21CY681 - INTERNET PROTOCOL LAB - XII

Name: Surya S Nair

Register Number: CB.EN.P2CYS22007

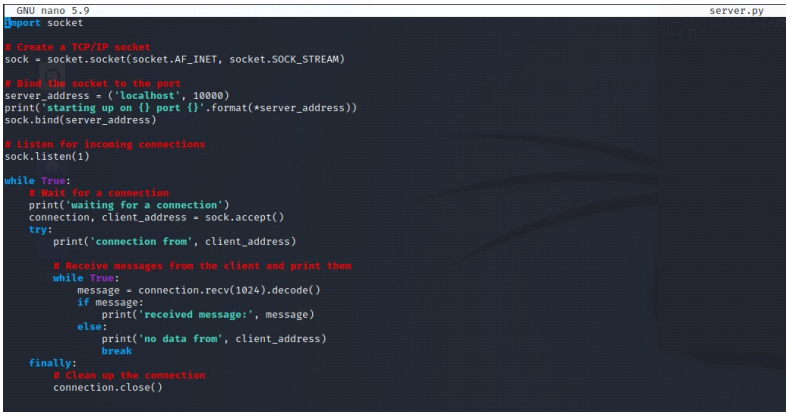
Date: 26th December 2022

Assignment Topic: Application of Cryptogrpahical Algorithms using Socket programming

Aim: To study about bittorrent and to analyse using wireshark

Normal Client and Server Connection

- Below is the Python program for server for establishing a connection between client and server.



- Below is the python program for client for establishing a connection between client and server.

```
GNU nano 5.9                                                client.py
import socket

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = ('localhost', 10000)
print('connecting to {} port {}'.format(*server_address))
sock.connect(server_address)

try:
    while True:
        # Read a message from the user and send it to the server
        message = input("Enter a message to send to the server (enter 'q' to quit): ")
        if message == 'q':
            break
        sock.sendall(message.encode())
finally:
    sock.close
```

- This is the scapy captured pcap which is being shown in wireshark.

```
▸ Frame 11: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 51278, Dst Port: 10000, Seq: 3, Ack: 1, Len: 5
▸ [2 Reassembled TCP Segments (7 bytes): #7(2), #11(5)]
▾ Data (7 bytes)
    Data: 686968656c6c6f
    [Length: 7]

0000  68 69 68 65 6c 6c 6f                              hihello
```

Client and Server connection using RSA Encryption key

This shows a python program for the server where RSA encryption key is being used to make the message encrypted.

```
GNU nano 5.9                                                server.py
import socket
import rsa

# Generate a new 2048-bit RSA key pair
(pubkey, privkey) = rsa.newkeys(2048)

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = ('localhost', 10000)
print('starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        # Receive the client's public key
        client_pubkey = rsa.PublicKey.load_pkcs1(connection.recv(1024))

        # Send the server's public key to the client
        connection.sendall(rsa.PublicKey.save_pkcs1(pubkey))

        # Receive encrypted messages from the client and decrypt them using the server's private key
        while True:
            encrypted_message = connection.recv(1024)
            if encrypted_message:
                message = rsa.decrypt(encrypted_message, privkey).decode()
                print('received message:', message)
            else:
                print('no data from', client_address)
                break
    finally:
        # Clean up the connection
        connection.close()
```

This shows a python program for the client where RSA encryption key is being used to make the message encrypted.

```
 GNU nano 5.9                                                                client.py
import socket
import rsa

# Generate a new 2048-bit RSA key pair
(pubkey, privkey) = rsa.newkeys(2048)

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = ('localhost', 10000)
print('connecting to {} port {}'.format(*server_address))
sock.connect(server_address)

try:
    # Send the client's public key to the server
    sock.sendall(rsa.PublicKey.save_pkcs1(pubkey))

    # Receive the server's public key
    server_pubkey = rsa.PublicKey.load_pkcs1(sock.recv(1024))

    while True:
        # Read a message from the user and send it to the server
        message = input("Enter a message to send to the server (enter 'q' to quit): ")
        if message == 'q':
            break
        encrypted_message = rsa.encrypt(message.encode(), server_pubkey)
        sock.sendall(encrypted_message)
finally:
    sock.close()
```

Now we are using Scapy which is a tool used here to capture the traffic between the client and server.

```
>>> capture = sniff(iface="lo", count=20)
^C>>> capture.summary()
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin S
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin S
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 SA
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 SA
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 PA / Raw
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:39354 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:39354 A
>>> wrpcap("Socketprogramming.pcap", capture)
>>>
```

When the client program and server programs is running in parallel, the connection is being is made and the message is being sent.

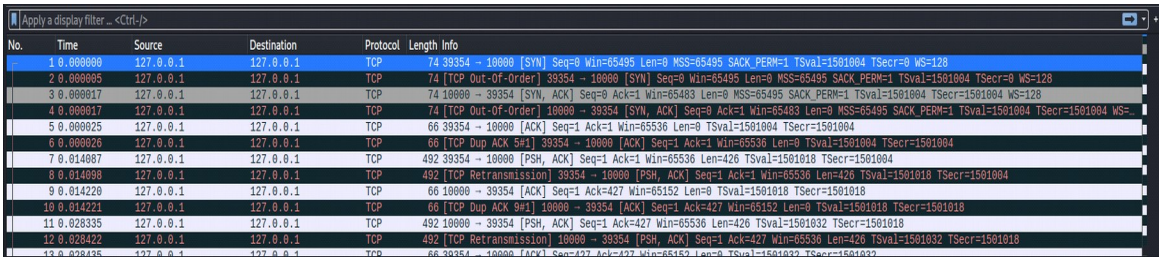
```
┌──(kali㉿kali)-[~/Desktop/IP]
└─$ python3 client.py
connecting to localhost port 10000
Enter a message to send to the server (enter 'q' to quit): hi
Enter a message to send to the server (enter 'q' to quit): hello
Enter a message to send to the server (enter 'q' to quit):
```

When the server is connected it receives messages from client.

```
┌──(kali㉿kali)-[~/Desktop/IP]
└─$ python3 server.py
starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 39354)
received message: hi
received message: hello
```

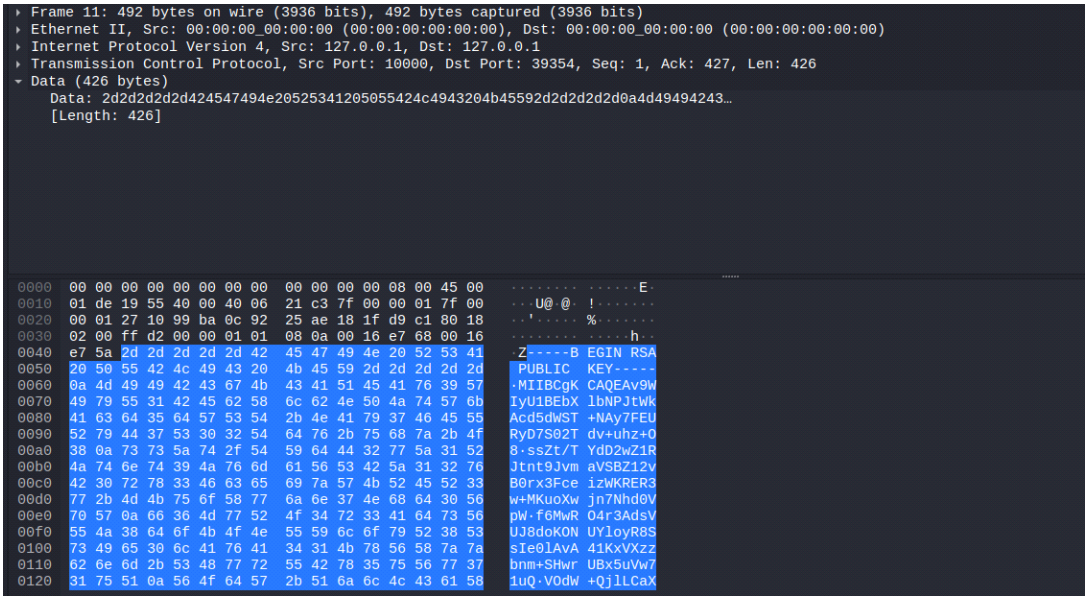
The scapy captured file is being saved it. The same pcap file is opened in

Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	39354 → 10000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1501004 TSecr=0 WS=128
2	0.000005	127.0.0.1	127.0.0.1	TCP	74	[TCP Out-Of-Order] 39354 → 10000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1501004 TSecr=0 WS=128
3	0.000017	127.0.0.1	127.0.0.1	TCP	74	10000 → 39354 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1501004 TSecr=1501004 WS=128
4	0.000017	127.0.0.1	127.0.0.1	TCP	74	[TCP Out-Of-Order] 10000 → 39354 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1501004 TSecr=1501004 WS=…
5	0.000025	127.0.0.1	127.0.0.1	TCP	66	39354 → 10000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1501004 TSecr=1501004
6	0.000026	127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 5#1] 39354 → 10000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1501004 TSecr=1501004
7	0.014087	127.0.0.1	127.0.0.1	TCP	492	39354 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=426 TSval=1501018 TSecr=1501004
8	0.014098	127.0.0.1	127.0.0.1	TCP	492	[TCP Retransmission] 39354 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=426 TSval=1501018 TSecr=1501004
9	0.014220	127.0.0.1	127.0.0.1	TCP	66	10000 → 39354 [ACK] Seq=1 Ack=427 Win=65152 Len=0 TSval=1501018 TSecr=1501018
10	0.014221	127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 9#1] 10000 → 39354 [ACK] Seq=1 Ack=427 Win=65152 Len=0 TSval=1501018 TSecr=1501018
11	0.028335	127.0.0.1	127.0.0.1	TCP	492	10000 → 39354 [PSH, ACK] Seq=1 Ack=427 Win=65536 Len=426 TSval=1501032 TSecr=1501018
12	0.028422	127.0.0.1	127.0.0.1	TCP	492	[TCP Retransmission] 10000 → 39354 [PSH, ACK] Seq=1 Ack=427 Win=65536 Len=426 TSval=1501032 TSecr=1501018

When we open a file we get the data.12



```
▸ Frame 11: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits)
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 10000, Dst Port: 39354, Seq: 1, Ack: 427, Len: 426
▾ Data (426 bytes)
    Data: 2d2d2d2d2d424547494e20525341205055424c4943204b45592d2d2d2d2d0a4d49494243…
    [Length: 426]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  01 de 19 55 40 00 40 06  21 c3 7f 00 00 01 7f 00   ···U@·@· !·······
0020  00 01 27 10 99 ba 0c 92  25 ae 18 1f d9 c1 80 18   ··'····· %·······
0030  02 00 ff d2 00 00 01 01  08 0a 00 16 e7 68 00 16   ········ ·····h··
0040  e7 5a 2d 2d 2d 2d 2d 42  45 47 49 4e 20 52 53 41   ·Z-----B EGIN RSA
0050  20 50 55 42 4c 49 43 20  4b 45 59 2d 2d 2d 2d 2d    PUBLIC  KEY-----
0060  0a 4d 49 49 42 43 67 4b  43 41 51 45 41 76 39 57   ·MIIBCgK CAQEAv9W
0070  49 79 55 31 42 45 62 58  6c 62 4e 50 4a 74 57 6b   IyU1BEbX lbNPJtWk
0080  41 63 64 35 64 57 53 54  2b 4e 41 79 37 46 45 55   Acd5dWST +NAy7FEU
0090  52 79 44 37 53 30 32 54  64 76 2b 75 68 7a 2b 4f   RyD7S02T dv+uhz+O
00a0  38 0a 73 73 5a 74 2f 54  59 64 44 32 77 5a 31 52   8·ssZt/T YdD2wZ1R
00b0  4a 74 6e 74 39 4a 76 6d  61 56 53 42 5a 31 32 76   Jtnt9Jvm aVSBZ12v
00c0  42 30 72 78 33 46 63 65  69 7a 57 4b 52 45 52 33   B0rx3Fce izWKRER3
00d0  77 2b 4d 4b 75 6f 58 77  6a 6e 37 4e 68 64 30 56   w+MKuoXw jn7Nhd0V
00e0  70 57 0a 66 36 4d 77 52  4f 34 72 33 41 64 73 56   pW·f6MwR O4r3AdsV
00f0  55 4a 38 64 6f 4b 4f 4e  55 59 6c 6f 79 52 38 53   UJ8doKON UYloyR8S
0100  73 49 65 30 6c 41 76 41  34 31 4b 78 56 58 7a 7a   sIe0lAvA 41KxVXzz
0110  62 6e 6d 2b 53 48 77 72  55 42 78 35 75 56 77 37   bnm+SHwr UBx5uVw7
0120  31 75 51 0a 56 4f 64 57  2b 51 6a 6c 4c 43 61 58   1uQ·VOdW +QjlLCaX
```