Ceaser

def encrypt(*text*, *s*):

    result = ""

*# transverse the plain text*

    for i in range(len(*text*)):

        char = *text*[i]

*# Encrypt uppercase characters in plain text*

        if char.isupper():

            result += chr((ord(char) + *s* - 65) % 26 + 65)

*# Encrypt lowercase characters in plain text*

        else:

            result += chr((ord(char) + *s* - 97) % 26 + 97)

    return result

*# Define the text*

text = "CEASER CIPHER DEMO 466 "

s = 4

print("Plain Text: " + text)

print("Shift pattern: " + str(s))

print("Cipher: " + encrypt(text, s))

hmac

import hmac

*# Initialize HMAC object with a secret key and choose a stronger hash function*

digest\_maker = hmac.new(b'secret-shared-key-goes-here', *digestmod*='sha256')

*# Open the file in binary read mode*

with open('hmac\_sha.py', 'rb') as f:

    try:

*# Read the file in chunks and update the HMAC object*

        while True:

            block = f.read(1024)

            if not block:

                break

            digest\_maker.update(block)

    finally:

        f.close()

*# Calculate the HMAC digest*

digest = digest\_maker.hexdigest()

print(digest)

diffie

from random import randint

if \_\_name\_\_ == '\_\_main\_\_':

*# Both the persons will be agreed upon the public keys G and P*

*# A Prime number 'P' is taken:*

    P = 23

*# A Primitive root for P, 'G' is taken*

    G = 9

    print('The Value of P is: %d' % P)

    print('The Value of G is: %d' % G)

*# Alice will choose the private key 'a'*

    a = 4

    print('The Private key for Alice is: %d' % a)

*# Gets the generated key*

    x = pow(G, a, P)

    b = 3

    print('The Private key b for BOB is: %d' % b)

*# Gets the generated key*

    y = pow(G, b, P)

*# Secret Key for Alice*

    ka = pow(y, a, P)

*# Secret key For Bob*

    kb = pow(x, b, P)

    print('The Secret key for Alice is: %d' % ka)

    print('The Secret key for Bob is: %d' % kb)

ras

import math

*# Function to calculate the greatest common divisor (GCD)*

def gcd(*a*, *h*):

    temp = 0

    while True:

        temp = *a* % *h*

        if temp == 0:

            return *h*

*a* = *h*

*h* = temp

*# Given prime numbers p and q*

p = 3

q = 7

*# Compute n (modulus) and phi (Euler's totient function)*

n = p \* q

phi = (p - 1) \* (q - 1)

*# Choose an encryption key 'e'*

e = 2

while e < phi:

    if gcd(e, phi) == 1:

        break

    else:

        e = e + 1

*# Choose a decryption key 'd'*

k = 2

d = (1 + (k \* phi)) // e  *# Use floor division to ensure 'd' is an integer*

*# Print some information*

print("Omprakash Yadav 466")

msg = int(input("Enter id to encrypt:"))

print("Message data =", msg)

*# Encryption*

c = pow(msg, e, n)  *# This is the correct way to compute the power modulo n*

print("Encrypted data =", c)

*# Decryption*

m = pow(c, d, n)  *# This is the correct way to compute the power modulo n*

print("Original message Sent =", m)

ceignere

*# Python code to implement Vigenere Cipher*

*# This function generates the key in a cyclic manner until its length isn't equal to the length of the original text*

def generateKey(*string*, *key*):

*key* = list(*key*)

    if len(*string*) == len(*key*):

        return *(key)*

    else:

        for i in range(len(*string*) - len(*key*)):

*key*.append(*key*[i % len(*key*)])

        return ("".join(*key*))

*# This function returns the encrypted text generated with the help of the key*

def cipherText(*string*, *key*):

    cipher\_text = []

    for i in range(len(*string*)):

        x = (ord(*string*[i]) + ord(*key*[i])) % 26

        x += ord('A')

        cipher\_text.append(chr(x))

    return ("".join(cipher\_text))

*# This function decrypts the encrypted text and returns the original text*

def originalText(*cipher\_text*, *key*):

    orig\_text = []

    for i in range(len(*cipher\_text*)):

        x = (ord(*cipher\_text*[i]) - ord(*key*[i]) + 26) % 26

        x += ord('A')

        orig\_text.append(chr(x))

    return ("".join(orig\_text))

*# Driver code*

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

    string = "CHATGPT"

    keyword = "OMPRAKASH 466"

    key = generateKey(string, keyword)

    cipher\_text = cipherText(string, key)

    print("Ciphertext:", cipher\_text)

    print("Original/Decrypted Text:", originalText(cipher\_text, key))

md5

import hashlib

*# initializing string*

str2hash = "312"

*# encoding using encode() and then sending MD5*

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

*# encoding using encode() and then sending SHA256*

m = hashlib.sha256(str2hash.encode())

*# printing the equivalent hexadecimal value of MD5 hash*

print("The Hexadecimal equivalent of hash using md5 is:", *end*=" ")

print(result.hexdigest())

print("MD5 digest size:", result.digest\_size)

*# printing the equivalent hexadecimal value of SHA256 hash*

print("The Hexadecimal equivalent of hash using sha256 is:", *end*=" ")

print(m.hexdigest())

print("SHA256 digest size:", m.digest\_size)

*# You can call hash\_update() as many times as you need to iteratively update the hash*

items = [b'Hello', b'', b'world']

h = hashlib.md5()

for item in items:

    h.update(item)

print("Hash Update Demo:", h.hexdigest())

print("Iteratively updated hash digest size:", h.digest\_size)

Nmap Commands:  
nmap -sn 10.10.50.210 (ICMP scanning)  
-->nmap –sP –v <target IP address>  
  
--> nmap –sP 10.10.50.0/24  
  
-->nmap –O 10.10.50.210  (OS scan)  
  
--> nmap –sT <ip address or range>  (Full open scan)  
  
--> nmap –sS <ip address or range>  (Half open or Stealth Scan)  
  
--> nmap –sX -v <ip address or range> (Xmas Scan)  
  
--> nmap –SF <ip address or range>  (FIN Scan)  
  
--> nmap –sN <ip address or range>  (Null Scan)  
  
--> nmap –sA <ip address or range>  (ACK Scan)  
  
-->nmap –sP –PE –PA<port numbers> <starting IP/ending IP>  
For example,  
nmap –sP –PE –PA 21,23,80,3389 <192.168.0.1-50>  
  
--> nmap –sU –v <ip address or range> (UDP)