VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Aditi Patra

21BIT0125

L55 + 56

LAB3

Digital Envelope

CODE:

```
#Digital Envelope


import random


print("Aditi Patra 21BIT0125")


# Key Generation

def generate_keypair(p, q):

    n = p * q

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


    e = random.randrange(2, phi)  # Start from 2 instead of phi
```

```
    while gcd(e, phi) != 1:
        e = random.randrange(2, phi)  # Start from 2 instead of phi

    d = modular_inverse(e, phi)

    return ((e, n), (d, n))

# Transfer Encrypted Message to Digital Envelope (Creation of Digital Envelope)
def encrypt(message, public_key):
    e, n = public_key
    encrypted_msg = pow(message, e, n)
    return encrypted_msg

def decrypt(encrypted_msg, private_key):
    d, n = private_key
    decrypted_msg = pow(encrypted_msg, d, n)
    return decrypted_msg

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        gcd, x, y = extended_gcd(b % a, a)
        return (gcd, y - (b // a) * x, x)
```

```
def modular_inverse(a, m):
    gcd, x, _ = extended_gcd(a, m)
    if gcd == 1:
        return x % m
    else:
        raise ValueError("Modular inverse does not exist.")

# Check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

# Take input for p and q, ensure they are prime numbers
p = int(input("Enter a prime number for p: "))
while not is_prime(p):
    p = int(input("Invalid input. Enter a prime number for p: "))

q = int(input("Enter a prime number for q: "))
while not is_prime(q):
    q = int(input("Invalid input. Enter a prime number for q: "))

public_key, private_key = generate_keypair(p, q)

m = int(input("Enter a number as the message: "))
```

```
# Generate random symmetric key
k = random.randint(2**2048, 2**4096)  # Generate a random key between 2 and n-1

# Encrypt the message using the public key
ct1 = encrypt(m, public_key)

# Encrypt the symmetric key using the public key
r = pow(k, public_key[0], public_key[1])
ct2 = encrypt(r, public_key)

# Store ct1 in a file
with open("ct1.txt", "w") as file:
    file.write(str(ct1))

# Store ct2 in a file
with open("ct2.txt", "w") as file:
    file.write(str(ct2))

# Decrypt the symmetric key using the private key
dec = decrypt(ct2, private_key)

# Obtain the original message by decrypting ct1 with dec
nm = decrypt(ct1, private_key)

print("Symmetric Key (k):", k)
print("Encrypted Symmetric Key (r):", r)
print("Encrypted Message (ct1):", ct1)
print("Decrypted Symmetric Key:", dec)
```

print("Decrypted Message (nm):", nm)

CODE SNIPPET:

DigitalEnvelope.py - C:/Users/patra/Desktop/books/5th sem/infosec/LAB 3/DigitalEnvelope.py (3.9.6)

File Edit Format Run Options Window Help

```
#Digital Envelope

import random

print("Aditi Patra 21BIT0125")

# Key Generation
def generate_keypair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)

    e = random.randrange(2, phi)  # Start from 2 instead of phi
    while gcd(e, phi) != 1:
        e = random.randrange(2, phi)  # Start from 2 instead of phi

    d = modular_inverse(e, phi)

    return ((e, n), (d, n))

# Transfer Encrypted Message to Digital Envelope (Creation of Digital Envelope)
def encrypt(message, public_key):
    e, n = public_key
    encrypted_msg = pow(message, e, n)
    return encrypted_msg

def decrypt(encrypted_msg, private_key):
    d, n = private_key
    decrypted_msg = pow(encrypted_msg, d, n)
    return decrypted_msg

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        gcd, x, y = extended_gcd(b % a, a)
        return (gcd, y - (b // a) * x, x)
```

```python
def modular_inverse(a, m):
    gcd, x, _ = extended_gcd(a, m)
    if gcd == 1:
        return x % m
    else:
        raise ValueError("Modular inverse does not exist.")

# Check if a number is prime
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

# Take input for p and q, ensure they are prime numbers
p = int(input("Enter a prime number for p: "))
while not is_prime(p):
    p = int(input("Invalid input. Enter a prime number for p: "))

q = int(input("Enter a prime number for q: "))
while not is_prime(q):
    q = int(input("Invalid input. Enter a prime number for q: "))

public_key, private_key = generate_keypair(p, q)

m = int(input("Enter a number as the message: "))

# Generate random symmetric key
k = random.randint(2**2048, 2**4096)  # Generate a random key between 2 and n-1

# Encrypt the message using the public key
ct1 = encrypt(m, public_key)

# Encrypt the symmetric key using the public key
r = pow(k, public_key[0], public_key[1])
ct2 = encrypt(r, public_key)

# Store ct1 in a file
with open("ct1.txt", "w") as file:
    file.write(str(ct1))
```

```python
# Store ct2 in a file
with open("ct2.txt", "w") as file:
    file.write(str(ct2))

# Decrypt the symmetric key using the private key
dec = decrypt(ct2, private_key)

# Obtain the original message by decrypting ct1 with dec
nm = decrypt(ct1, private_key)

print("Symmetric Key (k):", k)
print("Encrypted Symmetric Key (r):", r)
print("Encrypted Message (ct1):", ct1)
print("Decrypted Symmetric Key:", dec)
print("Decrypted Message (nm):", nm)
```

CT1:

CT2:

ct2 - Notepad

File Edit Format View Help

154

OUTPUT:

```
= RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 3/DigitalEnvelope.py
Aditi Patra 21BIT0125
Enter a prime number for p: 11
Enter a prime number for q: 5
Enter a number as the message: 10
Symmetric Key (k): 85391229193640386054489321162868882575204251325428910646920695914295052209045943267724521474455744811237798582392391578930071176933391090602222880100634551292219360936799791229823609144709123790479728230038958742495196700972952792921765901181594102166327263354467129876144080201246901734658702049777147722150053177440210563206736359601231808391473224849718645311210230908604218523289670745088340006442888304999200146049150678824003976228869506403590064331840113358841724837840379873583115979732000348014099482348058526064257498255252096557356231921265731243158879060434709205975841650189043250709140592304340593517109177251258509450551653390309459117960108241958700609134007631452389527444507625144472358126875919188967785182784023565137448783924055049780218496353699720421962073935197489612574853005409515966895456724521455081779793042104480383467524466469367517517241505829848351300759455781831332381652655201682750288073811817725541231718555827696490705649760753481656340505958825498898669153082235855960013481276550989273016973915552296338056827925707974935479730860703771545931236974698450968703711100868789267718533755949735329240723991016327173187064879139506170838087629701901323036346145930763853854400686870386038754108975
Encrypted Symmetric Key (r): 5
Encrypted Message (ct1): 10
Decrypted Symmetric Key: 5
Decrypted Message (nm): 10
>>>
```

```
= RESTART: C:/Users/patra/Desktop/books/5th sem/infosec/LAB 3/DigitalEnvelope.py
Aditi Patra 21BIT0125
Enter a prime number for p: 17
Enter a prime number for q: 23
Enter a number as the message: 42
Symmetric Key (k): 67470200562017589936673070958248955517835301175567925558868700636353629451115075227143387008761355560684743969189453734445720778972825835052083684855229783390695502356850522850798724593788919549031930383905532876837421787553852413837515658911004449049498381991111165332035682591035774317962915439230088511296366646403695506908383815328865925677013986479148055020014075063696940625042388237663106913832831940940772737821421183608417985958934486682555947940801597854111487864699555819966885890012944908091690826921336534655962237763424563663927302669119169419447770122552470342511391493495909137487911586979882691773503409038714575490162282634789109451007486213191163426253177759106071649050663565458299160151162864120056815228008902551232576371921352941765354986216407371041001335818164515510324622699415682716859698629761219041929555376953260281718450188346321099307006678288830098352915990745581288279353606619629160763319395883354339472672527440274525575523860995385015343422355964875707462444359900397952386156980715961622942828591893702519531035651418404516339089863500260012423470 2
Encrypted Symmetric Key (r): 239
Encrypted Message (ct1): 281
Decrypted Symmetric Key: 239
Decrypted Message (nm): 42
>>>
```