

Network Security and Forensics

Lab Session 8

Submitted To:- Submitted By:-

Dr. Lokesh Chauhan Sir Saloni Rangari

M.Tech. AIDS

**Assignment 1: Write a program to demonstrate Deffie Hellman algorithm by taking input of global parameters and private keys from a file and produce Public Key and Shares Keys into the output file.**

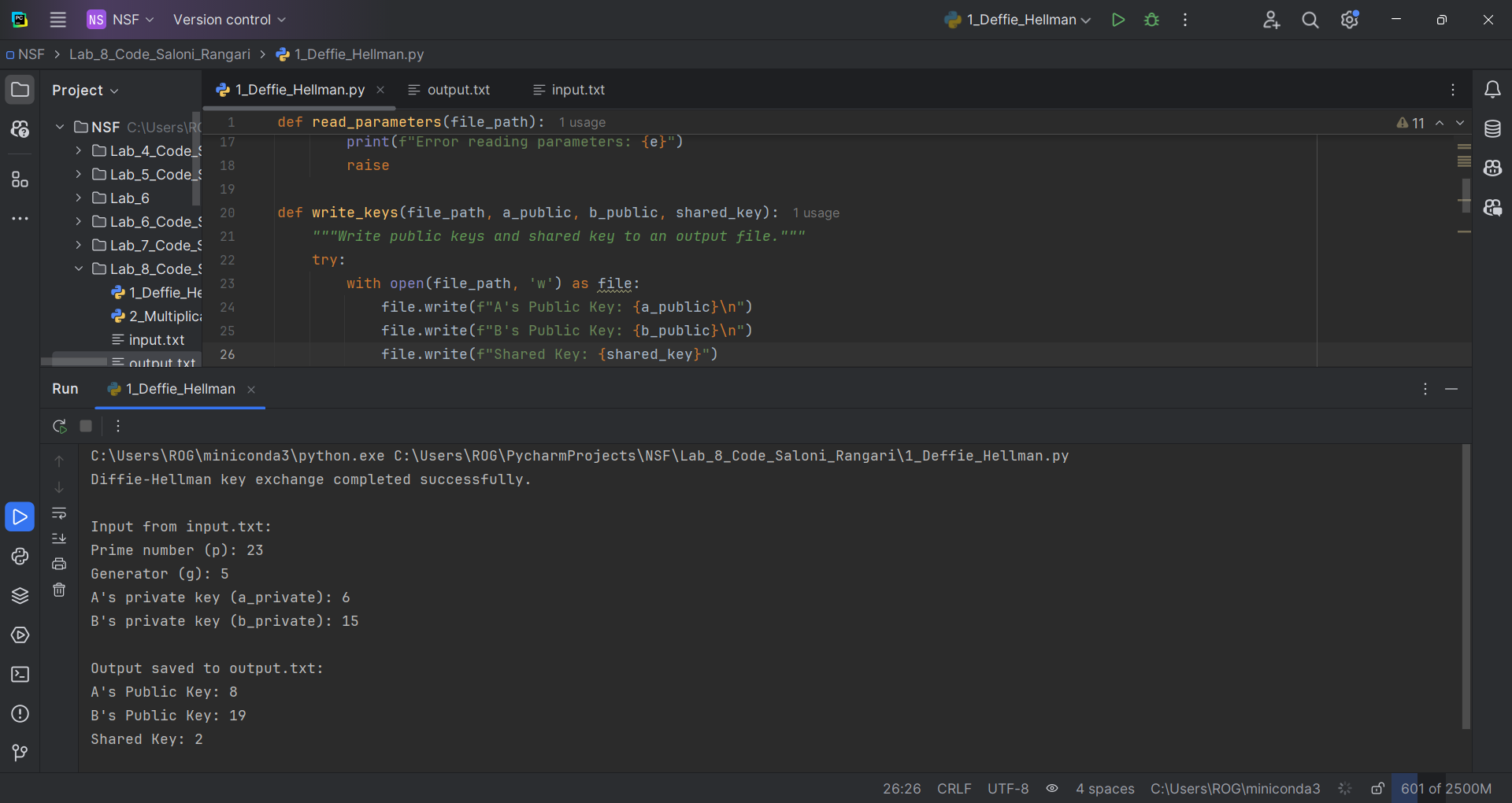
def read\_parameters(file\_path):try:  
 with open(file\_path, 'r') as file:  
 lines = file.readlines()  
 if len(lines) < 4:  
 raise ValueError("Input file must contain at least 4 lines.")  
 p = int(lines[0].strip()) # Prime number  
 g = int(lines[1].strip()) # Generator  
 a\_private = int(lines[2].strip()) # Private key for A  
 b\_private = int(lines[3].strip()) # Private key for B  
 return p, g, a\_private, b\_private  
 except ValueError as ve:  
 print(f"Value error reading parameters: {ve}")  
 raise  
 except Exception as e:  
 print(f"Error reading parameters: {e}")  
 raise  
  
def write\_keys(file\_path, a\_public, b\_public, shared\_key):try:  
 with open(file\_path, 'w') as file:  
 file.write(f"A's Public Key: {a\_public}\n")  
 file.write(f"B's Public Key: {b\_public}\n")  
 file.write(f"Shared Key: {shared\_key}")  
 except Exception as e:  
 print(f"Error writing keys: {e}")  
 raise  
  
def diffie\_hellman(file\_input, file\_output):try:  
 p, g, a\_private, b\_private = read\_parameters(file\_input)  
  
 # Calculate public keys  
 a\_public = pow(g, a\_private, p) # A's public key  
 b\_public = pow(g, b\_private, p) # B's public key  
  
 # Calculate shared secret  
 shared\_key\_a = pow(b\_public, a\_private, p) # Shared key calculated by A  
 shared\_key\_b = pow(a\_public, b\_private, p) # Shared key calculated by B  
  
 assert shared\_key\_a == shared\_key\_b # Both should be the same  
  
 write\_keys(file\_output, a\_public, b\_public, shared\_key\_a)  
 except Exception as e:  
 print(f"Error in Diffie-Hellman key exchange: {e}")  
 raise

if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file = 'input.txt'  
 output\_file = 'output.txt'  
  
 diffie\_hellman(input\_file, output\_file)  
 print("Diffie-Hellman key exchange completed successfully.\n")

# Show the input with labels  
 print("Input from input.txt:")  
 with open(input\_file, 'r') as file:  
 lines = file.readlines()  
 p = int(lines[0].strip()) # Prime number  
 g = int(lines[1].strip()) # Generator  
 a\_private = int(lines[2].strip()) # Private key for A  
 b\_private = int(lines[3].strip()) # Private key for B  
  
 print(f"Prime number (p): {p}")  
 print(f"Generator (g): {g}")  
 print(f"A's private key (a\_private): {a\_private}")  
 print(f"B's private key (b\_private): {b\_private}")

# Show the output  
 print("\nOutput saved to output.txt:")  
 with open(output\_file, 'r') as file:  
 print(file.read())

**Output:**



**Assignment 2: Write a program to calculate the multiplicative inverse of any number under mod operation.**

def extended\_gcd(a,b):  
 if a == 0:  
 return b,0,1  
 gcd,x1,y1 = extended\_gcd(b % a,a)  
 return gcd,y1 - (b // a) \* x1,x1  
  
  
def multiplicative\_inverse(a,m):  
 gcd,x,\_ = extended\_gcd(a % m,m)  
 if gcd != 1:  
 raise ValueError("Inverse doesn't exist")  
 return x % m  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Format: {number} \* {multiplicative\_inverse} mod {modulus} = 1\n")  
 number = int(input("Enter the number: "))  
 mod = int(input("Enter the modulus: "))  
  
 try:  
 inv = multiplicative\_inverse(number,mod)  
 print(f"The multiplicative inverse of {number} modulo {mod} is {inv}.")  
 print(f"{number} \* {inv} mod {mod} = 1")  
 except ValueError as e:  
 print(e)

**Output:**

**A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated**