IIT BHUBANESWAR

SYSTEM FORENSIC LAB

ASSIGNMENT-1

Team Member

NAME - CHANDAN KESHARI
ROLL NUMBER - 24CS06022
BRANCH - CSE(MTECH)

NAME - NAVEEN KUMAR
ROLL NUMBER - 24CS06011
BRANCH - CSE(MTECH)

Questions

1. Establish a connection between the client computers and develop a chat interface between both clients. Use socket programming for the same.
2. Use a Diffie hellman key exchange algorithm to communicate the private key between the client and the server.
3. Use an RC4 encryption methodology and try to encrypt the messages from the sender's end and decrypt them on the receiver’s end. You can assume that the private key is known to both the receiver and sender before establishing the communication. (Through the Diffie Hellman algorithm).
4. Implement the man-in-the-middle attack. In this case, you can assume that the attacker is able to receive the stream of messages. Assume that the range of private keys is very limited so that you can use brute force attack. Use this information to decrypt the messages.

Questions - 1

Objective: Create a real-time chat application using socket programming that allows two clients to communicate through a server.

Steps:

> Server Setup:

- Import the necessary libraries(e.g. `*WinSock2.h*`), Winsock2 library is a Windows-specific API that provides a standardized interface for network programming. It is an enhancement of the original Winsock API and offers improved functionality and support for modern networking protocols.
- First, we have to setup the ***WSAStartup*** function to enable several function calls used in socket programming.
- Create a socket object using ***socket*** function call, which returns a file descriptor.
- Setup the ***setsockopt,*** which is a crucial part of socket programming in the Winsock2 library, allowing developers to configure various options for a socket.
- Now, we setup the flow of code, that is whether our code will be in blocking mode or non-blocking mode. Using ***ioctlsocket*** which is used to control socket behavior and manage various socket options.
- Bind the server to a specific IP address and port using ***bind()***.
- Listen for incoming connections using ***listen()***.
- Accept connections from clients using ***accept()***.
- Create a thread for each client to handle messages concurrently.

> Client Setup:

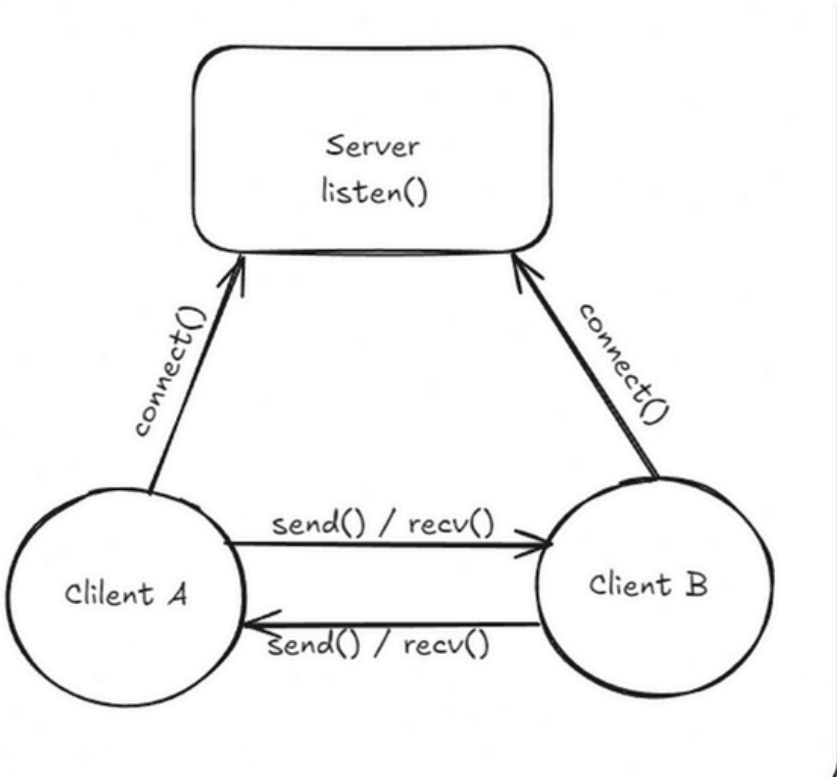
- Import the necessary libraries.
- Create a client socket, after setting up the ***WSAStartup*** API.
- Create a socket object and connect to the server using ***connect()***.
- Create a loop to ***send*** and ***receive*** messages, using functions as ***send()*** & ***recv()***.

> Unique Functionality:

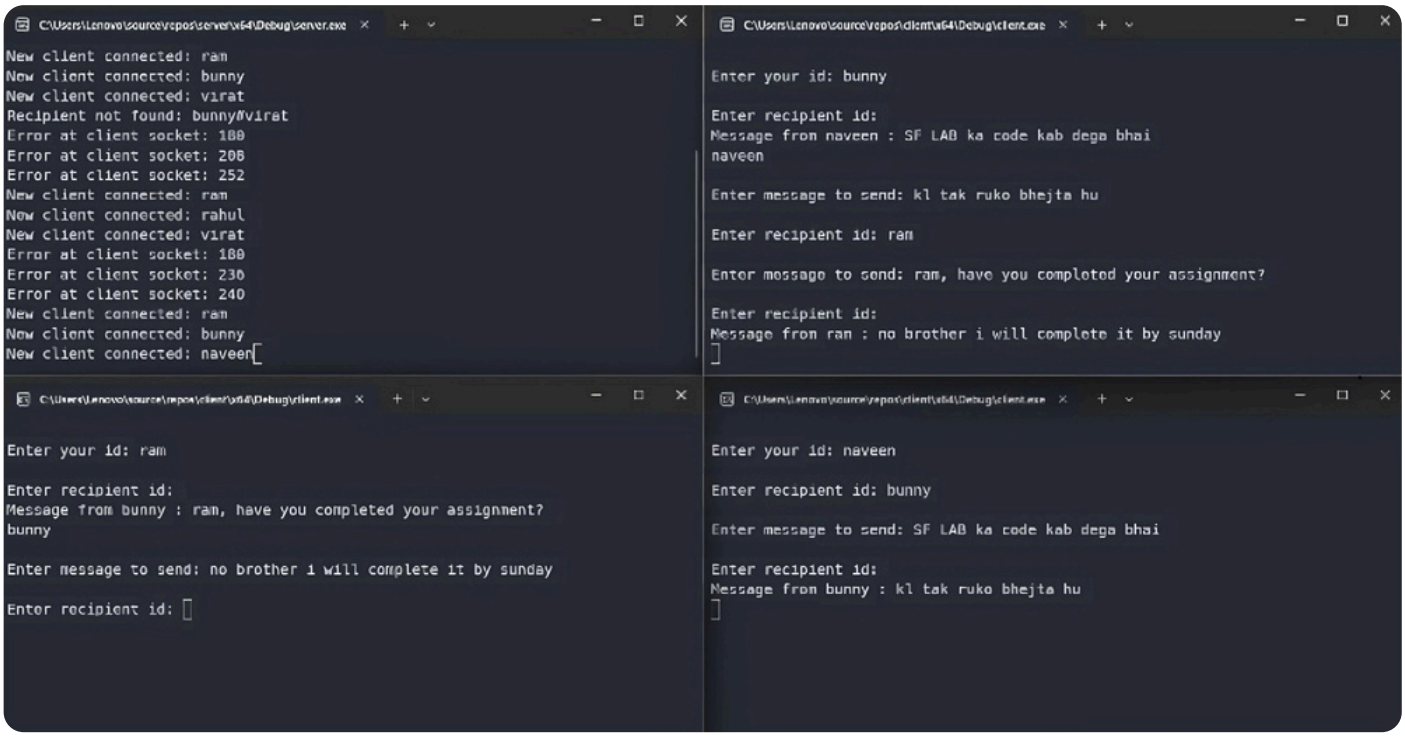
- The implementation of a unicasted client-client connection ensures that messages from one client are not broadcasted to every other client connected to the server on that port.

Questions - 1

Architecture:



Result:



Questions - 2

Objective: Securely exchange a private key between the client and server using the Diffie-Hellman algorithm.

Steps:

> Key Generation:

- Choose a large prime number ***p*** and a primitive root ***g***.
- Each party (client and server) generates a private key (randomly chosen integer) and computes their public key using ***g^{private key} mod p***.

> Key Exchange:

- Exchange public keys between the client and server.
- Each party computes the ***shared secret key*** using the received public key and their own private key:
  - ***Client computes:*** shared key = server ***s public key^{client s private key} mod p***.
  - ***Server computes:*** shared key = client ***s public key^{server s private key} mod p***.

> Verification:

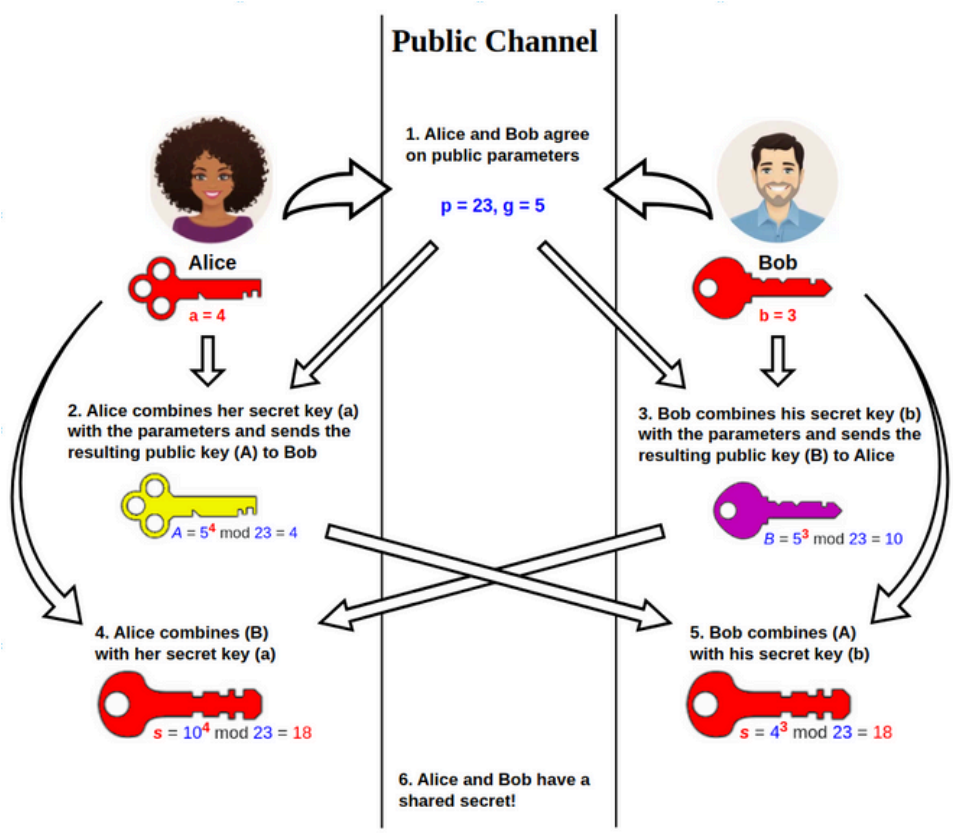
- Ensure both parties arrive at the same shared secret key.

> Unique Functionality:

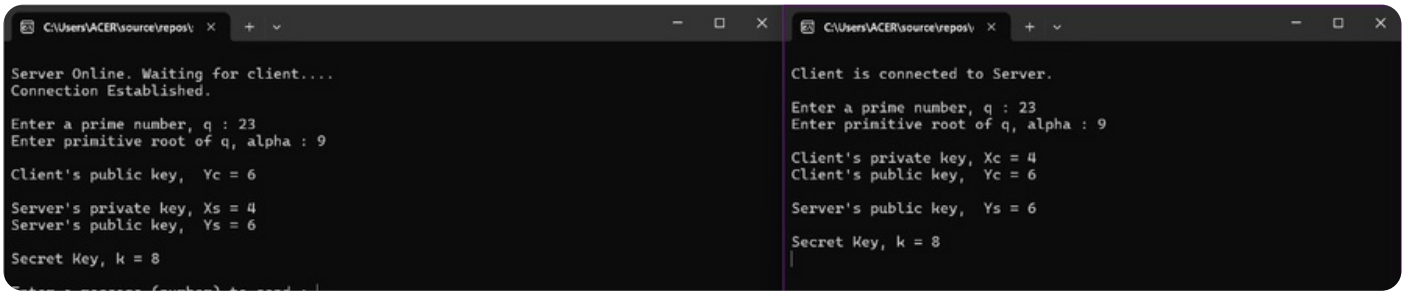
- The implementation of a Diffie-Hellman key exchange was demonstrated using C++ socket programming, facilitating communication via the terminal.

Questions - 2

Architecture:



Result:



Questions - 3

Objective: Encrypt messages at the sender's end and decrypt them at the receiver's end using the RC4 algorithm.

Overview of RC4:

- ***Stream Cipher:*** RC4 is a stream cipher, meaning it encrypts data one byte at a time, in contrast to block ciphers, which encrypt blocks of data at a time.
- ***Key Size:*** The key used in RC4 can vary in length, typically between 40 and 2048 bits.
- ***Simplicity:*** The algorithm is simple and efficient in software, making it suitable for fast encryption.

Steps:

> Key Scheduling Algorithm (KSA):

The KSA initializes a permutation of all 256 possible byte values (denoted as S), using the secret key provided.

Steps:

- Initialize an array S of size 256, such that S[i] = i for i in range 0 to 255.
- Initialize an index j = 0.
- Iterate over the array S, and for each element S[i], do the following:
  - Compute j = (j + S[i] + key[i mod key_length]) % 256.
  - Swap S[i] and S[j].
- After the KSA, S contains a permutation of all 256 possible byte values, and this array will be used to generate the keystream.

> Pseudo-Random Generation Algorithm (PRGA):

The PRGA generates a keystream that will be XORed with the plaintext to produce the ciphertext.

Steps:

- Initialize indices i = 0 and j = 0.
- For each byte of plaintext:
  - Increment i = (i + 1) % 256.
  - Update j = (j + S[i]) % 256.
  - Swap S[i] and S[j].
  - Compute the keystream byte K = S[(S[i] + S[j]) % 256].
  - XOR the keystream byte K with the plaintext byte to produce the ciphertext byte.

- The same PRGA is used during decryption, where the ciphertext is XORed with the same keystream to retrieve the original plaintext.

> Message Enryption:

- Convert the plaintext message into bytes.
- XOR the plaintext with the keystream to produce the ciphertext.

> Message Decryption:

- At the receiving end, use the same keystream to decrypt the message by XORing the ciphertext with the keystream.

Communication Process:

> Encryption at Sender's End:

- The sender initializes the RC4 cipher with the shared key K.
- The sender encrypts the message by XORing the plaintext with the generated keystream.
- The sender transmits the ciphertext to the receiver

> Decryption at Receiver's End:

- The receiver, using the same shared key K, initializes the RC4 cipher.
- The receiver generates the same keystream as the sender.
- The receiver decrypts the message by XORing the ciphertext with the keystream to retrieve the original plaintext.

Summary of the Process:

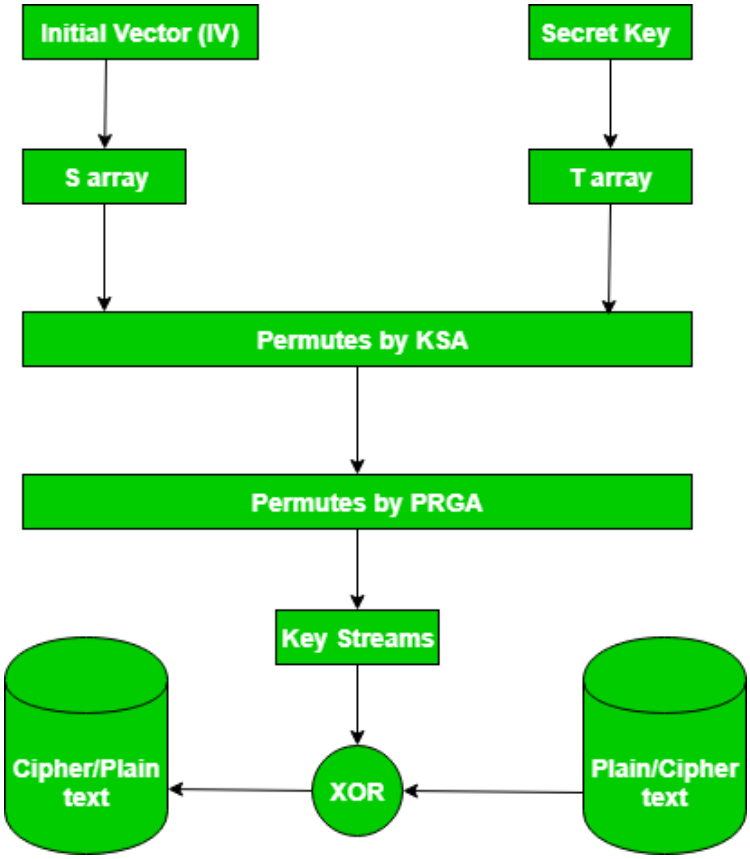
- ***Initialization (KSA):*** Sets up the permutation array S based on the key.
- ***Keystream Generation (PRGA):*** Generates a keystream, one byte at a time.
- ***Encryption/Decryption:*** XOR the plaintext with the keystream to produce ciphertext (or ciphertext with the keystream to retrieve plaintext).

Security Note:

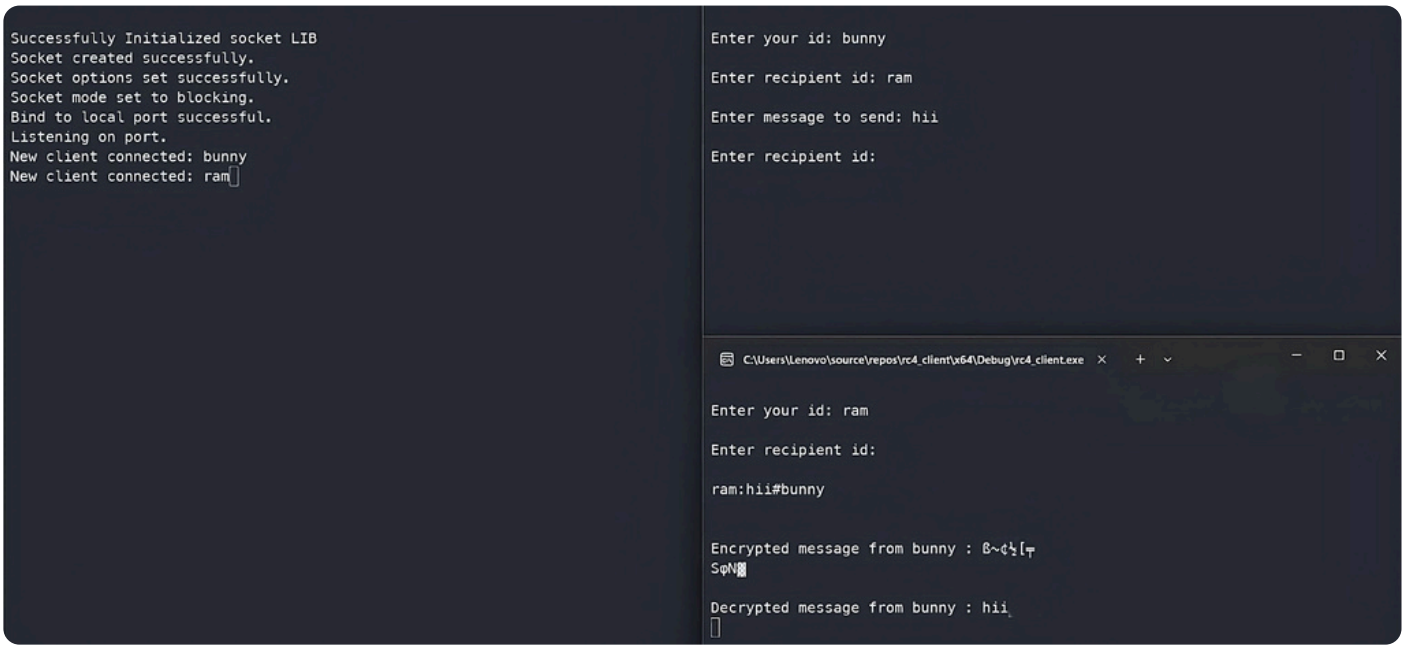
While RC4 was once widely used, it has known vulnerabilities and is considered insecure for many applications today, particularly due to biases in the keystream. It is recommended to use more secure alternatives like AES for new implementations.

Questions - 3

Architecture:



Result:



Questions - 4

Objective: Demonstrate a man-in-the-middle attack where an attacker intercepts messages between the client and server.

A Man-in-the-Middle (MitM) attack is a form of cyberattack where an attacker secretly intercepts and relays messages between two parties who believe they are communicating directly with each other. The attacker can manipulate or eavesdrop on the communication without either party being aware of the intrusion.

Steps:

> Interception:

The attacker positions themselves between the client and server. This can be achieved through various methods, such as ARP spoofing, DNS spoofing, or by compromising a network router.

> Key Exchange Interception:

During the Diffie-Hellman key exchange or any other key establishment protocol, the attacker intercepts the public keys exchanged between the client and server. The attacker can then generate their own keys to establish separate connections with both parties.

> Brute-Force Attack:

Given the limited range of private keys, the attacker can attempt to guess the private keys used by the client and server. This involves:

- Generating potential private keys within the known range.
- Calculating the corresponding public keys using the agreed-upon parameters (e.g., prime number and primitive root).
- Intercepting messages encrypted with the shared secret key and attempting to decrypt them using the guessed keys.

> Message Decryption:

Once the attacker successfully guesses the private key, they can compute the shared secret key and decrypt the messages sent between the client and server. The attacker can then read, modify, or inject messages into the communication stream.

> Relay Messages:

The attacker can relay the decrypted messages to the intended recipient, making it appear as though the communication is secure and uninterrupted.

Example of a Brute-Force Decryption Process:

Assuming the attacker has intercepted an encrypted message and knows the encryption method (e.g., RC4), the following steps outline how they might proceed with a brute-force attack:

1. Known Parameters:

- The attacker knows the prime number pp and the primitive root gg used in the Diffie-Hellman exchange.

2. Public Key Interception:

- The attacker captures the public keys A and B exchanged between the client and server.

3. Private Key Guessing:

The attacker generates possible private keys a*a* and b*b* within a limited range. For each guessed private key:

- Compute the corresponding public key:
  - ***A′ = g^a mod p***
  - ***B′ = g^b mod p***

4. Shared Key Calculation:

Using the guessed private keys, the attacker computes the shared secret key:

- ***k = B^a mod p*** (using the intercepted public key from the server)
- ***k′ = A^b mod p*** (using the intercepted public key from the client)

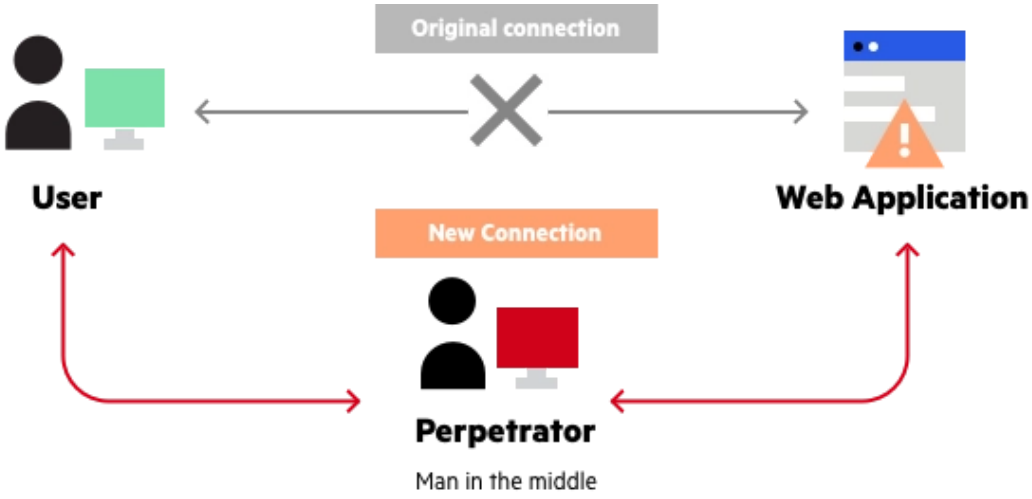
5. Message Decryption:

With the shared key kk, the attacker decrypts the intercepted messages:

- If the messages were encrypted using a symmetric encryption algorithm like RC4, the attacker can now decrypt the messages using the derived key.

Questions - 4

Architecture:



Result:

```
Intercepted public keys:
Public Alice: 6
Public Bob: 16

Attacker found secret keys:
Secret key of Alice: 4
Secret key of Bob: 3

Shared secret keys computed by attacker:
Shared secret (via Alice): 9
Shared secret (via Bob): 9

MITM Attack Successful. Shared secret: 9
```

References

1. ***WSAStartup:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-wsastartup*
2. ***Socket****: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket*
3. ***SockAddr:*** *https://learn.microsoft.com/en-us/windows/win32/api/ws2def/ns-ws2def-sockaddr_in*
4. ***setsockopt:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-setsockopt*
5. ***ioctlsocket:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-ioctlsocket*
6. ***bind:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-bind*
7. ***listen:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-listen*
8. ***select:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-select*
9. ***FD_SET:*** *https://learn.microsoft.com/en-us/windows/win32/api/winsock2/ns-winsock2-fd_set*
10. ***Diffe-Hellmen:*** *https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/*
11. ***RC4 Algorithm:*** *https://www.geeksforgeeks.org/implementation-of-rc4-algorithm/*
12. ***Man-in-the-middle Attack:*** *https://www.geeksforgeeks.org/man-in-the-middle-attack-in-diffie-hellman-key-exchange/*