The Data Encryption Standard 70
3.1 Introduction.. 70
3.2 Description of DES 70
FIGURE 3.1... 71
FIGURE 3.2... 72
FIGURE 3.3... 29
3.2.1 An Example of DES Encryption............. 79
3.3 The DES Controversy 82
3.4 DES in Practice 83
3.4.1 DES Modes of Operation 83
FIGURE 3.4.. 84
FIGURE 3.5.. 85
3.5 A Time-memory Trade-off 86
FIGURE 3.6... 87
FIGURE 3.7... 88
3.6 Differential Cryptanalysis.......................... 89
FIGURE 3.8... 91
3.6.1 An Attack on a 3-round DES 93
FIGURE 3.9.. 94
3.6.2 An Attack on a 6-round DES 98
FIGURE 3.10.. 99
FIGURE 3.12.. 99
FIGURE 3.13.. 101
FIGURE 3.14.. 104
3.6.3 Other examples of Differential 104
TABLE 3.1... 105
3.7 Notes and References............................. 110
Exercises.. 110
FIGURE 3.15... 112
FIGURE 3.16... 112

3

The Data Encryption Standard

3.1 Introduction

On May 15, 1973, the National Bureau of Standards published a solicitation for cryptosystems in the Federal Register. This lead ultimately to the development of the **Data Encryption Standard**, or **DES**, which has become the most widely used cryptosystem in the world. **DES** was developed at IBM, as a modification of an earlier system known as **LUCIFER**. **DES** was first published in the Federal Register of March 17, 1975. After a considerable amount of public discussion, **DES** was adopted as a standard for "unclassified" applications on January 15, 1977. **DES** has been reviewed by the National Bureau of Standards (approximately) every five years since its adoption. Its most recent renewal was in January 1994, when it was renewed until 1998. It is anticipated that it will not remain a standard past 1998.

3.2 Description of DES

A complete description of **DES** is given in the Federal Information Processing Standards Publication 46, dated January 15, 1977. **DES** encrypts a plaintext bitstring x of length 64 using a key K which is a bitstring of length 56, obtaining a ciphertext bitstring which is again a bitstring of length 64. We first give a "high-level" description of the system.

The algorithm proceeds in three stages:

1. Given a plaintext x, a bitstring x_0 is constructed by permuting the bits of x according to a (fixed) *initial permutation* IP. We write $x_0 = \mathrm{IP}(x) = L_0R_0$, where L_0 comprises the first 32 bits of x_0 and R_0 the last 32 bits.
2. 16 iterations of a certain function are then computed. We compute L_iR_i,

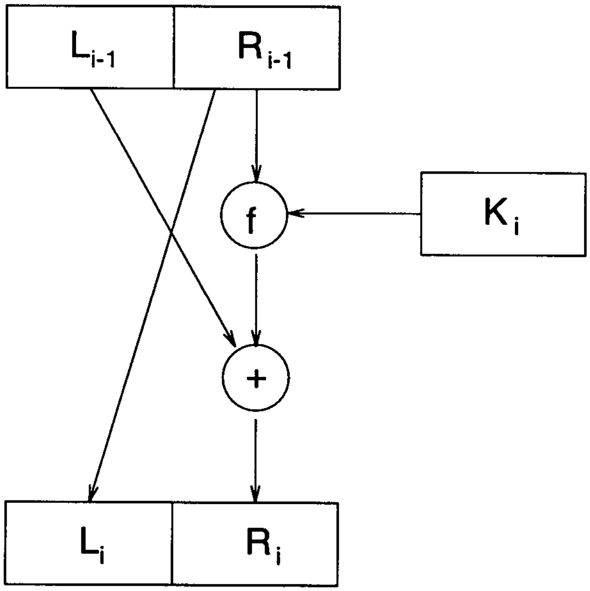

FIGURE 3.1
One round of DES encryption

$1 \leq i \leq 16$, according to the following rule:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i),$$

where $\oplus$ denotes the exclusive-or of two bitstrings. f is a function that we will describe later, and $K_1, K_2, \ldots, K_{16}$ are each bitstrings of length 48 computed as a function of the key K. (Actually, each K_i is a permuted selection of bits from K.) $K_1, K_2, \ldots, K_{16}$ comprises the *key schedule*. One round of encryption is depicted in Figure 3.1

3. Apply the inverse permutation IP^{-1} to the bitstring $R_{16}L_{16}$, obtaining the ciphertext y. That is, $y = \mathrm{IP}^{-1}(R_{16}L_{16})$. Note the inverted order of L_{16} and R_{16}.

The function f takes as input a first argument A, which is a bitstring of length 32, and a second argument J that is a bitstring of length 48, and produces as output a bitstring of length 32. The following steps are executed.

1. The first argument A is "expanded" to a bitstring of length 48 according to a fixed *expansion function* E. $\mathrm{E}(A)$ consists of the 32 bits from A, permuted in a certain way, with 16 of the bits appearing twice.

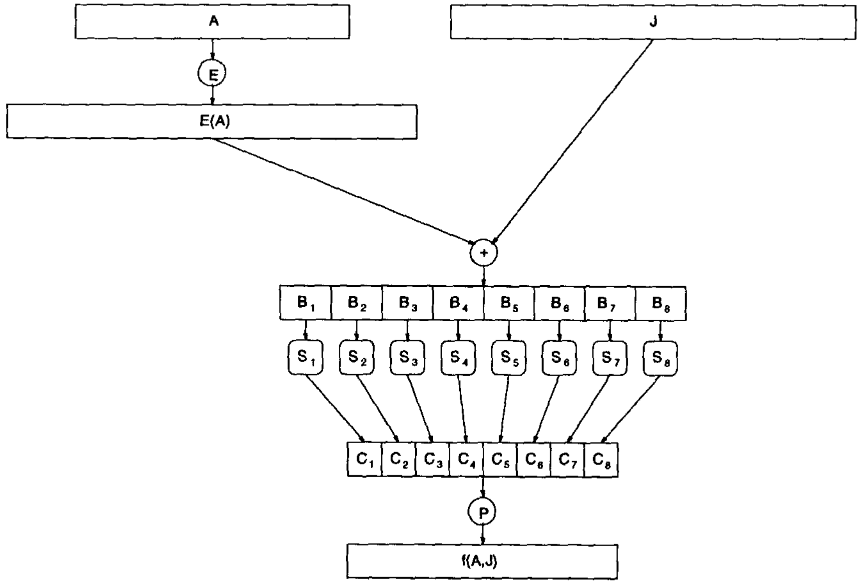

FIGURE 3.2
The DES f function

2. Compute $\mathrm{E}(A) \oplus J$ and write the result as the concatenation of eight 6-bit strings $B = B_1B_2B_3B_4B_5B_6B_7B_8$.
3. The next step uses eight *S-boxes* $S_1, \ldots, S_8$. Each S_i is a fixed 4×16 array whose entries come from the integers $0 - 15$. Given a bitstring of length six, say $B_j = b_1b_2b_3b_4b_5b_6$, we compute $S_j(B_j)$ as follows. The two bits b_1b_6 determine the binary representation of a row r of S_j $(0 \leq r \leq 3)$, and the four bits $b_2b_3b_4b_5$ determine the binary representation of a column c of S_j $(0 \leq c \leq 15)$. Then $S_j(B_j)$ is defined to be the entry $S_j(r, c)$, written in binary as a bitstring of length four. (Hence, each S_j can be thought of as a function that accepts as input a bitstring of length two and one of length four, and produces as output a bitstring of length four.) In this fashion, we compute $C_j = S_j(B_j)$, $1 \leq j \leq 8$.
4. The bitstring $C = C_1C_2C_3C_4C_5C_6C_7C_8$ of length 32 is permuted according to a fixed permutation P. The resulting bitstring $\mathrm{P}(C)$ is defined to be $f(A, J)$.

The f function is depicted in Figure 3.2. Basically, it consists of a substitution (using an S-box) followed by the (fixed) permutation P. The 16 iterations of f comprise a product cryptosystem, as described in Section 2.5.

In the remainder of this section, we present the specific functions used in DES.

The initial permutation IP is as follows:

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

This means that the 58th bit of x is the first bit of IP(x); the 50th bit of x is the second bit of IP(x), etc.

The inverse permutation IP^{-1} is:

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

The expansion function E is specified by the following table:

E bit-selection table					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

The eight S-boxes and the permutation P are now presented:

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Finally, we need to describe the computation of the key schedule from the key K. Actually, K is a bitstring of length 64, of which 56 bits comprise the key and 8 bits are parity-check bits (for error-detection). The bits in positions $8, 16, \ldots, 64$ are defined so that each byte contains an odd number of 1's. Hence, a single error can be detected within each group of 8 bits. The parity-check bits are ignored in the computation of the key schedule.

1. Given a 64-bit key K, discard the parity-check bits and permute the remaining bits of K according to a (fixed) permutation PC-1. We will write PC-1$(K) = C_0D_0$, where C_0 comprises the first 28 bits of PC-1(K) and D_0 the last 28 bits.
2. For i ranging from 1 to 16, compute

$$C_i = LS_i(C_{i-1})$$
$$D_i = LS_i(D_{i-1}),$$

   and K_i = PC-2(C_iD_i). LS_i represents a cyclic shift (to the left) of either one or two positions, depending on the value of i: shift one position if $i = 1, 2, 9$ or 16, and shift two positions otherwise. PC-2 is another fixed permutation.

The key schedule computation is depicted in Figure 3.3.

The permutations PC-1 and PC-2 used in the key schedule computation are as follows:

PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

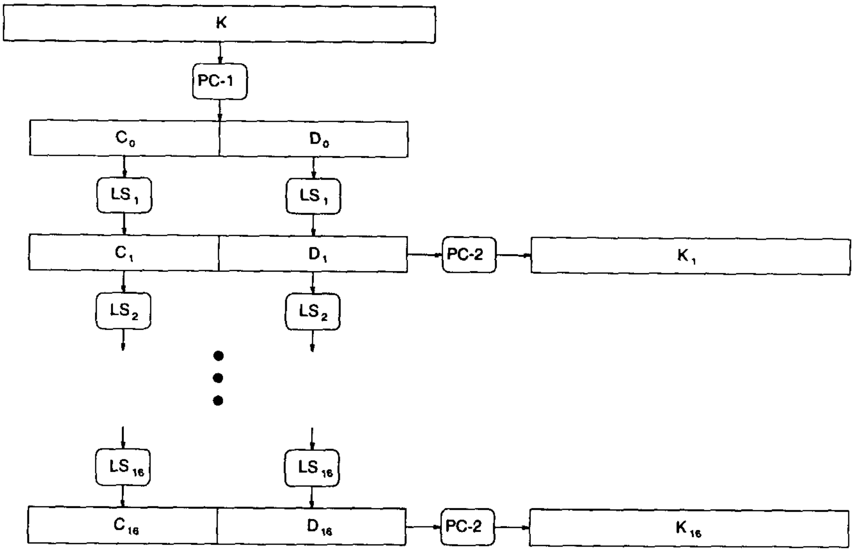

FIGURE 3.3
Computation of DES key schedule

PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

We now display the resulting key schedule. As mentioned above, each round uses a 48-bit key comprised of 48 of the bits in K. The entries in the tables below refer to the bits in K that are used in the various rounds.

Round 1											
10	51	34	60	49	17	33	57	2	9	19	42
3	35	26	25	44	58	59	1	36	27	18	41
22	28	39	54	37	4	47	30	5	53	23	29
61	21	38	63	15	20	45	14	13	62	55	31

Round 2											
2	43	26	52	41	9	25	49	59	1	11	34
60	27	18	17	36	50	51	58	57	19	10	33
14	20	31	46	29	63	39	22	28	45	15	21
53	13	30	55	7	12	37	6	5	54	47	23

Round 3											
51	27	10	36	25	58	9	33	43	50	60	18
44	11	2	1	49	34	35	42	41	3	59	17
61	4	15	30	13	47	23	6	12	29	62	5
37	28	14	39	54	63	21	53	20	38	31	7

Round 4											
35	11	59	49	9	42	58	17	27	34	44	2
57	60	51	50	33	18	19	26	25	52	43	1
45	55	62	14	28	31	7	53	63	13	46	20
21	12	61	23	38	47	5	37	4	22	15	54

Round 5											
19	60	43	33	58	26	42	1	11	18	57	51
41	44	35	34	17	2	3	10	9	36	27	50
29	39	46	61	12	15	54	37	47	28	30	4
5	63	45	7	22	31	20	21	55	6	62	38

Round 6											
3	44	27	17	42	10	26	50	60	2	41	35
25	57	19	18	1	51	52	59	58	49	11	34
13	23	30	45	63	62	38	21	31	12	14	55
20	47	29	54	6	15	4	5	39	53	46	22

Round 7											
52	57	11	1	26	59	10	34	44	51	25	19
9	41	3	2	50	35	36	43	42	33	60	18
28	7	14	29	47	46	22	5	15	63	61	39
4	31	13	38	53	62	55	20	23	37	30	6

Round 8											
36	41	60	50	10	43	59	18	57	35	9	3
58	25	52	51	34	19	49	27	26	17	44	2
12	54	61	13	31	30	6	20	62	47	45	23
55	15	28	22	37	46	39	4	7	21	14	53

Round 9											
57	33	52	42	2	35	51	10	49	27	1	60
50	17	44	43	26	11	41	19	18	9	36	59
4	46	53	5	23	22	61	12	54	39	37	15
47	7	20	14	29	38	31	63	62	13	6	45

Round 10											
41	17	36	26	51	19	35	59	33	11	50	44
34	1	57	27	10	60	25	3	2	58	49	43
55	30	37	20	7	6	45	63	38	23	21	62
31	54	4	61	13	22	15	47	46	28	53	29

Round 11											
25	1	49	10	35	3	19	43	17	60	34	57
18	50	41	11	59	44	9	52	51	42	33	27
39	14	21	4	54	53	29	47	22	7	5	46
15	38	55	45	28	6	62	31	30	12	37	13

Round 12											
9	50	33	59	19	52	3	27	1	44	18	41
2	34	25	60	43	57	58	36	35	26	17	11
23	61	5	55	38	37	13	31	6	54	20	30
62	22	39	29	12	53	46	15	14	63	21	28

Round 13											
58	34	17	43	3	36	52	11	50	57	2	25
51	18	9	44	27	41	42	49	19	10	1	60
7	45	20	39	22	21	28	15	53	38	4	14
46	6	23	13	63	37	30	62	61	47	5	12

Round 14											
42	18	1	27	52	49	36	60	34	41	51	9
35	2	58	57	11	25	26	33	3	59	50	44
54	29	4	23	6	5	12	62	37	22	55	61
30	53	7	28	47	21	14	46	45	31	20	63

Round 15											
26	2	50	11	36	33	49	44	18	25	35	58
19	51	42	41	60	9	10	17	52	43	34	57
38	13	55	7	53	20	63	46	21	6	39	45
14	37	54	12	31	5	61	30	29	15	4	47

Round 16											
18	59	42	3	57	25	41	36	10	17	27	50
11	43	34	33	52	1	2	9	44	35	26	49
30	5	47	62	45	12	55	38	13	61	31	37
6	29	46	4	23	28	53	22	21	7	63	39

Decryption is done using the same algorithm as encryption, starting with y as the input, but using the key schedule $K_{16}, \ldots, K_1$ in reverse order. The output will be the plaintext x.

3.2.1 An Example of DES Encryption

Here is an example of encryption using the **DES**. Suppose we encrypt the (hexadecimal) plaintext

`0123456789ABCDEF`

using the (hexadecimal) key

`133457799BBCDFF1.`

The key, in binary, without parity-check bits, is

00010010011010010101101111001001101101111011011111111000.

Applying IP, we obtain L_0 and R_0 (in binary):

L_0	=	11001100000000001100110011111111
$L_1 = R_0$	=	11110000101010101111000010101010

The 16 rounds of encryption are then performed, as indicated.

$E(R_0)$	=	011110100001010101010101011110100001010101010101
K_1	=	000110110000001011101111111111000111000001110010
$E(R_0) \oplus K_1$	=	011000010001011110111010100001100110010100100111
S-box outputs		01011100100000101011010110010111
$f(R_0, K_1)$	=	00100011010010101010100110111011
$L_2 = R_1$	=	11101111010010100110010101000100

$E(R_1)$	=	011101011110101001010100001100001010101000001001
K_2	=	011110011010111011011001110110111100100111100101
$E(R_1) \oplus K_2$	=	000011000100010010001101111010110110001111101100
S-box outputs		11111000110100000011101010101110
$f(R_1, K_2)$	=	00111100101010111000011110100011
$L_3 = R_2$	=	11001100000000010111011100001001

$E(R_2)$	=	111001011000000000000010101110101110100001010011
K_3	=	010101011111110010001010010000101100111110011001
$E(R_2) \oplus K_3$	=	101100000111110010001000111110000010011111001010
S-box outputs		00100111000100001110000101101111
$f(R_2, K_3)$	=	01001101000101100110111010110000
$L_4 = R_3$	=	10100010010111000000101111110100

$E(R_3)$	=	010100000100001011111000000001010111111110101001
K_4	=	011100101010110111010110110110110011010100011101
$E(R_3) \oplus K_4$	=	001000101110111100101110110111100100101010110100
S-box outputs		00100001111011011001111100111010
$f(R_3, K_4)$	=	10111011001000110111011101001100
$L_5 = R_4$	=	01110111001000100000000001000101

$E(R_4)$	=	101110101110100100000100000000000000000100001010
K_5	=	011111001110110000000111111010110101001110101000
$E(R_4) \oplus K_5$	=	110001100000010100000011111010110101000110100010
S-box outputs		01010000110010000011000111101011
$f(R_4, K_5)$	=	00101000000100111010110111000011
$L_6 = R_5$	=	10001010010011111010011000110111

$E(R_5)$	=	110001010100001001011111110100001100000110101111
K_6	=	011000111010010100111110010100000111101100101111
$E(R_5) \oplus K_6$	=	101001101110011101100001100000001011101010000000
S-box outputs		01000001111100110100110000111101
$f(R_5, K_6)$	=	10011110010001011100110100101100
$L_7 = R_6$	=	11101001011001111100110101101001

$E(R_6)$	=	111101010010101100001111111001011010101101010011
K_7	=	111011001000010010110111111101100001100010111100
$E(R_6) \oplus K_7$	=	000110011010111110111000000010011101100111101111
S-box outputs		00010000011101010100000010101101
$f(R_6, K_7)$	=	10001100000001010001110000100111
$L_8 = R_7$	=	00000110010010101011101000010000

$E(R_7)$	=	000000001100001001010101010111101000000010100000
K_8	=	111101111000101000111010110000010011101111111011
$E(R_7) \oplus K_8$	=	111101110100100001101111100111100111101101011011
S-box outputs		01101100000110000111110010101110
$f(R_7, K_8)$	=	00111100000011101000011011111001
$L_9 = R_8$	=	11010101011010010100101110010000

$E(R_8)$	=	011010101010101101010010101001010111110010100001
K_9	=	111000001101101111101011111011011110011110000001
$E(R_8) \oplus K_9$	=	100010100111000010111001010010001001101100100000
S-box outputs		00010001000011000101011101110111
$f(R_8, K_9)$	=	00100010001101100111110001101010
$L_{10} = R_9$	=	00100100011111001100011001111010

$E(R_9)$	=	000100001000000111111001011000001100001111110100
K_{10}	=	101100011111001101000111101110100100011001001111
$E(R_9) \oplus K_{10}$	=	101000010111000010111110110110101000010110111011
S-box outputs		11011010000001000101001001110101
$f(R_9, K_{10})$	=	01100010101111001001110000100010
$L_{11} = R_{10}$	=	10110111110101011101011110110010

$E(R_{10})$	=	010110101111111010101011111010101111110110100101
K_{11}	=	001000010101111111010011110111101101001110000110
$E(R_{10}) \oplus K_{11}$	=	011110111010000101111000001101000010111000100011
S-box outputs		01110011000001011101000100000001
$f(R_{10}, K_{11})$	=	11100001000001001111101000000010
$L_{12} = R_{11}$	=	11000101011110000011110001111000

$E(R_{11})$	=	011000001010101111111000000001111111000001111110001
K_{12}	=	011101010111000111110101100101000110011111101001
$E(R_{11}) \oplus K_{12}$	=	000101011101101000000101100010111110010000011000
S-box outputs		01111011100010110010011000110101
$f(R_{11}, K_{12})$	=	11000010011010001100111111101010
$L_{13} = R_{12}$	=	01110101101111010001100001011000

$E(R_{12})$	=	001110101011110111111010100011110000001011110000
K_{13}	=	100101111100010111010001111110101011101001000001
$E(R_{12}) \oplus K_{13}$	=	101011010111100000101011011101011011100010110001
S-box outputs		10011010110100011000101101001111
$f(R_{12}, K_{13})$	=	11011101101110110010100100100010
$L_{14} = R_{13}$	=	00011000110000110001010101011010

$E(R_{13})$	=	000011110001011000000110100010101010101011110100
K_{14}	=	010111110100001110110111111100101110011100111010
$E(R_{13}) \oplus K_{14}$	=	010100000101010110110001011110000100110111001110
S-box outputs		01100100011110011001101011110001
$f(R_{13}, K_{14})$	=	10110111001100011000111001010101
$L_{15} = R_{14}$	=	11000010100011001001011000001101

$E(R_{14})$	=	111000000101010001011001010010101100000001011011
K_{15}	=	101111111001000110001101001111010011111100001010
$E(R_{14}) \oplus K_{15}$	=	010111111100010111010100011101111111111101010001
S-box outputs		10110010111010001000110100111100
$f(R_{14}, K_{15})$	=	01011011100000010010011101101110
$L_{16} = R_{15}$	=	01000011010000100011001000110100

$E(R_{15})$	=	001000000110101000000100000110100100000110101000
K_{16}	=	110010110011110110001011000011100001011111110101
$E(R_{15}) \oplus K_{16}$	=	111010110101011110001111000101000101011001011101
S-box outputs		10100111100000110010010000101001
$f(R_{15}, K_{16})$	=	11001000110000000100111110011000
R_{16}	=	00001010010011001101100110010101

Finally, applying IP^{-1} to L_{16}, R_{16}, we obtain the ciphertext, which (in hexadecimal form) is:

```
85E813540F0AB405.
```

3.3 The DES Controversy

When **DES** was proposed as a standard, there was considerable criticism. One objection to **DES** concerned the S-boxes. All computations in **DES**, with the exception of the S-boxes, are *linear*, e.g., computing the exclusive-or of two outputs is the same as forming the exclusive-or of two inputs and then computing the output. The S-boxes, being the non-linear component of the cryptosystem, are vital to its security (We saw in Chapter 1 how linear cryptosystems, such as the Hill Cipher, could easily be cryptanalyzed by a known plaintext attack.) However, the design criteria of the S-boxes are not completely known. Several people have suggested that the S-boxes might contain hidden "trapdoors" which would allow the National Security Agency to decrypt messages while maintaining that **DES** is "secure." It is, of course, impossible to disprove such an assertion, but no evidence has come to light that indicates that trap-doors in **DES** do in fact exist.

In 1976, the National Security Agency (NSA) asserted that the following properties of the S-boxes are design criteria:

P0 Each row of each S-box is a permutation of the integers $0, \ldots, 15$.

P1 No S-box is a linear or affine function of its inputs.

P2 Changing one input bit to an S-box causes at least two output bits to change.

P3 For any S-box and any input x, $S(x)$ and $S(x \oplus 001100)$ differ in at least two bits (here x is a bitstring of length 6).

Two other properties of the S-boxes were designated as "caused by design criteria" by NSA.

P4 For any S-box, for any input x, and for $e, f \in \{0,1\}$, $S(x) \neq S(x \oplus 11ef00)$.

P5 For any S-box, if one input bit is fixed, and we look at the value of one fixed output bit, the number of inputs for which this output bit equals 0 will be "close to" the number of inputs for which the output bit equals 1. (Note that if we fix the value of either the first or sixth input bit, then 16 inputs will cause a particular output bit to equal 0 and 16 inputs will cause the output to equal 1. For the second through fifth input bits, this will not be true, but the resulting distribution will be "close to" uniform. More precisely, for any S-box, if the value of any input bit is fixed, then the number of inputs for which any fixed output bit has the value 0 (or 1) is always between 13 and 19.)

It is not publicly known if further design criteria were used in the construction of the S-boxes.

The most pertinent criticism of **DES** is that the size of the keyspace, 2^{56}, is too small to be really secure. Various special-purpose machines have been proposed for a known plaintext attack, which would essentially perform an exhaustive search for the key. That is given a 64-bit plaintext x and corresponding ciphertext y, every possible key would be tested until a key K such that $e_K(x) = y$ is found (and note that there may be more than one such key K).

As early as 1977, Diffie and Hellman suggested that one could build a VLSI chip which could test 10^6 keys per second. A machine with 10^6 keys could search the entire key space in about a day. They estimated that such a machine could be built for about \$20,000,000.

At the CRYPTO '93 Rump Session, Michael Wiener gave a very detailed design of a key search machine. The machine is based on a key search chip which is pipelined, so that 16 encryptions take place simultaneously. This chip can test 5×10^7 keys per second, and can be built using current technology for \$10.50 per chip. A frame consisting of 5760 chips can be built for \$100,000. This would allow a **DES** key to be found in about 1.5 days on average. A machine using 10 frames would cost \$1,000,000, but would reduce the average search time to about 3.5 hours.

3.4 DES in Practice

Even though the description of **DES** is quite lengthy, it can be implemented very efficiently, either in hardware or in software. The only arithmetic operations to be performed are exclusive-ors of bitstrings. The expansion function E, the S-boxes, the permutations IP and P, and the computation of $K_1, K_2, \ldots, K_{16}$ can all be done in constant time by table look-up (in software) or by hard-wiring them into a circuit.

Current hardware implementations can attain extremely fast encryption rates. Digital Equipment Corporation announced at CRYPTO '92 that they have fabricated a chip with 50K transistors that can encrypt at the rate of 1 Gbit/second using a clock rate of 250 MHz! The cost of this chip is about \$300. As of 1991, there were 45 hardware and firmware implementations of **DES** that had been validated by the National Bureau of Standards.

One very important application of **DES** is in banking transactions, using standards developed by the American Bankers Association. **DES** is used to encrypt personal identification numbers (PINs) and account transactions carried out by automated teller machines (ATMs). **DES** is also used by the Clearing House Interbank Payments System (CHIPS) to authenticate transactions involving over $\$1.5 \times 10^{12}$ per week.

DES is also widely used in government organizations, such as the Department of Energy, the Justice Department, and the Federal Reserve System.

3.4.1 DES Modes of Operation

Four modes of operation have been developed for **DES**: *electronic codebook mode* (ECB), *cipher feedback mode* (CFB), *cipher block chaining mode* (CBC)

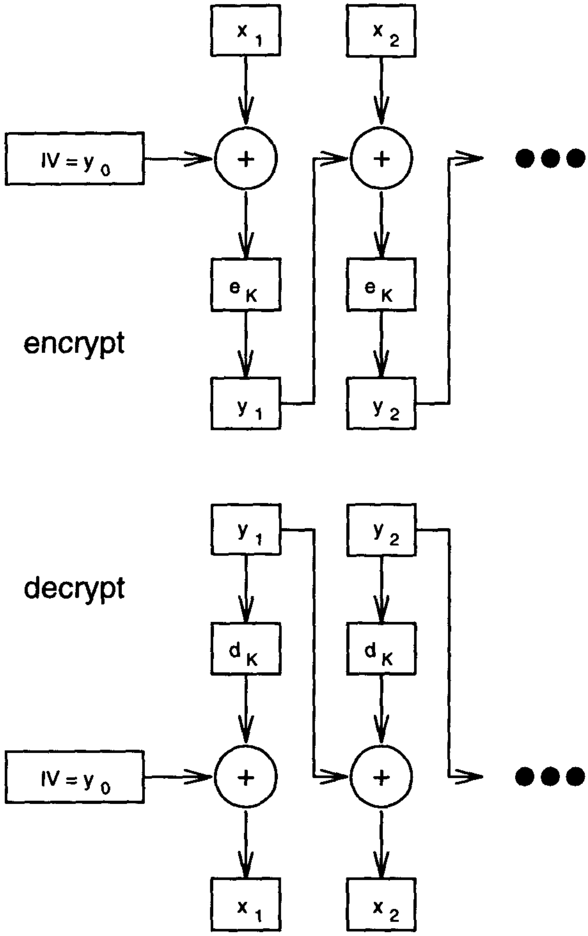

FIGURE 3.4
CBC mode

and *output feedback mode* (OFB).

ECB mode corresponds to the usual use of a block cipher: given a sequence $x_1x_2\dots$ of 64-bit plaintext blocks, each x_i is encrypted with the same key K, producing a string of ciphertext blocks, $y_1y_2\dots$.

In CBC mode, each ciphertext block y_i is x-ored with the next plaintext block x_{i+1} before being encrypted with the key K. More formally, we start with a 64-bit initialization vector IV, and define $y_0 = \mathrm{IV}$. Then we construct $y_1, y_2, \dots$ from the rule $y_i = e_K(y_{i-1} \oplus x_i)$, $i \geq 1$. The use of CBC mode is depicted in Figure 3.4.

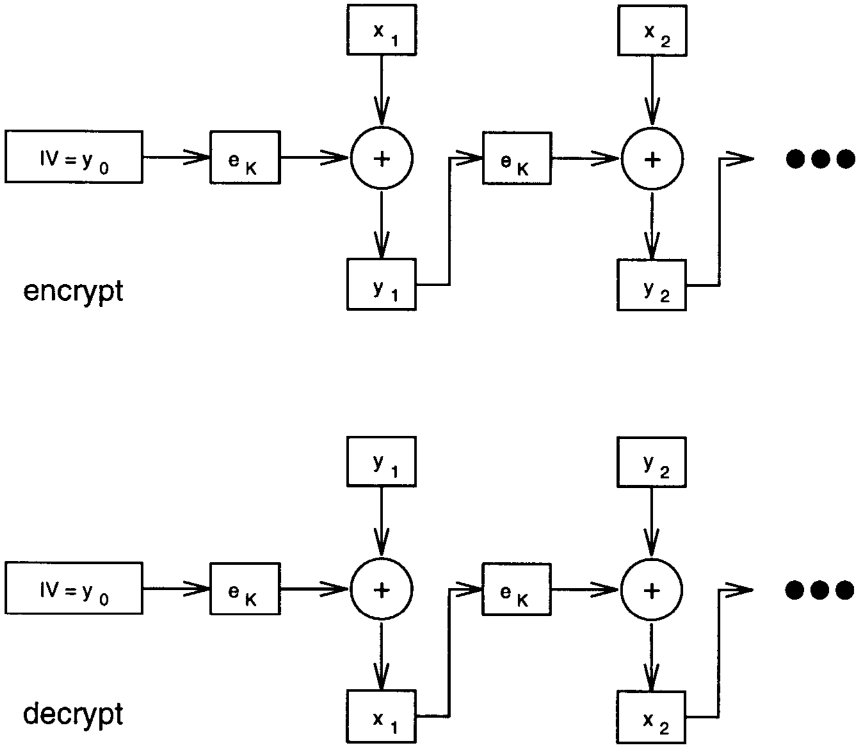

FIGURE 3.5
CFB mode

In OFB and CFB modes, a keystream is generated which is then x-ored with the plaintext (i.e., it operates as a stream cipher, cf. Section 1.1.7). OFB is actually a synchronous stream cipher: the keystream is produced by repeatedly encrypting a 64-bit initialization vector, IV. We define $z_0 = \text{IV}$, and then compute the keystream $z_1 z_2 \ldots$ from the rule $z_i = e_K(z_{i-1})$, $i \geq 1$. The plaintext sequence $x_1 x_2 \ldots$ is then encrypted by computing $y_i = x_i \oplus z_i$, $i \geq 1$.

In CFB mode, we start with $y_0 = \text{IV}$ (a 64-bit initialization vector) and we produce the keystream element z_i by encrypting the previous ciphertext block. That is, $z_i = e_K(y_{i-1})$, $i \geq 1$. As in OFB mode, $y_i = x_i \oplus z_i$, $i \geq 1$. The use of CFB is depicted in Figure 3.5 (note that the **DES** encryption function e_K is used for both encryption and decryption in CFB and OFB modes).

There are also variations of OFB and CFB mode called k-bit feedback modes $(1 \leq k \leq 64)$. We have described the 64-bit feedback modes here. 1-bit and 8-bit feedback modes are often used in practice for encrypting data one bit (or byte) at a time.

The four modes of operation have different advantages and disadvantages. In ECB and OFB modes, changing one 64-bit plaintext block, x_i, causes the corresponding ciphertext block, y_i, to be altered, but other ciphertext blocks are not affected. In some situations this might be a desirable property. For example, OFB mode is often used to encrypt satellite transmissions.

On the other hand, if a plaintext block x_i is changed in CBC and CFB modes, then y_i and all subsequent ciphertext blocks will be affected. This property means that CBC and CFB modes are useful for purposes of authentication. More specifically, these modes can be used to produce a *message authentication code*, or MAC. The MAC is appended to a sequence of plaintext blocks, and is used to convince Bob that the given sequence of plaintext originated with Alice and was not tampered with by Oscar. Thus the MAC guarantees the integrity (or authenticity) of a message (but it does not provide secrecy, of course).

We will describe how CBC mode is used to produce a MAC. We begin with the initialization vector IV consisting of all zeroes. Then construct the ciphertext blocks $y_1, \ldots, y_n$ with key K, using CBC mode. Finally, define the MAC to be y_n. Then Alice transmits the sequence of plaintext blocks, $x_1 \ldots x_n$, along with the MAC. When Bob receives $x_1 \ldots x_n$, he can reconstruct $y_1, \ldots, y_n$ using the (secret) key K, and verify that y_n is the same as the MAC that he received.

Note that Oscar cannot produce a valid MAC since he does not know the key K being used by Alice and Bob. Further, if Oscar intercepts a sequence of plaintext blocks $x_1 \ldots x_n$, and changes one or more of them, then it is highly unlikely that Ocsar can change the MAC so that it will be accepted by Oscar.

It is often desirable to combine authenticity and secrecy. This could be done as follows: Alice first uses key K_1 to produce a MAC for $x_1 \ldots x_n$. Then she defines x_{n+1} to be the MAC, and she encrypts the sequence $x_1 \ldots x_{n+1}$ using a second key, K_2, yielding $y_1 \ldots y_{n+1}$. When Bob receives $y_1 \ldots y_{n+1}$, he first decrypts (using K_2) and then checks that x_{n+1} is the MAC for $x_1 \ldots x_n$ using K_1.

Alternatively, Alice could use K_1 to encrypt $x_1 \ldots x_n$, obtaining $y_1 \ldots y_n$, and then use K_2 to produce a MAC y_{n+1} for $y_1 \ldots y_n$. Bob would use K_2 to verify the MAC, and then use K_1 to decrypt $y_1 \ldots y_n$.

3.5 A Time-memory Trade-off

In this section, we describe an interesting time-memory tradeoff for a chosen plaintext attack. Recall that in a chosen plaintext attack, Oscar obtains a plaintext-ciphertext pair produced using the (unknown) key K. So Oscar has x and y, where $y = e_K(x)$, and he wants to determine K.

A feature of this time-memory trade-off is that it does not depend on the "structure" of **DES** in any way. The only aspects of **DES** that are relevant to the attack are that plaintexts and ciphertexts have 64 bits, while keys have 56 bits.

FIGURE 3.6
Computation of $X(i, j)$

$$\begin{array}{|ccccccc|}
X(1,0) & \xrightarrow{g} & X(1,1) & \xrightarrow{g} & \dots & \xrightarrow{g} & X(1,t) \\
X(2,0) & \xrightarrow{g} & X(2,1) & \xrightarrow{g} & \dots & \xrightarrow{g} & X(2,t) \\
\vdots & & & & & & \vdots \\
X(m,0) & \xrightarrow{g} & X(m,1) & \xrightarrow{g} & \dots & \xrightarrow{g} & X(m,t)
\end{array}$$

We have already discussed the idea of exhaustive search: given a plaintext-ciphertext pair, try all 2^{56} possible keys. This requires no memory but, on average, 2^{55} keys will be tried before the correct one is found. On the other hand, for a given plaintext x, Oscar could precompute $y_K = e_K(x)$ for all 2^{56} keys K, and construct a table of ordered pairs (y_K, K), sorted by their first coordinates. At a later time, when Oscar obtains the ciphertext y which is an encryption of plaintext x, he looks up the value y in the table, immediately obtaining the key K. Now the actual determination of the key requires only constant time, but we have a large memory requirement and a large precomputation time. (Note that this approach would yield no advantage in total computation time if only one key is to be found, since constructing the table takes at least as much time as an exhaustive search. The advantage occurs when several keys are to be found over a period of time, since the same table can be used in each case.)

The time-memory trade-off combines provides a smaller computation time than exhaustive search with a smaller memory requirement than table look-up. The algorithm can be described in terms of two parameters m and t, which are positive integers. The algorithm requires a *reduction function* R which reduces a bitstring of length 64 to one of length 56. (R might just discard eight of the 64 bits, for example.) Let x be a fixed plaintext string of length 64. Define the function $g(K_0) = R(e_{K_0}(x))$ for a bitstring K_0 of length 56. Note that g is a function that maps 56 bits to 56 bits.

In the pre-processing stage, Oscar chooses m random bitstrings of length 56, denoted $X(i, 0)$, $1 \leq i \leq m$. Oscar computes $X(i, j)$ for $1 \leq j \leq t$ according to the recurrence relation $X(i, j) = g(X(i, j-1))$, $1 \leq i \leq m$, $1 \leq j \leq t$, as indicated in Figure 3.6.

Then Oscar constructs a table of ordered pairs $T = (X(i,t), X(i,0))$, sorted by their first coordinate (i.e., only the first and last columns of X are stored).

At a later time, Oscar obtains a ciphertext y which is an encryption of the chosen plaintext x (as before). He again wants to determine K. He is going to determine if K is in the first t columns of the array X, but he will do this by looking only at the table T.

Suppose that $K = X(i, t-j)$ for some j, $1 \leq j \leq t$ (i.e., suppose that K is in the first t columns of X). Then it is clear that $g^j(K) = X(i,t)$, where g^j denotes

FIGURE 3.7
DES time-memory trade-off

```
1.  compute y_1 = R(y)
2.  for j = 1 to t do
3.      if y_j = X(i,t) for some i then
4.          compute X(i,t-j) from X(i,0) by iterating the g
            function t-j times
5.          if y = e_{X(i,t-j)}(x) then
                set K = X(i,t-j) and QUIT
7.      compute y_{j+1} = g(y_j)
```

the function obtained by iterating g, j times. Now, observe that

$$\begin{aligned} g^j(K) &= g^{j-1}(g(K)) \\ &= g^{j-1}(R(e_K(x)) \\ &= g^{j-1}(R(y)). \end{aligned}$$

Suppose we compute y_j, $1 \leq j \leq t$, from the recurrence relation

$$y_j = \begin{cases} R(y) & \text{if } j = 1 \\ g(y_{j-1}) & \text{if } 2 \leq j \leq t. \end{cases}$$

Then it follows that $y_j = X(i,t)$ if $K = X(i,t-j)$. However, note that $y_j = X(i,t)$ is not sufficient to ensure that $K = X(i,t-j)$. This is because the reduction function R is not an injection: The domain of R has cardinality 2^{64} and the range of R has cardinality 2^{56}, so, on average, there are $2^8 = 256$ pre-images of any given bitstring of length 56. So we need to check whether $y = e_{X(i,t-j)}(x)$, to see if $X(i,t-j)$ is indeed the key. We did not store the value $X(i,t-j)$, but we can easily re-compute it from $X(i,0)$ by iterating the g function $t-j$ times.

Oscar proceeds according to the algorithm presented in Figure 3.7.

By analyzing the probability of success for the algorithm, it can be shown that if $mt^2 \approx N = 2^{56}$, then the probability that $K = X(i,t-j)$ for some i, j is about $0.8mt/N$. The factor 0.8 accounts for the fact that the numbers $X(i,t)$ may not all be distinct. It is suggested that one should take $m \approx t \approx N^{1/3}$ and construct about $N^{1/3}$ tables, each using a different reduction function R. If this is done, the memory requirement is $112 \times N^{2/3}$ bits (since we need to store $2 \times N^{2/3}$ integers, each of which has 56 bits). The precomputation time is easily seen to be $O(N)$.

The running time is a bit more dificult to analyze. First, note that step 3 can be implemented to run in (expected) constant time (using hash coding) or (worstcase) time $O(\log m)$ using a binary search. If step 3 is never satisfied (i.e., the search fails), then the running time is $O(N^{2/3})$. A more detailed analysis shows that even when the running time of steps 4 and 5 is taken into account, the expected running time increases by only a constant factor.

3.6 Differential Cryptanalysis

One very well-known attack on **DES** is the method of "differential cryptanalysis" introduced by Biham and Shamir. This is a chosen-plaintext attack. Although it does not provide a practical method of breaking the usual 16-round **DES**, it does succeed in breaking **DES** if the number of rounds of encryption is reduced. For instance, 8-round **DES** can be broken in only a couple of minutes on a small personal computer.

We will now describe the basic ideas used in this technique. For the purposes of this attack, we can ignore the initial permutation IP and its inverse (it has no effect on cryptanalysis). As mentioned above, we consider **DES** restricted to n rounds, for various values of $n \leq 16$. So, in this setting, we will regard L_0R_0 as the plaintext, and L_nR_n as the ciphertext, in an n-round **DES**. (Note also that we are not inverting L_nR_n.)

Differential cryptanalysis involves comparing the x-or (exclusive-or) of two plaintexts to the x-or of the corresponding two ciphertexts. In general, we will be looking at two plaintexts L_0R_0 and $L_0^*R_0^*$ with a specified x-or value $L_0'R_0' = L_0R_0 \oplus L_0^*R_0^*$. Throughout this discussion, we will use prime markings ($'$) to indicate the x-or of two bitstrings.

DEFINITION 3.1 *Let S_j be a particular S-box ($1 \leq j \leq 8$). Consider an (ordered) pair of bitstrings of length six, say (B_j, B_j^*). We say that the **input x-or** (of S_j) is $B_j \oplus B_j^*$ and the **output x-or** (of S_j) is $S_j(B_j) \oplus S_j(B_j^*)$.*

Note that an input x-or is a bitstring of length six and an output x-or is a bitstring of length four.

DEFINITION 3.2 *For any $B_j' \in (\mathbb{Z}_2)^6$, define the set $\Delta(B_j')$ to consist of the ordered pairs (B_j, B_j^*) having input x-or B_j'.*

It is easy to see that any set $\Delta(B_j')$ contains $2^6 = 64$ pairs, and that

$$\Delta(B_j') = \{(B_j, B_j \oplus B_j') : B_j \in (\mathbb{Z}_2)^6\}.$$

For each pair in $\Delta(B_j')$, we can compute the output x-or of S_j and tabulate the resulting distribution. There are 64 output x-ors, which are distributed among

$2^4 = 16$ possible values. The non-uniformity of these distributions will be the basis for the attack.

Example 3.1
Suppose we consider the first S-box, S_1, and the input x-or 110100. Then

$$\Delta(110100) = \{(000000, 110100), (000001, 110101), \ldots, (111111, 001011)\}.$$

For each ordered pair in the set $\Delta(110100)$, we compute output x-or of S_1. For example, $S_1(000000) = E_{16} = 1110$ and $S_1(110100) = 9_{16} = 1001$, so the output x-or for the pair (000000, 110100) is 0111.

If this is done for all 64 pairs in $\Delta(110100)$, then the following distribution of output x-ors is obtained:

0000	0001	0010	0011	0100	0101	0110	0111
0	8	16	6	2	0	0	12

1000	1001	1010	1011	1100	1101	1110	1111
6	0	0	0	0	8	0	6

□

In Example 3.1, only eight of the 16 possible output x-ors actually occur. This particular example has a very non-uniform distribution. In general, if we fix an S-box S_j and an input x-or B'_j, then on average, it turns out that about 75 – 80% of the possible output x-ors actually occur.

It will be convenient to have some notation to describe these distributions and how they arise, so we make the following definitions.

DEFINITION 3.3 *For $1 \leq j \leq 8$, and for bitstrings B'_j of length six and C'_j of length four, define*

$$IN_j(B'_j, C'_j) = \{B_j \in (\mathbb{Z}_2)^6 : S_j(B_j) \oplus S_j(B_j \oplus B'_j) = C'_j\}$$

and

$$N_j(B'_j, C'_j) = |IN_j(B'_j, C'_j)|.$$

$N_j(B'_j, C'_j)$ counts the number of pairs with input x-or equal to B'_j which have output x-or equal to C'_j for the S-box S_j. The actual pairs having the specified input x-ors and giving rise to the specified output x-ors can be obtained from the set $IN_j(B'_j, C'_j)$. Observe that this set can be partitioned into $N_j(B'_j, C'_j)/2$ pairs, each of which has (input) x-or equal to B'_j.

Note that the distribution tabulated in Example 3.1 consists of the values $N_1(110100, C'_1)$, $C'_1 \in (\mathbb{Z}_2)^4$. The sets $IN_1(110100, C'_1)$ are listed in Figure 3.8.

FIGURE 3.8
Possible inputs with input x-or 110100

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

For each of the eight S-boxes, there are 64 possible input x-ors. Thus, there are 512 distributions which can be computed. These could easily be tabulated by computer.

Recall that the input to the S-boxes in round i is formed as $B = E \oplus J$, where $E = \mathrm{E}(R_{i-1})$ is the expansion of R_{i-1} and $J = K_i$ consists of the key bits for round i. Now, the input x-or (for all eight S-boxes) can be computed as follows:

$$\begin{aligned} B \oplus B^* &= (E \oplus J) \oplus (E^* \oplus J) \\ &= E \oplus E^*. \end{aligned}$$

It is very important to observe that the input x-or does not depend on the key bits

J. (However, the output x-or certainly does depend on these key bits.)

We will write each of B, E and J as the concatenation of eight 6-bit strings:

$$B = B_1B_2B_3B_4B_5B_6B_7B_8$$
$$E = E_1E_2E_3E_4E_5E_6E_7E_8$$
$$J = J_1J_2J_3J_4J_5J_6J_7J_8,$$

and we write B^*, E^*, J^* in a similar way. Let us suppose for the moment that we know the values E_j and E_j^* for some j, $1 \leq j \leq 8$, and the value of the output x-or for S_j, $C_j' = S_j(B_j) \oplus S_j(B_j^*)$. Then it must be the case that

$$E_j \oplus J_j \in IN_j(E_j', C_j'),$$

where $E_j' = E_j \oplus E_j^*$.

Suppose we define a set $test_j$ as follows:

DEFINITION 3.4 *Suppose E_j and E_j^* are bitstrings of length six, and C_j' is a bitstring of length four. Define*

$$test_j(E_j, E_j^*, C_j') = \{B_j \oplus E_j : B_j \in IN_j(E_j', C_j')\},$$

where $E_j' = E_j \oplus E_j^*$.

That is, we take the x-or of E_j with every element of the set $IN_j(E_j', C_j')$.

The following result is an immediate consequence of the discussion above.

THEOREM 3.1
Suppose E_j and E_j^* are two inputs to the S-box S_j, and the output x-or for S_j is C_j'. Denote $E_j' = E_j \oplus E_j^*$. Then the key bits J_j occur in the set $test_j(E_j, E_j^*, C_j')$.

Observe that there will be exactly $N_j(E_j', C_j')$ bitstrings of length six in the set $test_j(E_j, E_j^*, C_j')$; the correct value of J_j must be one of these possibilities.

Example 3.2
Suppose $E_1 = 000001$, $E_1^* = 110101$ and $C_1' = 1101$. Since $N_1(110100, 1101) = 8$, there will be exactly eight bitstrings in the set $test_1(000001, 110101, 1101)$. From Figure 3.8, we see that

$$IN_1(110100, 1101) = \{000110, 010000, 010110, 011100, 100010, 100100, 101000, 110010\}.$$

Hence,

$$test_1(000001, 110101, 1101) = \{000111, 010001, 010111, 011101, 100011, 100101, 101001, 110011\}. \quad \square$$

If we have a second such triple E_1, E_1^*, C_1', then we can obtain a second set $test_1$ of possible values for the keybits in J_1. The true value of J_1 must be in the intersection of both sets. If we have several such triples, then we can quickly determine the key bits in J_1. One straightforward way to do this is to maintain an array of 64 counters, representing the 64 possibilities for the six key bits in J_1. A counter is incremented every time the corresponding key bits occur in a set $test_1$ for a particular triple. Given t triples, we hope to find a unique counter which has the value t; this will correspond to the true value of the keybits in J_1.

3.6.1 An Attack on a 3-round DES

Let's now see how the ideas of the previous section can be applied in a chosen plaintext attack of a 3-round **DES**. We will begin with a pair of plaintexts and corresponding ciphertexts: L_0R_0, $L_0^*R_0^*$, L_3R_3 and $L_3^*R_3^*$. We can express R_3 as follows:

$$\begin{aligned} R_3 &= L_2 \oplus f(R_2, K_3) \\ &= R_1 \oplus f(R_2, K_3) \\ &= L_0 \oplus f(R_0, K_1) \oplus f(R_2, K_3). \end{aligned}$$

R_3^* can be expressed in a similar way, and hence

$$R_3' = L_0' \oplus f(R_0, K_1) \oplus f(R_0^*, K_1) \oplus f(R_2, K_3) \oplus f(R_2^*, K_3).$$

Now, suppose we have chosen the plaintexts so that $R_0 = R_0^*$, i.e., so that

$$R_0' = 00\ldots0.$$

Then $f(R_0, K_1) = f(R_0^*, K_1)$ and so

$$R_3' = L_0' \oplus f(R_2, K_3) \oplus f(R_2^*, K_3).$$

At this point, R_3' is known since it can be computed from the two ciphertexts, and L_0' is known since it can be computed from the two plaintexts. This means that we can compute $f(R_2, K_3) \oplus f(R_2^*, K_3)$ from the equation

$$f(R_2, K_3) \oplus f(R_2^*, K_3) = R_3' \oplus L_0'.$$

Now, $f(R_2, K_3) = \mathrm{P}(C)$ and $f(R_2^*, K_3) = \mathrm{P}(C^*)$, where C and C^*, respectively, denote the two outputs of the eight S-boxes (recall that P is a fixed, publicly known permutation). Hence,

$$\mathrm{P}(C) \oplus \mathrm{P}(C^*) = R_3' \oplus L_0',$$

FIGURE 3.9
Differential attack on 3-round DES

	Input: L_0R_0, $L_0^*R_0^*$, L_3R_3 and $L_3^*R_3^*$, where $R_0 = R_0^*$
1.	compute $C' = \mathrm{P}^{-1}(R_3' \oplus L_0')$
2.	compute $E = \mathrm{E}(L_3)$ and $E^* = \mathrm{E}(L_3^*)$
3.	**for** $j = 1$ **to** 8 **do**
	compute $test_j(E_j, E_j^*, C_j')$

and consequently

$$C' = C \oplus C^* = \mathrm{P}^{-1}(R_3' \oplus L_0'). \tag{3.1}$$

This is the output x-or for the eight S-boxes in round three.

Now, $R_2 = L_3$ and $R_2^* = L_3^*$ are also known (they are part of the ciphertexts). Hence, we can compute

$$E = \mathrm{E}(L_3) \tag{3.2}$$

and

$$E^* = \mathrm{E}(L_3^*) \tag{3.3}$$

using the publicly known expansion function E. These are the inputs to the S-boxes for round three. So, we now know E, E^*, and C' for the third round, and we can proceed, as in the previous section, to construct the sets $test_1$, ..., $test_8$ of possible values for the key bits in $J_1, \ldots, J_8$.

A pseudo-code description of this algorithm is given in Figure 3.9. The attack will use several such triples E, E^*, C'. We set up eight arrays of counters, and thereby determine the 48 bits in K_3, the key for the third round. The 56 bits in the key can then be computed by an exhaustive search of the $2^8 = 256$ possibilities for the remaining eight key bits.

Let's look at an example to illustrate.

Example 3.3

Suppose we have the following three pairs of plaintexts and ciphertexts, where the plaintexts have the specified x-ors, that are encrypted using the same key. We use a hexadecimal representation, for brevity:

plaintext	ciphertext
748502CD38451097	03C70306D8A09F10
3874756438451097	78560A0960E6D4CB
486911026ACDFF31	45FA285BE5ADC730
375BD31F6ACDFF31	134F7915AC253457
357418DA013FEC86	D8A31B2F28BBC5CF
12549847013FEC86	0F317AC2B23CB944

From the first pair, we compute the S-box inputs (for round 3) from Equations (3.2) and (3.3). They are:

$$E = 000000000111111000001110100000000110100000001100$$

$$E^* = 101111110000001010101100000001010100000001010010.$$

The S-box output x-or is calculated using Equation (3.1) to be:

$$C' = 10010110010111010101101101100111.$$

From the second pair, we compute the S-box inputs to be

$$E = 101000000101111111111010000001010100000001011110110$$

$$E^* = 100010100110101001011110101111110010100010101010$$

and the S-box output x-or is

$$C' = 10011100100111000001111101010110.$$

From the third pair, the S-box inputs are

$$E = 111011110001010100000110100011110110100101011111$$

$$E^* = 000001011110100110100010101111110101011000000100$$

and the S-box output x-or is

$$C' = 11010101011101011101101100101011.$$

Next, we tabulate the values in the eight counter arrays for each of the three pairs. We illustrate the procedure with the counter array for J_1 from the first pair. In this pair, we have $E'_1 = 101111$ and $C'_1 = 1001$. The set

$$IN_1(101111, 1001) = \{000000, 000111, 101000, 101111\}.$$

Since $E_1 = 000000$, we have that

$$J_1 \in test_1(000000, 101111, 1001) = \{000000, 000111, 101000, 101111\}.$$

Hence, we increment the values $0, 7, 40$, and 47 in the counter array for J_1.

The final tabulations are now presented. If we think of a bit-string of length six as being the binary representation of an integer between 0 and 63, then the 64 values correspond to the counts of $0, 1, \ldots, 63$. The counter arrays are as follows:

J_1															
1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

J_2															
0	0	0	1	0	3	0	0	1	0	0	1	0	0	0	0
0	1	0	0	0	2	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0
0	0	1	1	0	0	0	0	1	0	1	0	2	0	0	0

J_3															
0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
0	0	0	3	0	0	0	0	0	0	0	0	0	0	1	1
0	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0

J_4															
3	1	0	0	0	0	0	0	0	0	2	2	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	1
1	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	2	1

J_5															
0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	0	0	0	0	1	0	0	0	0	2	0

J_6															
1	0	0	1	1	0	0	3	0	0	0	0	1	0	0	1
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0

J_7															
0	0	2	1	0	1	0	3	0	0	0	1	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	2	0	0	0	2	0	0	0	0	1	2	1	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

J_8															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1
0	3	0	0	0	0	1	0	0	0	0	0	0	0	0	0

In each of the eight counter arrays, there is a unique counter having the value 3. The positions of these counters determine the key bits in $J_1, \ldots, J_8$. These positions are (respectively): $47, 5, 19, 0, 24, 7, 7, 49$. Converting these integers to binary, we obtain $J_1, \ldots, J_8$:

$$
\begin{aligned}
J_1 &= 101111\\
J_2 &= 000101\\
J_3 &= 010011\\
J_4 &= 000000\\
J_5 &= 011000\\
J_6 &= 000111\\
J_7 &= 000111\\
J_8 &= 110001.
\end{aligned}
$$

We can now construct 48 bits of the key, by looking at the key schedule for round 3. It follows that K has the form

```
0001101   0110001   01?01?0   1?00100
0101001   0000??0   111?11?   ?100011
```

where parity bits are omitted and "?" denotes an unknown key bit. The complete key (in hexadecimal, including parity bits), is:

`1A624C89520DEC46.`

□

3.6.2 An Attack on a 6-round DES

We now describe an extension of these ideas to a probabilistic attack on a 6-round **DES**. The idea is to carefully choose a pair of plaintexts with a specified x-or, and then to determine the probabilities of a specified sequence of x-ors through the rounds of encryption. We need to define an important concept now.

DEFINITION 3.5 *Let $n \geq 1$ be an integer. An n-**round characteristic** is a list of the form*

$$L_0', R_0', L_1', R_1', p_1, \ldots, L_n', R_n', p_n,$$

which satisfies the following properties:

1. *$L_i' = R_{i-1}'$ for $1 \leq i \leq n$.*
2. *Let $1 \leq i \leq n$, and let L_{i-1}, R_{i-1} and L_{i-1}^*, R_{i-1}^* be chosen such that $L_{i-1} \oplus L_{i-1}^* = L_{i-1}'$ and $R_{i-1} \oplus R_{i-1}^* = R_{i-1}'$. Suppose L_i, R_i and L_i^*, R_i^* are computed by applying one round of* **DES** *encryption. Then the probability that $L_i \oplus L_i^* = L_i'$ and $R_i \oplus R_i^* = R_i'$ is precisely p_i. (Note that this probability is computed over all possible 48-tuples $J = J_1 \ldots J_8$.)*

*The **probability** of the characteristic is defined to be the product $p = p_1 \times \ldots \times p_n$.*

REMARK Suppose we choose L_0, R_0 and L_0^*, R_0^* so that $L_0 \oplus L_0^* = L_0'$ and $R_0 \oplus R_0^* = R_0'$ and we apply n rounds of **DES** encryption, obtaining $L_1, \ldots, L_n$ and $R_1, \ldots, R_n$. Then we cannot claim that the probability that $L_i \oplus L_i^* = L_i'$ and $R_i \oplus R_i^* = R_i'$ for all i $(1 \leq i \leq n)$ is $p_1 \times \ldots \times p_n$. This is because the 48-tuples in the key schedule $K_1, \ldots, K_n$, are not mutually independent. (If these n 48-tuples were chosen independently at random, then the assertion would be true.) But we nevertheless expect $p_1 \times \ldots \times p_n$ to be a fairly accurate estimate of this probability.

We also need to recognize that the probabilities p_i in a characteristic are defined with respect to an arbitrary (but fixed) pair of plaintexts having a specified x-or, where the 48 key bits for one round of **DES** encryption vary over all 2^{48} possibilities. However, a cryptanalyst is attempting to determine a fixed (but unknown) key. He is going to choose plaintexts at random (such that they have specified x-ors), hoping that the probabilities that the x-ors during the n rounds of encryption agree with the x-ors specified in the characteristic are fairly close to $p_1, \ldots, p_n$, respectively. ∎

As a simple example, we present in Figure 3.10 a 1-round characteristic which was the basis of the attack on the 3-round **DES** (as before, we use hexadecimal representations). We depict another 1-round characteristic in Figure 3.11.

Let's look at the characteristic in Figure 3.11 in more detail. When $f(R_0, K_1)$ and $f(R_0^*, K_1)$ are computed, the first step is to expand R_0 and R_0^*. The resulting

FIGURE 3.10
A 1-round characteristic

L'_0	$=$	anything	R'_0	$=$	00000000_{16}	
L'_1	$=$	00000000_{16}	R'_1	$=$	L'_0	$p = 1$

FIGURE 3.11
Another 1-round characteristic

L'_0	$=$	00000000_{16}	R'_0	$=$	60000000_{16}	
L'_1	$=$	60000000_{16}	R'_1	$=$	00808200_{16}	$p = 14/64$

x-or of the two expansions is

$$001100\ldots0.$$

So the input x-or to S_1 is 001100 and the input x-ors for the other seven S-boxes are all 000000. The output x-ors for S_2 through S_8 will all be 0000. The output x-or for S_1 will be 1110 with probability $14/64$ (since it can be computed that $N_1(001100, 1110) = 14$). So we obtain

$$C' = 11100000000000000000000000000000$$

with probability $14/64$. Applying P, we get

$$\mathrm{P}(C) \oplus \mathrm{P}(C^*) = 00000000100000001000001000000000,$$

which in hexadecimal is 00808200_{16}. When this is x-ored with L'_0, we get the specified R'_1 with probability $14/64$. Of course $L'_1 = R'_0$ always.

The attack on the 6-round **DES** is based on the 3-round characteristic given in Figure 3.12. In the 6-round attack, we will start with L_0R_0, $L_0^*R_0^*$, L_6R_6 and $L_6^*R_6^*$, where we have chosen the plaintexts so that $L'_0 = 40080000_{16}$ and

FIGURE 3.12
A 3-round characteristic

L'_0	$=$	40080000_{16}	R'_0	$=$	04000000_{16}	
L'_1	$=$	04000000_{16}	R'_1	$=$	00000000_{16}	$p = 1/4$
L'_2	$=$	00000000_{16}	R'_2	$=$	04000000_{16}	$p = 1$
L'_3	$=$	04000000_{16}	R'_3	$=$	40080000_{16}	$p = 1/4$

$R'_0 = 04000000_{16}$. We can express R_6 as follows:

$$\begin{aligned} R_6 &= L_5 \oplus f(R_5, K_6) \\ &= R_4 \oplus f(R_5, K_6) \\ &= L_3 \oplus f(R_3, K_4) \oplus f(R_5, K_6). \end{aligned}$$

R_6^* can be expressed in a similar way, and hence we get

$$R'_6 = L'_3 \oplus f(R_3, K_4) \oplus f(R_3^*, K_4) \oplus f(R_5, K_6) \oplus f(R_5^*, K_6). \tag{3.4}$$

(Note the similarity with the 3-round attack.)

R'_6 is known. From the characteristic, we estimate that $L'_3 = 04000000_{16}$ and $R'_3 = 40080000_{16}$ with probability $1/16$. If this is in fact the case, then the input x-or for the S-boxes in round 4 can be computed by the expansion function to be:

$$001000000000000001010000\ldots0.$$

The input x-ors for S_2, S_5, S_6, S_7 and S_8 are all 000000, and hence the output x-ors are 0000 for these five S-boxes in round 4. This means that we can compute the output x-ors of these five S-boxes in round 6 from Equation (3.4). So, suppose we compute

$$C'_1C'_2C'_3C'_4C'_5C'_6C'_7C'_8 = \mathrm{P}^{-1}(R'_6 \oplus 04000000_{16})$$

where each C_i is a bitstring of length four. Then with probability $1/16$, it will be the case that C'_2, C'_5, C'_6, C'_7 and C'_8 are respectively the output x-ors of S_2, S_5, S_6, S_7 and S_8 in round 6. The inputs to these S-boxes in round 6 can be computed to be E_2, E_5, E_6, E_7 and E_8, and E_2^*, E_5^*, E_6^*, E_7^* and E_8^*, where

$$E_1E_2E_3E_4E_5E_6E_7E_8 = \mathrm{E}(R_5) = \mathrm{E}(L_6)$$

and

$$E_1^*E_2^*E_3^*E_4^*E_5^*E_6^*E_7^*E_8^* = \mathrm{E}(R_5^*) = \mathrm{E}(L_6^*)$$

can be computed from the ciphertexts, as indicated in Figure 3.13.

We would like to determine the 30 key bits in J_2, J_5, J_6, J_7 and J_8 as we did in the 3-round attack. The problem is that the hypothesized output x-or for round 6 is correct only with probability $1/16$. So $15/16$ of the time we will obtain random garbage rather than possible key bits. We somehow need to be able to determine the correct key from the given data, $15/16$ of which is incorrect. This might not seem very promising, but fortunately our prospects are not as bleak as they initially appear.

DEFINITION 3.6 *Suppose $L_0 \oplus L_0^* = L'_0$ and $R_0 \oplus R_0^* = R'_0$. We say that the pair of plaintexts L_0R_0 and $L_0^*R_0^*$ is* ***right pair*** *with respect to a characteristic if $L_i \oplus L_i^* = L'_i$ and $R_i \oplus R_i^* = R'_i$ for all i, $1 \le i \le n$. The pair is a defined to be* ***wrong pair****, otherwise.*

FIGURE 3.13
Differential attack on 6-round DES

Input: L_0R_0, $L_0^*R_0^*$, L_6R_6 and $L_6^*R_6^*$, where $L_0' = 40080000_{16}$ and $R_0' = 04000000_{16}$
1. compute $C' = \mathrm{P}^{-1}(R_6' \oplus 40080000_{16})$
2. compute $E = \mathrm{E}(L_6)$ and $E^* = \mathrm{E}(L_6^*)$
3. **for** $j \in \{2, 5, 6, 7, 8\}$ **do**
   compute $test_j(E_j, E_j^*, C_j')$.

We expect that about $1/16$ of our pairs are right pairs and the rest are wrong pairs with respect to our 3-round characteristic.

Our strategy is to compute E_j, E_j^*, and C_j', as described above, and then to determine $test_j(E_j, E_j^*, C_j')$, for $j = 2, 5, 6, 7, 8$. If we start with a right pair, then the correct key bits for each J_j will be included in the set $test_j$. If the pair is a wrong pair, then the value of C_j' will be incorrect, and it seems reasonable to hypothesize that each set $test_j$ will be essentially random.

We can often identify a wrong pair by this method: If $|test_j| = 0$, for any $j \in \{2, 5, 6, 7, 8\}$, then we necessarily have a wrong pair. Now, given a wrong pair, we might expect that the probability that $|test_j| = 0$ for a particular j is approximately $1/5$. This is a reasonable assumption since $N_j(E_j', C_j') = |test_j|$ and, as mentioned earlier, the probability that $N_j(E_j', C_j') = 0$ is approximately $1/5$. The probability that all five $test_j$'s have positive cardinality is estimated to be $.8^5 \approx .33$, so the probability that at least one $test_j$ has zero cardinality is about .67. So we expect to eliminate about 2/3 of the wrong pairs by this simple observation, which we call the *filtering operation*. The proportion of right pairs that remain after filtering is approximately $(1/16)/(1/3) = 3/16$.

Example 3.4

Suppose we have the following plaintext-ciphertext pair:

plaintext	ciphertext
86FA1C2B1F51D3BE	1E23ED7F2F553971
C6F21C2B1B51D3BE	296DE2B687AC6340

Observe that $L'_0 = 40080000_{16}$ and $R'_0 = 04000000_{16}$. The S-box inputs and outputs for round 6 are computed to be the following:

j	E_j	E^*_j	C'_j
2	111100	010010	1101
5	111101	111100	0001
6	011010	000101	0010
7	101111	010110	1100
8	111110	101100	1101

Then, the sets $test_j$ are as follows:

j	$test_j$
2	14, 15, 26, 30, 32, 33, 48, 52
5	
6	7, 24, 36, 41, 54, 59
7	
8	34, 35, 48, 49

We see that both $test_5$ and $test_7$ are empty sets, so this pair is a wrong pair and is discarded by the filtering operation. □

Now suppose that we have a pair such that $|test_j| > 0$ for $j = 2, 5, 6, 7, 8$, so that it survives the filtering operation. (Of course, we do not know if the pair is a right pair or a wrong pair.) We say that the bitstring $J_2J_5J_6J_7J_8$ of length 30 is *suggested* by the pair if $J_j \in test_j$ for $j = 2, 5, 6, 7, 8$. The number of suggested bitstrings is

$$\prod_{j \in \{2,5,6,7,8\}} |test_j|.$$

It is not unusual for the number of suggested bitstrings to be quite large (for example, greater than 80000).

Suppose we were to tabulate all the suggested bitstrings obtained from the N pairs that were not discarded by the filtering operation. For every right pair, the correct bitstring $J_2J_5J_6J_7J_8$ will be a suggested bitstring. This correct bitstring will be counted about $3N/16$ times. Incorrect bitstrings should occur much less often, since they will occur essentially at random and there are 2^{30} possibilities (a very large number).

It would get extremely unwieldy to tabulate all the suggested bitstrings, so we use an algorithm that requires less space and time. We can encode any $test_j$ as a vector T_j of length 64, where the ith coordinate of T_j is set to 1 (for $0 \leq i \leq 63$) if the bitstring of length six that is the binary representation of i is in the set $test_j$; and the ith coordinate is set to 0 otherwise (this is essentially the same as the counter array representation that we used in the 3-round attack).

For each remaining pair, construct these vectors as described above, and name them T_j^i, $j = 2,5,6,7,8$, $1 \leq i \leq N$. For $I \subseteq \{1,\ldots,N\}$, we say that I is *allowable* if for each $j \in \{2,5,6,7,8\}$, there is at least one coordinate equal to $|I|$ in the vector

$$\sum_{i \in I} T_j^i.$$

If the ith pair is a right pair for every $i \in I$, then the set I is allowable. Hence, we expect there to be an allowable set of size (approximately) $3N/16$, which we hope will suggest the correct key bits and no other. It is a simple matter to construct all the allowable sets I by means of a recursive algorithm.

Example 3.5

We did some computer runs to test this approach. A random sample of 120 pairs of plaintexts with the specified x-ors was generated, and these were encrypted using the same (random) key. We present the 120 pairs of ciphertexts and corresponding plaintexts in hexadecimal form in Table 3.1.

When we compute the allowable sets, we obtain n_i allowable sets of cardinality i, for the following values:

i	n_i
2	111
3	180
4	231
5	255
6	210
7	120
8	45
9	10
10	1

The unique allowable set of size 10 is

$$\{24, 29, 30, 48, 50, 52, 55, 83, 92, 118\}.$$

In fact, it does arise from the 10 right pairs. This allowable set suggests the correct key bits for J_2, J_5, J_6, J_7 and J_8 and no others. They are as follows:

$$J_2 = 011001$$

$$J_5 = 110000$$

$$J_6 = 001001$$

$$J_7 = 101010$$

FIGURE 3.14
Another 3-round characteristic

L_0'	=	00200008_{16}	R_0'	=	00000400_{16}	
L_1'	=	00000400_{16}	R_1'	=	00000000_{16}	$p = 1/4$
L_2'	=	00000000_{16}	R_2'	=	00000400_{16}	$p = 1$
L_3'	=	00000400_{16}	R_3'	=	00200008_{16}	$p = 1/4$

$$J_8 = 100011$$

Note that all the allowable sets of cardinality at least 6, and all but three of the allowable sets of cardinality 5, arise from right pairs, since $\binom{10}{5} = 252$ and $\binom{10}{i} = n_i$ for $6 \leq i \leq 10$.

This method yields 30 of the 56 key bits. By means of a different 3-round characteristic, presented in Figure 3.14, it is possible to compute 12 further key bits, namely those in J_1 and J_4. Now only 14 key bits remain unknown. Since $2^{14} = 16384$ is quite small, an exhaustive search can be used to determine the remaining 14 key bits.

The entire key (in hexadecimal, including parity-check bits) is:

```
34E9F71A20756231.
```

As mentioned above, the 120 pairs are given in Table 3.1. In the second column, a * denotes that a pair is a right pair, while a ** denotes that the pair is an identifiable wrong pair and is discarded by the filtering operation. Of the 120 pairs, 73 are identified as being wrong pairs by the filtering process, so 47 pairs remain as "possible" right pairs. □

3.6.3 Other examples of Differential Cryptanalysis

Differential cryptanalysis techniques can be used to attack **DES** with more than six rounds. An 8-round **DES** requires 2^{14} chosen plaintexts, and 10-, 12-, 14- and 16-round **DES**s can be broken with 2^{24}, 2^{31}, 2^{39} and 2^{47} chosen plaintexts, respectively. The attacks on more than 10 rounds are probably not practical at this time.

Several substitution-permutation product ciphers other than **DES** are also susceptible (to varying degrees) to differential cryptanalysis. These cryptosystems include several substitution-premutation cryptosystems that have been proposed in recent years, such as FEAL, REDOC-II, and LOKI.

TABLE 3.1
Cryptanalysis of 6-round DES

pair	right pair?	plaintext	ciphertext
1	**	86FA1C2B1F51D3BE	1E23ED7F2F553971
		C6F21C2B1B51D3BE	296DE2B687AC6340
2	**	EDC439EC935E1ACD	0F847EFE90466588
		ADCC39EC975E1ACD	93E84839F374440B
3	**	9468A0BE00166155	3D6A906A6566D0BF
		D460A0BE04166155	3BC3B236398379E1
4	**	D4FF2B18A5A8AAC8	26B14738C2556BA4
		94F72B18A1A8AAC8	15753FDE86575A8F
5		09D0F2CF277AF54F	15751F4F11308114
		49D8F2CF237AF54F	6046A7C863F066AF
6		CBC7157240D415DF	7FCDC300FB9698E5
		8BCF157244D415DF	522185DD7E47D43A
7		0D4A1E84890981C1	E7C0B01E32557558
		4D421E848D0981C1	912C6341A69DF295
8	**	6CE6B2A9B8194835	75D52E028A5C48A3
		2CEEB2A9BC194835	6C88603B48E5A8CE
9	**	799F63C3C9322C1A	A6DA322B8F2444B5
		399763C3CD322C1A	6634AA9DF18307F4
10	**	1B36645E381EDF48	1F91E295D559091B
		5B3E645E3C1EDF48	D094FC12C02C17CA
11		85CA13F50B4ADBB9	ED108EE7397DDE0A
		C5C213F50F4ADBB9	3F405F4A3E254714
12	**	7963A8EFD15BC4A1	8C714399715A33BA
		396BA8EFD55BC4A1	C344C73CC97E4AC4
13		7BCFF7BCA455E65E	475A2D0459BCCE62
		3BC7F7BCA055E65E	8E94334AEF359EF8
14		0C505CEDB499218C	D3C66239E89CC076
		4C585CEDB099218C	9A316E801EE18EB1
15		6C5EA056CDC91A14	BC7EBA159BCA94E6
		2C56A056C9C91A14	67DB935C21FF1A8D
16	**	6622A441A0D32415	35F8616FEBA62883
		262AA441A4D32415	4313E1925F5B64BC
17		C0333C994AFF1C99	D46A4CF1C0221B11
		803B3C994EFF1C99	D22B42DB150E2CE8
18		9E7B2974F00E1A6E	172D286D9606E6FE
		DE732974F40E1A6E	2217A91F8C427D27
19	**	CF592897BFD70C7E	FB892B59E7DCE7EC
		8F512897BBD70C7E	C328B765E1CC6653
20		E976CF19124A9FA1	905BF24188509FA6
		A97ECF19164A9FA1	9ADDBA0C23DD724F
21	**	5C09696E7363675D	92D60E5C71801A99
		1C01696E7763675D	DD90908A4FE8168F
22	**	A8145AB3C1B2C7DE	F68FC9F80564847B
		E81C5AB3C5B2C7DE	51C041B5711B8132
23		47DF6A0BB1787159	52E36C4CA22EA5A2
		07D76A0BB5787159	373EAFD503F68DE4
24	*	7CE65464329B4E6D	832A9D7032015D9F
		3CEE5464369B4E6D	85E2CE665571E99C

pair	right pair?	plaintext	ciphertext
25	**	421FB6AD95791BA7	D1E730BA1DB565E7
		0217B6AD91791BA7	188E61735FA4F3CE
26	**	C58E9A361368FFD6	795EB9D30CAE6879
		85869A361768FFD6	26D37AC4867ACC61
27	**	DD86B6C74C8EA4E2	CC3B6915C9A348DF
		9D8EB6C7488EA4E2	104C2394555645F0
28	**	43DB9D2F483CA585	E3E4DA503D1B9396
		03D39D2F4C3CA585	4EA02C0061332443
29	*	855A309F96FEA5EA	85AD6E9E352AFAFA
		C552309F92FEA5EA	929D22370ACAB80D
30	*	AB3CA25B02BD18C8	0F7D768E9203F786
		EB34A25B06BD18C8	A1313BC26A99D353
31	**	A9F7A6F4A7C00E06	F26B385E6BA057FD
		E9FFA6F4A3C00E06	203D8384F8F54D19
32	**	688B9ACD856D1312	C41D99C107B4EF76
		28839ACD816D1312	6CC817CA025A7DAC
33	**	76BF0621C03D4CD9	BBE1F95AFC1E052A
		36B70621C43D4CD9	561F4801F2EB0C63
34	**	014CF8D1F981B8EE	D27091C4314CBFE8
		4144F8D1FD81B8EE	B7976D6A80E3DB61
35	**	487D66EDE0405F8C	8136325C0AEB84CE
		087566EDE4405F8C	8C638BC4495B69A0
36	**	DDCA47093A362521	51040CF16B600FAA
		9DC247093E362521	7FC75515AC3CAAF9
37	**	45A9D34A3996F6D9	F2004B854AE6C46C
		05A1D34A3D96F6D9	546825016B03D193
38	**	295D2FBFB00875EA	A309DF027E69C265
		69552FBFB40875EA	4F633FFB95A0C11E
39		964C8B98D590D524	1FF1D0271D6F6C18
		D6448B98D190D524	8CF2D8D401EBFC0F
40		60383D2BAF0836BC	10A82D55FC480640
		20303D2BAB0836BC	602346173581EF79
41	**	5CF8D539A22A1CAD	92685D806FBE8738
		1CF0D539A62A1CAD	17006DAB2D28081C
42		F95167CAB6565609	C52E2EB27446054E
		B95967CAB2565609	0C219F686840E57A
43		49F1C83615874122	2680C8ECDF5E51CD
		09F9C83611874122	5022A7B69B4E75EF
44	**	ACB2EC1941B03765	D6B593460098DEC5
		ECBAEC1945B03765	D3190A0200FC6B9B
45		CCCC129D5CB55EC0	3AD22B7EF59E0D5E
		8CC4129D58B55EC0	A48C92CBEC17E430
46	**	917FF8E2EE6B78D5	EF847E058DB71724
		D177F8E2EA6B78D5	F243F0554A00E4C5
47	**	51DBCF028E96DE00	574897CA1EE73885
		11D3CF028A96DE00	9F0FD0A5B2C2B5FD
48	*	2094942E093463CE	59F6A018C6A0D820
		609C942E0D3463CE	799FE001432346C0

pair	right pair?	plaintext	ciphertext
49	**	50FB0723D7CD1081	16AF758395EA3A7D
		10F30723D3CD1081	CDCB23392D144BED
50	*	740815A4F6CDCABB	4A84D2ED4D9351AB
		340015A4F2CDCABB	5923D04CE94D6111
51	**	EDA46A1AE93735DC	0B302A51B7E5476A
		ADAC6A1AED3735DC	5F817F0ABC770E75
52	*	08BC39B766B2C128	DFB5F3F500BC0100
		48B439B762B2C128	B7B9FED8AC93EBFA
53	**	A74E29BBA98F2312	A2B352B7F922E8DA
		E74629BBAD8F2312	D6BC4B89CED2DEAC
54	**	D6F50D31EE4E68AB	4D464847065C0938
		96FD0D31EA4E68AB	7554D87AEDCE5634
55	*	06191AA594891CF5	649C1D084F920F9E
		46111AA590891CF5	BE12A10384365E19
56		5EA7EFD557946962	15E664293F4D77EE
		1EAFEFD553946962	E23396A758DC9CE6
57	**	41FB7704781CC88A	8ABD385C441FD6CE
		01F377047C1CC88A	06DE8D55777AB65C
58	**	9689B9123F7C5431	E1E63120742099BB
		D681B9123B7C5431	1AF88A2CF6649A4A
59		6F25032B4A309BFE	48FE50DE774288D7
		2F2D032B4E309BFE	47950691260D5E10
60	**	D8C4B02D8E8BF1E9	F34D565E6AE85683
		98CCB02D8A8BF1E9	A4D2DB548622A8E8
61	**	F663E8CCEE86805B	51BD62C9D5D0F0BB
		B66BE8CCEA86805B	D2ABB03CF9D26C0A
62	**	428B29BFDFA838DB	006D62A65761089F
		028329BFDBA838DB	9FD73EF6124B0C11
63	**	04BE2D22D81EDC66	26D99536D99B5707
		44B62D22DC1EDC66	94144EBDA0CDEB55
64	**	667B779123A3EF80	5D09CBF2CE7E5A69
		2673779127A3EF80	5EFF8BFCA7BAA152
65	**	BC86D401D6572438	E05572AAA5F6C377
		FC8ED401D2572438	3C670BC455144F61
66	**	6FE5E9547659E401	2C465BF6F52F864C
		2FEDE9547259E401	B71D106444F95F31
67	**	27D3BAC6453BE3DE	8F160E29000461CD
		67DBBAC6413BE3DE	2A6660F46487F885
68	**	1D864E7642A7023A	65F91EEBFD8A9C05
		5D8E4E7646A7023A	84761791B3C36661
69	**	5256CA6894707CBA	91527F9349ABCF15
		125ECA6890707CBA	30F28F06A7B0A35A
70	**	C05383B8EFCD2BD7	710B6EC61BF63E9C
		805B83B8EBCD2BD7	53AC029D8E0179D5
71		50EB21CA13F9A96E	26D95BA4DE4C85CF
		10E321CA17F9A96E	8F01A90F638AFFF6
72	**	60EB1229ACD90EDC	3890EE8567782F96
		20E31229A8D90EDC	EE404DF7BE537589

pair	right pair?	plaintext	ciphertext
73		8E9A17D17B173B99	885C3933627EDEF0
		CE9217D17F173B99	B7ABB6DF5835E962
74		6EC5CD0802C98817	A985ADFB1FEE013C
		2ECDCD0806C98817	0428DE024B7E4604
75	**	1E81712FF1145C06	417E667A99B3CFA5
		5E89712FF5145C06	5C24AA056EB1ADBA
76	**	DF3C5C13311AEC7C	BF01675096F1C48A
		9F345C13351AEC7C	243D99BCE12DB864
77	**	7C34472994127C2D	713915DA311A7CF4
		3C3C472990127C2D	E9733D11D787E20B
78	**	37304DABA75EAFB3	EFB5C37FA0238ADF
		77384DABA35EAFB3	A728F7407AF958B3
79		D03A16E4C2D8B54B	423FC0AC24CEFEDD
		903216E4C6D8B54B	047D8595DB4D372E
80	**	8CED882B5D91832E	0006E2DE3AF5C2B5
		CCE5882B5991832E	00F6AA9ED614001B
81	**	1BB0E6C79EFBEC41	E9AED4363915775A
		5BB8E6C79AFBEC41	655BC48F1FFB5165
82		D41B8346DA9E2252	34F5E0BCC5B042EA
		94138346DE9E2252	702D2C48CDBE5173
83	*	02A9D0A0A91F6304	E2F1C10E59AF07C5
		42A1D0A0AD1F6304	BDEE6AA00F25F840
84	**	841B3E27C8F0A561	2B288E554D712C92
		C4133E27CCF0A561	FF8609C9E7301162
85	**	CDF0A8D6EE909185	5D661834D1C76324
		8DF8A8D6EA909185	22034D57D21FFB56
86	**	4C31AC854F44EA34	BD016309AEDB9BB1
		0C39AC854B44EA34	C72EEDC4FA1D9312
87		DB3FC0703C972930	296ABCFBF01DF991
		9B37C07038972930	CA4700686F9F83A2
88		E4B362BFD6A7CFD1	20FDAF335F25B1DA
		A4BB62BFD2A7CFD1	008C24D75E14ACBD
89		F234232A0E0A4A28	90CFD699F2DEC5BD
		B23C232A0A0A4A28	2918D3DE0C1B689C
90	**	71265345A5874004	3052CE3CE88710AE
		312E5345A1874004	38F0FC685DF30564
91	**	3E6364548C857110	0E8581E42C9FEC6F
		7E6B645488857110	4DD1751861EC5529
92	*	464FBEDBD78900A7	90F5F9ADEDED627A
		0647BEDBD38900A7	2EF4C540425E339B
93	**	373B75F847480BB0	5408B964F8442D16
		773375F843480BB0	805287D52599E9F0
94	**	D714E87810DE97AC	4EC4D623108FA909
		971CE87814DE97AC	0AA0725CED10D6A3
95		B9B5932EF54B2C60	4B438B3CCF36DEC9
		F9BD932EF14B2C60	054C6A337709280D
96	**	2F283C38D2E4E1DD	83515FB6DFEA90B8
		6F203C38D6E4E1DD	09BCC4FF38C78C23

pair	right pair?	plaintext	ciphertext
97	**	1EB8ADAA43BBD575	21A1E04813616E42
		5EB0ADAA47BBD575	D044BA3F25DFD02A
98	**	3164AA5454D9F991	9382C6C1883F1038
		716CAA5450D9F991	5CDFED4FF2117DEC
99		D78C1C5C6F2243D2	1CCEB091E030E6A6
		97841C5C6B2243D2	4DA2CD67CC449B21
100		BBE212A7D3CE3D14	2917C207B4D93E0D
		FBEA12A7D7CE3D14	A01D50E5A2B902D8
101	**	104917795E98D0FB	40916A71385C2803
		504117795A98D0FB	413FD26EF671F46D
102	**	4DDA114D6EFEEEB4	2E2C65E1D5CBAC31
		0DD2114D6AFEEEB4	A16FF03BC0913ED6
103		E0BED7B285BF0A77	5D9EFEFF0AD10490
		A0B6D7B281BF0A77	4C6CA1FAC36A8E5B
104	**	0AE1555FA1716214	378400BCED39EB81
		4AE9555FA5716214	A1E0C758BD8912C2
105	**	4657C26790FCB354	588BA079B2E7ED20
		065FC26794FCB354	DA90827AEED7A41F
106	**	32BD719B0DC1B091	F3477C7552BCB05D
		72B5719B09C1B091	EFF444449D66BE9E
107	**	0992F8C8C73A9BFE	9F3FFD0F158295F6
		499AF8C8C33A9BFE	C138358DCECC8FC7
108		02C3F061A237BBEB	AC28B0307127EA7C
		42CBF061A637BBEB	3FF1DAED9E0FCBC5
109	**	80E529E69EDE6827	1DF1DB7B66BA1AF1
		C0ED29E69ADE6827	15700151A5804549
110		B55E84630067B8D5	88321611FF9DA421
		F55684630467B8D5	90649D7EACF91F9A
111		2749C2EBC603BFF2	A62B23A7348E2C3A
		6741C2EBC203BFF2	EB760A09C7FF5153
112	**	C4C5E14D4C5D9FF5	ABC2312FBFD94DF5
		84CDE14D485D9FF5	D2BB5954E5062D53
113	**	1566BA21F2647E18	A247ED988457CB78
		556EBA21F6647E18	5E99F231005F5249
114	**	2D093D426D922F92	5DF62030B9F23AE9
		6D013D4269922F92	5D92DA1FA3D07BA1
115		004518468E0C96C3	F28D85FF7E84F38F
		404D18468A0C96C3	52541B0443053C57
116	**	437B70A98AE03344	04B3FBF9823B4CF7
		037370A98EE03344	14EBEC79DAD3093E
117		2D01F1073D3E375B	F10B3E1EE356226C
		6D09F107393E375B	6FF26DA5E3525B62
118	*	66573DD7E0D7F110	F2F26204C29FE51E
		265F3DD7E4D7F110	083A4ECE57E429AC
119		0846DB9538155201	F120D0D2AE788057
		484EDB953C155201	00CC914A33034782
120		ABB34FC195C820D1	5F17AE066B50FC81
		EBBB4FC191C820D1	2858DD63A2FA4B53

3.7 Notes and References

A nice article on the history **DES** is by Smid and Branstad [SB92]. Federal Information Processing Standards (FIPS) publications concerning **DES** include the following: description of **DES** [NBS77]; implementing and using **DES** [NBS81]; modes of operation of **DES** [NBS80]; and authentication using **DES** [NBS85].

Some properties of the S-boxes are studied by Brickell, Moore, and Purtill [BMP87].

The DEC **DES** chip is described in [EB93]. Wiener's key search machine was described at CRYPTO '93 [WI94].

The time-memory trade-off for **DES** is due to Hellman [HE80]. A more general time-memory trade-off is presented by Fiat and Naor in [FN91].

The technique of differential cryptanalysis was developed by Biham and Shamir [BS91] (see also [BS93A] and their book [BS93], where cryptanalysis of other cryptosystems is also discussed). Our treatment of differential cryptanalysis is based largely on [BS93].

Another new method of cryptanalysis that can be used to attack **DES** and other similar cryptosystems is the linear cryptanalysis of Matsui [MA94, MA94A].

Descriptions of other substitution-permutation cryptosystems can be found in the following sources: LUCIFER [FE73]; FEAL [MI91]; REDOC-II [CW91]; and LOKI [BKPS90].

Exercises

3.1 Prove that **DES** decryption can be done by applying the **DES** encryption algorithm to the plaintext with the key schedule reversed.

3.2 Let **DES**(x, K) represent the encryption of plaintext x with key K using the **DES** cryptosystem. Suppose y = **DES**(x, K) and y' = **DES**$(c(x), c(K))$, where $c(\cdot)$ denotes the bitwise complement of its argument. Prove that $y' = c(y)$ (i.e., if we complement the plaintext and the key, then the ciphertext is also complemented). Note that this can be proved using only the "high-level" description of **DES** — the actual structure of S-boxes and other components of the system are irrelevant.

3.3 One way to strengthen **DES** is by *double encryption*: Given two keys, K_1 and K_2, define $y = e_{K_2}(e_{K_1}(x))$ (of course, this is just the product of **DES** with itself). If it happened that the encryption function e_{K_2} was the same as the decryption function d_{K_1}, then K_1 and K_2 are said to be *dual keys*. (This is very undesirable for double encryption, since the resulting ciphertext is identical to the plaintext.) A key is *self-dual* if it is its own dual key.

(a) Prove that if C_0 is either all 0's or all 1's and D_0 is either all 0's or all 1's, then K is self-dual.

(b) Prove that the following keys (given in hexadecimal notation) are self-dual:

`0101010101010101`

```
FEFEFEFEFEFEFEFE
1F1F1F1F0E0E0E0E
E0E0E0E0F1F1F1F1
```

(c) Prove that if $C_0 = 0101\ldots01$ or $1010\ldots10$ (in binary), then the x-or of the bitstrings C_i and C_{17-i} is $1111\ldots11$, for $1 \le i \le 16$ (a similar statement holds for the D_i's).

(d) Prove that the following pairs of keys (given in hexadecimal notation) are dual:

```
E001E001F101F101   01E001E001F101F1
FE1FFE1FFE0EFE0E   1FFE1FFE0EFE0EFE
E01FF01FFF10FF10   1FE01FE00EF10EF1
```

3.4 A message authentication code (MAC) can be produced by using CFB mode, as well as by using CBC mode. Given a sequence of plaintext blocks $x_1 \ldots x_n$, suppose we define the initialization vector IV to be x_1. Then encrypt $x_2 \ldots x_n$ using key K in CFB mode, obtaining $y_1 \ldots y_{n-1}$ (note that there are only $n-1$ ciphertext blocks). Finally, define the MAC to be $e_K(y_{n-1})$. Prove that this MAC is identical to the MAC produced in Section 3.4.1 using CBC mode.

3.5 Suppose a sequence of plaintext blocks, $x_1 \ldots x_n$, is encrypted using **DES**, producing ciphertext blocks $y_1 \ldots y_n$. Suppose that one ciphertext block, say y_i, is transmitted incorrectly (i.e., some 1's are changed to 0's and vice versa). Show that the number of plaintext blocks that will be decrypted incorrectly is equal to one if ECB or OFB modes were used for encryption; and equal to two if CBC or CFB modes were used.

3.6 The purpose of this question is to investigate a simplified time-memory trade-off for a chosen plaintext attack. Suppose we have a cryptosystem in which $\mathcal{P} = \mathcal{C} = \mathcal{K}$, which attains perfect secrecy. Then it must be the case that $e_K(x) = e_{K_1}(x)$ implies $K = K_1$. Denote $\mathcal{P} = Y = \{y_1, \ldots, y_N\}$. Let x be a fixed plaintext. Define the function $g : Y \to Y$ by the rule $g(y) = e_y(x)$. Define a directed graph G having vertex set Y, in which the edge set consists of all the directed edges of the form $(y_i, g(y_i))$, $1 \le i \le N$.

(a) Prove that G consists of the union of disjoint directed cycles.

(b) Let T be a desired time parameter. Suppose we have a set of elements $Z = \{z_1, \ldots, z_m\} \subseteq Y$ such that, for every element $y_i \in Y$, either y_i is contained in a cycle of length at most T, or there exists an element $z_j \ne y_i$ such that the distance from y_i to z_j (in G) is at most T. Prove that there exists such a set Z such that

$$|Z| \le \frac{2N}{T},$$

so $|Z|$ is $O(N/T)$.

(c) For each $z_j \in Z$, define $g^{-T}(z_j)$ to be the element y_i such that $g^T(y_i) = z_j$, where g^T is the function that consists of T iterations of g. Construct a table X consisting of the ordered pairs $(z_j, g^{-T}(z_j))$, sorted with respect to their first coordinates.

A pseudo-code description of an algorithm to find K, given $y = e_K(x)$, is presented in Figure 3.15. Prove that this algorithm finds K in at most T steps. (Hence the time-memory trade-off is $O(N)$.)

FIGURE 3.15
Time-memory trade-off

```
1.  y_start = y
2.  backup = false
3.  while g(y) ≠ y_start do
4.      if y = z_j for some j and not backup then
5.          y = g^{-T}(z_j)
6.          backup = true
        else
7.          y = g(y)
8.      K = y
```

FIGURE 3.16
Differential attack on 4-round DES

```
    Input: L_0 R_0, L_0* R_0*, L_3 R_3 and L_3* R_3*, where L_0' = 20000000_16 and R_0' =
    00000000_16
1.  compute C' = P^{-1}(R_4')
2.  compute E = E(L_4) and E* = E(L_4*)
3.  for j = 2 to 8 do
        compute test_j(E_j, E_j*, C_j')
```

(d) Describe a pseudo-code algorithm to construct the desired set Z in time $O(NT)$ without using an array of size N.

3.7 Compute the probabilities of the following 3-round characteristic:

L_0'	=	00200008_{16}	R_0'	=	00000400_{16}	
L_1'	=	00000400_{16}	R_1'	=	00000000_{16}	$p = ?$
L_2'	=	00000000_{16}	R_2'	=	00000400_{16}	$p = ?$
L_3'	=	00000400_{16}	R_3'	=	00200008_{16}	$p = ?$

3.8 Here is a differential attack on a 4-round **DES**. It uses the following characteristic, which is a special case of the characteristic presented in Figure 3.10:

L_0'	=	20000000_{16}	R_0'	=	00000000_{16}	
L_1'	=	00000000_{16}	R_1'	=	20000000_{16}	$p = 1$

(a) Suppose that the following algorithm presented in Figure 3.16 is used to compute sets $test_2, \ldots test_8$. Show that $J_j \in test_j$ for $2 \le j \le 8$.

(b) Given the following plaintext-ciphertext pairs, determine the key bits in $J_2, \ldots, J_8$.

plaintext	ciphertext
18493AC485B8D9A0	E332151312A18B4F
38493AC485B8D9A0	87391C27E5282161
482765DDD7009123	B5DDD8339D82D1D1
682765DDD7009123	81F4B92BD94B6FD8
ABCD098733731FF1	93A4B42F62EA59E4
8BCD098733731FF1	ABA494072BF411E5
13578642AAFFEDCB	FDEB526275FB9D94
33578642AAFFEDCB	CC8F72AAE685FDB1

(c) Compute the entire key (14 key bits remain to be determined, which can be done by exhaustive search).