VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

BCSE309P – CRYPTOGRAPHY AND NETWORK SECURITY LABORATORY

School of Computer Science and Engineering
B.Tech. Computer Science and Engineering

LAB RECORD

Name : Aastha Kumar	Reg No : 21BCE5067
Date of submission : 25/04/24	Course Code : BCSE309P
Faculty : Prof. N G Bhuvaneshwari	Slot : L5 + L6

TABLE OF CONTENTS

.

Experiment 1a 05/01/24

TITLE:

Write a socket program to demonstrate Caesar Cipher.

ALGORITHM:

1. Setting Up:
   - Save the server code as `CaesarCipherServer.java` and the client code as `CaesarCipherClient.java`.
   - Compile both files:

     Bash
     ```
     javac CaesarCipherServer.java CaesarCipherClient.java
     ```
2. Running the Server:
   - Open a terminal window and navigate to the directory where you saved the compiled files.
   - Run the server:

     Bash
     ```
     java CaesarCipherServer
     ```
   - The server will print a message indicating it's listening on a specific port (default 4444).
3. Running the Client:
   - Open another terminal window (separate from the server window).
   - Run the client:

     Bash
     ```
     java CaesarCipherClient
     ```
   - The client will prompt you to enter the message to encrypt and then the shift value (between 1 and 25).
4. Encryption Process:
   - The client sends the shift value and message to the server.
   - The server prints the received shift value and message on its console.
   - The server encrypts the message using the received shift value.
   - The server sends the encrypted message back to the client.
   - The client receives and displays the encrypted message as confirmation.

5. Closing Connections:

- Both client and server will automatically close their connections after the encryption process is complete.

CODE:

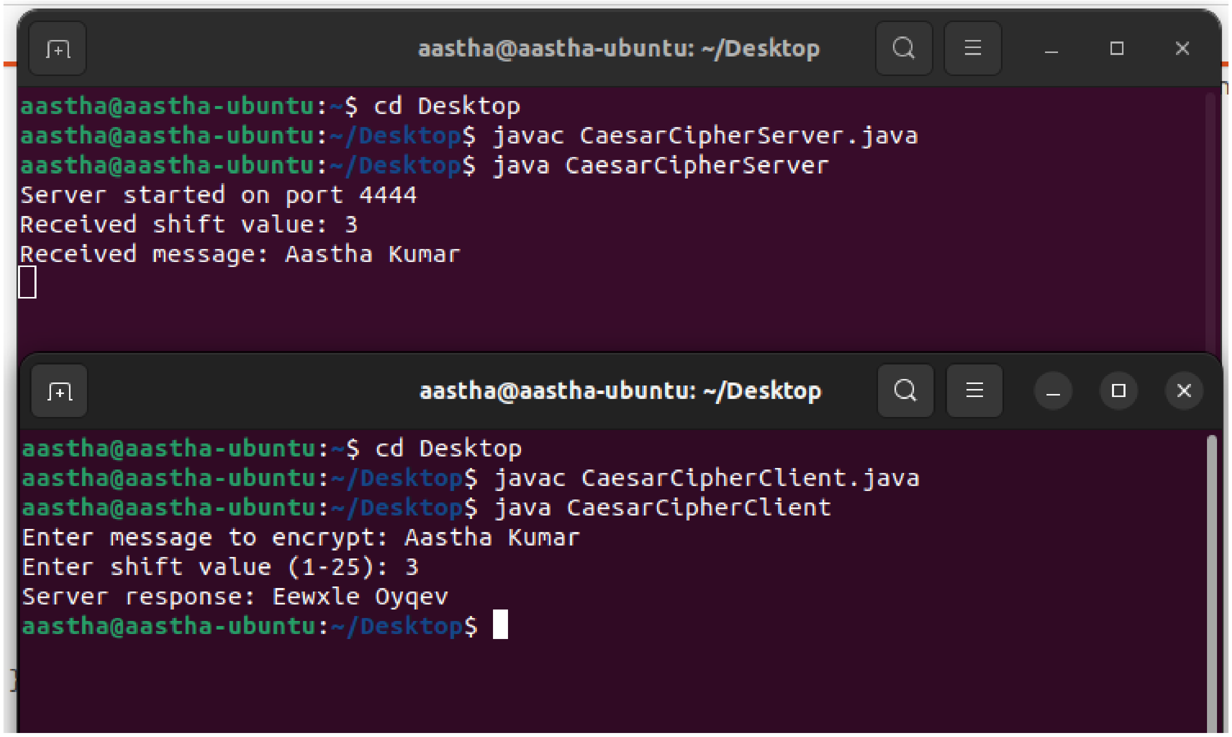
CaesarCipherClient.java

```
import java.io.*;
import java.net.*;

public class CaesarCipherClient {

  public static void main(String[] args) throws IOException {
    String host = "localhost";
    int port = 4444;
    String message;

    // Get message from user
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Enter message to encrypt: ");
    message = in.readLine();

    int shift; // Shift value (key)

    // Get shift value from user
    do {
      System.out.print("Enter shift value (1-25): ");
      shift = Integer.parseInt(in.readLine());
    } while (shift < 1 || shift > 25);

    // Connect to server
    Socket clientSocket = new Socket(host, port);

    // Get input and output streams
    BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    PrintWriter outToServer = new PrintWriter(clientSocket.getOutputStream(), true);

    // Send shift value
    outToServer.println(shift);

    // Send message to server
    outToServer.println(message);

    // Receive encrypted message from server
    String encryptedMessage = inFromServer.readLine();
    System.out.println("Server response: " + encryptedMessage);

    // Close connection
    clientSocket.close();
  }
}
```

CaesarCipherServer.java

```
import java.io.*;
import java.net.*;

public class CaesarCipherServer {

  public static void main(String[] args) throws IOException {
    int port = 4444; // Port to listen on

    ServerSocket serverSocket = new ServerSocket(port);
    System.out.println("Server started on port " + port);

    while (true) {
      // Wait for a client connection
      Socket clientSocket = serverSocket.accept();

      // Get input and output streams
      BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

      // Get shift value
      int shift = Integer.parseInt(in.readLine());
      System.out.println("Received shift value: " + shift); // Print received shift

      // Get message from client
      String message = in.readLine();
      System.out.println("Received message: " + message); // Print received message

      // Encrypt the message
      String encryptedMessage = encrypt(message, shift);

      // Send the encrypted message back to the client
      out.println(encryptedMessage);

      // Close connection
      clientSocket.close();
    }
  }

  public static String encrypt(String message, int shift) {
    StringBuilder encrypted = new StringBuilder();
    for (char c : message.toCharArray()) {
      if (Character.isAlphabetic(c)) {
        int newChar = c + shift;
        if (Character.isUpperCase(c)) {
          newChar = newChar % 'Z' + 1; // Wrap around for uppercase
          if (newChar < 'A') {
            newChar += 'A' - 1;
          }
        } else {
          newChar = newChar % 'z' + 1; // Wrap around for lowercase
          if (newChar < 'a') {
            newChar += 'a' - 1;
          }
        }
        encrypted.append((char) newChar);
      } else {
        encrypted.append(c); // Keep non-alphabetic characters
      }
    }
    return encrypted.toString();
  }
}
```

OUTPUT :



aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac CaesarCipherServer.java
aastha@aastha-ubuntu:~/Desktop$ java CaesarCipherServer
Server started on port 4444
Received shift value: 3
Received message: Aastha Kumar
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac CaesarCipherClient.java
aastha@aastha-ubuntu:~/Desktop$ java CaesarCipherClient
Enter message to encrypt: Aastha Kumar
Enter shift value (1-25): 3
Server response: Eewxle Oyqev
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 1b 05/01/24

TITLE:

Write a socket program to demonstrate Rail Fence Cipher.

ALGORITHM:

Server (MyServer.java):

1. Create a ServerSocket**:** Initialize a ServerSocket object on a specific port (e.g., 6666).
2. Wait for Connections: Use accept() method to wait for incoming client connections. This method blocks until a client connects.
3. Accept Client Connection: When a client connects, accept the connection and get a Socket object representing the client.
4. Read Data from Client: Create a DataInputStream from the client socket to read data sent by the client (e.g., the message to encrypt).
5. Encrypt the Message: Implement the Railfence cipher encryption algorithm (you can use the railfenceEncrypt method) to encrypt the received message.
6. Send Encrypted Message to Client: Create a DataOutputStream from the client socket and use writeUTF() to send the encrypted message back to the client.
7. Close Server Socket: Close the server socket using close().

Client (MyClient.java):

1. Create a Socket: Initialize a Socket object to connect to the server (e.g., “localhost” and port 6666).
2. Get Output Stream: Get the output stream from the socket using getOutputStream().
3. Send Message to Server: Create a DataOutputStream and use writeUTF() to send a message to the server (e.g., “Hello, server! This is the client.”).
4. Read Encrypted Message from Server: Create a DataInputStream from the socket to read the encrypted message sent by the server.
5. Close Client Socket: Close the client socket using close().

CODE:

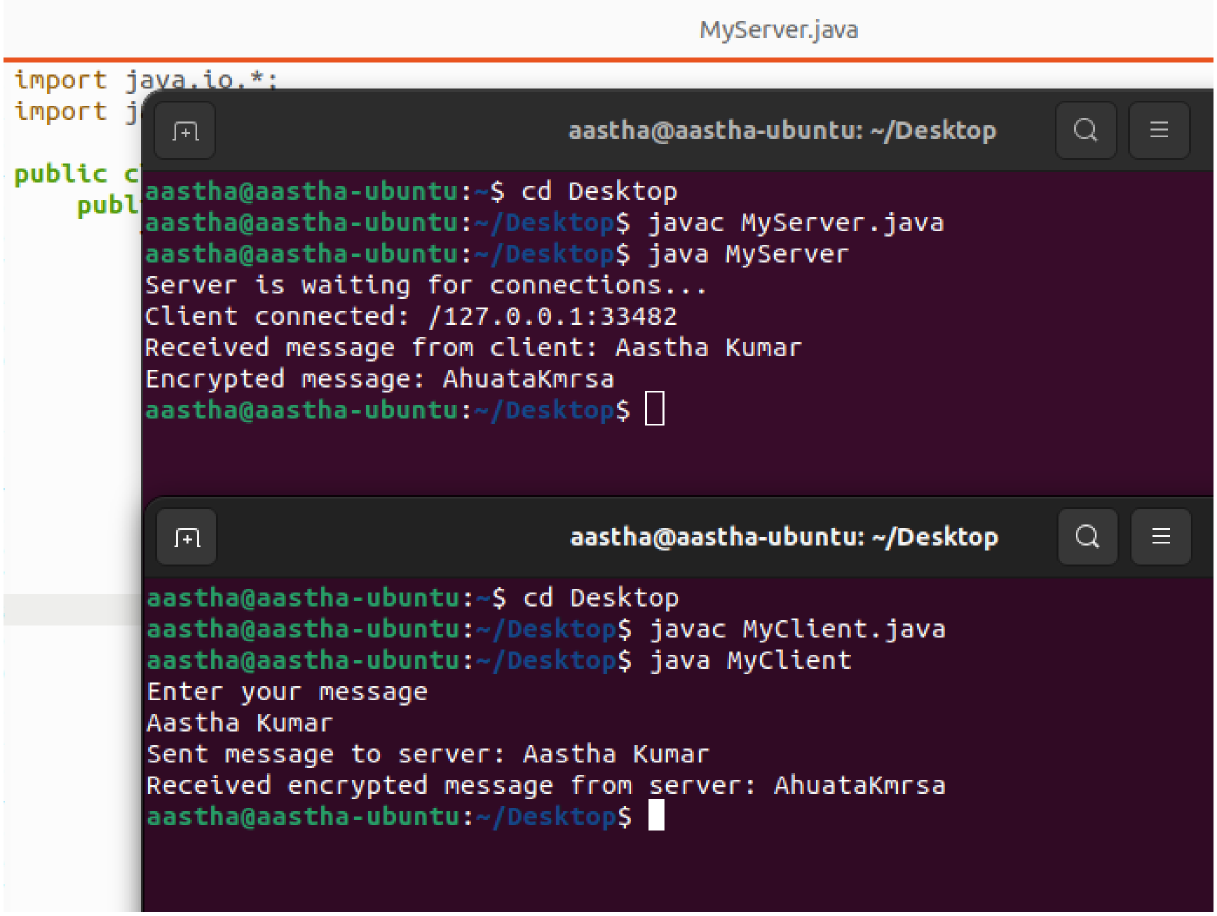
MyClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MyClient {
   public static void main(String[] args) {
   Scanner sc=new Scanner (System.in);
      try {
            // Connect to the server running on localhost at port 6666
            Socket s = new Socket("localhost", 6666);

            // Get the output stream to send data to the server
            OutputStream outToServer = s.getOutputStream();
            DataOutputStream dos = new DataOutputStream(outToServer);

            // Send a message to the server
            System.out.println("Enter your message");
            String message = sc.nextLine();
            dos.writeUTF(message);
            System.out.println("Sent message to server: " + message);

            // Read the encrypted message from the server
            DataInputStream dis = new DataInputStream(s.getInputStream());
            String encryptedMessage = dis.readUTF();
            System.out.println("Received encrypted message from server: " + encryptedMessage);

            // Close the client socket
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyServer.java

```
import java.io.*;
import java.net.*;

public class MyServer {
    public static void main(String[] args) {
        try {
            // Create a server socket on port 6666
            ServerSocket ss = new ServerSocket(6666);
            System.out.println("Server is waiting for connections...");

            // Accept incoming client connections
            Socket s = ss.accept();
            System.out.println("Client connected: " + s.getRemoteSocketAddress());

            // Read data from the client
            DataInputStream dis = new DataInputStream(s.getInputStream());
            String receivedMessage = dis.readUTF();
            System.out.println("Received message from client: " + receivedMessage);

            // Encrypt the received message using Railfence cipher
            int numRails = 3; // You can adjust the number of rails
            String encryptedMessage = railfenceEncrypt(receivedMessage, numRails);
            System.out.println("Encrypted message: " + encryptedMessage);
            // Send the encrypted message back to the client
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());
            dos.writeUTF(encryptedMessage);

            // Close the server socket
            ss.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String railfenceEncrypt(String message, int rails) {
        // Initialize the rail matrix
        char[][] railMatrix = new char[rails][message.length()];
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < message.length(); j++) {
                railMatrix[i][j] = ' ';
            }
        }

        // Fill the rail matrix with the message characters
        int row = 0, col = 0;
        boolean directionDown = false;
        for (char c : message.toCharArray()) {
            railMatrix[row][col] = c;
            if (row == 0 || row == rails - 1) {
                directionDown = !directionDown;
            }
            if (directionDown) {
                row++;
            } else {
                row--;
                col++;
            }
        }

        // Read the encrypted message from the rail matrix
        StringBuilder encryptedMessage = new StringBuilder();
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < message.length(); j++) {
                if (railMatrix[i][j] != ' ') {
                    encryptedMessage.append(railMatrix[i][j]);
                }
            }
        }

        return encryptedMessage.toString();
    }
}
```

OUTPUT:


MyServer.java
aastha@aastha-ubuntu: ~/Desktop

```
import java.io.*;
import j

public c
    publ
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyServer.java
aastha@aastha-ubuntu:~/Desktop$ java MyServer
Server is waiting for connections...
Client connected: /127.0.0.1:33482
Received message from client: Aastha Kumar
Encrypted message: AhuataKmrsa
aastha@aastha-ubuntu:~/Desktop$
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyClient.java
aastha@aastha-ubuntu:~/Desktop$ java MyClient
Enter your message
Aastha Kumar
Sent message to server: Aastha Kumar
Received encrypted message from server: AhuataKmrsa
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 1c 05/01/24

Title:

Write a socket program to demonstrate Playfair Cipher.

Algorithm:

Playfair Cipher Server (MyPlayfairServer.java):

1. **Create a ServerSocket**: Initialize a ServerSocket object on a specific port (e.g., 7777).
2. **Wait for Connections**: Use accept() method to wait for incoming client connections. This method blocks until a client connects.
3. **Accept Client Connection**: When a client connects, accept the connection and get a Socket object representing the client.
4. **Read Data from the Client**: Create a DataInputStream from the client socket to read data sent by the client (e.g., the plaintext message).
5. **Generate the Key Matrix**: Implement the generateKeyMatrix method to create a 5x5 key matrix based on the provided key.
6. **Prepare the Plaintext**: Implement the preparePlaintext method to remove spaces, convert to uppercase, replace ‘J’ with ‘I’, and add ‘X’ for odd length.
7. **Encrypt the Plaintext**: Implement the Playfair cipher rules to encrypt the prepared plaintext using the key matrix.
8. **Send the Encrypted Message to the Client**: Create a DataOutputStream from the client socket and use writeUTF() to send the encrypted message back to the client.
9. **Close the Server Socket**: Close the server socket using close().

Playfair Cipher Client (MyPlayfairClient.java):

1. **Create a Socket**: Initialize a Socket object to connect to the server (e.g., “localhost” and port 7777).
2. **Get Output Stream**: Get the output stream from the socket using getOutputStream().
3. **Send a Plaintext Message to the Server**: Create a DataOutputStream and use writeUTF() to send a plaintext message to the server.
4. **Read the Encrypted Message from the Server**: Create a DataInputStream from the socket to read the encrypted message sent by the server.
5. **Close the Client Socket**: Close the client socket using close().

CODE:

MyPlayfairClient.java

```
import java.io.*;
import java.net.*;
```

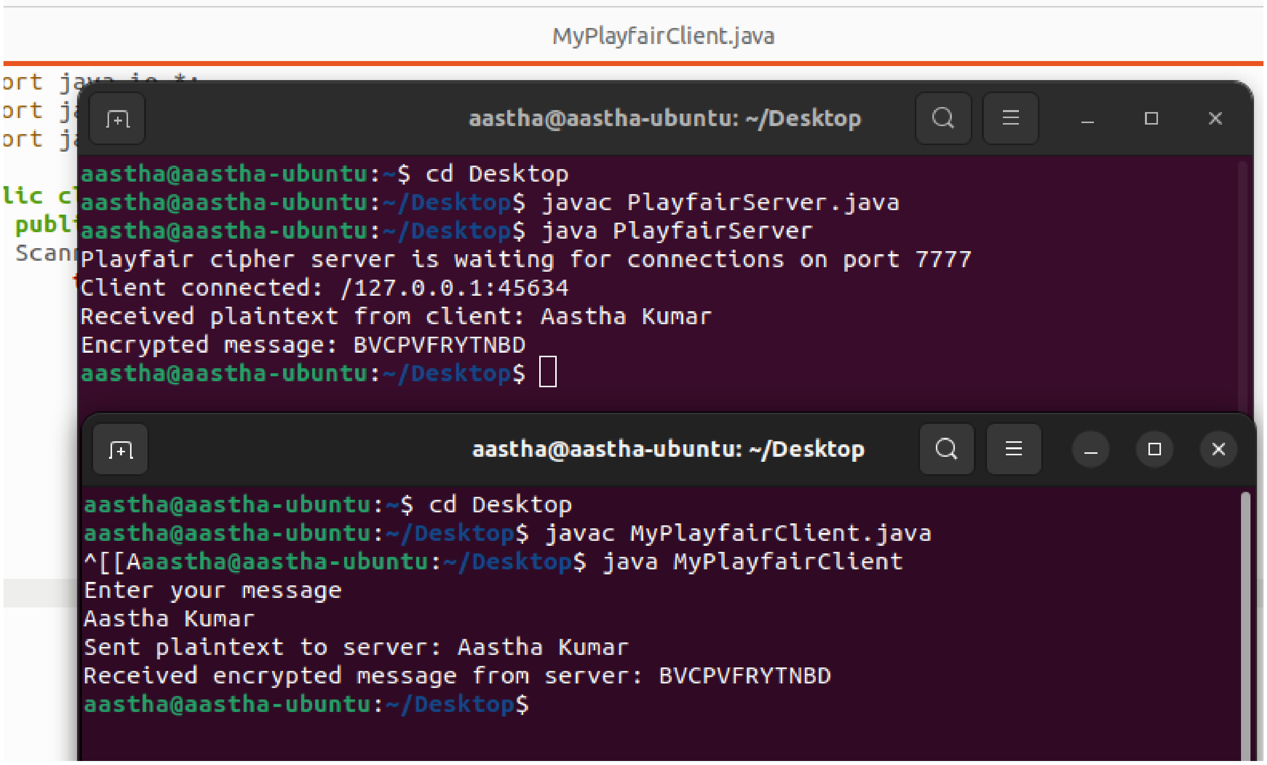
```
public class MyPlayfairClient {
    public static void main(String[] args) {
        try {
            // Connect to the server running on localhost at port 7777
            Socket s = new Socket("localhost", 7777);

            // Get the output stream to send data to the server
            OutputStream outToServer = s.getOutputStream();
            DataOutputStream dos = new DataOutputStream(outToServer);

            // Send a plaintext message to the server
            String plaintext = "HELLOPLAYFAIR";
            dos.writeUTF(plaintext);
            System.out.println("Sent plaintext to server: " + plaintext);

            // Read the encrypted message from the server
            DataInputStream dis = new DataInputStream(s.getInputStream());
            String encryptedMessage = dis.readUTF();
            System.out.println("Received encrypted message from server: " + encryptedMessage);

            // Close the client socket
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

PlayfairServer.java

```
import java.io.*;
import java.net.*;
import java.util.HashSet;
import java.util.Set;

public class PlayfairServer {

    public static void main(String[] args) {
        try {
            // Server socket setup
            int port = 7777;
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Playfair cipher server is waiting for connections on port " + port);

            // Accept client connection
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getRemoteSocketAddress());

            // Input and output streams
            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream dos = new DataOutputStream(clientSocket.getOutputStream());
            // Receive plaintext from client
            String plaintext = dis.readUTF();
            System.out.println("Received plaintext from client: " + plaintext);

            // Choose or receive key (if necessary)
            String key = "KEYWORD"; // Example key (modify as needed)
            // String key = dis.readUTF(); // To receive key from client

            // Validate key (optional)
            if (!validateKey(key)) {
                throw new IllegalArgumentException("Invalid key: Key must be unique characters
(excluding 'J')");
            }

            // Prepare plaintext
            String preparedPlaintext = preparePlaintext(plaintext);

            // Encrypt plaintext using Playfair cipher
            String encryptedText = playfairEncrypt(preparedPlaintext, key);
            System.out.println("Encrypted message: " + encryptedText);

            // Send encrypted message back to client
            dos.writeUTF(encryptedText);

            // Close connections
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
        }
    }

    public static char[][] generateKeyMatrix(String key) {
        // Remove duplicates and convert to uppercase
        key = String.join("", key.toUpperCase().chars().distinct().mapToObj(ch ->
String.valueOf((char) ch)).toArray(String[]::new));

        // Initialize the key matrix
        char[][] keyMatrix = new char[5][5];
        String alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"; // No 'J' in Playfair

        int row = 0, col = 0;
        for (char ch : key.toCharArray()) {
            keyMatrix[row][col] = ch;
            col++;
            if (col == 5) {
                col = 0;
                row++;
            }
        }

        // Fill the remaining empty cells with the remaining alphabet
        for (char ch : alphabet.toCharArray()) {
            if (ch != 'J' && key.indexOf(ch) == -1) {
                keyMatrix[row][col] = ch;
                col++;
                if (col == 5) {
                    col = 0;
                    row++;
                }
            }
        }

        return keyMatrix;
      }

    public static String preparePlaintext(String plaintext) {
        // Remove spaces and convert to uppercase
        plaintext = plaintext.replace(" ", "").toUpperCase();

        // Replace 'J' with 'I' and add 'X' for odd length
        StringBuilder preparedPlaintext = new StringBuilder();
        for (int i = 0; i < plaintext.length(); i++) {
            char ch = plaintext.charAt(i);
            preparedPlaintext.append(ch);
            if (i + 1 < plaintext.length() && ch == plaintext.charAt(i + 1)) {
                preparedPlaintext.append('X');
            }
        }

        if (preparedPlaintext.length() % 2 != 0) {
            preparedPlaintext.append('X');
        }

        return preparedPlaintext.toString();
    }


    public static String playfairEncrypt(String plaintext, String key) {
    // Remove duplicates and convert to uppercase
    key = String.join("", key.toUpperCase().chars().distinct().mapToObj(ch ->
String.valueOf((char) ch)).toArray(String[]::new));

    // Initialize the key matrix
    char[][] keyMatrix = new char[5][5];
    String alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"; // No 'J' in Playfair

    int row = 0, col = 0;
    for (char ch : key.toCharArray()) {
        keyMatrix[row][col] = ch;
        col++;
        if (col == 5) {
            col = 0;
            row++;
        }
    }

    // Fill the remaining empty cells with the remaining alphabet
    for (char ch : alphabet.toCharArray()) {
        if (ch != 'J' && key.indexOf(ch) == -1) {
            keyMatrix[row][col] = ch;
            col++;
            if (col == 5) {
                col = 0;
                row++;
            }
        }
    }

    // Prepare the plaintext (replace 'J' with 'I' and add 'X' for odd length)
    plaintext = preparePlaintext(plaintext);

    // Encrypt the plaintext using Playfair rules
    StringBuilder encryptedText = new StringBuilder();
    for (int i = 0; i < plaintext.length(); i += 2) {
        char firstChar = plaintext.charAt(i);
        char secondChar = plaintext.charAt(i + 1);

        int[] firstPos = findPosition(keyMatrix, firstChar);
        int[] secondPos = findPosition(keyMatrix, secondChar);

        if (firstPos[0] == secondPos[0]) { // Same row
            encryptedText.append(keyMatrix[firstPos[0]][(firstPos[1] + 1) % 5]);
            encryptedText.append(keyMatrix[secondPos[0]][(secondPos[1] + 1) % 5]);
        } else if (firstPos[1] == secondPos[1]) { // Same column
            encryptedText.append(keyMatrix[(firstPos[0] + 1) % 5][firstPos[1]]);
            encryptedText.append(keyMatrix[(secondPos[0] + 1) % 5][secondPos[1]]);
        } else { // Form a rectangle
            encryptedText.append(keyMatrix[firstPos[0]][secondPos[1]]);
            encryptedText.append(keyMatrix[secondPos[0]][firstPos[1]]);
        }
    }

    return encryptedText.toString();
}

    public static int[] findPosition(char[][] keyMatrix, char ch) {
        for (int i = 0; i < keyMatrix.length; i++) {
            for (int j = 0; j < keyMatrix[i].length; j++) {
                if (keyMatrix[i][j] == ch) {
```

```
                    return new int[]{i, j};
                }
            }
        }
        return null;  // Character not found
    }

    public static boolean validateKey(String key) {
        if (key.length() < 5) {
            return false; // Key must be at least 5 characters long
        }

        Set<Character> charSet = new HashSet<>();
        for (char ch : key.toUpperCase().toCharArray()) {
            if (ch == 'J') {
                return false; // 'J' is not allowed
            }
            if (!charSet.add(ch)) {
                return false; // Duplicate characters found
            }
        }
        return true;
    }
}
```

OUTPUT:


MyPlayfairClient.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac PlayfairServer.java
aastha@aastha-ubuntu:~/Desktop$ java PlayfairServer
Playfair cipher server is waiting for connections on port 7777
Client connected: /127.0.0.1:45634
Received plaintext from client: Aastha Kumar
Encrypted message: BVCPVFRYTNBD
aastha@aastha-ubuntu:~/Desktop$
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyPlayfairClient.java
^[[Aaastha@aastha-ubuntu:~/Desktop$ java MyPlayfairClient
Enter your message
Aastha Kumar
Sent plaintext to server: Aastha Kumar
Received encrypted message from server: BVCPVFRYTNBD
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 1d 05/01/24

TITLE:

Write a socket program to demonstrate Row Transposition Cipher.

ALGORITHM:

Row Transposition Cipher Server:

1. **Create a ServerSocket**: Initialize a ServerSocket object on a specific port (e.g., 8888).
2. **Wait for Connections**: Use accept() method to wait for incoming client connections. This method blocks until a client connects.
3. **Accept Client Connection**: When a client connects, accept the connection and get a Socket object representing the client.
4. **Read Data from the Client**: Create a DataInputStream from the client socket to read data sent by the client (e.g., the plaintext message).
5. **Encrypt the Plaintext**: Implement the Row Transposition Cipher encryption algorithm using the provided key (e.g., "KEYWORD").
6. **Send the Encrypted Message to the Client**: Create a DataOutputStream from the client socket and use writeUTF() to send the encrypted message back to the client.
7. **Close the Server Socket**: Close the server socket using close().

Row Transposition Cipher Client:

1. **Create a Socket**: Initialize a Socket object to connect to the server (e.g., "localhost" and port 8888).
2. **Get Output Stream**: Get the output stream from the socket using getOutputStream().
3. **Send a Plaintext Message to the Server**: Create a DataOutputStream and use writeUTF() to send a plaintext message to the server.
4. **Read the Encrypted Message from the Server**: Create a DataInputStream from the socket to read the encrypted message sent by the server.
5. **Close the Client Socket**: Close the client socket using close().

CODE:

MyRowTranspositionClient.java

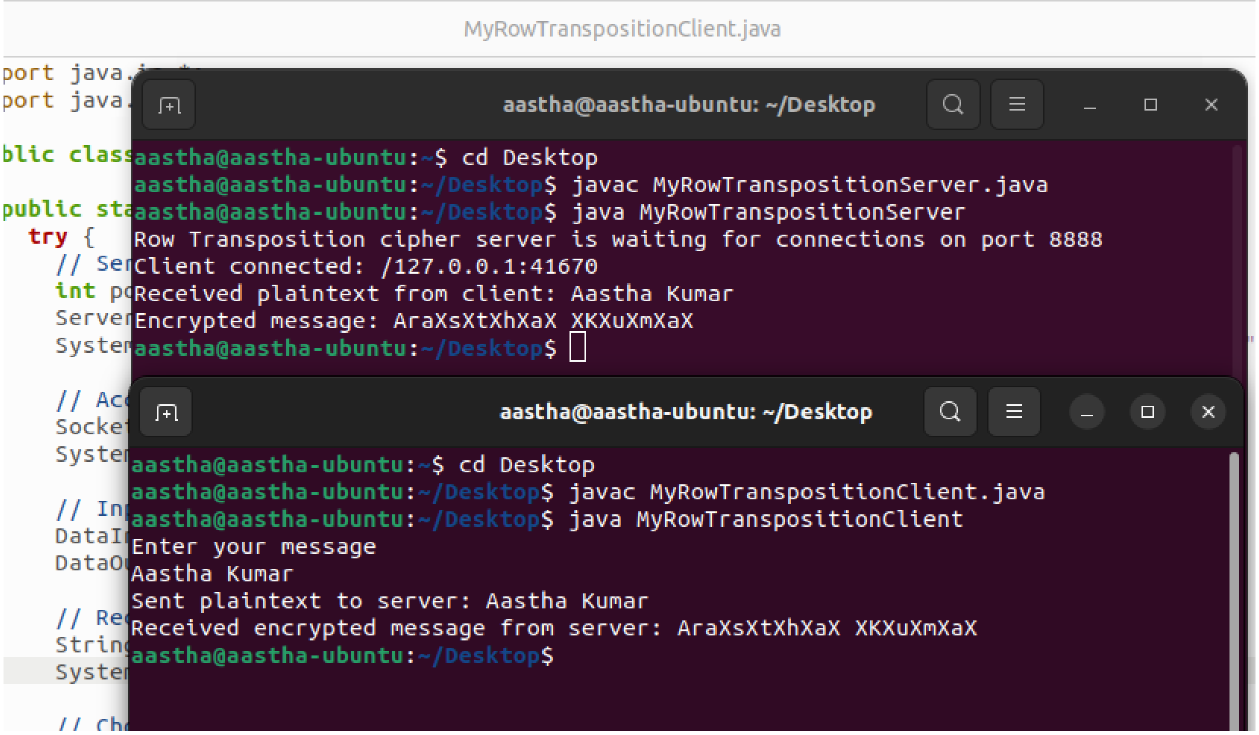
```
import java.io.*;
import java.net.*;
import java.util.*;

public class MyRowTranspositionClient {

  public static void main(String[] args) {
```

```
  Scanner sc=new Scanner(System.in);
    try {
      // Connect to the server running on localhost at port 8888
      Socket s = new Socket(“localhost”, 8888);

      // Get the output stream
      DataOutputStream dos = new DataOutputStream(s.getOutputStream());

      // Send a plaintext message to the server
      System.out.println(“Enter your message”);
      String plaintext = sc.nextLine();
      dos.writeUTF(plaintext);
      System.out.println(“Sent plaintext to server: “ + plaintext);

      // Get the input stream
      DataInputStream dis = new DataInputStream(s.getInputStream());

      // Wait for data before reading (loop)
      while (dis.available() == 0) {
        try {
          Thread.sleep(100); // Sleep for a short time to avoid busy waiting
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }

      // Read the encrypted message from the server
      String encryptedMessage = dis.readUTF();
      System.out.println(“Received encrypted message from server: “ +
encryptedMessage);

      // Close the client socket
      s.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

MyRowTranspositionServer.java

```
import java.io.*;
import java.net.*;

public class MyRowTranspositionServer {
  public static void main(String[] args) {
    try {
      // Server socket setup
      int port = 8888;
      ServerSocket ss = new ServerSocket(port);
      System.out.println("Row Transposition cipher server is waiting for connections on
port " + port);

      // Accept client connection
      Socket s = ss.accept();
      System.out.println("Client connected: " + s.getRemoteSocketAddress());

      // Input and output streams
      DataInputStream dis = new DataInputStream(s.getInputStream());
      DataOutputStream dos = new DataOutputStream(s.getOutputStream());

      // Receive plaintext from client
      String plaintext = dis.readUTF();
      System.out.println("Received plaintext from client: " + plaintext);

      // Choose or receive key (if necessary)
      String key = "ABCDEFGHIJK"; // Example key (modify as needed)
      // String key = dis.readUTF(); // To receive key from client

      // Encrypt the plaintext using Row Transposition cipher
      String encryptedText;
      if (plaintext.isEmpty()) {
        // Handle empty plaintext (optional: throw exception or send error message)
        encryptedText = "Error: Plaintext cannot be empty";
      } else {
        encryptedText = rowTranspositionEncrypt(plaintext, key);
      }
      System.out.println("Encrypted message: " + encryptedText);

      // Send the encrypted message back to the client
      dos.writeUTF(encryptedText);

      // Close connections
      s.close();
      ss.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public static String rowTranspositionEncrypt(String plaintext, String key) {
    // Ensure key is not empty
    if (key.isEmpty()) {
      throw new IllegalArgumentException("Key cannot be empty");
    }

    // Remove duplicates and convert to uppercase
    key = String.join("", key.toUpperCase().chars().distinct().mapToObj(ch ->
String.valueOf((char) ch)).toArray(String[]::new));

    // Calculate the number of columns based on the key length
    int numColumns = key.length();

    // Pad the plaintext with 'X' if needed
    while (plaintext.length() % numColumns != 0) {
      plaintext += 'X';
    }

    // Create the key order (indices of columns)
    int[] keyOrder = new int[numColumns];
    for (int i = 0; i < numColumns; i++) {
      char keyChar = (char) ('A' + i); // Use the actual character for comparison
      int index = key.indexOf(keyChar);
      if (index == -1) {
        // Handle case where character not found in key (optional: use default value or
throw exception)
        throw new IllegalArgumentException("Invalid key: Character '" + keyChar + "' not
found");
      }
      keyOrder[i] = index;
    }

    // Initialize the matrix
    char[][] matrix = new char[plaintext.length() / numColumns][numColumns];

    // Fill the matrix with the plaintext
    for (int i = 0; i < plaintext.length(); i++) {
      int row = i / numColumns;
      int col = keyOrder[i % numColumns];
      matrix[row][col] = plaintext.charAt(i);
    }

    // Read the encrypted message column-wise
    StringBuilder encryptedMessage = new StringBuilder();
    for (int col = 0; col < numColumns; col++) {
      for (int row = 0; row < matrix.length; row++) {
        encryptedMessage.append(matrix[row][col]);
      }
    }

    return encryptedMessage.toString();
  }
}
```

OUTPUT:



MyRowTranspositionClient.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyRowTranspositionServer.java
aastha@aastha-ubuntu:~/Desktop$ java MyRowTranspositionServer
Row Transposition cipher server is waiting for connections on port 8888
Client connected: /127.0.0.1:41670
Received plaintext from client: Aastha Kumar
Encrypted message: AraXsXtXhXaX XKXuXmXaX
aastha@aastha-ubuntu:~/Desktop$
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyRowTranspositionClient.java
aastha@aastha-ubuntu:~/Desktop$ java MyRowTranspositionClient
Enter your message
Aastha Kumar
Sent plaintext to server: Aastha Kumar
Received encrypted message from server: AraXsXtXhXaX XKXuXmXaX
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 2a 07/01/24

TITLE:

Develop Euclidean Algorithm.

ALGORITHM:

Euclidean Algorithm Server:

1. **Create a ServerSocket**: Initialize a ServerSocket object on a specific port (e.g., 9999).
2. **Wait for Connections**: Use accept() method to wait for incoming client connections. This method blocks until a client connects.
3. **Accept Client Connection**: When a client connects, accept the connection and get a Socket object representing the client.
4. **Read Data from the Client**: Create a DataInputStream from the client socket to read data sent by the client (e.g., two integers for which we want to find the greatest common divisor).
5. **Compute the GCD using Euclidean Algorithm**: Implement the Euclidean Algorithm to find the greatest common divisor (GCD) of the two integers.
6. **Send the GCD back to the Client**: Create a DataOutputStream from the client socket and use writeInt() to send the GCD back to the client.
7. **Close the Server Socket**: Close the server socket using close().

Euclidean Algorithm Client:

1. **Create a Socket**: Initialize a Socket object to connect to the server (e.g., “localhost” and port 9999).
2. **Get Input and Output Streams**: Get the input and output streams from the socket using getInputStream() and getOutputStream().
3. **Send Two Integers to the Server**: Create a DataOutputStream and use writeInt() to send two integers to the server.
4. **Read the GCD from the Server**: Create a DataInputStream from the socket to read the GCD sent by the server.
5. **Close the Client Socket**: Close the client socket using close()

CODE:

MyEuclideanClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
```

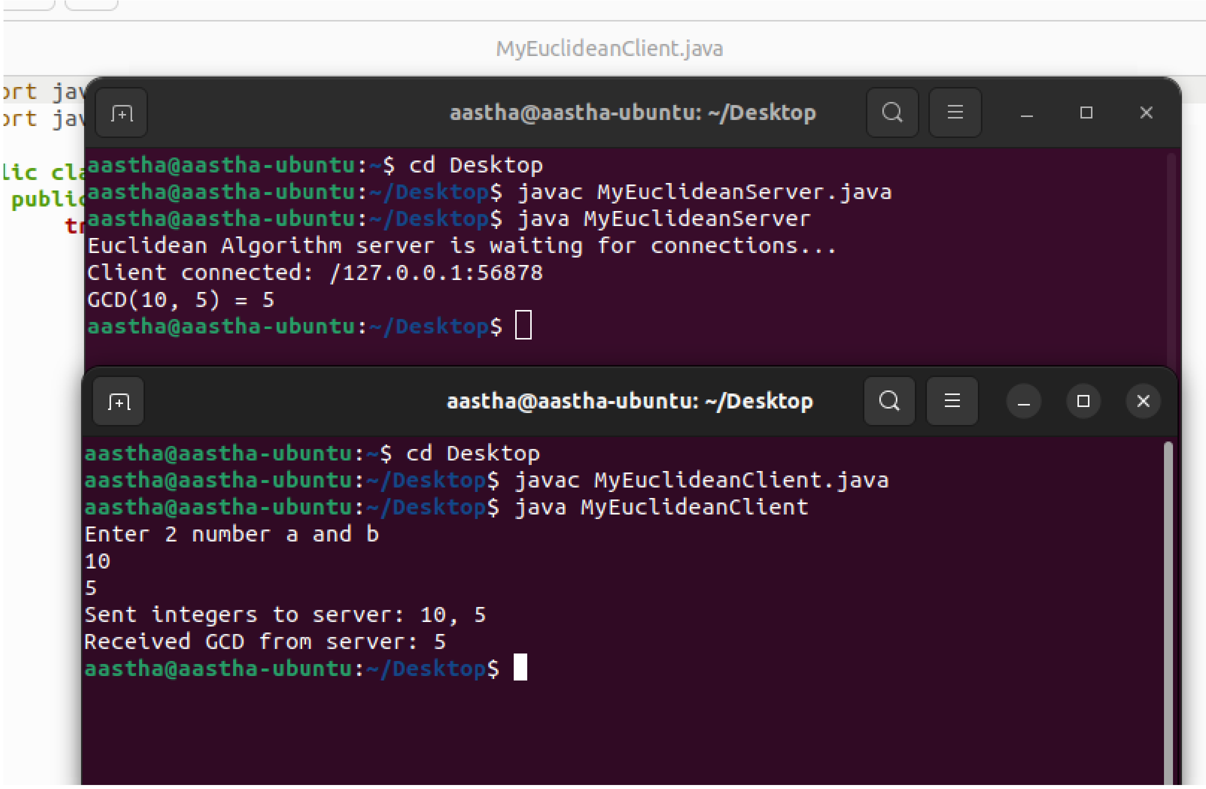
```
public class MyEuclideanClient {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
        try {
            // Connect to the server running on localhost at port 9999
            Socket s = new Socket("localhost", 9999);

            // Get the output stream to send data to the server
            OutputStream outToServer = s.getOutputStream();
            DataOutputStream dos = new DataOutputStream(outToServer);

            // Send two integers to the server
            System.out.println("Enter 2 number a and b");
            int a = sc.nextInt();
            int b = sc.nextInt();
            dos.writeInt(a);
            dos.writeInt(b);
            System.out.println("Sent integers to server: " + a + ", " + b);

            // Read the GCD from the server
            DataInputStream dis = new DataInputStream(s.getInputStream());
            int gcd = dis.readInt();
            System.out.println("Received GCD from server: " + gcd);

            // Close the client socket
            s.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyEuclideanServer.java

```
import java.io.*;
import java.net.*;
```

```
public class MyEuclideanServer {
    public static void main(String[] args) {
        try {
            // Create a server socket on port 9999
            ServerSocket ss = new ServerSocket(9999);
            System.out.println("Euclidean Algorithm server is waiting for connections...");

            // Accept incoming client connections
            Socket s = ss.accept();
            System.out.println("Client connected: " + s.getRemoteSocketAddress());

            // Read two integers from the client
            DataInputStream dis = new DataInputStream(s.getInputStream());
            int a = dis.readInt();
            int b = dis.readInt();

            // Compute the GCD using Euclidean Algorithm
            int gcd = computeGCD(a, b);
            System.out.println("GCD(" + a + ", " + b + ") = " + gcd);

            // Send the GCD back to the client
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());
            dos.writeInt(gcd);

            // Close the server socket
            ss.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static int computeGCD(int a, int b) {
        if (b == 0) {
            return a;
        }
        return computeGCD(b, a % b);
    }
}
```

OUTPUT:



MyEuclideanClient.java
ort jav
ort jav
lic cla
publi
tr
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyEuclideanServer.java
aastha@aastha-ubuntu:~/Desktop$ java MyEuclideanServer
Euclidean Algorithm server is waiting for connections...
Client connected: /127.0.0.1:56878
GCD(10, 5) = 5
aastha@aastha-ubuntu:~/Desktop$
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MyEuclideanClient.java
aastha@aastha-ubuntu:~/Desktop$ java MyEuclideanClient
Enter 2 number a and b
10
5
Sent integers to server: 10, 5
Received GCD from server: 5
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 2b 07/01/24

AIM:

Demonstrate extended Euclidean Algorithm.

ALGORITHM:

Extended Euclidean Algorithm Server (MyExtendedEuclideanServer.java):

1. **Create a ServerSocket**: Initialize a ServerSocket object on a specific port (e.g., 7777).
2. **Wait for Connections**: Use accept() method to wait for incoming client connections. This method blocks until a client connects.
3. **Accept Client Connection**: When a client connects, accept the connection and get a Socket object representing the client.
4. **Read Data from the Client**: Create a DataInputStream from the client socket to read data sent by the client (e.g., two integers for which we want to find the Bézout coefficients).
5. **Compute the Bézout Coefficients using Extended Euclidean Algorithm**: Implement the Extended Euclidean Algorithm to find the Bézout coefficients (s and t) such that as + bt = gcd(a, b).
6. **Send the Bézout Coefficients back to the Client**: Create a DataOutputStream from the client socket and use writeInt() to send the Bézout coefficients back to the client.
7. **Close the Server Socket**: Close the server socket using close().

Extended Euclidean Algorithm Client (MyExtendedEuclideanClient.java):

1. **Create a Socket**: Initialize a Socket object to connect to the server (e.g., “localhost” and port 7777).
2. **Get Input and Output Streams**: Get the input and output streams from the socket using getInputStream() and getOutputStream().
3. **Send Two Integers to the Server**: Create a DataOutputStream and use writeInt() to send two integers to the server.
4. **Read the Bézout Coefficients from the Server**: Create a DataInputStream from the socket to read the Bézout coefficients sent by the server.
5. **Close the Client Socket**: Close the client socket using close()

CODE:

MyExtendedEuclideanClient.java

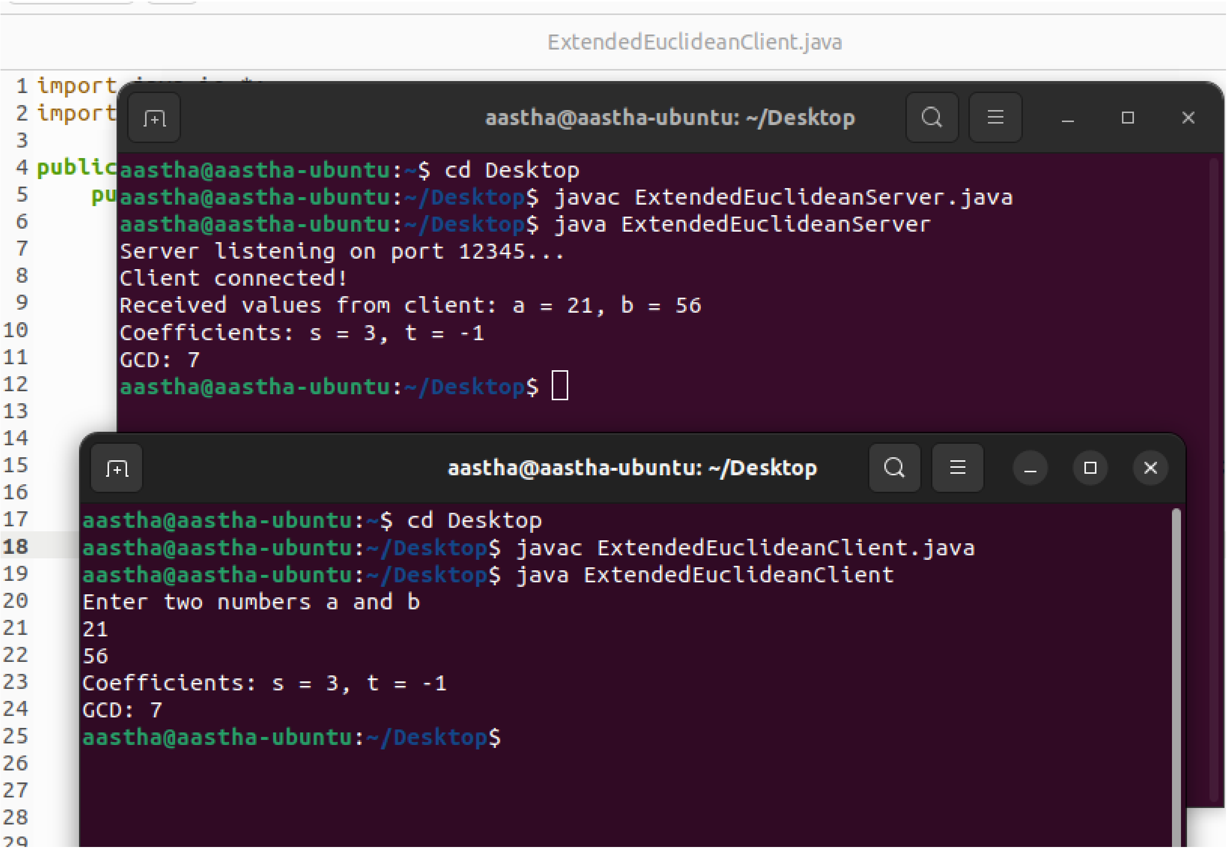
```
import java.io.*;
import java.net.*;
import java.util.*;

public class ExtendedEuclideanClient {
    public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
        try {
            Socket socket = new Socket("localhost", 12345); // Server address and port

            DataInputStream in = new DataInputStream(socket.getInputStream());
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());

            // Input values
            System.out.println("Enter two numbers a and b");
            long a = sc.nextInt(); // First number
            long b = sc.nextInt(); // Second number

            // Send input values to server
            out.writeLong(a);
            out.writeLong(b);

            // Receive coefficients and GCD from server
            long s = in.readLong();
            long t = in.readLong();
            long gcd = in.readLong();

            System.out.println("Coefficients: s = " + s + ", t = " + t);
            System.out.println("GCD: " + gcd);

            // Clean up
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyExtendedEuclideanServer.java

```
import java.io.*;
import java.net.*;
```

```
public class ExtendedEuclideanServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345); // Port number

            System.out.println("Server listening on port 12345...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected!");

            DataInputStream in = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());

            // Read input values from client
            long a = in.readLong();
            long b = in.readLong();

            System.out.println("Received values from client: a = " + a + ", b = " + b);

            // Perform extended Euclidean algorithm
            long oldR = a, r = b;
            long oldS = 1, s = 0;
            long oldT = 0, t = 1;

            while (r != 0) {
                long quotient = oldR / r;
                long rTemp = r;
                r = oldR - quotient * r;
                oldR = rTemp;

                long sTemp = s;
                s = oldS - quotient * s;
                oldS = sTemp;

                long tTemp = t;
                t = oldT - quotient * t;
                oldT = tTemp;
            }

            System.out.println("Coefficients: s = " + oldS + ", t = " + oldT);
            System.out.println("GCD: " + oldR);

            // Send coefficients and GCD back to client
            out.writeLong(oldS);
            out.writeLong(oldT);
            out.writeLong(oldR);

            // Clean up
            in.close();
            out.close();
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:


ExtendedEuclideanClient.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac ExtendedEuclideanServer.java
aastha@aastha-ubuntu:~/Desktop$ java ExtendedEuclideanServer
Server listening on port 12345...
Client connected!
Received values from client: a = 21, b = 56
Coefficients: s = 3, t = -1
GCD: 7
aastha@aastha-ubuntu:~/Desktop$
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac ExtendedEuclideanClient.java
aastha@aastha-ubuntu:~/Desktop$ java ExtendedEuclideanClient
Enter two numbers a and b
21
56
Coefficients: s = 3, t = -1
GCD: 7
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 2c 07/01/24

TITLE:

Demonstrate Euler’s Theorem.

ALGORITHM:

Server-Side Algorithm:

1. **Create a TCP socket** using the socket() function.
2. **Bind the socket** to a specific IP address and port using the bind() function.
3. **Listen for incoming connections** using the listen() function.
4. When a client connects, **accept the connection** using the accept() function.
5. **Receive data from the client** using the recv() function.
6. **Process the received data** (e.g., perform calculations, handle requests).
7. **Send a response back to the client** using the send() function.
8. **Close the connection** using the close() function.

Client-Side Algorithm:

1. **Create a TCP socket** using the socket() function.
2. **Connect to the server** (identified by its IP address and port) using the connect() function.
3. **Send data to the server** using the send() function.
4. **Receive a response from the server** using the recv() function.
5. **Process the server’s response** (e.g., display it to the user, save it to a file).
6. **Close the connection** using the close() function

CODE:

EulerClient.java

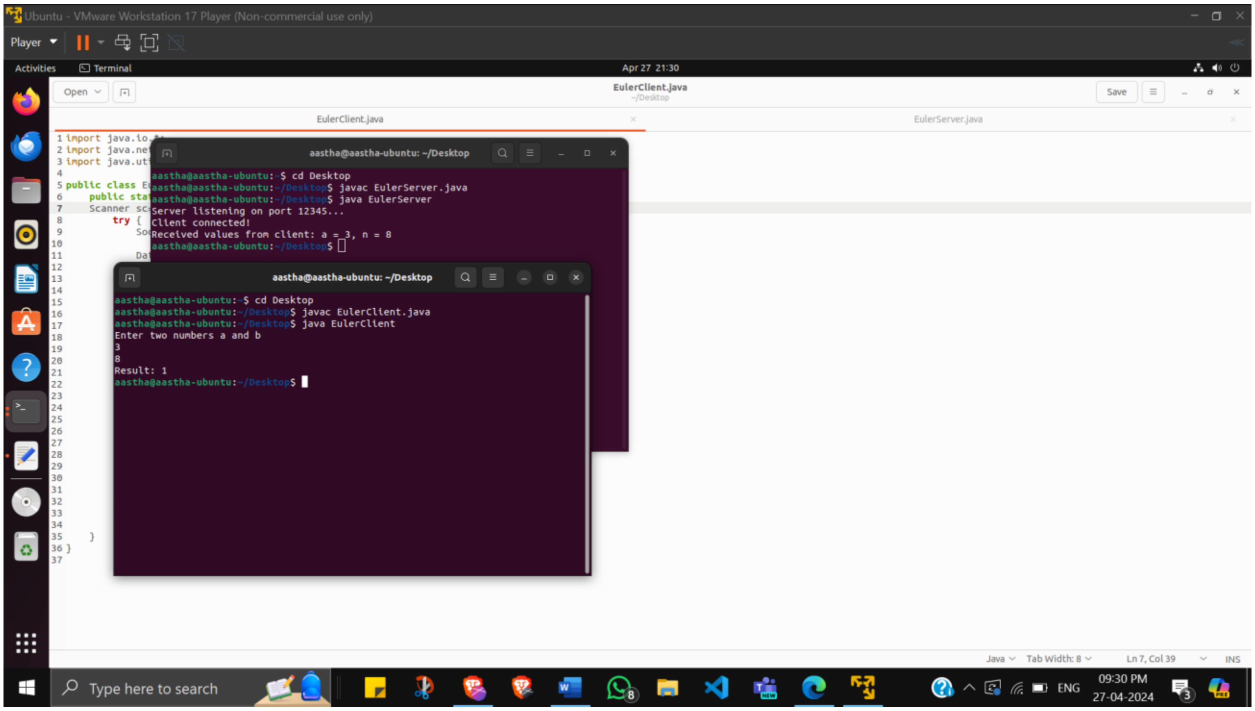
```
import java.io.*;
import java.net.*;
import java.util.*;

public class EulerClient {
  public static void main(String[] args) {
  Scanner sc=new Scanner(System.in);
    try {
      Socket socket = new Socket("localhost", 12345); // Server address and port

      DataInputStream in = new DataInputStream(socket.getInputStream());
      DataOutputStream out = new DataOutputStream(socket.getOutputStream());
```

```
            // Input values
            System.out.println("Enter two numbers a and b");
            int a = sc.nextInt();
            int n = sc.nextInt();

            // Send input values to server
            out.writeInt(a);
            out.writeInt(n);

            // Receive the result from the server
            int result = in.readInt();

            System.out.println("Result: " + result);

            // Clean up
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

EulerServer.java

```
import java.io.*;
import java.net.*;

public class EulerServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345); // Port number

            System.out.println("Server listening on port 12345...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected!");

            DataInputStream in = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());

            // Read input values from client
            int a = in.readInt(); // Integer 'a'
            int n = in.readInt(); // Positive integer 'n'

            System.out.println("Received values from client: a = " + a + ", n = " + n);

            // Calculate Euler's totient function
            int phiN = calculateTotient(n);
            // Calculate a^phiN mod n
            int result = modPow(a, phiN, n);

            // Send the result back to the client
            out.writeInt(result);

            // Clean up
            in.close();
            out.close();
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Calculate Euler's totient function
    private static int calculateTotient(int n) {
        int result = n;
        for (int i = 2; i * i <= n; ++i) {
            if (n % i == 0) {
                while (n % i == 0) {
                    n /= i;
                }
                result -= result / i;
            }
        }
        if (n > 1) {
            result -= result / n;
        }
        return result;
    }

    // Calculate (base^exponent) % modulus
    private static int modPow(int base, int exponent, int modulus) {
        int result = 1;
        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) % modulus;
            }
            base = (base * base) % modulus;
            exponent /= 2;
        }
        return result;
    }
}
```

OUTPUT:



aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac EulerServer.java
aastha@aastha-ubuntu:~/Desktop$ java EulerServer
Server listening on port 12345...
Client connected!
Received values from client: a = 3, n = 8
aastha@aastha-ubuntu:~/Desktop$ 
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac EulerClient.java
aastha@aastha-ubuntu:~/Desktop$ java EulerClient
Enter two numbers a and b
3
8
Result: 1
aastha@aastha-ubuntu:~/Desktop$ 
```

Experiment 2d 07/01/24

TITLE:

Demonstrate Miller Rabin Algorithm.

ALGORITHM:

Server-Side Algorithm:

1. **Create a server socket** using the ServerSocket class and specify a port number.
2. **Listen for incoming connections** using the accept() method. This will block until a client connects.
3. **Accept the client connection** and create a new socket for communication.
4. **Receive data from the client** using the input stream of the socket.
5. **Process the received data** (e.g., perform calculations, handle requests).
6. **Send a response back to the client** using the output stream of the socket.
7. **Close the client socket** after communication is complete.
8. **Repeat steps 3-7** to handle multiple client connections.

Client-Side Algorithm:

1. **Create a client socket** using the Socket class and specify the server's IP address and port number.
2. **Connect to the server** using the connect() method.
3. **Send data to the server** using the output stream of the socket.
4. **Receive a response from the server** using the input stream of the socket.
5. **Process the server's response** (e.g., display it to the user, save it to a file).
6. **Close the client socket** after communication is complete.

CODE:

MillerRabinClient.java

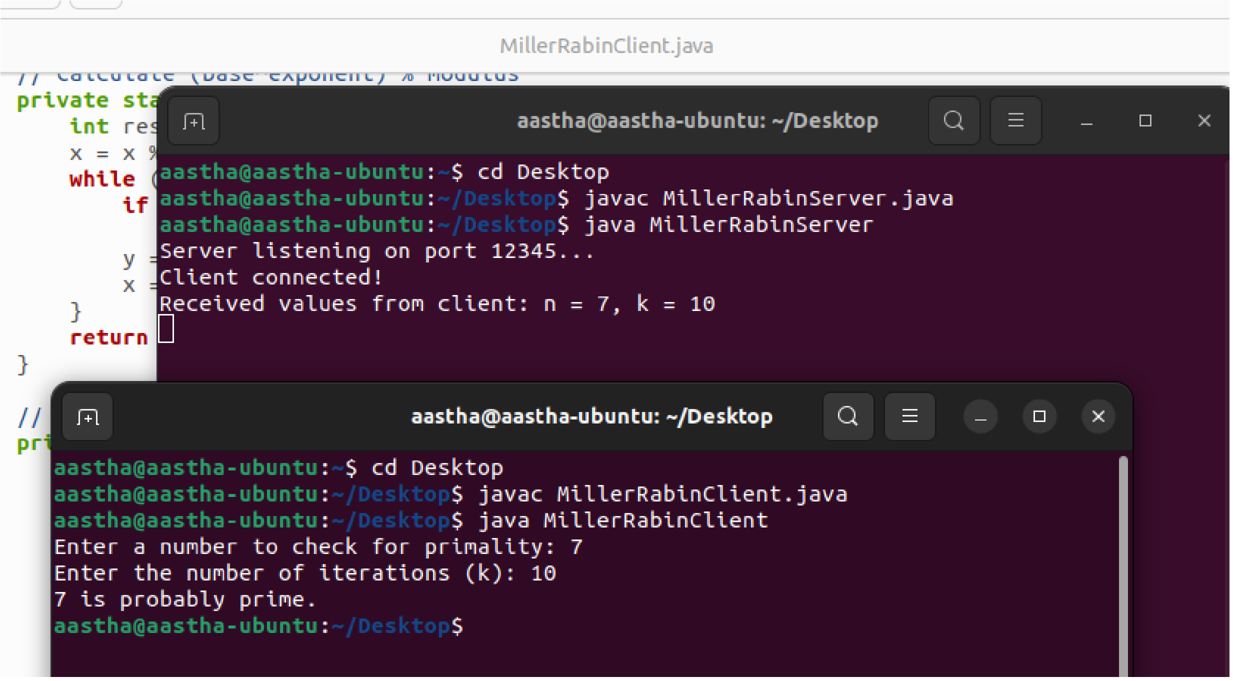
```
import java.io.*;
import java.net.*;
import java.util.*;

public class MillerRabinClient {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
            Socket socket = new Socket("localhost", 12345); // Server address and port
            DataInputStream in = new DataInputStream(socket.getInputStream());
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());

            // Input values
            System.out.print("Enter a number to check for primality: ");
            int n = sc.nextInt();
            System.out.print("Enter the number of iterations (k): ");
            int k = sc.nextInt();

            // Send input values to server
            out.writeInt(n);
            out.writeInt(k);

            // Receive the result from the server
            boolean isPrime = in.readBoolean();

            if (isPrime) {
                System.out.println(n + " is probably prime.");
            } else {
                System.out.println(n + " is composite.");
            }

            // Clean up
            in.close();
            out.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MillerRabinServer.java

```
import java.io.*;
import java.net.*;
import java.util.Random;
```

```
public class MillerRabinServer {

    // Calculate (base^exponent) % modulus
    private static int power(int x, int y, int p) {
        int res = 1;
        x = x % p;
        while (y > 0) {
            if (y % 2 == 1)
                res = (res * x) % p;
            y = y >> 1;
            x = (x * x) % p;
        }
        return res;
    }

    // Perform a single Miller-Rabin test
    private static boolean millerTest(int d, int n) {
        Random rand = new Random();
        int a = 2 + rand.nextInt(n - 4); // Pick a random number 'a' in range [2, n-2]
        int x = power(a, d, n);
        if (x == 1 || x == n - 1)
            return true;
        while (d != n - 1) {
            x = (x * x) % n;
            d *= 2;
            if (x == 1 || x == n - 1)
                return true;
        }
        return false;
    }

    // Check if a number is prime using Miller-Rabin
    private static boolean isPrime(int n, int k) {
        if (n <= 1 || n == 4)
            return false;
        if (n <= 3)
            return true;

        int d = n - 1;
        while (d % 2 == 0)
            d /= 2;

        for (int i = 0; i < k; i++) {
            if (!millerTest(d, n))
                return false;
        }
        return true;
    }

    // Main method to handle client connections
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345); // Port number

            System.out.println("Server listening on port 12345...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected!");

                DataInputStream in = new DataInputStream(clientSocket.getInputStream());
                DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());

                // Read input values from client
                int n = in.readInt(); // Number to check for primality
                int k = in.readInt(); // Number of iterations

                System.out.println("Received values from client: n = " + n + ", k = " + k);

                boolean isPrime = isPrime(n, k);

                // Send the result back to the client
                out.writeBoolean(isPrime);

                // Clean up
                in.close();
                out.close();
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:


MillerRabinClient.java
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MillerRabinServer.java
aastha@aastha-ubuntu:~/Desktop$ java MillerRabinServer
Server listening on port 12345...
Client connected!
Received values from client: n = 7, k = 10
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac MillerRabinClient.java
aastha@aastha-ubuntu:~/Desktop$ java MillerRabinClient
Enter a number to check for primality: 7
Enter the number of iterations (k): 10
7 is probably prime.
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 3 18/01/24

AIM:

Consider a sender and receiver who need to exchange data confidentially using symmetric encryption. Write a program that implements DES encryption and decryption using a 64-bit key size and 64-bit block size.

ALGORITHM:

Server (DesServer.java)

1. Server Initialization:

- Open a server socket on a specific port (e.g., 12345).
- Enter an infinite loop to continuously accept client connections.

2. Client Connection and Key Generation:

- For each accepted client connection, create a new thread for concurrent processing.
- Inside the thread:
  - Generate a new DES secret key (optional: consider key exchange for better security).
  - Create an initialization vector (IV) for the encryption mode.

3. Receive Message and Choose Action:

- Read the message sent by the client using DataInputStream.
- **Choose based on your need:**
  - **Option 1: Server stores encrypted files:**
    - Encrypt the message received from the client using the generated key and IV.
    - **(Modify server code):** Store the encrypted data on the server filesystem (modify file path as needed).
  - **Option 2: Send encrypted data back to client:**
    - Create a Cipher object in encryption mode using the generated key and IV.
    - Create a CipherOutputStream that wraps the client socket's output stream and encrypts data using the Cipher object.
    - Write the message (obtained from the client) to the CipherOutputStream, effectively sending the encrypted data back to the client.

4. Send Success Message:

- Send a success message to the client indicating successful encryption (optional).

5. Close Connections:

- Close the input and output streams associated with the client connection.
- Close the client socket.

Client (DesClient.java)

1. Connection and Stream Creation:

- Specify the server hostname and port number.
- Create a socket connection to the server.
- Obtain input and output streams associated with the client socket.

2. Send Message:

- Read a message to be encrypted from the user (modify as needed).
- Send the message to the server using DataOutputStream.

3. Receive Encrypted Data (if server sends it):

- Read bytes from the server's output stream using DataInputStream.
- The number of bytes read indicates the size of the received encrypted data.
- **Handle the received data based on your need:**
  - You can store the encrypted data locally on the client for further processing.
  - Here, the code demonstrates printing the received data in hex format (for demonstration purposes).

4. Close Connections:

- Close the input and output streams associated with the client socket.
- Close the client socket.

<u>CODE:</u>

DESClient.java

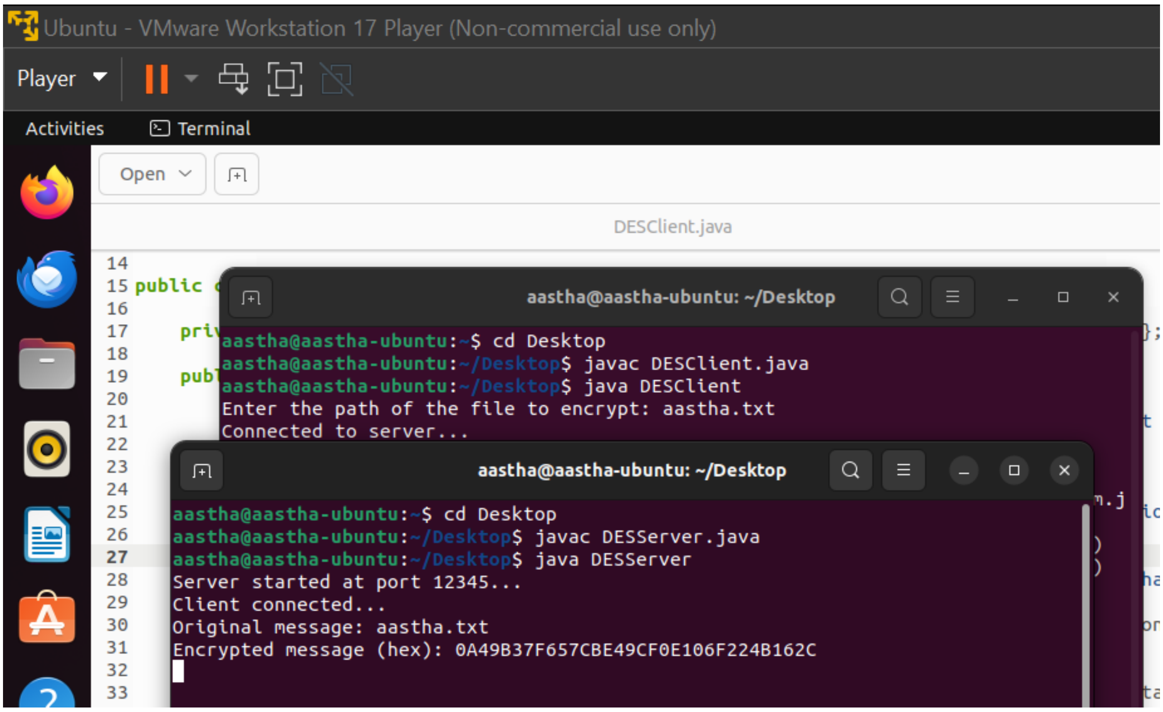
```
import java.io.*;

import java.net.*;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.AlgorithmParameterSpec;

import javax.crypto.Cipher;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.spec.IvParameterSpec;
```

```
public class DESClient {

    public static void main(String[] args) {
        try {
            // Server hostname and port (modify as needed)
            String serverHostname = "localhost";
            int portNumber = 12345;

            // Get the path of the file to be encrypted from the user
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Enter the path of the file to encrypt: ");
            String filePath = reader.readLine();

            // Create a socket connection to the server
            Socket clientSocket = new Socket(serverHostname, portNumber);
            System.out.println("Connected to server...");

            // Send the path of the file to the server
            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());
            out.writeUTF(filePath);

            // Receive a message from the server indicating success or failure
            DataInputStream in = new DataInputStream(clientSocket.getInputStream());
            String serverMessage = in.readUTF();
            System.out.println(serverMessage);
            // Close connections
            in.close();
            out.close();
            clientSocket.close();
        } catch (UnknownHostException e) {
            System.err.println("Error: Could not find server hostname.");
            e.printStackTrace();
        } catch (IOException e) {
            System.err.println("Error: I/O error during communication.");
            e.printStackTrace();
        }
    }
}
```

DESServer.java

```
import java.io.*;
import java.net.*;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class DESServer {

    private static final byte[] initialization_vector = { 22, 33, 11, 44, 55, 99, 66, 77 };

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345); // Server listens on port
12345
            System.out.println("Server started at port 12345...");

            while (true) {
                Socket clientSocket = serverSocket.accept(); // Wait for a client connection
                System.out.println("Client connected...");

                // Generate a new DES key for each connection (optional, consider key exchange
for better security)
                SecretKey secretKey = KeyGenerator.getInstance("DES").generateKey();
                AlgorithmParameterSpec ivParameterSpec = new
IvParameterSpec(initialization_vector);

                // Handle client communication in a separate thread for concurrency
                new Thread(new ClientHandler(clientSocket, secretKey,
ivParameterSpec)).start();
            }
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }


    private static class ClientHandler implements Runnable {


        private final Socket clientSocket;

        private final SecretKey secretKey;

        private final AlgorithmParameterSpec ivParameterSpec;


        public ClientHandler(Socket clientSocket, SecretKey secretKey,
AlgorithmParameterSpec ivParameterSpec) {

            this.clientSocket = clientSocket;

            this.secretKey = secretKey;

            this.ivParameterSpec = ivParameterSpec;

        }


        @Override

        public void run() {

            try (DataInputStream in = new DataInputStream(clientSocket.getInputStream());

                 DataOutputStream out = new
DataOutputStream(clientSocket.getOutputStream())) {


                // Receive the message from the client

                String message = in.readUTF();


                // Print the original message

                System.out.println("Original message: " + message);


                // Perform encryption
                byte[] encryptedData = encryption(message.getBytes(), secretKey,
ivParameterSpec);


                // Print the encrypted message in hex format (for demonstration)
                System.out.println("Encrypted message (hex): " + bytesToHex(encryptedData));


                // Send a success message to the client (optional)
                // out.writeUTF("Encryption successful!");


                // You can choose to send the encrypted data back to the client here (modify client
code to receive)
            } catch (IOException | InvalidKeyException | NoSuchAlgorithmException |
NoSuchPaddingException | InvalidAlgorithmParameterException e) {
                e.printStackTrace();
                // Send an error message to the client if encryption fails (optional)
            } finally {
                try {
                    clientSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }


        private byte[] encryption(byte[] message, SecretKey secretKey,
AlgorithmParameterSpec ivParameterSpec)
                throws IOException, InvalidKeyException, NoSuchAlgorithmException,
NoSuchPaddingException, InvalidAlgorithmParameterException {
            Cipher encrypt = Cipher.getInstance("DES/CBC/PKCS5Padding");
            encrypt.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec);
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            CipherOutputStream cos = new CipherOutputStream(baos, encrypt);
            cos.write(message);
            cos.close();
            return baos.toByteArray();
        }

        private static String bytesToHex(byte[] bytes) {
            StringBuilder sb = new StringBuilder();
            for (byte b : bytes) {
                sb.append(String.format("%02X", b));
            }
            return sb.toString();
        }
    }
}
```

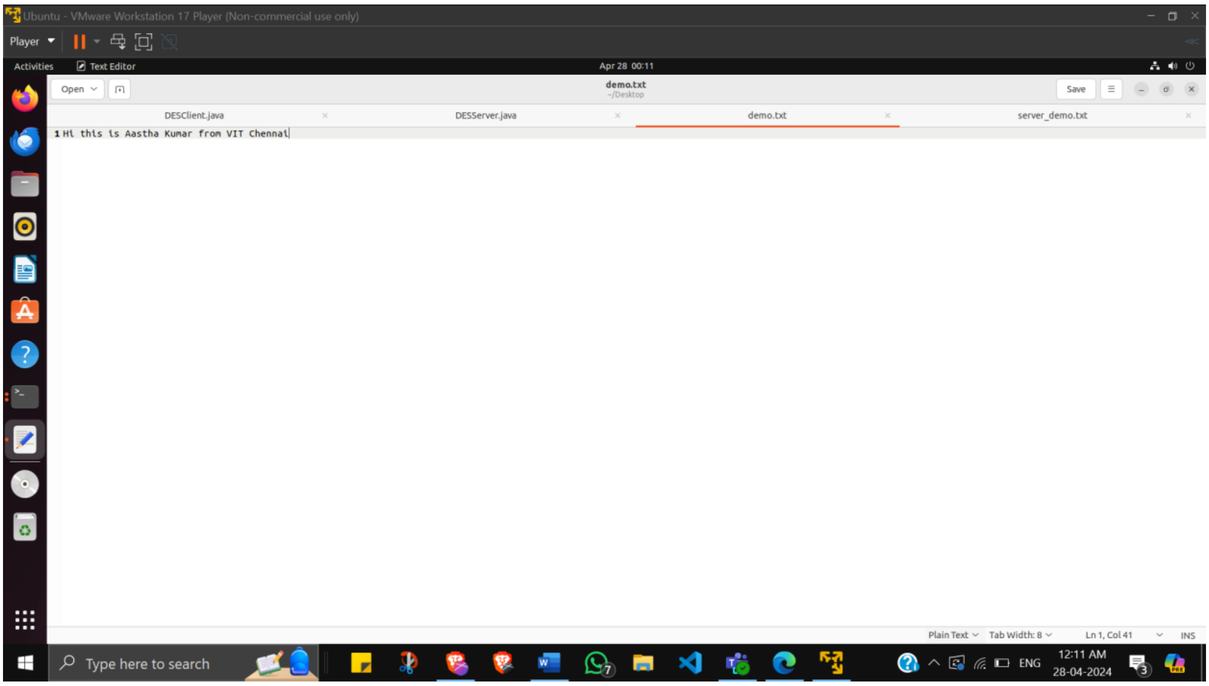
OUTPUT:



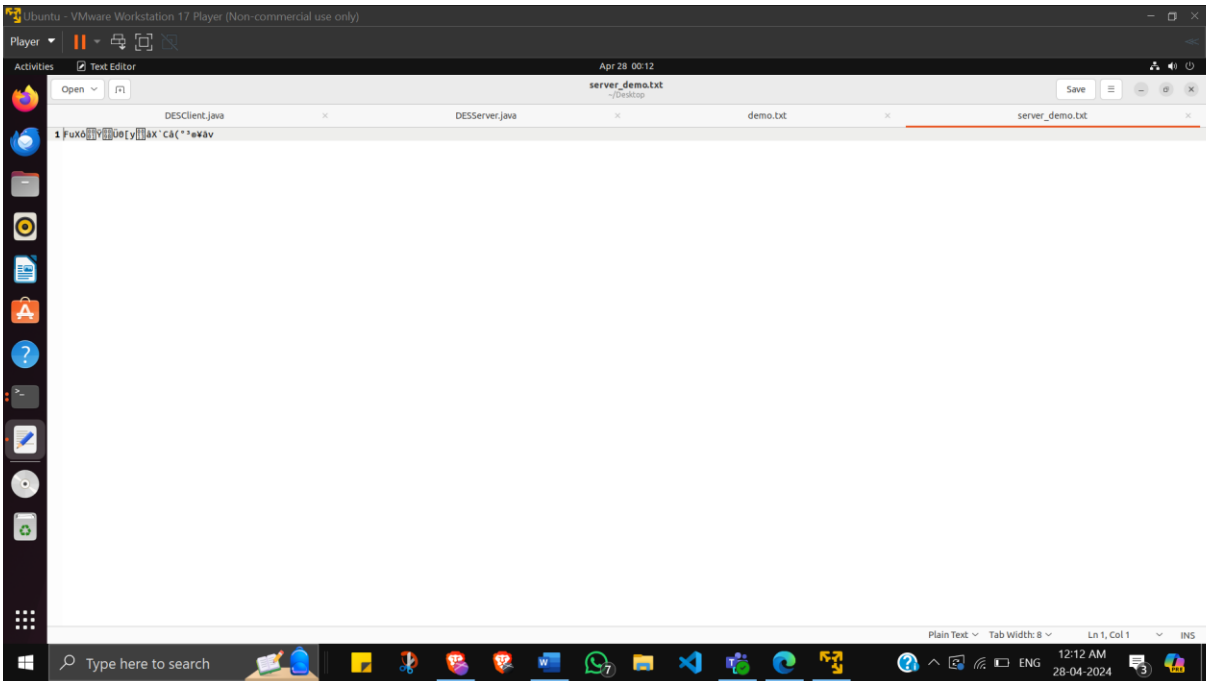
Ubuntu - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Open
DESClient.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac DESClient.java
aastha@aastha-ubuntu:~/Desktop$ java DESClient
Enter the path of the file to encrypt: aastha.txt
Connected to server...
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac DESServer.java
aastha@aastha-ubuntu:~/Desktop$ java DESServer
Server started at port 12345...
Client connected...
Original message: aastha.txt
Encrypted message (hex): 0A49B37F657CBE49CF0E106F224B162C
```



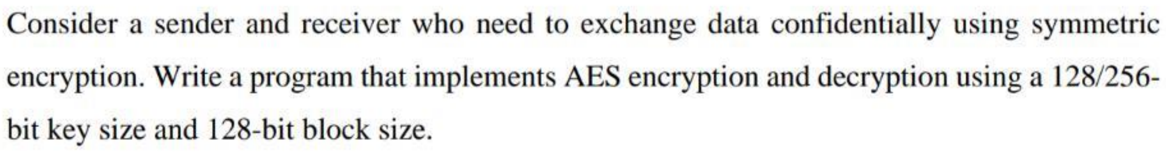
```
Hi this is Aastha Kumar from VIT Chennai
```



```
FuXô□Ÿ□Üθ[y□àX`Cå(°³●¥åv
```

Experiment 4 28/01/24

TITLE:



Consider a sender and receiver who need to exchange data confidentially using symmetric encryption. Write a program that implements AES encryption and decryption using a 128/256-bit key size and 128-bit block size.

ALGORITHM:

Server Side Algorithm:

1. Server Initialization:

   - Create a `ServerSocket` on port 9003.
   - Use an infinite loop to accept client connections.
   - Spawn a new `RealEchoHandler` thread for each incoming client.

2. RealEchoHandler Thread (Server Worker):

   - Initialize an AES key (`key1`).
   - Convert the key to a byte array (`key2`).
   - Create a `SecretKeySpec` for AES encryption using the key.
   - Define a message (`msg`) to be encrypted.
   - Initialize a cipher (`Cipher`) for AES encryption.
   - Encrypt the message using the AES key and cipher.
   - Set up input and output streams for communication with the client.

3. Communication Loop:

   - Send a connection message to the client.
   - Use a loop to interact with the client:
     - Prompt the client with `">"`.
     - Read the client's input.
     - Encrypt the predefined message using AES.
     - Send the encrypted message to the client.
     - Continue the loop until the client terminates the connection.

Client Side Algorithm:

1. Client Initialization:

  - Create a socket to connect to the server at IP address "127.0.0.1" and port 9003.
  - Set up input and output streams for communication.

2. Communication Loop:

  - Receive and print the initial connection message from the server.
  - Use a loop to interact with the server:
    - Receive and print the server's prompt (`">"`).
    - Read the user's input from the console.
    - Send the user's input to the server.
    - Receive and print the encrypted message from the server.
    - Attempt to decrypt the message using AES.
    - Print the decrypted message.
    - Continue the loop until the server terminates the connection.

CODE:

AESServer.java

```
import java.io.*;

import java.net.*;

import java.security.*;

import javax.crypto.*;

import javax.crypto.spec.*;


public class AESServer{

public static void main(String[] args ){

int i = 1;

try{
```

```
ServerSocket s = new ServerSocket(9003);


for (;;){

Socket incoming = s.accept( );

System.out.println("Spawning " + i);

new RealEchoHandler(incoming, i).start();

i++;

}

} catch (Exception e){ System.out.println(e); }

}

}


class RealEchoHandler extends Thread{

DataInputStream in;

DataOutputStream out;

private Socket incoming;

private int counter;


public RealEchoHandler(Socket i, int c){

incoming = i;

counter = c;

}


public void run(){

try {
```

```
String key1 = "1234567812345678";

byte[] key2 = key1.getBytes();

SecretKeySpec secret = new SecretKeySpec(key2, "AES");

String msg = "Singapore Malaysia Japan India Indonesia HongKong Taiwan China England";

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT_MODE, secret);

byte[] encrypted = cipher.doFinal(msg.getBytes());


in = new DataInputStream(incoming.getInputStream());

out = new DataOutputStream(incoming.getOutputStream());


boolean done = false;

String str="";

out.writeUTF("Connected!\n");

out.flush();

while (!done){

out.writeUTF(">");

out.flush();

str = in.readUTF();

System.out.println(in+":"+str);

if (str == null)

done = true;

else{

System.out.println("Sending Ciphertext : " + new String(encrypted)); out.writeUTF(new
String(encrypted)); out.flush();

}

}
```

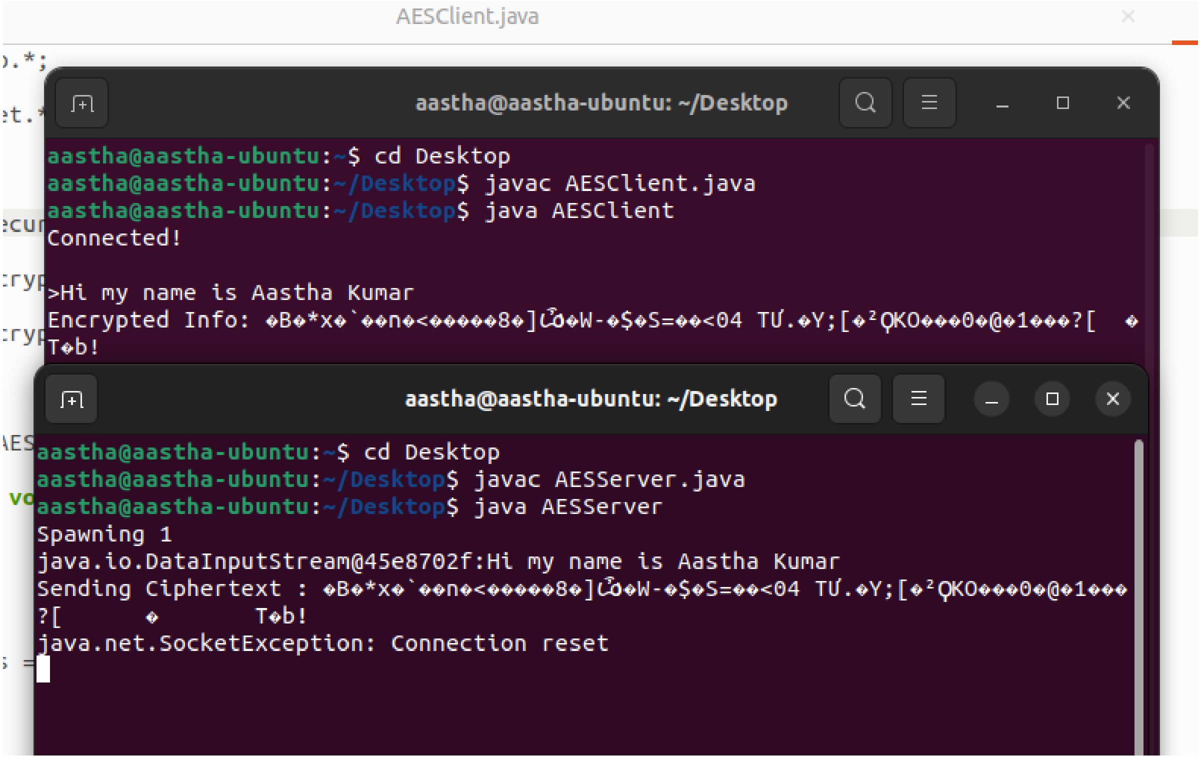
```
incoming.close();

} catch (Exception e){ System.out.println(e);

}

}

}
```

Client.java

```
import java.io.*;

import java.net.*;

import java.security.*;

import javax.crypto.*;

import javax.crypto.spec.*;

import java.util.*;


class AESClient{

public static void main(String[] args) throws NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, IllegalBlockSizeException, BadPaddingException{



String str = "";

String str2 = "";

DataOutputStream out;

DataInputStream in;



try {

Socket t = new Socket("127.0.0.1", 9003);
in = new DataInputStream(t.getInputStream());

out = new DataOutputStream(t.getOutputStream());

BufferedReader br = new BufferedReader (new InputStreamReader(System.in));
```

```
boolean more = true;

System.out.println(in.readUTF());



while (more) {

str = in.readUTF();

System.out.print(str);

str2 = br.readLine();

out.writeUTF(str2);

out.flush();

str = in.readUTF();
System.out.println("Encrypted Info: " + str);



try {



String key1 = "1234567812345678";

byte[] key2 = key1.getBytes();

SecretKeySpec secret = new SecretKeySpec(key2, "AES");



Cipher cipher = Cipher.getInstance("AES");



cipher.init(Cipher.DECRYPT_MODE, secret); byte[] decrypted = cipher.doFinal(str.getBytes());
System.out.println("Decrypted Info: " + new String(decrypted)); }

catch(BadPaddingException e){

System.out.println("Wrong Key!");
}

catch(InvalidKeyException f) {

System.out.println("Invalid Key!");

}
}

}

catch(IOException e){

System.out.println("Error");

}

}

}
```

OUTPUT:



AESClient.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac AESClient.java
aastha@aastha-ubuntu:~/Desktop$ java AESClient
Connected!

>Hi my name is Aastha Kumar
Encrypted Info: �B�*x�`��n�<������8�]Ꮯ�W-�$�S=��<04 TƯ.�Y;[�²ϘKO���0�@�1���?[  �
T�b!
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac AESServer.java
aastha@aastha-ubuntu:~/Desktop$ java AESServer
Spawning 1
java.io.DataInputStream@45e8702f:Hi my name is Aastha Kumar
Sending Ciphertext : �B�*x�`��n�<������8�]Ꮯ�W-�$�S=��<04 TƯ.�Y;[�²ϘKO���0�@�1���
?[      �       T�b!
java.net.SocketException: Connection reset
```

Experiment 6 04/02/24

AIM:

Develop a cipher scheme using RSA algorithm.

ALGORITHM:

Client:

1. Connect to the server using a Socket.
2. Generate the RSA keys:
   - Choose two prime numbers p and q.
   - Compute n = p * q.
   - Compute phi = (p - 1) * (q - 1).
   - Choose an integer e such that 1 < e < phi, and e is co-prime to phi.
   - Compute d such that (d * e) % phi = 1.
3. Encrypt the message:
   - Choose a message to be encrypted, represented as an integer msg.
   - Compute c = (msg ^ e) % n. This is the encrypted message.
4. Send p, q, and c to the server using a DataOutputStream.

Server:

1. Start a ServerSocket to accept connections.
2. Accept a connection from a client using serverSocket.accept().
3. Receive p, q, and c from the client using a DataInputStream.
4. Print the received values of p, q, and c.
5. Decrypt the message:
   - Compute n = p * q.
   - Compute phi = (p - 1) * (q - 1).
   - Choose an integer e such that 1 < e < phi, and e is co-prime to phi.
   - Compute d such that (d * e) % phi = 1.
   - Compute m = (c ^ d) % n. This is the decrypted message.
6. Print the decrypted message.

CODE:

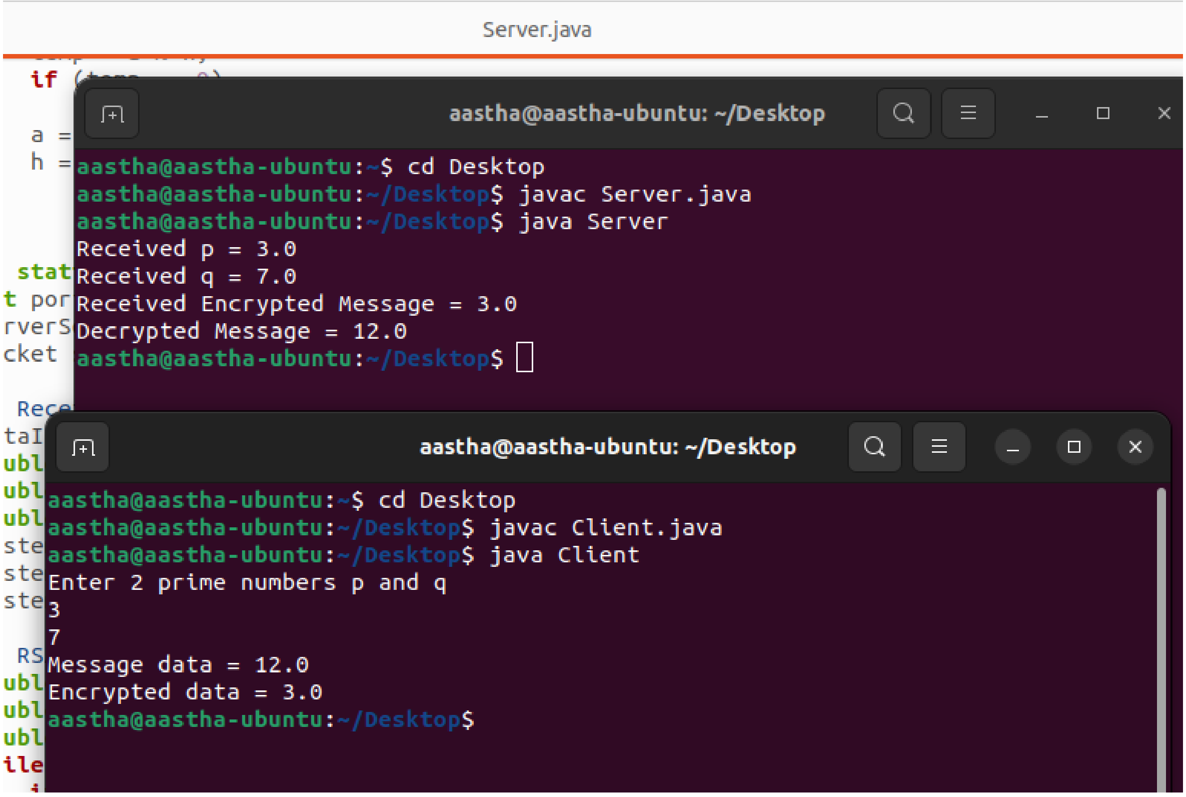
Client.java

```
import java.io.*;
import java.net.*;
import java.math.*;
import java.util.*;
import java.security.*;

public class Client {
   public static double gcd(double a, double h) {
      double temp;
      while (true) {
          temp = a % h;
          if (temp == 0)
              return h;
          a = h;
          h = temp;
      }
   }

   public static void main(String[] args) throws Exception {
   Scanner sc=new Scanner(System.in);
      String host = "localhost";
      int port = 8000;
      Socket socket = new Socket(host, port);

      // RSA Key Generation
      System.out.println("Enter 2 prime numbers p and q");
      double p = sc.nextInt();
      double q = sc.nextInt();
      double n = p * q;
      double e = 2;
      double phi = (p - 1) * (q - 1);
      while (e < phi) {
          if (gcd(e, phi) == 1)
              break;
          else
              e++;
      }
      int k = 2;
      double d = (1 + (k * phi)) / e;

      // Message to be encrypted
      double msg = 12;
      System.out.println("Message data = " + msg);

      // Encrypt message
      double c = Math.pow(msg, e);
      c = c % n;
      System.out.println("Encrypted data = " + c);

      // Send p, q, and encrypted message
      DataOutputStream dataOut = new DataOutputStream(socket.getOutputStream());
      dataOut.writeDouble(p);
      dataOut.writeDouble(q);
      dataOut.writeDouble(c);

      socket.close();
   }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.math.*;
import java.util.*;
import java.security.*;

public class Server {
    public static double gcd(double a, double h) {
        double temp;
        while (true) {
            temp = a % h;
            if (temp == 0)
                return h;
            a = h;
            h = temp;
        }
    }

    public static void main(String[] args) throws Exception {
        int port = 8000;
        ServerSocket serverSocket = new ServerSocket(port);
        Socket socket = serverSocket.accept();

        // Receive p, q, and encrypted message
        DataInputStream dataIn = new DataInputStream(socket.getInputStream());
        double p = dataIn.readDouble();
        double q = dataIn.readDouble();
        double c = dataIn.readDouble();
        System.out.println("Received p = " + p);
        System.out.println("Received q = " + q);
        System.out.println("Received Encrypted Message = " + c);

        // RSA Key Generation
        double n = p * q;
        double e = 2;
        double phi = (p - 1) * (q - 1);
        while (e < phi) {
            if (gcd(e, phi) == 1)
                break;
            else
                e++;
        }
        int k = 2;
        double d = (1 + (k * phi)) / e;

        // Decrypt message
        double m = Math.pow(c, d);
        m = m % n;
        System.out.println("Decrypted Message = " + m);

        socket.close();
        serverSocket.close();
    }}
```

OUPTUT:



```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Received p = 3.0
Received q = 7.0
Received Encrypted Message = 3.0
Decrypted Message = 12.0
aastha@aastha-ubuntu:~/Desktop$ 
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Enter 2 prime numbers p and q
3
7
Message data = 12.0
Encrypted data = 3.0
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 6 18/02/24

TITLE:

Develop a cipher scheme using Elgamal public key cryptographic algorithm.

ALGORITHM:

Client:

1. Connect to the server using a Socket.
2. Generate the ElGamal keys:
   - Choose two prime numbers p and q.
   - Compute g as a primitive root modulo p.
   - Choose a random integer d such that $1 < d < p-1$.
   - Compute $y = g^d \bmod p$.
3. Encrypt the message:
   - Choose a message to be encrypted, represented as an integer m.
   - Choose a random integer k such that $1 < k < p-1$.
   - Compute $a = g^k \bmod p$ and $b = y^k * m \bmod p$. The pair (a, b) is the encrypted message.
4. Send p, g, d, m, and the encrypted message (a, b) to the server using an ObjectOutputStream.

Server:

1. Start a ServerSocket to accept connections.
2. Accept a connection from a client using serverSocket.accept().
3. Receive p, g, d, m, and the encrypted message (a, b) from the client using an ObjectInputStream.
4. Print the received values of p, g, d, m, and the encrypted message (a, b).
5. Decrypt the message:
   - Compute $s = a^d \bmod p$.
   - Compute the original message $m' = b * s^{-1} \bmod p$.
6. Print the decrypted message m'.

<u>CODE:</u>

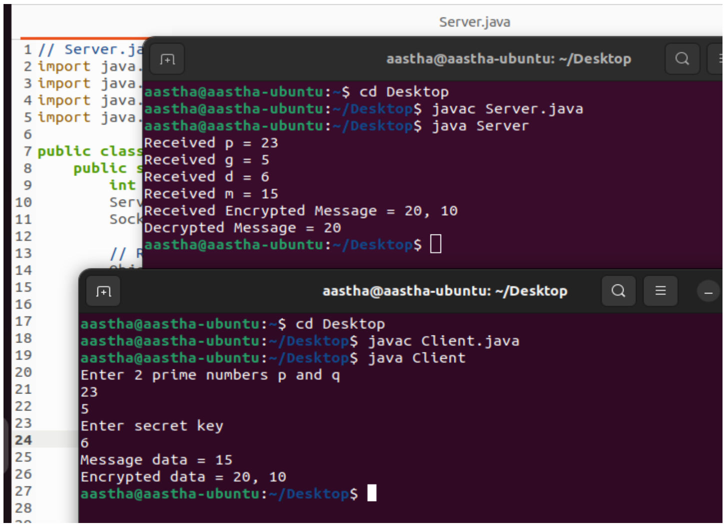
Client.java

```
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.SecureRandom;
import java.util.*;

public class Client {
    private static BigInteger computeY(BigInteger p, BigInteger g, BigInteger d) {
        return g.modPow(d, p);}

    private static BigInteger[] encrypt(BigInteger m, BigInteger p, BigInteger g, BigInteger y) {
        BigInteger k = new BigInteger(p.bitLength(), new SecureRandom());
        BigInteger a = g.modPow(k, p);
        BigInteger b = y.modPow(k, p).multiply(m).mod(p);
        return new BigInteger[]{a, b};}

    public static void main(String[] args) throws IOException {
    Scanner sc=new Scanner(System.in);
        String host = "localhost";
        int port = 8000;
        Socket socket = new Socket(host, port);

        // ElGamal Key Generation
        System.out.println("Enter 2 prime numbers p and q");
        BigInteger p = sc.nextBigInteger();
        BigInteger g = sc.nextBigInteger();
        System.out.println("Enter secret key");
        BigInteger d = sc.nextBigInteger();
        BigInteger y = computeY(p, g, d);
        // Message to be encrypted
        BigInteger m = new BigInteger("15");
        System.out.println("Message data = " + m);

        // Encrypt message
        BigInteger[] encrypted = encrypt(m, p, g, y);
        System.out.println("Encrypted data = " + encrypted[0] + ", " + encrypted[1]);

        // Send p, g, d, m, and encrypted message to server
        ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
        oos.writeObject(new BigInteger[]{p, g, d, m});
        oos.writeObject(encrypted);

        socket.close();}}
```

Server.java

```
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.SecureRandom;

public class Server {
    public static void main(String[] args) throws IOException {
        int port = 8000;
        ServerSocket serverSocket = new ServerSocket(port);
        Socket socket = serverSocket.accept();

        // Receive p, g, d, m, and encrypted message from client
        ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
        try {
            BigInteger[] keyAndMessage = (BigInteger[]) ois.readObject();
            BigInteger[] encrypted = (BigInteger[]) ois.readObject();
            System.out.println("Received p = " + keyAndMessage[0]);
            System.out.println("Received g = " + keyAndMessage[1]);
            System.out.println("Received d = " + keyAndMessage[2]);
            System.out.println("Received m = " + keyAndMessage[3]);
            System.out.println("Received Encrypted Message = " + encrypted[0] + ", " + encrypted[1]);

            // Decrypt message
            BigInteger y = computeY(keyAndMessage[0], keyAndMessage[1], keyAndMessage[2]);
            BigInteger decrypted = encrypted[0].modPow(keyAndMessage[2], keyAndMessage[0]).multiply(encrypted[1].modInverse(keyAndMessage[0])).mod(keyAndMessage[0]);
            System.out.println("Decrypted Message = " + decrypted);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();}

        socket.close();
        serverSocket.close();}
    private static BigInteger computeY(BigInteger p, BigInteger g, BigInteger d) {
        return g.modPow(d, p);}}
```

OUTPUT:


Server.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
// Server.ja
import java.
import java.
import java.
import java.

public class
    public s
        int
        Serv
        Sock

        // R
        obj
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Received p = 23
Received g = 5
Received d = 6
Received m = 15
Received Encrypted Message = 20, 10
Decrypted Message = 20
aastha@aastha-ubuntu:~/Desktop$
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Enter 2 prime numbers p and q
23
5
Enter secret key
6
Message data = 15
Encrypted data = 20, 10
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 7 25/02/24

TITLE:

Develop a cipher scheme using Elliptic Curve Cryptography (ECC)

ALGORITHM :

Server:

1. The server creates a ServerSocket that listens on port 8080.
2. The server waits for a client to connect.
3. Once a client connects, the server generates an elliptic curve key pair using the secp256r1 curve.
4. The server retrieves the public and private keys from the key pair.
5. The server sends the public key to the client.
6. The server waits for the client to send its public key.
7. Once the server receives the client's public key, it performs the ECDH key agreement protocol using its private key and the client's public key to generate a shared secret.
8. The server prints the shared secret.

Client:

1. The client creates a Socket that connects to the server on port 8080.
2. The client generates an elliptic curve key pair using the secp256r1 curve.
3. The client retrieves the public and private keys from the key pair.
4. The client sends the public key to the server.
5. The client waits for the server to send its public key.
6. Once the client receives the server's public key, it performs the ECDH key agreement protocol using its private key and the server's public key to generate a shared secret.
7. The client prints the shared secret.

CODE :

Client.java

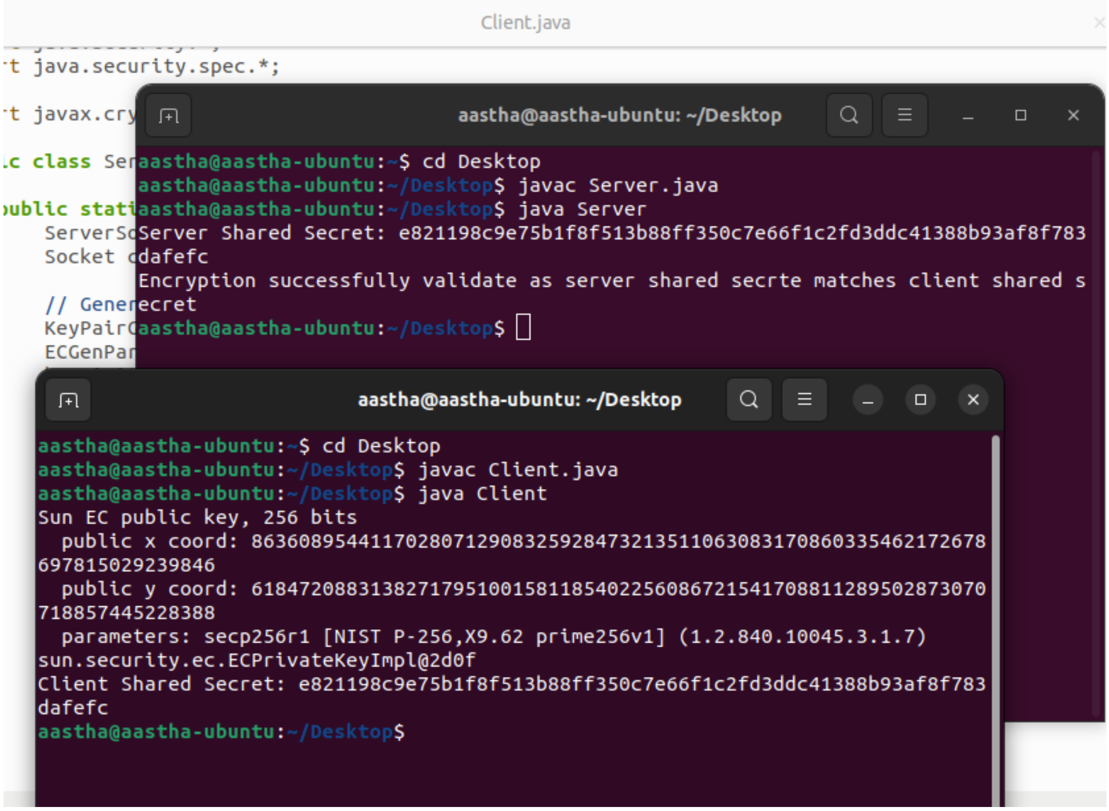
```
import java.io.*;
import java.net.*;
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;

import javax.crypto.KeyAgreement;
```

```
public class Client {

    public static void main(String args[]) throws Exception {
        Socket socket = new Socket("localhost", 8080);  // Connect to server on port 8080

        // Generate an EC key pair (choose the same curve as server)
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC");
        ECGenParameterSpec spec = new ECGenParameterSpec("secp256r1"); // Example curve name (match server's curve)
        kpg.initialize(spec);
        KeyPair keyPair = kpg.genKeyPair();

        // Get public and private keys
        PublicKey publicKey = keyPair.getPublic();
        System.out.println(publicKey);
        PrivateKey privateKey = keyPair.getPrivate();
        System.out.println(privateKey);

        // Send the public key to the server
        ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
        oos.writeObject(publicKey);

        // Receive the server's public key
        ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
        PublicKey serverPublicKey = (PublicKey) ois.readObject();

        // Perform ECDH key derivation
        KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");
        keyAgreement.init(privateKey);
        keyAgreement.doPhase(serverPublicKey, true); // true for private key

        // Generate the shared secret
        byte[] sharedSecret = keyAgreement.generateSecret();

        System.out.println("Client Shared Secret: " + new BigInteger(1, sharedSecret).toString(16));

        oos.close();
        ois.close();
        socket.close();
    }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;

import javax.crypto.KeyAgreement;

public class Server {

    public static void main(String args[]) throws Exception {
        ServerSocket serverSocket = new ServerSocket(8080); // Listen on port 8080
        Socket clientSocket = serverSocket.accept();  // Wait for a client connection

        // Generate an EC key pair (choose a curve like secp256r1)
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC");
        ECGenParameterSpec spec = new ECGenParameterSpec("secp256r1"); // Example curve
name
        kpg.initialize(spec);
        KeyPair keyPair = kpg.genKeyPair();

        // Get public and private keys
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // Send the public key to the client
        ObjectOutputStream oos = new ObjectOutputStream(clientSocket.getOutputStream());
        oos.writeObject(publicKey);

        // Receive the client's public key
        ObjectInputStream ois = new ObjectInputStream(clientSocket.getInputStream());
        PublicKey clientPublicKey = (PublicKey) ois.readObject();

        // Perform ECDH key derivation
        KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");
        keyAgreement.init(privateKey);
        keyAgreement.doPhase(clientPublicKey, true); // true for private key

        // Generate the shared secret
        byte[] sharedSecret = keyAgreement.generateSecret();

        System.out.println("Server Shared Secret: " + new BigInteger(1,
sharedSecret).toString(16));
        System.out.println("Encryption successfully validate as server shared secrte matches client
shared secret");

        oos.close();
        ois.close();
        clientSocket.close();
        serverSocket.close();
    }
}
```

OUTPUT :


Client.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
t java.security.spec.*;

t javax.cry

c class Ser

ublic stati
    ServerSc
    Socket c

    // Gener
    KeyPairG
    ECGenPar
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Server Shared Secret: e821198c9e75b1f8f513b88ff350c7e66f1c2fd3ddc41388b93af8f783
dafefc
Encryption successfully validate as server shared secrte matches client shared s
ecret
aastha@aastha-ubuntu:~/Desktop$ 
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Sun EC public key, 256 bits
  public x coord: 863608954411702807129083259284732135110630831708603354621726 78
697815029239846
  public y coord: 618472088313827179510015811854022560867215417088112895028730 70
718857445228388
  parameters: secp256r1 [NIST P-256,X9.62 prime256v1] (1.2.840.10045.3.1.7)
sun.security.ec.ECPrivateKeyImpl@2d0f
Client Shared Secret: e821198c9e75b1f8f513b88ff350c7e66f1c2fd3ddc41388b93af8f783
dafefc
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 8 03/03/24

TITLE:

Design a Diffie Hellman multi-party key exchange protocol and perform man in the middle attack.

ALGORITHM:

Server.java:

1. Import necessary libraries.
2. Define a method to calculate the shared key.
3. Define a method to generate a random private key.
4. In the main method:
    - Create a server socket and accept client connections.
    - Create data input and output streams.
    - Read the public key, modulus, and client's public key from the client.
    - Generate a private key.
    - Calculate the server's public key and send it to the client.
    - Calculate the shared key.
    - Print the shared key.
    - Close the socket and server socket.

Client.java:

1. Import necessary libraries.
2. Define a method to calculate the shared key.
3. Define a method to generate a random private key.
4. In the main method:
    - Create a socket to connect to the server.
    - Create data input and output streams.
    - Read the prime number p and another prime number g from the user.
    - Generate a private key.
    - Calculate the client's public key.
    - Send p, g, and the client's public key to the server.
    - Read the server's public key from the server.
    - Calculate the shared key.
    - Print the shared key.

<u>CODE:</u>

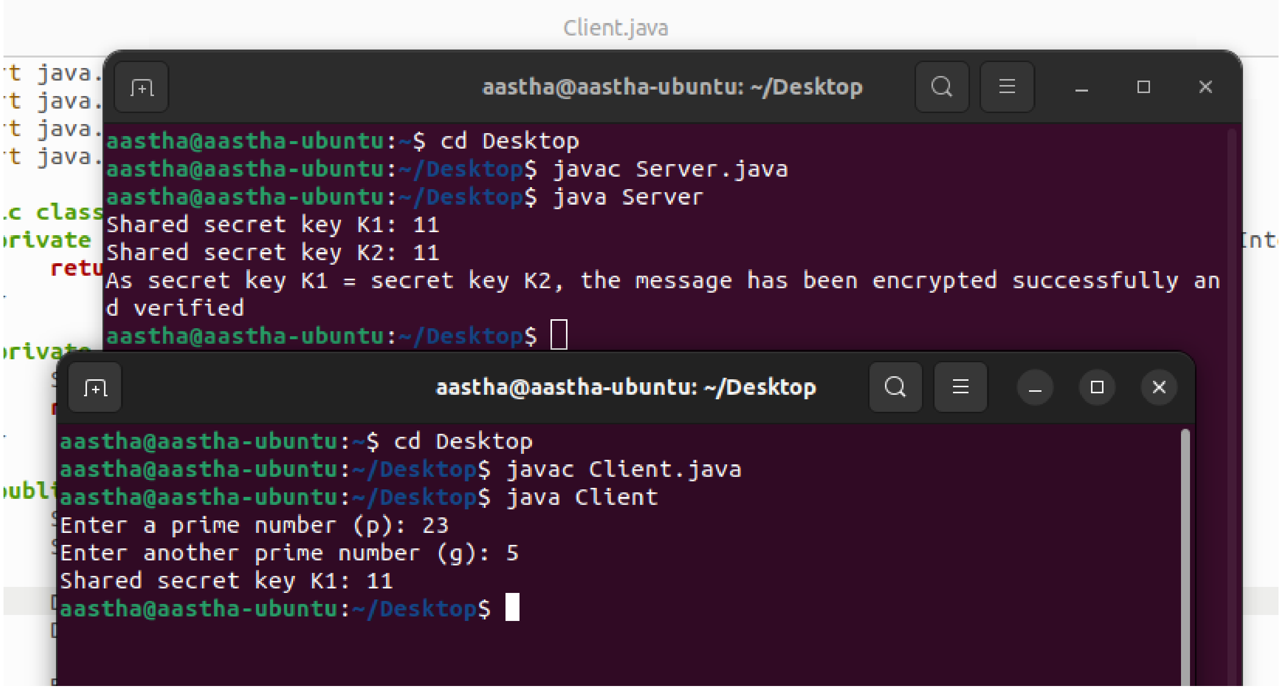
Client.java

```
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.SecureRandom;
import java.util.Scanner;

public class Client {
    private static BigInteger calculateKey(BigInteger receivedKey, BigInteger privateKey,
BigInteger modulus) {
        return receivedKey.modPow(privateKey, modulus);
    }

    private static BigInteger generateRandomKey(BigInteger p) {
        SecureRandom random = new SecureRandom();
        return new BigInteger(p.bitLength(), random).mod(p.subtract(BigInteger.ONE));
    }

    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a prime number (p): ");
        BigInteger p = scanner.nextBigInteger();
        System.out.print("Enter another prime number (g): ");
        BigInteger g = scanner.nextBigInteger();

        BigInteger privateKey = generateRandomKey(p);
        BigInteger clientPublicKey = g.modPow(privateKey, p);

        dataOutputStream.writeUTF(p.toString());
        dataOutputStream.writeUTF(g.toString());
        dataOutputStream.writeUTF(clientPublicKey.toString());

        BigInteger serverPublicKey = new BigInteger(dataInputStream.readUTF());

        BigInteger sharedKey = calculateKey(serverPublicKey, privateKey, p);

        System.out.println("Shared secret key K1: " + sharedKey);

        socket.close();
    }
}
```

Server.java

```
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.SecureRandom;

public class Server {
   private static BigInteger calculateKey(BigInteger receivedKey, BigInteger privateKey,
BigInteger modulus) {
      return receivedKey.modPow(privateKey, modulus);
   }

   private static BigInteger generateRandomKey(BigInteger p) {
      SecureRandom random = new SecureRandom();
      return new BigInteger(p.bitLength(), random).mod(p.subtract(BigInteger.ONE));
   }

   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket = new ServerSocket(5000);
      Socket socket = serverSocket.accept();

      DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
      DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());

      BigInteger p = new BigInteger(dataInputStream.readUTF());
      BigInteger g = new BigInteger(dataInputStream.readUTF());
      BigInteger clientPublicKey = new BigInteger(dataInputStream.readUTF());

      BigInteger privateKey = generateRandomKey(p);
      BigInteger serverPublicKey = g.modPow(privateKey, p);
      dataOutputStream.writeUTF(serverPublicKey.toString());

      BigInteger sharedKey = calculateKey(clientPublicKey, privateKey, p);

      System.out.println("Shared secret key K1: " + sharedKey);
         System.out.println("Shared secret key K2: " + sharedKey);
         System.out.println("As secret key K1 = secret key K2, the message has been encrypted
successfully and verified");
      socket.close();
      serverSocket.close();
   }
}
```

OUTPUT:


Client.java
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Shared secret key K1: 11
Shared secret key K2: 11
As secret key K1 = secret key K2, the message has been encrypted successfully an
d verified
aastha@aastha-ubuntu:~/Desktop$ 
```

aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Enter a prime number (p): 23
Enter another prime number (g): 5
Shared secret key K1: 11
aastha@aastha-ubuntu:~/Desktop$ 
```

MAN IN THE MIDDLE ATTACK:

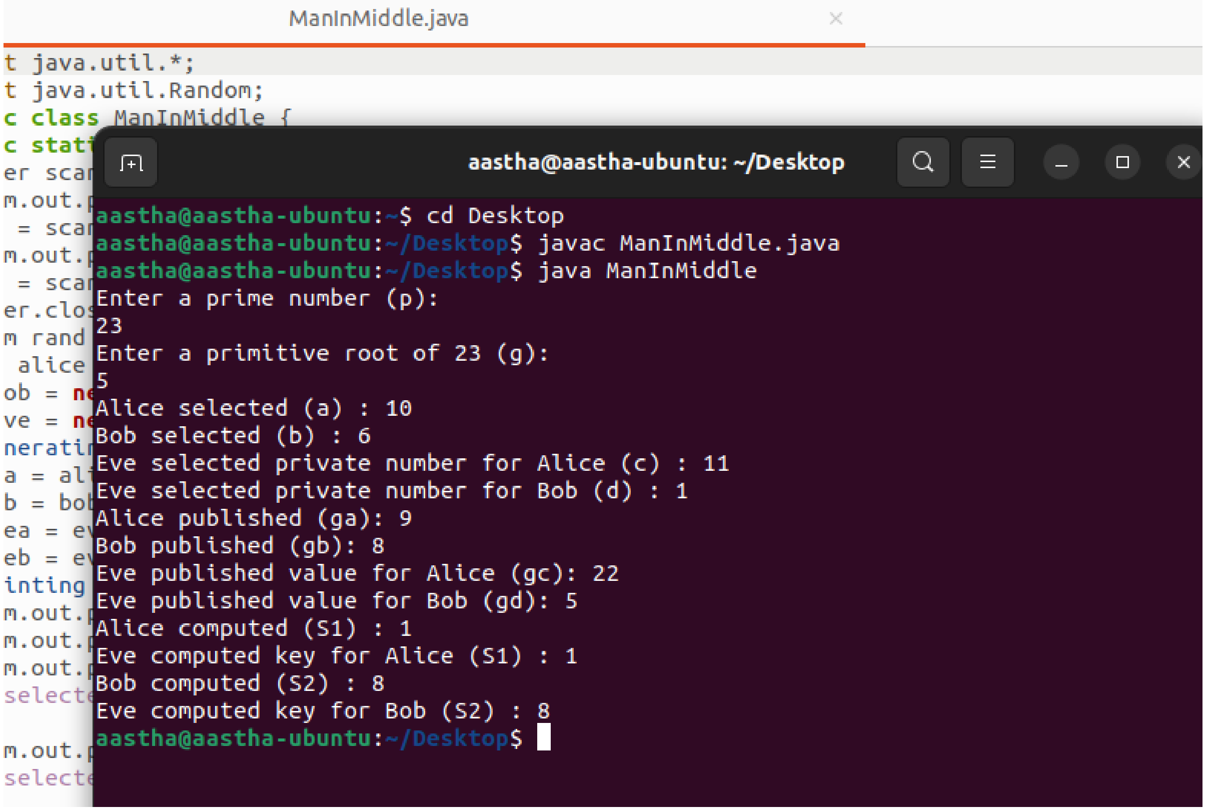
CODE:

```
import java.util.*;
import java.util.Random;
public class ManInMiddle {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("Enter a prime number (p):");
int p = scanner.nextInt();
System.out.println("Enter a primitive root of " + p + " (g):");
int g = scanner.nextInt();
scanner.close();
Random rand = new Random();
Alice alice = new Alice(p, g, rand);
Bob bob = new Bob(p, g, rand);
Eve eve = new Eve(p, g, rand);
// Generating public values
int ga = alice.publish();
int gb = bob.publish();
int gea = eve.publish(0);
int geb = eve.publish(1);
// Printing out the private selected number by Alice and Bob
System.out.println("Alice selected (a) : " + alice.getN());
System.out.println("Bob selected (b) : " + bob.getN());
System.out.println(
"Eve selected private number for Alice (c) : " + eve.getA()
);
System.out.println(
"Eve selected private number for Bob (d) : " + eve.getB()
);
// Printing out the public values
System.out.println("Alice published (ga): " + ga);
System.out.println("Bob published (gb): " + gb);
System.out.println("Eve published value for Alice (gc): " + gea);
System.out.println("Eve published value for Bob (gd): " + geb);
// Computing the secret key
int sa = alice.computeSecret(gea);
int sea = eve.computeSecret(ga, 0);
int sb = bob.computeSecret(geb);
int seb = eve.computeSecret(gb, 1);
System.out.println("Alice computed (S1) : " + sa);
System.out.println("Eve computed key for Alice (S1) : " + sea);
System.out.println("Bob computed (S2) : " + sb);
System.out.println("Eve computed key for Bob (S2) : " + seb);
}
}
class Alice {
private int p;
private int g;
private int n;
```

```
public Alice(int p, int g, Random rand) {
this.p = p;
this.g = g;
this.n = rand.nextInt(p - 1) + 1;
}
public int getN() {
return n;
}
public int publish() {
return (int) (Math.pow(g, n) % p);
}
public int computeSecret(int gb) {
return (int) (Math.pow(gb, n) % p);
}
}
class Bob {
private int p;
private int g;
private int n;
public Bob(int p, int g, Random rand) {
this.p = p;
this.g = g;
this.n = rand.nextInt(p - 1) + 1;
}
public int getN() {
return n;
}
public int publish() {
return (int) (Math.pow(g, n) % p);
}
public int computeSecret(int ga) {
return (int) (Math.pow(ga, n) % p);
}
}
```

```
class Eve {
private int p;
private int g;
private int a;
private int b;
public Eve(int p, int g, Random rand) {
this.p = p;
this.g = g;
this.a = rand.nextInt(p - 1) + 1;
this.b = rand.nextInt(p - 1) + 1;
}
public int getA() {
return a;
}
public int getB() {
return b;
}
public int publish(int i) {
if (i == 0) {
return (int) (Math.pow(g, a) % p);
} else {
return (int) (Math.pow(g, b) % p);
}
}
public int computeSecret(int gVal, int i) {
if (i == 0) {
return (int) (Math.pow(gVal, a) % p);
} else {
return (int) (Math.pow(gVal, b) % p);
}
}
}
```

OUTPUT:



```
t java.util.*;
t java.util.Random;
c class ManInMiddle {
c stati
er scan
m.out.p
 = scan
m.out.p
 = scan
er.clos
m rand
 alice
ob = ne
ve = ne
neratin
a = ali
b = bob
ea = ev
eb = ev
inting
m.out.p
m.out.p
m.out.p
selecte

m.out.p
selecte
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac ManInMiddle.java
aastha@aastha-ubuntu:~/Desktop$ java ManInMiddle
Enter a prime number (p):
23
Enter a primitive root of 23 (g):
5
Alice selected (a) : 10
Bob selected (b) : 6
Eve selected private number for Alice (c) : 11
Eve selected private number for Bob (d) : 1
Alice published (ga): 9
Bob published (gb): 8
Eve published value for Alice (gc): 22
Eve published value for Bob (gd): 5
Alice computed (S1) : 1
Eve computed key for Alice (S1) : 1
Bob computed (S2) : 8
Eve computed key for Bob (S2) : 8
aastha@aastha-ubuntu:~/Desktop$ 
```

Experiment 9 10/03/24

TITLE:

Demonstrate SHA-512 and print the hash code and final value of all buffers.

ALGORITHM:

Server.java:

1. Import necessary libraries.
2. In the main method:
   - Create a server socket and accept client connections.
   - Create data input and output streams.
   - Read the input string from the client.
   - Calculate the SHA-512 hash of the input string.
   - Send the hash back to the client.
   - Print the input string and hash to the console.
   - Close the socket and server socket.

Client.java:

1. Import necessary libraries.
2. In the main method:
   - Create a socket to connect to the server.
   - Create data input and output streams.
   - Send a string to the server.
   - Read the hash from the server.
   - Print the string and hash to the console.
   - Close the socket.

CODE:

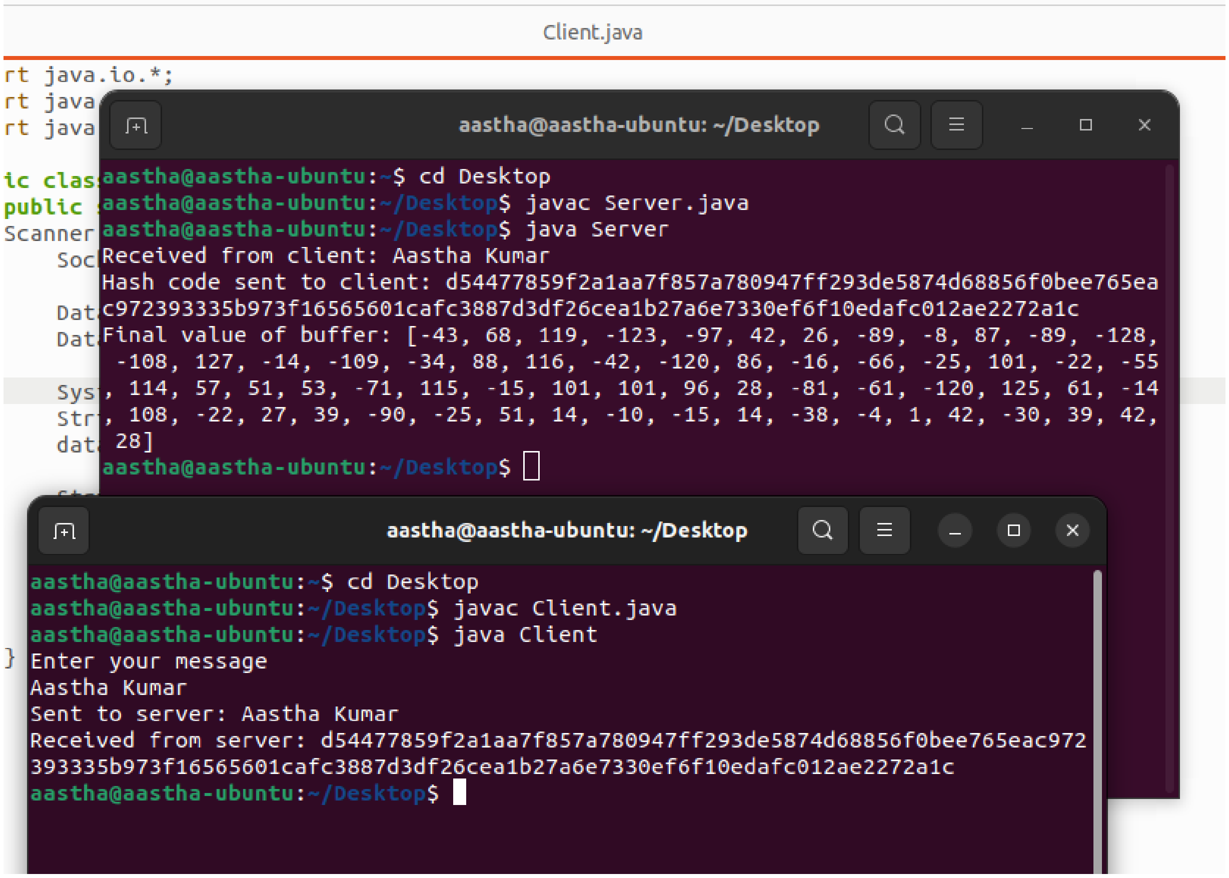
Client.java

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());

        String input = "Hello, world!";
        dataOutputStream.writeUTF(input);

        String output = dataInputStream.readUTF();

        System.out.println("Sent to server: " + input);
        System.out.println("Received from server: " + output);

        socket.close();
    }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.math.BigInteger;

public class Server {
    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        ServerSocket serverSocket = new ServerSocket(5000);
        Socket socket = serverSocket.accept();

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());

        String input = dataInputStream.readUTF();

        MessageDigest md = MessageDigest.getInstance("SHA-512");
        byte[] messageDigest = md.digest(input.getBytes());
        BigInteger no = new BigInteger(1, messageDigest);
        String hashtext = no.toString(16);

        dataOutputStream.writeUTF(hashtext);

        System.out.println("Received from client: " + input);
        System.out.println("Hash code sent to client: " + hashtext);
        System.out.println("Final value of buffer: " + Arrays.toString(messageDigest));

        socket.close();
        serverSocket.close();
    }
}
```

OUTPUT:


Client.java
aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Received from client: Aastha Kumar
Hash code sent to client: d54477859f2a1aa7f857a780947ff293de5874d68856f0bee765ea
c972393335b973f16565601cafc3887d3df26cea1b27a6e7330ef6f10edafc012ae2272a1c
Final value of buffer: [-43, 68, 119, -123, -97, 42, 26, -89, -8, 87, -89, -128,
 -108, 127, -14, -109, -34, 88, 116, -42, -120, 86, -16, -66, -25, 101, -22, -55
, 114, 57, 51, 53, -71, 115, -15, 101, 101, 96, 28, -81, -61, -120, 125, 61, -14
, 108, -22, 27, 39, -90, -25, 51, 14, -10, -15, 14, -38, -4, 1, 42, -30, 39, 42,
 28]
aastha@aastha-ubuntu:~/Desktop$ 
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Enter your message
Aastha Kumar
Sent to server: Aastha Kumar
Received from server: d54477859f2a1aa7f857a780947ff293de5874d68856f0bee765eac972
393335b973f16565601cafc3887d3df26cea1b27a6e7330ef6f10edafc012ae2272a1c
aastha@aastha-ubuntu:~/Desktop$ 
```

Experiment 10 15/03/24

TITLE:

Demonstrate MD5 hash algorithm that finds Message authentication code (MAC).

ALGORITHM:

Server.java:

1. Import necessary libraries.
2. In the main method:
   - Create a server socket and accept client connections.
   - Create data input and output streams.
   - Read the input string from the client.
   - Calculate the MD5 hash of the input string.
   - Send the hash back to the client.
   - Print the input string and hash to the console.
   - Close the socket and server socket.

Client.java:

1. Import necessary libraries.
2. In the main method:
   - Create a socket to connect to the server.
   - Create data input and output streams.
   - Send a string to the server.
   - Read the hash from the server.
   - Print the string and hash to the console.
   - Close the socket

CODE :

Client.java

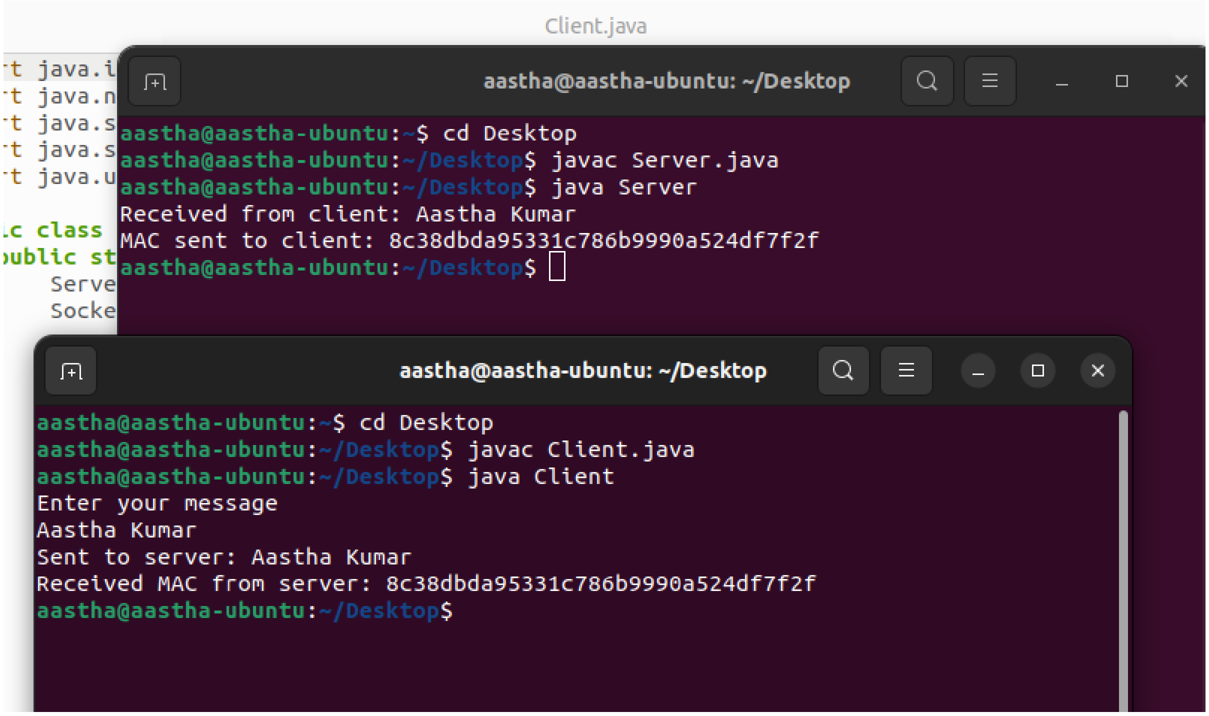
```
import java.io.*;
import java.net.*;
import java.util.*;

public class Client {
   public static void main(String[] args) throws IOException {
   Scanner sc =new Scanner(System.in);
```

```
        Socket socket = new Socket("localhost", 5000);

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());

            System.out.println("Enter your message");
        String input = sc.nextLine();
        dataOutputStream.writeUTF(input);
        String output = dataInputStream.readUTF();
        System.out.println("Sent to server: " + input);
        System.out.println("Received MAC from server: " + output);
        socket.close();
    }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;

public class Server {
    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        ServerSocket serverSocket = new ServerSocket(5000);
        Socket socket = serverSocket.accept();
        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new
DataOutputStream(socket.getOutputStream());
        String input = dataInputStream.readUTF();
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(input.getBytes());
        byte[] digest = md.digest();
        StringBuilder sb = new StringBuilder();
        for (byte b : digest) {
            sb.append(String.format("%02x", b));
        }
        String myHash = sb.toString();
        dataOutputStream.writeUTF(myHash);
        System.out.println("Received from client: " + input);
        System.out.println("MAC sent to client: " + myHash);
        socket.close();
        serverSocket.close();
    }
}
```

<u>OUTPUT :</u>



```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Received from client: Aastha Kumar
MAC sent to client: 8c38dbda95331c786b9990a524df7f2f
aastha@aastha-ubuntu:~/Desktop$
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Enter your message
Aastha Kumar
Sent to server: Aastha Kumar
Received MAC from server: 8c38dbda95331c786b9990a524df7f2f
aastha@aastha-ubuntu:~/Desktop$
```

Experiment 11 07/04/24

TITLE:

Develop DSS for verifying the legal communicating parties.

ALGORITHM:

Server Code:

1. The server starts by creating a ServerSocket on port 5000 and waits for a client to connect.
2. Once a client connects, it creates DataInputStream and DataOutputStream objects for communication.
3. It then generates a key pair (public and private keys) using the Digital Signature Algorithm (DSA).
4. The server asks the user to input a message via the console.
5. The message is then signed using the private key, creating a digital signature.
6. The digital signature and the original message are sent to the client.
7. Finally, the server closes the socket connection.

Client Code:

1. The client creates a Socket connection to the server running on “localhost” and port 5000.
2. It creates DataInputStream and DataOutputStream objects for communication.
3. The client receives the digital signature and the original message from the server.
4. It generates a public key (for simplicity, a new key pair is generated and the public key is used. In a real-world scenario, this public key would be agreed upon beforehand or sent over a secure channel).
5. The client then verifies the digital signature using the public key.
6. The received digital signature and the result of the verification (true if the signature is valid, false otherwise) are printed to the console.
7. Finally, the client closes the socket connection

CODE:

Client.java

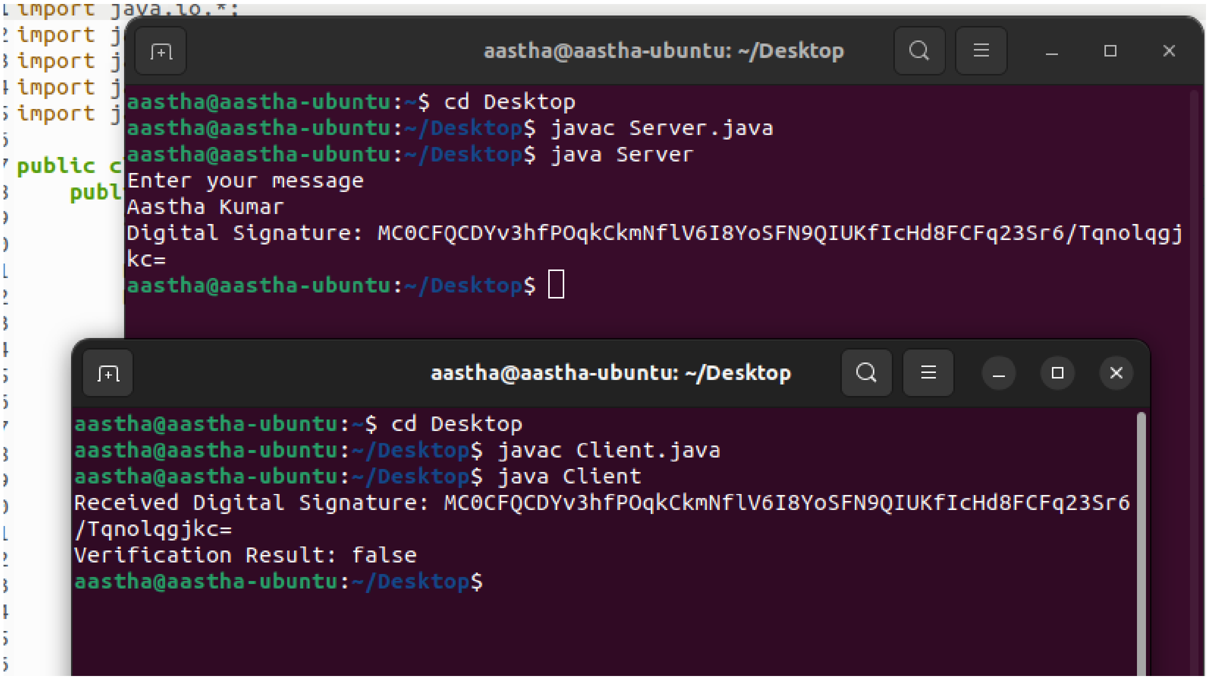
```
import java.io.*;
import java.net.*;
import java.security.*;
import java.security.spec.*;
import java.util.Base64;

public class Client {
  public static void main(String[] args) throws IOException, NoSuchAlgorithmException,
InvalidKeyException, SignatureException, InvalidKeySpecException {
    Socket socket = new Socket("localhost", 5000);

    DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
    DataOutputStream dataOutputStream = new DataOutputStream(socket.getOutputStream());

    // Receive the digital signature and message from the server
    String digitalSignature = dataInputStream.readUTF();
    String message = dataInputStream.readUTF();

    // The public key would typically be agreed upon beforehand or sent over a secure channel
    // For simplicity, we're generating a new key pair and using the public key
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
    keyPairGenerator.initialize(1024);
    KeyPair keyPair = keyPairGenerator.generateKeyPair();
    PublicKey publicKey = keyPair.getPublic();

    // Verify the digital signature using the public key
    Signature signature = Signature.getInstance("SHA256withDSA");
    signature.initVerify(publicKey);
    signature.update(message.getBytes());
    boolean verified = signature.verify(Base64.getDecoder().decode(digitalSignature));
```

```
        // Print the digital signature
        System.out.println("Received Digital Signature: " + digitalSignature);

        // Print the verification result
        System.out.println("Verification Result: " + verified);

        socket.close();
    }
}
```

Server.java

```
import java.io.*;
import java.net.*;
import java.security.*;
import java.security.spec.*;
import java.util.Base64;
import java.util.*;

public class Server {
    public static void main(String[] args) throws IOException, NoSuchAlgorithmException,
InvalidKeyException, SignatureException {
    Scanner sc=new Scanner(System.in);
        ServerSocket serverSocket = new ServerSocket(5000);
        Socket socket = serverSocket.accept();

        DataInputStream dataInputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream dataOutputStream = new DataOutputStream(socket.getOutputStream());

        // Generate key pair (public and private keys)
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
        keyPairGenerator.initialize(1024);
        KeyPair keyPair = keyPairGenerator.generateKeyPair();
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // Create a signature object based on the DSA algorithm
        Signature signature = Signature.getInstance("SHA256withDSA");

        // Generate a message to be signed
        System.out.println("Enter your message");
        String message = sc.nextLine();

        // Sign the message using the private key
        signature.initSign(privateKey);
        signature.update(message.getBytes());
        byte[] digitalSignature = signature.sign();

        // Send the digital signature and message to the client
        dataOutputStream.writeUTF(Base64.getEncoder().encodeToString(digitalSignature));
        dataOutputStream.writeUTF(message);

        // Print the digital signature
        System.out.println("Digital Signature: " +
Base64.getEncoder().encodeToString(digitalSignature));

        socket.close();
        serverSocket.close();
    }
}
```

OUTPUT:


aastha@aastha-ubuntu: ~/Desktop
aastha@aastha-ubuntu: ~/Desktop

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Server.java
aastha@aastha-ubuntu:~/Desktop$ java Server
Enter your message
Aastha Kumar
Digital Signature: MC0CFQCDYv3hfPOqkCkmNflV6I8YoSFN9QIUKfIcHd8FCFq23Sr6/Tqnolqgj
kc=
aastha@aastha-ubuntu:~/Desktop$
```

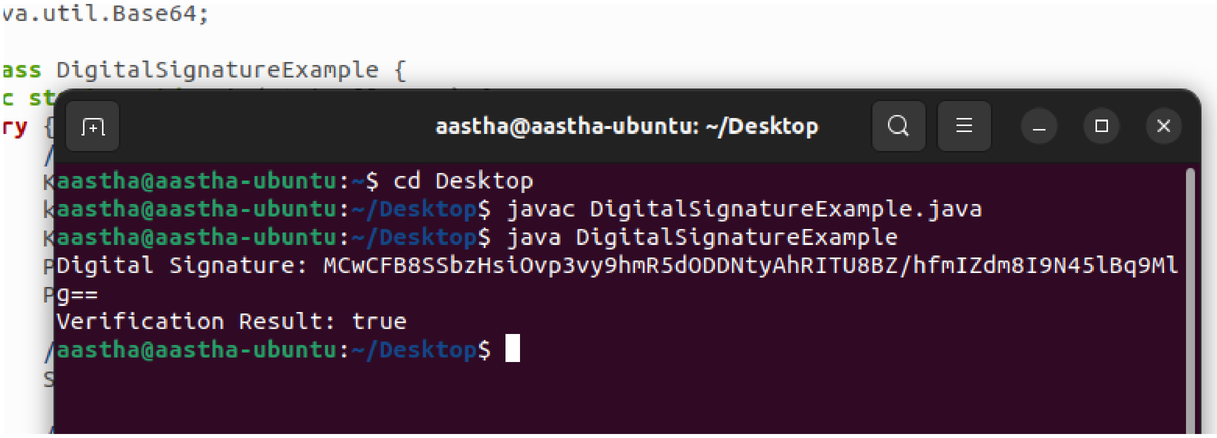
```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac Client.java
aastha@aastha-ubuntu:~/Desktop$ java Client
Received Digital Signature: MC0CFQCDYv3hfPOqkCkmNflV6I8YoSFN9QIUKfIcHd8FCFq23Sr6
/Tqnolqgjkc=
Verification Result: false
aastha@aastha-ubuntu:~/Desktop$
```

CODE:

DigitalSignatureExample.java

```
import java.security.*;
import java.security.spec.*;
import java.util.Base64;

public class DigitalSignatureExample {
    public static void main(String[] args) {
        try {
            // Generate key pair (public and private keys)
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
            keyPairGenerator.initialize(1024);
            KeyPair keyPair = keyPairGenerator.generateKeyPair();
            PublicKey publicKey = keyPair.getPublic();
            PrivateKey privateKey = keyPair.getPrivate();

            // Create a signature object based on the DSA algorithm
            Signature signature = Signature.getInstance("SHA256withDSA");

            // Generate a message to be signed
            String message = "Hello, World!";

            // Sign the message using the private key
            signature.initSign(privateKey);
            signature.update(message.getBytes());
            byte[] digitalSignature = signature.sign();

            // Print the digital signature
            System.out.println("Digital Signature: " +
Base64.getEncoder().encodeToString(digitalSignature));

            // Verify the digital signature using the public key
            signature.initVerify(publicKey);
            signature.update(message.getBytes());
            boolean verified = signature.verify(digitalSignature);

            // Print the verification result
            System.out.println("Verification Result: " + verified);
        } catch (NoSuchAlgorithmException | InvalidKeyException | SignatureException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



```
va.util.Base64;

ass DigitalSignatureExample {
c st
ry {
```

```
aastha@aastha-ubuntu:~$ cd Desktop
aastha@aastha-ubuntu:~/Desktop$ javac DigitalSignatureExample.java
aastha@aastha-ubuntu:~/Desktop$ java DigitalSignatureExample
Digital Signature: MCwCFB8SSbzHsiOvp3vy9hmR5dODDNtyAhRITU8BZ/hfmIZdm8I9N45lBq9Ml
g==
Verification Result: true
aastha@aastha-ubuntu:~/Desktop$ 
```

Experiment 12 14/04/24

AIM:

Develop a simple client and server application using SSL socket communication

ALGORITHM:

Server.java:

1. Load the keystore file (`keystore.jks`) containing the server's private key.
2. Create an SSL context and initialize it with the keystore.
3. Create an SSL server socket factory using the SSL context.
4. Create a server socket and bind it to a specific port.
5. Continuously listen for client connections in a loop.
6. When a client connects, accept the connection and obtain the client socket.
7. Start a new thread to handle client communication.
8. In the client communication thread:
   - Create input and output streams for the client socket.
   - Read messages from the client and print them.
   - Send a response back to the client.
   - Continue this loop until there are no more messages from the client.
   - Close the client socket.

Client.java:

1. Load the truststore file (`truststore.jks`) containing the server's public key.
2. Create an SSL context and initialize it with the truststore.
3. Create an SSL socket factory using the SSL context.
4. Create an SSL socket and connect it to the server.

5. Create input and output streams for the socket.

6. Start a loop to read input from the user:

   - Read a message from the user.

   - Send the message to the server.

   - Read the server's response.

   - Print the server's response.

   - Continue this loop until there are no more messages from the user.

7. Close the input stream, output stream, and socket.

PRE-REQUISITE:

To create a keystore (`keystore.jks`) and truststore (`truststore.jks`) files, you can use the Java `keytool` utility. This utility is included with the Java Development Kit (JDK).

1. Open a terminal or command prompt.

2. Generate a self-signed certificate and store it in the keystore:

```
keytool -genkeypair -alias server -keyalg RSA -keysize 2048 -keystore keystore.jks
```

This command will prompt you to enter various details, such as the keystore password, the certificate's distinguished name, etc. Provide the requested information accordingly.

3. Export the server's certificate from the keystore and import it into the *truststore:*

```
keytool -exportcert -alias server -keystore keystore.jks -file
server.crt
```

```
keytool -importcert -alias server -file server.crt -keystore truststore.jks
```

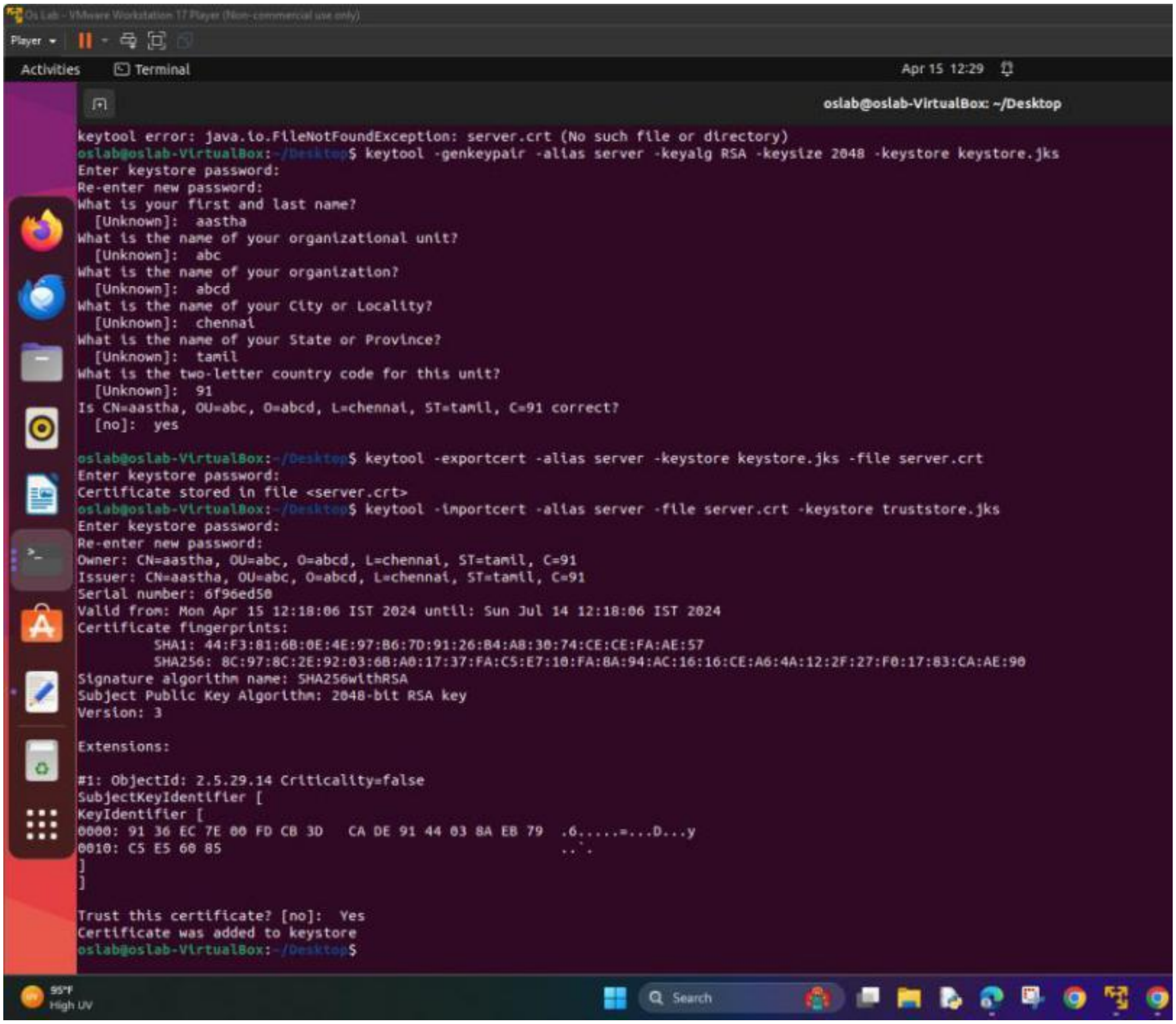
The first command exports the server's certificate from the keystore and saves it as `server.crt`. The second command imports the certificate into the truststore.

During the import command, you'll be prompted to trust the certificate. Confirm by entering `yes` or `y`.

4. Clean up by deleting the temporary certificate
   file: rm server.crt

   This command removes the `server.crt` file that was created during the export/import process.

OUTPUT:


Os Lab - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Apr 15 12:29
oslab@oslab-VirtualBox: ~/Desktop
95°F
High UV
Search

```
keytool error: java.io.FileNotFoundException: server.crt (No such file or directory)
oslab@oslab-VirtualBox:~/Desktop$ keytool -genkeypair -alias server -keyalg RSA -keysize 2048 -keystore keystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  aastha
What is the name of your organizational unit?
  [Unknown]:  abc
What is the name of your organization?
  [Unknown]:  abcd
What is the name of your City or Locality?
  [Unknown]:  chennai
What is the name of your State or Province?
  [Unknown]:  tamil
What is the two-letter country code for this unit?
  [Unknown]:  91
Is CN=aastha, OU=abc, O=abcd, L=chennai, ST=tamil, C=91 correct?
  [no]:  yes

oslab@oslab-VirtualBox:~/Desktop$ keytool -exportcert -alias server -keystore keystore.jks -file server.crt
Enter keystore password:
Certificate stored in file <server.crt>
oslab@oslab-VirtualBox:~/Desktop$ keytool -importcert -alias server -file server.crt -keystore truststore.jks
Enter keystore password:
Re-enter new password:
Owner: CN=aastha, OU=abc, O=abcd, L=chennai, ST=tamil, C=91
Issuer: CN=aastha, OU=abc, O=abcd, L=chennai, ST=tamil, C=91
Serial number: 6f96ed50
Valid from: Mon Apr 15 12:18:06 IST 2024 until: Sun Jul 14 12:18:06 IST 2024
Certificate fingerprints:
         SHA1: 44:F3:81:6B:0E:4E:97:B6:7D:91:26:B4:A8:30:74:CE:CE:FA:AE:57
         SHA256: 8C:97:8C:2E:92:03:6B:A0:17:37:FA:C5:E7:10:FA:8A:94:AC:16:16:CE:A6:4A:12:2F:27:F0:17:83:CA:AE:90
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 91 36 EC 7E 00 FD CB 3D   CA DE 91 44 03 8A EB 79  .6.....=...D...y
0010: C5 E5 60 85                                        ..`.
]
]

Trust this certificate? [no]:  Yes
Certificate was added to keystore
oslab@oslab-VirtualBox:~/Desktop$
```

CODE:

Server.java

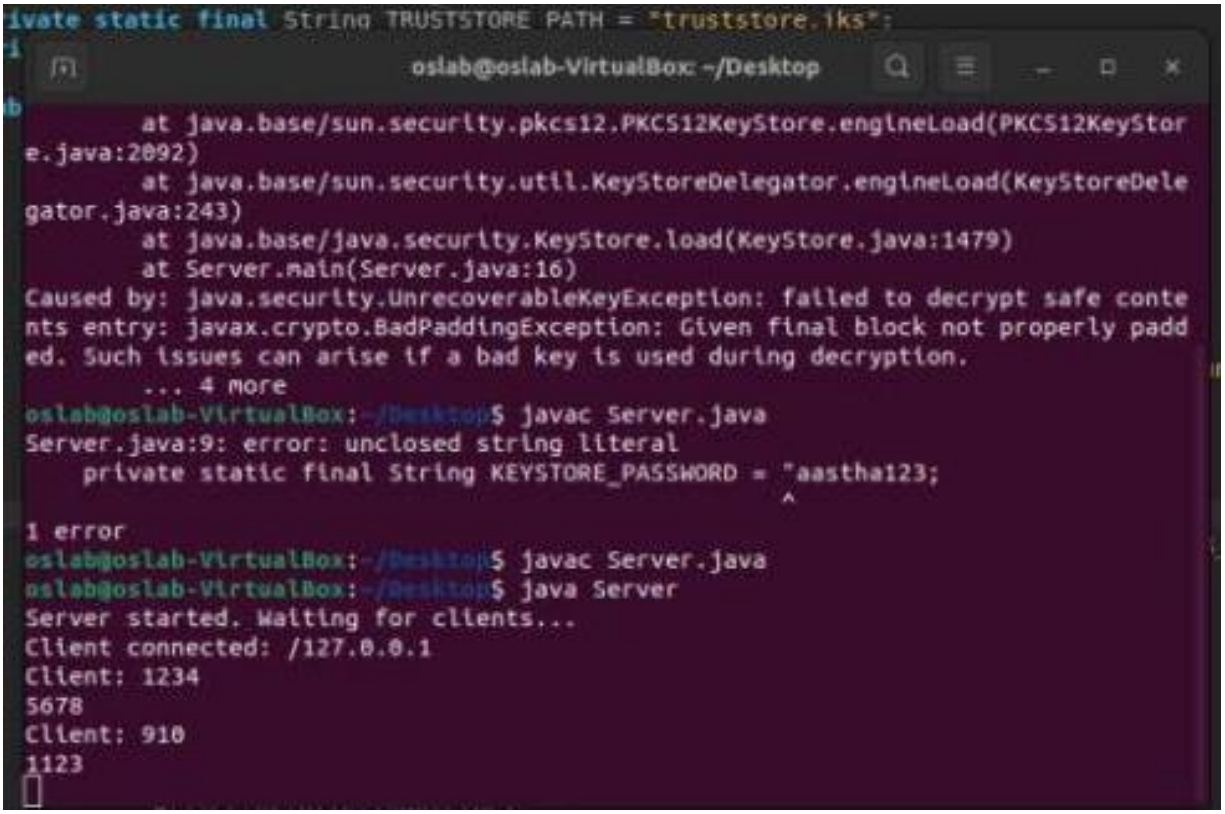
```
import javax.net.ssl.*;

import java.io.*;

import java.security.KeyStore;


public class Server {

  private static final int PORT = 8888;

  private static final String KEYSTORE_PATH =
  "keystore.jks"; private static final String
  KEYSTORE_PASSWORD = "aastha123";


  public static void main(String[] args) {

    try {
      // Load the keystore containing the server's
      private key KeyStore keyStore =
      KeyStore.getInstance("JKS");

      FileInputStream fileInputStream = new
      FileInputStream(KEYSTORE_PATH);
      keyStore.load(fileInputStream,
      KEYSTORE_PASSWORD.toCharArray());


      // Create SSL context
      SSLContext sslContext = SSLContext.getInstance("TLS");

      KeyManagerFactory keyManagerFactory =
      KeyManagerFactory.getInstance("SunX509"); keyManagerFactory.init(keyStore,
      KEYSTORE_PASSWORD.toCharArray());
      sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

      // Create SSL server socket factory
      SSLServerSocketFactory socketFactory = sslContext.getServerSocketFactory();
      SSLServerSocket serverSocket = (SSLServerSocket)
      socketFactory.createServerSocket(PORT);

      System.out.println("Server started. Waiting for clients...");

      while (true) {
        // Accept client connection
        SSLSocket clientSocket = (SSLSocket) serverSocket.accept();
        System.out.println("Client connected: "+clientSocket.getInetAddress());

        //  Start a new thread to handle client communication
        ClientHandler clientHandler = new
        ClientHandler(clientSocket); clientHandler.start();
      }
      } catch (Exception e) {
        e.printStackTrace();
    }
  }

  static class ClientHandler extends Thread {
    private final SSLSocket clientSocket;
```

```
    public ClientHandler(SSLSocket clientSocket) {

      this.clientSocket = clientSocket;

    }

    @Override

    public void run() {

      try (

        BufferedReader reader = new
BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

    PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true)

      ) {

        String inputLine;

        while ((inputLine = reader.readLine()) != null) {

          System.out.println("Client: " + inputLine);

          writer.println("Server: " + inputLine);

        }

      } catch (IOException e) {
        e.printStackTrace();

      } finally
        { try {

          clientSocket.close();

        } catch (IOException e) {

          e.printStackTrace();

        }

      }

    }

  }
}
```

Client.java

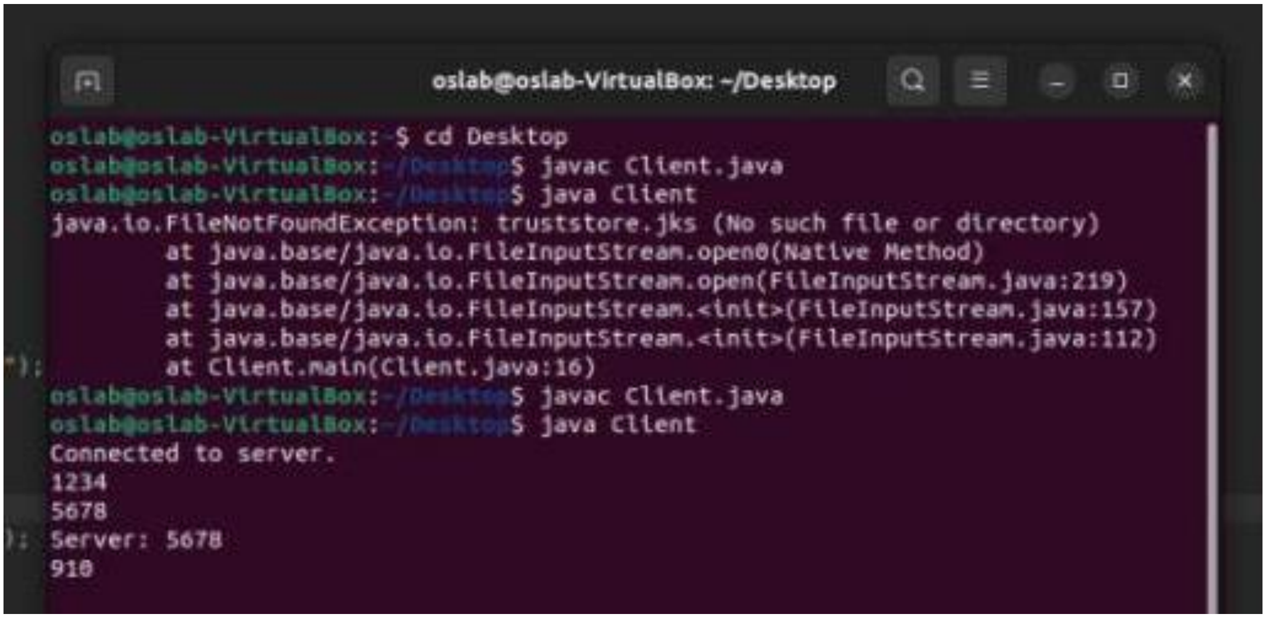
```
import javax.net.ssl.*;

import java.io.*;

import java.security.KeyStore;


public class Client {

  private static final String SERVER_HOST =

  "localhost"; private static final int

  SERVER_PORT = 8888;

  private static final String TRUSTSTORE_PATH =

  "truststore.jks"; private static final String

  TRUSTSTORE_PASSWORD = "aastha123";


  public static void main(String[] args) {

    try {
      // Load the truststore containing the server's public

      key KeyStore trustStore =

      KeyStore.getInstance("JKS");

      FileInputStream fileInputStream = new

      FileInputStream(TRUSTSTORE_PATH);

      trustStore.load(fileInputStream,

      TRUSTSTORE_PASSWORD.toCharArray());


      // Create SSL context

      SSLContext sslContext = SSLContext.getInstance("TLS");

      TrustManagerFactory trustManagerFactory =

      TrustManagerFactory.getInstance("SunX509"); trustManagerFactory.init(trustStore);

      sslContext.init(null, trustManagerFactory.getTrustManagers(), null);
      // Create SSL socket factory

      SSLSocketFactory socketFactory = sslContext.getSocketFactory();

       SSLSocket socket = (SSLSocket) socketFactory.createSocket(SERVER_HOST,
                                      SERVER_PORT);


      System.out.println("Connected to server.");

      BufferedReader reader = new BufferedReader(new

      InputStreamReader(System.in)); PrintWriter writer = new

      PrintWriter(socket.getOutputStream(), true);



      String inputLine;

      while ((inputLine = reader.readLine()) != null) {

        writer.println(inputLine);

        String response = reader.readLine();

        System.out.println("Server: " + response);
      }

      reader.close();

      writer.close();

      socket.close();

      } catch (Exception e) {

        e.printStackTrace();

    }

  }

}
```

OUTPUT:



```
ivate static final String TRUSTSTORE PATH = "truststore.jks";
```

```
        at java.base/sun.security.pkcs12.PKCS12KeyStore.engineLoad(PKCS12KeyStor
e.java:2092)
        at java.base/sun.security.util.KeyStoreDelegator.engineLoad(KeyStoreDele
gator.java:243)
        at java.base/java.security.KeyStore.load(KeyStore.java:1479)
        at Server.main(Server.java:16)
Caused by: java.security.UnrecoverableKeyException: failed to decrypt safe conte
nts entry: javax.crypto.BadPaddingException: Given final block not properly padd
ed. Such issues can arise if a bad key is used during decryption.
        ... 4 more
oslab@oslab-VirtualBox:~/Desktop$ javac Server.java
Server.java:9: error: unclosed string literal
    private static final String KEYSTORE_PASSWORD = "aastha123;
                                                    ^

1 error
oslab@oslab-VirtualBox:~/Desktop$ javac Server.java
oslab@oslab-VirtualBox:~/Desktop$ java Server
Server started. Waiting for clients...
Client connected: /127.0.0.1
Client: 1234
5678
Client: 910
1123
```



```
oslab@oslab-VirtualBox:~$ cd Desktop
oslab@oslab-VirtualBox:~/Desktop$ javac Client.java
oslab@oslab-VirtualBox:~/Desktop$ java Client
java.io.FileNotFoundException: truststore.jks (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:112)
        at Client.main(Client.java:16)
oslab@oslab-VirtualBox:~/Desktop$ javac Client.java
oslab@oslab-VirtualBox:~/Desktop$ java Client
Connected to server.
1234
5678
Server: 5678
910
```

Experiment 13 21/01/24

TITLE:

Develop a web application that implements JSON web token

OVERVIEW:

- JSON web token (JWT) is an open standard for securely transmitting information over the web between two parties as a JSON object.
- JWT is an open, industry standard method for representing claims securely between two parties
- Which means, a server can determine whether an information in JSON format sent by the client has not been modified and has effectively been issued by said server.
- These claims are used to identify the user and contain information such as the user's ID roles, or any other claim the issuer defines.
- JWTs are compact, self- contained and digitally signed, making them suitable for authentication and information exchange

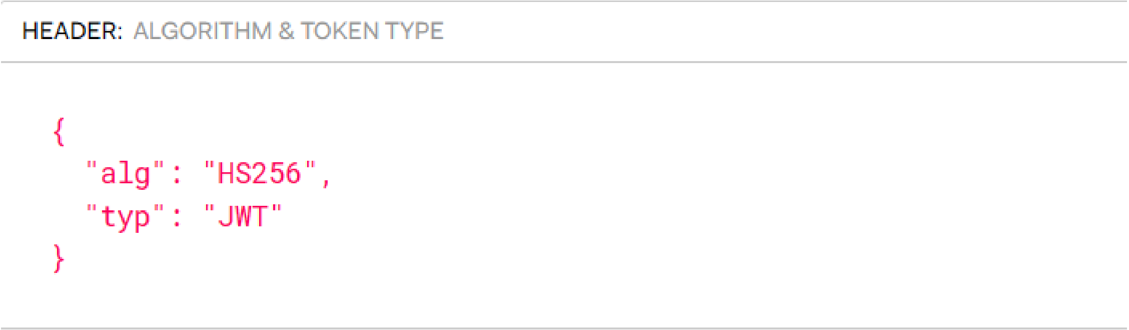
A JSON WEB TOKEN is composed of three parts separated by (.) which are:

1. HEADER
2. PAYLOAD
3. SIGNATURE

A JWT typically looks like this: xxxxx.yyyyy.zzzzz

HEADER:

- The header typically consists of two parts:
- The type of the token: which is JWT
- The hashing algorithm such as HMAC, SHA256 or RSA
- For example:

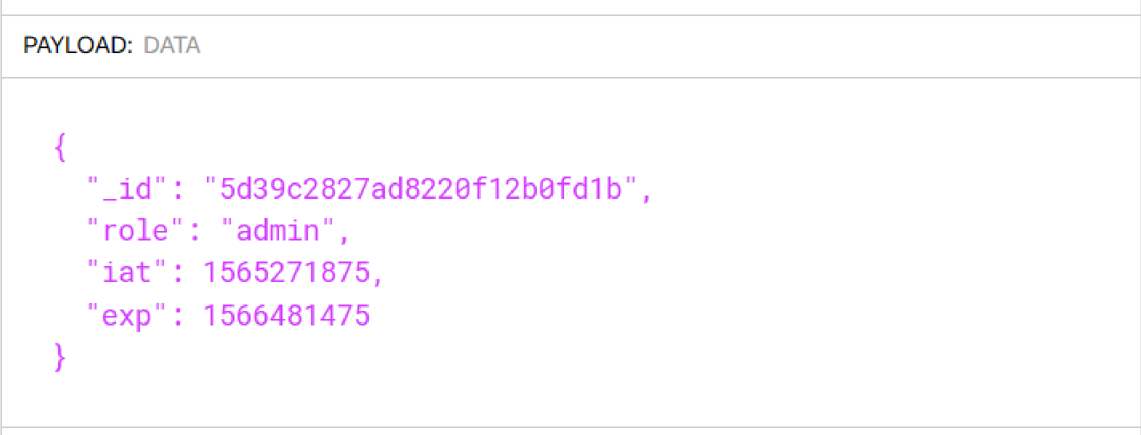


HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

- This sections contains the claims.
- Claims are statements about the user (like user ID, user roles, expiration time) and additional metadata.
- There are three types of claims: 1. registered 2. public 3.private
- Registered: set of predefined claims, which are not mandatory but recommended, thought to provide a set of useful, interoperable claims. Some of them are: iss(issuer), exp(expiration time), sub(subject), aud(audience), among others.
- Public: these can be defined by those using JWTs
- Private: these are the custom claims created to share information between partied that agree on using them.

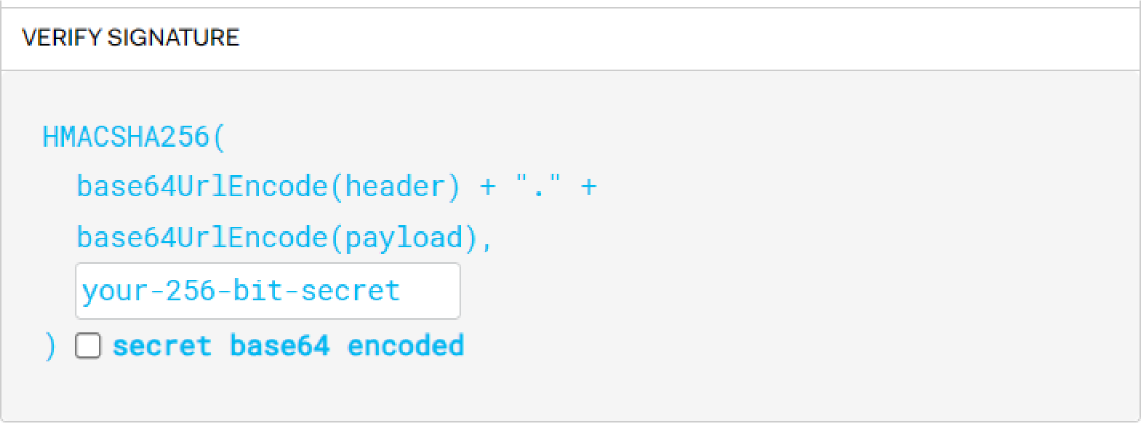


PAYLOAD: DATA

```
{
  "_id": "5d39c2827ad8220f12b0fd1b",
  "role": "admin",
  "iat": 1565271875,
  "exp": 1566481475
}
```

SIGNATURE:

- The signature contains the encoded header, encoded payload and a secret key (that only the server knows) and is signed by the algorithm specified in the header
- If the hashing algorithm to be used was SHA256, then the signature created would be as such:



VERIFY SIGNATURE

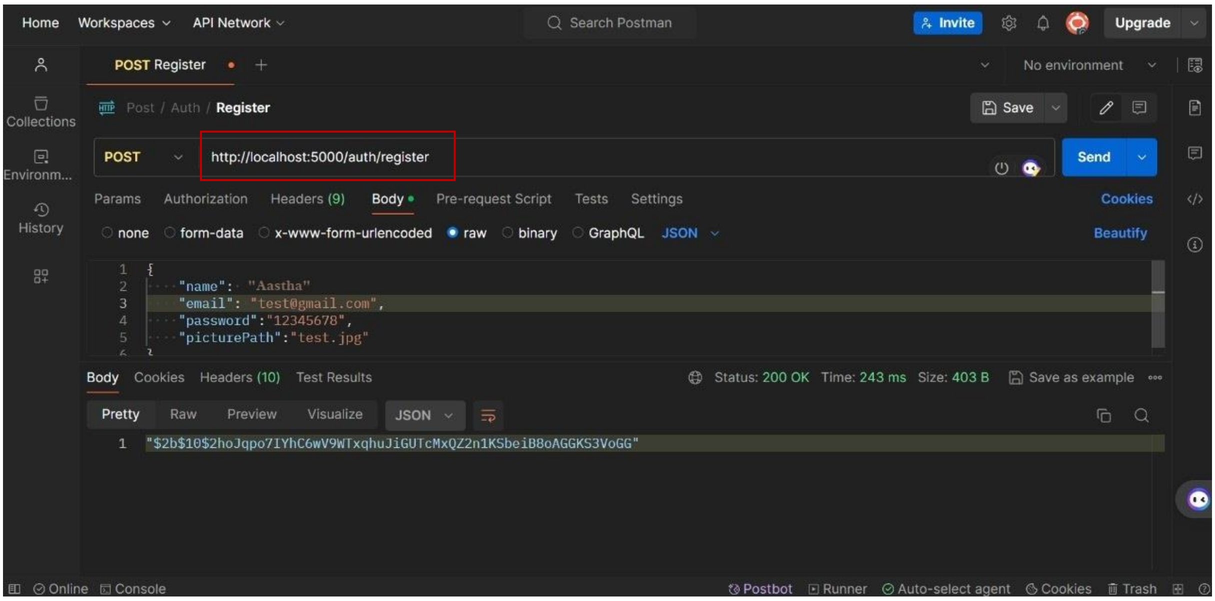
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

ALGORITHM:

1. The first move is made by the client. This could be a web frontend application, a mobile app, etc. Basically anything that tries to interact with your backend application (for example: a REST API). It sends their login credentials to the server for it to become verified.
2. When the server receives the login request, it first makes sure that the username/email and password match up with information stored in the database. When the credentials are correct, this means for the server that this user is who he says he is.
3. Next, the JWT token is being generated. Here, information that's important for identifying the user are being passed into the payload. It's also a good idea to include issue and expiration dates. So a session would never be longer valid than the time you indicate.
4. The token then is returned to the client as a response to his login attempt. When he receives a token, that means for him the login has been successful. The token should be stored somewhere locally on the client-side. This can be local Store for web applications or somewhere in a device variable for mobile applications.
5. For all further communication with the server, the client adds an Authentication header to each request. The client includes the token in the headers of subsequent requests. This looks as such:
   Authentication: Bearer<token>
6. When a new request to a protected resource arrives at the server, the first thing it does is to check if an Authentication header is passed along with the request. Is this the case, it tries to verify if the token checks out. If it's not a valid token (it has been tampered with, it has expired, etc.), the request shall be denied immediately. So the server validated the token's authenticity and integrity.
7. If the token is valid however, it's safe to assume for the server that the user is still who he says he is and can return the requested resource as response to the client.

OUTPUT:

Server side scripting:



POST http://localhost:5000/auth/register

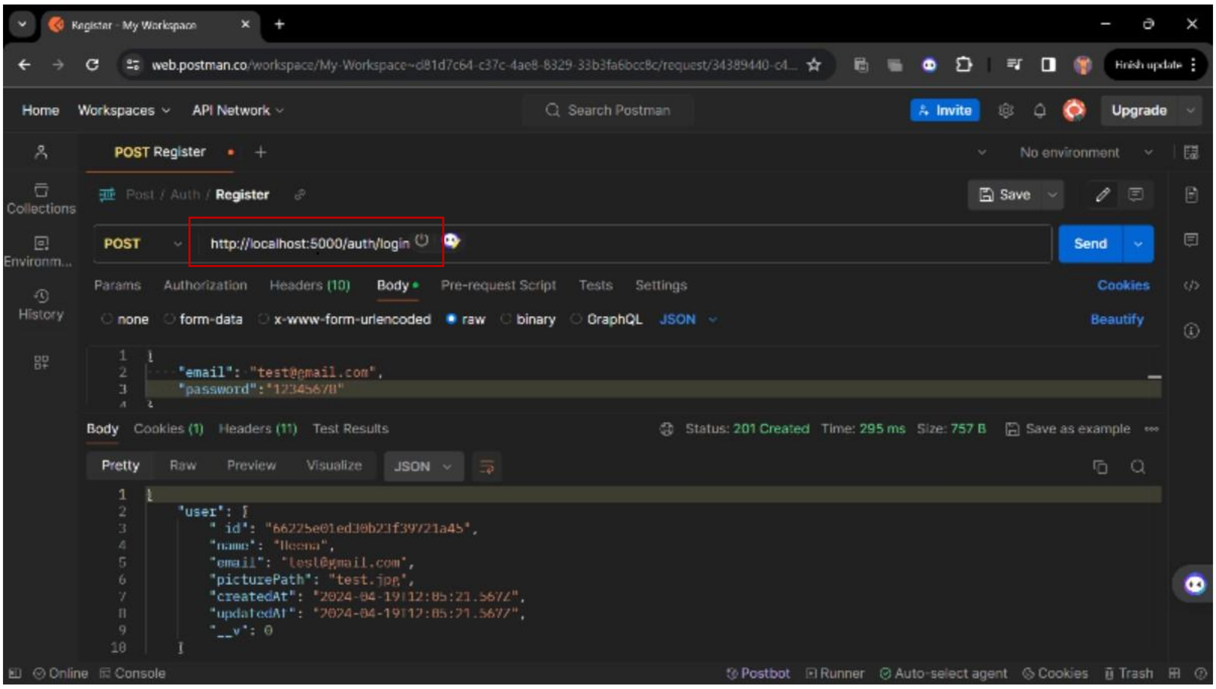
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "name":  "Aastha"
    "email": "test@gmail.com",
    "password":"12345678",
    "picturePath":"test.jpg"
}
```

Body Cookies Headers (10) Test Results — Status: 200 OK Time: 243 ms Size: 403 B

```
"$2b$10$2hoJqpo7IYhC6wV9WTxqhuJiGUTcMxQZ2n1KSbeiB8oAGGKS3VoGG"
```

Successful Login:



POST http://localhost:5000/auth/login

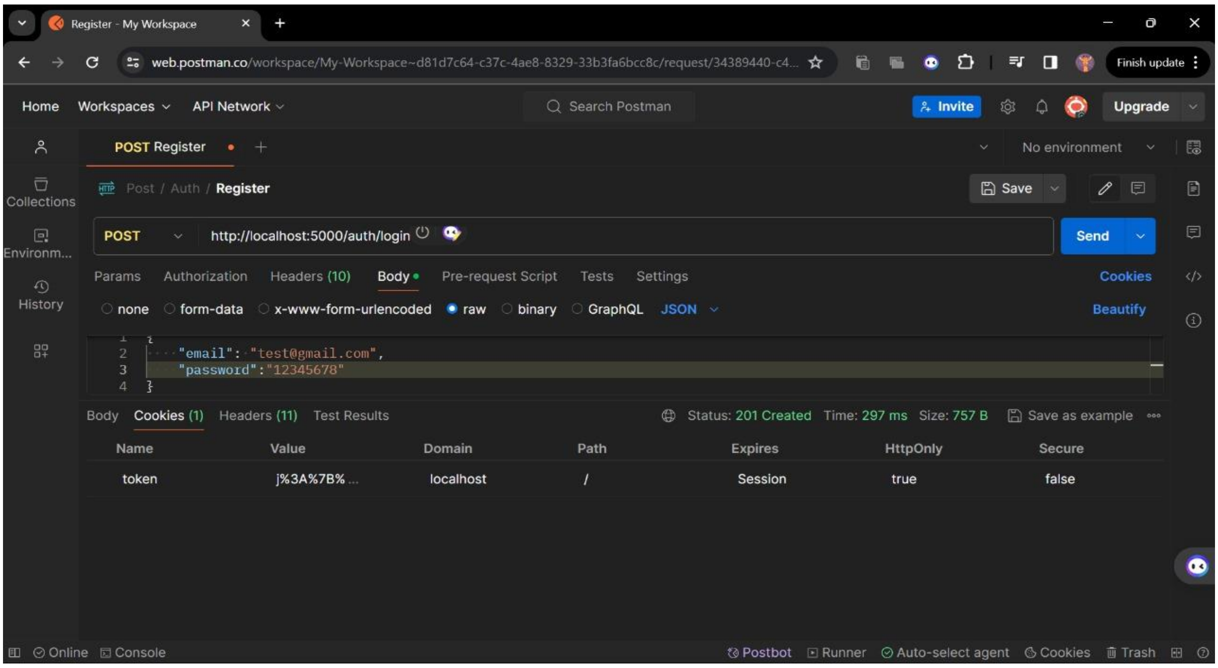
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "email": "test@gmail.com",
    "password":"12345678"
}
```

Body Cookies (1) Headers (11) Test Results — Status: 201 Created Time: 295 ms Size: 757 B

```
{
    "user": {
        "_id": "66225e01ed30b23f39721a45",
        "name": "Heena",
        "email": "test@gmail.com",
        "picturePath": "test.jpg",
        "createdAt": "2024-04-19T12:05:21.567Z",
        "updatedAt": "2024-04-19T12:05:21.567Z",
        "__v": 0
    }
```

Cookie is returned successfully:


Register - My Workspace
web.postman.co/workspace/My-Workspace~d81d7c64-c37c-4ae8-8329-33b3fa6bcc8c/request/34389440-c4...
Home Workspaces API Network Search Postman Invite Upgrade
POST Register
No environment
Post / Auth / Register
Save
Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies
Beautify
Collections Environm... History
Online Console Postbot Runner Auto-select agent Cookies Trash

POST http://localhost:5000/auth/login Send

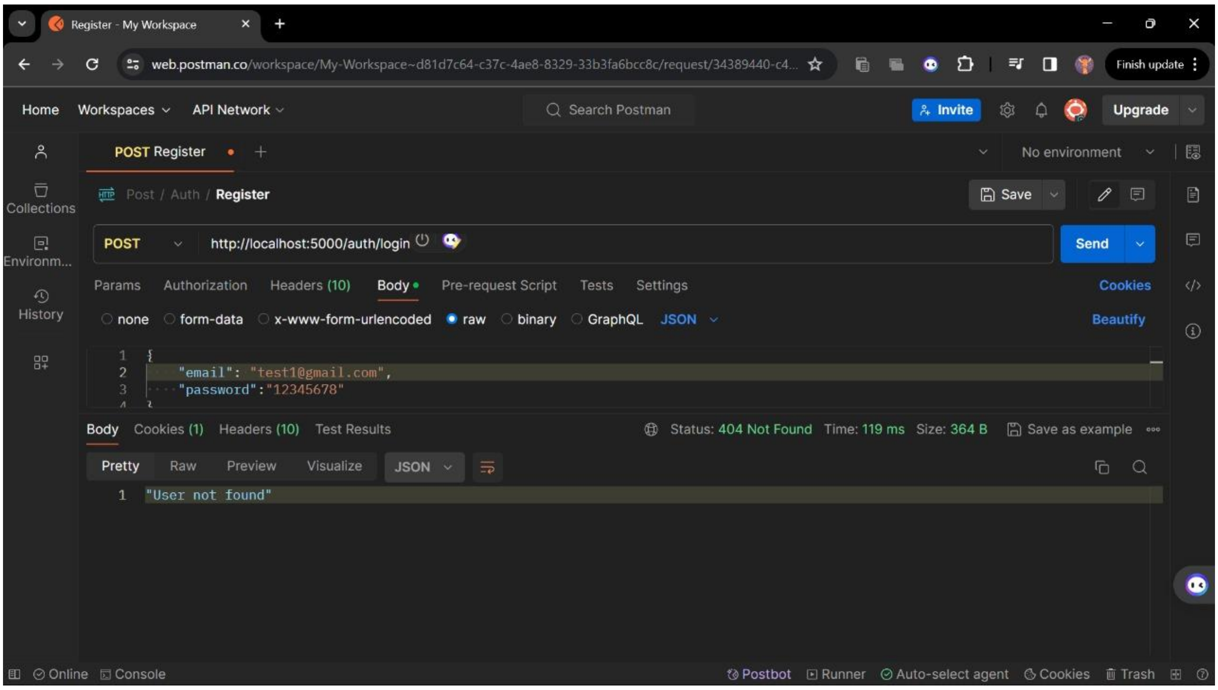
Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "email": "test@gmail.com",
    "password":"12345678"
}
```

Body Cookies (1) Headers (11) Test Results Status: 201 Created Time: 297 ms Size: 757 B Save as example

Name	Value	Domain	Path	Expires	HttpOnly	Secure
token	j%3A%7B% ...	localhost	/	Session	true	false

Trying to login with wrong user mail :


Register - My Workspace
web.postman.co/workspace/My-Workspace~d81d7c64-c37c-4ae8-8329-33b3fa6bcc8c/request/34389440-c4...
Home Workspaces API Network Search Postman Invite Upgrade
POST Register
No environment
Post / Auth / Register
Save
Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies
Beautify
Collections Environm... History
Pretty Raw Preview Visualize JSON
Online Console Postbot Runner Auto-select agent Cookies Trash

POST http://localhost:5000/auth/login Send

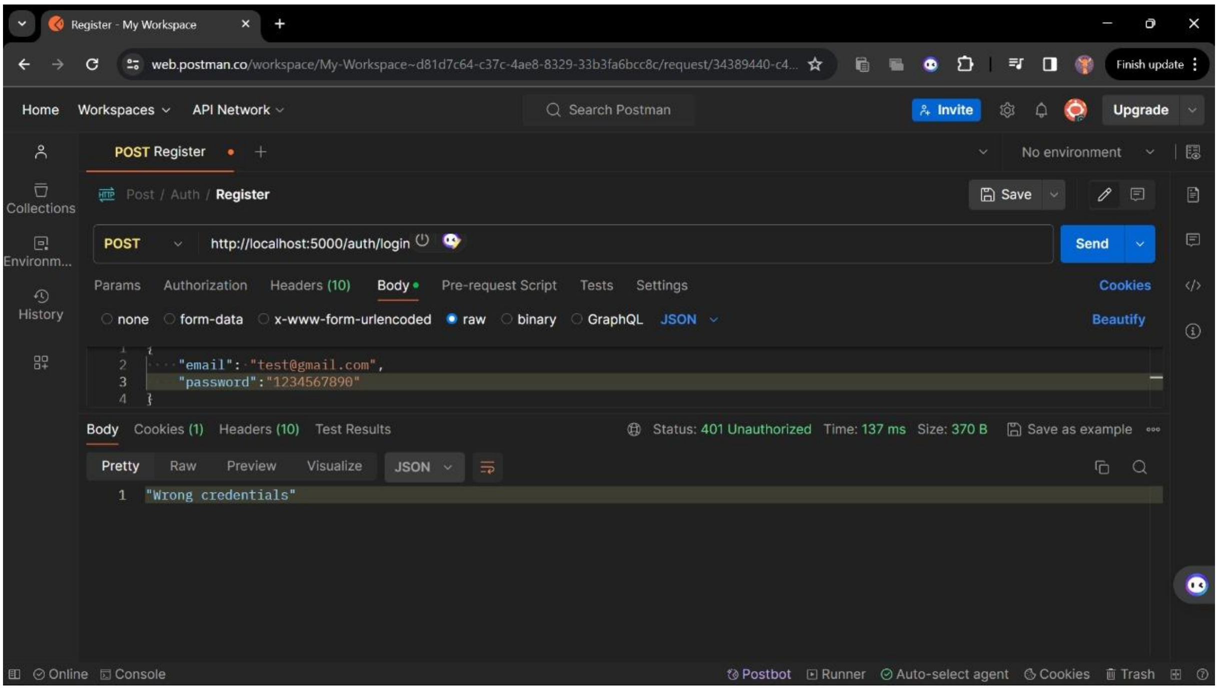
Body: ○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "email": "test1@gmail.com",
    "password":"12345678"
}
```

Body Cookies (1) Headers (10) Test Results Status: 404 Not Found Time: 119 ms Size: 364 B Save as example

```
"User not found"
```

Trying to login with correct user but wrong password :



POST Register

Post / Auth / Register

POST http://localhost:5000/auth/login

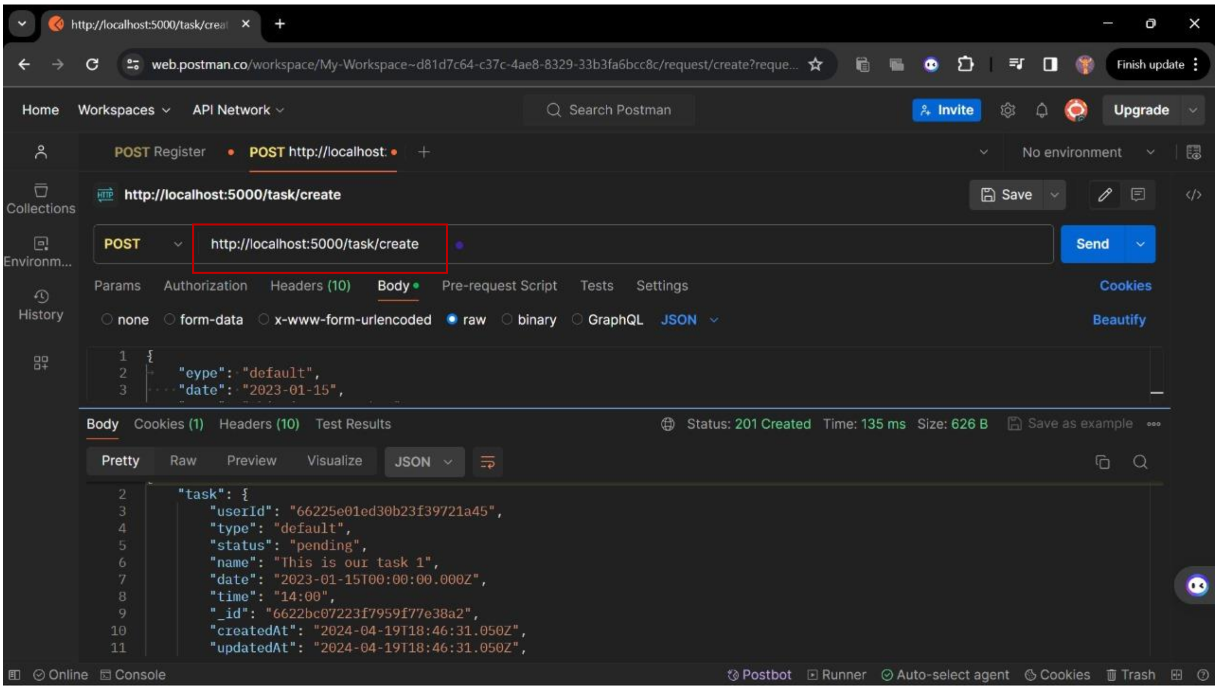
Body ● raw JSON

```
{
    "email": "test@gmail.com",
    "password":"1234567890"
}
```

Status: 401 Unauthorized Time: 137 ms Size: 370 B

```
"Wrong credentials"
```

Task saved successfully:



http://localhost:5000/task/create

POST http://localhost:5000/task/create

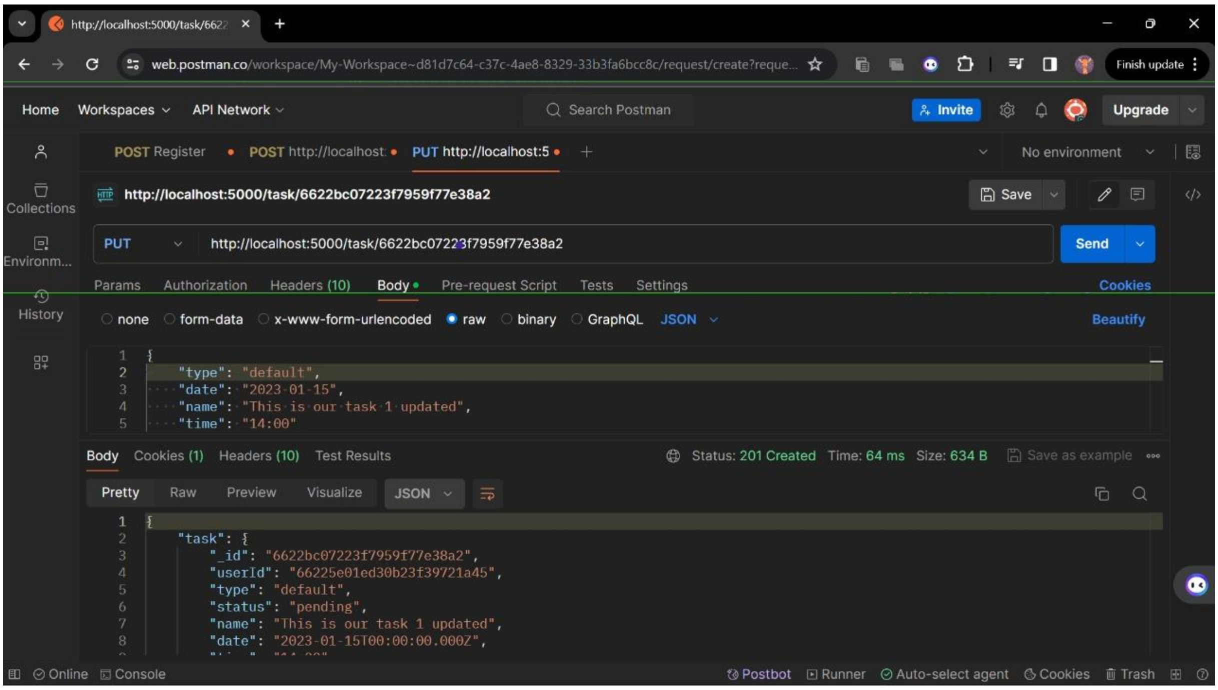
Body ● raw JSON

```
{
    "eype": "default",
    "date": "2023-01-15",
```

Status: 201 Created Time: 135 ms Size: 626 B

```
    "task": {
        "userId": "66225e01ed30b23f39721a45",
        "type": "default",
        "status": "pending",
        "name": "This is our task 1",
        "date": "2023-01-15T00:00:00.000Z",
        "time": "14:00",
        "_id": "6622bc07223f7959f77e38a2",
        "createdAt": "2024-04-19T18:46:31.050Z",
        "updatedAt": "2024-04-19T18:46:31.050Z",
```

Task updated successfully:



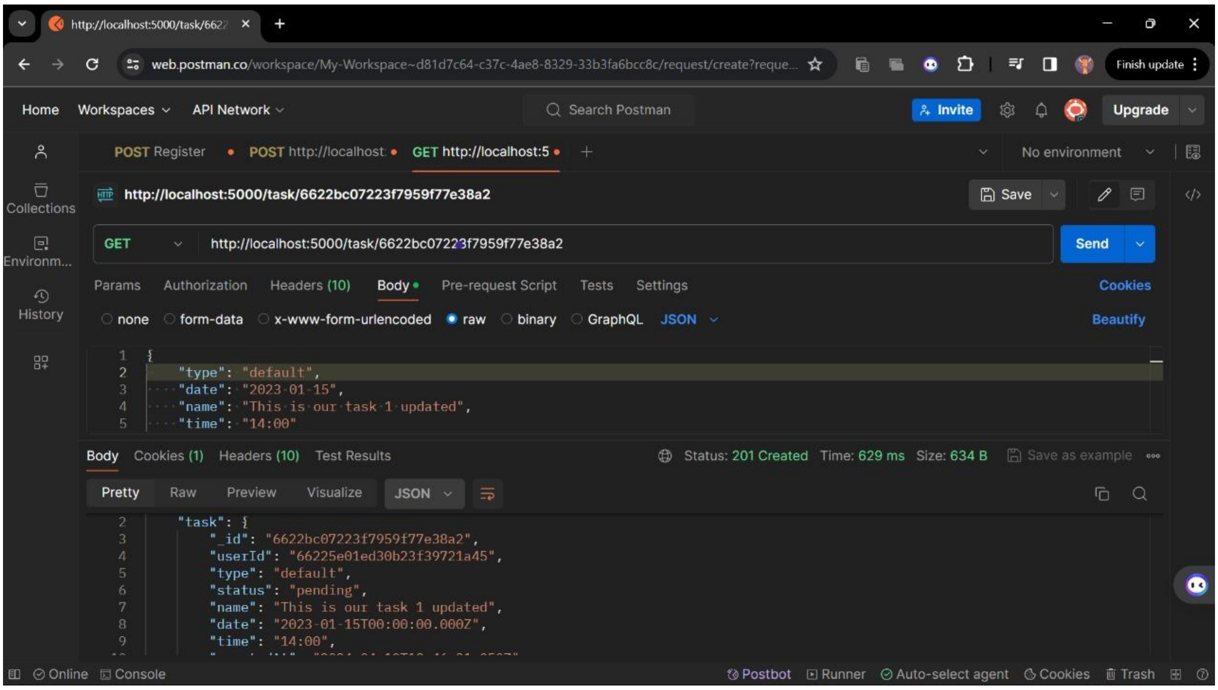
PUT http://localhost:5000/task/6622bc07223f7959f77e38a2

```
{
    "type": "default",
    "date": "2023-01-15",
    "name": "This is our task 1 updated",
    "time": "14:00"
```

Status: 201 Created Time: 64 ms Size: 634 B

```
{
    "task": {
        "_id": "6622bc07223f7959f77e38a2",
        "userId": "66225e01ed30b23f39721a45",
        "type": "default",
        "status": "pending",
        "name": "This is our task 1 updated",
        "date": "2023-01-15T00:00:00.000Z",
```

Get API successfully:



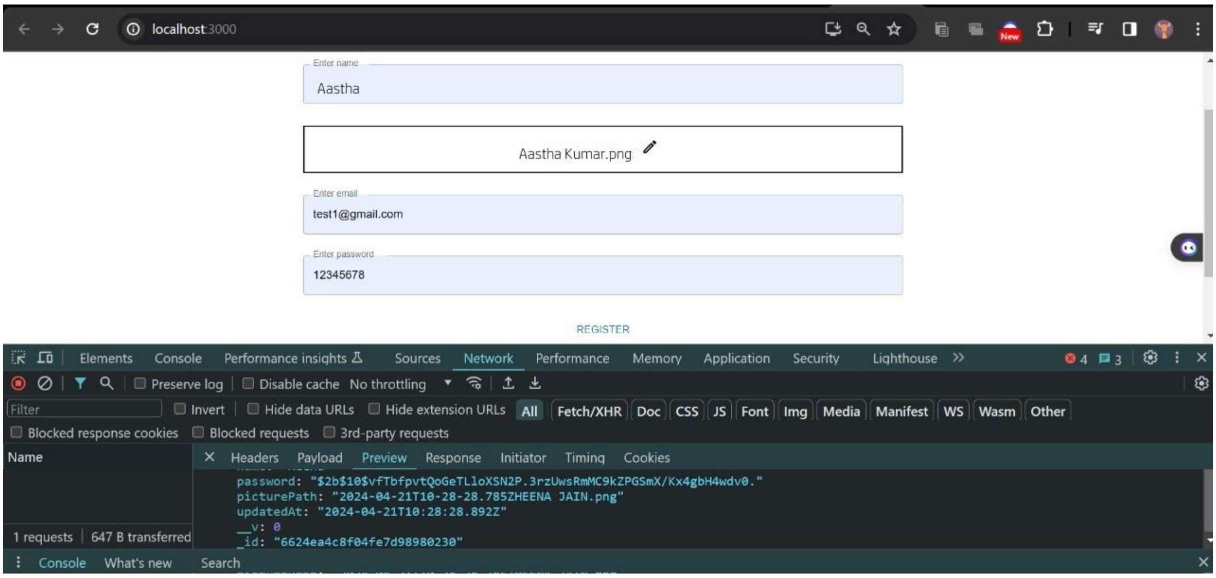
GET http://localhost:5000/task/6622bc07223f7959f77e38a2

```
{
    "type": "default",
    "date": "2023-01-15",
    "name": "This is our task 1 updated",
    "time": "14:00"
```

Status: 201 Created Time: 629 ms Size: 634 B

```
    "task": {
        "_id": "6622bc07223f7959f77e38a2",
        "userId": "66225e01ed30b23f39721a45",
        "type": "default",
        "status": "pending",
        "name": "This is our task 1 updated",
        "date": "2023-01-15T00:00:00.000Z",
        "time": "14:00",
```

Client side Scripting :



Enter name: Aastha

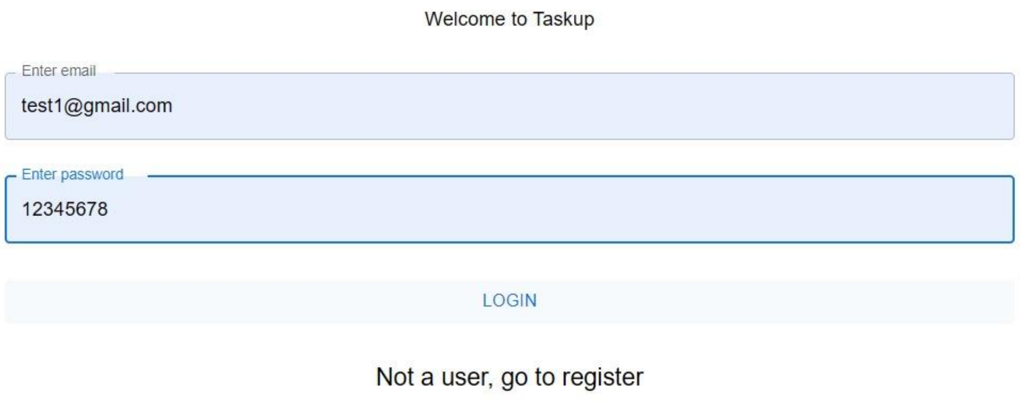
Aastha Kumar.png

Enter email: test1@gmail.com

Enter password: 12345678

```
password: "$2b$10$vfTbfpvtQoGeTLloXSN2P.3rzUwsRmMC9kZPGSmX/Kx4gbH4wdv0."
picturePath: "2024-04-21T10-28-28.785ZHEENA JAIN.png"
updatedAt: "2024-04-21T10:28:28.892Z"
__v: 0
_id: "6624ea4c8f04fe7d98980230"
```

Login :



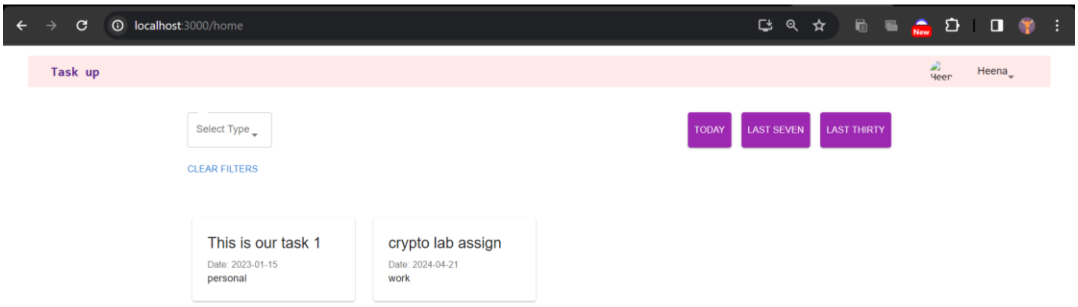
Welcome to Taskup

Enter email: test1@gmail.com

Enter password: 12345678

Not a user, go to register

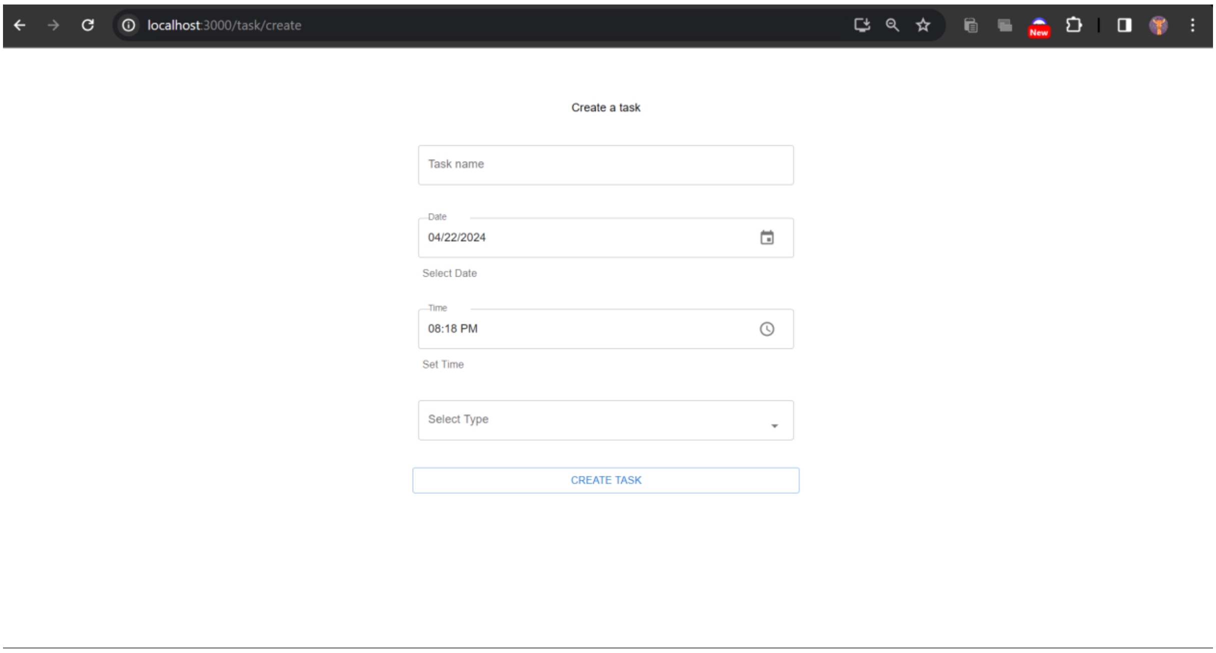
Log in success : Home Page



This is our task 1
Date: 2023-01-15
personal

crypto lab assign
Date: 2024-04-21
work

Create task:



localhost:3000/task/create

Create a task

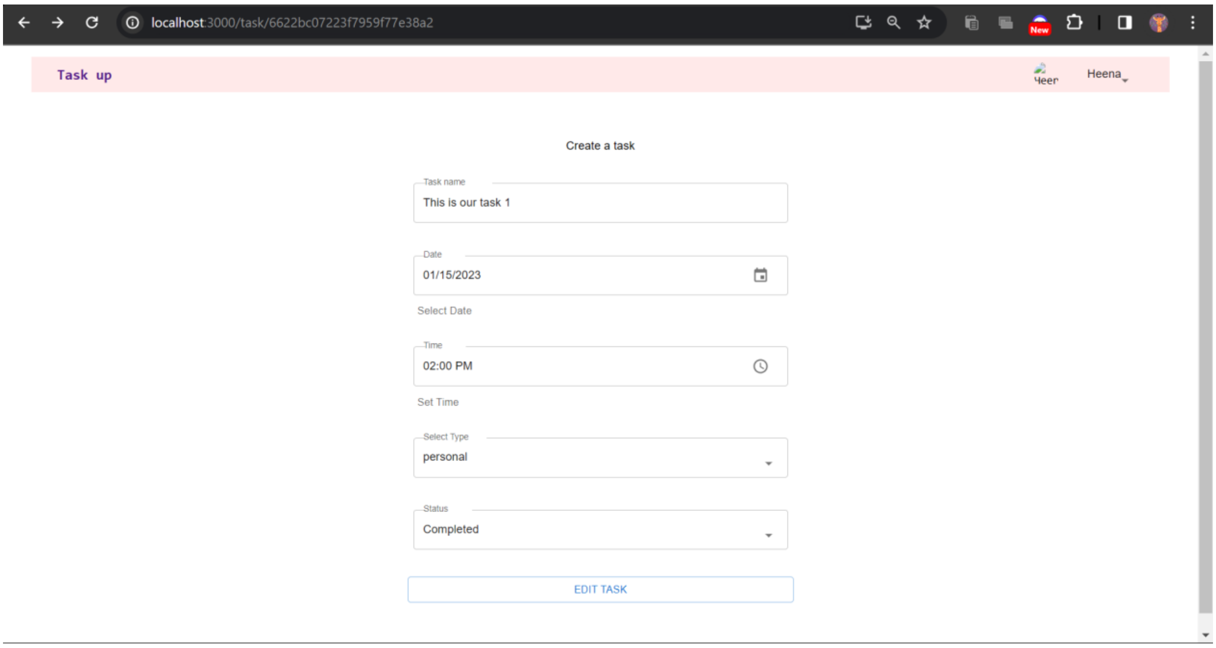
Task name

Date
04/22/2024
Select Date

Time
08:18 PM
Set Time

Select Type

Edit task:



localhost:3000/task/6622bc07223f7959f77e38a2

Create a task

Task name
This is our task 1

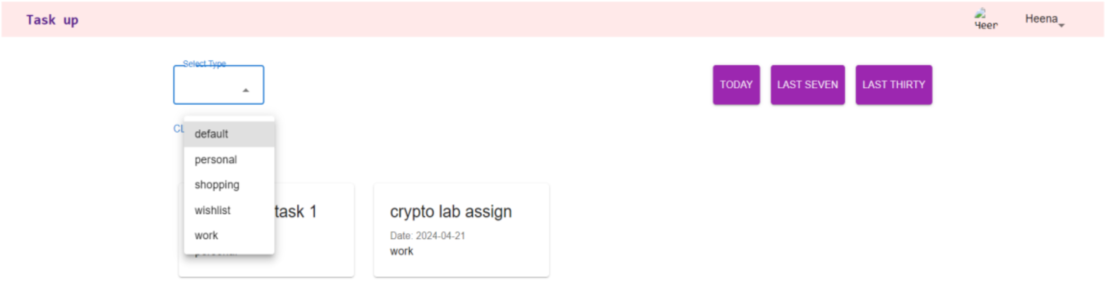
Date
01/15/2023
Select Date

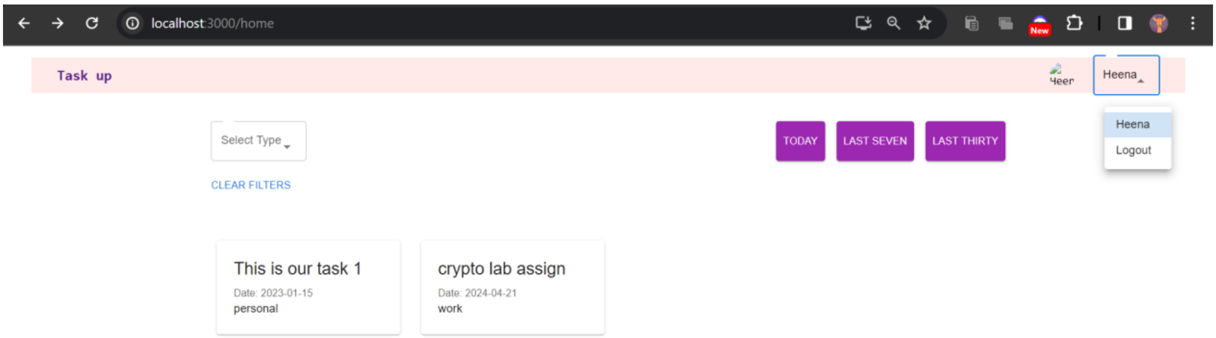
Time
02:00 PM
Set Time

Select Type
personal

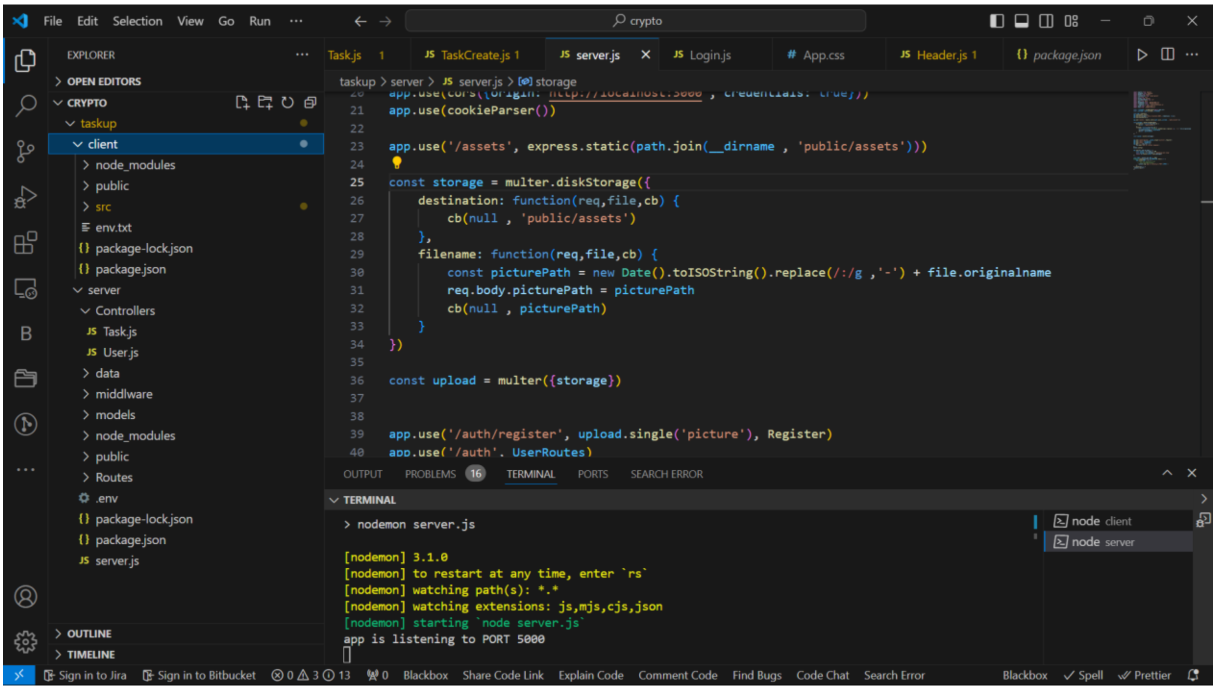
Status
Completed

Perosnlaized filters an dlogout functionality:





Folder Structure:



```
app.use(cors({origin: "http://localhost:3000" , credentials: true}))
app.use(cookieParser())

app.use('/assets', express.static(path.join(__dirname , 'public/assets')))

const storage = multer.diskStorage({
    destination: function(req,file,cb) {
        cb(null , 'public/assets')
    },
    filename: function(req,file,cb) {
        const picturePath = new Date().toISOString().replace(/:/g ,'-') + file.originalname
        req.body.picturePath = picturePath
        cb(null , picturePath)
    }
})

const upload = multer({storage})


app.use('/auth/register', upload.single('picture'), Register)
app.use('/auth', UserRoutes)
```

```
> nodemon server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
app is listening to PORT 5000
```