Lab-8

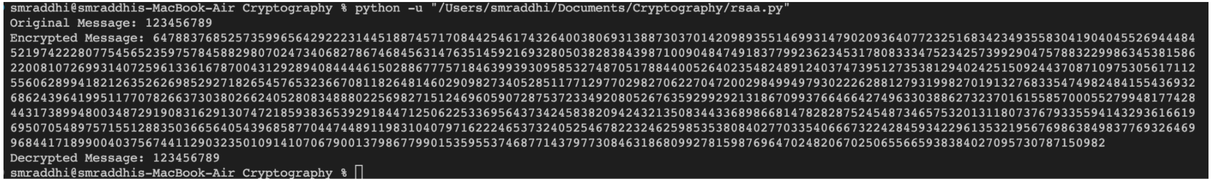
Cryptography

Name: Smraddhi Rathore

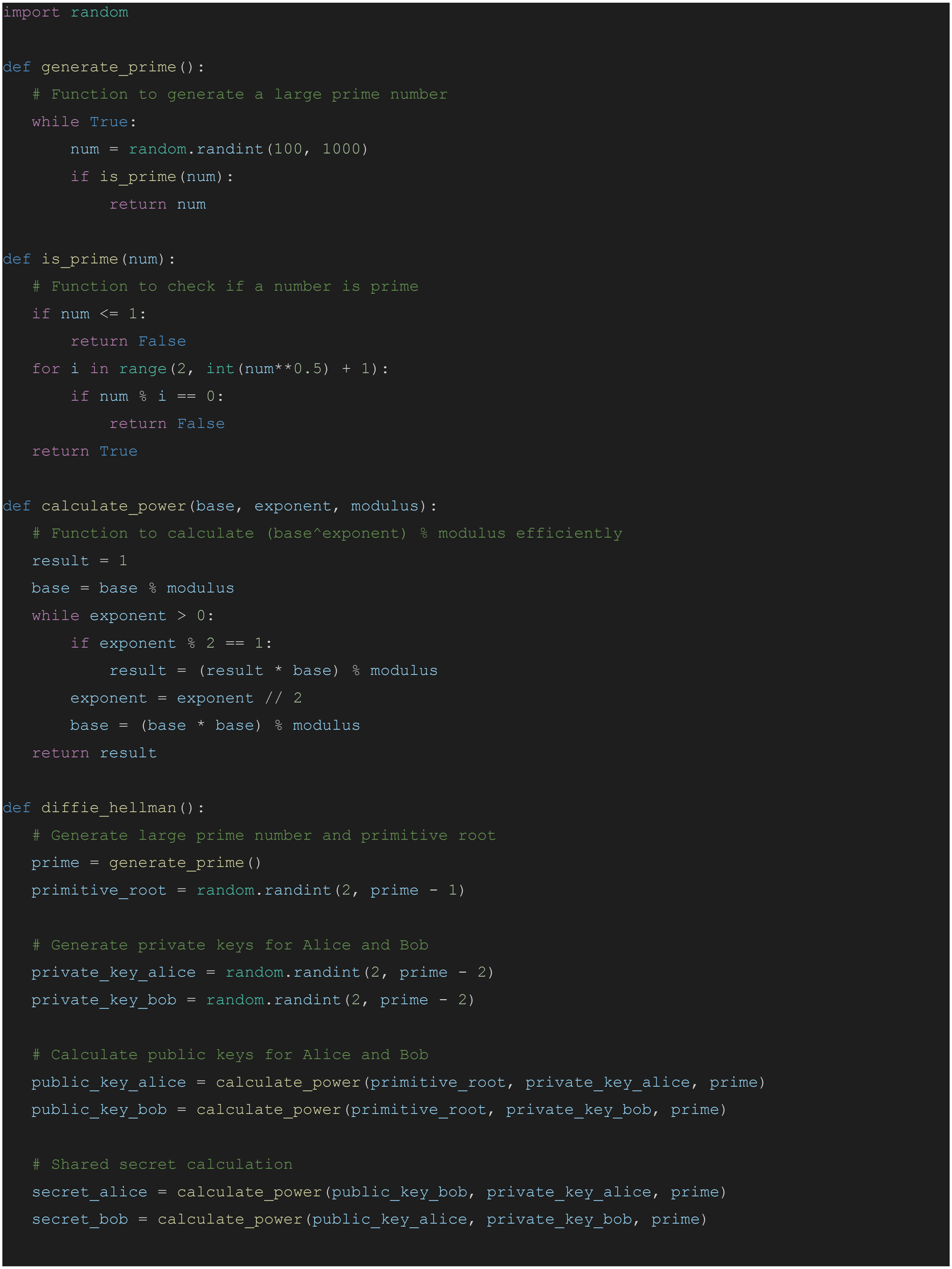
Roll No.: 2021bcy0025

Python

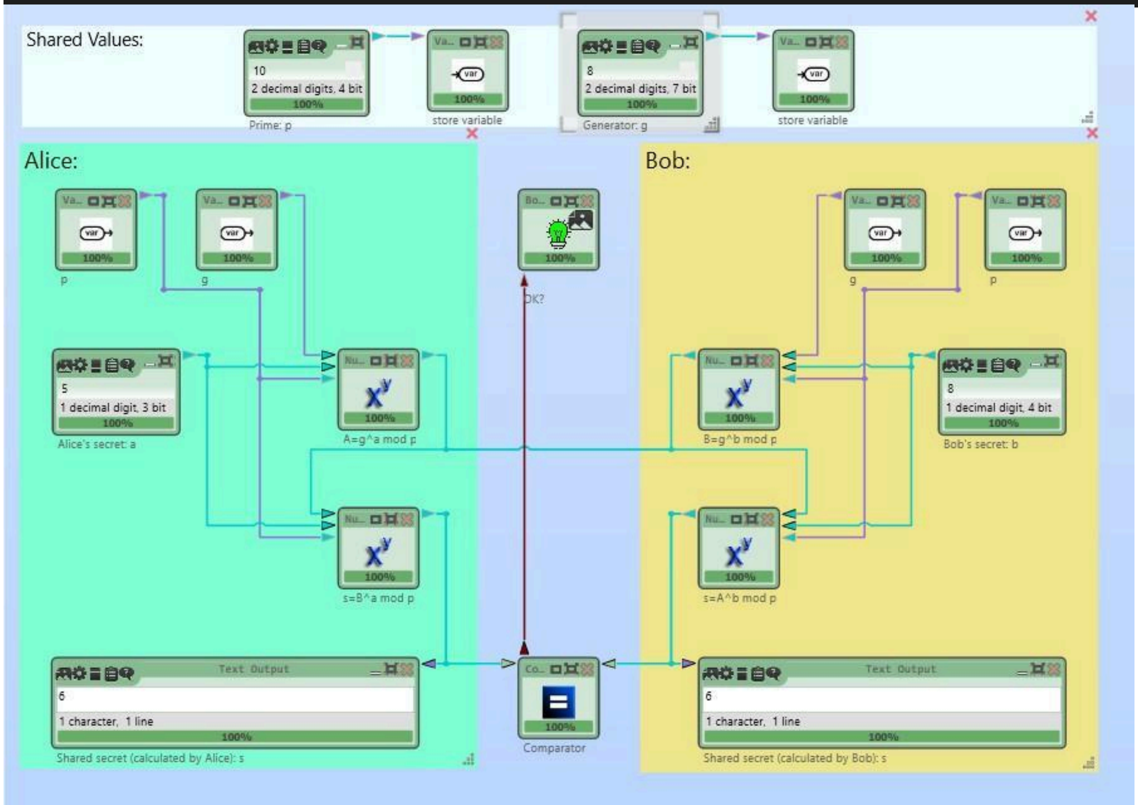
```python
from Crypto.Util import number

def generate_rsa_keypair(key_size):
   # Generate RSA keypair
   p = number.getPrime(key_size)
   q = number.getPrime(key_size)
   n = p * q
   phi = (p - 1) * (q - 1)
   e = 65537  # Common choice for e
   d = number.inverse(e, phi)
   return ((e, n), (d, n))

def rsa_encrypt(plaintext, public_key):
   # RSA encryption
   e, n = public_key
   ciphertext = pow(plaintext, e, n)
   return ciphertext

def rsa_decrypt(ciphertext, private_key):
   # RSA decryption
   d, n = private_key
   plaintext = pow(ciphertext, d, n)
   return plaintext

# Example usage:
public_key, private_key = generate_rsa_keypair(2048)
message = 123456789
encrypted_message = rsa_encrypt(message, public_key)
decrypted_message = rsa_decrypt(encrypted_message, private_key)

print("Original Message:", message)
print("Encrypted Message:", encrypted_message)
print("Decrypted Message:", decrypted_message)
```



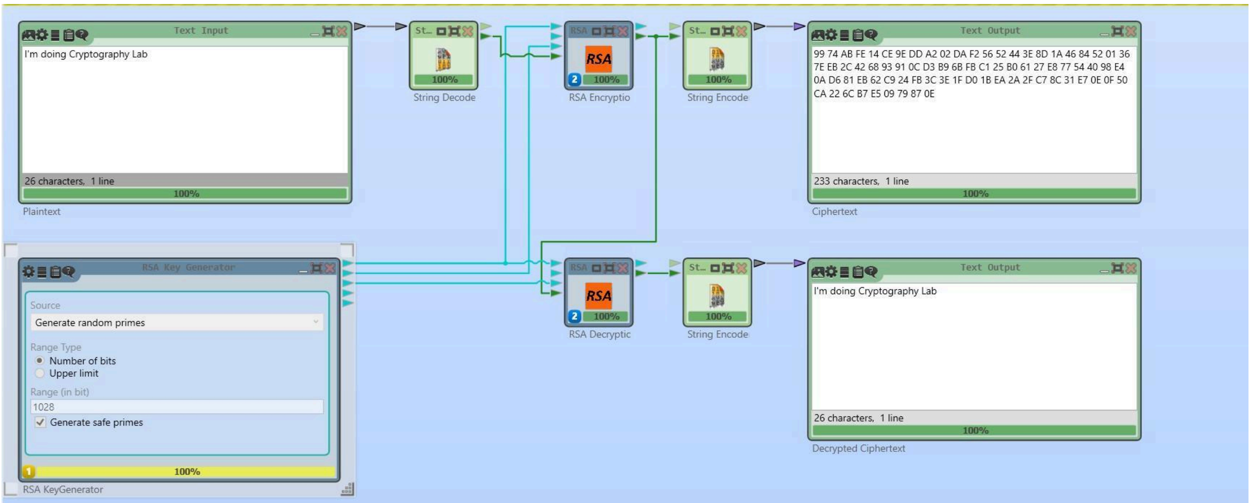
```
smraddhi@smraddhis-MacBook-Air Cryptography % python -u "/Users/smraddhi/Documents/Cryptography/rsaa.py"
Original Message: 123456789
Encrypted Message: 6478837685257359965642922231445188745717084425461743264003806931388730370142098935514699314790209364077232516834234935583041904045526944484
5219742228077545652359757845882980702473406827867468456314763514592169328050382838439871009048474918377992362345317808333475234257399290475788322998634538158 6
2200810726993140725961336167870043129289408444461502886777571846399393095853274870517884400526402354824891240374739512735381294024251509244370871097530561 7112
5560628994182126352626985292718265457653236670811826481460290982734052851177129770298270622704720029849949793022262881279319982701913276833547498248415543 6932
6862439641995117707826637303802662405280834888022569827151246960590728753723349208052676359299292131867099376646642749633038862732370161558570005527994817742 8
4431738994800348729190831629130747218593836539291844712506225336956437342458382094243213508344336898668147828287524548734657532013118073767933559414329361 6619
6950705489757155128835036656405439685877044744891198310407971622246537324052546782232462598535380840277033540666732242845934229613532195676986384983776932 6469
968441718990040375674411290323501091410706790013798677990153595537468771437977308463186809927815987696470248206702506556659383840270957307 87150982
Decrypted Message: 123456789
smraddhi@smraddhis-MacBook-Air Cryptography % ▯
```



```python
import random

def generate_prime():
    # Function to generate a large prime number
    while True:
        num = random.randint(100, 1000)
        if is_prime(num):
            return num

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

def calculate_power(base, exponent, modulus):
    # Function to calculate (base^exponent) % modulus efficiently
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent // 2
        base = (base * base) % modulus
    return result

def diffie_hellman():
    # Generate large prime number and primitive root
    prime = generate_prime()
    primitive_root = random.randint(2, prime - 1)

    # Generate private keys for Alice and Bob
    private_key_alice = random.randint(2, prime - 2)
    private_key_bob = random.randint(2, prime - 2)

    # Calculate public keys for Alice and Bob
    public_key_alice = calculate_power(primitive_root, private_key_alice, prime)
    public_key_bob = calculate_power(primitive_root, private_key_bob, prime)

    # Shared secret calculation
    secret_alice = calculate_power(public_key_bob, private_key_alice, prime)
    secret_bob = calculate_power(public_key_alice, private_key_bob, prime)
    return prime, primitive_root, public_key_alice, public_key_bob, secret_alice, secret_bob

 # Example usage:
 prime, primitive_root, public_key_alice, public_key_bob, secret_alice, secret_bob =
 diffie_hellman()
 print("Prime:", prime)
 print("Primitive Root:", primitive_root)
 print("Public Key Alice:", public_key_alice)
 print("Public Key Bob:", public_key_bob)
 print("Shared Secret Alice:", secret_alice)
 print("Shared Secret Bob:", secret_b
```



Range Type : Number of Bits



Plaintext: I'm doing Cryptography Lab

26 characters, 1 line

Ciphertext:

```
99 74 AB FE 14 CE 9E DD A2 02 DA F2 56 52 44 3E 8D 1A 46 84 52 01 36
7E EB 2C 42 68 93 91 0C D3 B9 6B FB C1 25 B0 61 27 E8 77 54 40 98 E4
0A D6 81 EB 62 C9 24 FB 3C 3E 1F D0 1B EA 2A 2F C7 8C 31 E7 0E 0F 50
CA 22 6C B7 E5 09 79 87 0E
```

233 characters, 1 line

Decrypted Ciphertext: I'm doing Cryptography Lab

26 characters, 1 line

RSA KeyGenerator:

Source: Generate random primes

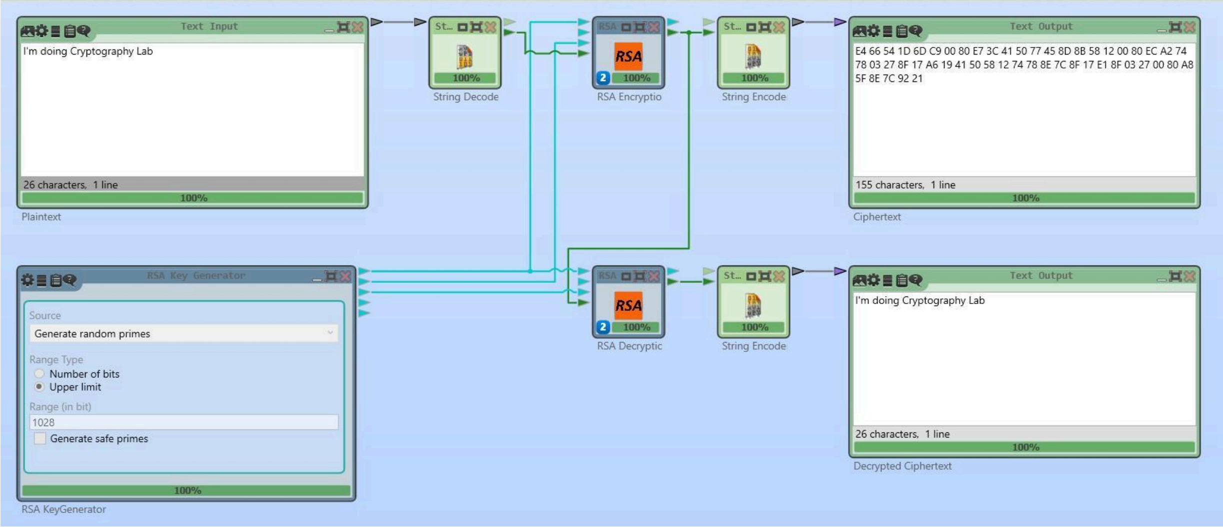
Range Type

● Number of bits
○ Upper limit

Range (in bit): 1028

☑ Generate safe primes

Range Type : Upper Limit



Plaintext: I'm doing Cryptography Lab

26 characters, 1 line

Ciphertext:

```
E4 66 54 1D 6D C9 00 80 E7 3C 41 50 77 45 8D 8B 58 12 00 80 EC A2 74
78 03 27 8F 17 A6 19 41 50 58 12 74 78 8E 7C 8F 17 E1 8F 03 27 00 80 A8
5F 8E 7C 92 21
```

155 characters, 1 line

Decrypted Ciphertext: I'm doing Cryptography Lab

26 characters, 1 line

RSA KeyGenerator:

Source: Generate random primes

Range Type

○ Number of bits
● Upper limit

Range (in bit): 1028

□ Generate safe primes