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.

PROCDURE:

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

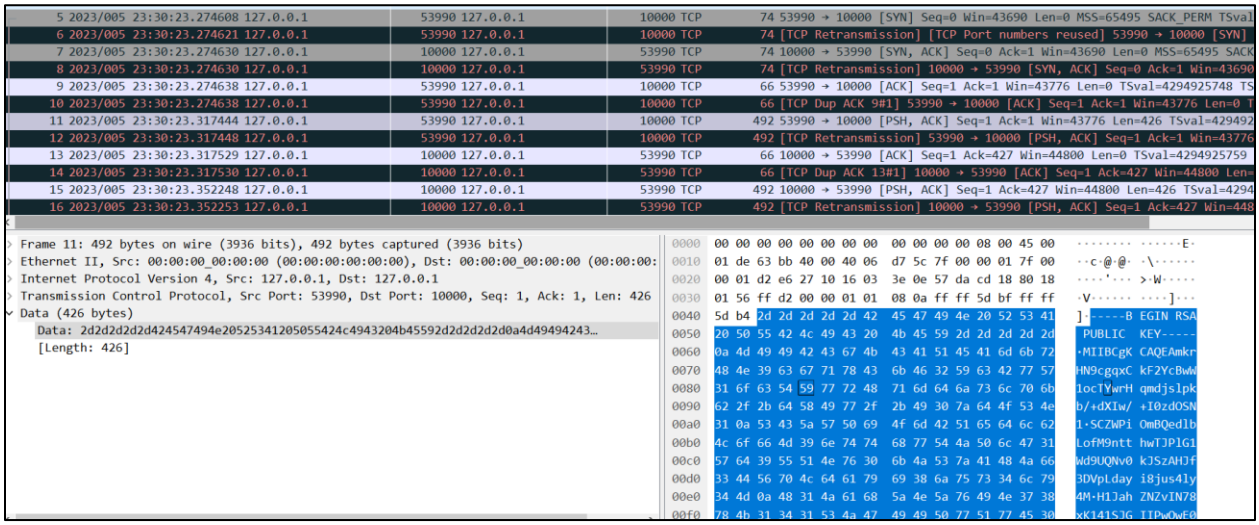
```
root@VM:/home/seed# python3 server.py
starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 53990)
received message: first message
received message: second message
received message: third message
no data from ('127.0.0.1', 53990)
waiting for a connection
```

```
>>> 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.



RESULT:

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