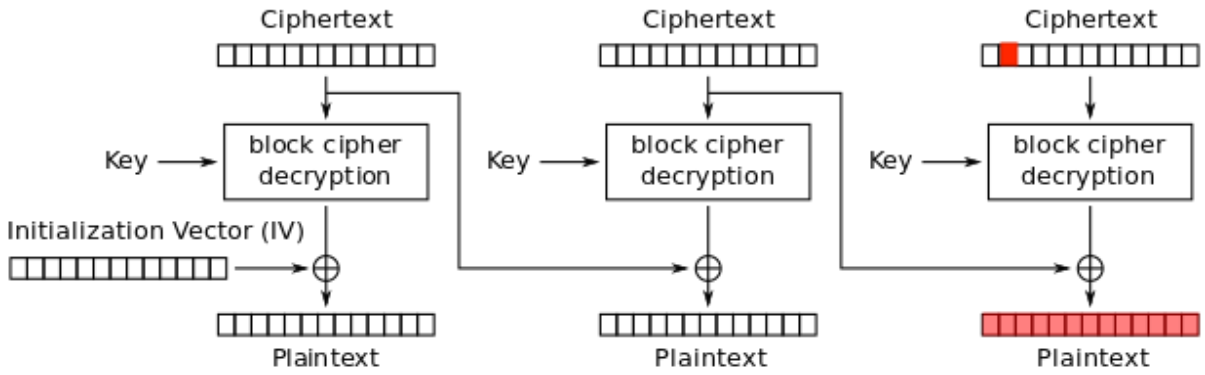
Modern Cryptographic Algorithms
Lab Report

JIGAR THUMMAR
(352063)

Q 21.15: Alice encrypts a message using AES and sends the ciphertext to Bob. Unfortunately, during the transmission, the 2nd bit of the third block in the ciphertext is corrupted. How much of the plaintext can Bob still recover if the mode of encryption is one of the following: CBC, CFB, OFB, or CTR?

Ans:
CBC:



Cipher Block Chaining (CBC) mode decryption

Because of self-healing property of CBC we'll only lose 3rd block and 4th block's second bit, Other blocks we can recover without any corruption.

CFB: it will invert the correspondin bit in the same plaintext block and mess up the whole next plaintext block.

OFB: it will affect only third block and invert the corresponding bit in the corresponding plaintext block.

CTR: it will affect only third block and invert the 2nd bit in the corresponding plaintext block.

Q 22.4: An attacker gets a copy of the shadow file, and he tries to guess Bob's password. It was said that the salt value makes it much more difficult for the attacker to do so. Do you agree or not?

A:
No, because shadow file already have salt value stored with hashed password and attacker can still use brute force attack for Bob's password and can get it. But it doesn't mean that salts are useless. As long as you use a unique salt for each row, then the salt will *slow down* an attack. The attacker will need to mount a brute force attack, rather than using rainbow tables against the password hashes.

Q 22.5 Currently, the salt used in the shadow file is saved in the file in plaintext. Isn't it better not to save the salt in plaintext? Please explain.

A :
It's perfectly secure to save salt in plaintext. Main use of salt is to prevent an attacker from using his dictionary for multiple ciphertexts. But we should make sure to use different salts for each ciphertexts.

Q 22.7 In Linux, the password hash is produced by applying a hash function for many rounds (e.g., 5000 rounds for SHA-512). This seems to waste time. Why does Linux do this?

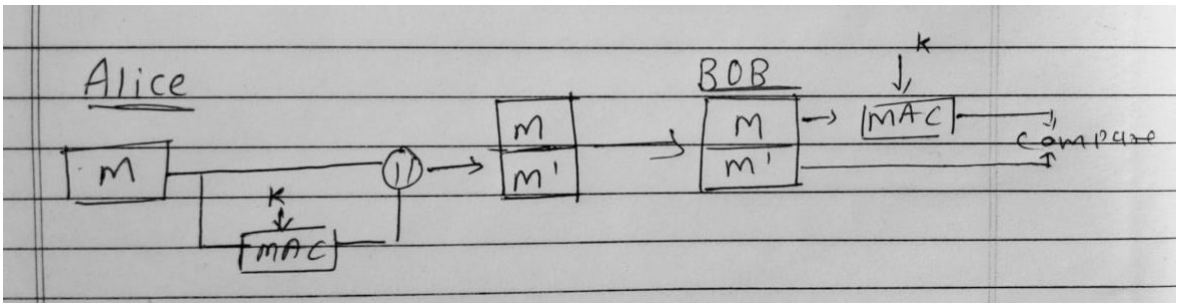
A:

We can not store password in their plain text so we need password storage where nobody can know what password is. So the solution that linux used is hash function.
Password field has three parts : algorithm used(ex 6 for SHA-512), salt, password hash.
Linux use multiple rounds of hash function to **slow down brute-force attack.**

Q 22.13 Charlie has arranged a blind date for Alice and Bob, who are both cryptographers, and they do not know each other before. Charlie also gave Alice and Bob a secret number K (nobody else knows K). Bob wants to make sure that the person he is dating is Alice, not somebody else. Please describe how Bob can ask Alice to securely prove that she is Alice (Alice will not reveal the secret number K to anybody).

A:
Alice and Bob can use MAC(Message Authentication Function).

MAC stands for Message Authentication Code. Here in MAC, sender and receiver share same key where sender generates a fixed size output called Cryptographic checksum or Message Authentication code and appends it to the original message. On receiver's side, receiver also generates the code and compares it with what he/she received thus ensuring the originality of the message.



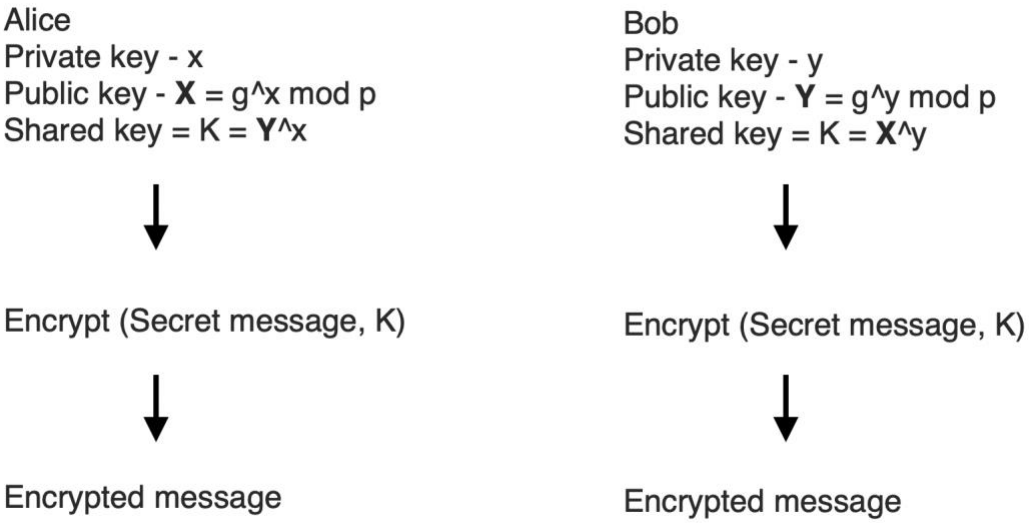
Alice will send message to Bob lets suppose message is **M**. First of all Alice will generate the MAC value **M'** by passing that message and key **K** into MAC algorithm and merge that **M'** with **M**. Now, she will send that **MM'** to Bob. Bob will receive the message and MAC value and pass the message to MAC algorithm with secret number **K**. He will get his MAC value and compare it to Alice's MAC value, if both are same than Bob will be sure that other person is Alice.

Q-23.1 In the Diffie-Hellman key exchange, Alice sends g^x mod p to Bob, and Bob sends g^y mod p to Alice. How do they get a common secret?

A:

- Suppose :

  – Alice's Private key : 'x' and public key : 'X'.
  – Bob's Private key : 'y' and public key : 'Y'.
  – We have two public parameters -
  p: some large prime number
  g: some integer.
  – Private keys x and y are randomly selected numbers from a finite field having p-1 elements.



- To send message :

  — Alice's public key X = g^x mod p.
  — Alice sends X to Bob.
  — Bob's public key Y = g^y mod p.
  — Bob sends Y to Alice.
  — Now, Alice has key 'K' define as Y^x and Bob has key K define as X^y.
  — Now, Alice and Bob can transfer secret message to each other by encrypting message with K and generating cipher text.

- How can X^y and Y^x be same??

  — X^y can also be rewritten as (g^x)^y.

  Bob's shared key = K = X^y = (g^x mod p)^y = g^xy mod p

  — Y^x can also be rewritten as (g^y)^x.

  Alice's shared key = K = Y^x = (g^y mod p)^x = g^yx mod p

  — g^xy mod p = g^yx mod p

  Since both can generate (g^y)^x or (g^x)^y without knowing other's private key, they can decipher the cipher text to find out secret.

Q23.4: Let n = 2419 = 41 * 59 and e = 7. (1) Find the private key d. (2) Encrypt the message 6. (3) Sign the message 10. Assume RSA is used. You only need to show the formula; there is no need to calculate the final results.

A:

n = 2419
p = 41
q = 59
e = 7
(PT = Plain Text and CT = Cipher Text)

(1)Private key(d) :

(d*e) mod (p-1)(q-1) = 1 or (d = e^-1 mod (p-1)(q-1))
7d mod (40*58) = 1
7d mod 2320 = 1
d = 663

(2) Encrypt the message 6 :

CT = PT^e mod n
CT = 6^7 mod 2419
CT = 1751

(3) Sign tha message 10 :

PT = CT^d mod n
PT = 10^663 mod 2419

Q 3.1 A generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter P, substitute the ciphertext letter C as shown below

C = E ([a , b], p) = (ap + b) mod 26

A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p ≠ q , then E(k , p) ≠ E(k , q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character.

The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then E([a , b], 0) = E([a , b], 13) = 3.

(a) Are there any limitations on the value of b ? Explain why or why not.

(b) Determine which values of a are not allowed.

A:

(a) No, Making any change in the value of *b* shifts the relationship between plaintext letters and ciphertext letters to the left or right uniformly, so that if the mapping is one-to-one it remains one-to-one.

(b) *a* must be chosen such that *a* and *26* are coprime. So, 2, 4, 6, 8, 10, 12, 13, 14, 16, 18, 20, 22, 24 are not allowed. Any value of *a* larger than 25 is equivalent to *a* mod 26.

Q 3.20 This problem explores the use of a one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.

(a) Encrypt the plaintext *sendmoremoney* with the key stream 3 11 5 7 17 21 0 11 14 8 7 13 9

(b) Using the ciphertext produced in part (a), find a key so that the ciphertext decrypts to the plaintext *cashnotneeded*

A :

(A)

– Plain text : sendmoremoney

– Key : 3 11 5 7 17 21 0 11 14 8 7 13 9

- **Encryption**

  $CT_i = (PT_i + K_i) \bmod 26$

- **Decryption**

  $PT_i = (CT_i - K_i + 26) \bmod 26$

– Now,

Assign numbers 0 to 25 to alphabets ex.- a=0, b=1 and so on..

S = 18 and key = 3

18+3 =21

21 = v

so , s will be v

– Plain text : sendmoremoney

Ans – Cipher Text : vpskdjrpawurh

(B)

– CT : *vpskdjrpawurh*

– PT : *cashnotneeded*

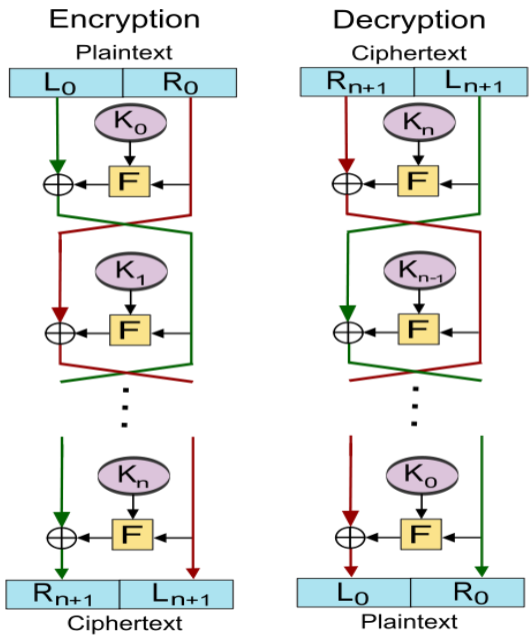
Ans – Key : 19 15 0 3 16 21 24 2 22 18 17 13 4

Q 4.2 Consider a Feistel cipher composed of sixteen rounds with a block length of 128 bits and a key length of 128 bits.

Suppose that, for a given k, the key scheduling algorithm determines values for the first eight round keys, k1 , k2 ,...,k8 , and then sets k9 = k8 , k10 = k7 , k11 = k6 , ..., k16 = k1.

Suppose you have a ciphertext c. Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you, and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.

A:



So, this is basically Encyption and Decryption process for Feistel cipher and in the key schedule k9 will be k8 and k10 will be k7 and so on because of that message is getting encrypted till K8 and from k9 it will start to be decrypted so at the end of all rounds of encoding eventually we'll get our plain text back.
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*.

Q: 7.7 For the ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed?

A:

Padding helps to make messages more secure and prevent messages from easily getting decrypted. Padding is used in cryptographic systems to increase the length of a message to the multiple of a given block size. For this padding method, the padding bits can be removed unambiguously, provided the receiver can determine that the message is indeed padded. One way to ensure that the receiver does not mistakenly remove bits from an unpadded message is to require the sender to pad every message, including messages in which the final block is already complete. For such messages, an entire block of padding is appended.

Q 7.11: "*This is a very interesting case, Watson*," Holmes said. "*The young man loves a girl, and she loves him too. However, her father is a strange fellow who insists that his would-be son- in-law must design a simple and secure protocol for an appropriate public-key cryptosystem he could use in his company's computer network*."

The young man came up with the following protocol for communication between two parties. For example, user A wishing to send message M to user B: messages exchanged are in the format (sender's name, text, receiver's name)

1. A send B the following block : (A,E(PU_B,[M,A]),B)
2. B acknowledges receipt by sending to A the following block: (B , E(PU_A , [M , B]), A)

"*You can see that the protocol is really simple. But the girl's father claims that the young man has not satisfied his call for a simple protocol, because the proposal contains a certain redundancy and can be further simplified to the following*:"

1. A sends B the block : (A,E(PU_B,M),B)
2. B acknowledges receipt by sending to A the block: (B , E(PU_A , M), A)

"*On the basis of that, the girl's father refuses to allow his daughter to marry the young man, thus making them both unhappy. The young man was just here to ask me for help*."

"*Hmm, I don't see how you can help him*." Watson was visibly unhappy with the idea that the sympathetic young man has to lose his love.

"*Well, I think I could help. You know, Watson, redundancy is sometimes good to ensure the security of protocol. Thus, the simplification the girl's father has proposed could make the new protocol vulnerable to an attack the original protocol was able to resist*," mused Holmes. "*Yes, it is so, Watson. Look, all an adversary needs is to be one of the users of the network and to be able to intercept messages exchanged between A and B. Being a user of the network, he has his own public encryption key and is able to send his own messages to A or to B and to receive theirs. With the help of the simplified protocol, he could then obtain message M user A has previously sent to B using the following procedure*:"

Complete the description.

A:

Lets assume C as adversary.

Now, as per format (sender's name, text, receiver's name)

– C grabs message sent by A to B : (A,E(PU_b,M),B)

– C sends B (C, E(PU_b, M), B)

– B acknowledges receipt by sending C [B, E(PU_c, M), C]

– C decrypts E(PU_c, M) using his secret key, and will get message M

Q 11.12:
(a) Draw figures to depict the overall *tth* logic and the compression function logic.

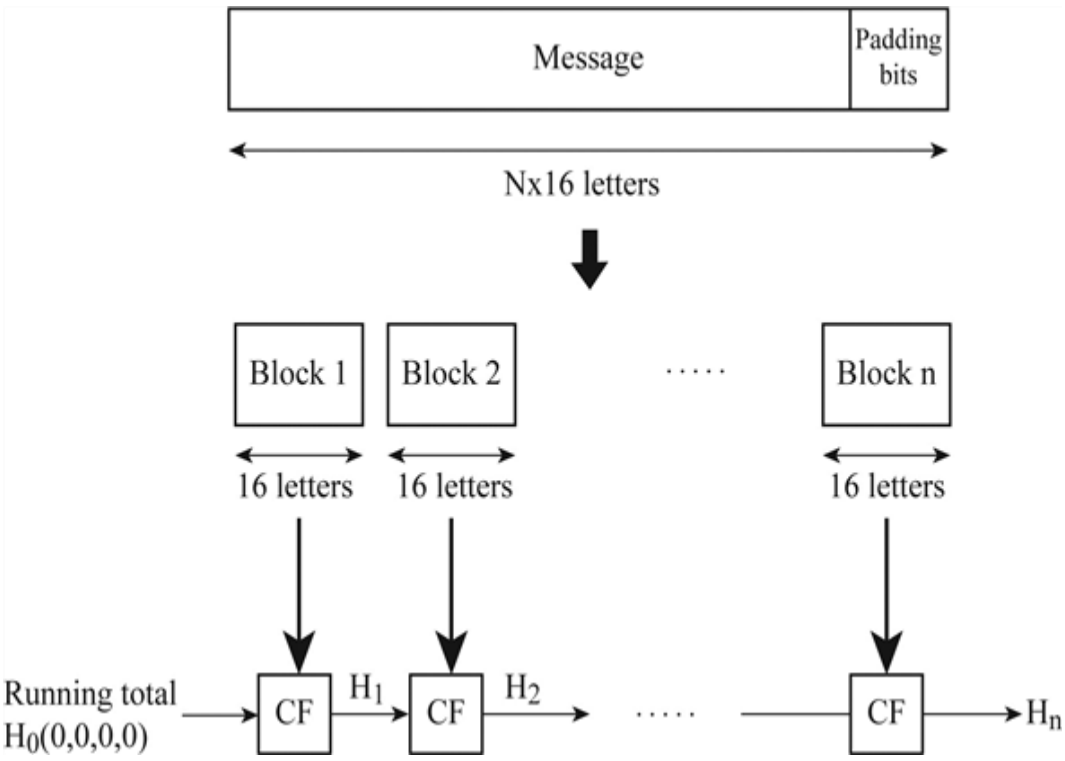
(b) Calculate the hash function for the 22-letter message "*Practice makes us perfect*"

(c) To demonstrate the weakness of tth, find a message of length 32-letter to produce the same hash.

A:

(a)

Toy tetragraph hash(tth) diagram:

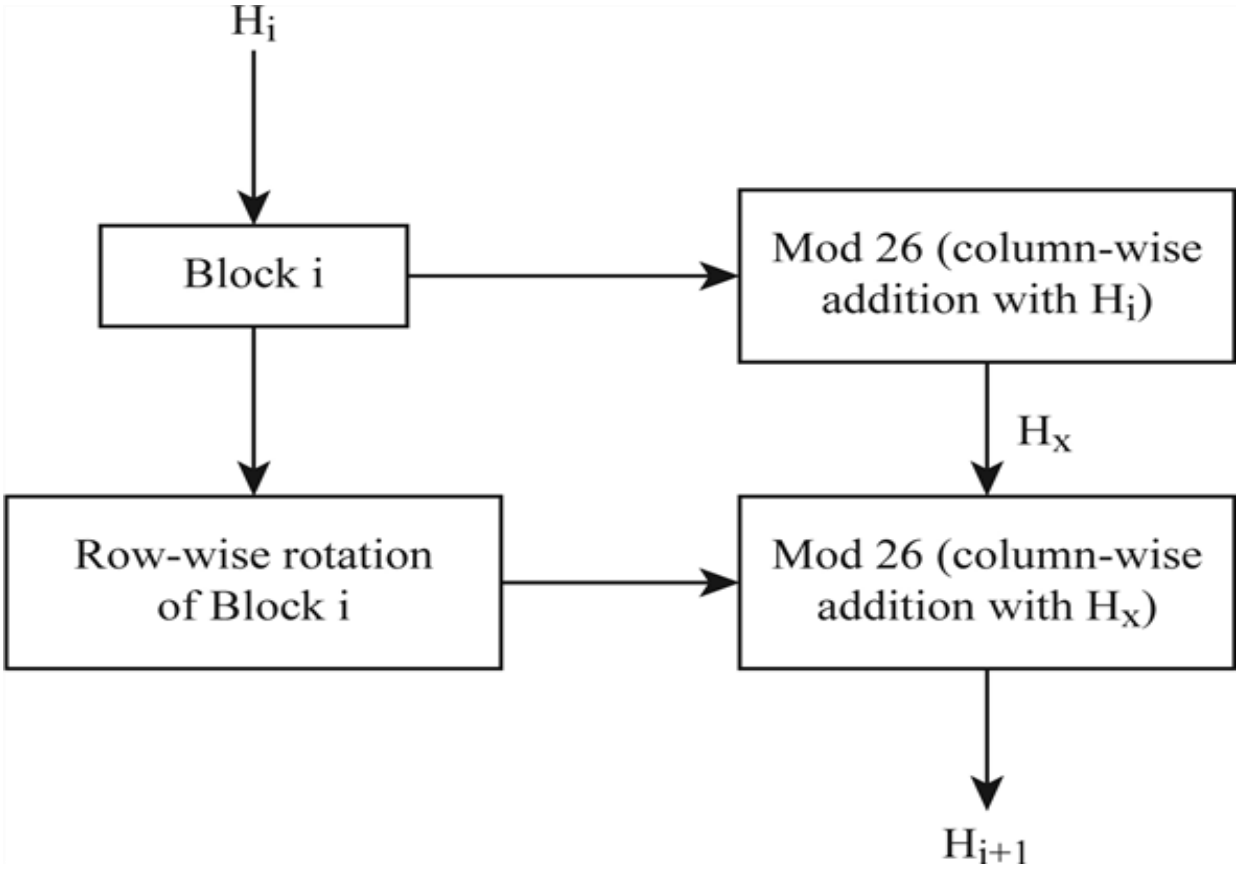


Explanation:

- Check if message length is not divisible by 16 then add padding with null to make message length divisible by 16.
- Divide message into blocks of 16 letters.
- Represent each block as a row-wise 4× 4 block of text.
- Each block is compressed using the compression function (CF) where the input of the CF is a block of 16 letters with hash value of 4 letters and it produces an output hash of 4 letters.

- Initially, the four-number running total (0,0,0,0) is the input to a compression function (CF) to process the first block.

Diagram of CF :



- Each letter in the column of 4×4 block of text is replaced with the numeric value. The Starting letter of the alphabets "A" is replaced with "0", "B" is replaced with "1" and so on.
- Each column is added with the corresponding hash value and performs the mod 26 operations and it produces a 4-letter hash value.
- Then perform row-wise rotation and each column in the rotated 4 ×4 block of text is added with the corresponding hash value of the running total and perform the mod 26 operation and it again produces a 4-letter hash value.
- This process continues for all the blocks. After the final block is processed, convert the four-number running total to letters.

(b) : Calculate the hash function for the 22-letter message "*Practice makes us perfect*"

– Total letters are 22 and it's not divisible by 16 so we have add some null values as padding.

PRACTICEMAKESUSPERFECT

Block 1

– Round-1:

P	R	A	C
T	I	C	E
M	A	K	E
S	U	S	P

15	17	0	2
19	8	2	4
12	0	10	4
18	20	18	15

Initial running total = (0,0,0,0)

New total = (12,19,4,25)

– Round-2 :

R	A	C	P
C	E	T	I
E	M	A	K
P	S	U	S

17	0	2	15
2	4	19	8
4	12	0	10
15	18	20	18

Runnig total = (12,19,4,25)

New total = (24,1,19,24)

Block 2

– Round-1 :

E	R	F	E
C	T	A	A
A	A	A	A
A	A	A	A

4	17	5	4
2	19	0	0
0	0	0	0
0	0	0	0

Runnig total = (24,1,19,24)

New total = (4,11,24,2)

– Round-2 :

R	F	E	E
A	A	C	T
A	A	A	A
A	A	A	A

17	5	4	4
0	0	2	19
0	0	0	0
0	0	0	0

Runnig total = (4,11,24,2)

New total = (21,16,4,25)

Ans = VQEZ

(c) : To demonstrate the weakness of tth, find a message of length 32-letter to produce the same hash.

– Start with all A's (0's) in your spreadsheet and manipulate the first few letters. It is easy to notice the patter. For example, the table below shows all zeros (A's) except for 3 letters:

A(0)	Q(16)	A(0)	Z(25)
F(5)	A(0)	A(0)	A(0)
A(0)	A(0)	A(0)	A(0)
A(0)	A(0)	A(0)	A(0)

– I noticed that 16 + 0 + 5 mod 26 should be 21(V), 0+16 mod 26 should be 16(Q), 25+0+5 mod 26 should be 4(E) and 25 is correct value of 25(Z).

– The hash value of "*Practice makes us perfect"* and hash value of *"AQAZFAAAAAAAAAAAAAAAAAAAAAAAAA"* are same.