DAY - 2 : CRYPTOGRAPHY AND NETWORK SECURITY

5) Write a C program for 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: 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?

b. Determine which values of a are not allowed.

```
def egcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
```

```
            return None # modular inverse does not exist
        else:
            return x % m


def affine_encrypt(text, key):
        '''
        C = (a*P + b) % 26
        '''
        return ''.join([ chr((( key[0]*(ord(t) - ord('A')) + key[1] ) % 26)
                            + ord('A')) for t in text.upper().replace(' ', '') ])


def affine_decrypt(cipher, key):
        '''
        P = (a^-1 * (C - b)) % 26
        '''
        return ''.join([ chr((( modinv(key[0], 26)*(ord(c) - ord('A') - key[1]))
                                    % 26) + ord('A')) for c in cipher ])


def main():
        # declaring text and key
        text = input("Enter the text :")
        key = [17, 20]
```

```
    # calling encryption function
    affine_encrypted_text = affine_encrypt(text, key)

    print('Encrypted Text: {}'.format( affine_encrypted_text ))

    # calling decryption function
    print('Decrypted Text: {}'.format
    ( affine_decrypt(affine_encrypted_text, key) ))


if __name__ == '__main__':
    main()
```

RESULT :

```
Enter the text :Saveetha school of Engineering
Encrypted Text: OUNKKFJUOCJYYZYBKHSAHKKXAHS
Decrypted Text: SAVEETHASCHOOLOFENGINEERING
```

#Q5 C – code

```
#include <stdio.h>
#include <string.h>
void affineCipher(char plain[], int key[])
{
    int i, x;
```

```
        char cipher[strlen(plain)];
        for (i = 0; i < strlen(plain); i++) {
                x = plain[i] - 'a';
                x = (key[0] * x + key[1]) % 26;
                cipher[i] = x + 'a';
        }
        printf("Ciphertext: %s", cipher);
}
int main()
{
        int key[] = { 17, 20 };
        char plain[] = "twenty fifteen";
        affineCipher(plain, key);
        return 0;
}
```

6) Write a High level code for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

Enter text: ardunonano

Most frequent letter: n

Second most frequent letter: a

```
import java.util.HashMap;
import java.util.Map;
```

```
import java.util.Scanner;
public class AffineCipherBreaker {
    private static final String ALPHABET = " ardunonano";
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
        String ciphertext = read.nextLine();
        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : ciphertext.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
        }
        char mostFrequent = 'A';
        char secondMostFrequent = 'A';
        int highestFrequency = 0;
        int secondHighestFrequency = 0;
        for (char c : frequencyMap.keySet()) {
            int frequency = frequencyMap.get(c);
            if (frequency > highestFrequency) {
                secondMostFrequent = mostFrequent;
                secondHighestFrequency = highestFrequency;
                mostFrequent = c;
                highestFrequency = frequency;
            } else if (frequency > secondHighestFrequency) {
                secondMostFrequent = c;
                secondHighestFrequency = frequency;
```

```
        }
      }
      System.out.println("Most frequent letter: " + mostFrequent);
      System.out.println("Second most frequent letter: " +
secondMostFrequent);
   }
}
```

RESULT :

Most frequent letter: n

Second most frequent letter: a

7) Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.

For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal order and match this against the plaintext letters:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

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

```
#include<stdio.h>
char monocipher_encr(char);
char alpha[27][3] = { { 'a', 'c' }, { 'b', 'i' }, { 'c', 'p' }, { 'd', 'h' }, {
```

```
        'e', 'e' }, { 'f', 'r' }, { 'g', 'a' }, { 'h', 'b' }, { 'i', 'd' }, {
        'j', 'f' }, { 'k', 'g' }, { 'l', 'j' }, { 'm', 'k' }, { 'n', 'l' }, {
        'o', 'm' }, { 'p', 'n' }, { 'q', 'o' }, { 'r', 'q' }, { 's', 's' }, {
        't', 't' }, { 'u', 'u' }, { 'v', 'v' }, { 'w', 'w' }, { 'x', 'x' }, {
        'y', 'y' }, { 'z', 'z' } };
char str[20];
int main() {
    char str[20], str2[20];
    int i;
    printf("\n enter a plaintext..");
    gets(str);
    for (i = 0; str[i]; i++) {
        str2[i] = monocipher_encr(str[i]);
    }
    str2[i] = '\0';
    printf("\n Before Decryption..%s", str);
    printf("\n After Decryption..%s\n", str2);
}
char monocipher_encr(char a) {
    int i;
    for (i = 0; i < 27; i++) {
        if (a == alpha[i][0])
            break;
    }
```

```
    return alpha[i][1];
}
```

8) Write a program for Playfair matrix:

[M F H I K

U N O P Q

Z V W X Y

E L A R G

D S T B C]

Encrypt this message: Must see you over

```
key_matrix = [
    ['M', 'F', 'H', 'I', 'J', 'K'],
    ['U', 'N', 'O', 'P', 'Q', ' '],
    ['Z', 'V', 'W', 'X', 'Y', ' '],
    ['E', 'L', 'A', 'R', 'G', ' '],
    ['D', 'S', 'T', 'B', 'C', ' ']
]

# function to find the position of a letter in the key matrix
def find_position(matrix, letter):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] == letter:
```

```
            return (i, j)
    return None

# function to encrypt a message using the Playfair cipher
def playfair_encrypt(plaintext, key_matrix):
    # preprocess the plaintext to remove spaces and replace "J" with "I"
    plaintext = plaintext.upper().replace(" ", "").replace("J", "I")
    # split the plaintext into pairs of letters
    plaintext_pairs = []
    for i in range(0, len(plaintext), 2):
        if i+1 < len(plaintext) and plaintext[i] == plaintext[i+1]:
            plaintext_pairs.append((plaintext[i], "X"))
            plaintext = plaintext[:i+1] + "X" + plaintext[i+1:]
        else:
            plaintext_pairs.append((plaintext[i], plaintext[i+1] if i+1 < len(plaintext) else "X"))
    # encrypt each pair of letters
    ciphertext = ""
    for pair in plaintext_pairs:
        # find the positions of the two letters in the key matrix
        pos1 = find_position(key_matrix, pair[0])
        pos2 = find_position(key_matrix, pair[1])
        # if the two letters are in the same row, replace each letter with the letter to its right
```

```
        if pos1[0] == pos2[0]:
            ciphertext +=
key_matrix[pos1[0]][(pos1[1]+1)%len(key_matrix[pos1[0]])]
            ciphertext +=
key_matrix[pos2[0]][(pos2[1]+1)%len(key_matrix[pos2[0]])]
        # if the two letters are in the same column, replace each letter
with the letter below it
        elif pos1[1] == pos2[1]:
            ciphertext +=
key_matrix[(pos1[0]+1)%len(key_matrix)][pos1[1]]
            ciphertext +=
key_matrix[(pos2[0]+1)%len(key_matrix)][pos2[1]]
        # otherwise, replace each letter with the letter in the same row
and opposite corner of the rectangle
        else:
            ciphertext += key_matrix[pos1[0]][pos2[1]]
            ciphertext += key_matrix[pos2[0]][pos1[1]]
    return ciphertext


# example usage
plaintext = input("Enter the plain text :")
ciphertext = playfair_encrypt(plaintext, key_matrix)
print(ciphertext)
```

RESULT :

Enter the plain text :Saveetha School of Engineering

>>> TLZLADOTT OWNANHLURJULLGFPRY

9) Write a high-level code for possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.

```
import itertools
def find_keyword():
    alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'
    combinations = itertools.combinations(alphabet, 25)
    for keyword in combinations:
        matrix = [[0]*5 for _ in range(5)]
        for i, letter in enumerate(keyword):
            row = i // 5
            col = i % 5
            matrix[row][col] = letter
        valid = True
        for row in range(5):
            for col in range(5):
                if matrix[row][col] == 0:
                    valid = False
                    break
                if matrix[row][col] == 'I' or matrix[row][col] == 'J':
```

```
                matrix[row][col] = 'IJ'
            if matrix[row][col] in matrix[row][col+1:] + [matrix[i][col] for
i in range(row+1, 5)]:

                valid = False
                break
        if not valid:
            break
    if valid:
        return keyword
  return None
keyword = find_keyword()
if keyword is not None:
  print(f"The keyword is {keyword}.")
  print(f"Its approximate power of 2 is {2**(len(keyword)*5):,.0f}.")
else:
  print("No valid keyword was found.")
```

10) Write a high level code to Encrypt the message “Meet me at the usual place at ten rather than eight o clock” using the Hill cipher with the key.

9 4

5 7

a. Show your calculations and the result.

b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext.

```
# Define the key matrix
key = [[9, 4], [5, 7]]

# Define the plaintext message
plaintext = input("Enter the plain text :")
# Convert the plaintext to uppercase and remove spaces
plaintext = plaintext.upper().replace(" ", "")

# Pad the plaintext with "X" if necessary to make the length a multiple of 2
if len(plaintext) % 2 != 0:
    plaintext += "X"

# Split the plaintext into pairs of 2 letters and convert each pair to a vector
plaintext_vectors = []
for i in range(0, len(plaintext), 2):
    pair = plaintext[i:i+2]
    vector = [ord(pair[0])-65, ord(pair[1])-65]
    plaintext_vectors.append(vector)
```

```
# Multiply each plaintext vector by the key matrix to get the corresponding ciphertext vector
ciphertext_vectors = []
for vector in plaintext_vectors:
    ciphertext_vector = [(key[0][0]*vector[0] + key[0][1]*vector[1]) % 26, (key[1][0]*vector[0] + key[1][1]*vector[1]) % 26]
    ciphertext_vectors.append(ciphertext_vector)

# Convert each ciphertext vector back to a pair of letters
ciphertext = ""
for vector in ciphertext_vectors:
    pair = chr(vector[0]+65) + chr(vector[1]+65)
    ciphertext += pair

# Print the ciphertext
print(ciphertext)
```

RESULT :

Enter the plain text :Meet me at the usual place at ten rather than eight o clock

>>>UKIXUKYDROMEIWSZXWIOKUNUKHXHROAJROANQYEBTLKJEGAD

11) Write a high level language program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 1 and 26. For example, if the key is 3 19 5 . . . ,

then the first letter of the 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.

Encrypt the plaintext send more money with the key stream

9 0 1 7 23 15 21 14 11 11 2 8 9.

```
def vigenere_otp_encrypt(plaintext, key_stream):
    ciphertext = ""
    key_index = 0
    for char in plaintext:
        shift = key_stream[key_index]
        if char.isalpha():
            if char.isupper():
                ciphertext += chr((ord(char) - 65 + shift) % 26 + 65)
            else:
                ciphertext += chr((ord(char) - 97 + shift) % 26 + 97)
            key_index = (key_index + 1) % len(key_stream)
        else:
            ciphertext += char
    return ciphertext

# Example usage
plaintext = input("Enter the plain text :")
key_stream = [9, 0, 1, 7, 23, 15, 21, 14, 11, 11, 2, 8, 9]
```

```
ciphertext = vigenere_otp_encrypt(plaintext, key_stream)
print(ciphertext)
```

RESULT :

Enter the plain text :Saveetha School of Engineering

Bawlbico Dnjwxu og Lkvdbpptqwp