PRN No: 2020BTECS00006

Name: Samrat Vishwas Jadhav

Batch: B1

Assignment: 9

Title of assignment: Implementation of Diffie Hellman Key Exchange Algorithm

1. **Aim:**
   Implementation of Diffie Hellman Key Exchange Algorithm

2. **Theory:**

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Example:
Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x = (9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

Code:

Client:

```
import socket

# Function to find mod: a^m mod n
def findExpoMod(a, m, n):
    # Decimal to binary conversion
    m_bin = bin(m).replace("0b", "")

    # Convert it into list (individual characters)
    m_bin_lst = [int(i) for i in m_bin]

    # Initialize the list
    a_lst = [a]

    # Functions to perform operations
    # If next value = 0
    def oneOperation(num):
        return (num*num) % n

    # If next value = 1
    def twoOperation(num):
        return (a * oneOperation(num)) % n

    for j in range(len(m_bin_lst)):
        if j+1 == len(m_bin_lst):
            break

        if(m_bin_lst[j+1] == 0):
            a_lst.append(oneOperation(a_lst[j]))
        else:
            a_lst.append(twoOperation(a_lst[j]))

    return a_lst[-1]
```

```
HOST = 'localhost'
PORT = 12345
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (HOST, PORT)  # Server address and port
client_socket.connect(server_address)
print(f"Connected to server at: {HOST}:{PORT}")

# Receive the server's public key, q, alpha
received_Ya = client_socket.recv(1024)
Ya = int(received_Ya.decode())
print(f"Received server's public key as: {Ya}")

received_q = client_socket.recv(1024)
q = int(received_q.decode())
print(f"Received large prime as: {q}")

received_alpha = client_socket.recv(1024)
alpha = int(received_alpha.decode())
print(f"Received alpha as: {alpha}")

# Client's Private Key
Xb = int(input(f"Enter the private key for B (Xb) [less than
{q}]:\n"))
if Xb >= q:
    print("Private key must be less than choosen prime!")
    exit()

# Client's Public Key
Yb = findExpoMod(alpha, Xb, q)
print(f"Client's Public key is: {Yb}")

# Send this to Server
print("Sending Client's Public Key to Server...")
send_Yb = str(Yb).encode()
client_socket.sendall(send_Yb)

# Receive Server's Shared key
```

```
received_Ks = client_socket.recv(1024)
Ks = int(received_Ks.decode())
print(f"Received Server's Shared key as: {Ks}")

# Compute shared key and send to server
Kc = findExpoMod(Ya, Xb, q)
print(f"Client's Shared key is: {Kc}")

print("Sending it to server...")
send_Kc = str(Kc).encode()
client_socket.sendall(send_Kc)

if Kc == Ks:
    print("Both shared keys are equal\nKeys exchanged
successfully!")
else:
    print("Both shared keys aren't equal.\nKey exchange failed!")

client_socket.close()
```

Server:

```
from generate_prime import is_prime, generate_prime_no
import socket

# Function to find mod: a^m mod n
def findExpoMod(a, m, n):
    # Decimal to binary conversion
    m_bin = bin(m).replace("0b", "")

    # Convert it into list (individual characters)
    m_bin_lst = [int(i) for i in m_bin]

    # Initialize the list
    a_lst = [a]

    # Functions to perform operations
```

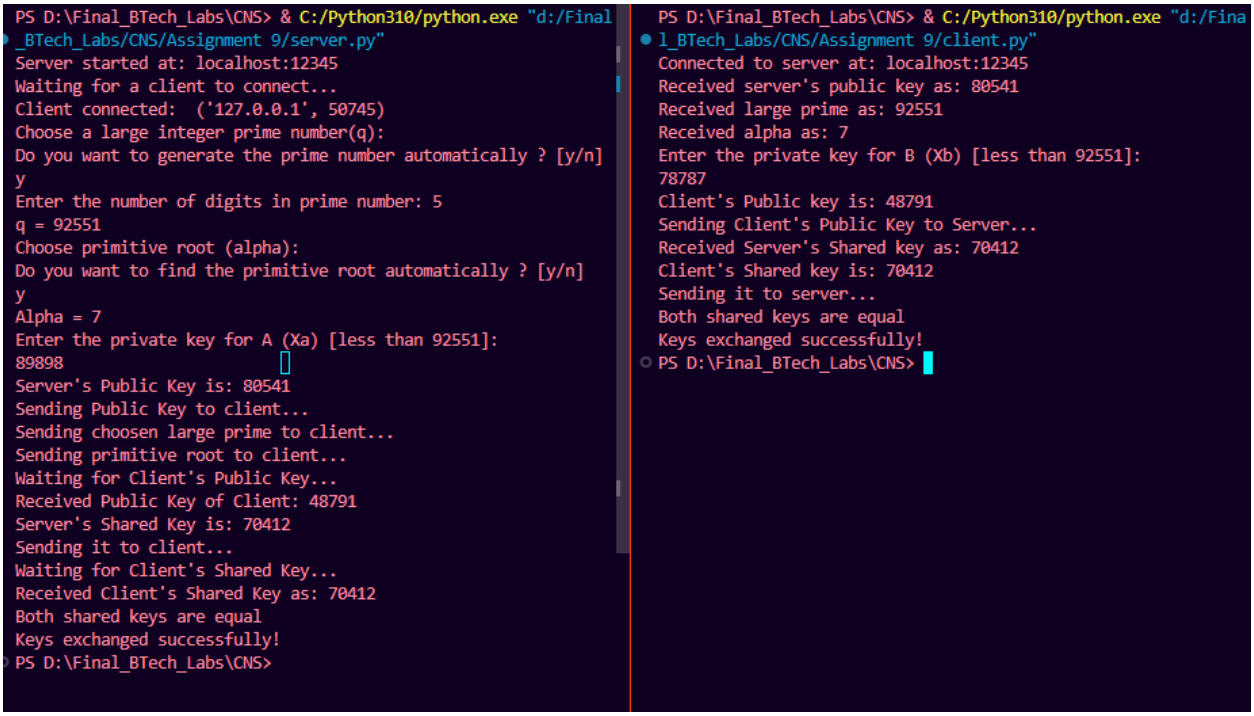
```
    # If next value = 0
    def oneOperation(num):
        return (num*num) % n

    # If next value = 1
    def twoOperation(num):
        return (a * oneOperation(num)) % n

    for j in range(len(m_bin_lst)):
        if j+1 == len(m_bin_lst):
            break

        if(m_bin_lst[j+1] == 0):
            a_lst.append(oneOperation(a_lst[j]))
        else:
            a_lst.append(twoOperation(a_lst[j]))

    return a_lst[-1]

def is_primitive_root(alpha, q):
    L = []

    for i in range(1, q):
        L.append(findExpoMod(alpha, i, q))

    for i in range(1, q):
        if L.count(i) > 1:
            L.clear()
            return False

        return True

# Initialize Socket
HOST = 'localhost'
PORT = 12345
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (HOST, PORT)  # Server address and port
```

```
server_socket.bind(server_address)
server_socket.listen(1)
print(f"Server started at: {HOST}:{PORT}")

print("Waiting for a client to connect...")
client_socket, client_address = server_socket.accept()
print("Client connected: ", client_address)

# DH Key-exchange
# Choose prime no. 'q'
print("Choose a large integer prime number(q):")
gen_r = input("Do you want to generate the prime number
automatically ? [y/n]\n")
if gen_r == 'y':
    dig_p = int(input("Enter the number of digits in prime
number: "))
    q = generate_prime_no(dig_p)
    print(f"q = {q}")
elif gen_r == 'n':
    q = int(input("Enter a large prime number:\n"))
    if not is_prime(q):
        print(f"Entered number is not prime!")
        exit()
else:
    print("Invaild choice!")
    exit()

# Choose primitive root 'alpha'
print("Choose primitive root (alpha):")
gen_pr = input("Do you want to find the primitive root
automatically ? [y/n]\n")
if gen_pr == 'y':
    for a in range(2, q):
        if is_primitive_root(a, q):
            alpha = a
            break
    print(f"Alpha = {alpha}")
```

```
elif gen_pr == 'n':
    alpha = int(input(f"Enter the primiitive root of {q}:\n"))
    if not is_primitive_root(alpha, q):
        print(f"This is not the primitive root!")
        exit()
else:
    print("Invaild choice!")
    exit()

# Server's Private key
Xa = int(input(f"Enter the private key for A (Xa) [less than
{q}]:\n"))
if Xa >= q:
    print("Private key must be less than choosen prime!")
    exit()

# Server's Public Key
Ya = findExpoMod(alpha, Xa, q)

# Send this data to client
print(f"Server's Public Key is: {Ya}")
print("Sending Public Key to client...")
send_Ya = str(Ya).encode()
client_socket.sendall(send_Ya)

print("Sending choosen large prime to client...")
send_q = str(q).encode()
client_socket.sendall(send_q)

print("Sending primitive root to client...")
send_alpha = str(alpha).encode()
client_socket.sendall(send_alpha)

print("Waiting for Client's Public Key...")

# Receive Client's Public Key
received_Yb = client_socket.recv(1024)
```

```
Yb = int(received_Yb.decode())
print(f"Received Public Key of Client: {Yb}")

# Compute shared key and send to client
Ks = findExpoMod(Yb, Xa, q)
print(f"Server's Shared Key is: {Ks}")

print("Sending it to client...")
send_Ks = str(Ks).encode()
client_socket.sendall(send_Ks)

# Receive Client's Shared Key
print("Waiting for Client's Shared Key...")
received_Kc = client_socket.recv(1024)
Kc = int(received_Kc.decode())
print(f"Received Client's Shared Key as: {Kc}")

if Ks == Kc:
    print("Both shared keys are equal\nKeys exchanged
successfully!")
else:
    print("Both shared keys aren't equal.\nKey exchange failed!")

client_socket.close()
server_socket.close()
```

Ouput:


PS D:\Final_BTech_Labs\CNS> & C:/Python310/python.exe "d:/Final_BTech_Labs/CNS/Assignment 9/server.py"
Server started at: localhost:12345
Waiting for a client to connect...
Client connected: ('127.0.0.1', 50745)
Choose a large integer prime number(q):
Do you want to generate the prime number automatically ? [y/n]
y
Enter the number of digits in prime number: 5
q = 92551
Choose primitive root (alpha):
Do you want to find the primitive root automatically ? [y/n]
y
Alpha = 7
Enter the private key for A (Xa) [less than 92551]:
89898
Server's Public Key is: 80541
Sending Public Key to client...
Sending choosen large prime to client...
Sending primitive root to client...
Waiting for Client's Public Key...
Received Public Key of Client: 48791
Server's Shared Key is: 70412
Sending it to client...
Waiting for Client's Shared Key...
Received Client's Shared Key as: 70412
Both shared keys are equal
Keys exchanged successfully!
PS D:\Final_BTech_Labs\CNS>
PS D:\Final_BTech_Labs\CNS> & C:/Python310/python.exe "d:/Final_BTech_Labs/CNS/Assignment 9/client.py"
Connected to server at: localhost:12345
Received server's public key as: 80541
Received large prime as: 92551
Received alpha as: 7
Enter the private key for B (Xb) [less than 92551]:
78787
Client's Public key is: 48791
Sending Client's Public Key to Server...
Received Server's Shared key as: 70412
Client's Shared key is: 70412
Sending it to server...
Both shared keys are equal
Keys exchanged successfully!
PS D:\Final_BTech_Labs\CNS>