**Batch: B1 Roll No.: 1411072**

**Experiment No. 4**

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

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:** Man in the Middle |

**Objective:**

To study Diffie Hellman algorithm and implement a man-in-middle attack for a encipherment system which functions using that algorithm.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 2 | Identify the principles of various cryptographic techniques and apply various cryptographic algorithms for securing systems. |

**Books/ Journals/ Websites referred:**

1. Cryptography and Network Security – Behrouz A. Forouzan

**Abstract**:

This document implements the Diffie Hellman key exchange algorithm, wherein Alice and Bob have to exchange their private keys. But, an eavesdropper, Eve tries to disrupt the system’s integrity by providing wrong keys to Alice and Bob.

**Related Theory:**

The Diffie Hellman Key exchange algorithm:

Diffie–Hellman key exchange (D–H) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols. Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical channel, such as paper key lists transported by a trusted courier. 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 channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo p, where p is prime, and g is a primitive root modulo p. These two values are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to p–1.

Example:

1. [Alice and Bob](https://en.wikipedia.org/wiki/Alice_and_Bob) agree to use a modulus *p* = 23 and base *g* = 5 (which is a [primitive root modulo](https://en.wikipedia.org/wiki/Primitive_root_modulo_n) 23).
2. Alice chooses a secret integer ***a*** = **6**, then sends Bob *A* = *g****a*** % *p  
   A* = 5**6** % 23 = 8
3. Bob chooses a secret integer ***b*** = **15**, then sends Alice *B* = *g****b*** % *p*  
   *B* = 5**15** % 23 = 19
4. Alice computes ***s*** = *B****a*** % *p*  
   ***s*** = 19**6** % 23 = **2**
5. Bob computes ***s*** = *A****b*** % *p****s*** = 8**15** % 23 = **2**
6. Alice and Bob now share a secret (the number **2**).

Both Alice and Bob have arrived at the same value s.

How man in middle attack is carried out?

The man in middle attack aims to capture the key exchange from Alice and Bob, the entities that communicate the key amongst themselves, and provide them with a wrong key.

Eve has her own private number – ‘z’. She calculates gz % p.

When she receives communication from Alice, she sends Bob this calculated value. Bob responds with his calculated value. Now, Eve forwards her calculated value to Alice.

Alice & Bob are unaware of the fact that the values received are not from their intended senders, and thus proceed to calculate the key, which is faulty. Thus, Eve disrupts the system.

**Implementation Details:**

* The attack is simulated in python, using socket programming & multi-threading.
* There are 3 files:
  + Eve.py  
    She is the eavesdropper who intends to meddle with the key exchange between Alice & Bob.
  + Senders.py  
    Alice & Bob are simulated using this script.
  + Public.py  
    All the common values required by the program are stored here for easy future manipulation.
* There are 2 servers and 2 clients, in all, in the simulation.
* Eve.py must be up as the server at the commencement of the simulation.
* Senders.py uses a random binary value to determine who (Alice/Bob) has to start the key exchange. The one who starts is the client and the other is the server. Both are implemented using threads.
* Client sends its calculated value to Server, which is shown to be intercepted by Eve. She now becomes a client and connects to the server, and sends her manipulated value. The server responds with its calculated value, and Eve replies the client, who started the exchange, with her manipulated value.
* The importance of my approach:
  + Socket Programming helps build a virtual system consisting of two different systems which are trying to communicate with each other.
  + Multithreading helps to simulate real-time simulation, just as two random systems may do.
  + The 2 server 2 client method, where Eve changes from Server to Client, makes the simulation more practical.

**Implementation:**

* ***Eve.py***

import socket

import public

import random as r

from struct import \*

"""

first: Who starts sending first.

second: To whom does first send.

z: Private value with Eve.

val\_to\_be\_sent: The manipulated value that Eve sends.

"""

first = "Alice"

second = "Bob"

z = r.randint(2, public.MAX\_VAL)

val\_to\_be\_sent = (public.g \*\* z) % public.p

s = socket.socket()

host = socket.gethostname()

s.bind((host, public.port\_eve))

print "Eve is up & running\n"

print "Manipulated value by Eve: "+str(val\_to\_be\_sent)+"\n"

s.listen(1)

client, \_ = s.accept()

who\_starts\_first = int(client.recv(1024))

client.send("ok")

if who\_starts\_first is 1:

first = "Bob"

second = "Alice"

val = int(client.recv(1024))

print "Received "+str(val)+" from "+first

s.close()

s = socket.socket()

port = public.get\_port\_no(who\_starts\_first)

connected\_to = public.get\_name(port)

print "Eve connecting to "+second

s.connect((host, port))

s.send(str(val\_to\_be\_sent))

print "Sent "+str(val\_to\_be\_sent)+" to "+second

val = int(s.recv(1024))

print "Received "+str(val)+" from "+second

s.close()

print "Sending "+str(val\_to\_be\_sent)+" to "+first

client.send(str(val\_to\_be\_sent))

client.close()

* ***Senders.py***

import public as p

import socket

import thread as th

import random as r

private\_alice = r.randint(2, p.MAX\_VAL)

private\_bob = r.randint(2, p.MAX\_VAL)

def server(who, port, private\_no):

print who+" is up & running as server\n"

s = socket.socket()

host = socket.gethostname()

s.bind((host, port))

s.listen(1)

client, \_ = s.accept()

k1 = int(client.recv(1024))

print(who+" received "+str(k1))

k = (k1\*\*private\_no) % p.p

print "The secret key with "+who+" is: "+str(k)

val = (p.g\*\*private\_no)%p.p

print who+" sent "+str(val)

client.send(str(val))

client.close()

def client(who, port, private\_no):

print who+" is up & running as client.\n"

s = socket.socket()

host = socket.gethostname()

s.connect((host, port))

val = (p.g\*\*private\_no) % p.p

print(who+" sent "+str(val))

s.send(str(who\_starts\_first))

s.recv(1024)

s.send(str(val))

k1 = int(s.recv(1024))

print(who+" received "+str(k1))

k = k1\*\*private\_no % p.p

print("The secret key with "+who+" is: "+str(k))

s.close()

who\_starts\_first = r.randint(0,1)

"""

who\_starts\_first:

0 -> Alice starts sending to Bob.

1 -> Bob starts sending to ALice.

"""

if who\_starts\_first == 0:

th.start\_new\_thread(client, ("Alice", p.port\_eve, private\_alice,) )

th.start\_new\_thread(server, ("Bob", p.port\_bob, private\_bob,) )

else:

th.start\_new\_thread(client, ("Bob", p.port\_eve, private\_bob,) )

th.start\_new\_thread(server, ("Alice", p.port\_alice, private\_alice,) )

########## Very Very Important for threads to abstain from termination! ############

\_ = raw\_input()

* ***Public.py***

p = 11

g = 7

port\_alice = 7777

port\_eve = 7780

port\_bob = 7779

s = 1

MAX\_VAL=25

ports = {1: port\_alice, 0: port\_bob}

names = {port\_alice: 'Alice', port\_bob: 'Bob'}

def get\_port\_no(whose):

return ports[whose]

def get\_name(which\_port):

return names[which\_port]

**Conclusion:-**

Thus we have successfully implemented the man in middle attack for a system facililtated with Diffie Hellman Key Exchange algorithm.

**Date: 17/08/2017 Signature of faculty in-charge**