EX.NO.:	TRANSMISSION CONTROL PROTOCOL
PAGE NO.:	USING ONE WAY COMMUNICATION

AIM:

To implement Transmission Control Protocol using one Way Communication

PROCEDURE:

SERVER SIDE:

1. Initialize the Socket: Create a socket for network Communication using AF_INET (IPv4) and SOCK_STREAM (TCP Protocol)
2. Bind to Address and Port: Bind the server to a address (i.e; localhost/ip address) on a port (e.g; 5001) This means the server will listen for connection on this ip and port.
3. Listen for connections: Setup the socket to listen for incoming connections
4. Accept Connection: Wait for a client to connect. Once a client connects, accept the connection and store the client socket object and address.
5. Receive Data: Read the data from the client and decode it (convert the bit to string)
6. Print Received Message: Display the received message from the client.
7. Send Response: Send a acknowledgment message from the client.
8. Close the server socket to end the connection.

Client-side:

1. Get message from user: Prompt the user to enter a message.
2. Initialize the Socket: Create a socket for network communication using AF_INET (IPv4) and SOCK_STREAM (TCP protocol).
3. Connect to Server: Establish a connection to the running server using server ip and port.
4. Send Message: Send the users message to the server, encoding it to byte.
5. Receive Response: Receive the data from the surver, decode it and store it as a response.
6. Print Response: Display the response received from the server.
7. Close Connection: Close the socket to end the connection.

EX. NO.: PAGE NO.:	TRANSMISSION CONTROL PROTOCOL USING TWO WAY COMMUNICATION

AIM:

TO implement Transmission Control Protocol Using Two Way Commounication.

PROCEDURE:

SEND FUNCTION:

1. Loop to continuously Send Messages.
2. GET a message from user to send.
3. Create a socket for network commounication using AF_INET (IPv4) and SOCK_STREAM (TCP Protocol).
4. Connect this socket to the running surver using server ip address and port.
5. Encode the users message to byte and send to the surver.
6. Recive the response data from the server. decode it and store it as an response
7. Print the response data received from the server.
8. Close the socket to end the connection after each message.

RECEIVE FUNCTION:

1. Create a new socket for network commounication using AF_INET (IPv4) and SOCK_STREAM (TCP Protocol).
2. Bind the surver to a address (i.e; localhost/ ip address) on a port (e.g; 5001). This means the surver will listen for connection on this ip and port.
3. Set up the socket to listen for incoming connections.
4. Loop to continuously receive message.

5. Wait for a client connection, then accept it, once a client connect, and store the client socket object and address
6. Receive the data from client and decode it to convert from byte to string.
7. Print the receive message from the client
8. Send an acknowledgment message for the client.
9. Close the server socket to end the connection.

RUN BOTH FUNCTION CONCURRENTLY

1. Use threading to run the function separate thread so it can operate independently and allow the program to send and receive message at the same time.
2. Start ~~now~~ SEND function in a new thread.
3. Start RECEIVE function in another separate thread.

EX.NO.:	MULTICAST
PAGE NO.:	

AIM:

To implement Multicast using UDP (User Datagram Protocol)

PROCEDURE: (SENDER)

1. Import libraries 'socket' for network communication and 'struct' to handle binary data.
2. Create a socket for network communication using AF_INET (IPv4), SOCK_DGRM (UDP Protocol) and also IPPROTO_UDP
3. Set Multicast Time-to-Live: IPPROTO_IP (IP level option), IP_MULTICAST_TTL (To set the TTL) and 2 is the TTL value which allow the packet to travel up-to one router away from the local router before it expires.
4. Enter a loop allowing a user to repeatedly send message.
5. Select group to send the message.
6. Take the message from user.
7. Send the message to the selected group multicast address and port.

RECEIVER:

1. Import libraries 'socket' for network communication and 'struct' to handle binary data.
2. Create a socket for network communication using AF_INET (IPv4), SOCK_DGRM and IPPROTO_UDP (For UDP Protocol)

3. Reuse address allow the socket to bind to an address even if it is already in use. This help when multiple socket need to share the same port. SOL-SOCKET (Option is at socket level), SO-REUSEADDR (Reuse the address option) 1 to enable the option.
4. Bind the socket to '0.0.0.0' with a 'port' to listen all the available network interface on this port.
5. Setting up multicast group to join with an IP (a reserved IP for multicast range of 224.0.0.0 to 239.255.255.255)
   Sock_INADDR_ANY (listen for multicast packets on any network interface available on the host.
6. Packed information passed to IP_ADD_MEMBERSHIP which join the multicast group.
7. Enter a loop where it wait to receive data from the multicast group
8. Received data with sender information.
9. Print message.

EX.NO.: PAGE NO.:	CAESAR CIPHAR

AIM:

To implement Caesar Cipher chyptographic algorithm.

PROCEDURE:

Encryption:

1. Take plaintext and shift value
2. Initialize an empty string to store the result.
3. For each character in plaintext.
4. Check if character is an alphabet.
5. Determine if character is lowercase or uppercase and set the 'base' ASCII code to 'a' for lowercase or 'A' for uppercase.
6. Shift character by the specified amount within the alphabet's range using (Order of character - base + shift) mod 26 + base)
7. Convert the shifted value back to the character and update.
8. Append character to ciphertext.
9. Return the ciphertext.

Decryption:

1. Take the ciphertext and shift.
2. Use the encryption function with shift of (26 - shift), effectively reversing the shift for decryption.
3. Return the decrypted text

EX.NO.: PAGE NO.:	PLAYFAIR CIPHER

AIM:

To implement Playfair Cipher

PROCEDURE:

1. Generate Key Matrix:
   Replace J with I in the key, then remove duplicate letters
   Add remaining letter of the alphabet except J to complete 5x5 matrix.
2. Find character position
   Loop through the key matrix to locate the row and column of a given character
3. Prepare text:
   Remove non-letter character, replace J with I
   Make a pair in text, if a pair has same letter insert 'X' in between them. If the text length is odd, add an 'X' at the end to make it even
4. Encrypt text:
   For each letter pair
   - Same row: replace each letter with next letter in that row.
   - Same column: replace each letter with the next letter in that column.
   - Rectangle Rule: Swap columns between the letters in the pair
5. Decrypt text:
   For each letter pair
   - Same row: replace with previous letter in row
   - Same column: replace with previous letter in column.
   - Rectangle: Swap column between letter and pair

EX. NO.:	HILL CIPHER
PAGE NO.:	

AIM:

To implement Hill Cipher algorithm

PROCEDURE:

1. Convart a character (A-Z) to its crosponding integer (0-25) using ASCII value.
2. Convart an integer (0-25) back to its crospon-ding uppercase character.
3. Computes the modulus of 'a' with 26 to keep values within the 0-25 range.

Encryption:

1. Convarts each character of plaintext into its numerical equivalant to plaintextVector.
2. Multiplies the key matrix with plaintextV-ector to compute ciphertextVector.
3. Each enery in ciphertextVector is value of multiplication value of key and plaintext followed by applying modulas 26 to keep the value within the range.
4. Convarts each number in ciphertext vector back to letter, ~~and~~ store and return it in ciphertext.

Key Matrix Construction: A 3X3 matrix from 9 character string where each character is convarted to its numerical equivalent.

Matrix Determinant:

Find the determinant of the key matrix and applied mod 26 to it.

Mod Inverse:

Find the modular inverse of a under modulo m it check for the value x such that

$(a * x) \% m == 1$

Find co-factor matrix:

Compute the co-factor matrix of 3x3 key matrix which is steps towards calculating the inverse.

Find Transpose Matrix:

Transposes the cofactor matrix, which is needed to find the adjugate.

Find Inverse Matrix:

1. Calculate the determinant of the key matrix and its modular inverse.
2. Compute the cofactor and then transposes it to get the adjugate.
3. Multiplies the adjugate by the modular inverse of the determinant to obtain the inverse key matrix, applying modulas 26 to each element.

Decryption:

Calls the Encryption function using the ciphertext and the inverse key to retrieve the original plaintext.

EX.NO.: PAGE NO:	VIGENERE CIPHER

AIM:

Implement Vigenere Cipher algorithm

PROCEDURE:

1. Take text and key as input
2. Remove space from both text and key and convart them to uppercase for consistent Processing.
3. Create two empty list 'encrypted' to store the encrypted character, and 'decrypted' to store the decrypted character.
4. For each character in text
   1. Convart the 'text' character to its Position in the alphabet.
   2. Convart the ~~xxxx~~ corresponding key Character to its position in alphabet and repeat the key if it's length is less than 'text'.
5. Encryption:
   1. Calculate the encrypted character by adding the number of character and number of key and use modulo 26 if necessary.
   2. Convert the encrypted value back to character and add it to encrypted list.
6. Decryption:
   1. Calculate the decrypted characted by reversing the encryption process by subtracting key number from encryption character.
   2. Convart the decrypted value back to the character and add it to the decrypted list.

EX. NO.: PAGE NO.:	DES (DATA ENCRYPTION STANDARD)

AIM:

To implement DES (Data Encryption Standard) Algorithm.

PROCEDURE:

Generate a key:

Generate a random 8-byte key using get_random_bytes for DES encryption.

DES Encryption:

1. Take plaintext and key as a input
2. Create a DES cipher object in ECB (Electronic Codebook) mode using the provided key.
3. Pad the plaintext to ensure its length is multiple of the DES block size.
4. Encrypt the padded plaintext using the DES cipher to get the encrypted text.
5. Return the encrypted text.

DES Decryption:

1. Take the ciphertext and key as input.
2. Create a DES cipher object in ECB mode using the provided key
3. Decrypt the ciphertext to get the decrypted text.
4. Unpad the decrypted text to remove any padding added during encryption.
5. Return the unpadded decrypted text.

EX. NO.:	RSA (RIVEST SHAMIR ADLEMAN)
PAGE. NO.:	

AIM:

To implement RSA (Rivest Shamir Adleman) cryptography algorithm

PROCEDURE:

GENERATE A LARGE PRIME NUMBER:

1. Generate a random number between a given range.
2. Check the number is prime
3. If it is prime, return it, otherwise repeat until a prime is found.

RSA Key Generation:

1. Select two distinc large prime number p and q (i.e; $p \neq q$).
2. Compute $n = p \times q$
3. Compute $phi(n) = (p-1) \times (q-1)$
4. Select an integer 'e' such that $1 < e < phi(n)$ and $gcd(e, phi(n)) = 1$
5. If e is not prime then find prime between 2 and $\phi(phi(n)-1)$
6. Compute d, the modulo inverse of e modulo phi(n)
7. Return public key = (e, n), Private key = (d, n)

Encryption:

1. Take plaintext and public key (e, n)
2. Convert the character to its ASCII integer value.
3. Encrypt it using the formulla
   Cipher = $char^e \bmod n$
4. Return encripted integer value.

Decryption:

1. Take ciphertext (encrypted integer) and a private key (d, n)
2. Decrypt it using the formula
   $char = cipher^{d} \bmod n$
3. Convert the resulting integer back to its character representation
4. Return the decrypted text as the original message

EX. NO.: PAGE NO.:	DIFFIEE-HELLMAN KEY EXCHANGE

AIM:

To implement Diffiee Hellman Key Exchange Algorithm

PROCEDURE:

Generate a Private Key

1. Accept a prime number as input.
2. Generate a random integer between 2 and (Prime - 2) as the private key.
3. Return the private key.

Generate a Public key

1. Accept a private key, prime, and generator as input.
2. Compute the public key using the formulla $generator^{Private key} \bmod Prime$ using Python pow function function for modular exponentiation.
3. Return the Public key.

Compute Shared Secret

1. Accept a public key from the other party, the Private key of the user and prime as input.
2. Compute the shared secret as the formulla

$$PublicKey^{Private key} \bmod Prime$$

using pow function

3. Retern the compute secret.

Key Exchange

1. Define a prime number and a generator
2. Generate private key and public key of Alice
3. Generate Private key and Public key of Bob
4. Alice compute her shared secret.
5. Bob compute his shared secret.
6. Both Alice and Bob should have the same shared secret.

EX. NO.:	MESSAGE-DIGEST ALGORITHM-5 (MD5)
PAGE NO.:	

AIM:

To implement Message Digest Algorithm 5 (MD5) in Python.

PROCEDURE:

1. Get a user input text and store it in a variable
2. Create a new MD5 hash object using hashlib.md5()
3. Convert input text to bytes using UTF-8 encoding so that MD5 can read and add it to hasher, then pass it to the MD5 hash object with md5.update().
4. Compute the MD5 hash of the text and store it as a hexadecimal string using md5.hexdigest().
5. Print the resulting MD5 hash in hexadecimal format.

EX. NO.:	SECURE HASH ALGORITHM-1 (SHA-1)
PAGE NO.:	

AIM:

To implement Secure Hash Algorithm-1 hashing algorithm in Python.

PROCEDURE:

1. Get user input text and store it in a variable
2. Create a new SHA-1 hash object using hashlib.sha1()
3. Convert input text to bytes using UTF-8 encoding and pass it to the SHA-1 hash object with sha1.update().
4. Generate Hex Digest: Compute the SHA-1 hash of the text and store it as a hexadecimal string using sha1.hexdigest().
5. Print the resulting SHA-1 hash in hexadecimal format

EX. NO.:	IMPLEMENT THE SIGNATURE SCHEME -
PAGE NO.:	DIGITAL SIGNATURE STANDARD

AIM:

To implement the Signature Scheme - Digital Signature Standard using Python.

PROCEDURE:

Generate Keys:

1. Use the DSA (Digital Signature Standard) to Create a pair of keys
2. Generate a private key the key size is 2048 bits
3. Generate a public key from the previously generated private key.
4. Return the both Private key and PublicKey

Sign a Message:

1. Take the private key and message
2. Convert the message into hash using SHA256 to ensure it is securely represented.
3. Use private key to create a digital signature for the message hash, and use SHA-256 a algorithm to harshing.
4. Return the digital signature.

Verify The Signature

1. Take the public key, original message and signature
2. Hash the original message using SHA-256
3. Use public key, message hash and SHA-256 algorithm to verify the signature agenest the hashed message.

4. If the verification is successful (i.e; the signature is valid) then return 'True'.
5. If the verification fail (i.e; the signature is invalid) then return 'False'.

EXERCISE: VIGENÈRE CIPHER

AIM: To implement Vigenére Cipher algorithm

PROCEDURE:

1. The user enters a plaintext string and a key string.
2. Both the string and the key are convarted to UpperCase, and space are removed.
3. Each character in the string and key is convarted to a number representing its position in the alphatet (A=0, B=1 ... Z=25)
4. For Encription: each character in the string, the crosponding key character is added to it, if the value excid more then 26 then mod it with 26 to form the cipher text.
5. For Decryption: The key value are subtracted from the ciphertext value (mod 26) to recover the original plaintext.
6. The encrypted text and the decrypted text are printed.

Verified

RESULT:

The program for the vigenere cipher has been implemented successfully and its output has been verified.

EXERCISE: HILL CIPHER

AIM: To implement Hill Cipher algorithm

PROCEDURE:

1. Convert characters ('A' to 'Z') to numbers (0 to 25)
2. Encryption: Multiply 3×1 plaintext matrix by a 3×3 key matrix, apply modulo 26, and convert back to character.
3. Decrypt: Similar to encryption, but use the inverse of the key matrix.
4. Convert 9 character key string into 3×3 integer matrix.
5. Find the determinant of key matrix.
6. Calculate the inverse of the key matrix if the determinant is non-zero.
7. Find the cofactor matrix and transpose it to compute the inverse matrix.

RESULT: The program of the hill cipher has been implemented successfully and the output has been verified.

EXPRIEMENT No: DES (Data Encryption Standard)

AIM: Encrypt and decrypt data using a symmetric key block cipher. DES.

PROCEDURE:

1. Import the necessary modules from crypto library for DES encryption.
2. Create a random 8 byte key using get-random-byte(8).
3. Create a DES cypher in ECB (Electronics Codebook)
   Pad the plaintext to ensure proper block size.
   Encrypt the paded text and return the cipher text.
4. For decryption
   Create a DES cypher in ECB mode
   Decrypt the cipher text
   Unpad the decrypted text and return the original plain text.

RESULT: The program of the encrypt and decrypt data using a symmetric key block cipher algorithm DES has been implemented successfully and the output has been verified.