Caeser cipher:

def caesar\_cipher(text, shift):

encrypted\_text = ''.join([chr((ord(char) - 97 + shift) % 26 + 97)

if char.islower()

else chr((ord(char) - 65 + shift) % 26 + 65)

if char.isupper()

else char

for char in text])

return encrypted\_text

text = input("Enter the plain text: ")

shift = int(input("key value: "))

encrypted\_text = caesar\_cipher(text, shift)

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

monoalphabetic:

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)

polyalphabetic:

def vigenere\_encrypt(plaintext, key):

key = key \* (len(plaintext) // len(key)) + key[:len(plaintext) % len(key)]

encrypted\_text = "".join(chr(((ord(p) - 97 + ord(k) - 97) % 26) + 97)

for p, k in zip(plaintext, key))

return encrypted\_text

def vigenere\_decrypt(ciphertext, key):

key = key \* (len(ciphertext) // len(key)) + key[:len(ciphertext) % len(key)]

decrypted\_text = "".join(chr(((ord(c) - 97 - ord(k) - 97) % 26) + 97)

for c, k in zip(ciphertext, key))

return decrypted\_text

key = "apple"

plaintext = "hello"

encrypted\_message = vigenere\_encrypt(plaintext, key)

print("Encrypted:", encrypted\_message)

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

print("Decrypted:", decrypted\_message)

playfair cipher:

#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);

}

2 power:

import math

# Using Stirling's approximation for factorial

def stirling\_approximation(n):

return math.sqrt(2 \* math.pi \* n) \* (n / math.e)\*\*n

# Compute approximate power of 2 for 25!

n\_factorial = stirling\_approximation(25)

x = math.log2(n\_factorial)

print(f"Approximate power of 2 for the number of keys: 2^{round(x)}")

affine cipher:

#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");

}

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)

RSA ALGORITHM:

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()

MD5:

import hashlib

str2hash = "geeks for geeks"

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

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

print(result.hexdigest())

diffiehellman:

def mod\_exp(base, exponent, modulus):

result = 1

while exponent > 0:

if exponent % 2 == 1:

result = (result \* base) % modulus

exponent //= 2

base = (base \* base) % modulus

return result

def generate\_shared\_secret(p, g, private\_key):

return mod\_exp(g, private\_key, p)

p = 23

g = 5

alice\_private\_key = 6

bob\_private\_key = 15

alice\_public\_key = generate\_shared\_secret(p, g, alice\_private\_key)

bob\_public\_key = generate\_shared\_secret(p, g, bob\_private\_key)

alice\_shared\_secret = generate\_shared\_secret(bob\_public\_key, g, alice\_private\_key)

bob\_shared\_secret = generate\_shared\_secret(alice\_public\_key, g, bob\_private\_key)

print("Alice's shared secret:", alice\_shared\_secret)

print("Bob's shared secret:", bob\_shared\_secret)

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()