Term Work - 02

Title : Application network security - RSA algorithm

Objective : The objective is to encrypt a message entered by the user and the decrypt it using the public and private key.

Theory :

RSA is an assymetric cryptographic algorithm, means it works on two different keys i.e public key and private key. As the name describes the public key is given to everyone and the private key is kept private.

RSA derives its security from the difficulty of guessing the public and private keys from unauthorized access makers - RSA security relies on the computational diffiulty of factoring large integers encryption strength is directly tied to key size.

Encryption : The conversion of original message into a secrete code or CIPHER TEXT using key.

Decryption : The conversion of encoded message or cipher text back to the original message using same key.

Algorithm :

1. Select 2 prime numbers , preferably large , p and q
2. Calculate $n = p * q$
3. Calculate $phi(n) = (p-1) * (q-1)$
4. Choose a value of e such that $1 < e < phi(n)$ and $gcd(phi(n), e) = 1$
5. Calculate d such that $d = (e^{-1}) \bmod phi(n)$
   - Here the public key is $\{e, n\}$ and private key is $\{d, n\}$
   - If M is the plain text then the cipher text $C = (M^e) \bmod n$. This is how data is encrypted in RSA algorithm.
   - Similarly, for decrypted, the plain text $M = (C^d) \bmod n$.

Source Code :

```
import random
import math
# Function to check if a number is prime
def is_prime(num):
    if num <= 1:
        return False
    for i in range (2, int (math.sqrt(num)) + 1):
        if num % 1 == 0:
            return False
    return True

# Function to generate random prime numbers
def generate_prime(bits):
    while True:
        num = random.getrandombits(bits)
        if is_prime(num):
            return num

# Function to compute the greatest common divisor (GCD)
def gcd (a,b):
    while b:
        a, b = b, a % b
    return a

# Function to find the modular multiplication inverse
def mod_inverse(a,m):
    m0, x0 , x1 = m, 0, 1
    while a>1:
        q = a//m
```

```
        m, a = a%m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

# Function to generate RSA key pairs
def mod_inverse(a, m):
def generate_key_pair(bits):
    p = generate_prime(bits)
    q = generate_prime(bits)
    n = p * q
    phi = (p-1) * (q-1)
    while True:
        e = random.randint(2, phi-1)
        if gcd(e, phi) == 1:
            break
    d = mod_inverse(e, phi)
    public_key = (n, e)
    private_key = (n, d)
    return public_key, private_key

# Function to encrypt a message
def encrypt(public_key, message):
    n, e = public_key
    cipher_text = [pow(ord(char), e, n) for char in message]
    return cipher_text


# Function to decrypt a message
def decrypt(private_key, cipher_text):
    n, d = private_key
```

```
    decrypted_message = ''.join([chr(pow(char, d, n)) for char in cipher_text])
    return decrypted_message

# main program
if __name__ == "__main__":
    bits = 8  # adjust the number of bits for your desired security level
    public_key, private_key = generate_key_pair(bits)
    print(f"Generated public key: {public_key}\nGenerated Private key
        : {private_key}")
    message = eval(input("Enter the Message to be Encrypted: "))
    print("Original message:", message)
    encrypted_message = encrypt(public_key, message)
    decrypted_message = decrypt(private_key, encrypted_message)
    print("Decrypted message:", decrypted_message)
```

Outcomes:

1. Learnt utilization of RSA encryption to secure sensitive data during transmission.
2. Implementation of secure RSA key generation fraction to maintain the overall security of applications network.

Conclusion: RSA algorithm is using easy to implement, confidential, data can be transmitted safely and securly using RSA algorithm. The algorithm involves a lot of complex mathematics which makes it more difficult to work.

Reference:

James F kurose and Keith W Ross, Computer Networking, A top down Approach 8th edition.

Term Work 03

Title : UDP - Connectionless Transport

Objective : Creating a datagram socket for sending data to the server enables continuous interactions, allowing user to send multiple messages. Receive datagram from client and extract the data for processing, providing a straigut-forward communications mechanism without the complenities of connection oreented protocols.

Theory :

User datagram Protocol (UDP) is one of the core transport layer protocols in computer networks, providing a connectionless and digutweigut communication mechanism. UDP sacrifies certain features for efficiency and speed in data transmission UDP is not explicitly establishes a connection before data is sent, it speeds up connection and communication. The enables data delivers to be quick but it also result in packet luing lost in data transmit. In UDP the procedure about handshake is skipped in Communications.

Handshake is a way for two devices to ensures that they are Speaking the same protocol and will be able to understand each other.

Algorithm:

UDP Server

1. Create a UDP socket
2. Bind the socket to server address
3. Wait until the datagram packet arrives from the client
4. Process the datagram packet and send a reply to the client
5. Go back to Step 3.

UDP Client

1. Create a UDP server
2. Send a message to server
3. Wait until a message response from the server is received.
4. Process the reply and go back to step 2, if necessary.
5. Use socket description and exit.

Source code :

```
# UDP Server code

import socket
#Create a UDP Socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Bind the socket to an address and port
server_address = ('local host', 12345)
server_socket.bind(server_address)
print('UDP server is waiting for messages...")
while True:
    # Receive data from client
    data, client_address = server_socket.recvfrom(1024)
    print(f' Received message from {client_address}:{data.decode()}')
# close the socket (this will never be reached in the example)
server_socket.close()
```

```
UDP Client code
import a UDP socket
#create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#server address and port
server_address = ('localhost', 12345)
while True:
    message = input('Enter a message')
    # send data to the server
    client_socket.sendto(message.encode(), server_address)
#close the socket (this will never be reached in this example)
client_socket.close()
```

Outcome :

1. Learnt Operation of UDP is in a connectionless manner, resulting in connection not being established
2. Enable efficient communication to multiple recipients simultaneously through broadcasting.

Conclusion :

The UDP is client server program and is connectionless accomadating a large number of clients without the resource intensive connection management associated with connection oriented protocols. It may not guarantee reliability its focus approach makes its strength.

References :

James F Kurose and Keith W Ross, Computer Networking : A top Down Approach, 8th edition.

Term Work 04

Title : TCP connection oriented transport

Objective : To ensure a reliable and ordered delivery of data by establishing a connection and functioning providing robust error detection and correction.

Theory: TCP connection oriented transport protocol, suits providing a connection oriented transport layer reliable communication between devices on a network. It establishes a connection between sender and reciever before data exchange, ensuring reliable and orderly transmission. It ensures reliable data delivery through acknowlegment mechanism and automatic retransmission of lost packets. TCP utilize and a 3 way handshake process during connection establishment involving SYN, SYN-ACK, AKK packets to initialize parameters and synchronize sender and reciever.

Algorithm: TCP Server

1. Create(), Create TCP socket
2. using bind(), Bind the socket to server address
3. using listen(), put the server socket in a passive mode, wait for client to approach the server to make the connection
4. using accept(), At this point, connection is established, ready for data transfer.
5. Go back to step 3.

TCP Client

1. Create TCP socket

2. Connect newly created client to server.

Program:

```
#TCP Server Code
import socket
# Define the server address and port
server_address = ('127.0.0.1',12345)
# create a socket object
Server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#Bind the socket to the server address and port
server_socket.bind(server_address)
#Listen for incoming connections
server_socket.listen(4)
print("Server is listening for incoming connections....")
#Accept the connection
client_socket, client_address = server_socket.address()
#print(f"connected to{client_address}")
# Receive data from the client
data = client_socket.recv(1024)
print(f"Received data from client :{data.decode('utf-8')}")
# send a response to the client
response = "Hello, client!"
client_socket.send(response.encode('utf-8'))
# close the sockets
client_socket.close()
server_socket.close()
```

TCP client code

```
import socket
#Define the server address and port
server_address = ('127.0.0.1', 12345)
#Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect to the server
client_socket.connect(server_address)
# send a message to the server
message = "Hello, server!"
client_socket.send(message.encode('utf-8'))
#Receive response from the server
data = client_socket.recv(1024)
print(f"Received response from server: {data.decode('utf-8')}")
# close the socket
client_socket.close()
```

Outcomes:

- Establishing a connection between sender and receiver before data exchange
- Implement flow control mechanism to manage rate of data flow.

Conclusion: Understanding of full duplex communication reliable connection and transmission of data.

Reference:

James F Kurose and Keith Whoss, Computer Networks A top down approach, 8th edition.

Term Work - 05

Title : Distance Vector routing algorithm.

Objective : To determine the optimal path for data transmission by evaluating the distance (cost) between the routers, aiming to find the shortest or most efficient route.
Implement mechanism to prevent routing loops, ensuring that data packets do not circulate endlessely between routers and avoiding inefficient data transmission.

Theory :
Distance vector routing is a class of routing algorithms used in computer networks to determine the best path for data transmission between nodes. These algorithms calculate the distance (cost) to other nodes in the network based on vector information. after expressed in forms of routing table.
Nodes in the network maintain vectors representing the distance to each destination. These vectors are periodically exchanged among neighbouring nodes. Each node maintains a routing table containing information about the cost and next hop for reaching all destinations in the networks.
Node exchange routing information at regular intervals, sharing these routing table with adjecent nodes. This helps in keeping all nodes update about the networks topology

Algorithm:
At each node x,
initialization
for all destinations y in N :
$D_x(y) = c(x, y)$ // If y is not a neighbour.

for each neighbour w
Dw(y) = ? for all destination y in N
for each neighbour vector
Dx = [Dx(y) : y in N] to w
loop
wait (until receive any distance vector from some neighbour)
for each y in N :
Dx(y) = min v{ c(x,y) + Dv(y)}
if Dx(y) is changed for any destination y
send distance vector Dx = [Dx(y) : y in N] to all neighbour.

Program :

```
#DVR
import sys
class Network :
    def __init__ (self, nodes) :
        self.nodes = nodes
        self. graph = {}   # Dictionary to store network topology.
        self. distance_vector = {}  # Dictionary to store distance vectors

    def add_link (self, node1, node2, cost):
    # add a link between two nodes with a given cost.
        if node1 not in self.graph:
            self. graph [node1] = {}
        self. graph [node1][node2] = cost

        if node2 not in self. graph:
            self.graph [node2] = {}
        self. graph [node2] [node1] = cost
```

```
def initialize_distance_vector(self,node):
    #initialize the distance vector for a node
    self.distance_vector[node] = {node:0}
    for n in self.nodes:
        if n!=node:
            self.distance_vector[node][n] = sys.maxsize

def update_distance_vector(self,node):
    #update the distance vector for a node
    for dest in self.nodes:
        if dest!=node:
            min_cost = sys.maxsize
            for neighbour in self.graph[node]:
                if dest in self.distance_vector[neighbour]:
                    cost = self.distance_vector[neighbour][dest]
                        + self.graph[node][neighbour]
                    if cost < min_cost:
                        min_cost = cost
            self.distance_vector[node][dest] = min_cost

def print_routing_table(self,node):
    #print the routing table for a node
    print(f"Routing table for node {node}:")
    print("Destination\t cost")
    for dest, cost in self.distance_vector[node].items():
        if dest!=node:
            print(f"{dest}\t\t{cost}")
    print()
```

```
if __name__ == "__main__":
    nodes = [1,2,3,4,5]
    network = Network(nodes)
    network.add_link(1,2,2)
    network.add_link(1,3,2)
    network.add_link(1,4,1)
    network.add_link(2,3,1)
    network.add_link(2,5,1)
    network.add_link(3,4,1)
    network.add_link(3,5,1)

    for node in nodes:
        network.initialize_distance_vector(node)

    num_iterations = 6 # Number of iterations to update distance vectors
    for _ in range(num_iterations):
        for node in nodes:
            network.update_distance_vector(node)

    for node in nodes:
        Network.print_routing_table(node)
```

Outcomes:

- Learnt determining the optimal path for data transmission by evaluating the distance between routers using distance vector routing algorithm.
- Implementing mechanisms to prevent routing loops

Conclusion :

Applying distance vector routing algorithm to find the optimal path for data transmission.

References :

James F Kurose and Keith Whoss, Computer Network, A top down approach, 8th edition.

Term Work 06

Title : Congestion control algorithm.

Objective : Its primary objective is to manage queue to packets with varying sizes. deducting packet sizes from a count with its insufficient and limiting the packets moved out of the queue.

Theory : Congestion state occures in network layer when the message traffic is so heavy that is slows down network response time. It causes delay and performance decreases. To solve this problem congestion control algorithms such as leaky bucket is used. It helps in smoothing out bursty traffic and preventing network congesting. The leaky bucket can hold a limited amount of water. Incoming data is like water being found into the bucket. If the bucket overflows, then data is discarded or marked for slower transmission. The leak rate controls the rate at which packets are removed from the packet, maintaining steady output. It prevents network congestion by smoothing out burst of traffic. Allows for a controlled and predictable data flow. Its often implemented in network devices and routers to regulate the rate of incoming devices.

Algorithm :

1. Initialize a counter to n at the tick of the clock.
   Set bucket size (maximum capacity)
   Set leak rate (rate at which tokens or data can be removed from bucket)
   Add incoming data (packet) to the bucket.

If the bucket size is exceeded, discard the excess data or mark it for slower transmission.

for each incoming packet
    add packet to bucket
    if bucket size exceds
        set bucket size back to initial size.

Program:

```
Count = 1000
packets = [200, 500, 600, 700, 450, 400, 200]
length = len(packet)
index = length - 1
while (index < length and index > 0):
    while (count > packets[index]):
        print ("packet moved out of queue is", packets[index])
        count = count - packets[index]
        index = index - 1
    print ("Count is less than the packet value")
    count = 1000
print ('All packets are moved out of queue sucessfully')
```

Outcomes: The leaky bucket algorithm is controlled and regulated data flow, preventing network congestion by smoothing out bursty traffic.

Conclusion: The leaky bucket algorithm provides controlled data

flow, preventing congestion and ensuring predictability in network.

References:

James F Kurose and Keith Ross, Computer Network: A top down approach, 8th edition.