**1-casear cipher**

alphabet = "abcdefghijklmnopqrstuvwxyz"

key = str(input("enter the key"))

plaintext = str(input("enter the plaintext"))

ciphertext = ""

for i in range(len(plaintext)):

index = alphabet.index(plaintext[i])

key\_index = i % len(key)

key\_char = key[key\_index]

key\_alphabet\_index = alphabet.index(key\_char)

cipher\_index = (index + key\_alphabet\_index) % 26

ciphertext += alphabet[cipher\_index]

print(ciphertext)

**2-monoalphabetic**

list\_1="abcdefghijklmnopqrstuvwxyz"

list\_2="bcdeaijklfghmnostuprrzyxvw"

plain\_text=input()

cipher\_text=""

l1=len(list\_1)

l2=len(list\_2)

for i in plain\_text:

for j in range(l1):

if i==list\_1[j]:

cipher\_text +=str(list\_2[j])

print(cipher\_text)

def monoalphabetic\_encrypt(text, key):

alphabet = 'abcdefghijklmnopqrstuvwxyz'

encrypted\_text = ''.join(key[alphabet.index(c)]

if c in alphabet

else c

for c in text)

return encrypted\_text

def monoalphabetic\_decrypt(encrypted\_text, key):

alphabet = 'abcdefghijklmnopqrstuvwxyz'

decrypted\_text = ''.join(alphabet[key.index(c)]

if c in key

else c

for c in encrypted\_text)

return decrypted\_text

key = 'phqgiumeaylnofdxjkrcvstzwb'

message = "hello world"

encrypted\_message = monoalphabetic\_encrypt(message, key)

print("Encrypted:", encrypted\_message)

decrypted\_message = monoalphabetic\_decrypt(encrypted\_message, key)

print("Decrypted:", decrypted\_message)

**3-polyalphabetic**

alphabet = "abcdefghijklmnopqrstuvwxyz"

key = str(input("enter the key"))

plaintext = str(input("enter the plaintext"))

ciphertext = ""

for i in range(len(plaintext)):

index = alphabet.index(plaintext[i])

key\_index = i % len(key)

key\_char = key[key\_index]

key\_alphabet\_index = alphabet.index(key\_char)

cipher\_index = (index + key\_alphabet\_index) % 26

ciphertext += alphabet[cipher\_index]

print(ciphertext)

**4-hill cipher**

keyMatrix = [[0] \* 3 for i in range(3)]

messageVector = [[0] for i in range(3)]

cipherMatrix = [[0] for i in range(3)]

def getKeyMatrix(key):

k = 0

for i in range(3):

for j in range(3):

keyMatrix[i][j] = ord(key[k]) % 65

k += 1

def encrypt(messageVector):

for i in range(3):

for j in range(1):

cipherMatrix[i][j] = 0

for x in range(3):

cipherMatrix[i][j] += (keyMatrix[i][x] \*messageVector[x][j])

cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):

getKeyMatrix(key)

for i in range(3):

messageVector[i][0] = ord(message[i]) % 65

encrypt(messageVector)

CipherText = []

for i in range(3):

CipherText.append(chr(cipherMatrix[i][0] + 65))

print("Ciphertext: ", "".join(CipherText))

def main():

message = "ACT"

key = "GYBNQKURP"

HillCipher(message, key)

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

main()

**5-play fair power 2**

import math

KEY\_SIZE = 25

def num\_possible\_keys():

total = 1

for i in range(KEY\_SIZE):

total \*= (KEY\_SIZE - i)

return total

keys = num\_possible\_keys()

print(f"The Playfair cipher has approximately {math.log2(keys):.2f} possible keys.")

**6-rsa**

def rsa\_encrypt(plaintext, e, n):

ciphertext = []

for char in plaintext:

encrypted\_char = pow(ord(char), e, n)

ciphertext.append(encrypted\_char)

return ciphertext

def rsa\_decrypt(ciphertext, d, n):

plaintext = []

for encrypted\_char in ciphertext:

decrypted\_char = chr(pow(encrypted\_char, d, n))

plaintext.append(decrypted\_char)

return ''.join(plaintext)

def main():

n = 2537 # Modulus

e = 17 # Public exponent

d = 2753 # Private exponent

message = "HELLO"

print("Original Message:", message)

ciphertext = rsa\_encrypt(message, e, n)

print("Encrypted:", ciphertext)

decrypted\_message = rsa\_decrypt(ciphertext, d, n)

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

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

main()

**7-sha**

import random

def has\_nonzero(state):

for row in state:

if any(row):

return True

return False

def simulate\_sha3(block\_size):

state = [[0] \* block\_size for \_ in range(block\_size)]

rounds = 0

while not has\_nonzero(state):

# Randomly change state

i, j = random.randint(0, block\_size-1), random.randint(0, block\_size-1)

state[i][j] ^= 1

rounds += 1

return rounds

def main():

print(f"Rounds needed: {simulate\_sha3(1024)}")

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

main()

**8-des3**

from Crypto.Cipher import DES3

import os

key = os.urandom(24)

iv = os.urandom(8)

cipher = DES3.new(key, DES3.MODE\_CBC, iv)

message = b"Hello, world!"

pad\_len = 8 - (len(message) % 8)

message += bytes([pad\_len]) \* pad\_len

ciphertext = iv + cipher.encrypt(message)

print(ciphertext)

**9-md5**

import hashlib

str2hash = "geeks for geeks"

result = hashlib.md5(str2hash.encode())

print("The hexadecimal equivalent of hash is : ", end ="")

print(result.hexdigest())

10-play flair

#include <stdio.h>

#include <string.h>

void encryptMessage(char matrix[5][5], char message[]) {

}

int main() {

char matrix[5][5] = {

{'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'}

};

char message[] = "Must see you over Cadogan West. Coming at once.";

encryptMessage(matrix, message);

printf("Encrypted Message: %s\n", message);

}

**10-affine ceasar**

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

int main() {

int a;

printf("Allowed values of 'a' for the affine Caesar cipher: ");

for (a = 1; a < 26; a++) {

if (gcd(a, 26) == 1) {

printf("%d ", a);

}

}

printf("\n");

}

\