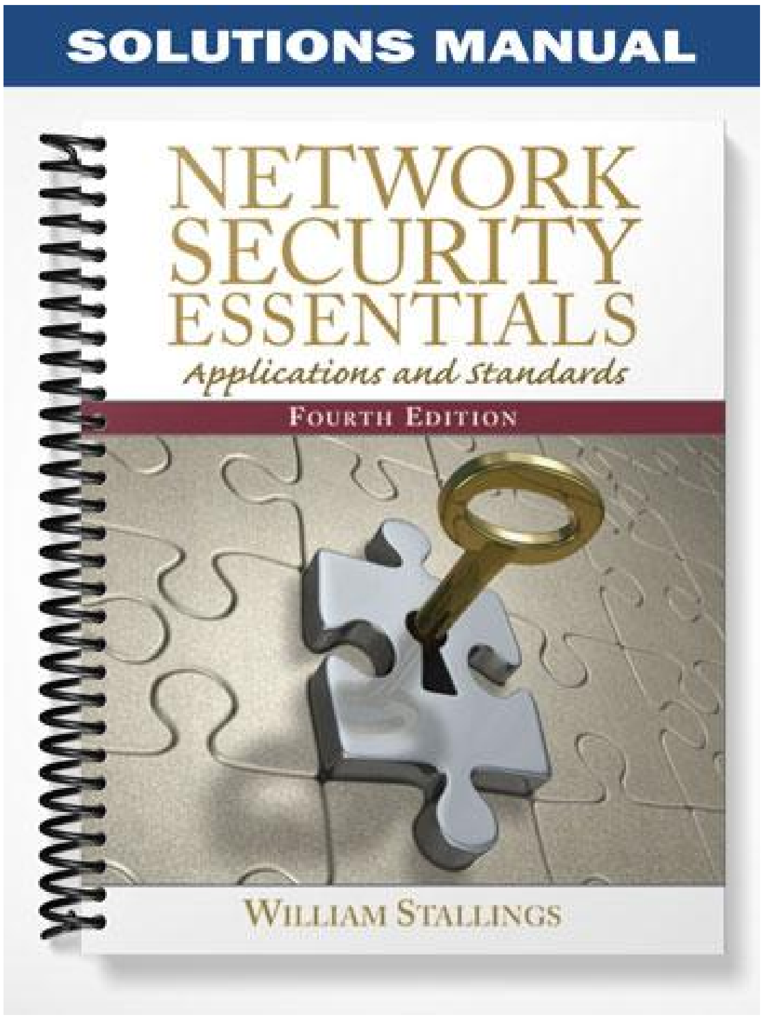


SOLUTIONS MANUAL

NETWORK SECURITY ESSENTIALS

Applications and Standards

FOURTH EDITION

WILLIAM STALLINGS

CHAPTER 2 SYMMETRIC ENCRYPTION AND MESSAGE CONFIDENTIALITY

ANSWERS TO QUESTIONS

2.1 Plaintext, encryption algorithm, secret key, ciphertext, decryption algorithm.

2.2 Permutation and substitution.

2.3 One secret key.

2.4 A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.

2.5 Cryptanalysis and brute force.

2.6 In some modes, the plaintext does not pass through the encryption function, but is XORed with the output of the encryption function. The math works out that for decryption in these cases, the encryption function must also be used.

2.7 With triple encryption, a plaintext block is encrypted by passing it through an encryption algorithm; the result is then passed through the same encryption algorithm again; the result of the second encryption is passed through the same encryption algorithm a third time. Typically, the second stage uses the decryption algorithm rather than the encryption algorithm.

2.8 There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES by repeating the key.

Answers to Problems

2.1 **a.**

2	8	10	7	9	6	3	1	4	5
C	R	Y	P	T	O	G	A	H	I
B	E	A	T	T	H	E	T	H	I
R	D	P	I	L	L	A	R	F	R
O	M	T	H	E	L	E	F	T	O
U	T	S	I	D	E	T	H	E	L
Y	C	E	U	M	T	H	E	A	T
R	E	T	O	N	I	G	H	T	A
T	S	E	V	E	N	I	F	Y	O
U	A	R	E	D	I	S	T	R	U
S	T	F	U	L	B	R	I	N	G
T	W	O	F	R	I	E	N	D	S

4	2	8	10	5	6	3	7	1	9
N	E	T	W	O	R	K	S	C	U
T	R	F	H	E	H	F	T	I	N
B	R	O	U	Y	R	T	U	S	T
E	A	E	T	H	G	I	S	R	E
H	F	T	E	A	T	Y	R	N	D
I	R	O	L	T	A	O	U	G	S
H	L	L	E	T	I	N	I	B	I
T	I	H	I	U	O	V	E	U	F
E	D	M	T	C	E	S	A	T	W
T	L	E	D	M	N	E	D	L	R
A	P	T	S	E	T	E	R	F	O

```
ISRNG  BUTLF  RRAFR  LIDLP  FTIYO  NVSEE  TBEHI  HTETA
EYHAT  TUCME  HRGTA  IOENT  TUSRU  IEADR  FOETO  LHMET
NTEDS  IFWRO  HUTEL  EITDS
```

b. The two matrices are used in reverse order. First, the ciphertext is laid out in columns in the second matrix, taking into account the order dictated by the second memory word. Then, the contents of the second matrix are read left to right, top to bottom and laid out in columns in the first matrix, taking into account the order dictated by the first memory word. The plaintext is then read left to right, top to bottom.

c. Although this is a weak method, it may have use with time-sensitive information and an adversary without immediate access to good cryptanalysis (e.g., tactical use). Plus it doesn't require anything more than paper and pencil, and can be easily remembered.

2.2 **a.** Let –X be the additive inverse of X. That is $-X \boxed{+} X = 0$. Then:

$$P = (C \boxed{+} -K_1) \oplus K_0$$

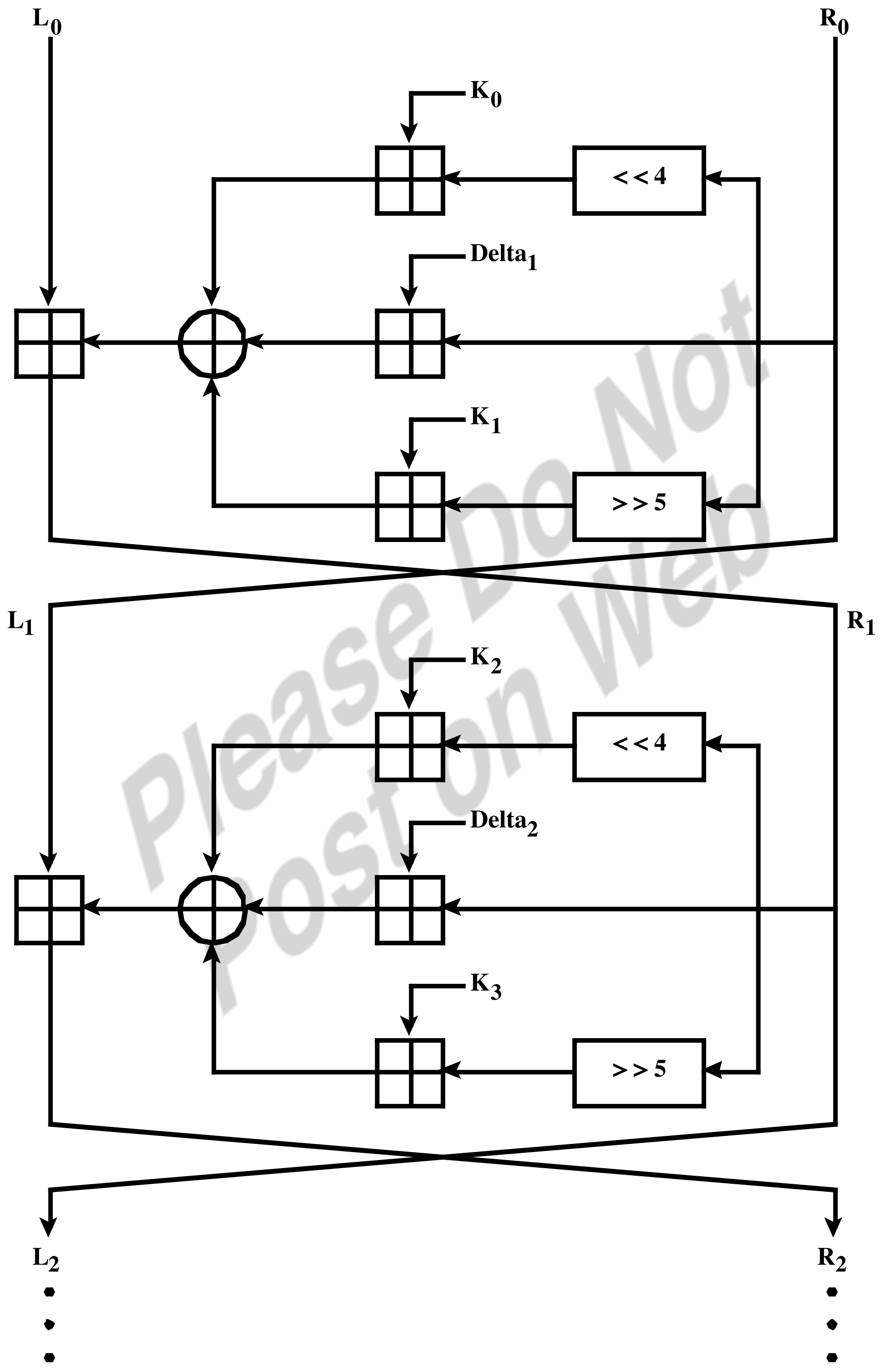
b. First, calculate –C'. Then $-C' = (P' \oplus K_0) \boxed{+} (-K_1)$. We then have:

$C \boxed{+} -C' = (P \oplus K_0) \boxed{+} (P' \oplus K_0)$

However, the operations $\boxed{+}$ and $\oplus$ are not associative or distributive with one another, so it is not possible to solve this equation for K_0.

2.3 **a.** The constants ensure that encryption/decryption in each round is different.

b. First two rounds:



c. First, let's define the encryption process:

$L_2 = L_0 \boxed{+} [(R_0 << 4) \boxed{+} K_0] \oplus [R_0 \boxed{+} \delta_1] \oplus [(R_0 >> 5) \boxed{+} K_1]$

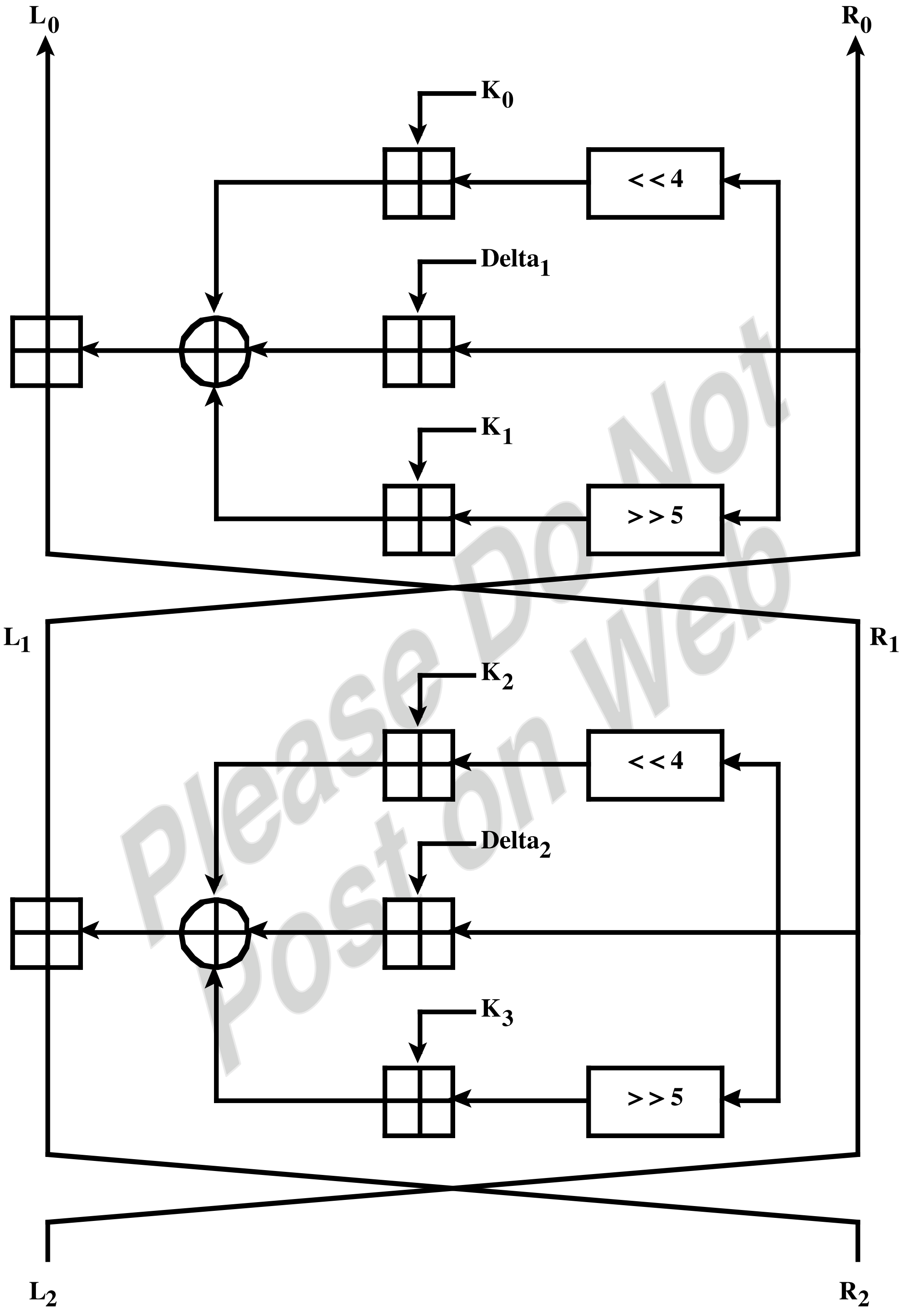
$R_2 = R_0 \boxed{+} [(L_2 << 4) \boxed{+} K_2] \oplus [L_2 \boxed{+} \delta_2] \oplus [(L_2 >> 5) \boxed{+} K_3]$

Now the decryption process. The input is the ciphertext (L_2, R_2), and the output is the plaintext (L_0, R_0). Decryption is essentially the same as encryption, with the subkeys and delta values applied in reverse order. Also note that it is not necessary to use subtraction because there is an even number of additions in each equation.

$R_0 = R_2 \boxed{+} [(L_2 << 4) \boxed{+} K_2] \oplus [L_2 \boxed{+} \delta_2] \oplus [(L_2 >> 5) \boxed{+} K_3]$

$L_0 = L_2 \boxed{+} [(R_0 << 4) \boxed{+} K_0] \oplus [R_0 \boxed{+} \delta_1] \oplus [(R_0 >> 5) \boxed{+} K_1]$

d.



2.4 To see that the same algorithm with a reversed key order produces the correct result, consider the Figure 2.2, which shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm (the result would be the same for any number of rounds). For clarity, we use the notation LE_i and RE_i for data traveling through the encryption algorithm and LD_i and RD_i for data traveling through the decryption algorithm.

The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the ith encryption round be $LE_i || RE_i$ (L_i concatenated with R_i). Then the corresponding input to the (16 – i)th decryption round is $RD_i || LD_i$.

Let us walk through the figure to demonstrate the validity of the preceding assertions. To simplify the diagram, it is unwrapped, not showing the swap that occurs at the end of each iteration. But note that the intermediate result at the end of the ith stage of the encryption process is the 2w-bit quantity formed by concatenating LE_i and RE_i, and that the intermediate result at the end of the ith stage of the decryption process is the 2w-bit quantity formed by concatenating LD_i and RD_i. After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is $RE_{16} || LE_{16}$. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is $RE_{16} || LE_{16}$, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that:

$$\begin{aligned} LE_{16} &= RE_{15} \\ RE_{16} &= LE_{15} \oplus F(RE_{15}, K_{16}) \end{aligned}$$

On the decryption side:

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\ &= RE_{16} \oplus F(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \end{aligned}$$

The XOR has the following properties:

$$[A \oplus B] \oplus C = A \oplus [B \oplus C]$$
$$D \oplus D = 0$$
$$E \oplus 0 = E$$

Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$. Therefore, the output of the first round of the decryption process is $LE_{15} || RE_{15}$, which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the ith iteration of the encryption algorithm:

$$\begin{aligned} LE_i &= RE_{i-1} \\ RE_i &= LE_{i-1} \oplus F(RE_{i-1}, K_i) \end{aligned}$$

Rearranging terms:

$$\begin{aligned} RE_{i-1} &= LE_i \\ LE_{i-1} &= RE_i \oplus F(RE_{i-1}, K_i) = RE_i \oplus F(LE_i, K_i) \end{aligned}$$

Thus, we have described the inputs to the ith iteration as a function of the outputs, and these equations confirm the assignments shown in the right-hand side of the following figure.

Finally, we see that the output of the last round of the decryption process is $RE_0 || LE_0$. A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.

2.5 Because of the key schedule, the round functions used in rounds 9 through 16 are mirror images of the round functions used in rounds 1 through 8. From this fact we see that encryption and decryption are identical. We are given a ciphertext *c*. Let *m*' = *c*. Ask the encryption oracle to encrypt *m*'. The ciphertext returned by the oracle will be the decryption of *c*.

2.6 For $1 \le i \le 128$, take $c_i \in \{0, 1\}^{128}$ to be the string containing a 1 in position i and then zeros elsewhere. Obtain the decryption of these 128 ciphertexts. Let $m_1, m_2, \ldots, m_{128}$ be the corresponding plaintexts. Now, given any ciphertext c which does not consist of all zeros, there is a unique nonempty subset of the c_i's which we can XOR together to obtain c. Let $I(c) \subseteq \{1, 2, \ldots, 128\}$ denote this subset. Observe

$$c = \bigoplus_{i \in I(c)} c_i = \bigoplus_{i \in I(c)} E(m_i) = E\left(\bigoplus_{i \in I(c)} m_i\right)$$

Thus, we obtain the plaintext of c by computing $\bigoplus_{i \in I(c)} m_i$. Let **0** be the all-zero string. Note that $\mathbf{0} = \mathbf{0} \oplus \mathbf{0}$. From this we obtain $E(\mathbf{0}) = E(\mathbf{0} \oplus \mathbf{0}) = E(\mathbf{0}) \oplus E(\mathbf{0}) = \mathbf{0}$. Thus, the plaintext of $c = \mathbf{0}$ is $m = \mathbf{0}$. Hence we can decrypt every $c \in \{0, 1\}^{128}$.

2.7 **a.**

Pair	Probability
00	$(0.5 - \partial)^2 = 0.25 - \partial + \partial^2$
01	$(0.5 - \partial) \times (0.5 + \partial) = 0.25 - \partial^2$
10	$(0.5 + \partial) \times (0.5 - \partial) = 0.25 - \partial^2$
11	$(0.5 + \partial)^2 = 0.25 + \partial + \partial^2$

b. Because 01 and 10 have equal probability in the initial sequence, in the modified sequence, the probability of a 0 is 0.5 and the probability of a 1 is 0.5.

c. The probability of any particular pair being discarded is equal to the probability that the pair is either 00 or 11, which is $0.5 + 2\partial^2$, so the expected number of input bits to produce *x* output bits is $x/(0.25 - \partial^2)$.

d. The algorithm produces a totally predictable sequence of exactly alternating 1's and 0's.

2.8 **a.** For the sequence of input bits $a_1, a_2, \ldots, a_n$, the output bit *b* is defined as:

$$b = a_1 \oplus a_2 \oplus \ldots \oplus a_n$$

b. $0.5 - 2\partial^2$
c. $0.5 - 8\partial^4$
d. The limit as n goes to infinity is 0.5.

2.9 Use a key of length 255 bytes. The first two bytes are zero; that is K[0] = K[1] = 0. Thereafter, we have: K[2] = 255; K[3] = 254; … K[255]= 2.

2.10 a. Simply store i, j, and S, which requires $8 + 8 + (256 \times 8) = 2064$ bits
b. The number of states is $[256! \times 256^2] \approx 2^{1700}$. Therefore, 1700 bits are required.

2.11 a. By taking the first 80 bits of $v \parallel c$, we obtain the initialization vector, v. Since v, c, k are known, the message can be recovered (i.e., decrypted) by computing RC4($v \parallel k$) $\oplus$ c.
b. If the adversary observes that $v_i = v_j$ for distinct i, j then he/she knows that the same key stream was used to encrypt both m_i and m_j. In this case, the messages m_i and m_j may be vulnerable to the type of cryptanalysis carried out in part (a).
c. Since the key is fixed, the key stream varies with the choice of the 80-bit v, which is selected randomly. Thus, after approximately $\sqrt{\frac{\pi}{2}2^{80}} \approx 2^{40}$ messages are sent, we expect the same v, and hence the same key stream, to be used more than once.
d. The key k should be changed sometime before 2^{40} messages are sent.

2.12 a. No. For example, suppose C_1 is corrupted. The output block P_3 depends only on the input blocks C_2 and C_3.
b. An error in P_1 affects C_1. But since C_1 is input to the calculation of C_2, C_2 is affected. This effect carries through indefinitely, so that all ciphertext blocks are affected. However, at the receiving end, the decryption algorithm restores the correct plaintext for blocks except the one in error. You can show this by writing out the equations for the decryption. Therefore, the error only effects the corresponding decrypted plaintext block.

2.13 In CBC encryption, the input block to each forward cipher operation (except the first) depends on the result of the previous forward cipher operation, so the forward cipher operations cannot be performed in parallel. In CBC decryption, however, the input blocks for the inverse cipher function (i.e., the ciphertext blocks) are immediately available, so that multiple inverse cipher operations can be performed in parallel.

2.14 If an error occurs in transmission of ciphertext block C_i, then this error propagates to the recovered plaintext blocks P_i and P_{i+1}.

2.15 After decryption, the last byte of the last block is used to determine the amount of padding that must be stripped off. Therefore there must be at least one byte of padding.

2.16 **a.** Assume that the last block of plaintext is only L bytes long, where $L < 2w/8$. The encryption sequence is as follows (The description in RFC 2040 has an error; the description here is correct.):

1. Encrypt the first $(N - 2)$ blocks using the traditional CBC technique.
2. XOR P_{N-1} with the previous ciphertext block C_{N-2} to create Y_{N-1}.
3. Encrypt Y_{N-1} to create E_{N-1}.
4. Select the first L bytes of E_{N-1} to create C_N.
5. Pad P_N with zeros at the end and exclusive-OR with E_{N-1} to create Y_N.
6. Encrypt Y_N to create C_{N-1}.

The last two blocks of the ciphertext are C_{N-1} and C_N.

b. $P_{N-1} = C_{N-2} \oplus \mathrm{D}(K, [C_N \| X])$
$P_N \| X = (C_N \| 00\ldots0) \oplus \mathrm{D}(K, [C_{N-1}])$
P_N = left-hand portion of $(P_N \| X)$
where $\|$ is the concatenation function

2.17 **a.** Assume that the last block (P_N) has j bits. After encrypting the last full block (P_{N-1}), encrypt the ciphertext (C_{N-1}) again, select the leftmost j bits of the encrypted ciphertext, and XOR that with the short block to generate the output ciphertext.

b. While an attacker cannot recover the last plaintext block, he can change it systematically by changing individual bits in the ciphertext. If the last few bits of the plaintext contain essential information, this is a weakness.

2.18 Nine plaintext characters are affected. The plaintext character corresponding to the ciphertext character is obviously altered. In addition, the altered ciphertext character enters the shift register and is not removed until the next eight characters are processed.