

Name:- MOHAMMED TOUSIF
Roll No:- CB.EN.P2CYS22008

APPLICATION OF CRYPTOGRAPHICAL ALGORITHMS USING SOCKET PROGRAMMING

Date :- 05-01-2023

Aim :- To create an application of cryptographical algorithms using socket programming.

Tools Required :- Linux OS , Scapy , Wireshark.

Procedure :-

Creating a server and client socket programs.

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()
```

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()
```

Client - Server communication

```
└─$ python3 client.py
connecting to localhost port 10000
Enter a message to send to the server (enter 'q' to quit): Client
Enter a message to send to the server (enter 'q' to quit): msg
Enter a message to send to the server (enter 'q' to quit): q
```

```

(tousif@kalitousif)-[~/sockets]
$ python3 server.py
starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 41186)
received message: Client
received message: msg
no data from ('127.0.0.1', 41186)
waiting for a connection

```

Now we capture the communication through scapy :

Using **capture=sniff(iface='lo')** command to sniff the packets.
The captured packets will be:

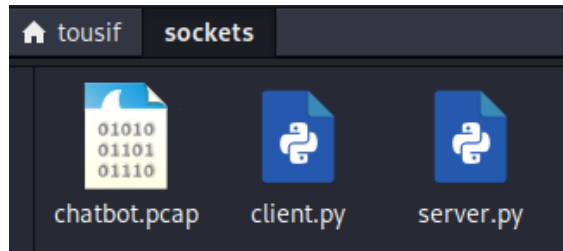
```

>>> capture.summary()
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 S
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 S
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 SA
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 SA
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 PA / Raw
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 PA / Raw
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 PA / Raw
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 FA
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 FA
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 FA
Ether / IP / TCP 127.0.0.1:10003 > 127.0.0.1:54320 FA
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
Ether / IP / TCP 127.0.0.1:54320 > 127.0.0.1:10003 A
>>>

```

Saving the captured traffic as pcap file.

```
Ether / IP / UDP 192.168.59.1:63910 > 239.255.255.250:1900 / Raw
>>> wrpcap("chatbot.pcap",capture)
```



Analyzing the packets captured.

Time	Source	Destination	Protocol	Length	Info
1	2023/005 22:39:21.406868 127.0.0.1	127.0.0.1	TCP	74	54320 → 10003 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=2945
2	2023/005 22:39:21.406875 127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 54320 → 10003 [SYN] Seq=0
3	2023/005 22:39:21.406922 127.0.0.1	127.0.0.1	TCP	74	10003 → 54320 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM
4	2023/005 22:39:21.406924 127.0.0.1	127.0.0.1	TCP	74	[TCP Retransmission] 10003 → 54320 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=
5	2023/005 22:39:21.406935 127.0.0.1	127.0.0.1	TCP	66	54320 → 10003 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2945653267 TSecr=2
6	2023/005 22:39:21.406943 127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 5#1] 54320 → 10003 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=
7	2023/005 22:39:21.461836 127.0.0.1	127.0.0.1	TCP	492	54320 → 10003 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=426 TSval=2945653322
8	2023/005 22:39:21.461852 127.0.0.1	127.0.0.1	TCP	492	[TCP Retransmission] 54320 → 10003 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=
9	2023/005 22:39:21.461865 127.0.0.1	127.0.0.1	TCP	66	10003 → 54320 [ACK] Seq=1 Ack=427 Win=65152 Len=0 TSval=2945653322 TSecr=
10	2023/005 22:39:21.461866 127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 9#1] 10003 → 54320 [ACK] Seq=1 Ack=427 Win=65152 Len=0 TSva
11	2023/005 22:39:21.514186 127.0.0.1	127.0.0.1	TCP	492	10003 → 54320 [PSH, ACK] Seq=1 Ack=427 Win=65536 Len=426 TSval=294565337
12	2023/005 22:39:21.514273 127.0.0.1	127.0.0.1	TCP	492	[TCP Retransmission] 10003 → 54320 [PSH, ACK] Seq=1 Ack=427 Win=65536 Le
13	2023/005 22:39:21.514294 127.0.0.1	127.0.0.1	TCP	66	54320 → 10003 [ACK] Seq=427 Ack=427 Win=65152 Len=0 TSval=2945653374 Tse
14	2023/005 22:39:21.514295 127.0.0.1	127.0.0.1	TCP	66	[TCP Dup ACK 13#1] 54320 → 10003 [ACK] Seq=427 Ack=427 Win=65152 Len=0
15	2023/005 22:39:25.734483 127.0.0.1	127.0.0.1	TCP	322	54320 → 10003 [PSH, ACK] Seq=427 Ack=427 Win=65536 Len=256 TSval=2945657
16	2023/005 22:39:25.734532 127.0.0.1	127.0.0.1	TCP	322	[TCP Retransmission] 54320 → 10003 [PSH, ACK] Seq=427 Ack=427 Win=65536

▼ Data (256 bytes)

Data: 2a49938ef42a42562cecdcc6fffd25bcb728a9bfd76dc4b1798a087693a4f3afed37ecd28...
[Length: 256]

0040	2a fa 2a 49 93 8e f4 2a 42 56 2c ec dd c6 ff f2	*.I...* BV,....
0050	5b cb 72 8a 9b fd 76 dc 4b 17 98 a0 87 69 3a 4f	[.r...v. K...i:0
0060	3a fe d3 7e cd 28 09 43 05 a4 50 94 50 4a fb 6e	:...~(.C ..P.PJ.n
0070	80 a4 99 cf 54 0f 24 0d 36 a1 97 4f df a7 61 6dT.\$ 6..0..am
0080	08 f1 e4 ed 15 62 eb 4c df 01 c7 45 0a ea ea 63b.L ...E...c
0090	3c 21 ea 7a 9c 77 0e c7 6f be e9 c4 ea 68 6b 91	<!·z·w·· o...hk·
00a0	b9 a1 db eb 27 51 37 ab 4c 57 42 db 3c a2 07 86'Q7· LWB<...·
00b0	6e 57 d1 3c 0b db ca 73 45 90 5f dc c4 55 ca 75	nW·<...s E·...U·u
00c0	42 39 c5 96 43 d9 21 ab 16 e0 4c 99 5d 2b a2 8b	B9·C·!· ..L·]+..
00d0	b9 33 55 06 80 bb cf a8 45 60 ca 4b ce 19 4c 35	·3U·.... E`·K·L5
00e0	af da 5d 39 ff 7e 53 a9 9a 65 eb 08 bd 8c 6d f0	..]9~S· ·e....m·
00f0	fd 93 65 ac 13 04 0d 63 4c 2c cc 40 f6 22 fe d0	·e....c L,·@·"...
0100	4e 0c d6 b3 1e b2 d9 97 13 e2 a1 a0 28 17 43 e6	N·..... ..(.C·

RSA Encryption :

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAuCM2zDPdEveF1hHm17/z1q/zRhN055PklytkhqF3S2RBTExmu3ru
XfJJth/n9PV4EhRrNIvTggK9aqxojdMNwDFrKN4b7k8R59s4xMeZxvyYY4hcttHl
MHPu7V9xyKa0DqXrnbjNH+93JUHPgHmf4FjqF2jpV79P52TjBKUoGnoyZAbiceBh
NItxLkL07Hv200dDrGM30F12VTOSomxDkVhBYC7fu2xDyuixwLd+bayvkKHdGk8Z
USFuVa7K7amebs7Z/9+8g40Zj2m4aKaCjiDsmkXBmQl05NivcTPP1Ih18mPYqStS
yemHYBNnzRubI29/4kH7UDs78s81GFarBQIDAQAB
-----END RSA PUBLIC KEY-----
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAjfl9go9rJOv5S7FCYhS7NwpgjTjiCiHcMtipynFFsPN019cB2pPi
/xJKEa/yr5NEUyHjXPTs2JsbUkOu0BpL9TUJXNzEBPxUsFszs94uVL60SZIOdVAO
j1MQzPbNnxBgNWmjrfNUzDzMH73+N1FGKGNu5nr9HrCEwASSPp210xds0THcYTHu
5TdrpW6/p8aKwGs0Mi1YGRTlW6Q2R/bho2tdlvnDAoHBSNY2vUQ6kADD7CKSXA2f
eX0p9eDLYNZgOIgC015L7c91qp8tmMC/XfGCRT+wPuyZC1vitMNV2kTGN7bn5rni
1SYJIUNNwSGdvstyoFcDFrPeu/88qa03QIDAQAB
-----END RSA PUBLIC KEY-----
.a.....8..'.....<;..)k7...{...aB>...+hz.V...&..[.R`.W'
H.\.1q..W.....a1..8..k,
..      a%...R...yA.+J*...<.E.1.wd....l..7n....S...IYs[....BHL0.....9.....V% .A.....}._.w.2E...x..|O.....X.|
D..>...].<A^D=L\9.....WCy....o...?.....i.r.U.b...X.kA=..a..|...*I...*BV,.....[.r...v.K....i:O:..~.(
C..P.PJ.n....T.$
6..O..am....b.L...E
...<!..z.w.o....hk....'Q7.LWB.<...nW.<...sE._..U.uB9..C.!...L.]++...3U.....E`.K..L5..]9.^S...e....m...e...
cL,._@..".N.....(.C.....t....      Toi      ...k...?dv...fJ...#>.]L:{.@.3bb.....
```

Above screenshot shows captured packet of RSA Public key.