

**Experiment No. 6**

**Title:** Diffie-Hellman Key Exchange Protocol

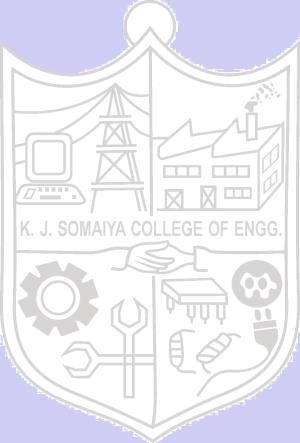
**Batch:A3 Roll No.:16010421073 Experiment No.: 6 Title:** To implement Diffie-Hellman key exchange protocol.

**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\* = qa mod p and

b\* = qb mod 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 mod p

Bob computes x using the formula

x = (a\*)b mod 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 :**Make use of client-server chatting application**.**

## 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 RA = ga mod p* and send it to the Server/Receiver
  6. Upon receiving RB from the Server/Receiver, calculate shared key KAB= (RB)a*modp*

## Server/Receiver:

* 1. Select any natural number (server secrete) *b*
  2. Upon receiving *p* and *g*, calculate *RB = ba mod p* and send it to the Client/Sender
  3. Upon receiving RA from the Client/Sender, calculate shared key KAB= (RA )b*modp*

**NOTE/OBSERVATION :**Manually verify that theKABat both the ends is same.

# Code:

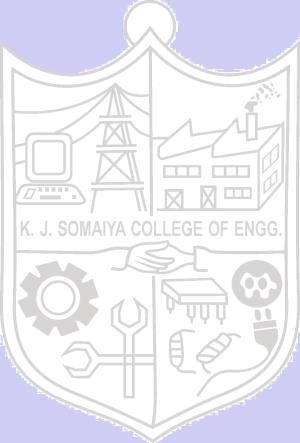
## Server Side:

import socket import random

HOST = "127.0.0.1"

PORT = 65432

def primitive\_root(p):

number\_list = [numbers for numbers in range(1,p)] prime\_number\_list = list(sympy.primerange(2,p))

for prime\_num in prime\_number\_list: checking\_number\_list = []

for num in number\_list: checking\_number\_list.append((prime\_num \*\* (num -1)) % p) checking\_number\_list.sort()

if number\_list == checking\_number\_list: return prime\_num

def key\_calculation(value, random\_num, p): return ((value \*\* random\_num) % p)

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s: s.bind((HOST, PORT))

s.listen()

conn, addr = s.accept() with conn:

print(f"Connected by {addr}") p = conn.recv(1024)

p = int(p.decode()) g = conn.recv(1024) g = int(g.decode())

print(f"Received p ({p}) and g ({g}).") Ra = conn.recv(1024)

Ra = int(Ra.decode()) print(f"Received Ra ({Ra}).")

b = random.randint(1, p - 1) Rb = key\_calculation(g, b, p)

conn.send(str(Rb).encode()) print(f"Sent Rb ({Rb}).")

Kab = key\_calculation(Ra, b, p) print("Shared key on server side: " , Kab)

## Client Side:

import socket import sympy import random

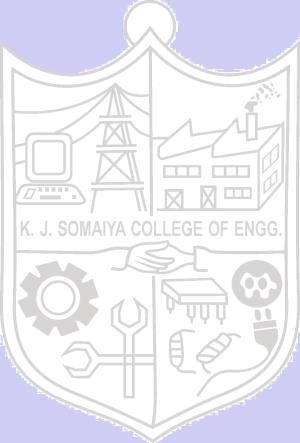
HOST = "127.0.0.1"

PORT = 65432

def primitive\_root(p):

number\_list = [numbers for numbers in range(1,p)] prime\_number\_list = list(sympy.primerange(2,p))

for prime\_num in prime\_number\_list: checking\_number\_list = []

for num in number\_list: checking\_number\_list.append((prime\_num \*\* (num -1)) % p) checking\_number\_list.sort()

if number\_list == checking\_number\_list: return prime\_num

def key\_calculation(value, random\_num, p): return ((value \*\* random\_num) % p)

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s: s.connect((HOST, PORT))

p = int(input("Please enter a prime number: ")) g = primitive\_root(p)

s.send(str(p).encode())

s.send(str(g).encode()) print(f"Sent p ({p}) and g ({g}).")

a = random.randint(1, p - 1) Ra = key\_calculation(g, a, p)

s.send(str(Ra).encode())

print(f"Sent Ra ({Ra}).") Rb = s.recv(1024)

Rb = int(Rb.decode()) print(f"Received Rb ({Rb}).") Kab = key\_calculation(Rb, a, p)

print("Shared key on client side: " , Kab)

# Output:

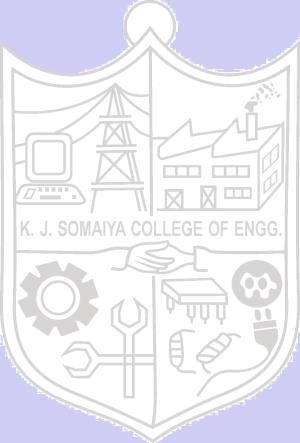
## Server Side:

**ClientSide:**



## Questions:

1. **Explain any one attack on Diffie-Hellman key exchange protocol.**

**Answer:** Man-in-the-Middle attack is very much possible on the existing Diffie- Hellman algorithm. In a man-in-the-middle attack, the attacker exists in the public channel, the attacker receives the public key of both sender and receiver and sends public keys to the sender and receiver which is generated by his own. This is how a man-in-the-middle attack is possible on the Diffie-Hellman algorithm. Denial of service attack is another attack that is found common on Diffie-Hellman. In this attack, the attacker tries to stop the communication happening between sender and receiver and the attacker can do this by deleting messages or by confusing the parties with miscommunication.

## Discuss the possible solution to mitigate the attack.

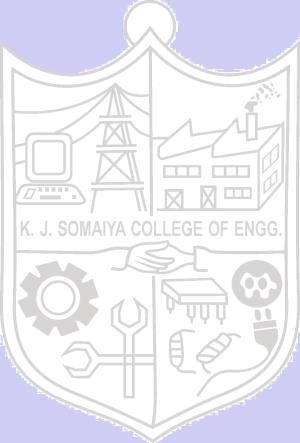
**Answer:** To mitigate Man-in-the-Middle attacks in the Diffie-Hellman key exchange protocol, we can implement the following solutions:

* 1. **Use Public Key Infrastructure (PKI):** Implement a PKI system to distribute and verify public keys. This involves using digital certificates issued by trusted Certificate Authorities (CAs). Each party can sign their public keys with their private keys and share the public keys along with the digital certificates. The other party can verify the certificate's authenticity and public key's integrity. If the certificate is valid, they can trust the public key for key exchange.
  2. **Key Pinning:** Implement key pinning at the client's end. Key pinning involves hardcoding the server's public key or certificate in the client application. This way, even if an attacker tries to replace the server's public key during the MitM attack, the client will still use the hardcoded key for the key exchange.
  3. **Out-of-Band Verification:** Ensure that the public keys are exchanged out-of-band (e.g., through a secure channel such as in-person meetings or secure phone calls). This way, even if an attacker intercepts the public keys in transit, they won't be able to modify them.
  4. **Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP):** Maintain CRLs or use OCSP to check the validity of digital certificates. This can help detect if a certificate has been revoked by the CA, indicating potential tampering.
  5. **Perfect Forward Secrecy (PFS):** Use Diffie-Hellman Ephemeral (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) to provide Perfect Forward Secrecy. With PFS, even if an attacker records the encrypted communication, they cannot decrypt past sessions if they compromise the private key later.

## Outcomes:

CO2: Illustrate different cryptographic algorithms for security.

## Conclusion:

We understood the concept of Diffie-Hellman key exchange algorithm and the types of attacks possible for Diffie-Hellman. We also implemented Diffie-Hellman key exchange algorithm in python.

## 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