**CRYPTOGRAPHY AND NETWORK SECURITY**

**LAB PROGRAMS**

1. **Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.**

PROGRAM :

def caesar\_cipher(text, key):

encrypted\_text = ""

for char in text:

if char.isalpha():

shift = ord('a') if char.islower() else ord('A')

encrypted\_char = chr((ord(char) - shift + key) % 26 + shift)

encrypted\_text += encrypted\_char

else:

encrypted\_text += char

return encrypted\_text

# Example usage

plaintext =input("enter the text:")

key = int(input("enter the key value:"))

encrypted\_text = caesar\_cipher(plaintext, key)

print("Encrypted:", encrypted\_text)

OUTPUT

enter the text:hello

enter the key value:3

Encrypted: khoor

1. **Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.**

PROGRAM :

class MonoalphabeticCipher:

def \_\_init\_\_(self, key=None):

if key:

self.key = key.upper()

else:

self.key = 'ZYXWVUTSRQPONMLKJIHGFEDCBA' # Default key as reverse alphabet

def encrypt(self, plaintext):

plaintext = plaintext.upper()

ciphertext = ''

for char in plaintext:

if char.isalpha():

idx = ord(char) - ord('A')

ciphertext += self.key[idx]

else:

ciphertext += char

return ciphertext

def decrypt(self, ciphertext):

plaintext = ''

for char in ciphertext:

if char.isalpha():

idx = self.key.index(char)

plaintext += chr(idx + ord('A'))

else:

plaintext += char

return plaintext

# Example usage

key = "KJFABSWVURMTOHECXDYPLQGNIZ" # You can provide your own key here

cipher = MonoalphabeticCipher(key)

plaintext = "HELLO, WORLD!"

encrypted\_text = cipher.encrypt(plaintext)

print("Encrypted:", encrypted\_text)

decrypted\_text = cipher.decrypt(encrypted\_text)

print("Decrypted:", decrypted\_text)

OUTPUT :

Enter the text:hello woRLD

Encrypted: VBTTE GEDTA

Decrypted: HELLO WORLD

1. **Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.**

**PROGRAM:**

def generate\_playfair\_matrix(key):

key = key.replace("J", "I") # Treat 'J' as 'I' in the key

key = key.upper()

key = "".join(dict.fromkeys(key)) # Remove duplicates while maintaining order

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

matrix = [[0] \* 5 for \_ in range(5)]

key\_index = 0

for i in range(5):

for j in range(5):

if key\_index < len(key):

matrix[i][j] = key[key\_index]

key\_index += 1

else:

for letter in alphabet:

if letter not in key and letter not in [matrix[x][y] for x in range(5) for y in range(5)]:

matrix[i][j] = letter

break

return matrix

def find\_positions(matrix, char):

for i in range(5):

for j in range(5):

if matrix[i][j] == char:

return i, j

def playfair\_encrypt(plain\_text, key):

matrix = generate\_playfair\_matrix(key)

encrypted\_text = ""

plain\_text = plain\_text.upper().replace("J", "I")

pairs = [plain\_text[i:i + 2] for i in range(0, len(plain\_text), 2)]

for pair in pairs:

if len(pair) == 1:

pair += "X"

row1, col1 = find\_positions(matrix, pair[0])

row2, col2 = find\_positions(matrix, pair[1])

if row1 == row2: # Same row

encrypted\_text += matrix[row1][(col1 + 1) % 5] + matrix[row2][(col2 + 1) % 5]

elif col1 == col2: # Same column

encrypted\_text += matrix[(row1 + 1) % 5][col1] + matrix[(row2 + 1) % 5][col2]

else: # Different row and column

encrypted\_text += matrix[row1][col2] + matrix[row2][col1]

return encrypted\_text

# Example usage

plaintext = input("Enter the text:")

key = input("Enter the key:")

encrypted\_text = playfair\_encrypt(plaintext, key)

print("Encrypted:", encrypted\_text)

OUTPUT :

Enter the text:HELLO

Enter the key:NETWORK

Encrypted: FWMMTZ

1. **Write a C program for polyalphabetic substitution cipher uses a separate monoalphabetic substitution cipher for each successive letter of plaintext, depending on a key.**

PROGRAM:

class PolyalphabeticCipher:

def \_\_init\_\_(self, key):

self.key = key.upper()

def extend\_key(self, text):

extended\_key = self.key

while len(extended\_key) < len(text):

extended\_key += self.key

return extended\_key[:len(text)]

def encrypt(self, plaintext):

plaintext = plaintext.upper()

extended\_key = self.extend\_key(plaintext)

ciphertext = ''

for i in range(len(plaintext)):

if plaintext[i].isalpha():

shift = ord(extended\_key[i]) - ord('A')

encrypted\_char = chr(((ord(plaintext[i]) - ord('A') + shift) % 26) + ord('A'))

ciphertext += encrypted\_char

else:

ciphertext += plaintext[i]

return ciphertext

def decrypt(self, ciphertext):

ciphertext = ciphertext.upper()

extended\_key = self.extend\_key(ciphertext)

plaintext = ''

for i in range(len(ciphertext)):

if ciphertext[i].isalpha():

shift = ord(extended\_key[i]) - ord('A')

decrypted\_char = chr(((ord(ciphertext[i]) - ord('A') - shift) % 26) + ord('A'))

plaintext += decrypted\_char

else:

plaintext += ciphertext[i]

return plaintext

# Example usage

key = "KEY" # You can provide your own key here

cipher = PolyalphabeticCipher(key)

plaintext = "HELLO, WORLD!"

encrypted\_text = cipher.encrypt(plaintext)

print("Encrypted:", encrypted\_text)

decrypted\_text = cipher.decrypt(encrypted\_text)

print("Decrypted:", decrypted\_text)

OUTPUT :

Encrypted: RIJVS, AMBPB!

Decrypted: HELLO, WORLD!

1. **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.**
   1. **Are there any limitations on the value of b?**
   2. **Determine which values of a are not allowed.**

PROGRAM :

def gcd(a, b):

while b:

a, b = b, a % b

return a

def mod\_inverse(a, m):

for i in range(1, m):

if (a \* i) % m == 1:

return i

return None

def affine\_caesar\_encrypt(text, a, b):

encrypted\_text = ""

for char in text:

if char.isalpha():

shift = ord('A') if char.isupper() else ord('a')

encrypted\_char = chr((a \* (ord(char) - shift) + b) % 26 + shift)

encrypted\_text += encrypted\_char

else:

encrypted\_text += char

return encrypted\_text

def affine\_caesar\_decrypt(ciphertext, a, b):

mod\_inv = mod\_inverse(a, 26)

if mod\_inv is None:

return "Error: 'a' value is not valid (no modular inverse exists)"

decrypted\_text = ""

for char in ciphertext:

if char.isalpha():

shift = ord('A') if char.isupper() else ord('a')

decrypted\_char = chr((mod\_inv \* ((ord(char) - shift) - b)) % 26 + shift)

decrypted\_text += decrypted\_char

else:

decrypted\_text += char

return decrypted\_text

# Main program

plaintext = input("Enter the text:")

a = int(input("Enter the A value:"))

b = int(input("Enter the B value:"))

encrypted\_text = affine\_caesar\_encrypt(plaintext, a, b)

print("Encrypted:", encrypted\_text)

decrypted\_text = affine\_caesar\_decrypt(encrypted\_text, a, b)

print("Decrypted:", decrypted\_text)

OUTPUT:

Enter the text:HELLO

Enter the A value:3

Enter the B value:4

Encrypted: ZQLLU

Decrypted: HELLO

1. **Write a C program 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.**

PROGRAM:

def gcd(a, b):

while b:

a, b = b, a % b

return a

def mod\_inverse(a, m):

for i in range(1, m):

if (a \* i) % m == 1:

return i

return None

def affine\_decrypt(ciphertext, a, b):

mod\_inv = mod\_inverse(a, 26)

if mod\_inv is None:

return "Error: 'a' value is not valid (no modular inverse exists)"

decryption = ""

for char in ciphertext:

if char.isalpha():

shift = ord('A') if char.isupper() else ord('a')

decrypted\_char = chr(((mod\_inv \* (ord(char) - shift - b)) % 26) + shift)

decryption += decrypted\_char

else:

decryption += char

return decryption

ciphertext = "BUUKWZB BUBUBU"

most\_frequent = "B"

second\_most\_frequent = "U"

# Calculate the key values a and b

# In an affine cipher, a must be coprime to 26 (modular inverse exists)

# To find b, we can use the fact that E(a, p) - E(a, q) = a(p - q) mod 26

# Since we know B decrypts to 'E' and U decrypts to 'T', we can use these equations

a = (ord(second\_most\_frequent) - ord(most\_frequent)) \* mod\_inverse(ord('T') - ord('E'), 26) % 26

# Using a, we can find b using one of the ciphertext letters

b = (ord(second\_most\_frequent) - a \* ord('T') - ord('E')) % 26

decrypted\_message = affine\_decrypt(ciphertext, a, b)

print("Decrypted Message:")

print(decrypted\_message)

OUTPUT :

Decrypted Message:

BQQEIJB BQBQBQ

**7. Write a C program for the following ciphertext was generated using a simple substitution algorithm. 53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83(88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;**

**Decrypt this message.**

1. **As you know, the most frequently occurring letter in English is e. Therefore, the first or second (or perhaps third?) most common character in the message is likely to stand for e. Also, e is often seen in pairs (e.g., meet, fleet, speed, seen, been,**

**agree, etc.). Try to find a character in the ciphertext that decodes to e.**

1. **The most common word in English is “the.” Use this fact to guess the characters that stand for t and h.**
2. **Decipher the rest of the message by deducing additional words.**

**PROGRAM:**

def decrypt\_simple\_substitution(ciphertext, key):

decryption = ""

for char in ciphertext:

if char.isalpha():

decrypted\_char = key[char]

decryption += decrypted\_char

else:

decryption += char

return decryption

ciphertext = "53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83 " \

"(88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\* " \

";4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 " \

"(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;"

# Hints

hints = {

'†': 'E', # E is the most frequent letter

'4': 'T', # T is one of the most common letters

'8': 'H', # H often follows T

'†': 'E', # E is often seen in pairs

'3': 'R', # R is common and could follow H

'1': 'A', # A is common and could follow T

';': 'N', # N is common and could follow A

'6': 'I', # I is common and could follow A

'5': 'S', # S is common and could follow H

'0': 'O', # O is common and could follow T

'—': 'F', # F is common and could follow O

':': 'U', # U is common and could follow Q

']': 'L', # L could follow U

'(': 'W', # W is a possibility for second most common letter

'(': 'W', # W is a possibility for second most common letter

')': 'Y', # Y could follow W

'?': 'G', # G is a possibility for third most common letter

}

# Decrypt the message using the provided hints

decryption\_key = {k: v for k, v in hints.items() if k.isalpha()}

decrypted\_message = decrypt\_simple\_substitution(ciphertext, decryption\_key)

print("Decrypted Message:")

print(decrypted\_message)

OUTPUT:

Decrypted Message:

53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83 (88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\* ;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;

1. 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

PROGRAM:

def generate\_cipher\_sequence(keyword):

keyword = keyword.upper()

cipher\_sequence = list(keyword)

for letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":

if letter not in cipher\_sequence:

cipher\_sequence.append(letter)

return cipher\_sequence

def monoalphabetic\_encrypt(plaintext, cipher\_sequence):

plaintext = plaintext.upper()

encrypted\_text = ""

for char in plaintext:

if char == ' ':

encrypted\_text += ' '

else:

index = ord(char) - ord('A')

encrypted\_text += cipher\_sequence[index]

return encrypted\_text

def monoalphabetic\_decrypt(ciphertext, cipher\_sequence):

ciphertext = ciphertext.upper()

decrypted\_text = ""

for char in ciphertext:

if char == ' ':

decrypted\_text += ' '

else:

index = cipher\_sequence.index(char)

decrypted\_text += chr(index + ord('A'))

return decrypted\_text

def main():

keyword = "CIPHER"

cipher\_sequence = generate\_cipher\_sequence(keyword)

plaintext = input("Enter the plaintext: ")

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

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

decrypted\_text = monoalphabetic\_decrypt(encrypted\_text, cipher\_sequence)

print("Decrypted text:", decrypted\_text)

if \_\_name\_\_ == "\_\_main\_\_":

main()

OUTPUT:

Enter the plaintext: cipher

Encrypted text: PDNBEQ

Decrypted text: CIPHER

1. Write a C program for 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 prepare\_key(key):

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

key = "".join(sorted(set(key), key=key.index))

key\_matrix = [key[i:i+5] for i in range(0, len(key), 5)]

return key\_matrix

def find\_position(key\_matrix, letter):

for row in range(5):

for col in range(5):

if key\_matrix[row][col] == letter:

return row, col

return -1, -1

def decrypt\_playfair(ciphertext, key):

key\_matrix = prepare\_key(key)

plaintext = ""

if len(ciphertext) % 2 == 1:

ciphertext += 'X' # Add a placeholder character

for i in range(0, len(ciphertext), 2):

char1, char2 = ciphertext[i], ciphertext[i+1]

row1, col1 = find\_position(key\_matrix, char1)

row2, col2 = find\_position(key\_matrix, char2)

if row1 == row2:

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

elif col1 == col2:

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

else:

plaintext += key\_matrix[row1][col2] + key\_matrix[row2][col1]

return plaintext

# Provided ciphertext and key

ciphertext = "KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI WTTTU OLKSY CAJPO BOTEI ZONTX BYBNT GONEY CUZWR GDSON SXBOU YWRHE BAAHY USEDQ"

key = "PLAYFIREXMBCDGHKNOQSTUVWZ"

# Decrypt the message

decrypted\_message = decrypt\_playfair(ciphertext, key)

print("Decrypted Message:")

print(decrypted\_message)

OUTPUT :

Decrypted Message:

QIVMFWNLIDMVWVMDMVUXPWZUDMAYSZNIMDMVDQAYMTVZZZZUNASQFWDLTFSVDKVIMTVSKUMWGPCKZTDQORFWRNWVMUDCQNSUQMDKZUWQMCMVDPFDFWZNAEGY

1. Write a C program 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 :