Experiment No. 8

<u>Aim</u>: To implement Diffie Hellman Algorithm in python and virtual lab.

<u>Theory</u>:

Whitefield Diffie and Martin Hellman develop Diffie Hellman key exchange Algorithms in 1976 to overcome the problem of key agreement and exchange. It enables the two parties who want to communicate with each other to agree on a symmetric key, a key that can be used for encrypting and decryption; Diffie Hellman key exchange algorithm can be used for only key exchange, not for encryption and decryption process. The algorithm is based on mathematical principles.

Diffie Hellman Algorithm

1. Shared values

- p: p is a prime number
- g: $g <$ p and $g \in < Z_p^*, * >$ is a generator.

2. Key generation for user A

- Select a Private key X_A Here, $X_A < g$

Now, Calculation of Public key $R_A = g^{X_A} mod\, p$

3. Key generation for user B

- Select a Private key X_B Here, $X_B < g$
- Now, Calculation of Public key $R_B = g^{X_B} mod\, p$

4. Calculation of Secret Key by A

- $key_A = R_B^{X_A} \, mod \, p$

5. Calculation of Secret Key by B

- $key_B = R_A^{X_B} \, mod \, p$

$$key = key_A = key_B = g^{X_A X_B} mod \, p$$

Example

$p = 61, g = 45, X_A = 36, X_B = 29$

$$R_A = 45^{36} mod 61 = 20$$

$$R_B = 45^{29} mod 61 = 19$$

$$key = 45^{29*36} \, mod 61 = 58$$

<u>Implementation</u>:

```
import random
import math

def check_multiplicative_inverse(x,y):
    if not y:
        return x

    return check_multiplicative_inverse(y,x%y)

class DH:

    def __init__(self,prime):
        assert self.check_prime(prime), "Number is not prime"
        self.prime = prime

        self.relative_primes = []
        for x in range(2,self.prime):
```

```
            if check_multiplicative_inverse(self.prime,x) == 1:
                self.relative_primes.append(x)
        self.generator = random.choice(self.relative_primes)

        self.alice_R_one()
        self.bob_R_two()
        self.alice_secret_key()
        self.bob_secret_key()

    def check_prime(self,prime):

        if prime == 1:
            return False
        if prime == 2:
            return True
        if not prime%2:
            return False

        for x in range(3,math.ceil(math.sqrt(prime))):
            if not prime%x:
                return False
        return True

    def alice_R_one(self):
        self.x = random.randint(0,self.prime)
        self.R_one = pow(self.generator,self.x,self.prime)

    def bob_R_two(self):
        self.y = random.randint(0,self.prime)
        self.R_two = pow(self.generator,self.y,self.prime)

    def alice_secret_key(self):
        self.key_A = pow(self.R_two,self.x,self.prime)


    def bob_secret_key(self):
        self.key_B = pow(self.R_one,self.y,self.prime)

obj = DH(int(input("Please Enter a prime Number: ")))
print(f"prime number: {obj.prime},  generator: {obj.generator}\n\
A secret key: {obj.x},  B secret key: {obj.y}\n\
A public key: {obj.R_one},  B public key: {obj.R_two}\n\
Secret key calculated by A: {obj.key_A},  Secret key calculated by B: {obj.key_B}")
```

Output:

```
Please Enter a prime Number: 67
prime number: 67,  generator: 19
A secret key: 2,  B secret key: 36
A public key: 26,  B public key: 25
Secret key calculated by A: 22,  Secret key calculated by B: 22
```

```
Please Enter a prime Number: 15

AssertionError: Number is not prime
```

```
Please Enter a prime Number: 83
prime number: 83,  generator: 8
A secret key: 9,  B secret key: 32
A public key: 5,  B public key: 33
Secret key calculated by A: 75,  Secret key calculated by B: 75
```

Vlab Output: