ **(X),** …

f.

New Secret	=	**MD5 (Label | Seed) $\oplus$ SHA-1 (Label | Seed)**

g.

Master Secret	=	**PRF (PM, "Master Secret", CR | SR)**

h.

Key Material	=	**PRF (M, "Key Expansion", SR | CR)**

i.

Hash Digest	=	**Hash (Handshake Message)**

j.

Hash Digest	=	**PRF (M | Finished Label | MD5 (Handshake Message) | SHA-1 (Handshake Message))**

k.

HMAC	=	**Hash (MAC Secret $\oplus$ opad | Hash (MAC Secret $\oplus$ opad | X | Compressed Fragment)**

X = Sequence number | Compressed type | Compressed version | Compress Length

24. The handshake protocol uses the idea of symmetric-key agreement for establishing session secrets between two parties. If strong key exchange algorithms (such as RSA, Ephemeral Diffie-Hellman, or Fixed Diffie-Hellman) is used for key agreement, the protocol is more immune to the man-in-the-middle attack. If weak key exchange algorithms (such as Anonymous Diffie-Hellman) is used for key agreement, the protocol is less immune to the man-in-the-middle attack.

25. The key size in SSL or TLS depends on the algorithm used for encryption. If an encryption algorithm with small key-size (such as single DES) is used, the protocol is less immune to brute-force attack. If an encryption algorithm with a large key size is used (such as 3DES), the protocol is more immune to brute-force attack.

26. The main attack on short keys is the brute-force attack. As described in Exercise 25. Some algorithms (such as single DES) use small key size that make either protocol (SSL or TLS) vulnerable to brute-force attack.

27. The two protocol are equally immune to the man-in-the-middle attack. The immunity depends on the type of algorithm used for key exchange as discussed in Exercise 24.

CHAPTER 18

IPSec

(Solution to Practice Set)

Review Questions

1. IPSec operates in one of the two modes: the transport mode or tunnel mode.
   a. In the transport mode, IPSec protects what is delivered from the transport layer to the network layer. In other words, the transport mode protects the network layer payload.
   b. In the tunnel mode, IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec algorithm to the entire packet, and then adds a new IP header.
2. AH is the simpler protocol in IPSec. It only adds a header (and some padding) to the IP payload. AH provides source authentication and data integrity, but not privacy.
3. ESP is the more sophisticated protocol in IPSec. It adds both a header and a trailer to the IP payload. ESP provides source authentication, data integrity, and privacy.
4. A Security Association is a contract between two parties; it creates a secure channel between them. SA establishes the security parameters between the two parties.
5. SAD is a set of SAs that can be collected into a database. The database can be thought of as a two-dimensional table with each row defining a single SA.
6. SP defines the type of security applied to a packet when it is to be sent or when it has arrived. Before using a SA, a host must determine the predefined policy for the packet. Each host that is using the IPSec protocol needs to keep a Security Policy Database (SPD).
7. The Internet Key Exchange (IKE) is a protocol designed to create Security Associations. When a peer needs to send an IP packet, it consults the Security Policy Database (SPD) to see if there is an SA for that type of traffic. If there is no SA, IKE is called to establish one. In other words, IKE creates SAs for IPSec.
8. IKE uses two phases: phase I and phase II. Phase I creates SAs for phase II; phase II creates SAs for a data exchange protocol such as IPSec.

9. The ISAKMP protocol is designed to carry messages for the IKE exchange.
10. ISAKMP uses fourteen payload types as shown below:
    a. ***None***: Shows the end of the payloads.
    b. ***SA***: Defines a packet that start the negotiation
    c. ***Proposal***: Defines a packet that contains information used during SA negotiation
    d. ***Transform***: Defines a packet that actually defines parameters to create a secure channel
    e. ***Key Exchange***: Defines a packet that carries data used for generating keys
    f. ***Identification***: Defines a packet that carries the identification of communication peers
    g. ***Certification***: Defines a packet that carries a public-key certificate
    h. ***Certification Request***: Defines a packet that carries a request for certificate from the other party
    i. ***Hash***: Defines a packet that carries data generated by a hash function
    j. ***Signature***: Defines a packet that carries data generated by a signature function
    k. ***Nonce***: Defines a packet that carries randomly generated data as a nonce
    l. ***Notification***: Defines a packet that carries error message or status associated with an SA
    m. ***Delete***: Defines a packet that carries one more SA that the sender has deleted
    n. ***Vendor***: Defines a packet that carries vendor-specification extensions

Exercise

11. Since the sequence number of the packet (181) is out of the window (200 to 264), the packet is discarded. It is either duplicate or its arrival time has expired. The window span does not change.
12. The sequence number of the packet (208) is inside the window (200 to 263). If the packet is new (the corresponding slot is not marked), the packet is accepted and the corresponding slot is marked. If the corresponding slot is marked, the packet is a duplicate packet; it is discarded. There is no change in the window span in either case.
13. The sequence number of the packet (331) is at the right of the window. Since the packet is authenticated, it is accepted and the window span will change as shown in Figure S18.13.

14. Figure S18.14 shows the diagrams.

Figure S18.13 *Solution to Exercise 13*

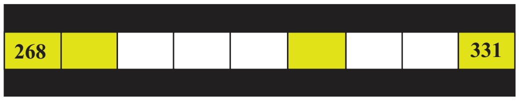
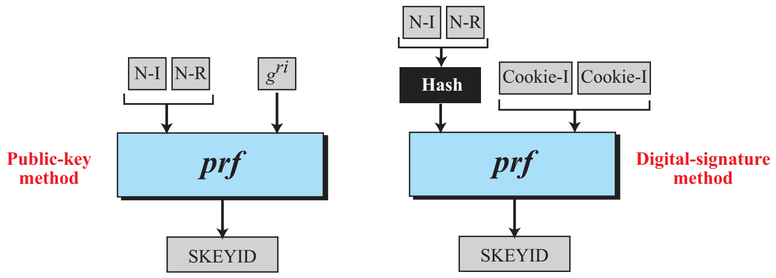
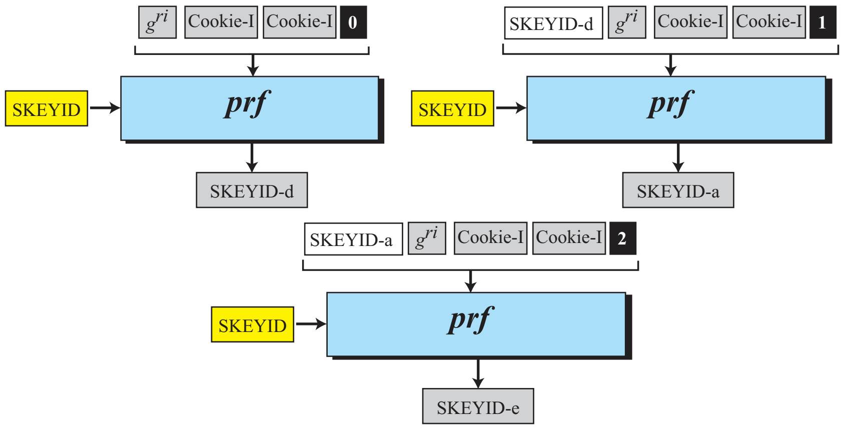


Figure S18.14 *Solution to Exercise 14*



15. Figure S18.15 shows the diagram.

Figure S18.15 *Solution to Exercise 15*



16. Figure S18.16 shows the diagram.

17. Figure S18.17 shows the diagram.

18. Figure S18.18 shows the diagram.

Figure S18.16 *Solution to Exercise 16*

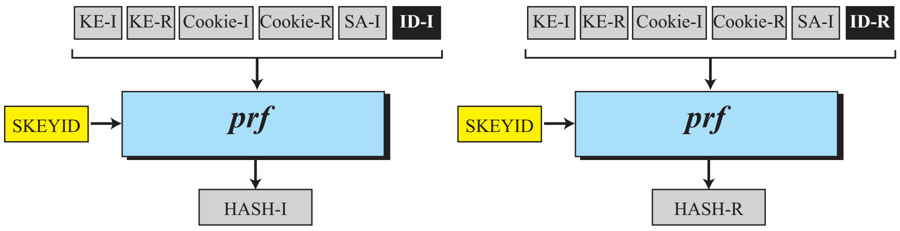


Figure S18.17 *Solution to Exercise 17*

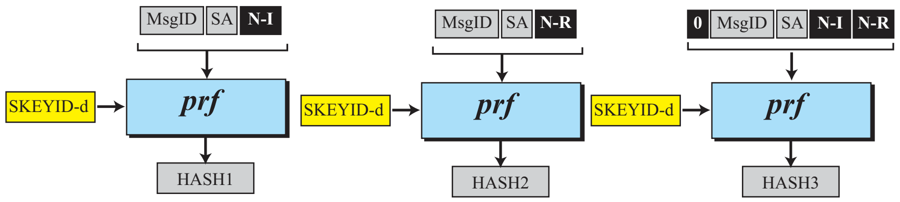
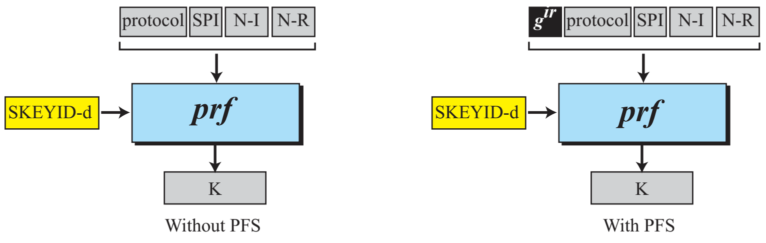
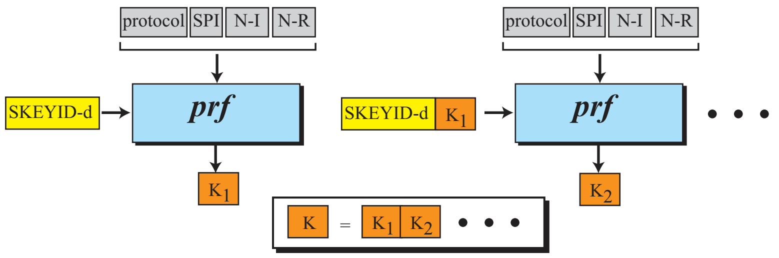


Figure S18.18 *Solution to Exercise 18*



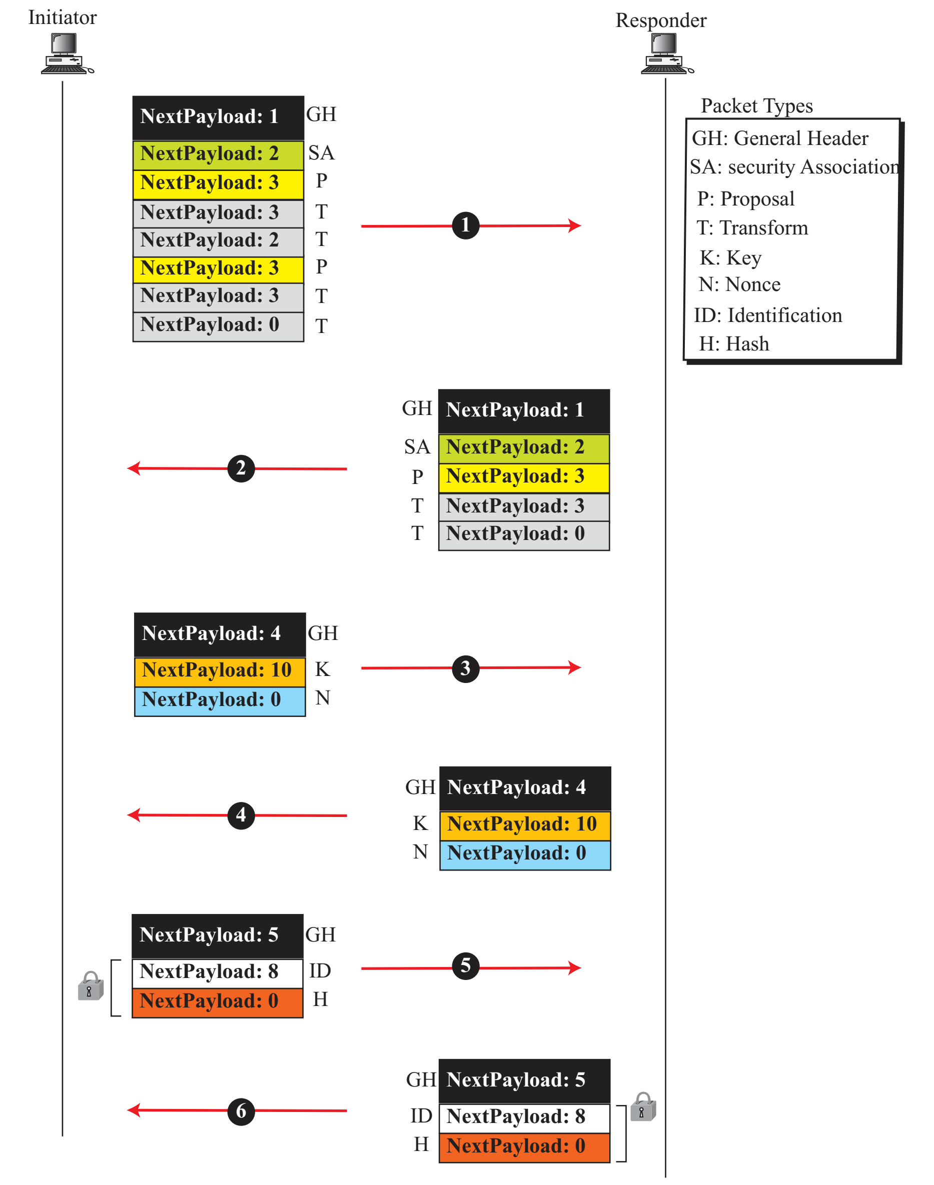
19. Figure S18.19 shows the diagram without using PSF. The one with PSF is similar (see the solution to Exercise 18).

Figure S18.19 *Solution to Exercise 19*



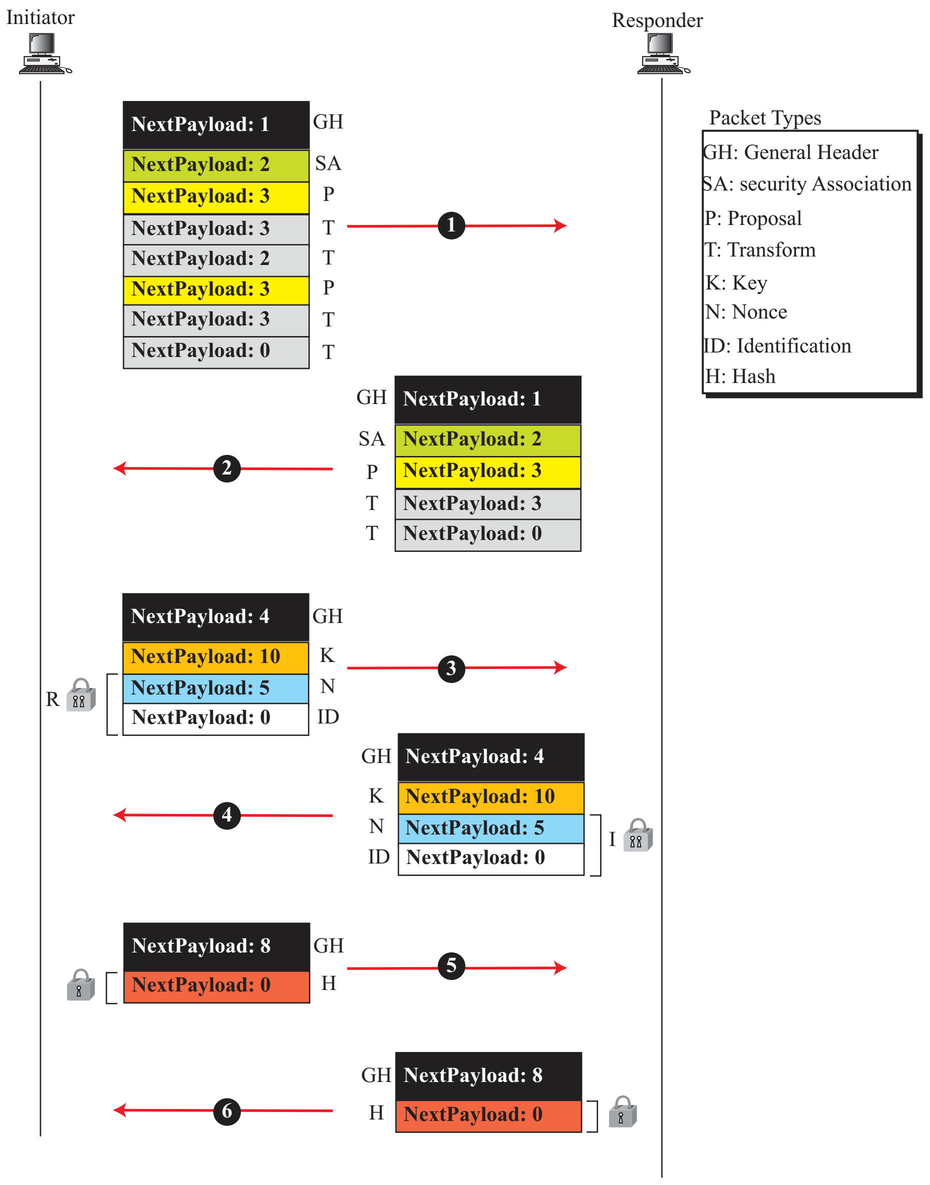
20. Figure S18.20 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet.

Figure S18.20 *Solution to Exercise 20*



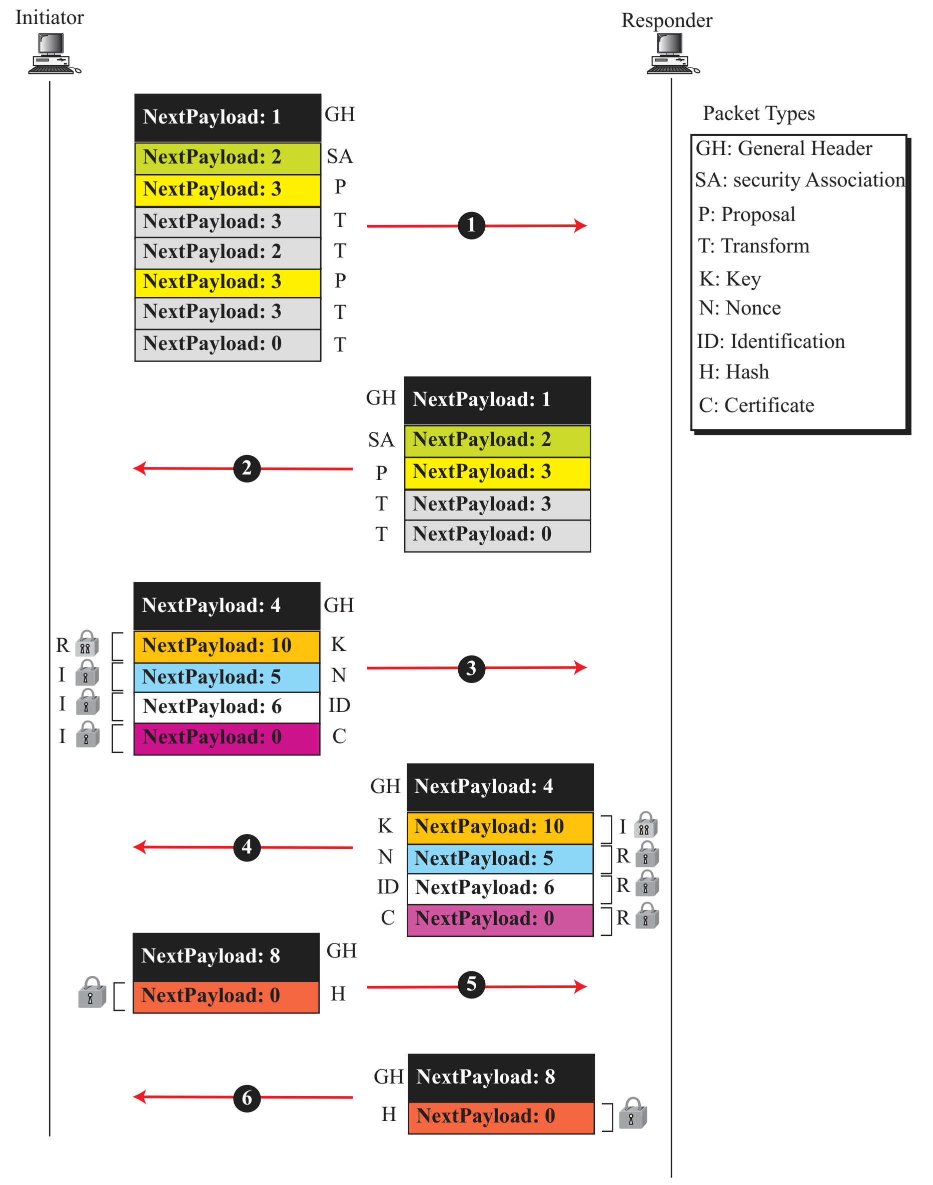
21. Figure S18.21 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet.

Figure S18.21 *Solution to Exercise 21*



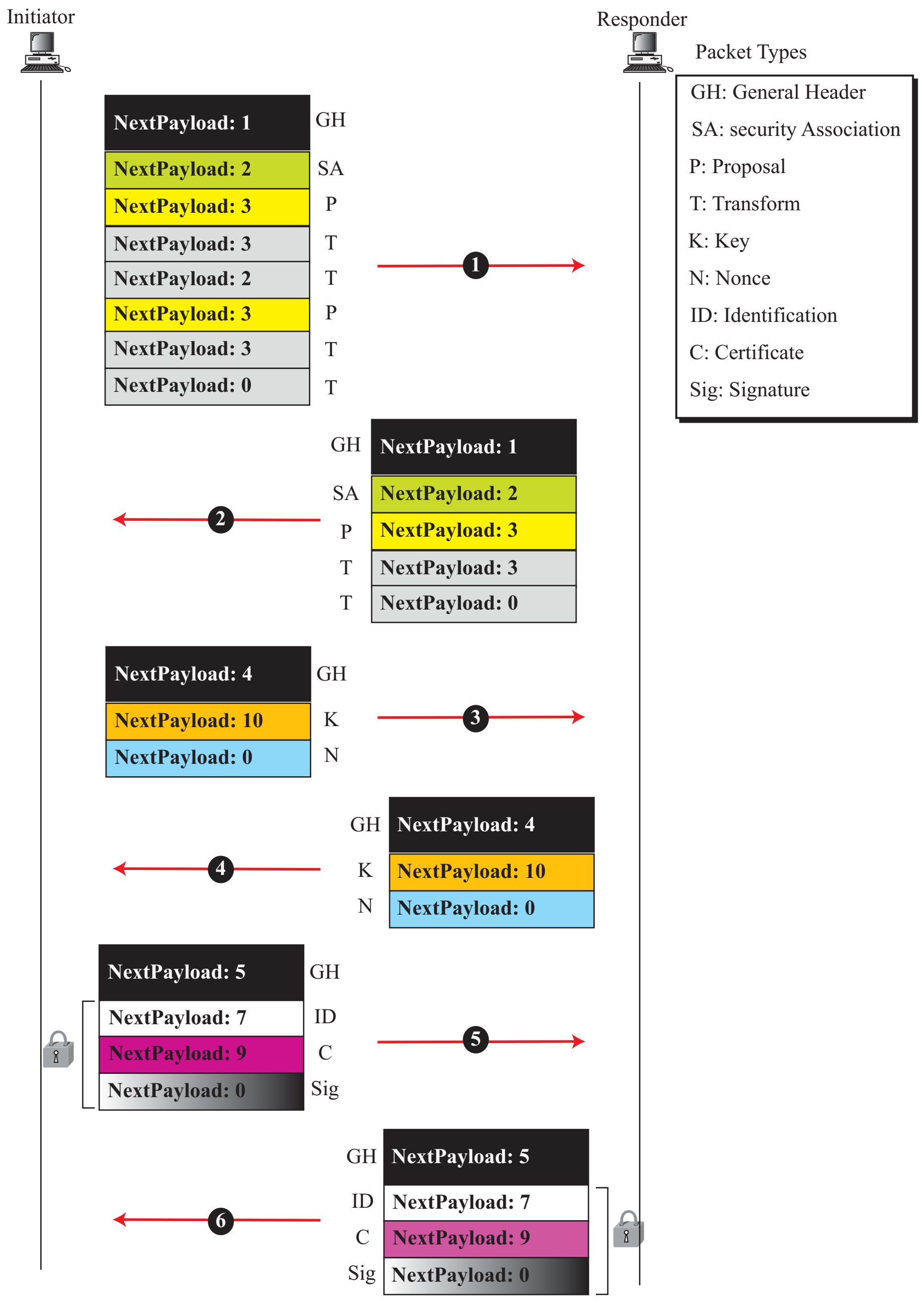
22. Figure S18.22 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet.

Figure S18.22 *Solution to Exercise 22*



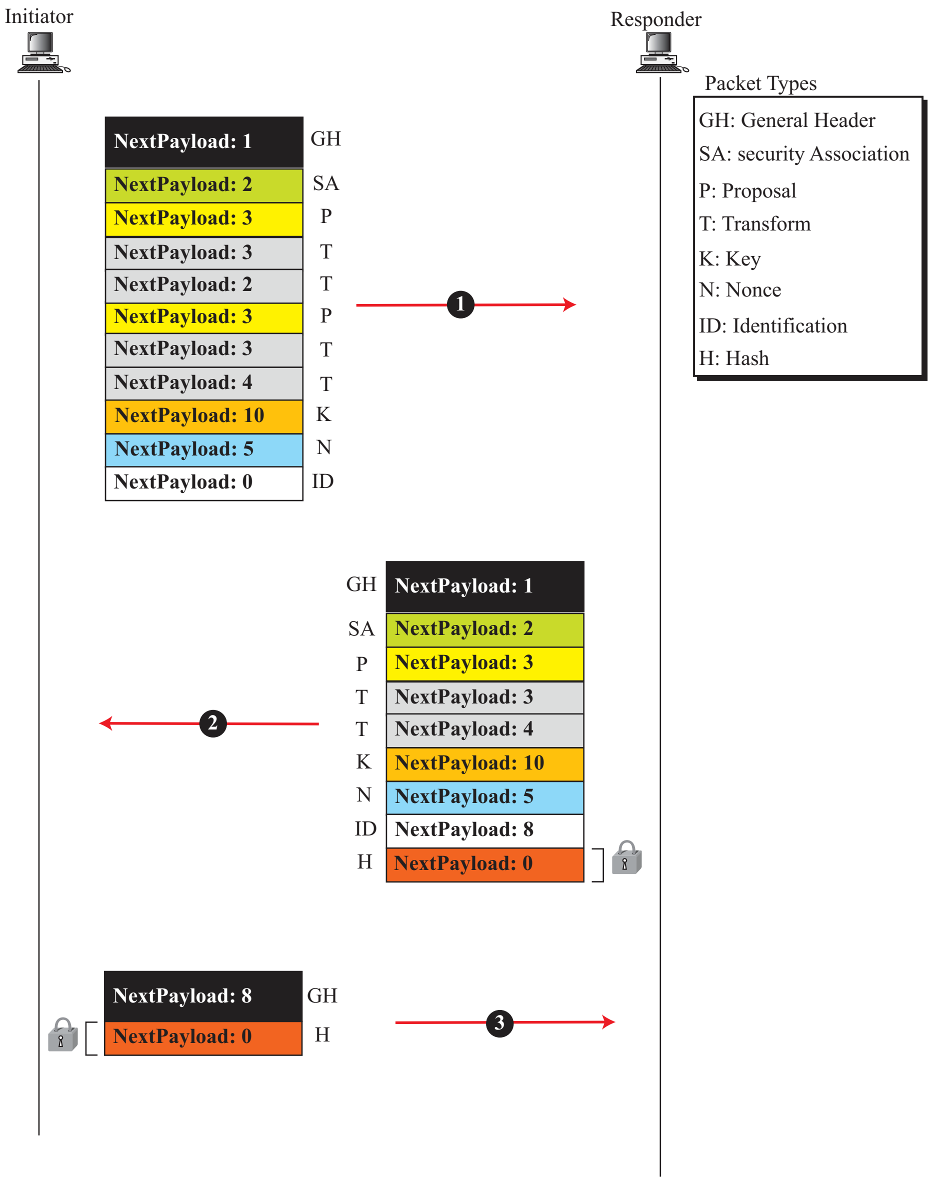
23. Figure S18.23 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet.

Figure S18.23 *Solution to Exercise 23*



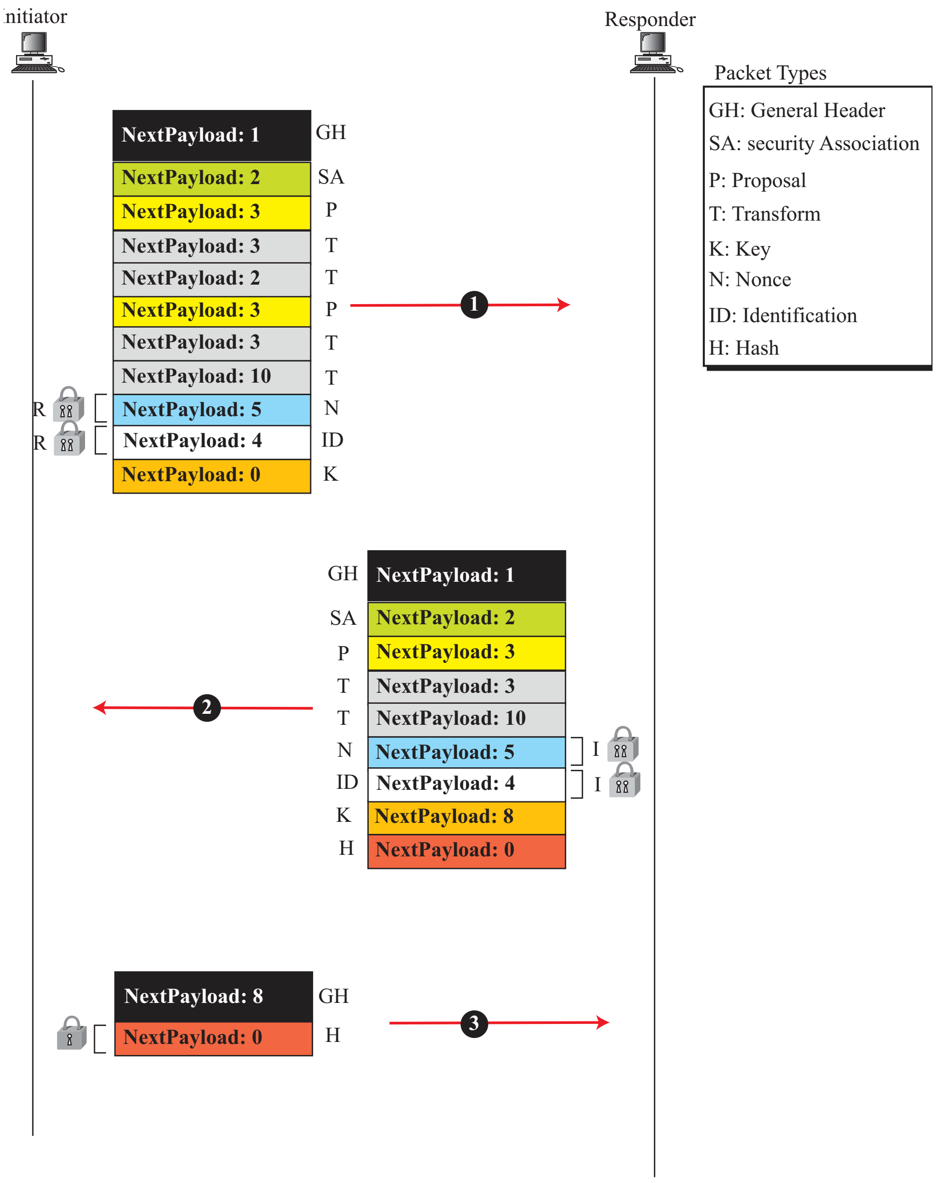
24. Figure S18.24 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet. Compare this method with the corresponding method in the main mode.

Figure S18.24 *Solution to Exercise 24*



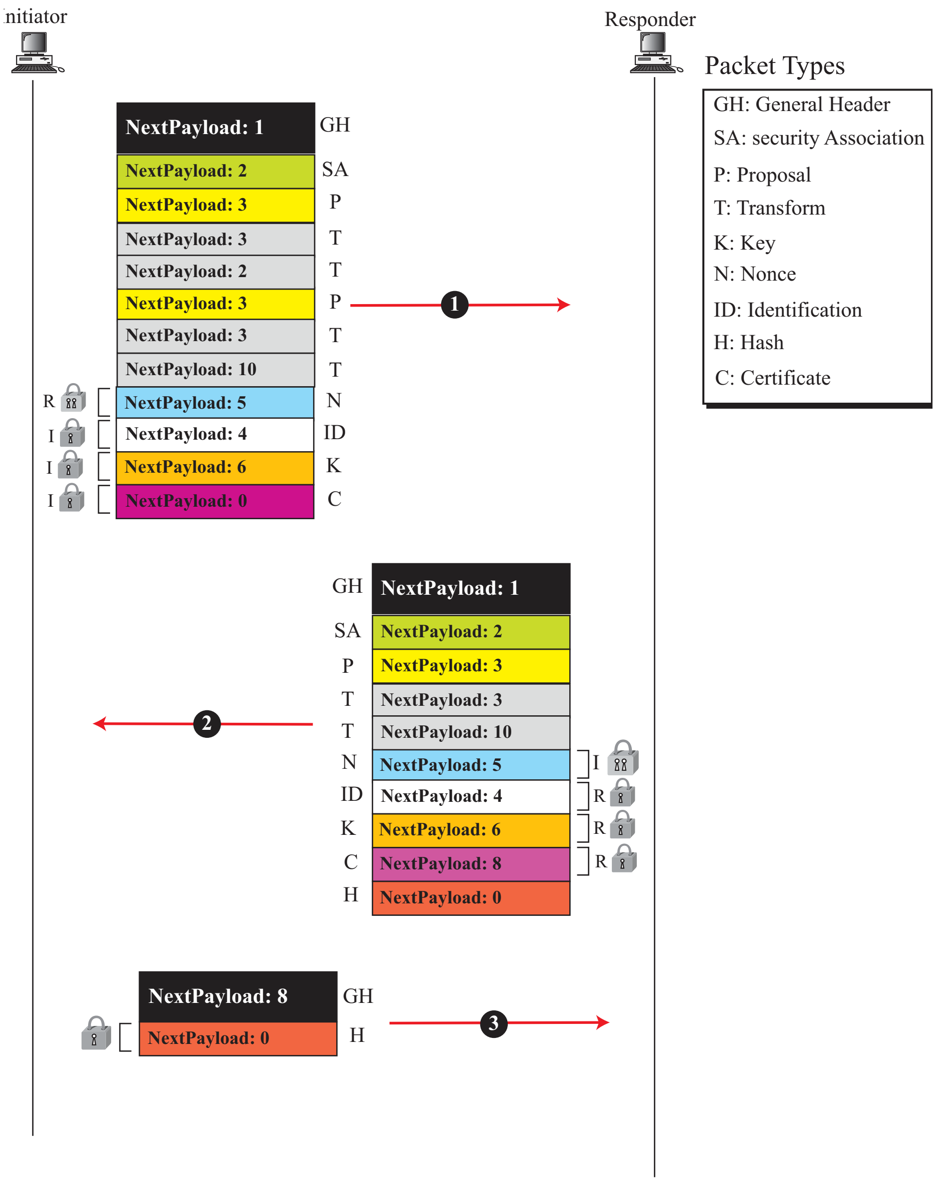
25. Figure S18.25 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet. Compare this method with the corresponding method in the main mode.

Figure S18.25 *Solution to Exercise 25*



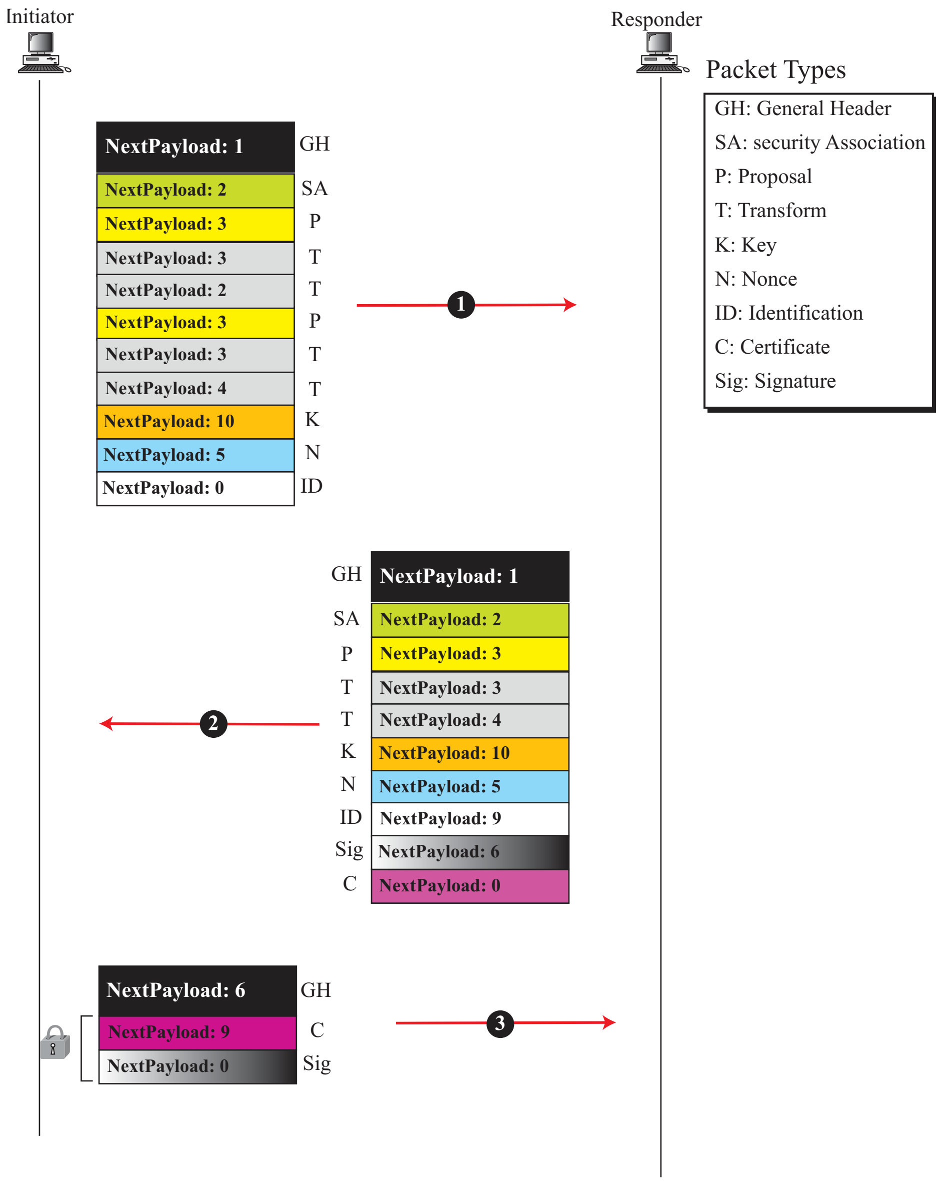
26. Figure S18.26 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet. Compare this method with the corresponding method in the main mode.

Figure S18.26 *Solution to Exercise 26*



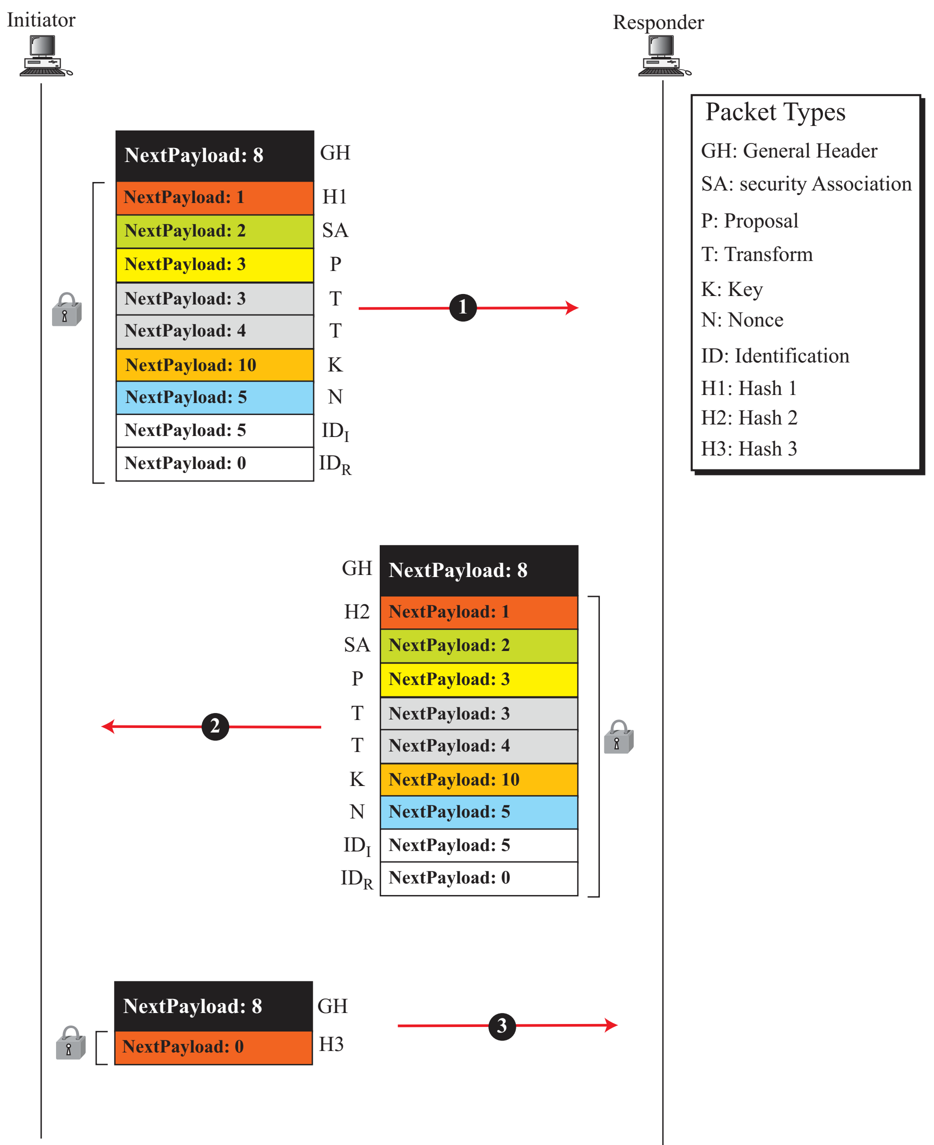
27. Figure S18.27 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet. Compare this method with the corresponding method in the main mode.

Figure S18.27 *Solution to Exercise 27*



28. Figure S18.28 shows only the layout of the packets and next payload value. When a packet is encrypted, a padlock is inserted next to the packet. Note that here two ID packets, ID-I and ID-R is included in the first and second message. Note that we also have three hash functions: Hash1, Hash2, and Hash3. However, the packet is used to carry these hashes are the same. A very interesting point is that only the third message is encrypted in this method; the signature in the second message is not encrypted.

Figure S18.28 *Solution to Exercise 28*



29.

- **a.** The security compromise is that the ID's of the initiator and the responder are not encrypted in the aggressive mode.
- **b.** The gain is that only three messages are sent instead of six.

30.

- **a.** The security compromise is that the hash created by the responder is not encrypted in the aggressive mode.
- **b.** The gain is that only three messages are sent instead of six.

31.

- **a.** The security compromise is that the hash created by the responder is not encrypted in the aggressive mode.
- **b.** The gain is that only three messages are sent instead of six.

32.

- **a.** The security compromise is that the certificate and the signature of the responder is not encrypted.
- **b.** The gain is that only three messages are sent instead of six.

33. SKEYID is calculated differently in different method, but provision is made in each method to protect it from the intrusion.

- **a.** In the preshared secret-key method, SKEYID is calculated from the preshared secret key method which is supposed to be secured from intrusion.
- **b.** In the public-key method, SKEYID is calculated from N-I and N-R, which are secretly exchanged using the public keys of two parties.
- **c.** In the digital signature, SKEYID is calculated from the hashed values of N-R and N-R, which are secured if the hash function is a secured one.

34. In the preshared secret-key method, the preshared secret key is a function of the IP address. Now if the ID is also a function of the IP address, then there is a vicious circle. The receiver of the fifth or sixth message in the main mode, needs to know the ID of the sender to decrypt the message, but the ID itself is inside the message that needs to be decrypted.

35. All methods in the main mode protect the exchanges of ID's, some in the third or fourth messages and some in the fifth and sixth messages.

36. Only public-key methods in the aggressive method explicitly protect the exchanges of ID's in the aggressive mode. However, since the hash in the preshared secret method and the signature in the signature method contain a copy of ID's, we can say the exchange of ID's are implicitly protected in these methods.

37. The exchange of N-I and N-R in the third and fourth messages protects these messages from being replayed. The inclusion of these values again in the encrypted hash or signature in the fifth and sixth messages glues the whole session together and protects the session against partial replay.

38. The exchange of N-I and N-R in the first and second messages protects these messages from being replayed. The inclusion of these values again in the encrypted hash or signature in the third message glues the whole session together and protects the session against partial replay.

39. The exchange of encrypted N-I and N-R in the first and second messages protects these messages from being replayed. The inclusion of these values again in the encrypted hash or signature in the third message glues the whole session together and protects the session against partial replay.

40. The effectiveness of the brute-force attack depends on the size of the secrets exchanged between two parties. Since IKE allows the concatenation of keys to create a larger key, brute-force attack can be protected by creating a larger key.