

NAME: S.PRIYADHARSHINI

ROLL NO.: CB.EN.P2CYS22010

DATE: 26.12.2022

INTERNET PROTOCOL LAB –11

APPLICATION OF CRYPTOGRAPHICAL ALGORITHMS USING SOCKET PROGRAMMING

AIM:

To apply RSA algorithm using socket programming and sniffing the packets between client and server.

TOOLS REQUIRED:

Ubuntu 16, scapy, wireshark.

PROCEDURE:

We are writing a python program for the server (server.py) where RSA encryption key is being used to encrypt the message received from the client.

```
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()
```

We are writing a python program for the client (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()
```

For using RSA package in python:

Upgrade pip3 and install rsa. If it works fine when importing rsa then this program can be executed.

Pip3 install –upgrade pip

Pip install rsa

python3

>>> import rsa

>>>

To install scapy for sniffing packets:

pip3 install scapy

1. First, open a terminal and start running scapy. Start sniffing the packets of a particular interface.
2. Open another terminal separately for server and client. Start running server.py. When it is waiting for a connection start running client.py.
3. Send message from client to server. If it is received in the server then, the execution is successful.
4. Stop the packet capture in Scapy. Store the packets captured in a pcap file and analyze it using Wireshark.

client.py

```
root@VM:/home/seed# python3 client.py
connecting to localhost port 10000
Enter a message to send to the server (enter 'q' to quit): first message
Enter a message to send to the server (enter 'q' to quit): second message
Enter a message to send to the server (enter 'q' to quit): third message
Enter a message to send to the server (enter 'q' to quit): q
root@VM:/home/seed#
```

server.py

```
root@VM:/home/seed# python3 server.py
starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 53990)
received message: first message
received message: second message
received message: third message
no data from ('127.0.0.1', 53990)
waiting for a connection
```

```
>>> pkts=sniff(iface="lo",count=75)
^C>>>
>>> pkts.summary()
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin S
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin S
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 SA
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 SA
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 PA / Raw
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin A
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:53990 > 127.0.0.1:webmin PA / Raw
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
Ether / IP / TCP 127.0.0.1:webmin > 127.0.0.1:53990 A
```

The sniffed packets are being saved in a pcap file using wrpcap command.

```
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IPv6 / UDP ::1:45616 > ::1:38251
Ether / IPv6 / UDP ::1:45616 > ::1:38251
>>> wrpcap("pgm1.pcap",pkts)
>>>
```

These packets can be analyzed using wireshark.

5	2023/005	23:30:23.274608	127.0.0.1	53990	127.0.0.1	10000	TCP	74	53990 → 10000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM TSval=
6	2023/005	23:30:23.274621	127.0.0.1	53990	127.0.0.1	10000	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 53990 → 10000 [SYN]
7	2023/005	23:30:23.274630	127.0.0.1	10000	127.0.0.1	53990	TCP	74	10000 → 53990 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK
8	2023/005	23:30:23.274630	127.0.0.1	10000	127.0.0.1	53990	TCP	74	[TCP Retransmission] 10000 → 53990 [SYN, ACK] Seq=0 Ack=1 Win=43690
9	2023/005	23:30:23.274638	127.0.0.1	53990	127.0.0.1	10000	TCP	66	53990 → 10000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=4294925748 TS
10	2023/005	23:30:23.274638	127.0.0.1	53990	127.0.0.1	10000	TCP	66	[TCP Dup ACK 9#1] 53990 → 10000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 T
11	2023/005	23:30:23.317444	127.0.0.1	53990	127.0.0.1	10000	TCP	492	53990 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=426 TSval=429492
12	2023/005	23:30:23.317448	127.0.0.1	53990	127.0.0.1	10000	TCP	492	[TCP Retransmission] 53990 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=43776
13	2023/005	23:30:23.317529	127.0.0.1	10000	127.0.0.1	53990	TCP	66	10000 → 53990 [ACK] Seq=1 Ack=427 Win=44800 Len=0 TSval=4294925759
14	2023/005	23:30:23.317530	127.0.0.1	10000	127.0.0.1	53990	TCP	66	[TCP Dup ACK 13#1] 10000 → 53990 [ACK] Seq=1 Ack=427 Win=44800 Len=
15	2023/005	23:30:23.352248	127.0.0.1	10000	127.0.0.1	53990	TCP	492	10000 → 53990 [PSH, ACK] Seq=1 Ack=427 Win=44800 Len=426 TSval=4294
16	2023/005	23:30:23.352253	127.0.0.1	10000	127.0.0.1	53990	TCP	492	[TCP Retransmission] 10000 → 53990 [PSH, ACK] Seq=1 Ack=427 Win=448

Frame 11: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits)	0000	00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E-
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:	0010	01 de 63 bb 40 00 40 06 d7 5c 7f 00 00 01 7f 00	..c.@.:\.....
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	00 01 d2 e6 27 10 16 03 3e 0e 57 da cd 18 80 18>W.....
Transmission Control Protocol, Src Port: 53990, Dst Port: 10000, Seq: 1, Ack: 1, Len: 426	0030	01 56 ff d2 00 00 01 01 08 0a ff ff 5d bf ff ff	.V.....[...]
Data (426 bytes)	0040	5d b4 2d 2d 2d 2d 2d 42 45 47 49 4e 20 52 53 41]-----B EGIN RSA
Data: 2d2d2d2d2d4547494e20525341205055424c4943204b45592d2d2d2d0a4d49494243...	0050	20 50 55 42 4c 49 43 20 4b 45 59 2d 2d 2d 2d 2d	PUBLIC KEY-----
[Length: 426]	0060	0a 4d 49 49 42 43 67 4b 43 41 51 45 41 6d 6b 72	+MIIBCgK CAQEAkrr
	0070	48 4e 39 63 67 71 78 43 6b 46 32 59 63 42 77 57	HN9cgqxC kF2YcBwW
	0080	31 6f 63 54 59 77 72 48 71 6d 64 6a 73 6c 70 6b	locTYwH qmdjslpk
	0090	62 2f 2b 64 58 49 77 2f 2b 49 30 7a 64 4f 53 4e	b/+dXlw/ +IOzd05M
	00a0	31 0a 53 43 5a 57 50 69 4f 6d 42 51 65 64 6c 62	1-SCZwP1 OmDQedIb
	00b0	4c 6f 66 4d 39 6e 74 74 68 77 54 4a 50 6c 47 31	LofM9ntt hwTJP1G1
	00c0	57 64 39 55 51 4e 76 30 6b 4a 53 7a 41 48 4a 66	Wd9UQNv0 kJSzAHJf
	00d0	33 44 56 70 4c 64 61 79 69 38 6a 75 73 34 6c 79	3DVpt.day i8jus4ly
	00e0	34 4d 0a 48 31 4a 61 68 5a 4e 5a 76 49 4e 37 38	4M-H1Jah ZNZvIN78
	00f0	78 4b 31 34 31 53 4a 47 49 49 50 77 51 77 45 30	xK141S3G ITPwWf0

RESULT:

Thus, socket programming has been executed and the packets has been sniffed using scapy successfully.