COMPUTER NETWORKS LABORATORY

MINI PROJECT

(19IT551)

CONCURRENT CLIENT SERVER SYSTEM WITH AES ENCRYPTION

A REPORT

Submitted by

M.HARI HARA SUDAN (9517202206021)

N.A KISHORE (9517202206024)

K.DHANIS AHMED (9517202206014)

DEPARTMENT OF INFORMATION TECHNOLOGY

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Institution affiliated to Anna University Chennai)

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project report is the bonafide work of **M.HARI HARA SUDAN (95172022060021-22BIT043)**, **K.DHANIS AHMED (9517202206014-22BIT037) , N.A KISHORE (9517202206024-22BIT045)** for their Mini Project on **CONCURRENT CLIENT SERVER SYSTEM WITH AES ENCRYPTION** in the 19IT551 COMPUTER NETWORKS LABORATORY at Mepco Schelenk Engineering College,Sivakasi during the year 2024-2025

SIGNATURE	**SIGNATURE**
Dr. R. Venitta Raj, M.E, PhD	**Dr. T. Revathi, M.E., Ph.D.,**
STAFF IN-CHARGE	**HEAD OF THE DEPARTMENT**
Associate Professor	**Senior Professor**
Information Technology	**Information Technology**
Mepco Schlenk Engineering College	**Mepco Schlenk Engineering College**
Virudhunagar Dt. – 626 005	**Virudhunagar Dt. – 626 005**

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and our reputed institution Mepco Schlenk Engineering College, Sivakasi. We would like to extend our sincere thanks to our Principal Dr. S. Arivazhagan ME., PhD.

We would like to express my special gratitude and thanks to Dr.T.Revathi ME., PhD. Senior Professor & Head of the Department for giving us such a wonderful experience.

We are highly indebted to Dr. T. Revathi Sr. Prof. & HOD, Dr. R. Venitta Raj Associate Professor and Mrs. P.Kaviya AP(SI.Grade) for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express our gratitude towards other teaching and non teaching staff members of IT Department for their kind co-operation and encouragement which help us in all means to complete this project successful.

Abstract:

The Client-Server Printing Application with AES Encryption is designed to securely transmit print jobs over a network by using Advanced Encryption Standard (AES) encryption for ensuring the confidentiality of sensitive data. This project creates a client-server system where the client sends encrypted print data to the server, which decrypts the data and sends it to a connected printer. The server manages multiple client requests, organizes jobs in a queue, estimates the wait time for each job, and provides real-time feedback to the clients. The feedback includes job status notifications, such as success or failure, based on printer availability and job processing. This system ensures a high level of data security while maintaining efficient handling of print jobs in a multi-client environment. The project serves as an essential tool for organizations that prioritize security, such as universities, government offices, and enterprises that regularly process sensitive documents. By implementing AES encryption and a multi-client print queue management system, this project addresses critical security and scalability concerns in a networked printing environment.

TABLE OF CONTENTS

1.Introduction:

In today's digital world, networked printing is a common practice, allowing multiple users to send print jobs to shared printers over a network. While the convenience of remote printing is undeniable, it often introduces significant security risks, particularly when printing sensitive documents. Data sent through an unsecured network is susceptible to interception, which could compromise confidentiality and lead to data breaches. This project aims to mitigate these risks by implementing AES encryption for secure transmission of print jobs from client devices to a central server. The server, upon receiving the encrypted print job, decrypts the file and processes it for printing.

The Client-Server Printing Application is based on the idea of a secure communication model, where clients can send encrypted print data over a TCP/IP network. The server, acting as a central hub, not only manages multiple print jobs but also provides real-time feedback to the clients, including wait times and job status (whether successful or failed). The server's role also extends to managing a print queue, which helps in organizing and prioritizing multiple client requests. This system is particularly useful for institutions or organizations where print data confidentiality is crucial, and where efficient job processing and feedback are necessary for smooth operations.

The integration of AES encryption ensures that the data remains protected even if intercepted, preventing unauthorized access to confidential information. By combining encryption with a robust print management system, the project addresses both security and efficiency concerns in the context of network printing.

Additionally, the application’s modular design allows for easy expansion, such as integrating it with advanced printers or adding more security layers.

1.1.Purpose:

1. Enhance Data Security: The project aims to secure data transmission between the client and the server using Advanced Encryption Standard (AES), ensuring the confidentiality and integrity of sensitive information.

2. Demonstrate Encryption Techniques: By integrating AES encryption, the project showcases the practical application of modern cryptographic algorithms in real-world scenarios, essential for secure data transfer in networked systems.

3. Efficient Client-Server Communication: The implementation aims to demonstrate effective socket programming in Java, enabling reliable and efficient data exchange between multiple clients and a central server over a network.

4. Simulate Real-World Networking: The project models a real-world client-server architecture where multiple clients can communicate securely with a central server, similar to systems used in banking, e-commerce, and cloud services.

5. Develop Multithreading Proficiency: By handling multiple client connections using threads, the project emphasizes the importance of multithreaded programming in creating responsive and high-performance network applications.

6. Implement Print Job Management: The project includes managing print jobs with status updates, simulating a real-life application in environments such as educational institutions or offices, where print job queues need efficient handling.

7. Error Handling and Resource Management: The project focuses on robust error handling and efficient resource management, which are critical aspects of any networking application to ensure reliability and performance.

8. User-Friendly Interface: By providing a graphical user interface (GUI) for the client application, the project demonstrates the importance of designing user-friendly and intuitive interfaces for non-technical users.

9.Learning and Educational Value: This project serves as a comprehensive educational tool for understanding core concepts in computer networks, such as encryption, socket programming, and concurrency, while also enhancing programming skills in Java.

1.2.Scope:

The scope of this project is to develop a Client-Server Printing Application with an emphasis on security and job management in a multi-client environment. This system will be able to securely transmit print jobs from clients to the server using AES encryption, effectively protecting sensitive data from interception during transmission. Once the server receives an encrypted print job, it will decrypt the job and send it to a printer for processing.

The server will handle multiple client requests simultaneously, ensuring that each client receives a unique queue position. It will manage the print jobs by estimating wait times for each client based on the number of jobs already in the queue. Clients will receive notifications about their queue position and estimated wait time, providing them with transparency about the status of their print job.

Key components

- AES encryption for securing print data before it is transmitted over the network.
- Client-server communication using TCP/IP protocols for data transmission.
- Job queue management to handle multiple print requests and ensure that jobs are processed in the correct order.
- Printer managementfor ensuring that the print jobs are sent to the printer in the correct sequence.
- Job feedback system to notify the client about the success or failure of their print job, including messages about printer availability and issues.

Limitations:

- The application will be developed for single-printer environments, though it could be extended to support multiple printers in the future.
- The project will work within a local network (LAN), not designed for wide-area network (WAN) environments or cloud-based printing.
- Error handling will focus primarily on job failures due to printer availability but will not extensively cover network failures or other external issues.

1.3.Definitions, Acronyms and Abbreviations:

1. AES (Advanced Encryption Standard):

A symmetric encryption algorithm widely used across the world to protect sensitive data. AES encrypts data in blocks of 128 bits and supports key sizes of 128, 192, and 256 bits. It is considered highly secure and efficient.

2. Client-Server Architecture:

A model where a client (user) makes requests to a server (central system), which processes the requests and sends back responses. The server provides services such as data storage, job management, or computation, while the client interacts with the server to initiate operations.

3. Encryption:

The process of converting data into a secure format using an algorithm (such as AES) to prevent unauthorized access. Only users with the correct decryption key can revert the data to its original format.

4. Decryption:

The reverse process of encryption, where encrypted data is converted back into its original form using the appropriate decryption key.

5. TCP/IP (Transmission Control Protocol/Internet Protocol):

A set of communication protocols used to connect devices over a network. TCP ensures reliable, error-free transmission of data, while IP handles the addressing and routing of data packets.

6. Job Queue:

A queue or list in which print jobs are stored temporarily until they are processed. In this system, jobs are assigned positions, and users are given an estimate of wait times based on their position in the queue.

7. Print Server:

A server that manages print jobs from multiple clients. It receives the print data, manages the print queue, and sends the print job to the printer for output.

8. Job ID:

A unique identifier assigned to each print job to track its progress through the system, from submission to completion.

9. Queue Position:

The relative position of a print job in the print queue. It determines the order in which jobs are printed and helps estimate the wait time for each user.

10. Printer Status:

A message indicating whether a print job has been successfully printed, failed, or encountered an issue (such as a printer being offline or unavailable).

Acronyms

1. AES – Advanced Encryption Standard

A symmetric encryption algorithm used to secure data transmission.

2. TCP/IP – Transmission Control Protocol/Internet Protocol

A suite of protocols used for communication over a network.

3. GUI – Graphical User Interface

A user interface that allows users to interact with the software using graphical elements such as icons and buttons.

4. LAN – Local Area Network

A network that connects computers and devices within a limited geographic area, such as a single building or campus.

5. WAN – Wide Area Network

A large-scale network that connects devices across broader geographical areas, typically spanning multiple cities or countries.

6. FTP – File Transfer Protocol

A standard network protocol used for transferring files between client and server over a TCP/IP network.

7. SSL – Secure Sockets Layer

A protocol for securing data transmission over the internet, often used for encrypting web traffic.

8. GUI – Graphical User Interface

A type of user interface that allows interaction through visual elements like buttons and windows.

Abbreviations

1. IP – Internet Protocol

A protocol used for addressing and routing data across networks.

2. OS – Operating System

The software that manages hardware resources and provides services for computer programs.

3. JDBC – Java Database Connectivity

An API for connecting and executing queries on a database in Java applications.

4. SMTP – Simple Mail Transfer Protocol

A protocol used for sending emails across the internet.

5. URL – Uniform Resource Locator

A web address used to locate resources on the internet.

6. RAM – Random Access Memory

A type of computer memory used for temporarily storing data that is actively being used or processed.

7. XML – Extensible Markup Language

A markup language used for encoding documents in a format that is both human-readable and machine-readable.

8. HTTP – HyperText Transfer Protocol

The protocol used for transferring web pages across the internet.

9. HTTPS – HyperText Transfer Protocol Secure

A secure version of HTTP that encrypts data exchanged between the client and server.

10. IDE – Integrated Development Environment

A software application that provides comprehensive facilities for software development, such as code editing, compiling, and debugging.

EXPLANATION:

The Client-Server Architecture is the foundation of distributed computing, where client devices interact with a centralized server that provides services or resources. In this system:

- **Client:** Initiates requests and interacts with the server (e.g., sending data to be printed).
- **Server:** Receives requests, processes data, and returns the results (e.g., handling the print queue, decrypting data).
- **Communication Protocols:** The server and client communicate using standard protocols such as TCP/IP (Transmission Control Protocol/Internet Protocol). TCP ensures reliable, ordered delivery of data, while IP handles the routing of packets across the network.

AES Encryption:

To ensure the confidentiality and security of sensitive data, the project uses AES (Advanced Encryption Standard), a symmetric encryption algorithm:

Symmetric Encryption: The same key is used for both encryption and decryption. This allows efficient processing and secure communication.

Encryption Process: Data is encrypted using a secret key and then sent to the server. Only the server with the correct key can decrypt the data and process it.

AES Block Cipher: It encrypts data in fixed-size blocks (128 bits) and supports multiple key lengths (128, 192, 256 bits), making it highly secure and suitable for large-scale data protection.

Print Job Management:

The print system utilizes a Job Queue to handle multiple print requests from different clients

Job Queue: Clients are assigned a position in the print queue based on the order of job submission. The server processes jobs one at a time, ensuring fair distribution and avoiding overloading the printer.

Queue Position & Wait Time: Each client is informed of their position in the queue and the estimated wait time. The wait time is based on the job size and the number of jobs ahead in the queue.

Print Availability Check: The server checks if a printer is connected and available before sending the print job. If no printer is available, the client receives a job failure message.

Data Flow and Security:

1. Data Encryption: Clients send encrypted files to the server using AES, ensuring the confidentiality of the data.

2. Server Processing: The server decrypts the data, adds the job to the print queue, and checks for printer availability.

3.Print Job Execution: If the printer is available, the server sends the job to the printer; otherwise, the job is flagged as failed, and a message is sent to the client.

Importance of Encryption in Networked Environments:

Encryption plays a critical role in network security by:

Protecting Sensitive Data: Ensuring that sensitive files (e.g., print documents) are unreadable to unauthorized users.

Data Integrity: Preventing unauthorized alterations to data during transmission.

Authentication: Verifying the identity of users and systems involved in the communication, ensuring only authorized parties can access or process the data.

Extended Features for Future Development:

- Integration with advanced print management systems to support features like color printing, double-sided printing, and paper size selection.
- A Graphical User Interface (GUI) for both clients and the server, allowing users to monitor print jobs, adjust settings, and manage queues more intuitively.

IMPLEMENTATION:

1. Multiple Client Support (Multithreading)

To handle multiple clients simultaneously, we can use multithreading. Each client will be processed by a separate thread.

Server Code (Multithreading):

```
import java.io.*;

import java.net.*;

import java.util.concurrent.*;

public class PrintServer {

   private static final int PORT = 12345;

   private static ExecutorService clientHandlerPool =
Executors.newFixedThreadPool(10); // Max 10 clients

public static void main(String[] args) {

      try (ServerSocket serverSocket = new ServerSocket(PORT)) {

         System.out.println("Server started on port " + PORT);

while (true) {

            Socket clientSocket = serverSocket.accept();

            System.out.println("New client connected: " +
clientSocket.getInetAddress());
```

```
                clientHandlerPool.submit(new      ClientHandler(clientSocket));      //
Handle client in a new thread
            }
        } catch (IOException e) {
            System.err.println("Server error: " + e.getMessage());
        }
    }
}
```

ClientHandler (Multithreaded Client Handling):

```
import java.io.*;
import java.net.*;
public class ClientHandler implements Runnable {
    private Socket clientSocket;
    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }
@Override
    public void run() {
        try {
            // Handle client communication
            BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(),
true); String clientMessage;
            while ((clientMessage = reader.readLine()) != null) {
                System.out.println("Received from client: " + clientMessage);
                // Process print job here (e.g., adding to queue, printing)
                writer.println("Job received: " + clientMessage); // Send response
back to client
            }
 } catch (IOException e) {
            System.err.println("Error handling client: " + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
                System.err.println("Error closing client socket: " + e.getMessage());
            }
        }
    }
```

2. Asynchronous Printing (Non-blocking)

Use ExecutorService to handle printing in the background.

PrintJob Handler (Asynchronous Printing):

```
import java.util.concurrent.*;
public class PrintJobHandler {
    private static ExecutorService printExecutor =
Executors.newSingleThreadExecutor();


    public static void submitPrintJob(String jobData) {
        printExecutor.submit(() -> {
            try {
                // Simulate long-running print job
                Thread.sleep(2000); // Simulate printing time
                System.out.println("Printing job: " + jobData + " completed successfully.");
            } catch (InterruptedException e) {
                System.err.println("Print job interrupted: " + e.getMessage());
            }
        }
}
```

Client Handler Update (Asynchronous Printing Call):

```
// Inside ClientHandler's run() method:
PrintJobHandler.submitPrintJob(clientMessage); // Submit print job asynchronously
```

3. Enhanced Queue Management (Priority Queueing)

Add a priority queue to manage print jobs based on their priority (for instance, urgent documents get higher priority).

Queue Management (Priority Queue):

```
import java.util.*;
public class PrintQueue {
    private static Queue<PrintJob> jobQueue = new PriorityQueue<>(Comparator.comparingInt(PrintJob::getPriority));
    public static void addJob(PrintJob job) {
        jobQueue.offer(job);
        System.out.println("Added job to queue: " + job.getJobId());
    }

    public static PrintJob getNextJob() {
        return jobQueue.poll();
    }
}
class PrintJob {
    private String jobId;
    private String jobData;
    private int priority; // Priority level: Lower number = higher priority

    public PrintJob(String jobId, String jobData, int priority) {
        this.jobId = jobId;
        this.jobData = jobData;

        this.priority = priority;

    }

public String getJobId() {

        return jobId;

    }

 public int getPriority() {

        return priority;

    }

}
```

Update ClientHandler to Add Job to Queue:

```
PrintJob job = new PrintJob(clientMessage, clientMessage, 1); // Job with highest priority

PrintQueue.addJob(job);
```

4. Error Handling and Validation

Implement error handling for invalid input files and encryption issues.

Error Handling (Validation):

```
public class InputValidator {
    public static boolean validateFile(String filename) {
        // Check if the file exists and is of a supported format (e.g., .txt)
        if (filename == null || !filename.endsWith(".txt")) {
            System.err.println("Invalid file format. Only .txt files are supported.");
            return false;
        }
        return true;
    }
}
```

ClientHandler Update to Validate Input:

```
if (!InputValidator.validateFile(clientMessage)) {
    writer.println("Invalid file format. Only .txt files are supported.");
    return;
}
```

5. Logging and Timestamps

Add logging and timestamps to keep track of when each job starts and finishes.

Logger Implementation:

```
import java.text.SimpleDateFormat;

import java.util.Date;

public class Logger {

    public static void log(String message) {

        String timestamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());

        System.out.println("[" + timestamp + "] " + message);

    }

}
```

Log Job Start/Completion:

```
Logger.log("Job started: " + clientMessage);

PrintJobHandler.submitPrintJob(clientMessage);

Logger.log("Job completed: " + clientMessage);
```

6. User Interface Enhancements

For simplicity, let’s enhance the client’s user interface by showing a progress bar and a job history.

Client UI Update with Progress Bar:

```
import javax.swing.*;
```

```
public class ClientUI {
    private static JProgressBar progressBar;
    public static void showProgressBar() {
        JFrame frame = new JFrame("Print Job Progress");
        progressBar = new JProgressBar(0, 100);
        progressBar.setValue(0);
        progressBar.setStringPainted(true);
        frame.add(progressBar);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
// Simulate job progress
        for (int i = 0; i <= 100; i++) {
            progressBar.setValue(i);
            try {
                Thread.sleep(50); // Simulate printing progress
            } catch (InterruptedException e) {
```

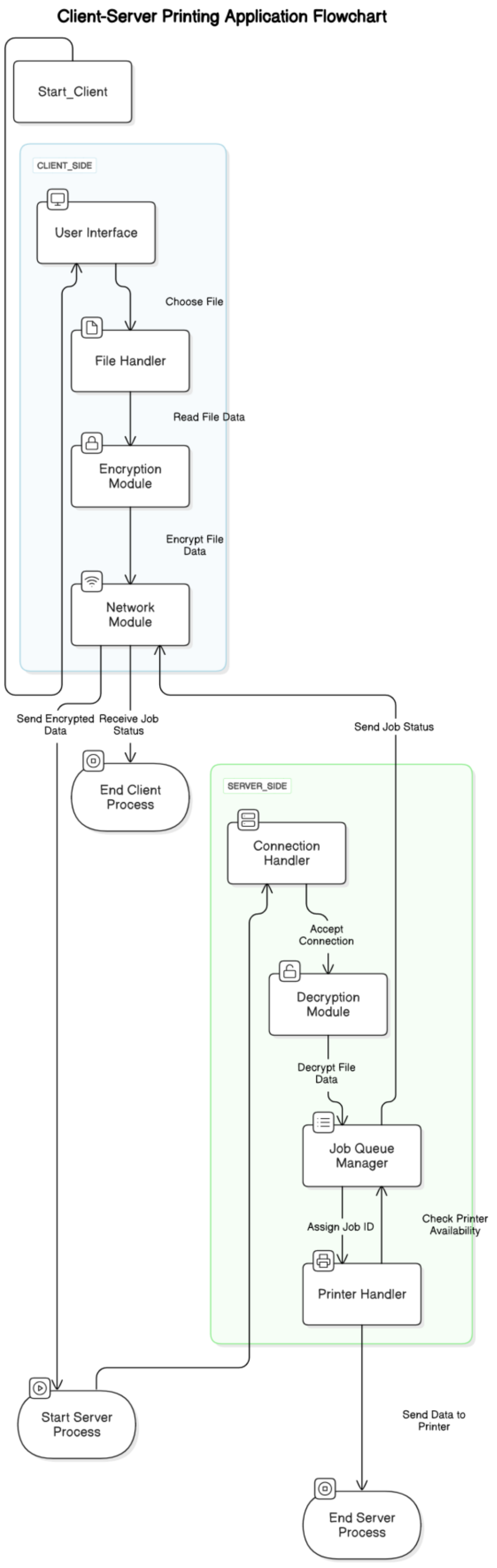
```
            e.printStackTrace();
          }
       }
   }
 public static void main(String[] args) {
       showProgressBar();
   }
}
```

Job History Update:

```
// Update client with job status (stored in a text area, for example)
JTextArea jobHistory = new JTextArea();
jobHistory.append("Job started: " + clientMessage + "\n");
jobHistory.append("Job completed: " + clientMessage + "\n");
```

ARCITECTURE DIAGRAM:

The architecture diagram emphasizes modular organization, clear data flow, and separation of responsibilities in the client-server communication, encryption, and print handling processes.


Client-Server Printing Application Flowchart
Start_Client
CLIENT_SIDE
User Interface
Choose File
File Handler
Read File Data
Encryption Module
Encrypt File Data
Network Module
Send Encrypted Data
Receive Job Status
Send Job Status
End Client Process
SERVER_SIDE
Connection Handler
Accept Connection
Decryption Module
Decrypt File Data
Job Queue Manager
Assign Job ID
Check Printer Availability
Printer Handler
Start Server Process
Send Data to Printer
End Server Process

Description of the Components

1. User Interface (Client):

- Provides the user with a simple GUI to interact with the application, with options to choose a file and initiate a print job.

2.File Handler (Client):

- Responsible for reading the file chosen by the user to prepare it for encryption and sending.

3. Encryption Module (Client):

- Encrypts file data using AES encryption before sending it to the server.

4. Network Module (Client):

- Manages the communication with the server, sending encrypted data and receiving job status responses.

5. Connection Handler (Server):

- Accepts incoming connections from clients and maintains the connection while handling multiple jobs.

6. Decryption Module (Server):

- Decrypts the received file data on the server side, making it ready for printing.

7. *Job Queue Manager (Server)*:

- Handles the queuing and ordering of print jobs, assigns job IDs, and estimates wait times for each client. Sends updates on job status back to the client.

8. Printer Handler (Server):

- Checks if the printer is connected and available, then sends decrypted data to the printer. Reports success or failure based on printer status.

Explanation of the Data Flow

1. The Client side starts the process when the user selects a file, which the File Handler reads and the Encryption Module encrypts.

2. The Network Module on the client sends the encrypted file data to the server.

3. The server side Connection Handler accepts the data, then uses the Decryption Module to decrypt it.

4. The Job Queue Manager assigns job IDs, estimates wait times, and updates clients on their status.

5. Finally, if the printer is available, the *Printer Handler* sends the decrypted data for printing, completing the job process.

FULL IMPLEMNETATION:

Client :

```
import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.net.*;

import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.io.*;

import java.awt.print.*;

import java.util.Base64;

public class Client {

    private static final int BUF_SIZE = 1024;

    private static SecretKey secretKey;

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

```
 try {
    // Generate AES Key
    secretKey = generateAESKey();
 } catch (Exception e) {
    e.printStackTrace();
    return;
 }
JFrame frame = new JFrame("Print Client");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(400, 300);
JButton chooseFileButton = new JButton("Choose File to Print");
 JButton printButton = new JButton("Print Decrypted Data");
 JTextArea encryptedTextArea = new JTextArea(10, 30);
 encryptedTextArea.setEditable(false);
 JScrollPane scrollPane = new JScrollPane(encryptedTextArea);
 chooseFileButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
        int returnValue = fileChooser.showOpenDialog(null);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();


         sendFileToServer(selectedFile, encryptedTextArea);
        }
    }
 });
printButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        printDecryptedText();
    }
 });
frame.getContentPane().setLayout(new FlowLayout());
```

```
        frame.getContentPane().add(chooseFileButton);

        frame.getContentPane().add(printButton);

        frame.getContentPane().add(scrollPane);

        frame.setVisible(true);

   }     private     static     void     sendFileToServer(File     file,     JTextArea
encryptedTextArea) {

        try (Socket socket = new Socket("127.0.0.1", 8087);

            OutputStream out = socket.getOutputStream();

            BufferedInputStream     fileIn     =     new     BufferedInputStream(new
FileInputStream(file));

            InputStream in = socket.getInputStream()) {

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

 // Read file and encrypt

           byte[] buffer = new byte[BUF_SIZE];

           int bytesRead;

           StringBuilder encryptedData = new StringBuilder();

while ((bytesRead = fileIn.read(buffer)) != -1) {

              byte[] encrypted = aesEncrypt(buffer, bytesRead);
```

```
encryptedData.append(Base64.getEncoder().encodeToString(encrypted));

            }

// Display encrypted data in JTextArea

            encryptedTextArea.setText(encryptedData.toString(

// Send encrypted data to the server

            out.write(encryptedData.toString().getBytes());

            out.flush();

System.out.println("Encrypted file sent to server.");

 // Receive response from the server

            byte[] responseBuffer = new byte[BUF_SIZE];

            int responseBytes = in.read(responseBuffer);

            String responseMessage = new String(responseBuffer, 0,
responseBytes);

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



        } catch (IOException e) {

            e.printStackTrace();
        }

    }

 private static byte[] aesEncrypt(byte[] data, int size) {

        try {

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

            cipher.init(Cipher.ENCRYPT_MODE, secretKey);

            return cipher.doFinal(data, 0, size);

        } catch (Exception e) {

            e.printStackTrace();

        }

        return new byte[0];

    }]

 private static void printDecryptedText() {

        // For now, you can print decrypted text. For actual decryption logic, add
server-side decryption

        System.out.println("Decrypted text to print.");

    }

private static SecretKey generateAESKey() throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128); // Use 128 bit key for AES
        return keyGen.generateKey();
    }
```

SERVER:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
import java.io.*;
import java.net.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantLock;
import javax.print.*;
import javax.print.attribute.*;
import javax.print.attribute.standard.Copies;
```

```
public class Server {

   private static final int PORT = 8087;

   private static final int BUF_SIZE = 1024;

   private static AtomicInteger nextJobId = new AtomicInteger(1); // Global
job ID counter

   private static AtomicInteger currentJobInQueue = new AtomicInteger(1); //
Track the current job position in queue

   private static final ReentrantLock lock = new ReentrantLock(); // Lock for
thread safety

   private static final String AES_KEY = "1234567890123456"; // 16-byte key
for AES-128

public static void main(String[] args) {

       try (ServerSocket serverSocket = new ServerSocket(PORT)) {

          System.out.println("Server listening for clients on port " + PORT + "...");

  while (true) {

              Socket clientSocket = serverSocket.accept();

              int clientJobId = nextJobId.getAndIncrement(); // Unique job ID for
each client
```

```
                System.out.println("Client                connected:                 "                 +
clientSocket.getInetAddress() + " (Job " + clientJobId + ")");

               // Handle each client in a new thread

                new Thread(new ClientHandler(clientSocket, clientJobId)).start();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

static class ClientHandler implements Runnable {

        private Socket clientSocket;

        private int clientJobId;

 public ClientHandler(Socket socket, int jobId) {

            this.clientSocket = socket;

            this.clientJobId = jobId;

        }

     @Override

        public void run() {
```

```
        try (BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            OutputStream out = clientSocket.getOutputStream()) {

 // Get the client's position in the queue and increment the queue position
for the next client

            int positionInQueue = currentJobInQueue.getAndIncrement();

            int waitTimeEstimate = (positionInQueue - 1) * 2; // Assume each job
takes 2 seconds

// Send queue message to the client

            String queueMessage = "You are job " + positionInQueue + " in the
print queue. Estimated wait time: " + waitTimeEstimate + " seconds.\n";

            out.write(queueMessage.getBytes());

            out.flush();

String startTime = getTimestamp();

            System.out.println("Job " + clientJobId + " started at " + startTime);

  // Read encrypted data from client, decrypt it, and store in a buffer

            char[] buffer = new char[BUF_SIZE];

            int bytesRead;
            StringBuilder decryptedData = new StringBuilder();
  while ((bytesRead = in.read(buffer)) != -1) {
               // Decrypt the received data using AES
               decryptedData.append(aesDecrypt(new String(buffer, 0,
bytesRead)));
            }
System.out.println("Sending data for Job " + clientJobId + " to printer...");
            boolean printerAvailable =
sendToPrinter(decryptedData.toString()); // Check printer availability
 // Send job status based on printer availability
            String jobStatusMessage;
            if (printerAvailable) {
               System.out.println("Data for Job " + clientJobId + " sent to
printer.");
               jobStatusMessage = "Print job " + clientJobId + " completed
successfully.\n";
            } else {
               System.out.println("Job " + clientJobId + " failed: Printer not
connected.");
```

```
                    jobStatusMessage = "Print job " + clientJobId + " failed: Printer not
connected.\n" }
                out.write(jobStatusMessage.getBytes());
                out.flush();
String endTime = getTimestamp();
                System.out.println("Job " + clientJobId + " ended at " + endTime);
 lock.lock();
                try {
                    currentJobInQueue.decrementAndGet();   //   Decrement   queue
counter once job is completed
                } finally {
                    lock.unlock();
                }
 } catch (IOException e) {
                e.printStackTrace();
                 try {
                clientSocket.close();
                } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    }
  }
 // AES decryption method
  private static String aesDecrypt(String encryptedText) {
    try {
      SecretKeySpec   keySpec   =   new   SecretKeySpec(AES_KEY.getBytes(),
"AES");
      Cipher cipher = Cipher.getInstance("AES");
      cipher.init(Cipher.DECRYPT_MODE, keySpec);
     byte[] decodedText = encryptedText.getBytes("UTF-8");
      byte[] decryptedText = cipher.doFinal(decodedText);

      return new String(decryptedText, "UTF-8");
    } catch (Exception e) {
      e.printStackTrace();
```

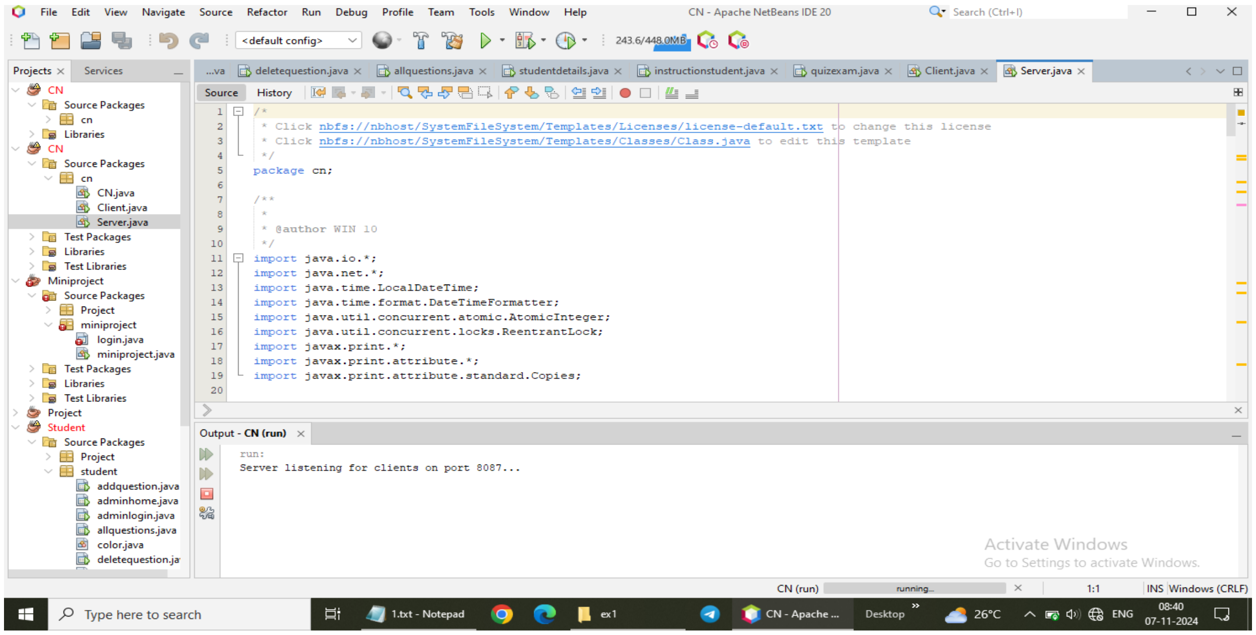
```
            return null;
        }
    }
 // Method to open print dialog and send the decrypted text to the printer
    private static boolean sendToPrinter(String decryptedData) {
        // Save the decrypted data to a temporary file
        File tempFile = new File("decrypted_document.txt");
        try (FileWriter writer = new FileWriter(tempFile)) {
            writer.write(decryptedData);
        } catch (IOException e) {
            e.printStackTrace();
            return false; // Failed to save decrypted data
        }
// Open print dialog and send file to printer
        try (FileInputStream fis = new FileInputStream(tempFile)) {
            DocFlavor flavor = DocFlavor.INPUT_STREAM.AUTOSENSE;
            PrintRequestAttributeSet attrs = new HashPrintRequestAttributeSet();
```

```
          attrs.add(new Copies(1));  // Specify number of copies
 PrintService[] printServices = PrintServiceLookup.lookupPrintServices(flavor,
attrs);
          if (printServices.length == 0) {
             System.out.println("No printers are available.");
             return false;
          }
 PrintService    selectedService    =    ServiceUI.printDialog(null,    200,    200,
printServices, null, flavor, attrs);
          if (selectedService == null) {
             System.out.println("Print job cancelled by the user.");
             return false; // User cancelled the print dialog
          }
        DocPrintJob job = selectedService.createPrintJob();
          Doc doc = new SimpleDoc(fis, flavor, null);   job.print(doc, attrs);
   System.out.println("Print      job      successfully      sent      to:      "      +
selectedService.getName());
 return true;
```

```
        } catch (IOException | PrintException e) {
            e.printStackTrace();
            return false; // Print job failed due to an error
        } finally {
            // Optionally, delete the temporary file after printing
            tempFile.delete();
        }
    }
    // Method to get current timestamp
    private static String getTimestamp() {
        DateTimeFormatter  dtf  =  DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");
        return dtf.format(LocalDateTime.now());
    }}
```

SCREENSHOTS

SERVER

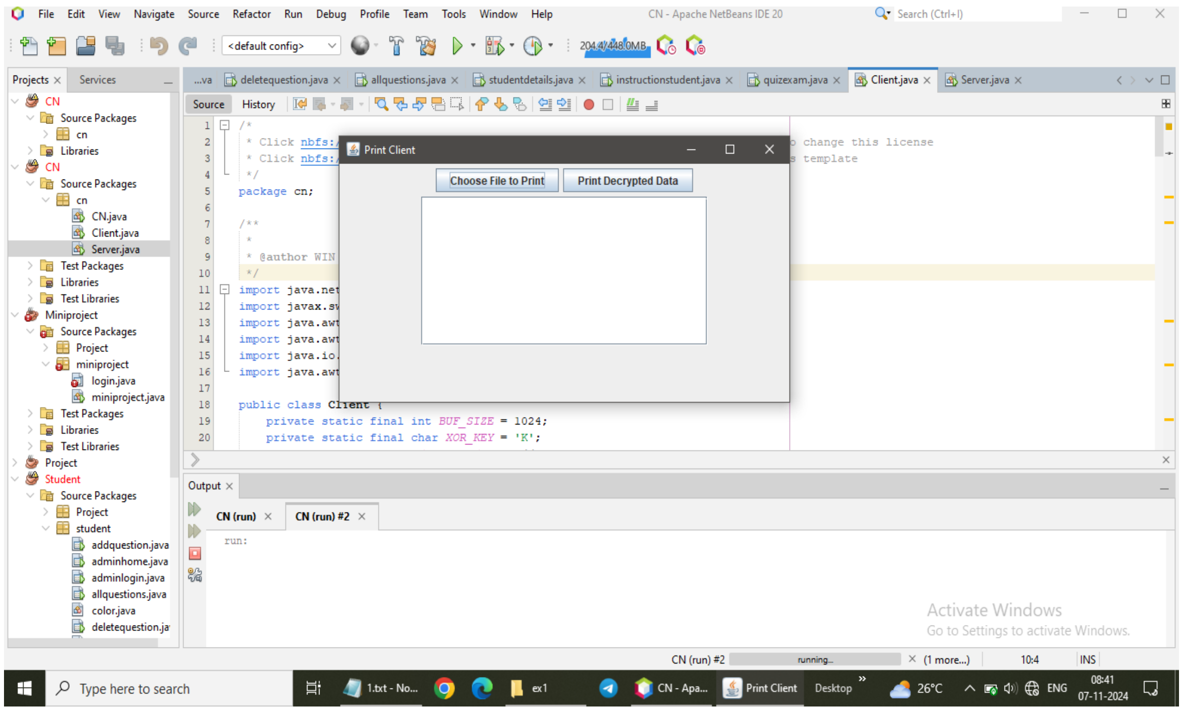


```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cn;

/**
 *
 * @author WIN 10
 */
import java.io.*;
import java.net.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantLock;
import javax.print.*;
import javax.print.attribute.*;
import javax.print.attribute.standard.Copies;
```

```
run:
Server listening for clients on port 8087...
```

CLIENT



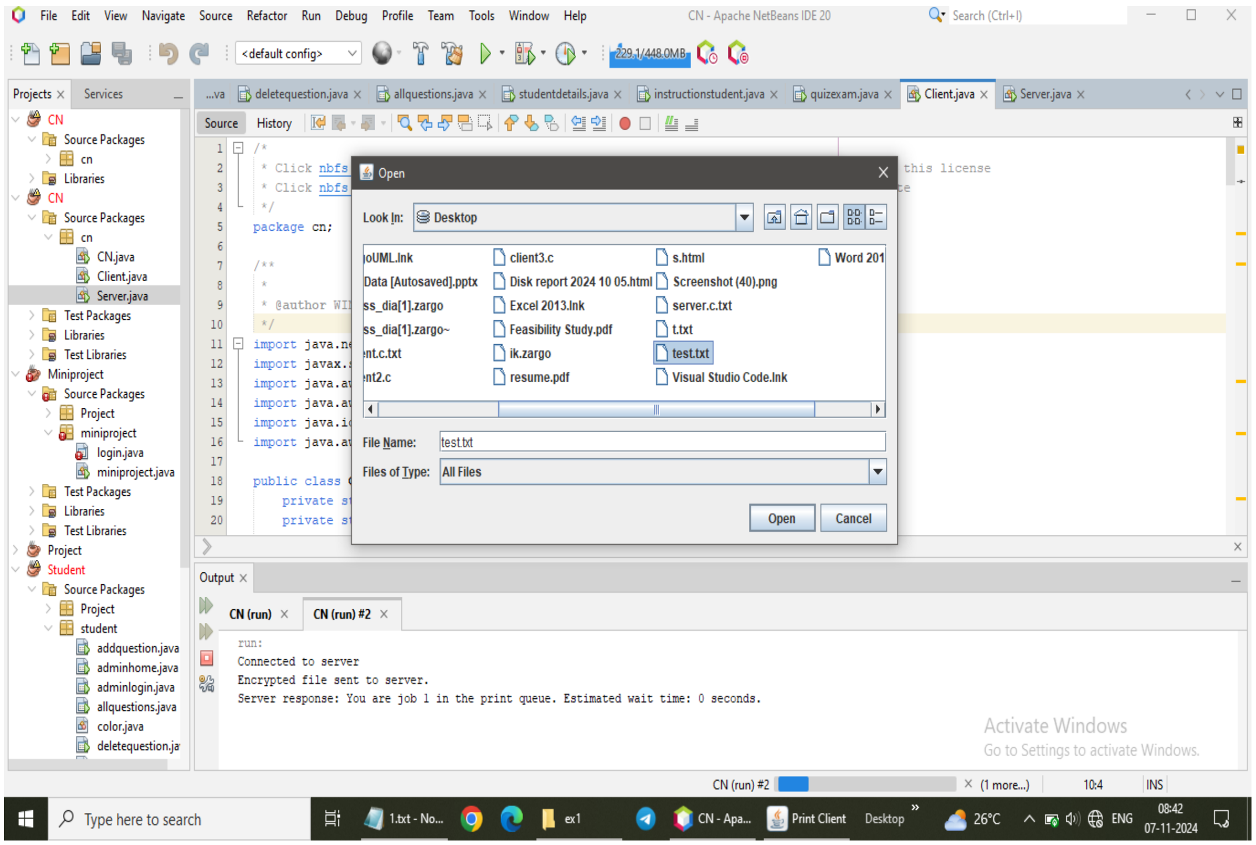
```
/*
 * Click nbfs:/
 * Click nbfs:/
 */
package cn;

/**
 *
 * @author WIN
 */
import java.net
import javax.sw
import java.awt
import java.awt
import java.io.
import java.awt

public class Client {
    private static final int BUF_SIZE = 1024;
    private static final char XOR_KEY = 'K';
```

```
run:
```

CHOOSING FILE PATH



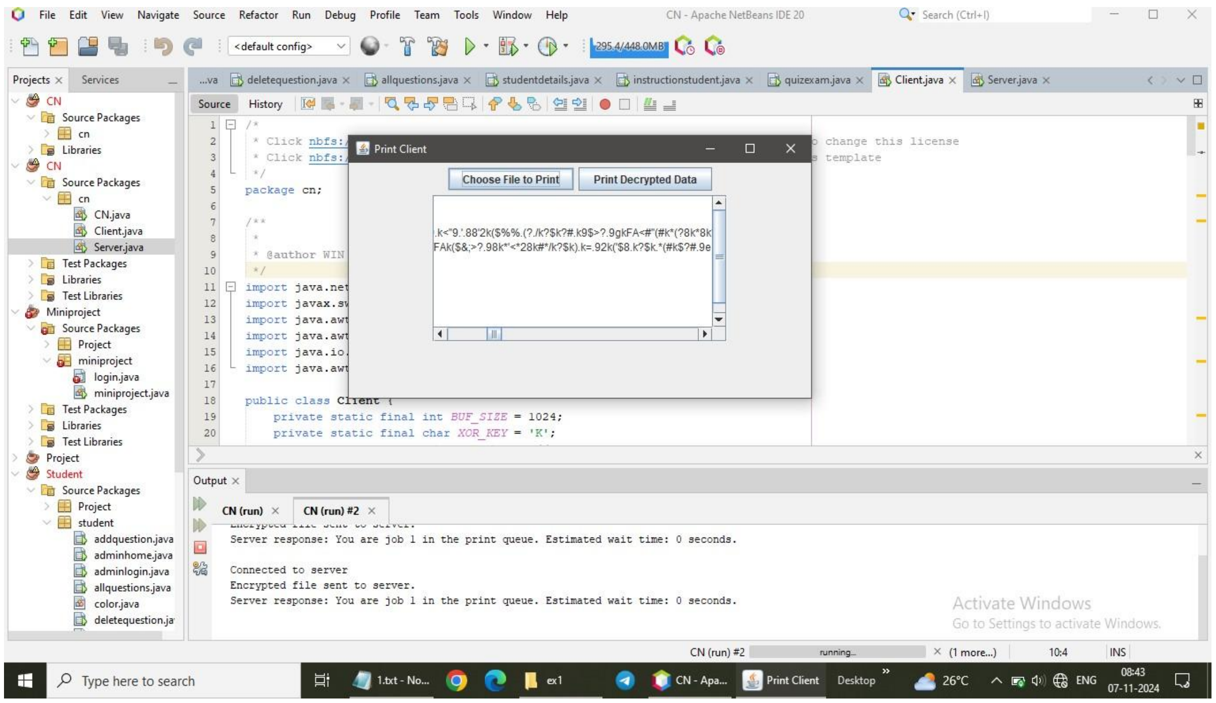
Look In: Desktop

File Name: test.txt

Files of Type: All Files

```
run:
Connected to server
Encrypted file sent to server.
Server response: You are job 1 in the print queue. Estimated wait time: 0 seconds.
```

ENCRYPTED TEXT



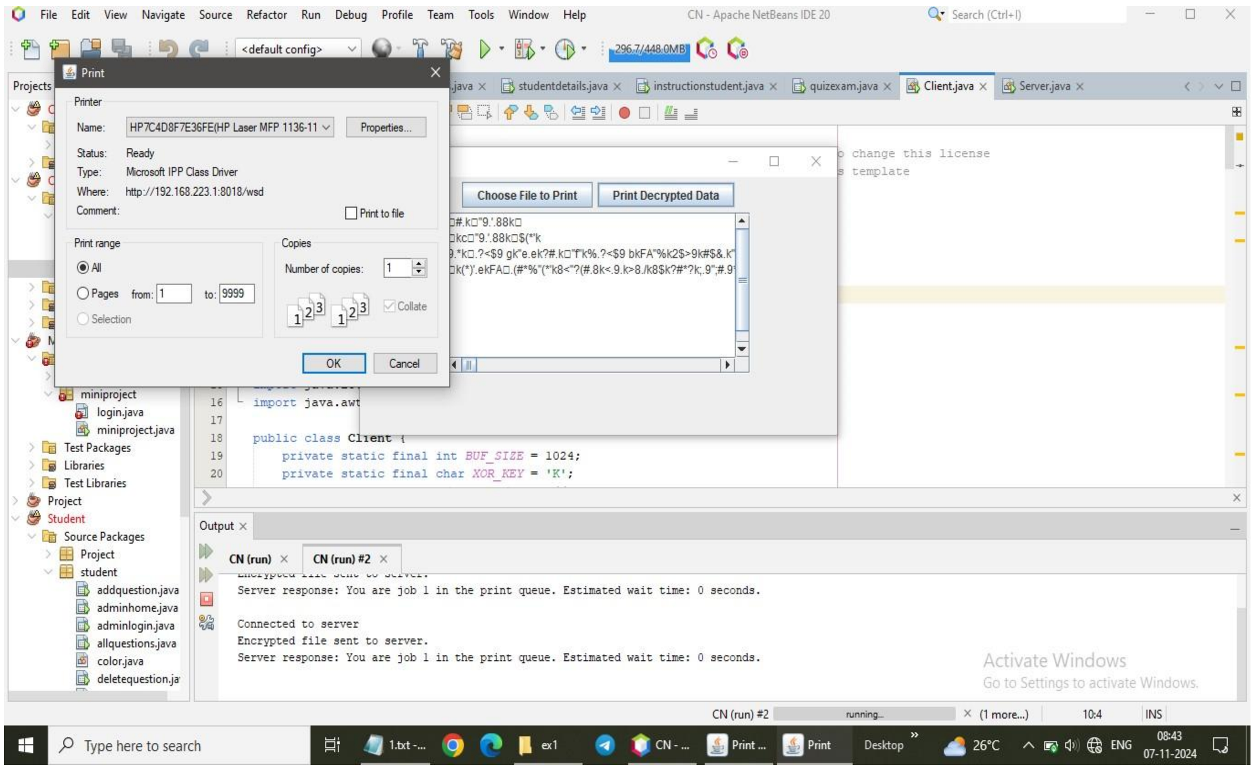
```
.k<"9.'.88"2k($%%.(?./k?$k?#.k9$>?.9gkFA<#"(#k*(?8k*8k
FAk($&;>?.98k*"<*28k#*/k?$k).k=.92k("$8.k?$k.*(#k$?#.9e
```

```
private static final int BUF_SIZE = 1024;
private static final char XOR_KEY = 'K';
```

```
Server response: You are job 1 in the print queue. Estimated wait time: 0 seconds.

Connected to server
Encrypted file sent to server.
Server response: You are job 1 in the print queue. Estimated wait time: 0 seconds.
```

PRINTER CONNECTED



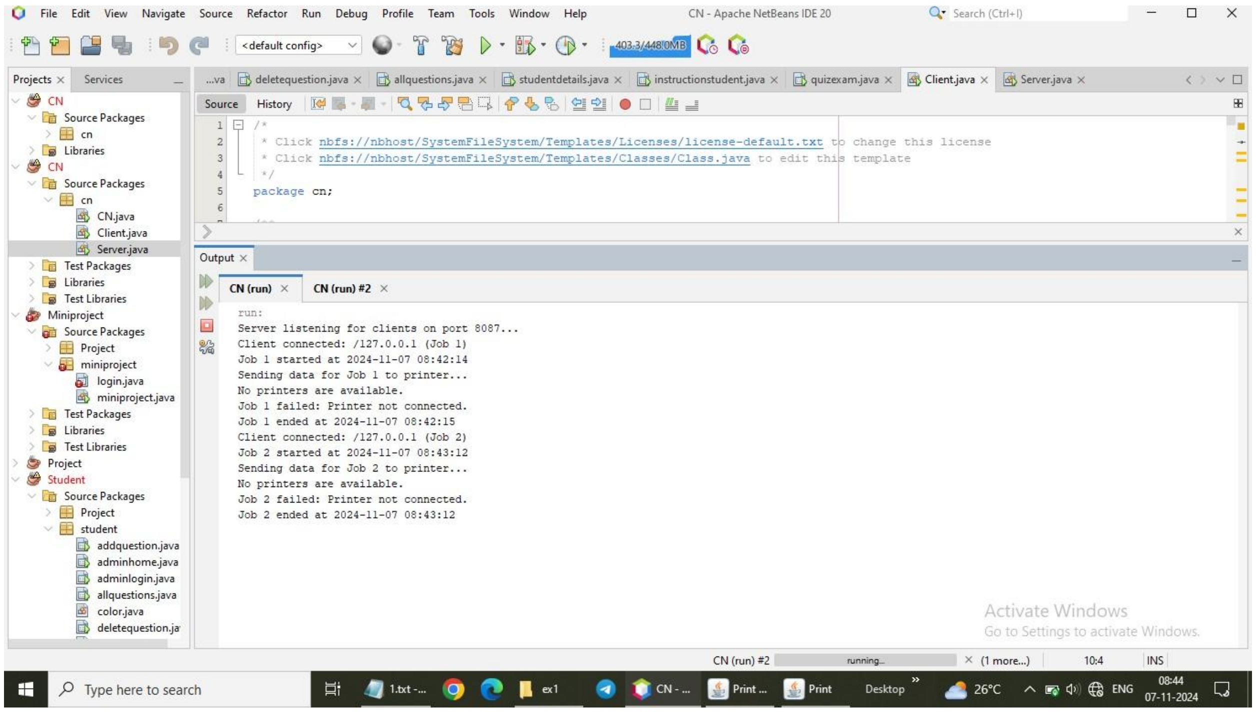
Printer
Name: HP7C4D8F7E36FE(HP Laser MFP 1136-11
Status: Ready
Type: Microsoft IPP Class Driver
Where: http://192.168.223.1:8018/wsd
Print range: All
Pages from: 1 to: 9999
Number of copies: 1

```java
import java.awt
public class Client {
    private static final int BUF_SIZE = 1024;
    private static final char XOR_KEY = 'K';
```

```
Server response: You are job 1 in the print queue. Estimated wait time: 0 seconds.

Connected to server
Encrypted file sent to server.
Server response: You are job 1 in the print queue. Estimated wait time: 0 seconds.
```

PRINTER NOT CONNECTED



```
run:
Server listening for clients on port 8087...
Client connected: /127.0.0.1 (Job 1)
Job 1 started at 2024-11-07 08:42:14
Sending data for Job 1 to printer...
No printers are available.
Job 1 failed: Printer not connected.
Job 1 ended at 2024-11-07 08:42:15
Client connected: /127.0.0.1 (Job 2)
Job 2 started at 2024-11-07 08:43:12
Sending data for Job 2 to printer...
No printers are available.
Job 2 failed: Printer not connected.
Job 2 ended at 2024-11-07 08:43:12
```

Conclusion:

The project successfully demonstrates a secure and efficient client-server communication system using AES encryption, which is crucial in today's networked environments to ensure data confidentiality and integrity. By implementing core concepts such as socket programming, multithreading, and cryptographic encryption, this project serves as a model for real-world applications in secure data transfer and networked printing solutions. The project emphasizes the importance of robust error handling, effective resource management, and user-friendly interfaces, making it a valuable learning tool for understanding secure network protocols and data management practices. Additionally, it highlights the growing need for secure communication in distributed systems and provides foundational knowledge for future enhancements in data security, scalability, and performance optimization.

REFRENCES:

Oracle Java Documentation – "Java SE Documentation," Oracle, https://docs.oracle.com/javase/8/docs/.

National Institute of Standards and Technology (NIST), "Advanced EncryptionStandard(AES)",https://csrc.nist.gov/publications/detail/fips/197/final.