Experiment 03

Aim: Analyze and implement Diffie-Hellman Key Exchange Algorithm

Tools: Python

Theory: DIFFIE–HELLMAN KEY EXCHANGE:

Diffie–Hellman key exchange (D–H) is a specific method of exchanging keys. It is one of the earliest practical examples of key exchange implemented within the field of cryptography. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

The Diffie–Hellman key agreement was invented in 1976 during a collaboration between Whitfield Diffie and Martin Hellman and was the first practical method for establishing a shared secret over an unprotected communication channel.

Diffie–Hellman establishes a shared secret that can be used for secret communications by exchanging data over a public network.

STEP 1: GLOBAL PUBLIC ELEMENTS:

Firstly, Alice and Bob agree on two large prime numbers, n and g. These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.

STEP 2: ASYMMETRIC KEY GENERATION BY USER 'A':
Alice chooses another large random number X, and calculates, the public key, A, such that:
$A = g^X \bmod n$

STEP 3: Alice sends the number A to Bob.

STEP 4: KEY GENERATION BY USER 'B':
Bob independently chooses another large random number Y, and calculates, the public key, B, such that:
$B = g^Y \bmod n$

STEP 5: Bob sends the number B to Alice.

STEP 6: SYMMETRIC KEY (K) GENERATION BY USER 'A':
A now computes the secret key, K1 as follows:
$K1 = B^X \bmod n$

STEP 7: SYMMETRIC KEY (K) GENERATION BY USER 'B':
B now computes the secret key, K2 as follows:
$K2 = A^Y \bmod n$

Implementation :
Code–
Sender–

```
import socket
import random
```

```
def generate_key(p, g, private_key):
    return (g ** private_key) % p

def send_public_key(connection, public_key):
    connection.send(str(public_key).encode())

def receive_public_key(connection):
    return int(connection.recv(1024).decode())

def calculate_shared_secret(public_key, private_key, p):
    return (public_key ** private_key) % p

def main():
    # Commonly agreed prime modulus and generator
    p = 23
    g = 5

    # Sender's private key
    sender_private_key = random.randint(1, 10)

    # Establish connection
    sender_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sender_socket.bind(('localhost', 12345))
    sender_socket.listen(1)
    print("Waiting for connection...")
    connection, address = sender_socket.accept()
    print("Connected to:", address)

    # Generate and send public key
    sender_public_key = generate_key(p, g, sender_private_key)
    send_public_key(connection, sender_public_key)

    # Receive receiver's public key
    receiver_public_key = receive_public_key(connection)

    # Calculate shared secret
    shared_secret = calculate_shared_secret(receiver_public_key, sender_private_key, p)
    print("Shared secret:", shared_secret)

    # Close connection
    connection.close()
    sender_socket.close()

if __name__ == "__main__":
    main()
```

Receiver–

```
import socket
import random

def generate_key(p, g, private_key):
    return (g ** private_key) % p

def send_public_key(connection, public_key):
    connection.send(str(public_key).encode())

def receive_public_key(connection):
    return int(connection.recv(1024).decode())

def calculate_shared_secret(public_key, private_key, p):
    return (public_key ** private_key) % p

def main():
    # Commonly agreed prime modulus and generator
    p = 23
    g = 5

    # Receiver's private key
    receiver_private_key = random.randint(1, 10)

    # Establish connection
    receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    receiver_socket.connect(('localhost', 12345))
    print("Connected to sender...")

    # Receive sender's public key
    sender_public_key = receive_public_key(receiver_socket)
```

```
    # Generate and send public key
    receiver_public_key = generate_key(p, g, receiver_private_key)
    send_public_key(receiver_socket, receiver_public_key)

    # Calculate shared secret
    shared_secret = calculate_shared_secret(sender_public_key, receiver_private_key, p)
    print("Shared secret:", shared_secret)

    # Close connection
    receiver_socket.close()

if __name__ == "__main__":
    main()
```

Output–

Sender–

```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lab324-7\Desktop\1533_TE COMP B_DH>python sender.py
Waiting for connection...
Connected to: ('127.0.0.1', 53910)
Shared secret: 19

C:\Users\lab324-7\Desktop\1533_TE COMP B_DH>
```

Receiver–

```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lab324-7\Desktop\1533_TE COMP B_DH>python receiver.py
Connected to sender...
Shared secret: 19

C:\Users\lab324-7\Desktop\1533_TE COMP B_DH>s
```

Result and Discussion:

In this Experiment, we implemented the Diffie-Hellman Algorithm for key Exchanging between two users. We got the desired output from the algorithm after exchanging keys i.e. K1==K2.

Learning Outcomes: The student will be able to

LO1: Understand the Diffie-Hellman Key Exchange Algorithm

LO2: Analyze and implement the Diffie-Hellman Key Exchange Algorithm

Course Outcomes: Upon completion of the course students will be able to analyze and implement Diffie-Hellman Key Exchange Algorithm for generation of shared symmetric key

Conclusion: After performing this Experiment, we understood about the Diffie-Hellman Key Exchange Algorithm in detail. We learned how an attacker can alter the messages and may alter the communication between two users if there private keys are easy to solve.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				