Experiment No. 6

Title: Diffie-Hellman Key Exchange Protocol

Batch: A3 **Roll No.: 16010421059** **Experiment No.:06**

Title: Perform VLab and implement Diffie-Hellman key exchange protocol.

Resources needed: Windows/Linux OS

Theory:

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that p is a prime number and q is a generator of p. The generator q is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers. The value of p may be large but the value of q is usually small. Once Alice and Bob have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys a* and b* based on their personal keys according to the formulas

$$a^* = q^a \bmod p$$

and

$$b^* = q^b \bmod p$$

The two users can share their public keys a* and b* over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number x can be generated by either user on the basis of their own personal keys. Alice computes x using the formula

$$x = (b^*)^a \bmod p$$

Bob computes x using the formula

$$x = (a^*)^b \bmod p$$

The value of x turns out to be the same according to either of the above two formulas. However, the personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.

Algorithm :

1) Perform Vlab - https://cse29-iiith.vlabs.ac.in/exp/diffie-hellman/index.html
2) Make use of client-server chatting application and implement following

A. Client/Sender

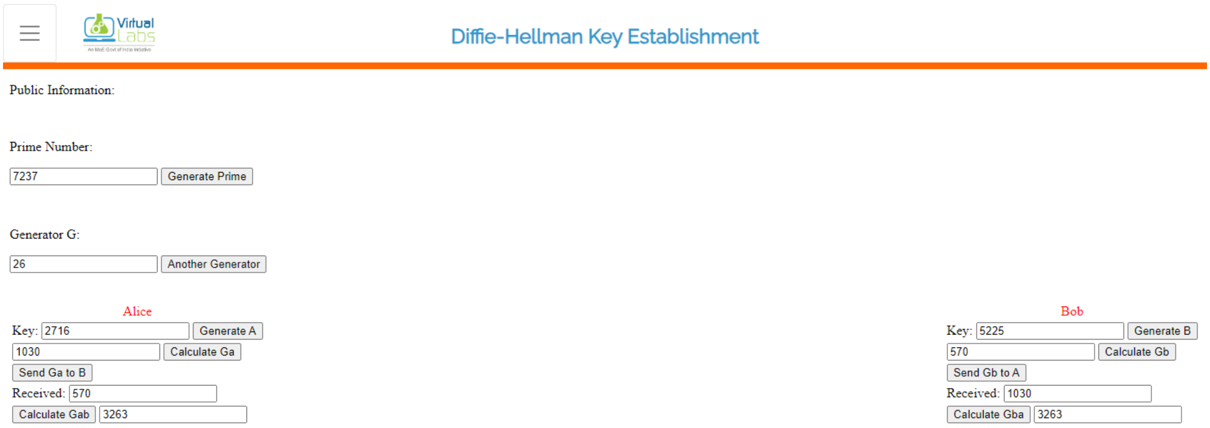
1. Choose a large prime number *p*
2. Calculate generator *g* of *p*
3. Share *p* and *g* with the Server/Receiver

4. Select any natural number (client secrete) a
5. *Calculate* $R_A = g^a \bmod p$ and send it to the Server/Receiver
6. Upon receiving R_B from the Server/Receiver, calculate shared key $K_{AB} = (R_B)^a \bmod p$

B. Server/Receiver:

1. Select any natural number (server secrete) b
2. Upon receiving p and g, calculate $R_B = b^a \bmod p$ and send it to the Client/Sender
3. Upon receiving R_A from the Client/Sender, calculate shared key $K_{AB} = (R_A)^b \bmod p$

C. NOTE/OBSERVE : Manually verify that the K_{AB} at both the ends is same.



INPUT:

```
import socket

# Create a socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_host = 'localhost'  # Server's IP address or hostname
server_port = 12345        # Port number

# Bind the socket to the server address and port
server_socket.bind((server_host, server_port))
```

```
# Listen for incoming connections
server_socket.listen(1)
print("Server is listening for incoming connections...")

# Accept a connection from a client
client_socket, client_address = server_socket.accept()
print(f"Accepted connection from {client_address}")

# Receive p and g from the client
p = int(input("Enter the prime number (p): "))
g = int(input("Enter the generator (g): "))

# Send p and g to the client
client_socket.send(f"{p} {g}".encode())

# Receive RA from the client
RA = int(client_socket.recv(1024).decode())

# Receive RB from the client
RB = int(client_socket.recv(1024).decode())

# Server's secret
b = int(input("Enter the server's secret (b): "))

# Calculate shared key KAB = (RA^b) mod p
KAB_server = pow(RA, b, p)

print("Shared Key (Server):", KAB_server)

# Send RB to the client
client_socket.send(str(RB).encode())

# Close the server and client sockets
server_socket.close()
client_socket.close()
```

Questions:

1. Man-in-the-Middle (MITM) Attack on Diffie-Hellman Key Exchange:

The Diffie-Hellman key exchange protocol is susceptible to a Man-in-the-Middle (MITM) attack, which can compromise the confidentiality of the shared secret key between two parties. Here's how the MITM attack on Diffie-Hellman works:

a. Alice and Bob both generate their public and private keys in the standard way in the presence of an attacker, Mallory.

b. When Alice sends her public key to Bob, Mallory intercepts it. Mallory then generates her own key pair and sends her public key to Bob, pretending to be Alice.

c. Bob receives Mallory's public key, thinking it's from Alice, and generates a shared secret with Mallory, believing it's with Alice.

d. Mallory, meanwhile, intercepts the public key that Bob sends back to Alice and substitutes it with her own public key.

e. Alice, unaware of the MITM attack, generates a shared secret with Mallory.

As a result, Alice and Bob both have shared secrets with Mallory, who can now decrypt and eavesdrop on their communications.

2. Mitigating the MITM Attack on Diffie-Hellman:

To mitigate the MITM attack on the Diffie-Hellman key exchange, you can employ various techniques and best practices:

a. Authentication: Implement a mechanism for authenticating the public keys exchanged between the parties. One common approach is to use digital signatures. Before accepting the public key, a recipient can verify the signature with the sender's known public key.

b. Use Certificate Authorities (CAs): Public key infrastructure (PKI) systems with trusted CAs can help verify the authenticity of public keys. Certificates issued by CAs can provide a level of trust in the public keys exchanged during the Diffie-Hellman key exchange.

c. Diffie-Hellman Key Exchange with Authentication: Use a variant of the Diffie-Hellman protocol that includes an authentication step. For example, the Station-to-Station (STS) protocol adds an authentication step to the basic Diffie-Hellman exchange to ensure that both parties authenticate each other.

d. Out-of-Band Verification: Communicate the public keys out of band, which means using a separate, trusted channel for sharing public keys. For example, exchange keys in person, over the phone, or through physical means to eliminate the risk of an eavesdropper.

e. Long-Term Public Keys: Use long-term, trusted public keys for Diffie-Hellman key exchange. These keys can be pre-shared and well-known between parties, making it more difficult for an attacker to replace them.

f. Perfect Forward Secrecy (PFS): Implement Perfect Forward Secrecy by generating temporary keys for each session, even if long-term keys are compromised. This ensures that past communications remain secure even if an attacker gains access to long-term keys.

Outcomes:

CO 2 : Illustrate different cryptographic algorithms for security

---

Conclusion:

Performed VLab and implemented Diffie-Hellman key exchange protocol successfully.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Mark Stamp, “Information security Principles and Practice” Wiley.