HW4

Problem 1 Vigenere cipher

plaintext: goodchocolatetastesgoodandbadchocolatetastesbad
key: zheng
ciphertext: fvsqigvgbrzaiggraifmnvhntcieqigvgbrzaiggraifhzk

key length and sequence	I.C
key = 2. fsivbzigafnhtiqggragrihk vqggragrimvnceivbzigafz	Average: 0.05862977602108037 0.057971014492753624 0.05928853754940711
key = 3 fqvrirfvtegbagiz vigzgamhcqvrirfk sgbaginniigzgah	Average: 0.05992063492063492 0.041666666666666664 0.03333333333333333 0.10476190476190476
key = 4 fibiantqgarh vgrgivcibiaz svzgfhigrgik qgarmnevzgf	Average: 0.04621212121212122 0.030303030303030304 0.07575757575757576 0.06060606060606061 0.01818181818181818
key = 5 fgzrncgzrz vvaavivaak sgiihegii qbgfnqbgf irgmtirgh	**Average: 0.15333333333333335** 0.1111111111111111 0.26666666666666666 0.19444444444444445 0.1111111111111111 0.08333333333333333
key = 6 fviftgai vggmcvif sbgniggh qrrvebgz izahqrrk gainiza	Average: 0.06944444444444443 0.07142857142857142 0.07142857142857142 0.10714285714285714 0.03571428571428571 0.03571428571428571 0.09523809523809523
key = 7 fggvqzi vbrhiaf srangih qzitvgz iafcggk gimibr vgnera	Average: 0.029931972789115642 0.047619047619047616 0.0 0.0 0.047619047619047616 0.047619047619047616 0.06666666666666667 0.0

key = 8 fbatgr vricba szfiri qamezf iinqah ggviiz vghggk grnvg	Average: 0.07083333333333333 0.0 0.0 0.06666666666666667 0.0 0.06666666666666667 0.13333333333333333 0.2 0.1
key = 9 frfeaz vzmqik sanig qivgg ighvr ggnga vrtbi gacrf biizh	Average: 0.06296296296296296 0.06666666666666667 0.0 0.0 0.1 0.0 0.3 0.0 0.0 0.1
key = 10 fzngr vavva sihgi qgnbf igtrh grczz vaiak giei bfqg rmig	Average: 0.08666666666666666 0.0 0.4 0.1 0.0 0.0 0.1 0.1 0.16666666666666666 0.0 0.0

Based on the above table, the maximal average Index of Coincidence appears when key length is 5, so we will choose it for Chi-square statistics.

Key Value, Decipher Text	Chi-sq
0, fgzrncgzrz	1242.312942955559
1, efyqmbfyqy	486.2445245500102
2, dexplaexpx	619.9453364044297
3, cdwokzdwow	194.55201852664533
4, bcvnjycvnv	179.4571351900276
5, abumixbumu	135.39364821509514

6, zatlhwatlt	155.78709160844605
7, yzskgvzsks	616.8322472712566
8, xyrjfuyrjr	361.5146164013797
9, wxqietxqiq	1215.9059700164685
10, vwphdswphp	74.32594264128103
11, uvogcrvogo	71.63026030213311
12, tunfbqunfn	148.86160457939212
13, stmeaptmem	42.961456190497046
14, rsldzosldl	166.22489662929524
15, qrkcynrkck	239.45021359262142
16, pqjbxmqjbj	1102.1044529923292
17, opiawlpiai	36.60753886177783
18, nohzvkohzh	575.2974433582609
19, mngyujngyg	133.996709806115
20, lmfxtimfxf	318.71090004716206
21, klewshlewe	40.14729317947141
22, jkdvrgkdvd	175.86724193846885
23, ijcuqfjcuc	409.4789307813627
24, hibtpeibtb	68.09302232479907
25, ghasodhasa	**22.55195388695461**

The first key letter is z because chi-sq is minimal in all 26 characters. Repeating this process for another 4 times (key length = 5), then we can find a possible key value.

Problem 2

Rule 1: "Work with bytes not text strings"

Reason: for the same string S, the encryption byte of S and the internal representation of S might be different, and this can cause problems when encrypting and decrypting. This is why the authors want users to transfer their text string into bytes so that the users have the control over how the byte is interpreted.

Rule 2: "Do not put ciphertext bytes directly into a string type"

Reason: because ciphertext is in binary format, this can be corrupted in transit and mis-interpreted by the receiver. Hexadecimal format can be transferred without corruption and be converted back to binary after receiving the ciphertext.

Byte Order Mark is used to indicate the byte order of a text. This is usually useful when a machine with higher byte order is trying to decrypt text in lower byte order so that the text is not mis-understood.

It's recommended to store ciphertext in a binary file and transfer ciphertext in hex format.

Problem 3

For ECB encryption, the key is equally divided into left key and right key. We can see 1010 becomes 10 and 10 – left key is the same as the right key. We apply the XOR operation on the input: 0101 1111 0101 1110 1010 1100 0011 1010

output: 1111 0101 1111 0100 0000 0110 1001 0000

For CBC, we are using a XOR block cipher. Since C(k) will be C(k-1) XOR P(k) XOR key value, the key value will be canceled out every 2 rounds.

IV = 1110. P1 = 0101. C1 = CIPH(P1 XOR IV) = 1011 XOR 1010 = 0001

P2 = 1111, C2 = CIPH(P2 XOR C1) = CIPH(1111 XOR 0001) = 1110 XOR 1010 = 0100

P3 = 0101, C3 = CIPH(P3 XOR C2) = CIPH(0101 XOR 0100) = 0001 XOR 1010 = 1011

P4 = 1110, C4 = CIPH(P4 XOR C3) = CIPH(1110 XOR 1011) = 0101 XOR 1010 = 1111

P5 = 1010, C5 = CIPH(P5 XOR C4) = CIPH(1010 XOR 1111) = 0101 XOR 1010 = 1111

P6 = 1100, C6 = CIPH(P6 XOR C5) = CIPH(1100 XOR 1111) = 0011 XOR 1010 = 1001

P7 = 0011, C7 = CIPH(P7 XOR C6) = CIPH(0011 XOR 1001) = 1010 XOR 1010 = 0000

P8 = 1010, C8 = CIPH(P8 XOR C7) = CIPH(1010 XOR 0000) = 1010 XOR 1010 = 0000

Output: 0001 0100 1011 1111 1111 1001 0000 0000

Problem 4

For many block cipher algorithms, it's required for the input size to be multiple of the block size. When this condition fails, we use padding technique by adding a padding string to satisfy the constraint.

Five methods of padding:

1. pad with the same value N when N is the number of padding bytes. If we need to pad 4 bytes, each padding string will be 0x04 0x04 0x04 0x04.
2. pad with 0x80 following 0x00. Padding 4 bytes will be 0x80 0x00 0x00 0x00
3. pad with 0x00 except for the last one to be the number of padding bytes. Padding 4 bytes will be 0x00 0x00 0x00 0x04
4. pad with 0x00 for all. Padding 4 bytes will be 0x00 0x00 0x00 0x00
5. pad with space in hex. Padding 4 bytes will be 0x20 0x20 0x20 0x20

When the input size M is always n*b when b is the block size.