Network Security and Forensics

Lab Session 8

Submitted To:-

Dr. Lokesh Chauhan Sir

Submitted By:-

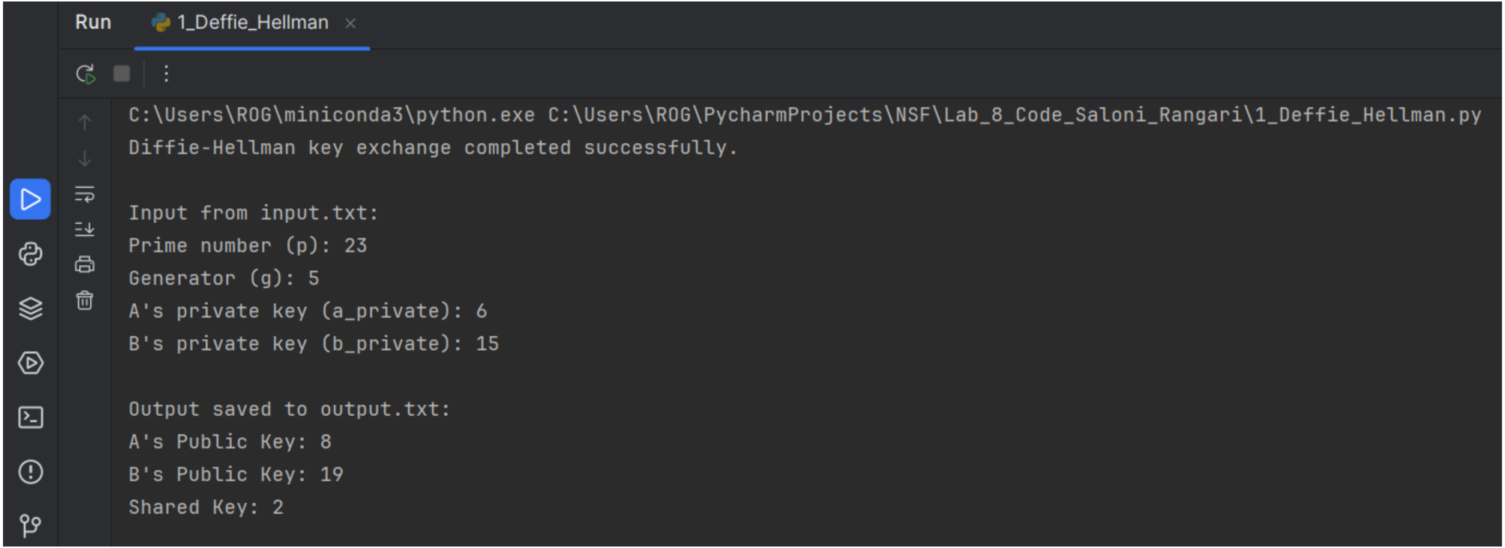
Saloni Rangari

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:

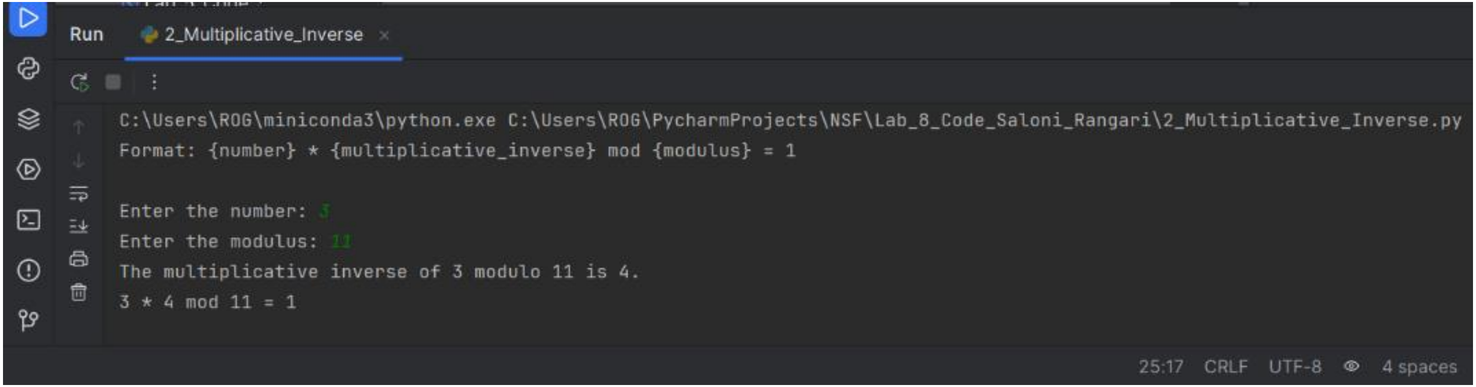


```
C:\Users\ROG\miniconda3\python.exe C:\Users\ROG\PycharmProjects\NSF\Lab_8_Code_Saloni_Rangari\1_Deffie_Hellman.py
Diffie-Hellman key exchange completed successfully.

Input from input.txt:
Prime number (p): 23
Generator (g): 5
A's private key (a_private): 6
B's private key (b_private): 15

Output saved to output.txt:
A's Public Key: 8
B's Public Key: 19
Shared Key: 2
```

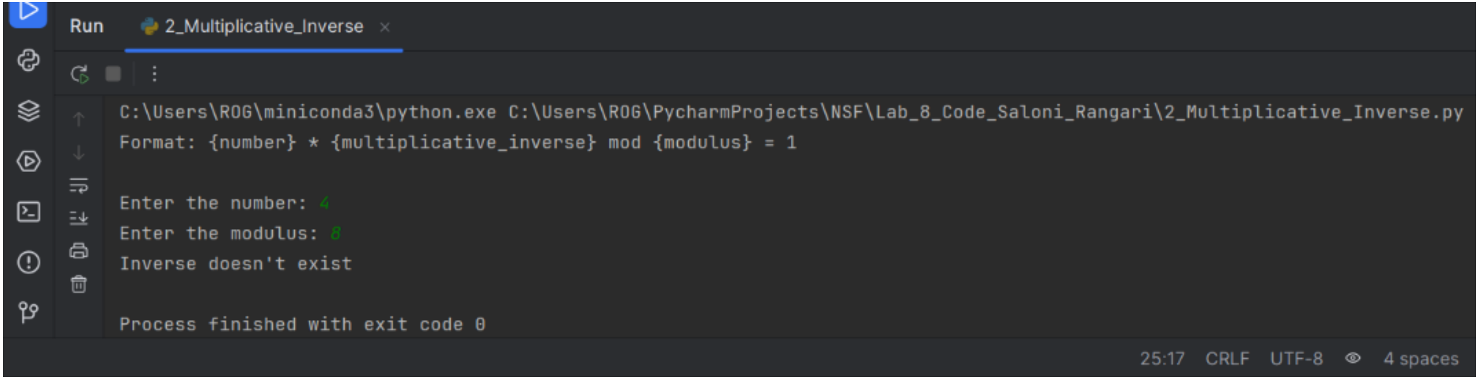
Assignment 2: Write a program to calculate the multiplicative inverse of any number under mod operation.

```
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:



```
C:\Users\ROG\miniconda3\python.exe C:\Users\ROG\PycharmProjects\NSF\Lab_8_Code_Saloni_Rangari\2_Multiplicative_Inverse.py
Format: {number} * {multiplicative_inverse} mod {modulus} = 1

Enter the number: 3
Enter the modulus: 11
The multiplicative inverse of 3 modulo 11 is 4.
3 * 4 mod 11 = 1
```



```
C:\Users\ROG\miniconda3\python.exe C:\Users\ROG\PycharmProjects\NSF\Lab_8_Code_Saloni_Rangari\2_Multiplicative_Inverse.py
Format: {number} * {multiplicative_inverse} mod {modulus} = 1

Enter the number: 4
Enter the modulus: 8
Inverse doesn't exist

Process finished with exit code 0
```