HSNC UNIVERSITY, MUMBAI
KISHINCHAND CHELLARAM COLLEGE
M.Sc. Computer Science Semester 4 (SY. 2024-25)

INDEX

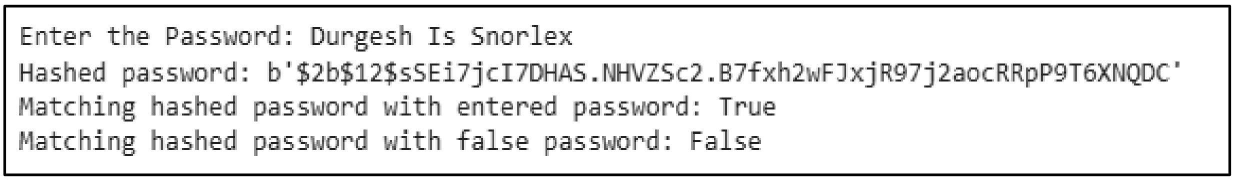
HSNC UNIVERSITY, MUMBAI
KISHINCHAND CHELLARAM COLLEGE
M.Sc. Computer Science Semester 4 (SY. 2024-25)

Practical 1	
Aim: Program to implement password salting and hashing to create secure passwords.	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 27 – 12 – 2024	Sign:

Code:

```
#pip install bcrypt

import bcrypt

# Get user password and prepare fake password for comparison
pwd = input('Enter the Password: ')
falsepwd = 'FalsePassword'

# Encode the passwords to bytes
bytepwd = pwd.encode('UTF-8')
bytefpwd = falsepwd.encode('UTF-8')

# Generate Salt
mySalt = bcrypt.gensalt()

# Hash the password
hash_val = bcrypt.hashpw(bytepwd, mySalt)
print('Hashed password:', hash_val)

# Check if entered password matches the hash
print('Matching hashed password with entered password:',
bcrypt.checkpw(bytepwd, hash_val))
print('Matching hashed password with false password:',
bcrypt.checkpw(bytefpwd, hash_val))
```

Output:



```
Enter the Password: Durgesh Is Snorlex
Hashed password: b'$2b$12$sSEi7jcI7DHAS.NHVZSc2.B7fxh2wFJxjR97j2aocRRpP9T6XNQDC'
Matching hashed password with entered password: True
Matching hashed password with false password: False
```

HSNC UNIVERSITY, MUMBAI
KISHINCHAND CHELLARAM COLLEGE
M.Sc. Computer Science Semester 4 (SY. 2024-25)

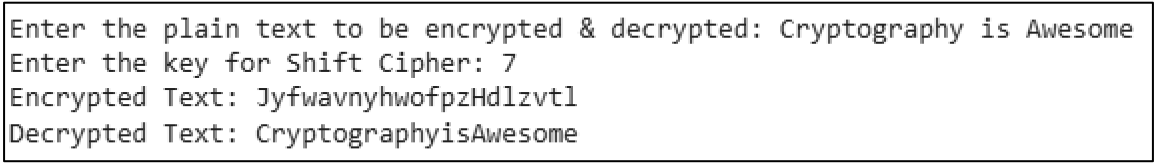
Practical 2	
Aim: Program to implement various classical ciphers- 1. Caesar Cipher 2. Vigenère Cipher 3. Affine cipher	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 27 – 12 – 2024	Sign:

1. Caesar Cipher

Code:

```
# Encryption function
def encrypt_words(plain_text, key):
  cipher_text = ''
  for word in plain_text:
    for i in word:
      val = ord(i.upper()) - 65
      enc = (val + key) % 26
      cipher_text += chr(65 + enc) if i.isupper() else chr(97 + enc)
  print('Encrypted Text:', cipher_text)
  return cipher_text

# Decryption function
def decrypt_words(cipher_text, key):
  plain_text = ''
  for word in cipher_text:
    for i in word:
      val = ord(i.upper()) - 65
      dec = (val - key) % 26
      plain_text += chr(65 + dec) if i.isupper() else chr(97 + dec)
  print('Decrypted Text:', plain_text)

# Main program
plain_text = input('Enter the plain text to be encrypted & decrypted: ').split()
key = int(input('Enter the key for Shift Cipher: '))
cipher_text = encrypt_words(plain_text, key)
decrypt_words(cipher_text, key)
```

Output:



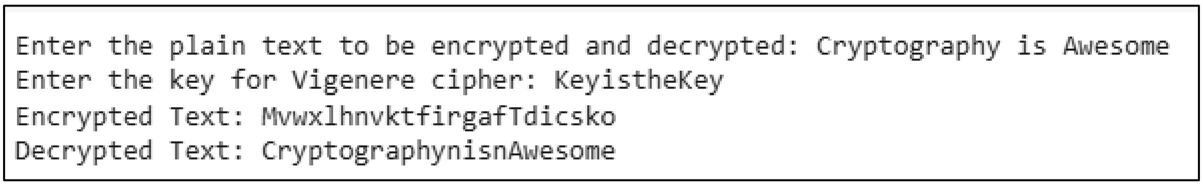
```
Enter the plain text to be encrypted & decrypted: Cryptography is Awesome
Enter the key for Shift Cipher: 7
Encrypted Text: JyfwavnyhwofpzHdlzvtl
Decrypted Text: CryptographyisAwesome
```

2. Vigenère Cipher

Code:

```
# Encryption function
def encrypt_words(plain_text, key):
    cipher_text = ''
    for word in plain_text:
        for i in word:
            val = ord(i.upper()) - 65
            enc = (val + key) % 26
            cipher_text += chr(65 + enc) if i.isupper() else chr(97 + enc)
    print('Encrypted Text:', cipher_text)
    return cipher_text

# Decryption function
def decrypt_words(cipher_text, key):
    plain_text = ''
    for word in cipher_text:
        for i in word:
            val = ord(i.upper()) - 65
            dec = (val - key) % 26
            plain_text += chr(65 + dec) if i.isupper() else chr(97 + dec)
    print('Decrypted Text:', plain_text)

# Main program
plain_text = input('Enter the plain text to be encrypted & decrypted: ').split()
key = int(input('Enter the key for Shift Cipher: '))
cipher_text = encrypt_words(plain_text, key)
decrypt_words(cipher_text, key)
```

Output:



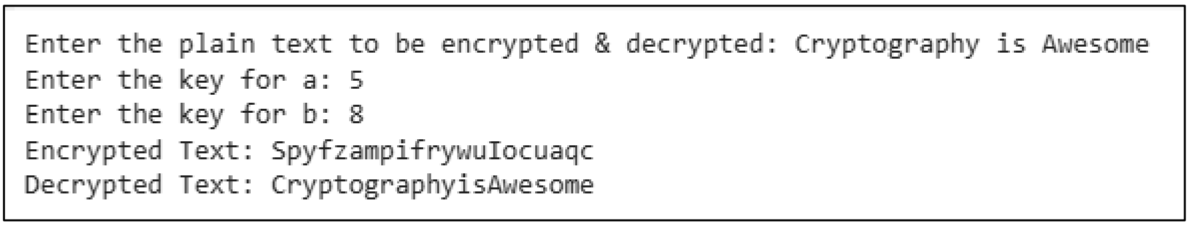
```
Enter the plain text to be encrypted and decrypted: Cryptography is Awesome
Enter the key for Vigenere cipher: KeyistheKey
Encrypted Text: MvwxlhnvktfirgafTdicsko
Decrypted Text: CryptographynisnAwesome
```

3. Affine Cipher

Code:

```
# Affine Cipher Encryption
def encrypt_words(plain_text, a, b):
    cipher_text = ''
    for word in plain_text:
        for i in word:
            base = 65 if i.isupper() else 97
            val = ord(i) - base
            enc = (a * val + b) % 26
            cipher_text += chr(base + enc)
    print('Encrypted Text:', cipher_text)
    return cipher_text

# Affine Cipher Decryption
def decrypt_words(cipher_text, a, b):
    plain_text = ''
    c = next(i for i in range(1, 26) if (a * i) % 26 == 1)  # Modular inverse
    for word in cipher_text:
        for i in word:
            base = 65 if i.isupper() else 97
            val = ord(i) - base
            dec = (c * (val - b)) % 26
            plain_text += chr(base + dec)
    print('Decrypted Text:', plain_text)

# Main
plain_text = input('Enter the plain text to be encrypted & decrypted: ').split()
a = int(input('Enter the key for a: '))
b = int(input('Enter the key for b: '))

cipher_text = encrypt_words(plain_text, a, b)
decrypt_words(cipher_text, a, b)
```

Output:



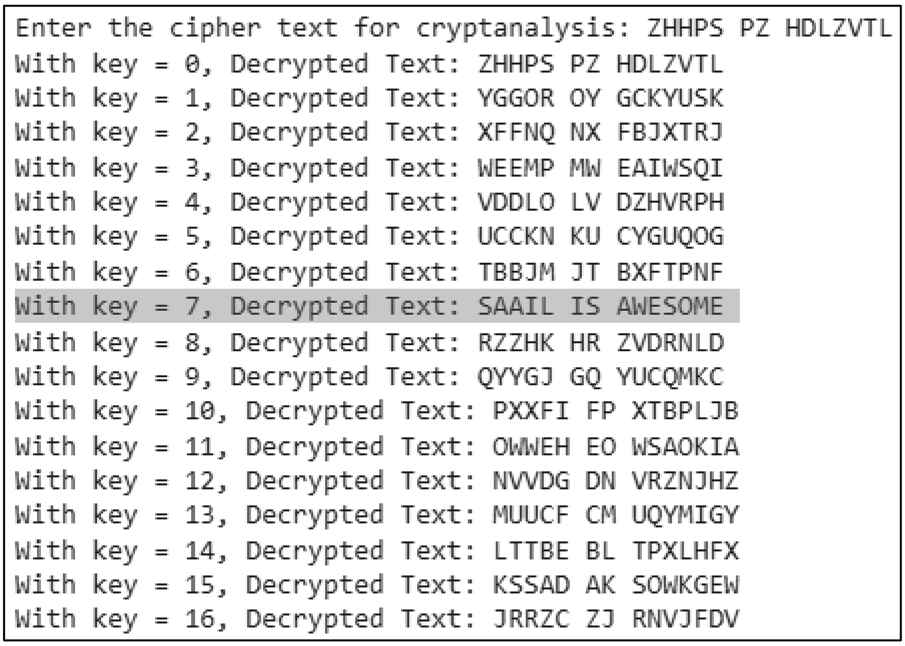
```
Enter the plain text to be encrypted & decrypted: Cryptography is Awesome
Enter the key for a: 5
Enter the key for b: 8
Encrypted Text: SpyfzampifrywuIocuaqc
Decrypted Text: CryptographyisAwesome
```

Practical 3	
Aim: Program to demonstrate cryptanalysis of Shift Cipher	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 12 – 02 – 2025	Sign:

Code:

```
def cryptanalysis():
    cipher_text = input('Enter the cipher text for cryptanalysis: ')

    for k in range(26):  # Try all possible shift values
        plain_text = ''
        for letter in cipher_text:
            if letter == ' ':
                plain_text += letter
            else:
                c = ord(letter) - 65  # Convert to 0–25 range
                e = (c - k) % 26
                plain_text += chr(e + 65)
        print(f'With key = {k}, Decrypted Text: {plain_text}')

# Run the function
cryptanalysis()
```

Output:



```
Enter the cipher text for cryptanalysis: ZHHPS PZ HDLZVTL
With key = 0, Decrypted Text: ZHHPS PZ HDLZVTL
With key = 1, Decrypted Text: YGGOR OY GCKYUSK
With key = 2, Decrypted Text: XFFNQ NX FBJXTRJ
With key = 3, Decrypted Text: WEEMP MW EAIWSQI
With key = 4, Decrypted Text: VDDLO LV DZHVRPH
With key = 5, Decrypted Text: UCCKN KU CYGUQOG
With key = 6, Decrypted Text: TBBJM JT BXFTPNF
With key = 7, Decrypted Text: SAAIL IS AWESOME
With key = 8, Decrypted Text: RZZHK HR ZVDRNLD
With key = 9, Decrypted Text: QYYGJ GQ YUCQMKC
With key = 10, Decrypted Text: PXXFI FP XTBPLJB
With key = 11, Decrypted Text: OWWEH EO WSAOKIA
With key = 12, Decrypted Text: NVVDG DN VRZNJHZ
With key = 13, Decrypted Text: MUUCF CM UQYMIGY
With key = 14, Decrypted Text: LTTBE BL TPXLHFX
With key = 15, Decrypted Text: KSSAD AK SOWKGEW
With key = 16, Decrypted Text: JRRZC ZJ RNVJFDV
```

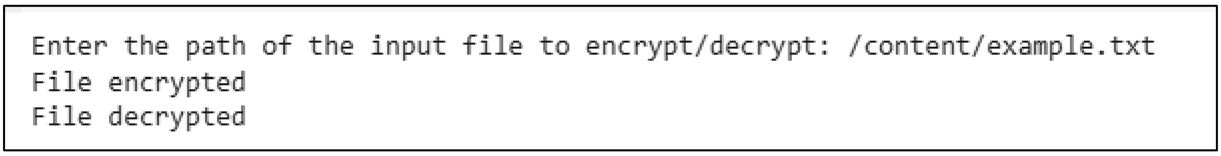
HSNC UNIVERSITY, MUMBAI
KISHINCHAND CHELLARAM COLLEGE
M.Sc. Computer Science Semester 4 (SY. 2024-25)

Practical 4	
Aim: Program to implement AES algorithm for file encryption and decryption	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 25 – 03 – 2025	Sign:

Code:

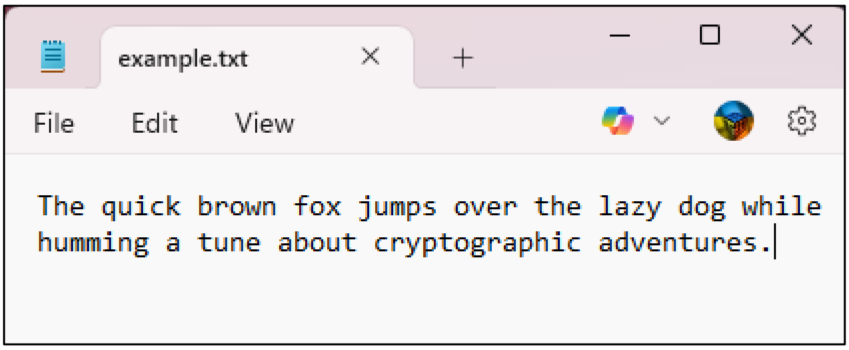
```
#pip install pycryptodome

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def encrypt_file(input_file, output_file, key):
    with open(input_file, 'rb') as f:
        data = f.read()
    cipher = AES.new(key, AES.MODE_CBC)
    encrypted = cipher.encrypt(pad(data, AES.block_size))
    with open(output_file, 'wb') as f:
        f.write(cipher.iv + encrypted)

def decrypt_file(input_file, output_file, key):
    with open(input_file, 'rb') as f:
        iv = f.read(16)
        encrypted = f.read()
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = unpad(cipher.decrypt(encrypted), AES.block_size)
    with open(output_file, 'wb') as f:
        f.write(decrypted)

def main():
    # Generate a random 16-byte (128-bit) AES key
    key = get_random_bytes(16)
    input_file = input("Enter the path of the input file to encrypt/decrypt: ").strip()
    encrypt_file(input_file, 'encrypt.txt', key)
    print(f"File encrypted ")
    decrypt_file('encrypt.txt', 'decrypt.txt', key)
    print(f"File decrypted ")

if __name__ == "__main__":
    main()
```

Output:



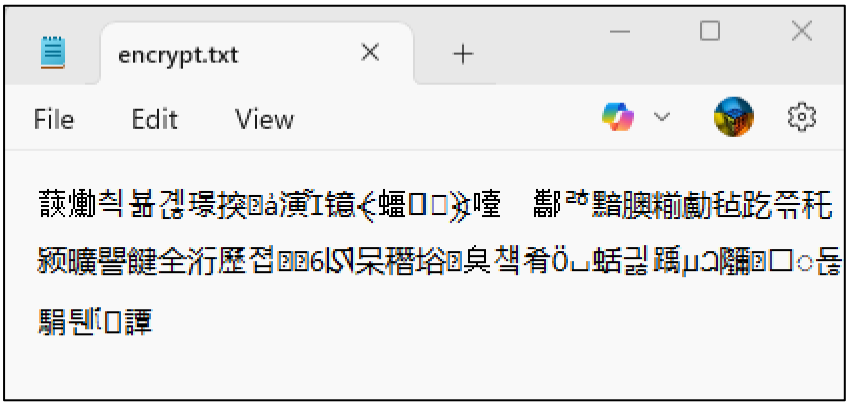
```
Enter the path of the input file to encrypt/decrypt: /content/example.txt
File encrypted
File decrypted
```

Input:



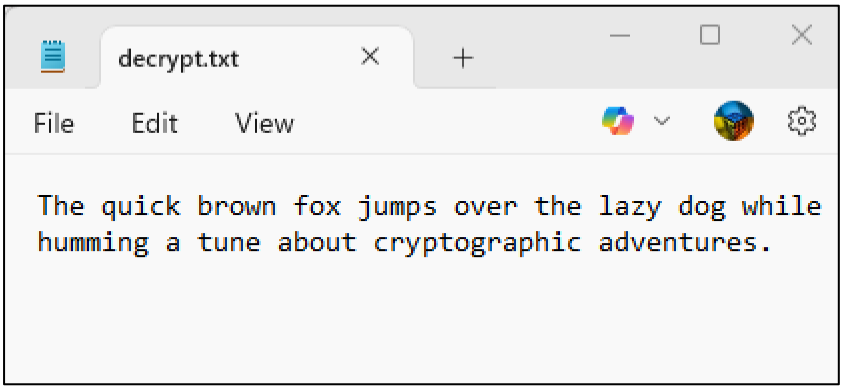
The quick brown fox jumps over the lazy dog while humming a tune about cryptographic adventures.

Encryption:



蒢爋칰븖걼璟挨▯ȧ演Ｉ镱《蝹□□》噎　鄻ᆬ黯腴糒勴毡敁岙秏
颍矌譼鍵全洐歷접▯▯6㔾呆穡埆▯臭챀肴ö␣蛞릙 䠠µↄ斸▯□○등
騙퇟□譚

Decryption:



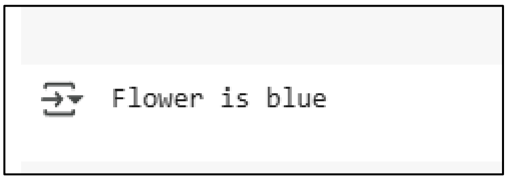
The quick brown fox jumps over the lazy dog while humming a tune about cryptographic adventures.

Practical 5	
Aim: Program to implement Steganography for hiding messages inside the image file.	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 12 – 02 – 2025	Sign:

Code:

```
#pip install stegano

from stegano import lsb

# Hide the message
steg = lsb.hide('/content/flower.png', 'Flower is blue')
steg.save('/content/flower-secret.png')

# Retrieve the hidden message
msg = lsb.reveal('/content/flower-secret.png')
print(msg)
```

Output:



```
Flower is blue
```

Image without hidden message

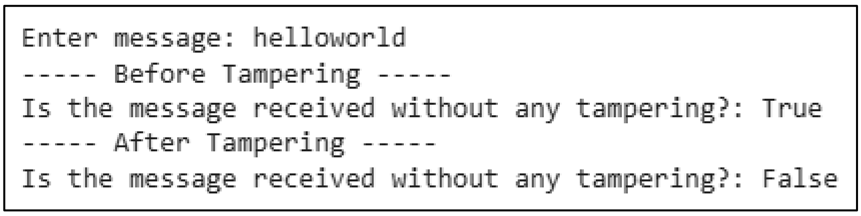
Image with hidden message

Practical 6	
Aim: Program to implement HMAC for signing messages.	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 26 – 03 – 2025	Sign:

Code:

```
import hmac
import hashlib
import secrets

# Initial sent message
sent_msg = input("Enter message: ")
key = secrets.token_bytes(100)  # Secret key generation

# Generate HMAC for sent message
s_md_1 = hmac.new(key=key, msg=sent_msg.encode(), digestmod=hashlib.md5)
init_msg_digest = s_md_1.hexdigest()

# Simulate receiving the same message
received = sent_msg
r_md_1 = hmac.new(key=key, msg=received.encode(), digestmod=hashlib.md5)
recv_msg_digest = r_md_1.hexdigest()

# Comparing sent and received message digests
print("----- Before Tampering -----")
print("Is the message received without any tampering?:",
    hmac.compare_digest(init_msg_digest, recv_msg_digest))

# Simulate tampering the message
tampered_msg = sent_msg[1:]  # remove first character (just for testing tamper)
md_2 = hmac.new(key=key, msg=tampered_msg.encode(), digestmod=hashlib.md5)
tampered_msg_digest = md_2.hexdigest()

# Comparing tampered message digest with original
print("----- After Tampering -----")
print("Is the message received without any tampering?:",
    hmac.compare_digest(init_msg_digest, tampered_msg_digest))
```

Output:



```
Enter message: helloworld
----- Before Tampering -----
Is the message received without any tampering?: True
----- After Tampering -----
Is the message received without any tampering?: False
```

HSNC UNIVERSITY, MUMBAI
KISHINCHAND CHELLARAM COLLEGE
M.Sc. Computer Science Semester 4 (SY. 2024-25)

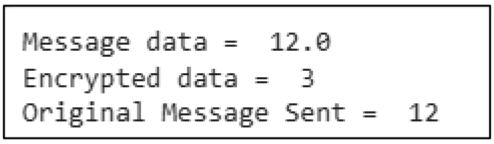
Practical 7	
Aim: Program to implement- 1. ElGamal Cryptosystem 2. Euclidean Algorithm	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 26 – 03 – 2025	Sign:

1. ElGamal Cryptosystem

Code:

```
import math
def gcd(a, b):
  while b: a, b = b, a % b
  return a
# Prime numbers
p = 3
q = 7

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

# Choose e such that 1 < e < phi and gcd(e, phi) = 1
e = 2
while e < phi and gcd(e, phi) != 1: e += 1

# Calculate d, the modular inverse of e
d = next(i for i in range(1, phi) if (e * i) % phi == 1)
# Message
msg = 12.0
print("Message data = ", msg)

# Encryption: c = (msg ^ e) % n
c = pow(int(msg), e, n)
print("Encrypted data = ", c)

# Decryption: m = (c ^ d) % n
m = pow(c, d, n)
print("Original Message Sent = ", m)
```

Output:



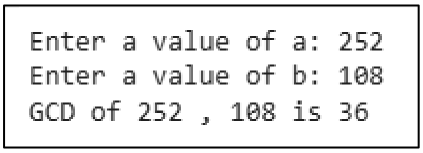
```
Message data =  12.0
Encrypted data =  3
Original Message Sent =  12
```

2. Euclidean Algorithm

Code:

```
def gcd(a, b):
    temp = 0
    while True:
        temp = a % b
        if temp == 0:
            return b
        a = b
        b = temp

# User Input
a = int(input("Enter a value of a: "))
b = int(input("Enter a value of b: "))

# Output GCD
print("GCD of", a, ",", b, "is", gcd(a, b))
```

Output:



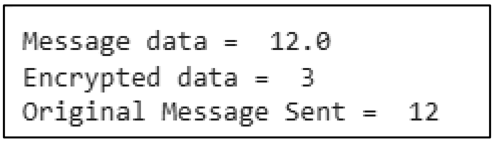
```
Enter a value of a: 252
Enter a value of b: 108
GCD of 252 , 108 is 36
```

Practical 8	
Aim: Program to implement RSA Encryption/ Decryption	
Name: Apurva Donde	Roll No: KSPMSCCS002
Performance date: 26 – 03 – 2025	Sign:

Code:

```
import math
# GCD function using Euclidean Algorithm
def gcd(a, b):
   while b: a, b = b, a % b
   return a
# Prime numbers
p = 3
q = 7

# Calculate n and phi
n = p * q
phi = (p - 1) * (q - 1)

# Choose e such that 1 < e < phi and gcd(e, phi) = 1
e = 2
while e < phi and gcd(e, phi) != 1: e += 1

# Calculate d, the modular inverse of e
d = next(i for i in range(1, phi) if (e * i) % phi == 1)
# Message
msg = 12.0
print("Message data = ", msg)

# Encryption: c = (msg ^ e) % n
c = pow(int(msg), e, n)
print("Encrypted data = ", c)

# Decryption: m = (c ^ d) % n
m = pow(c, d, n)
print("Original Message Sent = ", m)
```

Output:



```
Message data =  12.0
Encrypted data =  3
Original Message Sent =  12
```