Day 2

1.Write a High level code 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

Program:

def generate\_cipher(keyword):

keyword = ''.join(sorted(set(keyword.upper()), key=keyword.upper().index))

return keyword + ''.join(filter(str.isalpha, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'))

def monoalphabetic\_cipher(text, cipher\_sequence):

return ''.join(cipher\_sequence[ord(char) - ord('A')] if char.isalpha() else char for char in text.upper())

keyword = input("Enter the keyword: ")

cipher\_sequence = generate\_cipher(keyword)

plaintext = input("Enter the plaintext: ")

encrypted\_text = monoalphabetic\_cipher(plaintext, cipher\_sequence)

print("Cipher Sequence:", cipher\_sequence)

print("Encrypted Text:", encrypted\_text)

Output:

Enter the keyword: hello

Enter the plaintext: saveetha

Cipher Sequence: HELOABCDEFGHIJKLMNOPQRSTUVWXYZ

Encrypted Text: OHRAAPDH

2. Write a High level codefor PT-109 American patrol boat, under the command of Lieutenant John F.

Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in

Playfair code:

KXJEY UREBE ZWEHE WRYTU HEYFS

KREHE GOYFI WTTTU OLKSY CAJPO

BOTEI ZONTX BYBNT GONEY CUZWR

GDSON SXBOU YWRHE BAAHY USEDQ

Program:

def playfair\_decrypt(key, message):

key = key.replace("J", "I")

key\_square = []

for letter in key:

if letter not in key\_square:

key\_square.append(letter)

for letter in "ABCDEFGHIKLMNOPQRSTUVWXYZ":

if letter not in key\_square:

key\_square.append(letter)

message = message.replace("J", "I")

digraphs = []

i = 0

while i < len(message):

if i == len(message) - 1 or message[i] == message[i+1]:

digraphs.append(message[i] + "X")

i += 1

else:

digraphs.append(message[i:i+2])

i += 2

plaintext = ""

for digraph in digraphs:

row1, col1 = divmod(key\_square.index(digraph[0]), 5)

row2, col2 = divmod(key\_square.index(digraph[1]), 5)

if row1 == row2:

plaintext += key\_square[row1\*5 + (col1-1)%5] + key\_square[row2\*5 + (col2-1)%5]

elif col1 == col2:

plaintext += key\_square[((row1-1)%5)\*5 + col1] + key\_square[((row2-1)%5)\*5 + col2]

else:

plaintext += key\_square[row1\*5 + col2] + key\_square[row2\*5 + col1]

return plaintext

key = input("Enter the key: ")

message = input("Enter the message: ")

plaintext = playfair\_decrypt(key, message)

print("Decrypted message:", plaintext)

Output:

Enter the key: HELLO

Enter the message: SAVEETHAUNIVESRITY

Decrypted message: ULWHORAOTPBQLRQKNT

3. Write a High level code for Playfair 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

Encrypt this message: Must see you over Cadogan West. Coming at once.

Program:

def playfair\_encrypt(key, message):

key = key.replace("J", "I")

key\_square = []

for letter in key:

if letter not in key\_square:

key\_square.append(letter)

for letter in "ABCDEFGHIKLMNOPQRSTUVWXYZ":

if letter not in key\_square:

key\_square.append(letter)

message = message.upper().replace("J", "I")

message = "".join(filter(str.isalpha, message))

digraphs = []

i = 0

while i < len(message):

if i == len(message) - 1 or message[i] == message[i+1]:

digraphs.append(message[i] + "X")

i += 1

else:

digraphs.append(message[i:i+2])

i += 2

ciphertext = ""

for digraph in digraphs:

row1, col1 = divmod(key\_square.index(digraph[0]), 5)

row2, col2 = divmod(key\_square.index(digraph[1]), 5)

if row1 == row2:

ciphertext += key\_square[row1\*5 + (col1+1)%5] + key\_square[row2\*5 + (col2+1)%5]

elif col1 == col2:

ciphertext += key\_square[((row1+1)%5)\*5 + col1] + key\_square[((row2+1)%5)\*5 + col2]

else:

ciphertext += key\_square[row1\*5 + col2] + key\_square[row2\*5 + col1]

return ciphertext

key = "MFHIKUNOPQZVWXYELARGDSTBC"

message = "Must see you over Cadogan West. Coming at once"

ciphertext = playfair\_encrypt(key, message)

print(ciphertext)

Output:

UZTBDLGZPNNWLGTGTUEROVLDBDUHFPERHWQSRZ

4. 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.

a. Now take into account the fact that some Playfair keys produce the same encryption results. How many effectively unique keys does the Playfair cipher have

Program:

import math

def calculate\_possible\_keys():

possible\_keys = math.factorial(25)

return possible\_keys

def calculate\_effectively\_unique\_keys():

effectively\_unique\_keys = calculate\_possible\_keys() / (math.factorial(2) \*\* 2)

return effectively\_unique\_keys

possible\_keys = calculate\_possible\_keys()

effectively\_unique\_keys = calculate\_effectively\_unique\_keys()

print("Number of possible keys without considering duplicates:", possible\_keys)

print("Number of effectively unique keys accounting for duplicates:", effectively\_unique\_keys)

Output:

Number of possible keys without considering duplicates: 15511210043330985984000000

Number of effectively unique keys accounting for duplicates: 3.8778025108327465e+24

5. Write a High level code to Encrypt the message “meet me at the usual place at ten rather than eight oclock” 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.

Program:

import numpy as np

def prepare\_message(message, block\_size):

message = message.replace(" ", "").upper()

while len(message) % block\_size != 0:

message += 'X'

return message

def hill\_cipher\_encrypt(message, key\_matrix):

message = prepare\_message(message, 2)

key\_matrix = np.array(key\_matrix)

cipher\_text = ""

block\_size = 2

for i in range(0, len(message), block\_size):

block = message[i:i+block\_size]

block\_vector = np.array([ord(c) - ord('A') for c in block])

encrypted\_vector = np.dot(key\_matrix, block\_vector) % 26

encrypted\_block = ''.join([chr(v + ord('A')) for v in encrypted\_vector])

cipher\_text += encrypted\_block

return cipher\_text

def hill\_cipher\_decrypt(cipher\_text, key\_matrix):

key\_matrix = np.array(key\_matrix)

key\_matrix\_inverse = np.linalg.inv(key\_matrix)

key\_matrix\_inverse = (key\_matrix\_inverse \* np.linalg.det(key\_matrix)).round()

key\_matrix\_inverse = key\_matrix\_inverse.astype(int) % 26

plain\_text = ""

block\_size = 2

for i in range(0, len(cipher\_text), block\_size):

block = cipher\_text[i:i+block\_size]

block\_vector = np.array([ord(c) - ord('A') for c in block])

decrypted\_vector = np.dot(key\_matrix\_inverse, block\_vector) % 26

decrypted\_block = ''.join([chr(v + ord('A')) for v in decrypted\_vector])

plain\_text += decrypted\_block

return plain\_text

key\_matrix = [

[9, 4],

[5, 7]

]

message = "meet me at the usual place at ten rather than eight oclock"

cipher\_text = hill\_cipher\_encrypt(message, key\_matrix)

print("Encrypted message:", cipher\_text)

decrypted\_message = hill\_cipher\_decrypt(cipher\_text, key\_matrix)

print("Decrypted message:", decrypted\_message)

Output:

Encrypted message: UKIXUKYDROMEIWSZXWIOKUNUKHXHROAJROANQYEBTLKJEGAD

Decrypted message: WQQLWQALLPQCUCAFVFAIQALLQNDALPQDLPANQGYPLEIFEIOB.

6. Write a C program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext–

ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be

mounted.

Program:

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int modInverse(int a, int m) {

a = a % m;

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1) {

return x;

}

}

return 1;

}

void encrypt(int keyMatrix[3][3], char \*input, char \*output) {

int i, j, k;

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

output[i] += keyMatrix[i][j] \* (input[j] - 'A');

}

output[i] = output[i] % 26 + 'A';

}

}

void hillCipher(char \*input, char \*key) {

int keyMatrix[3][3];

// Convert key to uppercase

for (int i = 0; i < 9; i++) {

key[i] = toupper(key[i]);

}

// Convert key to a 3x3 matrix

int k = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

keyMatrix[i][j] = key[k] - 'A';

k++;

}

}

// Check if the key is invertible

int det = keyMatrix[0][0] \* (keyMatrix[1][1] \* keyMatrix[2][2] - keyMatrix[2][1] \* keyMatrix[1][2]) -

keyMatrix[0][1] \* (keyMatrix[1][0] \* keyMatrix[2][2] - keyMatrix[2][0] \* keyMatrix[1][2]) +

keyMatrix[0][2] \* (keyMatrix[1][0] \* keyMatrix[2][1] - keyMatrix[2][0] \* keyMatrix[1][1]);

det = (det % 26 + 26) % 26;

if (det == 0 || det % 2 == 0 || det % 13 == 0) {

printf("The key is not invertible. Choose a different key.\n");

return;

}

int modInverseDet = modInverse(det, 26);

for (int i = 0; i < strlen(input); i += 3) {

char inputBlock[3] = {input[i], input[i + 1], input[i + 2]};

char outputBlock[3] = {0};

encrypt(keyMatrix, inputBlock, outputBlock);

for (int j = 0; j < 3; j++) {

printf("%c", outputBlock[j]);

}

}

printf("\n");

}

int main() {

char input[] = "HELLO";

char key[] = "GYBNQKURP";

printf("Input: %s\n", input);

printf("Key: %s\n", key);

hillCipher(input, key);

return 0;

}

Output:

Input: HELLO

Key: GYBNQKURP

RIJVS UCYJK