Final Year B.Tech. (CSE) – VII [2024-25]

6CS451: Cryptography and Network Security Lab (C&NS Lab)

Date: 26/08/2024

Assignment 8

PRN: 21510042 **Name:** Omkar Rajesh Auti

---

1. **Implement the Diffie-Hellman Key Exchange algorithm for a given problem**

Ans:

The Diffie-Hellman Key Exchange is a cryptographic algorithm that allows two parties to securely share a secret key over a public channel. This shared key can then be used for encrypted communication. The algorithm allows two parties to generate a shared secret key that can be used for subsequent encryption and decryption, even if the exchange itself is observed by an eavesdropper.

How Diffie-Hellman Works:

1. **Public Parameters:**
    - Both parties agree on a large prime number p and a base g (a primitive root modulo p).
2. **Key Exchange Process:**
    - **Party A** selects a private key 'a' and computes A = g^ a mod p, then sends A to Party B.
    - **Party B** selects a private key 'b' and computes B = g^ b mod p, then sends B to Party A.
3. **Shared Secret:**
    - **Party A** computes the shared secret as S = B^ a mod p.
    - **Party B** computes the shared secret as S = A^ b mod p.

Since both calculations result in the same value, S becomes the shared secret key, even though an eavesdropper only knows p, g, A, and B.

The Diffie-Hellman algorithm securely establishes a shared secret key without transmitting it directly, making it fundamental for secure communications in protocols like SSL/TLS.

To implement the Diffie-Hellman Key Exchange algorithm for client-server communication across two different machines, we will create two Python programs: one for the client and one for the server. The server will generate its public key and share it with the client, and vice versa. Both will then calculate the shared secret key independently.

Python Code:

Server-side program:

```
import socket
import random


def generate_private_key(p):
    """Generate a private key."""
    return random.randint(2, p-2)


def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    return pow(g, private_key, p)


def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    return pow(public_key, private_key, p)
```

```python
def start_server(host='localhost', port=5000):
    # p = 23
    # g = 5
    p = 104729  # Shared prime number --> 104729 (which is
the 10000th prime number)
    g = 2   # Shared base --> Primitive Root g: 2

    # Generate server's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create server socket
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server started. Listening on {host}:{port}")

    conn, addr = server_socket.accept()
    print(f"\nConnected by {addr}")

    # Send the server's public key to the client
    conn.sendall(str(public_key).encode())

    # Receive the client's public key
    client_public_key = int(conn.recv(1024).decode())
    print(f"\nReceived Client's Public Key:
{client_public_key}")

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(client_public_key, private_key, p)
    print(f"\nShared Secret (Server): {shared_secret}")

    conn.close()
    server_socket.close()


if __name__ == "__main__":
```

```
    start_server()
```

Client-side program:

```
import socket
import random


def generate_private_key(p):
    """Generate a private key."""
    return random.randint(2, p-2)


def calculate_public_key(g, private_key, p):
    """Calculate the public key."""
    return pow(g, private_key, p)


def calculate_shared_secret(public_key, private_key, p):
    """Calculate the shared secret."""
    return pow(public_key, private_key, p)


def start_client(server_host='localhost', server_port=5000):
    # p = 23
    # g = 5
    p = 104729  # Shared prime number --> 104729 (which is
the 10000th prime number)
    g = 2   # Shared base --> Primitive Root g: 2

    # Generate client's private and public keys
    private_key = generate_private_key(p)
    public_key = calculate_public_key(g, private_key, p)

    # Create client socket
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect((server_host, server_port))
```

```
    # Receive the server's public key
    server_public_key =
int(client_socket.recv(1024).decode())
    print(f"\nReceived Server's Public Key:
{server_public_key}")

    # Send the client's public key to the server
    client_socket.sendall(str(public_key).encode())

    # Calculate the shared secret
    shared_secret =
calculate_shared_secret(server_public_key, private_key, p)
    print(f"\nShared Secret (Client): {shared_secret}")

    client_socket.close()


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

Output:

Server output–

```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 8> python server_diffie_hellman.py
Server started. Listening on localhost:5000

Connected by ('127.0.0.1', 52927)

Received Client's Public Key: 45195

Shared Secret (Server): 91540
```

Client output–

```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 8> python client_diffie_hellman.py

Received Server's Public Key: 45505

Shared Secret (Client): 91540
```

Practical Applications of Diffie–Hellman:

- **Secure Communication:** Establishing a shared secret for symmetric encryption over an insecure channel.
- **VPNs:** Secure key exchange for Virtual Private Networks.
- **TLS/SSL:** Part of the key exchange process in securing internet communications.

The Diffie-Hellman algorithm forms the basis of many modern cryptographic protocols and is crucial for secure communication in distributed systems.