PES
UNIVERSITY

COMPUTER NETWORKS

UE17EC351

COURSE PROJECT

DISTRIBUTED PASSWORD CRACKING SYSTEM USING PYTHON SOCKET PTOGRAMMING

under the guidance of

Prof. RAJASEKAR.M

SUBMITTED BY:

SHREYAS.R

(PES1201701714)

SHREYAS KULKARNI

(PES1201700450)

INDEX

1. INTRODUCTION

This project aims at making a distributed system that can crack the password generated by the server. It consists of set of operations on small chunks of data and no need for data sharing between the cracking nodes. The cracking system consists of one server and two clients running on a single system for our ease. The clients check for the password in their respective range and then give the confirmation that the password has ben found. We can give a user defined password or a randomly generated one.

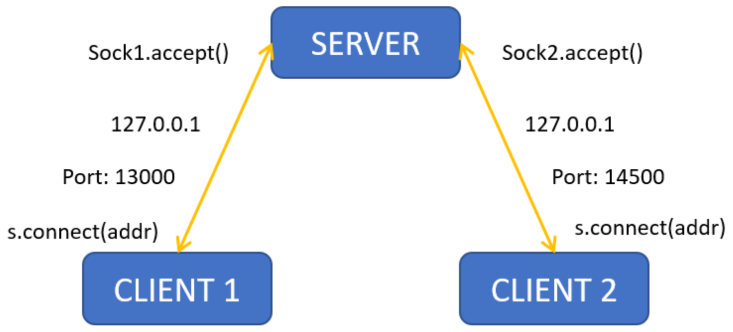
2. PROBLEM STATEMENT

Server generates a random/user defined alphanumeric password of length 4. The two clients that are connected to the server must decode the hashed password and give the output that it decodes.

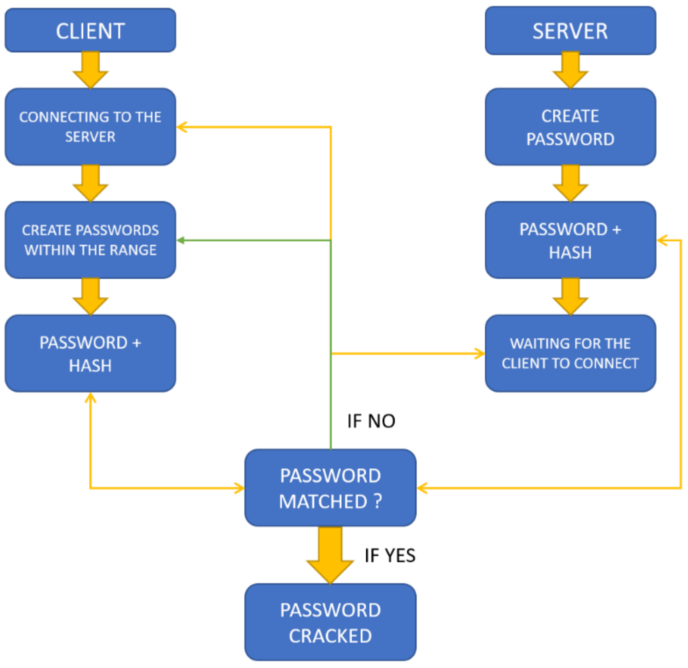
3. PROCEDURE

This project contains two codes, server code and the clients code. The server and the two clients are connected to the same IP address. But the clients have their own port numbers to communicate with the server. The generated password is encrypted using hashing technique. Next the total number of outcomes of the password is calculated. This total number of outcomes is the divided by the number of clients. The clients get the range in which they are supposed to search for the password. For an alphanumeric code of length 4, the total outcomes are 36^4. The reason why we use this number is we have 10 digits counting from 0-9 and 26 alphabets i.e. A-Z. 4th power is because the length of the password is 4. This is also called ‘base 36’. This gives 167916 possibilities. The clients connected via IP address have a ‘for’ loop to generate the passwords within their ranges. A new password combination is generated during each iteration and is hashed immediately and then it checked with the server generated password. If it does not match, then another iteration is made and checked for the next combination. This process continues until the right password is found. If the password is found, then the client gives the output telling that the password was found and also displays the password. The server gives the output telling which client has cracked the code. The process is the same in both the clients except for the fact that their port numbers are different.

4. NETWORK DIAGRAM



5. WORKFLOW



6. CODE

➢ SERVER

```
#importing all the libraries

import os

import random

from socket import *

import hashlib

import time

#variable declaration and initialisation

host = ""

port1 = 13000

port2 = 14500

buf = 1024

password = ""

addr1 = (host,port1)

addr2 = (host,port2)

#Creating sockets and making connections with the client

Sock1 = socket(AF_INET,SOCK_STREAM)

Sock1.bind(addr1)

Sock1.listen(10)

Sock2 = socket(AF_INET,SOCK_STREAM)

Sock2.bind(addr2)

Sock2.listen(10)

print("Waiting for client 1 to start.....")

print("Waiting for client 2 to start.....")

(newsock1,addr1) = Sock1.accept()

(newsock2,addr2) = Sock2.accept()

print("Connection established from " +str(addr1))

print("Connection established from " +str(addr2))

#Code logic

password = str(input("Enter a 4 digit password with capital letters and digits: "))

print("The password is : "+password)

hashvalue = hashlib.md5(password.encode())
```

```
finalvalue = hashvalue.hexdigest()
print("The hex representation of password = "+finalvalue)
range1_l = "0"
range1_h = "839807"
range2_l = "839808"
range2_h = "1679615"
newsock1.send(finalvalue.encode())
newsock1.send(range1_l.encode())
newsock2.send(finalvalue.encode())
newsock2.send(range2_l.encode())
time.sleep(2)
newsock1.send(range1_h.encode())
newsock2.send(range2_h.encode())
print("Hash and range sent")
print("Waiting for clients to process and find the password ......")
time.sleep(2)
print("hello")
reply1 = newsock1.recv(buf).decode()
reply2 = newsock2.recv(buf).decode()
if reply1 != "Password not found" :
    print("Password found !!!")
    print("Password found by client 1 : "+reply1)
if reply2 != "Password not found" :
    print("Password found !!!")
    print("Password found by client 2 : "+reply2)
Sock1.close()
Sock2.close()
os._exit(0)
```

➢ CLIENT1

```
#importing all the libraries
import os
import hashlib
from socket import *
#variable declaration and initialisation
host = "127.0.0.1" # set to IP address of target computer
port = 13000
addr = (host, port)
buf = 16384
buf2 = 32
flag = 0
password = ""
#Conversion of base 10 numbers to base 36 ranging from 0-9 & A-Z
def base36encode(number):
    alphabet='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    base36 = ''
    if 0 <= number < len(alphabet):
        return alphabet[number]
    while number != 0:
        number, i = divmod(number, len(alphabet))
        base36 = alphabet[i] + base36
    return base36
#Creating sockets and making connections with the server
s = socket(AF_INET,SOCK_STREAM)
s.connect(addr)
print("Connection established with "+host)
password_hash=str(s.recv(buf).decode())
print ("The password hash is "+password_hash)
password_range_l=str(s.recv(buf2).decode())
password_range_h=str(s.recv(buf2).decode())
print ("The password range is
("+base36encode(int(password_range_l))+","+base36encode(int(password_range_h))+")")
```

```
#Code Logic
for x in range(int(password_range_l),int(password_range_h)):
    password = base36encode(x)
    if len(password) == 1:
        password = "000"+password
    if len(password) == 2:
        password = "00"+password
    if len(password) == 3:
        password = "0"+password
    hashvalue = hashlib.md5(password.encode())
    finalvalue = hashvalue.hexdigest()
    if str(finalvalue) == password_hash :
        flag = 1
        break
if flag==1:
    print("Password found !!!!")
    s.send(password.encode())
else:
    print("Password not found")
    s.send("Password not found".encode())
s.close()
os._exit(0)
```

- <u>CLIENT2</u>

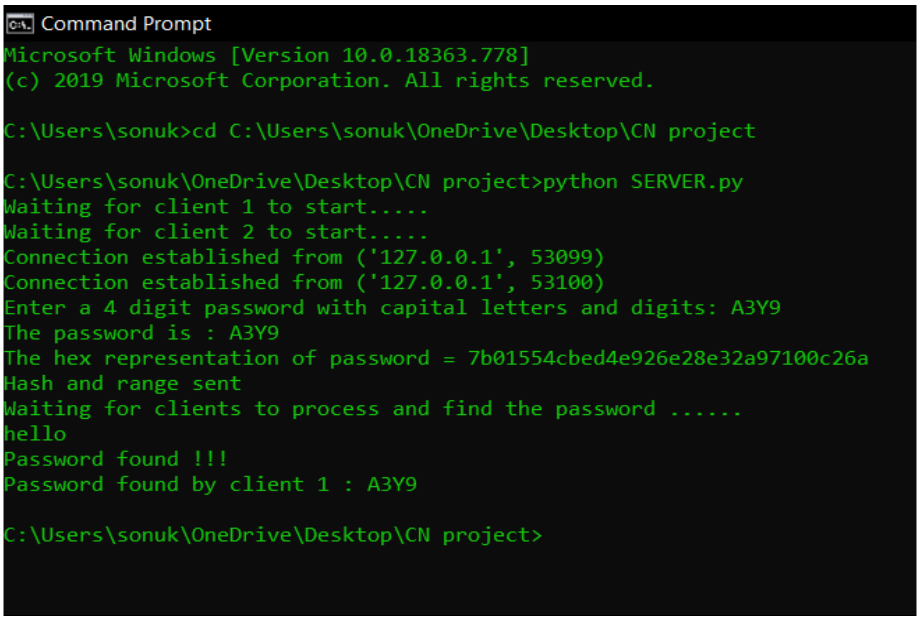
```
#importing all the libraries
import os
from socket import *
import hashlib
#variable declaration and initialisation
host = "127.0.0.1" # set to IP address of target computer
port = 14500
addr = (host, port)
buf = 16384
```

```
buf2 = 32
password = ""
flag = 0
#Conversion of base 10 numbers to base 36 ranging from 0-9 & A-Z
def base36encode(number):
    alphabet='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    base36 = ''
    if 0 <= number < len(alphabet):
        return alphabet[number]
    while number != 0:
        number, i = divmod(number, len(alphabet))
        base36 = alphabet[i] + base36
    return base36
#Creating sockets and making conncetions with the server
s = socket(AF_INET,SOCK_STREAM)
s.connect(addr)
print("Connection established with "+host)
password_hash=str(s.recv(buf).decode())
print ("The password hash is "+password_hash)
password_range_l=str(s.recv(buf2).decode())
password_range_h=str(s.recv(buf2).decode())
print ("The password range is
("+base36encode(int(password_range_l))+","+base36encode(int(password_range_h))+")")
#Code Logic
for x in range(int(password_range_l),int(password_range_h)):
    password = base36encode(x)
    hashvalue = hashlib.md5(password.encode())
    finalvalue = hashvalue.hexdigest()
    if str(finalvalue) == password_hash :
        flag = 1
        break
if flag==1:
    print("Password found !!!!")
    s.send(password.encode())
```

```
else:
    print("Password not found")
    s.send("Password not found".encode())
s.close()
os._exit(0)
```

7. OUTPUTS

➢ SERVER.py



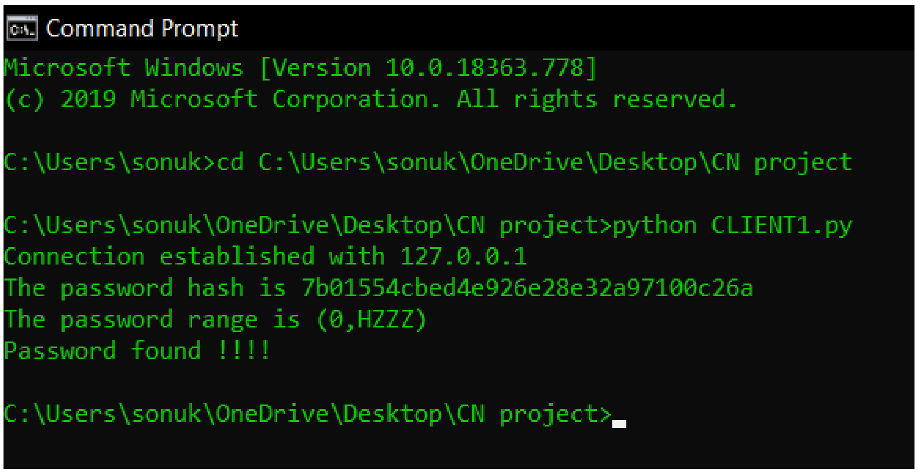
```
Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\sonuk>cd C:\Users\sonuk\OneDrive\Desktop\CN project

C:\Users\sonuk\OneDrive\Desktop\CN project>python SERVER.py
Waiting for client 1 to start.....
Waiting for client 2 to start.....
Connection established from ('127.0.0.1', 53099)
Connection established from ('127.0.0.1', 53100)
Enter a 4 digit password with capital letters and digits: A3Y9
The password is : A3Y9
The hex representation of password = 7b01554cbed4e926e28e32a97100c26a
Hash and range sent
Waiting for clients to process and find the password ......
hello
Password found !!!
Password found by client 1 : A3Y9

C:\Users\sonuk\OneDrive\Desktop\CN project>
```

➢ CLIENT1.py



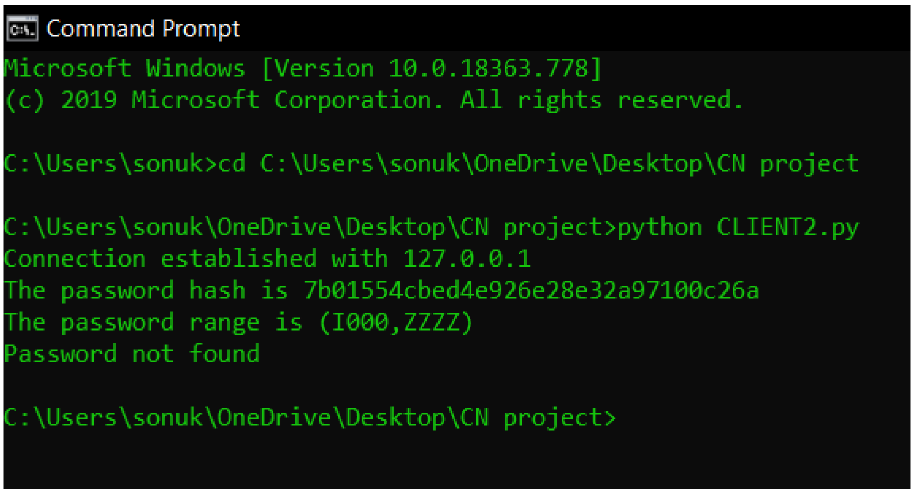
```
Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\sonuk>cd C:\Users\sonuk\OneDrive\Desktop\CN project

C:\Users\sonuk\OneDrive\Desktop\CN project>python CLIENT1.py
Connection established with 127.0.0.1
The password hash is 7b01554cbed4e926e28e32a97100c26a
The password range is (0,HZZZ)
Password found !!!!

C:\Users\sonuk\OneDrive\Desktop\CN project>
```

➢ CLIENT2.py



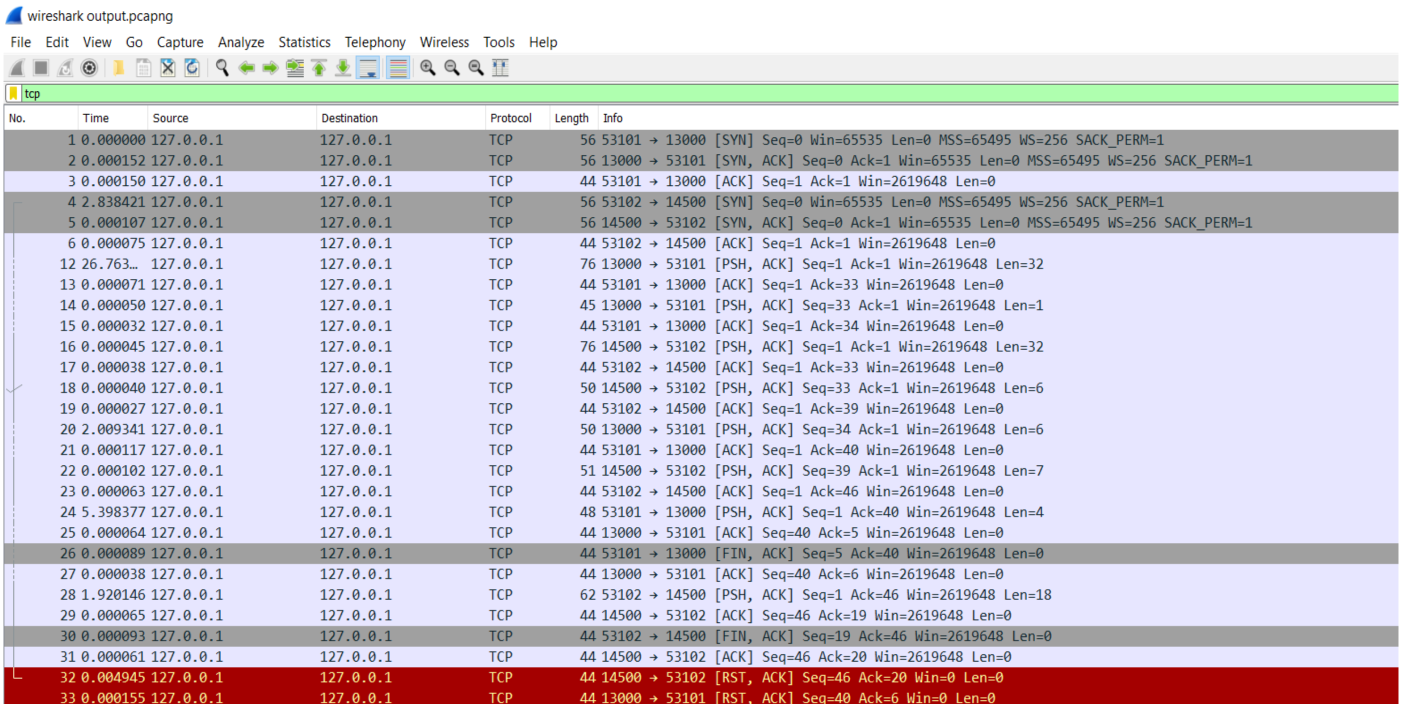
```
Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\sonuk>cd C:\Users\sonuk\OneDrive\Desktop\CN project

C:\Users\sonuk\OneDrive\Desktop\CN project>python CLIENT2.py
Connection established with 127.0.0.1
The password hash is 7b01554cbed4e926e28e32a97100c26a
The password range is (I000,ZZZZ)
Password not found

C:\Users\sonuk\OneDrive\Desktop\CN project>
```

8. WIRESHARK OUTPUT



wireshark output.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	53101 → 13000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000152	127.0.0.1	127.0.0.1	TCP	56	13000 → 53101 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000150	127.0.0.1	127.0.0.1	TCP	44	53101 → 13000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	2.838421	127.0.0.1	127.0.0.1	TCP	56	53102 → 14500 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
5	0.000107	127.0.0.1	127.0.0.1	TCP	56	14500 → 53102 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
6	0.000075	127.0.0.1	127.0.0.1	TCP	44	53102 → 14500 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
12	26.763…	127.0.0.1	127.0.0.1	TCP	76	13000 → 53101 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=32
13	0.000071	127.0.0.1	127.0.0.1	TCP	44	53101 → 13000 [ACK] Seq=1 Ack=33 Win=2619648 Len=0
14	0.000050	127.0.0.1	127.0.0.1	TCP	45	13000 → 53101 [PSH, ACK] Seq=33 Ack=1 Win=2619648 Len=1
15	0.000032	127.0.0.1	127.0.0.1	TCP	44	53101 → 13000 [ACK] Seq=1 Ack=34 Win=2619648 Len=0
16	0.000045	127.0.0.1	127.0.0.1	TCP	76	14500 → 53102 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=32
17	0.000038	127.0.0.1	127.0.0.1	TCP	44	53102 → 14500 [ACK] Seq=1 Ack=33 Win=2619648 Len=0
18	0.000040	127.0.0.1	127.0.0.1	TCP	50	14500 → 53102 [PSH, ACK] Seq=33 Ack=1 Win=2619648 Len=6
19	0.000027	127.0.0.1	127.0.0.1	TCP	44	53102 → 14500 [ACK] Seq=1 Ack=39 Win=2619648 Len=0
20	2.009341	127.0.0.1	127.0.0.1	TCP	50	13000 → 53101 [PSH, ACK] Seq=34 Ack=1 Win=2619648 Len=6
21	0.000117	127.0.0.1	127.0.0.1	TCP	44	53101 → 13000 [ACK] Seq=1 Ack=40 Win=2619648 Len=0
22	0.000102	127.0.0.1	127.0.0.1	TCP	51	14500 → 53102 [PSH, ACK] Seq=39 Ack=1 Win=2619648 Len=7
23	0.000063	127.0.0.1	127.0.0.1	TCP	44	53102 → 14500 [ACK] Seq=1 Ack=46 Win=2619648 Len=0
24	5.398377	127.0.0.1	127.0.0.1	TCP	48	53101 → 13000 [PSH, ACK] Seq=1 Ack=40 Win=2619648 Len=4
25	0.000064	127.0.0.1	127.0.0.1	TCP	44	13000 → 53101 [ACK] Seq=40 Ack=5 Win=2619648 Len=0
26	0.000089	127.0.0.1	127.0.0.1	TCP	44	53101 → 13000 [FIN, ACK] Seq=5 Ack=40 Win=2619648 Len=0
27	0.000038	127.0.0.1	127.0.0.1	TCP	44	13000 → 53101 [ACK] Seq=40 Ack=6 Win=2619648 Len=0
28	1.920146	127.0.0.1	127.0.0.1	TCP	62	53102 → 14500 [PSH, ACK] Seq=1 Ack=46 Win=2619648 Len=18
29	0.000065	127.0.0.1	127.0.0.1	TCP	44	14500 → 53102 [ACK] Seq=46 Ack=19 Win=2619648 Len=0
30	0.000093	127.0.0.1	127.0.0.1	TCP	44	53102 → 14500 [FIN, ACK] Seq=19 Ack=46 Win=2619648 Len=0
31	0.000061	127.0.0.1	127.0.0.1	TCP	44	14500 → 53102 [ACK] Seq=46 Ack=20 Win=2619648 Len=0
32	0.004945	127.0.0.1	127.0.0.1	TCP	44	14500 → 53102 [RST, ACK] Seq=46 Ack=20 Win=0 Len=0
33	0.000155	127.0.0.1	127.0.0.1	TCP	44	13000 → 53101 [RST, ACK] Seq=40 Ack=6 Win=0 Len=0

<u>Output Explanation</u>:

Here we are explaining the Wireshark output based on the serial numbers. Every time the client receives the data, it sends a confirmation message to the server by setting the ACK bit to len+1.

- **Sl nos.1,2** and **3** show the 3-way handshake between Client1 and the Server. 53101 is the port number of Client1 and 13000 is that of the server. The client connects to the server and the server sends back an ACK bit. Finally the client sets the SEQ bit and the handshake is complete.

- **Sl nos.4,5** and **6** show the 3-way handshake between the Server and Client2. 53102 is the port number of Client2 and 14500 is that of server. The handshake procedure is the same as in previous case.
- **Sl no.12** shows that the hashed password in the server is sent to Client1. This is a 32 bytes data.
- Then the Client1 sends an ACK bit to the Server telling that the hashed password has been received. This is shown in **Sl no.13**.
- The Server then sets the lower limit of the ranges that the clients must work on. In **Sl no.14** [PSH,ACK] denotes that a chunk of data has been pushed from server to client1.
- The 32 bytes hashed password is sent to Client2 from the Server. This is shown in **Sl no.16**.
- Now Client2 gets the lower limit of the range that it must work on. **Sl no.18** shows this process.
- Now we give the upper limits of the ranges to clients 1 and 2. **Sl nos.20** and **22** signify this.
- **Sl no.24** shows that the password has been detected by Client1 and it is pushed from Client1 to the Server. Hence len=4.
- [FIN, ACK] in **Sl no.26** denotes that the connection is closed between Client1 and the Server
- In **Sl no.28**, we can observe that len=18, i.e. Client2 is pushing the data ‘Password not found’ to the Server.
- [FIN, ACK] in **Sl no.30** closes the connection between Client2 and the Server.
- **Sl nos.32** and **33** make sure that the client-server connections are no more valid.

9. STEPS TO RUN THE FILES

I. First we run the SERVER.py python file on the command prompt within the defined directory.
II. Then run the client files namely CLIENT1.py and CLIENT2.py.
III. The server will ask the user to give a password of length 4.
IV. While doing this the client programs should be running in parallel.
V. Once the password is entered, it is left to the clients to find out the hashed code.
VI. It gives the output as shown in the previous images.

10. REFERENCES

a) Socket programming – python.org (https://docs.python.org/3/howto/sockets.html)
b) GeeksforGeeks – (https://www.geeksforgeeks.org/socket-programming-python/)
c) GitHub – (https://github.com/Mahedi-61/Distributed-Password-Cracker)