

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
Department of Computer Engineering

Course , Subject & Experiment Details

Academic Year	2021-22	Estimated Time	02 - Hours
Course & Semester	T.E. (CMPN)- Sem VI	Subject Name & Code	CSS -
Module No.	02 – Mapped to CO-2	Chapter Title	Key Management Techniques

Practical No:	2
Title:	Implementation of Diffie- Hellman Key exchange algorithm and Simulation of Man In the Middle attack
Date of Performance:	09.02.2022
Date of Submission:	21.02.2022
Roll No:	8875
Name of the Student:	Upmanyu Jha

Evaluation:

Sr. No	Rubric	Grade
1	On time submission Or completion (2)	
2	Preparedness(2)	
3	Skill (4)	

4	Output (2)	
---	------------	--

Signature of the Teacher:

Date:

Title: Implementation of Diffie- Hellman Key exchange algorithm and Simulation of Man In the Middle attack.

Lab Objective :

This lab provides insight into:

- The working of Diffie – Hellman Key Exchange Protocol.

Reference : “Cryptography and Network Security” B. A. Forouzan
“Cryptography and Network Security” Atul Kahate

Prerequisite: Any programming Language and Knowledge of Symmetric Key cryptography.

Theory:

Diffie-Hellman is a way of *generating* a shared secret between two people in such a way that the secret can't be seen by observing the communication.

This is particularly useful because you can use this technique to create an encryption key with someone, and then start encrypting your traffic with that key. And even if the traffic is recorded and later analyzed, there's absolutely no way to figure out what the key was, even though the exchanges that created it may have been visible.

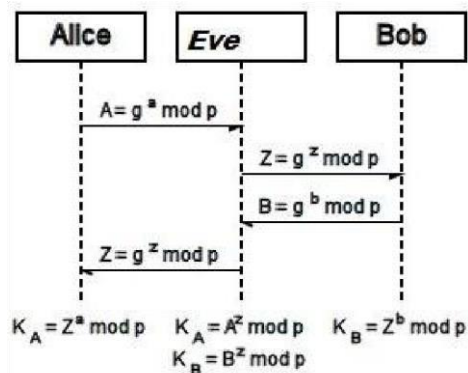
Diffie Hellman Key Exchange

	Alice	Evil Eve	Bob
	Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that $P > G$ and G is Primitive Root of P $G = 7, P = 11$	Evil Eve sees $G = 7, P = 11$	Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that $P > G$ and G is Primitive Root of P $G = 7, P = 11$
Step 1	Alice generates a random number: X_A $X_A = 6$ (Secret)		Bob generates a random number: X_B $X_B = 9$ (Secret)
Step 2	$Y_A = G^{X_A} \pmod{P}$ $Y_A = 7^6 \pmod{11}$ $Y_A = 4$		$Y_B = G^{X_B} \pmod{P}$ $Y_B = 7^9 \pmod{11}$ $Y_B = 8$
Step 3	Alice receives $Y_B = 8$ in clear-text	Evil Eve sees $Y_A = 4, Y_B = 8$	Bob receives $Y_A = 4$ in clear-text
Step 4	Secret Key $= Y_B^{X_A} \pmod{P}$ Secret Key $= 8^6 \pmod{11}$ ✔ Secret Key = 3		Secret Key $= Y_A^{X_B} \pmod{P}$ Secret Key $= 4^9 \pmod{11}$ ✔ Secret Key = 3

Copyright © 2015, Sage AI
http://www.sage-ai.com

Man – In – The –Middle Attack

Let us take the example illustrated by Diffie-Hellman to discuss the Man-in-the-Middle Attack. Let us that Eve is in the middle of Alice and Bob. Eve does not need the value of x or y to attack the protocol. She can fool both Alice and Bob by the following process.



1. Alice choose a, calculate $A = g^a \pmod{p}$
2. Eve, the intruder, intercepts A, she chooses z, calculate $Z = g^z \pmod{p}$, and sends Z to both Alice and Bob.

3. Bob choose b , calculate $B=g^b \text{ mod } p$, and sends B to Alice; B is interpreted by Eve and never reaches Alice.
4. Alice and Eve calculate the same key $g^{az} \text{ mod } p$, which become a shared key between Alice and Eve. Alice however think that it is a key shared between Bob and herself.
5. Eve and Bob calculate the same key $g^{bz} \text{ mod } p$, which become a shared key between Eve and Bob. Bob, however, thinks that it is a key shared between Alice and himself. This situation is called manin-the-middle attack.

Practical and Real Time Applications

- Used as a method of exchanging cryptography keys for **use** in symmetric encryption algorithms like AES
- Public key encryption schemes based on DF – ElGamal encryption
- Password-authenticated key agreement
- public key infrastructure - It is possible to use DF as part of PKI

Conclusion:

The program was tested for different sets of inputs.

Program is working SATISFACTORY NOT SATISFACTORY
(Tick appropriate outcome)

Post Lab Assignment:

1. In the Diffie- Hellman protocol , what happens if x and y have the same value, that is, Alice and Bob have accidentally chosen the same number? Are A and B (values exchanged by Alice and Bob to each other) the same? Do the session keys calculated by Alice and Bob have the same value? Use an example to prove your claims.
2. How to secure Diffie-Hellman from Man-in –the –Middle attack?

Diffie Hellman Implementation Code:

Client:

```
import socket
from random import randint
```

```

s = socket.socket()

port = 12345

s.connect(('127.0.0.1', port))

if __name__ == '__main__':
    P = input('Enter value of P:')
    G = input('Enter value of G:')
    b = input('Enter Private key for Bob:')
    print('The Private Key b for Bob is :%d'%(b))
    y = int(pow(G,b,P))
    s.send(str(y))
    data = s.recv(1024)
    x_ex = int(data)
    print('The value recieved from Alice is: %d'%(x_ex))
    kb = int(pow(x_ex,b,P))
    print('The Secret Key for Bob is : %d'%(kb))
    s.close()

```

Server:

```

import socket

from random import randint

s = socket.socket()
print ("Socket successfully created")

port = 12345

s.bind(('', port))
print ("socket binded to %s" %(port))

s.listen(5)
print ("socket is listening")

```

```

if _name_ == '_main_':
    P = input('Enter value of P:')    G =
input('Enter value of G:')    print('The Value of P
is :%d'%(P))    print('The Value of G is :%d'%(G))
    a = input('Enter Private key for Alice:')
    print('The Private Key a for Alice is :%d'%(a))
    x = int(pow(G,a,P))
    while True:

        c, addr = s.accept()
        print ('Got connection from', addr )

        from_client=""
        while True:

            data = c.recv(1024)

            if not data: break

            from_client += data                y_ex =
int(data)                print('The value recieved from Bob is:
%d'%(y_ex))                ka = int(pow(y_ex,a,P))
            print('The Secret Key for Alice is : %d'%(ka))

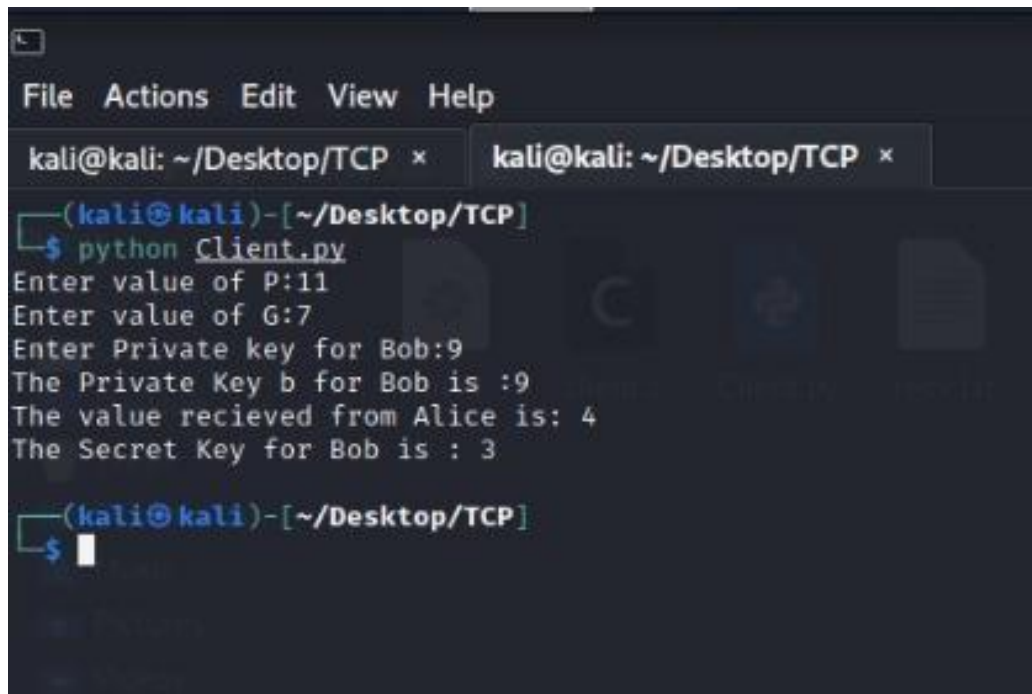
            c.send(str(x))

        c.close()

```

```
print ('Client Disconnected')  
  
break
```

OUTPUT:



```
(kali㉿kali)-[~/Desktop/TCP]  
$ python Client.py  
Enter value of P:11  
Enter value of G:7  
Enter Private key for Bob:9  
The Private Key b for Bob is :9  
The value recieved from Alice is: 4  
The Secret Key for Bob is : 3  
  
(kali㉿kali)-[~/Desktop/TCP]  
$
```

```
File Actions Edit View Help
kali@kali: ~/Desktop/TCP x kali@kali: ~/Desktop/TCP x
(kali@kali)-[~/Desktop/TCP]
$ python Server.py
Socket successfully created
socket binded to 12345
socket is listening
Enter value of P:11
Enter value of G:7
The Value of P is :11
The Value of G is :7
Enter Private key for Alice:6
The Private Key a for Alice is :6
('Got connection from', ('127.0.0.1', 47186))
The value recieved from Bob is: 8
The Secret Key for Alice is : 3
Client Disconnected

(kali@kali)-[~/Desktop/TCP]
$
```


Man In Middle:

Client:

```
from operator import mod
import socket
from multiprocessing import Process
from sympy import true
from time import sleep
n,g = input("Enter 2 prime numbers =").split()
n = int(n)
g = int(g)
x = int(input("Enter a random number x = "))
# Create a client socket
clientSocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# Connect to the server
clientSocket.connect(("127.0.0.1",9091))
clientSocket.send(repr(n).encode())
clientSocket.send(repr(g).encode())
while(True):
    print("Accepted a connection request from server")
    A= mod(g**x,n)
    clientSocket.send(repr(A).encode())
    B =clientSocket.recv(4)
    B = int(B.decode())
    print(B)
    k1 = mod(B**x,n)
    print("Key is = ", k1)
    break
```

Interpreter:

```
from operator import mod
import socket
from multiprocessing import Process
from time import sleep
x = input("Enter a random number x =")
y = input("Enter a random number y =")
# Create a client socket
client1Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client2Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```

client1Socket.bind(("127.0.0.1",9091))
client1Socket.listen()
(clientConnected1, clientAddress1) = client1Socket.accept()
print("Accepted a connection request from
%s:%s"%(clientAddress1[0], clientAddress1[1])) # Connect to the
server
client2Socket.connect(("127.0.0.1",9090))
n = clientConnected1.recv(4)
g = clientConnected1.recv(4)
client2Socket.send(n) client2Socket.send(g)
g = int(g.decode())
n = int(n.decode())
while(True):
    print("Accepted a connection request from server")
    A= mod(g**x,n)
    B = mod(g**y,n)
    client2Socket.send(repr(A).encode())
    clientConnected1.send(repr(B).encode())
    A= client2Socket.recv(4)
    A = clientConnected1.recv(4)
    B = int(B.decode())
    A = int(A.decode())
    k1 = mod(B**x,n)
    k2 = mod(A**y,n)
    print("Server Key is = ", k1)
    print("Client key is = ", k2)
    break

```

Server:

```

import socket from
numpy import mod from
sympy import true from
time import sleep
# Create a stream based socket(i.e, a TCP socket)
# operating on IPv4 addressing scheme y = int(input("Enter a
random number y = ")) serverSocket =
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind and listen serverSocket.bind(("127.0.0.1",9090))
serverSocket.listen()
# Accept connections
(clientConnected, clientAddress) =
serverSocket.accept() n = clientConnected.recv(4)
print(n) g = clientConnected.recv(4) print(n,g) g =
g.decode() n = n.decode() while(True):
    print("Accepted a connection request from
%s:%s"%(clientAddress[0], clientAddress[1]))    print(n,g)    B =
mod(g**y, n)    print(B)
    clientConnected.send(repr(B).encode())
    A =
clientConnected.recv(4)    A
= int(A.decode())    k2 =
mod(A**y, n)    print("Key is
= ", k2)    break

```

Output:

```
===== RESTART: D:\Python\ManInMiddle\client1.py =====
Enter 2 prime numbers = 11 3
Enter a random number x = 43
Accepted a connection request from server
97
Key is = 97
```

```
===== RESTART: D:\Python\ManInMiddle\intercepter.py =====
Enter a random number x = 4
Enter a random number y = 8
Accepted a connection request from 127.0.0.1:51001
Accepted a connection request from server
Server Key is = 109
Client key is = 97
```

```
===== RESTART: D:\Python\ManInMiddle\servermim.py =====
Enter a random number y = 8
b'113'
b'113' b'5'
Accepted a connection request from 127.0.0.1:51002
97
Key is = 109
```

Conclusion:

The program was tested for different sets of inputs.

Program is working	SATISFACTORY	NOT SATISFACTORY
(Tick appropriate outcome)		

Post Lab Assignment:

Q1. In the Diffie-Hellman protocol, what happens if x and y have the same value, that is, Alice and Bob have accidentally chosen the same number? Are A and B (values exchanged by Alice and Bob to each other) the same? Do the session keys calculated by Alice and Bob have the same value? Use an example to prove your claims.

Answer:

If x and y have the same value, then A and B will also be the same. The session keys calculated by Bob and Alice will also be the same.

```
===== RESTART: D:\Python\client.py =====  
Enter 2 prime numbers = 7 3  
Enter a random number x = 5  
Accepted a connection request from server  
Session key is: 5  
Key received from server is- 59  
Key is = 5
```

```
===== RESTART: D:\Python\server.py =====  
Enter a random number 5  
Accepted a connection request from 127.0.0.1:57162  
Session Key is = 59  
5  
Final Key is = 59
```

Q2. How to secure Diffie-Hellman from Man-in –the –Middle attack?**Answer:**

The key exchange process is a very important feature for many parts in cryptography. In data encryption, symmetric encryption techniques need to share the same secret key securely between two parties before encryption and this is a challenging task for secure data transmission. Most symmetric key encryptions and key management systems widely use Diffie-Hellman Key Exchange (DHKE) algorithm for the purpose of key distribution because it has simple computation and supports forward security. However, there is no key authentication and Man-In-The-Middle (MITM) attack has occurred during the key generation process. To overcome this problem, a new hash function is proposed to get the public key integrity during the public key sharing process of DHKE algorithm. This hash function is created by using six bitwise operators and operated in a variable length of the rounds depending on message length. Thus, the proposed system improves the security of DHKE and grants the user authentication requirements.