1. **XOR**

s=input("Enter a string:")

s\_list=list(s)

xor=[chr(ord(char)^27) for char in s\_list]

r\_xor=''.join(xor)

print("XOR: ",r\_xor)

and\_=[chr(ord(char)&27) for char in s\_list]

r\_and=''.join(and\_)

print("AND: ",r\_and)

1. **Caesar Cipher**

def encrypt(m,s):

res=""

for i in m:

if i.isupper():

res=res+chr((ord(i) + s - 65)%26 + 65)

elif i.isdigit():

res=res+chr((ord(i) + s - 48)%10 +48)

elif i.islower():

res=res+chr((ord(i) + s - 97)%26 + 97)

else:

res+=i

return res

def decrypt(e,s):

res=''

for i in e:

if i.isupper():

res=res+chr((ord(i) - s - 65)%26 + 65)

elif i.isdigit():

res=res+chr((ord(i) - s - 48)%10 + 48)

elif i.islower():

res=res+chr((ord(i) - s - 97)%26 +97)

else:

res+=i

return res

m=input("Enter plain text:")

s=int(input("Enter the shift value:"))

e=encrypt(m,s)

print("The encrypted message is:",e)

d=decrypt(e,s)

print("The decrypted message is:",d)

1. **Substitution cipher**
2. **Hill Cipher**
3. **Playfair Cipher**
4. **DES**

from Crypto.Cipher import DES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad,unpad

def en(pt,key):

cipher=DES.new(key,DES.MODE\_ECB)

padded\_text=pad(pt,DES.block\_size)

encrypted=cipher.encrypt(padded\_text)

return encrypted

def de(ci,key):

decipher=DES.new(key,DES.MODE\_ECB)

unpadded=decipher.decrypt(ci)

decrypted=unpad(unpadded,DES.block\_size)

return decrypted

key=get\_random\_bytes(8)

pt=input("Enter the text:")

pt=pt.encode('utf-8')

ci=en(pt,key)

print(ci)

print(de(ci,key).decode('utf-8'))

1. **AES**

import os

import pyaes

key = os.urandom(16)

aes\_encrypt = pyaes.AESModeOfOperationCTR(key)

plaintext = "hello world"

cipherText = aes\_encrypt.encrypt(plaintext.encode('utf-8'))

aes\_decrypt = pyaes.AESModeOfOperationCTR(key)

decrypted = aes\_decrypt.decrypt(cipherText).decode('utf-8')

print("Original plaintext:", plaintext)

print("Encrypted ciphertext:", repr(cipherText))

print("Decrypted plaintext:", decrypted)

1. **Blowfish**

from Crypto.Cipher import Blowfish

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

def encrypt\_text(key, plaintext):

cipher = Blowfish.new(key, Blowfish.MODE\_ECB)

padded\_plaintext = pad(plaintext, Blowfish.block\_size)

ciphertext = cipher.encrypt(padded\_plaintext)

return ciphertext

def decrypt\_text(key, ciphertext):

decipher = Blowfish.new(key, Blowfish.MODE\_ECB)

decrypted\_text = decipher.decrypt(ciphertext)

original\_plaintext = unpad(decrypted\_text, Blowfish.block\_size)

return original\_plaintext

# Generate a random 64-bit (8-byte) Blowfish key

key = get\_random\_bytes(8)

# Define the new plaintext to be encrypted

new\_plaintext = b"This is a different plaintext."

# Encrypt the plaintext

encrypted\_data = encrypt\_text(key, new\_plaintext)

# Decrypt the ciphertext

decrypted\_data = decrypt\_text(key, encrypted\_data)

# Print the results

print("Original plaintext:", new\_plaintext)

print("Encrypted ciphertext:", encrypted\_data)

print("Decrypted text:", decrypted\_data.decode("utf-8"))

\*\*BlowFish\*\*

1. **RC4**

def display(disp):

convert = [chr(char) for char in disp]

print("".join(convert))

temp = 0

ptext = input("\nEnter Plain Text: ")

key = input("Enter Key Text: ")

ptextLen = len(ptext)

keyLen = len(key)

cipher = [0] \* ptextLen

decrypt = [0] \* ptextLen

ptexti = [ord(char) for char in ptext]

keyi = [ord(char) for char in key]

s = list(range(256))

k = keyi \* (255 // keyLen + 1)

j = 0

for i in range(256):

j = (j + s[i] + k[i]) % 256

s[i], s[j]= s[j], s[i]

i = 0

j = 0

z = 0

for l in range(ptextLen):

i = (l + 1) % 256

j = (j + s[i]) % 256

s[i], s[j]= s[j], s[i]

z = s[(s[i] + s[j]) % 256]

cipher[l] = z ^ ptexti[l]

decrypt[l] = z ^ cipher[l]

print("ENCRYPTED: ", end="")

display(cipher)

print("\nDECRYPTED: ", end="")

display(decrypt)

1. **Rsa**

import math

def gcd(a, h):

if a==0:

return h

else:

return gcd(h%a,a)

p = 97

q = 3

n = p\*q

e = 2

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

while (e < phi):

# e must be co-prime to phi and

# smaller than phi.

if(gcd(e, phi) == 1):

break

else:

e = e+1

# Private key (d stands for decrypt)

# choosing d such that it satisfies

# d\*e = 1 + k \* totient

k = 2

d = (1 + (k\*phi))/e

# Message to be encrypted

msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n

c = pow(msg, e)

c = math.fmod(c, n)

print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n

m = pow(c, d)

m = math.fmod(m, n)

print("Original Message Sent = ", m)

1. **Diffie hellman**

import random

prime = 23

base = 9

private\_key\_alice = random.randint(1, prime - 1)

private\_key\_bob = random.randint(1, prime - 1)

# Function to calculate modular exponentiation (a^b % m)

def mod\_exp(a, b, m):

result = 1

a = a % m

while b > 0:

if b % 2 == 1:

result = (result \* a) % m

a = (a \* a) % m

b = b // 2

return result

# Calculate public keys for Alice and Bob

public\_key\_alice = mod\_exp(base, private\_key\_alice, prime)

#pa=pow(base,private)%prime

public\_key\_bob = mod\_exp(base, private\_key\_bob, prime)

# Exchange public keys (simulated over a network)

# In a real implementation, these values would be sent to each other securely

shared\_secret\_alice = mod\_exp(public\_key\_bob, private\_key\_alice, prime)

shared\_secret\_bob = mod\_exp(public\_key\_alice, private\_key\_bob, prime)

# Both Alice and Bob now have the same shared secret

print("Shared Secret (Alice):", shared\_secret\_alice)

print("Shared Secret (Bob):", shared\_secret\_bob)

1. **Simple columnar**
2. **Advanced columnar**
3. **Sha**

import hashlib

string=input("Enter the string:")

sha1=hashlib.sha1()

sha1.update(string.encode('utf-8'))

hashed=sha1.hexdigest()

print("SHA1 Hash:",hashed)

1. **Md5**

import hashlib

string=input("Enter the string:")

md5=hashlib.md5()

md5.update(string.encode('utf-8'))

hashed=md5.hexdigest()

print("MD5 hash: ",hashed)

1. **Euclidian**

def gcd(a,b):

if a==0:

return b

return gcd(b%a,a)

def extended\_gcd(a,b):

if a==0:

return b,0,1

g,x1,y1=extended\_gcd(b%a,a)

x=y1-(b//a)\*x1

y=x1

return g,x,y

a=int(input("Enter a number:"))

b=int(input("Enter a number:"))

g=gcd(a,b)

print("GCD = ",g)

gcd=extended\_gcd(a,b)

print("GCD = ",gcd)