

Network Security and Forensics

Lab Session 7

Submitted To:- Submitted By:-

Dr. Lokesh Chauhan Sir Saloni Rangari

M.Tech. AIDS

**Assignment 1: Write a program to calculate the mod exponent of a big number.**

def mod\_exponent(base, exp, mod):

result = 1

base = base % mod

while exp > 0:

if (exp % 2) == 1: # If exp is odd

result = (result \* base) % mod

exp = exp >> 1 # Divide exp by 2

base = (base \* base) % mod

return result

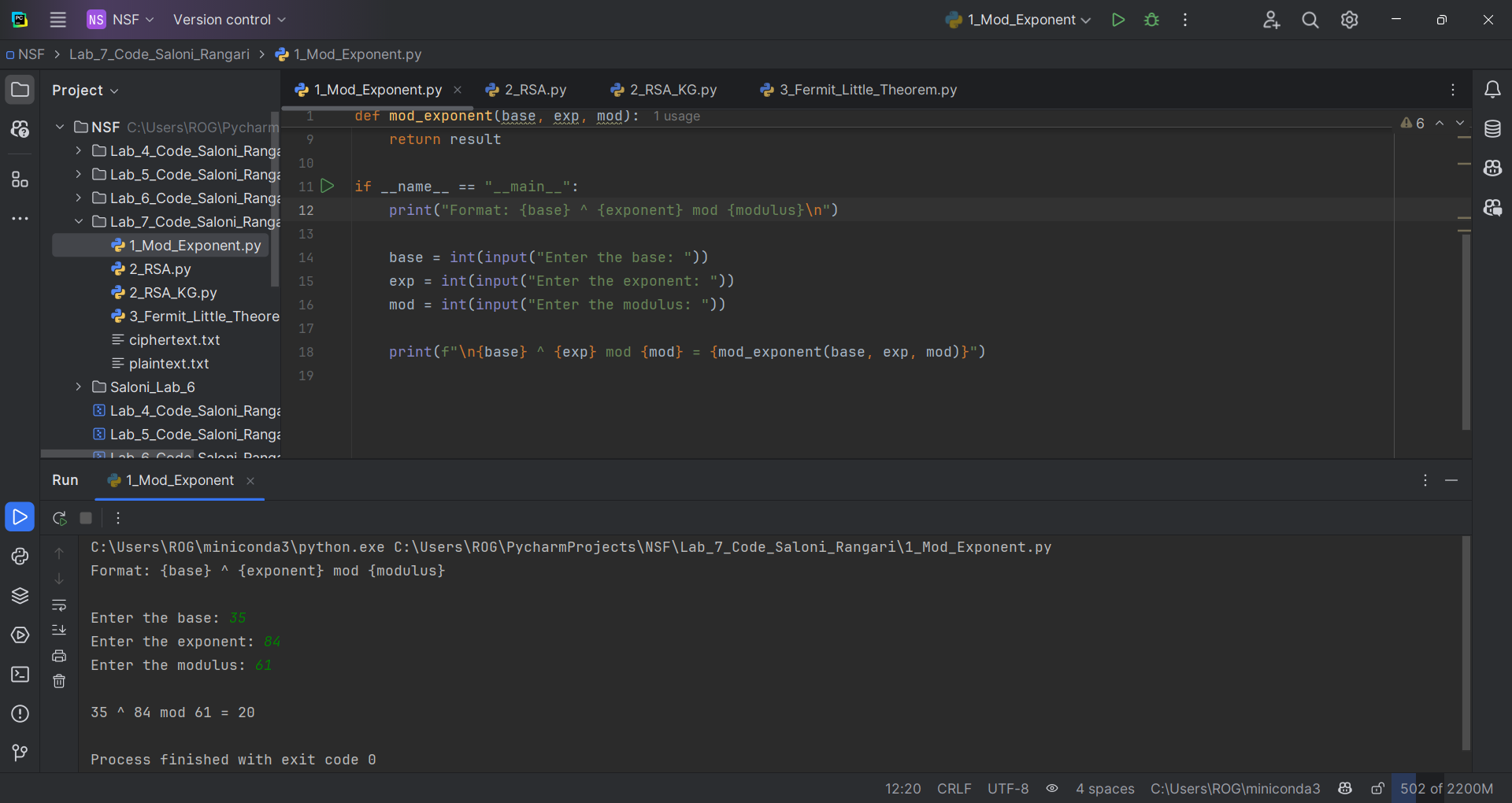
if \_\_name\_\_ == "\_\_main\_\_":

base = int(input("Enter the base: "))

exp = int(input("Enter the exponent: "))

mod = int(input("Enter the modulus: "))

print(f"{base}^{exp} mod {mod} = {mod\_exponent(base, exp, mod)}")

**Output:**

**Assignment 2: Write a program to demonstrate RSA algorithm by taking input plaintext from a file and produce output file into ciphertext file.**

import random

from sympy import isprime

def generate\_prime\_candidate(length):

p = random.getrandbits(length)

return p | (1 << length - 1) | 1 # Ensure p is odd and has the correct length

def generate\_prime\_number(length):

p = 4

while not isprime(p):

p = generate\_prime\_candidate(length)

return p

def gcd(a, b):

while b:

a, b = b, a % b

return a

def multiplicative\_inverse(e, phi):

d\_old, d\_new = 0, 1

r\_old, r\_new = phi, e

while r\_new > 0:

quotient = r\_old // r\_new

d\_old, d\_new = d\_new, d\_old - quotient \* d\_new

r\_old, r\_new = r\_new, r\_old - quotient \* r\_new

if r\_old > 1:

raise Exception('No modular inverse')

if d\_old < 0:

d\_old += phi

return d\_old

def rsa\_keypair(length):

p = generate\_prime\_number(length)

q = generate\_prime\_number(length)

n = p \* q

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

e = random.randrange(2, phi)

while gcd(e, phi) != 1:

e = random.randrange(2, phi)

d = multiplicative\_inverse(e, phi)

return ((e, n), (d, n)) # Public key and private key

def encrypt(plaintext, pubkey):

e, n = pubkey

plaintext\_int = int.from\_bytes(plaintext.encode(), 'big')

ciphertext\_int = pow(plaintext\_int, e, n)

return ciphertext\_int

def decrypt(ciphertext\_int, privkey):

d, n = privkey

plaintext\_int = pow(ciphertext\_int, d, n)

plaintext\_bytes = plaintext\_int.to\_bytes((plaintext\_int.bit\_length() + 7) // 8, 'big')

return plaintext\_bytes.decode()

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

length = int(input("Enter the key length (in bits): "))

public\_key, private\_key = rsa\_keypair(length)

with open('plaintext.txt', 'r') as file:

plaintext = file.read().strip()

ciphertext\_int = encrypt(plaintext, public\_key)

with open('ciphertext.txt', 'w') as file:

file.write(str(ciphertext\_int))

print(f"Ciphertext: {ciphertext\_int}")

decrypted\_text = decrypt(ciphertext\_int, private\_key)

print(f"Decrypted Text: {decrypted\_text}")

**Output:A screenshot of a computer

Description automatically generated**

**Assignment 3:  Write a program to implement Fermit Little Theorem.**

def fermat\_little\_theorem(a, p):

if p <= 1 or not isprime(p):

raise ValueError("p must be a prime number greater than 1.")

return mod\_exponent(a, p - 1, p)

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

a = int(input("Enter an integer a: "))

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

result = fermat\_little\_theorem(a, p)

print(f"{a}^{p-1} mod {p} ≡ {result} (according to Fermat's Little Theorem)")

**A screenshot of a computer program

Description automatically generatedOutput:**