



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

Vaibhav Khandelwal

21BCE5550

TABLE OF CONTENTS

Exp. No.	Date	Exp. Name	Page Nos.
1	08-01-2024	Implementation of Classical Cryptography	1-21
2	15-01-2024	Implementation of Number Theory Concepts	22-30
3	22-01-2024	Implementation of DES	31-37
4	29-01-2024	Implementation of AES	38-43
5	05-02-2024	Implementation of RSA Cryptosystem	44-50
6	19-02-2024	Implementation of Elgamal Cryptosystem	51-56
7	26-02-2024	Implementation of Elliptic Curve Cryptosystem	57-63
8	04-03-2024	Implementation of Diffie-Hellman Key Exchange Protocol	64-73
9	11-03-2024	Implementation of Secure Hash Algorithm (SHA)	74-77
10	16-03-2024	Message Digest-5 Algorithm	78-81
11	08-04-2024	Implementation of Digital Signature	82-86
12	15-04-2024	Implementation of SSL Socket Communication	87-96
13	22-04-2024	Web Application – JSON Web Token	97-102



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab Experiment-1

Implementation of Classical Cryptography

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date- 08-01-2024

Question 1: -

Write a socket program to demonstrate Caesar Cipher.

Server Code

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

public class CaesarServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234); // Replace with desired port
        Socket clientSocket = serverSocket.accept();

        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

        String text = in.readLine();
        int shift = Integer.parseInt(in.readLine());

        String encryptedText = encrypt(text, shift);
        out.println(encryptedText);
    }
}
```

```

        clientSocket.close();
        serverSocket.close();
    }

    public static String encrypt(String text, int shift) {
        StringBuilder result = new StringBuilder();
        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char base = Character.isUpperCase(ch) ? 'A' : 'a';
                char encrypted = (char) ((ch - base + shift) % 26 + base);
                result.append(encrypted);
            } else {
                result.append(ch);
            }
        }
        return result.toString();
    }
}

```

Client Code

```

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

public class CaesarClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 1234); // Replace with server address and port

        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        Scanner scanner = new Scanner(System.in);
    }
}

```

```

        System.out.print("Enter text: ");

        String text = scanner.nextLine();

        System.out.print("Enter shift value: ");

        int shift = scanner.nextInt();


        out.println(text);

        out.println(shift);


        String encryptedText = in.readLine();

        System.out.println("Encrypted text: " + encryptedText);


        socket.close();
    }
}

```

Algorithm

Server-Side Algorithm:

Create a Server Socket: Binds to a specific port to listen for incoming client connections.

Accept a client connection: Waits for a client to initiate a connection and accepts it.

Receive text and shift value from the client: Reads the text to be encrypted and the shift value from the client's input stream.

Encrypt the text: Calls the encrypt function to perform the Caesar Cipher encryption using the given shift value.

Send the encrypted text back to the client: Writes the encrypted text to the client's output stream.

Close the connections: Closes the client socket and the server socket.

Client-Side Algorithm:

Create a Socket: Connects to the server at the specified IP address and port.

Send text and shift value to the server: Prompts the user for the text to encrypt and the shift value, then sends them to the server's output stream.

Receive the encrypted text from the server: Reads the encrypted text from the server's input stream.

Display the encrypted text: Prints the encrypted text to the console.

Close the connection: Closes the socket.

Encryption Algorithm (encrypt function):

Iterate through each character in the text: Processes each character in the input string.

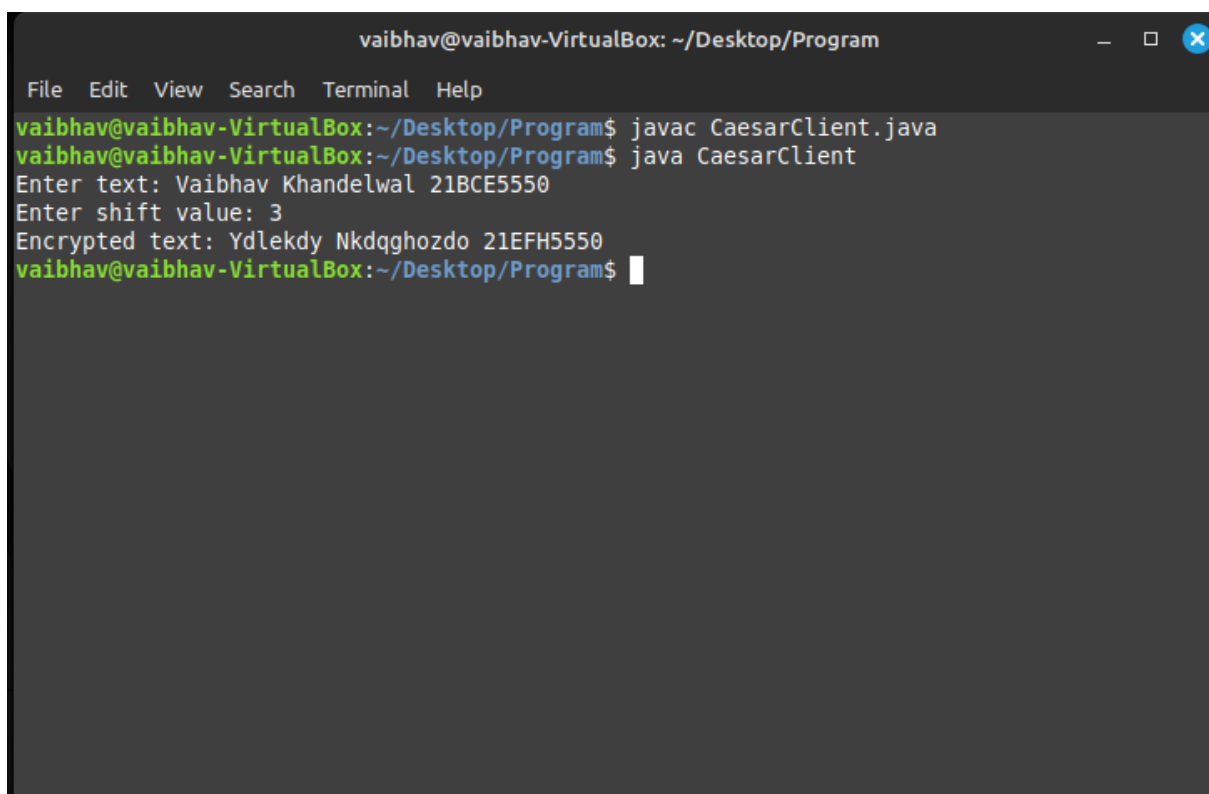
Check if the character is a letter: Only encrypts letters (uppercase or lowercase).

Calculate the encrypted character: Shifts the character's position in the alphabet by the given shift value, wrapping around if necessary.

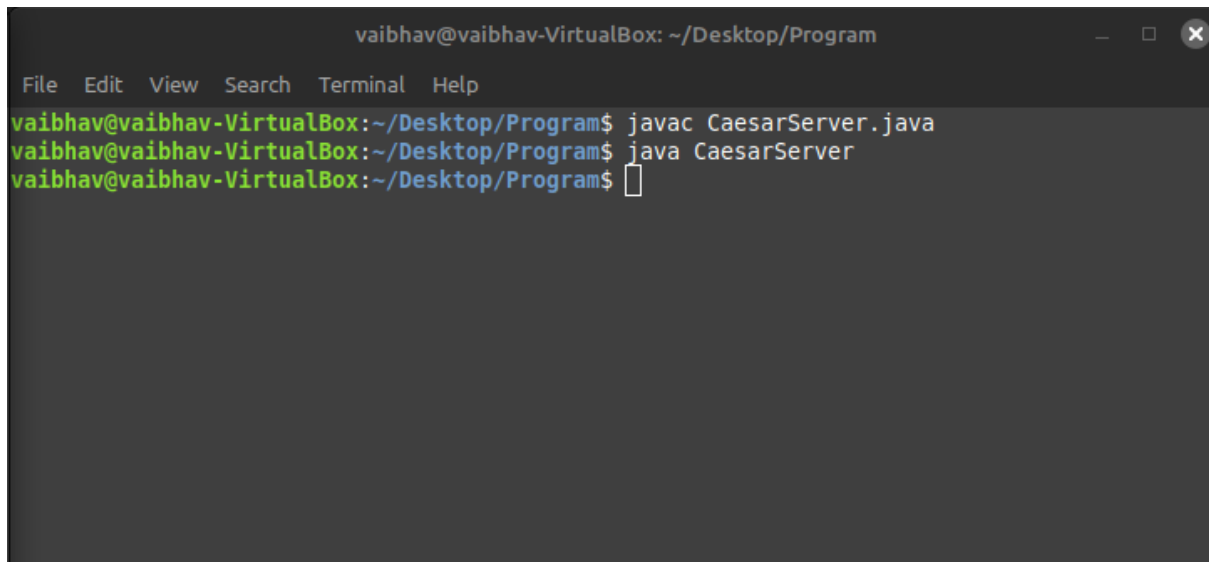
Append the encrypted character to the result string: Builds the encrypted text string.

Return the encrypted text: Returns the final encrypted string.

Output



```
vaibhav@vaibhav-VirtualBox: ~/Desktop/Program
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac CaesarClient.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java CaesarClient
Enter text: Vaibhav Khandelwal 21BCE5550
Enter shift value: 3
Encrypted text: Ydleky Nkdqghozdo 21EFH5550
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

A screenshot of a terminal window titled "vaibhav@vaibhav-VirtualBox: ~/Desktop/Program". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows three lines of commands: "javac CaesarServer.java", "java CaesarServer", and a blank line with a cursor. The prompt "vaibhav@vaibhav-VirtualBox:~/Desktop/Program\$" is visible on each line.

```
vaibhav@vaibhav-VirtualBox: ~/Desktop/Program
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac CaesarServer.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java CaesarServer
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

Question 2: -

Write a socket program to demonstrate Rail Fence Cipher

Server Code

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

public class Server {

    // Function to encrypt the message using Rail Fence Cipher
    private static String railFenceEncrypt(String text, int rails) {

        StringBuilder[] fence = new StringBuilder[rails];

        for (int i = 0; i < rails; i++) {

            fence[i] = new StringBuilder();

        }

        int rail = 0;

        boolean down = true;
```

```

for (char ch : text.toCharArray()) {
    fence[rail].append(ch);

    if (rail == 0) {
        down = true;
    } else if (rail == rails - 1) {
        down = false;
    }

    rail += down ? 1 : -1;
}

StringBuilder result = new StringBuilder();
for (StringBuilder railStr : fence) {
    result.append(railStr);
}

return result.toString();
}

// Function to decrypt the message using Rail Fence Cipher
private static String railFenceDecrypt(String text, int rails) {
    char[][] fence = new char[rails][text.length()];

    for (int i = 0; i < rails; i++) {
        for (int j = 0; j < text.length(); j++) {
            fence[i][j] = ' ';
        }
    }

    int rail = 0;
    boolean down = true;

```



```

    for (int i = 0; i < text.length(); i++) {
        fence[rail][i] = text.charAt(i);

        if (rail == 0) {
            down = true;
        } else if (rail == rails - 1) {
            down = false;
        }

        rail += down ? 1 : -1;
    }

    StringBuilder result = new StringBuilder();
    rail = 0;
    down = true;

    for (int i = 0; i < text.length(); i++) {
        result.append(fence[rail][i]);

        if (rail == 0) {
            down = true;
        } else if (rail == rails - 1) {
            down = false;
        }

        rail += down ? 1 : -1;
    }

    return result.toString();
}

public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(12345);

```

```

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

while (true) {
    Socket clientSocket = serverSocket.accept();
    System.out.println("Accepted connection from " + clientSocket.getInetAddress());

    BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    String message = in.readLine();
    int rails = Integer.parseInt(in.readLine());

    System.out.println("Received message: " + message);
    System.out.println("Received number of rails: " + rails);

    // Encrypt the message using Rail Fence Cipher
    String encryptedMessage = railFenceEncrypt(message, rails);
    System.out.println("Encrypted message: " + encryptedMessage);

    // Decrypt the message using Rail Fence Cipher
    String decryptedMessage = railFenceDecrypt(encryptedMessage, rails);
    System.out.println("Decrypted message: " + decryptedMessage);

    clientSocket.close();
}
}
}

```

Client Code

```

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

public class Client {

```

```

public static void main(String[] args) throws IOException {
    Socket clientSocket = new Socket("localhost", 12345);

    // Get user input for the message and the number of rails
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the message: ");
    String message = scanner.nextLine();

    System.out.print("Enter the number of rails: ");
    int rails = scanner.nextInt();

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

    // Send the message and the number of rails to the server
    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
    out.println(message);
    out.println(rails);

    clientSocket.close();
    scanner.close();
}
}

```

Algorithm

Server (Server.java):

Import Java Libraries: Import necessary Java libraries (java.io.*, java.net.*).

Define Rail Fence Cipher Functions: Implement railFenceEncrypt and railFenceDecrypt functions for encryption and decryption using the Rail Fence Cipher.

Main Server Logic: Create a ServerSocket on a specified port (e.g., 12345).

Enter an infinite loop to handle incoming client connections.

For each client connection: Accept the connection and obtain an input stream. Read the message and number of rails from the client. Encrypt the message using Rail Fence Cipher. Decrypt the message using Rail Fence Cipher. Close the client socket.

Client (Client.java):

Import Java Libraries: Import necessary Java libraries (java.io.*, java.net.*, java.util.Scanner).

User Input: Use Scanner to get user input for the message and the number of rails.

Connect to Server: Create a Socket to connect to the server (localhost, port 12345).

Send User Input to Server: Create an output stream to send the message and number of rails to the server.

Close Client Socket: Close the client socket after sending the data.

Execution:

Run the Server: Compile and run Server.java on one terminal to start the server listening on port 12345.

Run the Client: Compile and run Client.java on another terminal to connect to the server. Enter the message and the number of rails when prompted.

Server Processing: The server receives the message and number of rails. Encrypts the message using Rail Fence Cipher. Decrypts the message using Rail Fence Cipher. Outputs the original message, encrypted message, and decrypted message.

Client Exit: The client sends the message and number of rails to the server and then closes.

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Client.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Client
Enter the message: VaibhavKhandelwal21BCE5550
Enter the number of rails: 3
Original message: VaibhavKhandelwal21BCE5550
Number of rails: 3
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

```
at RailFenceServer.main(RailFenceServer.java:10)
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Server.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Server
Server listening on port 12345...
Accepted connection from /127.0.0.1
Received message: VaibhavKhandelwal21BCE5550
Received number of rails: 3
Encrypted message: VhhelC5abaKadla2BE50ivnw15
Decrypted message: VhhelC5abaKadla2BE50ivnw15
```

Question 3: -

Write a socket program to demonstrate Playfair Cipher.

Server Code

```

import java.io.*;

import java.net.*;

public class Server {

    private static char[][] keyMatrix;

    // Function to generate the Playfair key matrix
    private static void generateKeyMatrix(String key) {

        keyMatrix = new char[5][5];

        String keyString = prepareKey(key);

        int k = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                keyMatrix[i][j] = keyString.charAt(k++);
            }
        }
    }

    // Function to prepare the key (remove duplicate letters and replace 'J' with 'I')
    private static String prepareKey(String key) {

        key = key.toUpperCase().replaceAll("[^A-Z]", "");
        key = key.replace("J", "I");

        StringBuilder result = new StringBuilder();
        for (char ch : key.toCharArray()) {
            if (result.indexOf(String.valueOf(ch)) == -1) {
                result.append(ch);
            }
        }

        // Fill the matrix with remaining letters
    }

```

```

        for (char ch = 'A'; ch <= 'Z'; ch++) {
            if (ch != 'J' && result.indexOf(String.valueOf(ch)) == -1) {
                result.append(ch);
            }
        }

        return result.toString();
    }

    // Function to find the positions of two characters in the key matrix
    private static void findPositions(char ch, int[] pos) {
        if (ch == 'J') {
            ch = 'I';
        }

        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (keyMatrix[i][j] == ch) {
                    pos[0] = i;
                    pos[1] = j;
                    return;
                }
            }
        }
    }

    // Function to encrypt a pair of characters using Playfair Cipher
    private static String encryptPair(char[] pair) {
        int[] pos1 = new int[2];
        int[] pos2 = new int[2];

        findPositions(pair[0], pos1);
        findPositions(pair[1], pos2);
    }

```

```

StringBuilder result = new StringBuilder();

// Characters are in the same row
if (pos1[0] == pos2[0]) {
    result.append(keyMatrix[pos1[0]][(pos1[1] + 1) % 5]);
    result.append(keyMatrix[pos2[0]][(pos2[1] + 1) % 5]);
}

// Characters are in the same column
else if (pos1[1] == pos2[1]) {
    result.append(keyMatrix[(pos1[0] + 1) % 5][pos1[1]]);
    result.append(keyMatrix[(pos2[0] + 1) % 5][pos2[1]]);
}

// Characters form a rectangle
else {
    result.append(keyMatrix[pos1[0]][pos2[1]]);
    result.append(keyMatrix[pos2[0]][pos1[1]]);
}

return result.toString();
}

public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(12345);
    System.out.println("Server listening on port 12345...");

    while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("Accepted connection from " + clientSocket.getInetAddress());

        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        String message = in.readLine();
        String key = in.readLine();
    }
}

```

```

        System.out.println("Received message: " + message);
        System.out.println("Received key: " + key);

        // Generate the Playfair key matrix
        generateKeyMatrix(key);

        StringBuilder encryptedMessage = new StringBuilder();
        // Encrypt the message in pairs
        for (int i = 0; i < message.length(); i += 2) {
            char[] pair = new char[2];
            pair[0] = message.charAt(i);
            pair[1] = (i + 1 < message.length()) ? message.charAt(i + 1) : 'X';
            encryptedMessage.append(encryptPair(pair));
        }

        System.out.println("Encrypted message: " + encryptedMessage.toString());

        clientSocket.close();
    }
}

```

Client Code

```

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

public class Client {
    public static void main(String[] args) throws IOException {
        Socket clientSocket = new Socket("localhost", 12345);

        // Get user input for the message and the key

```



```

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the message: ");

String message = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");

System.out.print("Enter the key: ");

String key = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");


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

System.out.println("Key: " + key);


// Send the message and the key to the server

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

out.println(message);

out.println(key);


clientSocket.close();

scanner.close();

}

}

```

Algorithm

Server (Server.java):

Import Java Libraries: Import necessary Java libraries (java.io.*, java.net.*).

Define Play fair Cipher Functions: Implement functions to generate the Playfair key matrix (generateKeyMatrix), prepare the key (prepareKey), find positions of characters in the matrix (findPositions), and encrypt a pair of characters (encryptPair).

Main Server Logic: Create a ServerSocket on a specified port (e.g., 12345). Enter an infinite loop to handle incoming client connections.

For each client connection: Accept the connection and obtain an input stream. Read the message and the key from the client. Generate the Playfair key matrix. Encrypt the message in pairs using Play fair Cipher. Output the original message and the encrypted message.

Close the client socket.

Client (Client.java):

Import Java Libraries: Import necessary Java libraries (java.io.*, java.net.*, java.util.Scanner).

User Input: Use Scanner to get user input for the message and the key.

Connect to Server: Create a Socket to connect to the server (localhost, port 12345).

Send User Input to Server: Create an output stream to send the message and the key to the server.

Close Client Socket: Close the client socket after sending the data.

Execution:

Run the Server: Compile and run Server.java on one terminal to start the server listening on port 12345.

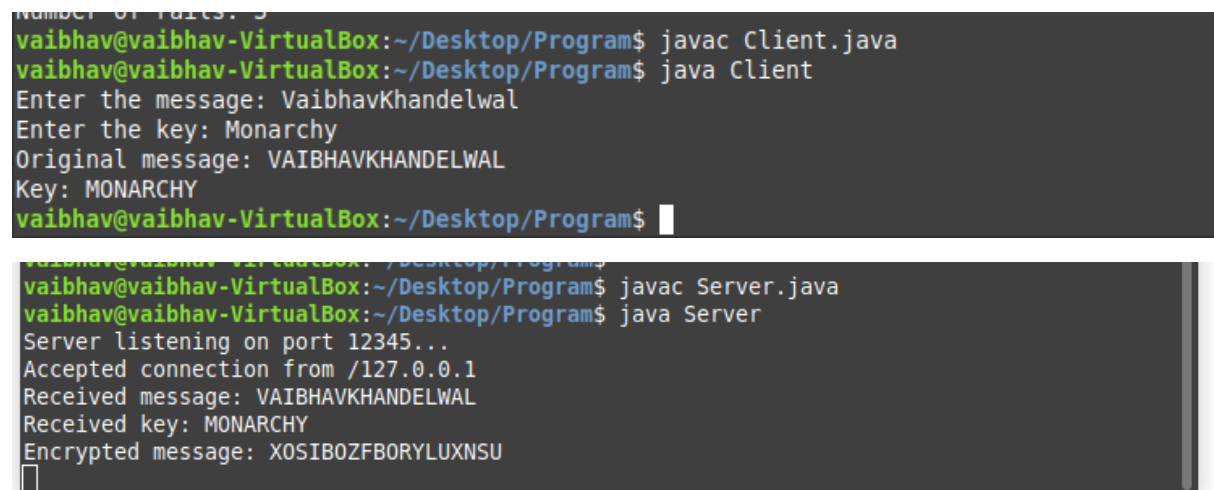
Run the Client: Compile and run Client.java on another terminal to connect to the server.

Enter the message and the key when prompted.

Server Processing: The server receives the message and key. Generates the Playfair key matrix. Encrypts the message in pairs using Playfair Cipher. Outputs the original message and the encrypted message.

Client Exit: The client sends the message and key to the server and then closes.

Output



```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Client.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Client
Enter the message: VaibhavKhandelwal
Enter the key: Monarchy
Original message: VAIBHAVKHANDELWAL
Key: MONARCHY
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$

vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Server.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Server
Server listening on port 12345...
Accepted connection from /127.0.0.1
Received message: VAIBHAVKHANDELWAL
Received key: MONARCHY
Encrypted message: XOSIBOZFBORYLUXNSU
```

Question 4: -

Write a socket program to demonstrate Row Transposition Cipher.

Server Code:

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

```

import java.net.*;

public class Server {

    // Function to perform row transposition encryption
    private static String rowTranspositionEncrypt(String plaintext, int[] key) {

        int numRows = (int) Math.ceil((double) plaintext.length() / key.length);

        char[][] grid = new char[numRows][key.length];

        int index = 0;

        // Fill the grid with the plaintext
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < key.length; j++) {
                if (index < plaintext.length()) {
                    grid[i][j] = plaintext.charAt(index++);
                } else {
                    grid[i][j] = 'X'; // Padding with 'X' if needed
                }
            }
        }

        StringBuilder ciphertext = new StringBuilder();

        // Read out the grid column by column using the key
        for (int k : key) {
            for (int i = 0; i < numRows; i++) {
                ciphertext.append(grid[i][k - 1]);
            }
        }

        return ciphertext.toString();
    }
}

```

```

public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new ServerSocket(12345);
    System.out.println("Server listening on port 12345...");

    while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("Accepted connection from " + clientSocket.getInetAddress());

        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        String plaintext = in.readLine();
        String keyString = in.readLine();

        System.out.println("Received plaintext: " + plaintext);
        System.out.println("Received key: " + keyString);

        // Convert the key from a string to an array of integers
        String[] keyArray = keyString.split(",");
        int[] key = new int[keyArray.length];
        for (int i = 0; i < keyArray.length; i++) {
            key[i] = Integer.parseInt(keyArray[i]);
        }

        // Encrypt the plaintext using row transposition cipher
        String ciphertext = rowTranspositionEncrypt(plaintext, key);
        System.out.println("Encrypted ciphertext: " + ciphertext);

        clientSocket.close();
    }
}

```

Client Code:

```
import java.io.*;

import java.net.*;

import java.util.Scanner;


public class Client {

    public static void main(String[] args) throws IOException {

        Socket clientSocket = new Socket("localhost", 12345);


        // Get user input for the plaintext and the key

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the plaintext: ");

        String plaintext = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");

        System.out.print("Enter the key (comma-separated numbers): ");

        String keyString = scanner.nextLine();


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

        System.out.println("Key: " + keyString);


        // Send the plaintext and the key to the server

        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

        out.println(plaintext);

        out.println(keyString);


        clientSocket.close();

        scanner.close();

    }

}
```

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Client.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Client
Enter the plaintext: vaibhav
Enter the key (comma-separated numbers): 2,1,4,3
Original plaintext: VAIBHAV
Key: 2,1,4,3
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Server.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Server
Server listening on port 12345...
Accepted connection from /127.0.0.1
Received plaintext: VAIBHAV
Received key: 2,1,4,3
Encrypted ciphertext: AAVHBXIV
█
```

Algorithm:

Server (Server.java):

Import Java Libraries: Import necessary Java libraries (java.io. *, java.net. *).

Define Row Transposition Cipher Function: Implement the rowTranspositionEncrypt function to perform row transposition encryption.

Main Server Logic:

Create a Server Socket on a specified port (e.g., 12345). Enter an infinite loop to handle incoming client connections.

For each client connection: Accept the connection and obtain an input stream. Read the plaintext and the key from the client. Convert the key from a string to an array of integers. Encrypt the plaintext using row transposition cipher. Output the original plaintext and the encrypted ciphertext.

Close the client socket.

Client (Client.java):

Import Java Libraries: Import necessary Java libraries (java.io.*, java.net.*, java.util.Scanner).

User Input: Use Scanner to get user input for the plaintext and the key.

Connect to Server: Create a Socket to connect to the server (localhost, port 12345).

Send User Input to Server: Create an output stream to send the plaintext and the key to the server.

Close Client Socket: Close the client socket after sending the data.

Execution:

Run the Server: Compile and run Server.java on one terminal to start the server listening on port 12345.

Run the Client: Compile and run Client.java on another terminal to connect to the server. Enter the plaintext and key when prompted.

Server Processing: The server receives the plaintext and key. Converts the key from a string to an array of integers. Encrypts the plaintext using row transposition cipher. Outputs the original plaintext and the encrypted ciphertext.

Client Exit: The client sends the plaintext and key to the server and then closes.



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab Experiment-2

Implementation of Number Theory Concepts

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-15-01-2024

Question 1: -

Demonstrate Euclidean Algorithm.

Code

```
import java.util.Scanner;

public class EuclideanAlgorithm {

    public static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```



```

    System.out.print("Enter the first number: ");

    int num1 = scanner.nextInt();

    System.out.print("Enter the second number: ");

    int num2 = scanner.nextInt();

    int gcd = gcd(num1, num2);

    System.out.println("GCD of " + num1 + " and " + num2 + " is: " + gcd);
}
}

```

Algorithm

1. Take input from the user:

Prompt the user to enter the first integer.

Read the entered integer and store it in a variable num1.

Prompt the user to enter the second integer.

Read the entered integer and store it in a variable num2.

2. Implement the gcd(int a, int b) function:

Initialization:

Start with a as the larger of the two input numbers and b as the smaller number.

Iterative process:

While b is not equal to zero, repeat the following steps:

Calculate the remainder of a divided by b and store it in a temporary variable temp.

Update a with the value of b.

Update b with the value of temp.

Return the GCD:

After the loop terminates, the value of a holds the GCD of the two numbers. Return this value.

3. Main program flow:

Call the gcd function with num1 and num2 as arguments.

Store the returned GCD in a variable.

Print the calculated GCD to the console.

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac EuclideanAlgorithm.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java EuclideanAlgorithm
Enter the first number: 60
Enter the second number: 24
GCD of 60 and 24 is: 12
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

Question 2: -

Demonstrate Extended Euclidean Algorithm.

Code

```
public class ExtendedEuclideanAlgorithm {

    // Method to find the greatest common divisor using Extended Euclidean algorithm recursively

    public static int gcdExtended(int a, int b, int[] x) {

        if (a == 0) {

            x[0] = 0;

            x[1] = 1;

            return b;

        }

        int[] y = new int[2];

        int gcd = gcdExtended(b % a, a, y);

        x[0] = y[1] - (b / a) * y[0];

        x[1] = y[0];

        return gcd;

    }

    public static void main(String[] args) {

        int a = 35, b = 15;
```

```

        int[] x = new int[2];

        int gcd = gcdExtended(a, b, x);

        System.out.println("GCD of " + a + " and " + b + " is: " + gcd);

        System.out.println("Roots x and y are: " + x[0] + " and " + x[1]);

    }
}

```

Algorithm

Create a class ExtendedEuclideanAlgorithm.

Define a static method extended Euclidean that takes two integers a and b as input and returns an array of three integers.

Inside the extended Euclidean method, create an integer array result of size 3 to store the GCD and the coefficients.

Check if a is equal to 0:

If true, set the first element of result to b, the second element to 0, and the third element to 1.

If false, proceed to the next steps.

Recursively call the extended Euclidean method with arguments $b \% a$ and a, and store the result in an integer array recursive Result.

Extract the GCD, coefficient x, and coefficient y from the recursive Result array.

Update the result array with the GCD and updated coefficients:

Set the first element of result to the GCD.

Set the second element of result to $y - (b / a) * x$.

Set the third element of result to x.

Return the result array.

In the main method, define two integers a and b with the values of 550 and 1769, respectively.

Call the extended Euclidean method with a and b, and store the result in an integer array result.

Extract the GCD, coefficient x, and coefficient y from the result array.

Print the GCD and the equation $a * x + b * y = \text{GCD}(a, b)$ to the console.

Output

```

vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac ExtendedEuclideanAlgorithm.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java ExtendedEuclideanAlgorithm
GCD of 550 and 1769 is: 1
Roots x and y are: 550 and -171
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$

```

Question 3: -

Demonstrate Euler's Theorem.

Code

```
import java.util.Scanner;

public class EulersTotientFunction {

    static int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }

    static int phi(int n) {
        int result = 1;
        for (int i = 2; i < n; i++) {
            if (gcd(i, n) == 1) {
                result++;
            }
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a positive integer: ");
        int n = sc.nextInt();
        System.out.println("phi(" + n + ") = " + phi(n));
    }
}
```

Algorithm

1. Greatest Common Divisor (GCD) Function:

Input: Two integers a and b.

Output: The greatest common divisor of a and b.

Steps:

If a is 0, return b.

Otherwise, recursively call $\text{gcd}(b \% a, a)$.

2. Euler's Totient Function (phi):

Input: A positive integer n.

Output: The number of positive integers less than or equal to n that are relatively prime to n.

Steps:

Initialize a counter result to 1 (counting 1 itself, which is always relatively prime to n).

Iterate through the integers i from 2 to n-1:

Calculate the GCD of i and n using the gcd function.

If the GCD is 1 (meaning i and n are relatively prime), increment result.

Return the final value of result.

3. Main Function:

Purpose: To prompt the user for input, calculate the totient value, and print the result.

Steps:

Create a Scanner object to read input from the console.

Prompt the user to enter a positive integer n.

Call the phi function to calculate the totient value of n.

Print the result to the console.

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac Eulers.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java Eulers
Enter a positive integer: 11
phi(11) = 10
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```

Question 4: -

Demonstrate the Miller Rabin Algorithm.

Code

```
import java.util.Random;

import java.math.BigInteger;

import java.util.Scanner;

public class MillerRabin {

    static boolean isPrime(BigInteger n, int k) {

        // Base cases

        if (n.equals(BigInteger.ONE)) return false;

        if (n.equals(BigInteger.TWO)) return true;

        if (n.mod(BigInteger.TWO).equals(BigInteger.ZERO)) return false;

        BigInteger d = n.subtract(BigInteger.ONE);

        int s = 0;

        while (d.mod(BigInteger.TWO).equals(BigInteger.ZERO)) {

            d = d.divide(BigInteger.TWO);

            s++;

        }

        for (int i = 0; i < k; i++) {

            Random rand = new Random();

            BigInteger a = new BigInteger(n.bitLength(), rand);

            // 2 <= a <= n-2

            a = a.add(BigInteger.TWO);

            a = a.mod(n.subtract(BigInteger.TWO)).add(BigInteger.TWO);
```

```

        BigInteger x = a.modPow(d, n);
        if (x.equals(BigInteger.ONE) || x.equals(n.subtract(BigInteger.ONE))) continue;

        for (int r = 1; r < s; r++) {
            x = x.modPow(BigInteger.TWO, n);
            if (x.equals(BigInteger.ONE)) return false;
            if (x.equals(n.subtract(BigInteger.ONE))) break;
        }

        if (!x.equals(n.subtract(BigInteger.ONE))) return false;
    }

    return true;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a number to test for primality: ");
    BigInteger n = scanner.nextBigInteger();
    int k = 5; // Number of iterations for accuracy
    if (isPrime(n, k)) {
        System.out.println(n + " is probably prime");
    } else {
        System.out.println(n + " is composite");
    }
}
}

```

Algorithm

1. Main Function:

Prompt User: Ask the user to enter a number to test for primality.

Read Input: Read the user's input as a BigInteger.

Perform Miller-Rabin Test: Call the isPrime function with the user-provided number and a chosen number of iterations (e.g., 5).

Print Result: Print whether the number is "probably prime" or "composite" based on the test's result.

2. isPrime(BigInteger n, int k) Function:

Base Cases:

If n is 1, return false (not prime).

If n is 2, return true (prime).

If n is even, return false (not prime).

Factor n-1: Decompose n-1 into $2^s * d$.

Iterations:

Repeat k times:

Choose a random base a in the range $2 \leq a \leq n-2$.

Perform Miller-Rabin test calculations:

If certain conditions are met, the number is composite, so return false.

Return True: If all iterations complete without finding compositeness, return true (probably prime).

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ javac MillerRabin.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$ java MillerRabin
Enter a number to test for primality: 17
17 is probably prime
vaibhav@vaibhav-VirtualBox:~/Desktop/Program$
```




VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-3

Implementation of DES

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-22-01-2024

Task

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.

Code

Client Side

```
import javax.crypto.Cipher;  
import javax.crypto.SecretKey;  
import javax.crypto.SecretKeyFactory;  
import javax.crypto.spec.DESKeySpec;  
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.net.Socket;  
import java.util.Base64;  
import java.util.Scanner;  
  
public class DesClient {
```

```

public static void main(String[] args) {
    try {
        // Connect to the server
        Socket socket = new Socket("localhost", 12345);

        // Generate a random 64-bit key (8 bytes)
        byte[] keyBytes = "MySecret".getBytes(); // Replace with a secure key generation
mechanism
        DESKeySpec desKeySpec = new DESKeySpec(keyBytes);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
        SecretKey key = keyFactory.generateSecret(desKeySpec);

        // Initialize the Cipher for encryption
        Cipher encryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        encryptCipher.init(Cipher.ENCRYPT_MODE, key);

        // Initialize the Cipher for decryption
        Cipher decryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        decryptCipher.init(Cipher.DECRYPT_MODE, key);

        // Setup input and output streams
        DataInputStream inputStream = new DataInputStream(socket.getInputStream());
        DataOutputStream outputStream = new
DataOutputStream(socket.getOutputStream());

        // Take input in the form of 0s and 1s from the user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a binary message (0s and 1s): ");
        String binaryMessage = scanner.nextLine();
    }
}

```

```

// Encrypt the message
byte[] encryptedBytes = encryptCipher.doFinal(binaryMessage.getBytes());
String encryptedMessage = Base64.getEncoder().encodeToString(encryptedBytes);

System.out.println("Encrypted message: " + encryptedMessage);

// Send the encrypted message to the server
outputStream.writeUTF(encryptedMessage);

// Close resources
socket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Server Side

```

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Base64;

```

```

import java.util.Scanner;

public class DesServer {

    public static void main(String[] args) {
        try {
            // Server socket setup
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server waiting for connection...");

            // Accept client connection
            Socket socket = serverSocket.accept();
            System.out.println("Client connected.");

            // Generate a random 64-bit key (8 bytes)
            byte[] keyBytes = "MySecret".getBytes(); // Replace with a secure key generation
            mechanism

            DESKeySpec desKeySpec = new DESKeySpec(keyBytes);
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey key = keyFactory.generateSecret(desKeySpec);

            // Initialize the Cipher for encryption
            Cipher encryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            encryptCipher.init(Cipher.ENCRYPT_MODE, key);

            // Initialize the Cipher for decryption
            Cipher decryptCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            decryptCipher.init(Cipher.DECRYPT_MODE, key);

            // Setup input and output streams

```

```

        DataInputStream inputStream = new DataInputStream(socket.getInputStream());

        DataOutputStream outputStream = new
DataOutputStream(socket.getOutputStream());

        // Receive encrypted message from client
        String encryptedMessage = inputStream.readUTF();

        System.out.println("Received encrypted message: " + encryptedMessage);

        // Decrypt the message
        byte[] decryptedBytes =
decryptCipher.doFinal(Base64.getDecoder().decode(encryptedMessage));

        String decryptedMessage = new String(decryptedBytes);

        System.out.println("Decrypted message: " + decryptedMessage);

        // Close resources
        serverSocket.close();

        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

Algorithm

Server Algorithm:

Setup Server: Create a ServerSocket and bind it to a specific port (e.g., 12345). Print a message indicating that the server is waiting for a connection.

Accept Connection:Use the `accept()` method of the `ServerSocket` to wait for and accept a connection from a client.Print a message indicating that a client has connected.

Generate Key:Generate a random 64-bit key (8 bytes). (Note: In a real-world scenario, use a secure key generation mechanism.)

Initialize Ciphers:Initialize two Cipher instances for encryption and decryption, using the DES algorithm in Electronic Codebook (ECB) mode with PKCS5Padding.

Setup Streams:Create `DataInputStream` and `DataOutputStream` to handle communication with the client.

Receive Encrypted Message:Use `readUTF()` to receive the encrypted message from the client.

Decrypt Message:Decode the Base64 string to obtain the byte array.Use the decryption Cipher to decrypt the message.

Display Decrypted Message:

Display the decrypted message on the server's console.

Close Resources:Close the `ServerSocket` and `Socket` connections.

Client Algorithm:

Setup Client:Create a `Socket` and connect it to the server's IP address and port (e.g., "localhost" and 12345).

Generate Key:Generate a random 64-bit key (8 bytes). (Note: In a real-world scenario, use a secure key generation mechanism.)

Initialize Ciphers:Initialize two Cipher instances for encryption and decryption, using the DES algorithm in Electronic Codebook (ECB) mode with PKCS5Padding.

Setup Streams:Create `DataInputStream` and `DataOutputStream` to handle communication with the server.

User Input:Use a Scanner to take input from the user in the form of 0s and 1s.

Encrypt Message:Use the encryption Cipher to encrypt the binary message.

Encode and Send Message:Convert the encrypted byte array to a Base64-encoded string.Use `writeUTF()` to send the encrypted message to the server.

Close Resources:Close the `Socket` connection.

Output:

```
Encrypted message: HjdXFskWhpoK0gTZwCYvUg==  
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$ javac DesClient.java  
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$ java DesClient  
Enter a binary message (0s and 1s): 11110000  
Encrypted message: HjdXFskWhpoK0gTZwCYvUg==  
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$
```

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$ javac DesServer.java  
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$ java DesServer  
Server waiting for connection...  
Client connected.  
Received encrypted message: HjdXFskWhpoK0gTZwCYvUg==  
Decrypted message: 11110000  
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-3$
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-4

Implementation of AES

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-29-01-2024

Task

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.

Code

Client Code

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.io.*;
import java.net.Socket;
import java.util.Base64;
import java.util.Scanner;

public class AESClient {
```



```

public static void main(String[] args) {
    try {
        Socket socket = new Socket("localhost", 5555);
        System.out.println("Connected to server.");

        // Receive the AES key from the server
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        SecretKey secretKey = (SecretKey) in.readObject();

        // Get user input
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter message to send to server: ");
        String message = scanner.nextLine();

        // Encrypt the message
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(message.getBytes());
        String encryptedMessage = Base64.getEncoder().encodeToString(encryptedBytes);

        // Send the encrypted message to the server
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
        out.writeObject(encryptedMessage);

        System.out.println("Sent encrypted message to server: " + encryptedMessage);

        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```
    }  
  }  
}
```

Server Code

```
import javax.crypto.Cipher;  
import javax.crypto.KeyGenerator;  
import javax.crypto.SecretKey;  
import java.io.*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.Base64;  
  
public class AESServer {  
  
    public static void main(String[] args) {  
        try {  
            ServerSocket serverSocket = new ServerSocket(5555);  
            System.out.println("Server waiting for client on port 5555...");  
  
            Socket socket = serverSocket.accept();  
            System.out.println("Client connected.");  
  
            // Generate AES key  
            KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");  
            keyGenerator.init(128); // You can use 256 for a 256-bit key  
            SecretKey secretKey = keyGenerator.generateKey();
```

```

// Send the key to the client
ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
out.writeObject(secretKey);

// Receive encrypted message from the client
ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
String encryptedMessage = (String) in.readObject();

// Decrypt the message
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, secretKey);
byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(encryptedMessage));
String decryptedMessage = new String(decryptedBytes);

System.out.println("Received encrypted message from client: " + encryptedMessage);
System.out.println("Decrypted message: " + decryptedMessage);

serverSocket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Algorithm

The provided Java code already implements the algorithm for a simple client-server interaction using AES encryption and decryption. Here's a high-level explanation of the algorithm:

Server Algorithm:Create ServerSocket:Create a ServerSocket that listens on a specified port (5555 in this case).

Accept Client Connection:Use serverSocket.accept() to wait for and accept a connection from a client.

Generate AES Key:Use KeyGenerator to generate an AES key with a specified key size (128 bits in this case).

Send AES Key to Client:Use ObjectOutputStream to send the generated AES key to the client.

Receive Encrypted Message from Client:Use ObjectInputStream to receive the encrypted message from the client.

Decrypt Message:Use Cipher in decryption mode to decrypt the received encrypted message using the generated AES key.

Print Decrypted Message:Print the decrypted message to the console.

Close ServerSocket:Close the ServerSocket after the communication is complete.

Client Algorithm:

Create Socket:Create a Socket to connect to the server's IP address and port.

Receive AES Key from Server:Use ObjectInputStream to receive the AES key generated by the server.

Get User Input:Use Scanner to get user input (the message to be sent to the server).

Encrypt Message:Use Cipher in encryption mode to encrypt the user's input message using the received AES key.

Send Encrypted Message to Server:Use ObjectOutputStream to send the encrypted message to the server.

Print Sent Message:Print the encrypted message that was sent to the server.

Close Socket:Close the Socket after the communication is complete.

Output

```
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$ javac AESServer.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$ java AESServer
Server waiting for client on port 5555...
Client connected.
Received encrypted message from client: 0EoWZARkx6ZfM9jUpvAWZd7NKkbt/nT+EfwCyWnpTfBEfN0/fL3aUeMy3SkbJi1
Decrypted message: 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$
```

```
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$ javac AESClient.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$ java AESClient
Connected to server.
Enter message to send to server: 54 68 61 74 73 20 6D 79 20 4B 75 6E 67 20 46 75
Sent encrypted message to server: 0EoWZARkx6ZfM9jUpvAWZd7NKkbbT/nT+EfwCywNpTfBEfN0/fL3aUeMy3SkbJil
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-4$
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-5

Implementation of RSA Cryptosystem

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-05-02-2024

Task

Develop a cipher scheme using the RSA algorithm.

Code

Server Code

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

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server is waiting for client...");
```

```

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected.");


// Receive p and q from client

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

int p = Integer.parseInt(in.readLine());
int q = Integer.parseInt(in.readLine());


// Calculate n and phi(n)

int n = p * q;
int phiN = (p - 1) * (q - 1);


// Ask for e from client

System.out.println("Enter a value for e such that gcd(e, phi(n)) = 1 and 1 < e < " +
phiN);

int e = Integer.parseInt(new BufferedReader(new
InputStreamReader(System.in)).readLine());


// Calculate d

int d = modInverse(e, phiN);


// Send public key (e, n) to client

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
out.println(e);
out.println(n);


// Receive message M from client

System.out.println("Enter the message M to be transferred:");

String message = new BufferedReader(new
InputStreamReader(System.in)).readLine();

```

```

        // Encrypt the message
        int encryptedMessage = modPow(Integer.parseInt(message), e, n);
        System.out.println("Encrypted message (C): " + encryptedMessage);

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

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

// Function to calculate modular inverse
private static int modInverse(int a, int m) {
    for (int i = 1; i < m; i++) {
        if ((a * i) % m == 1) {
            return i;
        }
    }
    return -1; // No modular inverse found
}

// Function to calculate modular exponentiation (a^b mod m)
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;
}
}

```

Client Code

```

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

public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);

            // Send p and q to server
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            System.out.println("Enter a prime number p:");
            out.println(new BufferedReader(new InputStreamReader(System.in)).readLine());
            System.out.println("Enter a prime number q:");
            out.println(new BufferedReader(new InputStreamReader(System.in)).readLine());

            // Receive public key (e, n) from server
            BufferedReader in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
            int e = Integer.parseInt(in.readLine());

```

```

int n = Integer.parseInt(in.readLine());

System.out.println("Received public key (e, n): {" + e + ", " + n + "}");

// Receive encrypted message C from server
int encryptedMessage = Integer.parseInt(in.readLine());

System.out.println("Received encrypted message (C): " + encryptedMessage);

// Ask for private key (d, n) from the client
System.out.println("Enter the private key (d) such that (e * d) % phi(n) = 1 and 1 < d < n:");

int d = Integer.parseInt(new BufferedReader(new
InputStreamReader(System.in)).readLine());

// Decrypt the message
int decryptedMessage = modPow(encryptedMessage, d, n);

System.out.println("Decrypted message (M): " + decryptedMessage);

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

// Function to calculate modular exponentiation (a^b mod m)
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;
}
}

```

Algorithm

Server Algorithm:

Open a ServerSocket on port 12345.

Wait for a client to connect (serverSocket.accept()).

Upon client connection, receive the prime numbers p and q from the client.

Calculate $n = p * q$ and $\phi(n) = (p - 1) * (q - 1)$.

Ask the client to provide a value for e such that $\gcd(e, \phi(n)) = 1$ and $1 < e < \phi(n)$.

Calculate d using the modular inverse of e modulo $\phi(n)$.

Send the public key {e, n} to the client.

Receive the message M from the client.

Encrypt the message using the public key: $C = M^e \bmod n$.

Send the encrypted message C to the client.

Close the server socket.

Client Algorithm:

Open a Socket to connect to the server on localhost:12345.

Prompt the user to input a prime number p and send it to the server.

Prompt the user to input another prime number q and send it to the server.

Receive the public key {e, n} from the server.

Prompt the user to input a value for e such that $\gcd(e, \phi(n)) = 1$ and $1 < e < \phi(n)$.

Send the chosen value of e to the server.

Receive the encrypted message C from the server.

Prompt the user to input a private key d such that $(e * d) \% \phi(n) = 1$ and $1 < d < n$.

Decrypt the message using the private key: $M = C^d \bmod n$.

Display the decrypted message M.

Close the client socket.

Output

```
oslab@oslab-VirtualBox:~$ javac Server.java
oslab@oslab-VirtualBox:~$ java Server
Server is waiting for client...
Client connected.
Enter a value for e such that gcd(e, phi(n)) = 1 and 1 < e < 20
7
Enter the message M to be transferred:
6
Encrypted message (C): 30
oslab@oslab-VirtualBox:~$
```

```
oslab@oslab-VirtualBox:~$ javac Client.java
oslab@oslab-VirtualBox:~$ java Client
Enter a prime number p:
3
Enter a prime number q:
11
Received public key (e, n): {7, 33}
Received encrypted message (C): 30
Enter the private key (d) such that (e * d) % phi(n) = 1 and 1 < d < n:
3
Decrypted message (M): 6
oslab@oslab-VirtualBox:~$
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-6

Implementation of Elgamal Cryptosystem

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-19-02-2024

Task

Develop a cipher scheme using the Elgamal public key cryptographic algorithm.

Code

Client Code

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

public class Client {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("localhost", 1234);
        System.out.println("Connected to server.");
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.print("Enter secret number Xa (1 <= Xa < q): ");
```

```

BigInteger Xa = new BigInteger(br.readLine());
BigInteger q = new BigInteger("19");
BigInteger alpha = new BigInteger("10");
BigInteger Ya = alpha.modPow(Xa, q);
System.out.println("Computed Ya: " + Ya);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println(Ya);

BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
BigInteger C1 = new BigInteger(in.readLine());
BigInteger C2 = new BigInteger(in.readLine());
System.out.println("Received C1: " + C1);
System.out.println("Received C2: " + C2);
BigInteger K = C1.modPow(Xa, q);
BigInteger M = C2.multiply(K.modInverse(q)).mod(q);
System.out.println("Computed recover key K: " + K);
System.out.println("Computed M: " + M);
socket.close();
}
}

```

Server Code

```

import java.io.*;
import java.net.*;
import java.math.*;

public class Server {

    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(1234);
        System.out.println("Server started. Waiting for client to connect...");
    }
}

```

```

Socket clientSocket = serverSocket.accept();
System.out.println("Client connected.");
BigInteger q = new BigInteger("19");
BigInteger alpha = new BigInteger("10");
BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
BigInteger Ya = new BigInteger(in.readLine());
System.out.println("Received Ya from client: " + Ya);
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
System.out.print("Enter message M ( $0 \leq M < q$ ): ");
BigInteger M = new BigInteger(br.readLine());
System.out.print("Enter random integer k ( $1 \leq k < q$ ): ");
BigInteger k = new BigInteger(br.readLine());
BigInteger K = Ya.modPow(k, q);
System.out.println("Computed one time key: " + K);
BigInteger C1 = alpha.modPow(k, q);
BigInteger C2 = K.multiply(M).mod(q);
System.out.println("Computed C1: " + C1);
System.out.println("Computed C2: " + C2);
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
out.println(C1);
out.println(C2);
clientSocket.close();
serverSocket.close();
}
}

```

Algorithm

Client Side

Import the necessary Java libraries: `java.io.*`, `java.net.*`, and `java.math.*`.

Create a class named Client.

In the main method, establish a socket connection to the server using the Socket class. Provide the hostname and port number as parameters.

Print a message indicating that the client is connected to the server.

Create a `BufferedReader` object to read input from the user.

Prompt the user to enter a secret number X_a ($1 \leq X_a < q$), where q is a predefined `BigInteger` value.

Read the user's input and store it in a `BigInteger` variable named X_a .

Define the `BigInteger` values q and α with their respective predefined values.

Calculate Y_a using the formula $Y_a = \alpha.\text{modPow}(X_a, q)$.

Print the computed value of Y_a .

Create a `PrintWriter` object to send data to the server via the socket's output stream.

Send Y_a to the server by calling the `println` method on the `PrintWriter` object.

Create a `BufferedReader` object to read data from the server via the socket's input stream.

Read the values of C_1 and C_2 from the server by calling the `readLine` method on the `BufferedReader` object.

Print the received values of C_1 and C_2 .

Calculate the recover key K using the formula $K = C_1.\text{modPow}(X_a, q)$.

Calculate M using the formula $M = C_2.\text{multiply}(K.\text{modInverse}(q)).\text{mod}(q)$.

Print the computed values of K and M .

Close the socket connection.

Server Side

Import the necessary Java libraries: `java.io.*`, `java.net.*`, and `java.math.*`.

Create a class named Server.

In the main method, create a `ServerSocket` object and bind it to port 1234.

Print a message indicating that the server has started and is waiting for a client to connect.

Accept a client connection by calling the `accept` method on the `ServerSocket` object. This will block until a client connects.

Print a message indicating that the client has connected.

Define the BigInteger values q and alpha with their respective predefined values.

Create a BufferedReader object to read input from the client via the socket's input stream.

Read the value of Ya from the client by calling the readLine method on the BufferedReader object.

Print the received value of Ya.

Create another BufferedReader object to read input from the server's console.

Prompt the user to enter a message M ($0 \leq M < q$) and a random integer k ($1 \leq k < q$).

Read the user's input for M and k using the readLine method on the second BufferedReader object.

Convert the user's input for M and k into BigInteger variables named M and k, respectively.

Calculate the one-time key K using the formula $K = Y_a.\text{modPow}(k, q)$.

Print the computed value of K.

Calculate C1 using the formula $C_1 = \alpha.\text{modPow}(k, q)$.

Calculate C2 using the formula $C_2 = K.\text{multiply}(M).\text{mod}(q)$.

Print the computed values of C1 and C2.

Create a PrintWriter object to send data to the client via the socket's output stream.

Send C1 and C2 to the client by calling the println method on the PrintWriter object.

Close the client socket and the server socket.

Output

```
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$ javac Server.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$ java Server
Server started. Waiting for client to connect...
Client connected.
Received Ya from client: 3
Enter message M ( $0 \leq M < q$ ): 17
Enter random integer k ( $1 \leq k < q$ ): 6
Computed one time key: 7
Computed C1: 11
Computed C2: 5
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$
```

```
vaibhav@vaibhav-VirtualBox: ~/Desktop/Lab-6
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$ javac Client.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$ java Client
Connected to server.
Enter secret number Xa ( $1 \leq Xa < q$ ): 5
Computed Ya: 3
Received C1: 11
Received C2: 5
Computed recover key K: 7
Computed M: 17
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-6$
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-7

Implementation of Elliptic Curve Cryptosystem

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-26-02-2024

Task

Develop a cipher scheme using the Elliptic curve public key cryptographic algorithm.

Code

Develop a cipher scheme using the Elliptic Curve cryptographic algorithm.

Server Side

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

public class EllipticCurveServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server is running...");
            while (true) {
                Socket socket = serverSocket.accept();
```

```

System.out.println("Client connected: " + socket);
new ClientHandler(socket).start();
}
} catch (IOException e) {
e.printStackTrace();
}
}

private static class ClientHandler extends Thread {
private final Socket socket;

public ClientHandler(Socket socket) {
this.socket = socket;
}

public void run() {
try {
BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
output.println("Shared Key = " + sharedKey);
socket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
}
}

```

Server Algorithm:

Initialize Server: Start the server and listen for incoming connections on a specific port.

Accept Connection: When a client connects, accept the connection and create a new

thread (or process) to handle communication with that client.

Handle Client Request: Inside the thread/process, read the input from the client (e.g., Elliptic Curve parameters) and perform the necessary calculations (e.g., computing shared key).

Send Response: Send the result (e.g., shared key) back to the client.

Close Connection: Close the connection with the client.

Repeat: Go back to step 2 and wait for the next client connection.

Client code:-

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

public class EllipticCurveClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 5000);
            BufferedReader input = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
            Scanner scanner = new Scanner(System.in);
            System.out.println("Let the Elliptic Curve be :  $y^2 = x^3 + ax + b \text{ mod } r$ ");
            System.out.println("Enter a value:");
            int a = Integer.parseInt(scanner.nextLine());
            output.println(a);
            String result = input.readLine();
            System.out.println("Server: " + result);
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}  
}
```

Client Algorithm:

Initialize Client: Start the client and establish a connection to the server using the server's IP address and port number.

Input Parameters: Prompt the user to input the necessary parameters for the Elliptic Curve calculations (e.g., a , b , r , $x1P$, $y1P$).

Send Parameters: Send the input parameters to the server.

Receive Result: Wait for the server to process the input and send back the result (e.g., shared key).

Display Result: Display the result received from the server.

Close Connection: Close the connection with the server.

Exit: End the client application.

Output

```
12P = [11, 11]  
13P = [2, 15]  
14P = [6, 4]  
15P = [7, 14]  
16P = [4, 8]  
17P = [0, 14]  
18P = [3, 5]  
19P = [9, 12]  
20P = [5, 12]  
21P = [5, 5]  
22P = [7, 12]  
23P = [13, 1]  
24P = [12, 7]  
25P = [15, 14]  
26P = [16, 14]  
27P = [11, 4]  
28P = [10, 10]  
29P = [3, 14]  
  
a value of first person?  
6  
Therefore, first person sends : [7, 3]  
  
a value of second person?  
2
```

```

12P = [11, 11]
13P = [2, 15]
14P = [6, 4]
15P = [7, 14]
16P = [4, 8]
17P = [0, 14]
18P = [3, 5]
19P = [9, 12]
20P = [5, 12]
21P = [5, 5]
22P = [7, 12]
23P = [13, 1]
24P = [12, 7]
25P = [15, 14]
26P = [16, 14]
27P = [11, 4]
28P = [10, 10]
29P = [3, 14]

```

a value of first person?

6

Therefore, first person sends : [7, 3]

a value of second person?

2

Therefore, second person sends : [9, 5]

The equation is : $y^2 = x^3 + (2 * x) + (2) \bmod 17$
Points on the Elliptic Curve :

```

P = [9, 5]
2P = [0, 3]
3P = [7, 3]
4P = [2, 2]
5P = [10, 14]
6P = [11, 11]
7P = [6, 4]
8P = [4, 8]
9P = [3, 5]
10P = [5, 12]
11P = [5, 5]
12P = [3, 12]
13P = [4, 9]
14P = [6, 13]
15P = [11, 6]
16P = [10, 3]
17P = [2, 15]
18P = [7, 14]
19P = [0, 14]
20P = [9, 12]
21P = [14, 6]

```

```

Let the Elliptic Curve be :  $y^2 = x^3 + ax + b \pmod{r}$ 
a value?
2
b value?
2
r value?
17
x1 value of P?
5
y1 value of P?
5
The value of P is [5,5]
The equation is :  $y^2 = x^3 + (2 * x) + (2) \pmod{17}$ 
Points on the Elliptic Curve :
P = [5, 5]
2P = [9, 5]
3P = [3, 12]
4P = [0, 3]
5P = [4, 9]
6P = [7, 3]
7P = [6, 13]
8P = [2, 2]
9P = [11, 6]
10P = [10, 14]
11P = [10, 3]

```

```

22P = [9, 12]
23P = [14, 6]
24P = [9, 12]
25P = [14, 6]
26P = [9, 12]
27P = [14, 6]
28P = [9, 12]
29P = [14, 6]

```

Therefore, from 6[9, 5] we get [11, 11]

```

The equation is :  $y^2 = x^3 + (2 * x) + (2) \pmod{17}$ 
Points on the Elliptic Curve :
P = [7, 3]
2P = [11, 11]
3P = [3, 5]
4P = [3, 12]
5P = [11, 6]
6P = [7, 14]
7P = [5, 15]
8P = [7, 14]
9P = [5, 15]
10P = [7, 14]
11P = [5, 15]
12P = [7, 14]

```


9P = [5, 15]
10P = [7, 14]
11P = [5, 15]
12P = [7, 14]
13P = [5, 15]
14P = [7, 14]
15P = [5, 15]
16P = [7, 14]
17P = [5, 15]
18P = [7, 14]
19P = [5, 15]
20P = [7, 14]
21P = [5, 15]
22P = [7, 14]
23P = [5, 15]
24P = [7, 14]
25P = [5, 15]
26P = [7, 14]
27P = [5, 15]
28P = [7, 14]
29P = [5, 15]

Therefore, from 2[7, 3] we get [11, 11]

Shared Key = 11



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-8

Implementation of Diffie-Hellman Key Exchange Protocol

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-04-03-2024

Task

Design a Diffie Hellman multiparty key exchange protocol and perform a Man-in-the-Middle Attack.

Client Code

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

public class UserOne {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Prime number q: ");
        BigInteger q = sc.nextBigInteger();
        int alpha = PrimitiveRoot.findPrimitive(q.intValue());
        System.out.println("Primitive Root: " + alpha);
        System.out.println("Private Number XA (<q): ");
```

```

BigInteger xa = sc.nextBigInteger();

BigInteger alphaBigInt = BigInteger.valueOf(alpha);

BigInteger ya = alphaBigInt.modPow(xa, q);

System.out.println("Public key (ya): " + ya);

try {

    Socket socket = new Socket("localhost", 9999);

    DataInputStream input = new DataInputStream(socket.getInputStream());

    DataOutputStream output = new
    DataOutputStream(socket.getOutputStream());

    output.writeUTF(q.toString());

    output.writeInt(alpha);

    output.writeUTF(ya.toString());

    output.flush();

    int yb = input.readInt();

    System.out.println("Received public key (yb) from UserTwo: " + yb);

    BigInteger secret = BigInteger.valueOf(yb).modPow(xa, q);

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

    input.close();

    output.close();

    socket.close();

} catch (IOException e) {

    e.printStackTrace();

}

}

}

```

Server Code

```

import java.io.*;

import java.math.BigInteger;

```

```

import java.net.*;
import java.util.Scanner;

public class UserTwo {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        try {

            ServerSocket serverSocket = new ServerSocket(9999);

            System.out.println("Waiting for connection from UserOne...");

            Socket socket = serverSocket.accept();

            System.out.println("Connection established with UserOne.");

            DataInputStream input = new DataInputStream(socket.getInputStream());

            DataOutputStream output = new
            DataOutputStream(socket.getOutputStream());

            BigInteger q = new BigInteger(input.readUTF());

            int alpha = input.readInt();

            System.out.println("Private Number XB (<q): ");

            BigInteger xb = sc.nextBigInteger();

            BigInteger alphaBigInt = BigInteger.valueOf(alpha);

            BigInteger yb = alphaBigInt.modPow(xb, q);

            System.out.println("Public key yb: " + yb);

            BigInteger ya = new BigInteger(input.readUTF());

            System.out.println("Received public key (ya) from UserOne: " + ya);

            output.writeInt(yb.intValue());

            output.flush();

            BigInteger secret = ya.modPow(xb, q);

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

            input.close();

            output.close();

            socket.close();

```

```
serverSocket.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}  
}
```

Primitive Root Function Code:

```
import java.util.*;  
class PrimitiveRoot{  
    static boolean isPrime(int n) {  
        if (n <= 1) {  
            return false;  
        }  
        if (n <= 3) {  
            return true;  
        }  
        if (n % 2 == 0 || n % 3 == 0) {  
            return false;  
        }  
        for (int i = 5; i * i <= n; i = i + 6) {  
            if (n % i == 0 || n % (i + 2) == 0) {  
                return false;  
            }  
        }  
        return true;  
    }  
    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;
}

static void findPrimefactors(HashSet<Integer> s, int n) {
    while (n % 2 == 0) {
        s.add(2);
        n = n / 2;
    }
    for (int i = 3; i <= Math.sqrt(n); i = i + 2) {
        while (n % i == 0) {
            s.add(i);
            n = n / i;
        }
    }
    if (n > 2) {
        s.add(n);
    }
}

public static int findPrimitive(int n) {
    HashSet<Integer> s = new HashSet<Integer>();
    // Check if n is prime or not
    if (isPrime(n) == false) {
        return -1;
    }
}

```

```

    }

    int phi = n - 1;
    findPrimefactors(s, phi);
    for (int r = 2; r <= phi; r++) {
        boolean flag = false;
        for (Integer a : s) {
            if (power(r, phi / (a), n) == 1) {
                flag = true;
                break;
            }
        }
        if (flag == false) {
            return r;
        }
    }
    return -1;
}

public static void main(String[] args) {
    int n = 761;
    System.out.println(
        " Smallest primitive root of " + n + " is " + findPrimitive(n)
    );
}
}

```

Algorithm

In the Diffie-Hellman key exchange, two parties can independently generate a shared secret key over an insecure channel:

1. Both parties agree on a large prime number (p) and a base g , where g is a primitive root modulo p .
2. Each party generates a private key a for party A, b for party B) and calculates their public key by raising the base to their private key modulo p . So, party A computes $A = g^a \bmod p$ and party B computes $B = g^b \bmod p$.
3. The public keys are exchanged over the insecure channel.
4. Each party then calculates the shared secret key using the received public key and their own private key. Party A computes $s = B^a \bmod p$ and party B computes $s = A^b \bmod p$.
5. Both parties now have the same shared secret key without ever transmitting it over the insecure channel.

In a man-in-the-middle attack on Diffie-Hellman, an attacker intercepts the communication between the two parties and establishes separate key exchanges with each party. The attacker then decrypts, re-encrypts, and forwards the messages between the two parties.

1. Party A generates its public key A and sends it to what it believes is party B.
2. The attacker intercepts A and sends its own public key A to party B.
3. Party B generates its public key B and sends it to what it believes is party A.
4. The attacker intercepts B and sends its own public key B to party A.
5. Party A calculates what it believes is the shared secret key with B as $s = B^a \bmod p$.
6. Party B calculates what it believes is the shared secret key with A as $s = A^b \bmod p$.
7. The attacker now has two separate shared secret keys: one with party A and one with party B.
8. The attacker can decrypt, read, modify, and re-encrypt all messages passing between party A and party B without their knowledge.

Output

Server-Side Output:

```
8c97997b61/redhat.java/jdt_ws/Java_35b6b82d/bin Server1
Waiting for connection from UserOne...
Connection established with UserOne.
Private Number XB (<q):
11
Public key yb: 785
Received public key (ya) from UserOne: 302
Shared secret: 537
```


Client-Side Output:

```
Prime number q:
809
Primitive Root: 3
Private Number XA (<q):
17
Public key (ya): 302
Received public key (yb) from UserTwo: 785
Shared secret: 537
```

Man in the middle attack

Code

```
import java.math.BigInteger;
import java.util.Random;

public class diffiehell {

    public static void main(String[] args) {

        // Public parameters
        BigInteger p = new BigInteger("227");
        BigInteger g = new BigInteger("14");

        // Alice's private key
        Random randomAlice = new Random();
        BigInteger a = new BigInteger(p.bitLength(), randomAlice);
        BigInteger A = g.modPow(a, p);

        // Bob's private key
        Random randomBob = new Random();
        BigInteger b = new BigInteger(p.bitLength(), randomBob);
        BigInteger B = g.modPow(b, p);

        // Eve intercepts Alice's public key and creates her own
        //public key for Alice
        BigInteger c = new BigInteger("65");
```

```

BigInteger C = g.modPow(c, p);
// Eve intercepts Bob's public key and creates her own
//public key for Bob
BigInteger d = new BigInteger("175");
BigInteger D = g.modPow(d, p);
// Alice and Bob compute their shared secret keys
BigInteger S1 = B.modPow(a, p);
BigInteger S2 = A.modPow(b, p);
// Eve computes her shared secret keys with Alice and Bob
BigInteger S3 = C.modPow(a, p);
BigInteger S4 = D.modPow(b, p);
System.out.println("Enter a prime number (p): " + p);
System.out.println("Enter a number (g): " + g);
System.out.println("Alice selected (a): " + a);
System.out.println("Bob selected (b): " + b);
System.out.println("Eve selected private number for Alice (c): " + c);
System.out.println("Eve selected private number for Bob(d): " + d);
System.out.println("Alice published (A): " + A);
System.out.println("Bob published (B): " + B);
System.out.println("Eve published value for Alice (C): " + C);
System.out.println("Eve published value for Bob (D): " + D);
System.out.println("Alice computed (S1): " + S1);
System.out.println("Eve computed key for Alice (S3): " + S3);
System.out.println("Bob computed (S2): " + S2);
System.out.println("Eve computed key for Bob (S4): " + S4);
}
}

```

Output

```
oslab@oslab-VirtualBox:~$ javac diffeheml.java
oslab@oslab-VirtualBox:~$ java diffeheml
Enter a prime number (p): 227
Enter a number (g): 14
Alice selected (a): 192
Bob selected (b): 24
Eve selected private number for Alice (c): 65
Eve selected private number for Bob (d): 175
Alice published (A): 7
Bob published (B): 90
Eve published value for Alice (C): 41
Eve published value for Bob (D): 32
Alice computed (S1): 69
Eve computed key for Alice (S3): 189
Bob computed (S2): 69
Eve computed key for Bob (S4): 99
oslab@oslab-VirtualBox:~$
```

Algorithm

Step 1: Selected public numbers p and g , p is a prime number, called the “modulus” and g is called the base.

Step 2: Selecting private numbers.

let Alice pick a private random number a and let Bob pick a private random number b , Malory picks 2 random numbers c and d .

Step 3: Intercepting public values,

Malory intercepts Alice’s public value ($ga \pmod p$), block it from reaching Bob, and instead sends Bob her own public value ($gc \pmod p$) and Malory intercepts Bob’s public value ($gb \pmod p$), block it from reaching Alice, and instead sends Alice her own public value ($gd \pmod p$)

Step 4: Computing secret key

Alice will compute a key $S1 = gda \pmod p$, and Bob will compute a different key, $S2 = gcb \pmod p$

Step 5: If Alice uses $S1$ as a key to encrypt a later message to Bob, Malory can decrypt it, re-encrypt it using $S2$, and send it to Bob. Bob and Alice won’t notice any problem and may assume their communication is encrypted, but in reality, Malory can decrypt, read, modify, and then re-encrypt all their conversations.



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-9

Implementation of Secure Hash Algorithm (SHA)

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-11-03-2024

Task

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

CODE:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA512Demo {
    public static void main(String[] args) {
        try {
            // Input string
            String input = "my name is adithya";
            // Convert the input string to bytes
            byte[] inputData = input.getBytes();
```

```

// Create a MessageDigest object for SHA-512
MessageDigest digest = MessageDigest.getInstance("SHA-512");

// Update the digest with the input data
digest.update(inputData);

// Get the hash bytes
byte[] hashBytes = digest.digest();

// Print the hash code
System.out.println("Hash Code: " + bytesToHex(hashBytes));

// Print the final value of the buffers
System.out.println("Final Value of the Buffers: " + digest.toString());
} catch (NoSuchAlgorithmException e) {
System.err.println("SHA-512 algorithm not available.");
}
}

// Helper method to convert bytes to hexadecimal string
private static String bytesToHex(byte[] bytes) {
StringBuilder sb = new StringBuilder();
for (byte b : bytes) {
sb.append(String.format("%02X", b));
}
return sb.toString();
}
}

```

Algorithm:

The SHA-512 algorithm is a cryptographic hash function that takes an input message of any length and produces a fixed-size output hash value of 512 bits. Below is the algorithm for SHA-512:

1. Initialization:- Initialize eight 64-bit variables (A, B, C, D, E, F, G, H) to the first 64 bits of the fractional parts of the square roots of the first eight prime numbers (2, 3, 5, 7, 11, 13, 17, 19) in hexadecimal.

2. Padding: - Append padding to the message to ensure its length is a multiple of 1024 bits (128 bytes). The padding consists of a single "1" bit, followed by as many "0" bits as needed to reach 896 bits before appending the length of the original message as a 128-bit integer.

3. Message Processing: - Divide the padded message into blocks of 1024 bits (128 bytes). - For each block: - Divide the block into 16 64-bit words ($W[0]$ to $W[15]$). - Extend the 16 words into 80 words using the SHA-512 message schedule. - Initialize working variables (a, b, c, d, e, f, g, h) with the current hash value (A, B, C, D, E, F, G, H). - Perform 80 rounds of computation using bitwise operations and addition modulo 2^{64} to update the working variables. - Update the hash value with the computed working variables.

4. Output: - Concatenate the final hash value of A, B, C, D, E, F, G, H to produce the 512-bit (64-byte) hash value. The details of the bitwise operations and constants used in each round are specified in the SHA-512 specification.

It's important to note that while the algorithm can be described in pseudocode or prose as above, implementing SHA-512 requires careful attention to bit manipulation and handling of 64-bit values, which are typically performed using low-level programming techniques and language features. Many programming languages provide libraries or built-in functions to compute SHA-512 hashes, avoiding the need for manual implementation in most cases.

OUTPUT:

```
java -cp /tmp/95EJdC0S21 SHA512Demo  
Hash Code: B60E737C299AB9CF6E9A4F95BFD28E6EF2772F319651A6F6F28048F710155C453D47C3841169414B036AEFC09033B20B87  
94C1075505DC82D68DB65E2B86C21B  
Final Value of the Buffers: SHA-512 Message Digest from SUN, <initialized>
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-10

Message Digest-5 Algorithm.

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-16-03-2024

Task

Develop a Message Digest-5 Algorithm

Code: -

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

// Java program to calculate MD5 hash value
public class MD5 {
    public static String[] getMd5(String input) {
        try {
            // Static getInstance method is called with hashing MD5
            MessageDigest md = MessageDigest.getInstance("MD5");
```



```

// digest() method is called to calculate message digest
// of an input digest() return array of byte
byte[] messageDigest = md.digest(input.getBytes());
// Convert byte array into signum representation
BigInteger no = new BigInteger(1, messageDigest);
// Convert message digest into hex value
String hashtext = no.toString(16);
while (hashtext.length() < 32) {
    hashtext = "0" + hashtext;
}
// Calculate buffer value and its length
String bufferValue = new String(messageDigest);
int bufferLength = messageDigest.length;
return new String[]{hashtext, bufferValue, Integer.toString(bufferLength)};
} catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e);
}
}

// Driver code
public static void main(String args[]) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a string to generate MD5 hash: ");
    String input = scanner.nextLine();
    scanner.close();

    // Get MD5 hash, buffer value, and buffer length
    String[] md5Result = getMd5(input);

```

```
String md5Hash = md5Result[0];
String bufferValue = md5Result[1];
String bufferLength = md5Result[2];
// Print input string and its length
System.out.println("Input String: " + input);
System.out.println("Input String Length: " + input.length());
// Print MD5 hash and its length
System.out.println("MD5 Hash: " + md5Hash);
System.out.println("MD5 Hash Length: " + md5Hash.length());
// Print buffer value and its length
// System.out.println("Buffer Value: " + bufferValue);
System.out.println("Buffer Length: " + bufferLength);
}
}
```

Algorithm

User Input: The program prompts the user to enter a string for which they want to generate the MD5

hash.

MD5 Calculation (getMd5 method):

The input string is converted to a byte array.

An instance of the MessageDigest class is created with the algorithm "MD5".

The digest() method of the MessageDigest instance is called with the input byte array to compute the

MD5 hash, which returns an array of bytes.

The byte array is converted to a BigInteger to obtain a positive number representation.

This BigInteger is then converted to a hexadecimal string representation, ensuring it's 32 characters long

(if not, padding with zeroes).

The buffer value is also calculated by converting the byte array to a string.

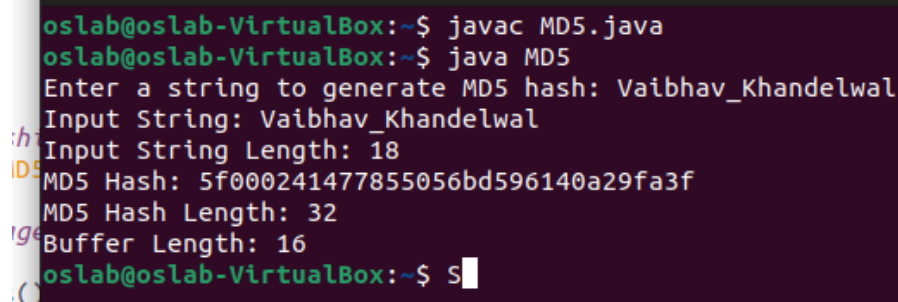
Printing Results (main method):

The input string and its length are printed.

The MD5 hash and its length are printed.

The buffer length is printed.

Output

A terminal window with a dark purple background and light green text. The text shows the compilation and execution of a Java program named MD5.java. The user enters the string 'Vaibhav_Khandelwal' and the program outputs the MD5 hash, its length (32), and the buffer length (16).

```
oslab@oslab-VirtualBox:~$ javac MD5.java
oslab@oslab-VirtualBox:~$ java MD5
Enter a string to generate MD5 hash: Vaibhav_Khandelwal
Input String: Vaibhav_Khandelwal
Input String Length: 18
MD5 Hash: 5f000241477855056bd596140a29fa3f
MD5 Hash Length: 32
Buffer Length: 16
oslab@oslab-VirtualBox:~$ S
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-11

Implementation of Digital Signature

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-08-04-2024

Task

Develop the Digital Signature Standard (DSS) for verifying the legal communicating parties.

Code

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

public class DigitalSignatureClient {
    private static final String SIGNING_ALGORITHM = "SHA256withRSA";
    private static final String RSA = "RSA";

    public static byte[] createDigitalSignature(byte[] input, PrivateKey key)
        throws Exception {
        Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
        signature.initSign(key);
        signature.update(input);
```

```

        return signature.sign();
    }

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

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 1234);
            ObjectOutputStream outputStream = new
ObjectOutputStream(socket.getOutputStream());

            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance(RSA);
            keyPairGenerator.initialize(2048);
            KeyPair keyPair = keyPairGenerator.generateKeyPair();

            String data = "HELLO THIS IS 21BCE1739";
            byte[] signature = createDigitalSignature(data.getBytes(),
keyPair.getPrivate());

            outputStream.writeObject(data.getBytes());
            outputStream.writeObject(signature);
            outputStream.writeObject(keyPair.getPublic());

            System.out.println("Data sent to server: " + data);
            System.out.println("Signature sent to server: " + bytesToHex(signature));

            outputStream.close();
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

SERVER:

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

public class DigitalSignatureServer {
    private static final String SIGNING_ALGORITHM = "SHA256withRSA";
    private static final String RSA = "RSA";

    public static boolean verifyDigitalSignature(byte[] input, byte[]
signatureToVerify, PublicKey key) throws Exception {
        Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
        signature.initVerify(key);
        signature.update(input);
        return signature.verify(signatureToVerify);
    }

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

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(1234);

            while (true) {
                System.out.println("Waiting for client...");
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");

                ObjectInputStream inputStream = new
ObjectInputStream(socket.getInputStream());

                byte[] data = (byte[]) inputStream.readObject();
                byte[] signature = (byte[]) inputStream.readObject();
```

```

        PublicKey publicKey = (PublicKey) inputStream.readObject();

        boolean verified = verifyDigitalSignature(data, signature, publicKey);

        System.out.println("Received data: " + new String(data));
        System.out.println("Received signature: " + bytesToHex(signature));
        System.out.println("Signature verified: " + verified);

        socket.close();
    }
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Algorithm

Client Side:

Message Digest Generation:

Calculate the hash value of the message (e.g., using SHA-256).

Encryption with Private Key:

Encrypt the hash value using the client's private key to generate the digital signature.

Let's denote the hash value as H , and the digital signature as $S = H^d \bmod n$, where ' d ' is the private exponent and ' n ' is the modulus.

Send Message with Signature:

Send the original message along with the digital signature to the server.

Server Side:

Receive Message and Signature:

Receive the message and the digital signature from the client.

Decryption with Public Key:

Decrypt the digital signature using the client's public key to retrieve the hash value.

Let's denote the decrypted hash value as $H' = S^e \bmod n$, where ' e ' is the public exponent and ' n ' is the modulus.

Message Digest Generation:

Calculate the hash value of the received message on the server side (using the same hashing algorithm as on the client side).

Signature Verification:

Compare the calculated hash value (H') obtained from the decrypted signature with the hash value generated from the received message.

If H' matches the calculated hash value, the signature is valid. Otherwise, it's invalid.

Output

```
at DigitalSignatureClient.main(DigitalSignatureClient.java:26)
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-11$ javac DigitalSignatureServer.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-11$ java DigitalSignatureServer
Waiting for client...
Client connected.
Received data: HELLO THIS IS 21BCE5550
Received signature: 0B1A9AB2F97E1320697828505C98162046EB2E9A61485DC8D22E5660FE87
03DC2E78CA2E64C986DCFA64AA46A2276FA26C2B4A070D890AA87F8F1D9C27CEDDCA9C75817DC48D
F8711C19A21C9252BD325BF97016F55962D8749751F7784153EDEF7518934EB33A409706A188AF2B
EA35D4A4C78D5E894D36C1DE8A5FCAB38784922FF53BF48D6C8378872BDB8898F01A9D109687A745
06D0453AD18B524BE0E35FD062BCBF04A8C400836D063D4C64C98372585DC55291EB777FCF41DE58
693A146F3EF1792FEB6B38066FE40D22726D1449B62A234E07DECA77B8B4E4850C300EB57E76317A
73C9786A0B6C31B185E99B3799912D4CC13FABCB083F8341CE2F
Signature verified: true
```

```
vaibhav@vaibhav-VirtualBox: ~/Desktop/Lab-11
File Edit View Search Terminal Help
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-11$ javac DigitalSignatureClient.java
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-11$ java DigitalSignatureClient
Data sent to server: HELLO THIS IS 21BCE5550
Signature sent to server: 0B1A9AB2F97E1320697828505C98162046EB2E9A61485DC8D22E56
60FE8703DC2E78CA2E64C986DCFA64AA46A2276FA26C2B4A070D890AA87F8F1D9C27CEDDCA9C7581
7DC48DF8711C19A21C9252BD325BF97016F55962D8749751F7784153EDEF7518934EB33A409706A1
88AF2BEA35D4A4C78D5E894D36C1DE8A5FCAB38784922FF53BF48D6C8378872BDB8898F01A9D1096
87A74506D0453AD18B524BE0E35FD062BCBF04A8C400836D063D4C64C98372585DC55291EB777FCF
41DE58693A146F3EF1792FEB6B38066FE40D22726D1449B62A234E07DECA77B8B4E4850C300EB57E
76317A73C9786A0B6C31B185E99B3799912D4CC13FABCB083F8341CE2F
vaibhav@vaibhav-VirtualBox:~/Desktop/Lab-11$
```




VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-12

Implementation of SSL Socket Communication

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-15-04-2024

Develop a simple client and server application using SSL socket communication.

Code: -

Server-Side

```
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <resolv.h>
#include "openssl/ssl.h"
#include "openssl/err.h"
#define FAIL -1
```

```

#define PORT 23451

int OpenListener(int port)
{
    int sd;
    struct sockaddr_in addr;
    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 )
    {
        perror("can't bind port");
        abort();
    }
    if ( listen(sd, 10) != 0 )
    {
        perror("Can't configure listening port");
        abort();
    }
    return sd;
}

SSL_CTX* InitServerCTX(void)
{
    SSL_METHOD *method;
    SSL_CTX *ctx;
    OpenSSL_add_all_algorithms(); /* load & register all cryptos, etc. */
    SSL_load_error_strings(); /* load all error messages */
    method = TLSv1_2_server_method(); /* create new server-method instance */

```

```

ctx = SSL_CTX_new(method); /* create new context from method */
if ( ctx == NULL )
{
ERR_print_errors_fp(stderr);
abort();
}
return ctx;
}

void LoadCertificates(SSL_CTX* ctx, char* CertFile, char* KeyFile)
{
/* set the local certificate from CertFile */
if ( SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM) <= 0 )
{
ERR_print_errors_fp(stderr);
abort();
}

/* set the private key from KeyFile (may be the same as CertFile) */
if ( SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM) <= 0 )
{
ERR_print_errors_fp(stderr);
abort();
}

/* verify private key */
if ( !SSL_CTX_check_private_key(ctx) )
{
fprintf(stderr, "Private key does not match the public certificate\n");
abort();
}
}

```

```

void Servlet(SSL* ssl) /* Serve the connection -- threadable */
{
    char buf[1024] = {0};
    int sd, bytes;
    const char* ServerValidResponse="Correct ID and Password";
    const char* ServerInvalidResponse = "Incorrect ID and Password";
    FILE* filePointer;
    int bufferLength = 255;
    char buffer[bufferLength];
    char *currUsername, *currPassword;
    filePointer = fopen("password", "r");
    if ( SSL_accept(ssl) == FAIL ) /* do SSL-protocol accept */
        ERR_print_errors_fp(stderr);
    else
    {
        bytes = SSL_read(ssl, buf, sizeof(buf)); /* get request */
        buf[bytes] = '\0';
        printf("Client msg: \"%s\"\n", buf);
        if ( bytes > 0 )
        {
            char *ptr = strtok(buf, " ");
            char *username=ptr; // get username from client
            ptr = strtok(NULL, " ");
            char *password=ptr; // get password from txt file
            while(fgets(buffer, bufferLength, filePointer)) {
                currUsername = strtok(buffer, " "); // get username from txt file
                currPassword = strtok(NULL, "\n"); // get password from txt file
                if(strcmp(username, currUsername)==0 && strcmp(password, currPassword)==0)
                {

```

```

    SSL_write(ssl, ServerValidResponse, strlen(ServerValidResponse)); /* send valid response */
}
}

SSL_write(ssl, ServerInvalidResponse, strlen(ServerInvalidResponse)); /* send not valid
response
*/
}
else
{
    ERR_print_errors_fp(stderr);
}
}

sd = SSL_get_fd(ssl); /* get socket connection */
SSL_free(ssl); /* release SSL state */
close(sd); /* close connection */
}

int main(int count, char *Argc[])
{
    SSL_CTX *ctx;
    int server;
    if ( count != 1 )
    {
        printf("No arguments needed! \n");
        printf("Usage: %s \n", Argc[0]);
        exit(0);
    }

    // Initialize the SSL library
    SSL_library_init();

    ctx = InitServerCTX(); /* initialize SSL */

    LoadCertificates(ctx, "key1.pem", "key2.pem"); /* load certs */

```

```

server = OpenListener(PORT); /* create server socket */
while (1)
{
    struct sockaddr_in addr;
    socklen_t len = sizeof(addr);
    SSL *ssl;
    int client = accept(server, (struct sockaddr*)&addr, &len); /* accept connection as usual */
    // printf("Connection: %s:%d\n",inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
    ssl = SSL_new(ctx); /* get new SSL state with context */
    SSL_set_fd(ssl, client); /* set connection socket to SSL state */
    Servlet(ssl); /* service connection */
}
close(server); /* close server socket */
SSL_CTX_free(ctx); /* release context */
}

```

Client-Side

```

#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <sys/socket.h>
#include <resolv.h>
#include <netdb.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#define FAIL -1
#define PORT 23451

```

```

int OpenConnection(const char *hostname, int port)
{
    int sd;
    struct hostent *host;
    struct sockaddr_in addr;
    if ( (host = gethostbyname(hostname)) == NULL )
    {
        perror(hostname);
        abort();
    }
    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = *(long*)(host->h_addr);
    if ( connect(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 )
    {
        close(sd);
        perror(hostname);
        abort();
    }
    return sd;
}

SSL_CTX* InitCTX(void)
{
    SSL_METHOD *method;
    SSL_CTX *ctx;
    OpenSSL_add_all_algorithms(); /* Load cryptos, et.al. */
    SSL_load_error_strings(); /* Bring in and register error messages */

```

```

method = TLSv1_2_client_method(); /* Create new client-method instance */
ctx = SSL_CTX_new(method); /* Create new context */
if ( ctx == NULL )
{
    ERR_print_errors_fp(stderr);
    abort();
}
return ctx;
}

int main(int count, char *strings[])
{
    SSL_CTX *ctx;
    int server;
    SSL *ssl;
    char buf[1024];
    char acClientRequest[1024] = {0};
    int bytes;
    char *hostname;
    if ( count != 2 )
    {
        printf("usage: %s <hostname> \n", strings[0]);
        exit(0);
    }
    SSL_library_init();
    hostname=strings[1];
    ctx = InitCTX();
    server = OpenConnection(hostname, PORT);
    ssl = SSL_new(ctx); /* create new SSL connection state */
    SSL_set_fd(ssl, server); /* attach the socket descriptor */

```



```

if ( SSL_connect(ssl) == FAIL ) /* perform the connection */
ERR_print_errors_fp(stderr);
else
{
char acUsername[16] = {0};
char acPassword[16] = {0};
const char *cpRequestMessage = "%s %s";
printf("Enter the User Name : ");
scanf("%s",acUsername);
printf("\nEnter the Password : ");
scanf("%s",acPassword);

sprintf(acClientRequest, cpRequestMessage, acUsername,acPassword); /* construct reply
*/

// printf("\nConnected with %s encryption\n", SSL_get_cipher(ssl));
SSL_write(ssl,acClientRequest, strlen(acClientRequest)); /* encrypt & send message */
bytes = SSL_read(ssl, buf, sizeof(buf)); /* get reply & decrypt */
buf[bytes] = 0;
printf("\nReceived: \"%s\"\n", buf);
SSL_free(ssl); /* release connection state */
}
close(server); /* close socket */
SSL_CTX_free(ctx); /* release context */
return 0;
}

```

Algorithm:-

- **Client Initialization:** The client initiates the connection by sending a request to the server.
- **Server Verification:** The server responds with its public key, which the client verifies against its local database or by receiving the key from a trusted source.

- **Encryption Algorithm Negotiation:** Both the client and server share their supported encryption protocols, and the entry with the highest server preference that is shared across both client and server wins, making it the algorithm used to encrypt the session.
- **Key Exchange:** The client and server perform a key exchange algorithm, such as DiffieHellman, to securely establish a shared secret key.
- **Symmetric Encryption:** The client and server use the shared secret key to encrypt and decrypt data using a symmetric encryption algorithm, such as AES.
- **User Authentication:** The client authenticates itself to the server using a method such as password authentication or SSH key pairs.
- **Access Granted:** Once the client is authenticated, the server grants access to the requested resources.

Output: -

```
oslab@oslab-VirtualBox:~/Desktop/SSL/Client-Server-Communication-using-SSL-master
r$ ./sslclient 127.0.0.1
Enter the User Name : alice

Enter the Password : 12345

Received: "Correct ID and Password"
oslab@oslab-VirtualBox:~/Desktop/SSL/Client-Server-Communication-using-SSL-master
r$ ./sslclient 127.0.0.1
Enter the User Name : bob

Enter the Password : 67890

Received: "Correct ID and Password"
oslab@oslab-VirtualBox:~/Desktop/SSL/Client-Server-Communication-using-SSL-master
r$ ./sslclient 127.0.0.1
Enter the User Name : alice

Enter the Password : 14628

Received: "Incorrect ID and Password"
oslab@oslab-VirtualBox:~/Desktop/SSL/Client-Server-Communication-using-SSL-master
r$ ./sslserver
Enter PEM pass phrase:
Client msg: "alice 12345"
Client msg: "bob 67890"
Client msg: "alice 14628"
```



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Cryptography & Network Security Lab

BCSE309P

Lab-13

Web Application – JSON Web Token

Name: Vaibhav Khandelwal

RegNo-21BCE5550

Date-22-04-2024

Develop a web application that implements JSON web token.

Code: -

Frontend HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<link rel="icon" type="image/x-icon" href="images/favicon.ico">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<script src="https://kit.fontawesome.com/0375b55703.js"
crossorigin="anonymous"></script>
<link rel="stylesheet" href="login-page/login.css">
<script src="../bidding-page/login-page/login.js" defer></script>
<script src="https://unpkg.com/axios/dist/axios.min.js" defer></script>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3y
D65VohhpUuCOMLASjC" defer crossorigin="anonymous">
```

```

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIa
xVXM" defer crossorigin="anonymous"></script>

<title>Login Page</title>

</head>

<body class="vh-100 gradient-custom">

<section>

<div class="container py-5 vh-100">

<div class="row d-flex justify-content-center align-items-center h-100">

<div class="col-12 col-md-8 col-lg-6 col-xl-5">

<div class="card bg-dark text-white" style="border-radius: 1rem;">

<div class="card-body p-5 mb-4">

<div class="mb-md-5 mt-md-4 pb-1">

<h2 class="fw-bold mb-4 mt-2 text-uppercase text-center">Login</h2>

<form>

<div class="form-floating form-white mb-4">

<input type="text" autocomplete="off" id="username" class="form-control form-control-lg"
placeholder="E-mail" />

<label style="color: black;" class="form-label" for="username">E-mail</label>

</div>

<div class="form-floating">

<input type="password" autocomplete="off" id="password" class="form-control"
placeholder="Password" />

<label style="color: black;" class="form-label" for="password">Password</label>

</div>

</form>

<div class="d-flex justify-content-center mb-3 mt-3">

<button id="signin" class="btn btn-outline-light btn-lg px-5">Login</button>

</div>

<div>

```



```

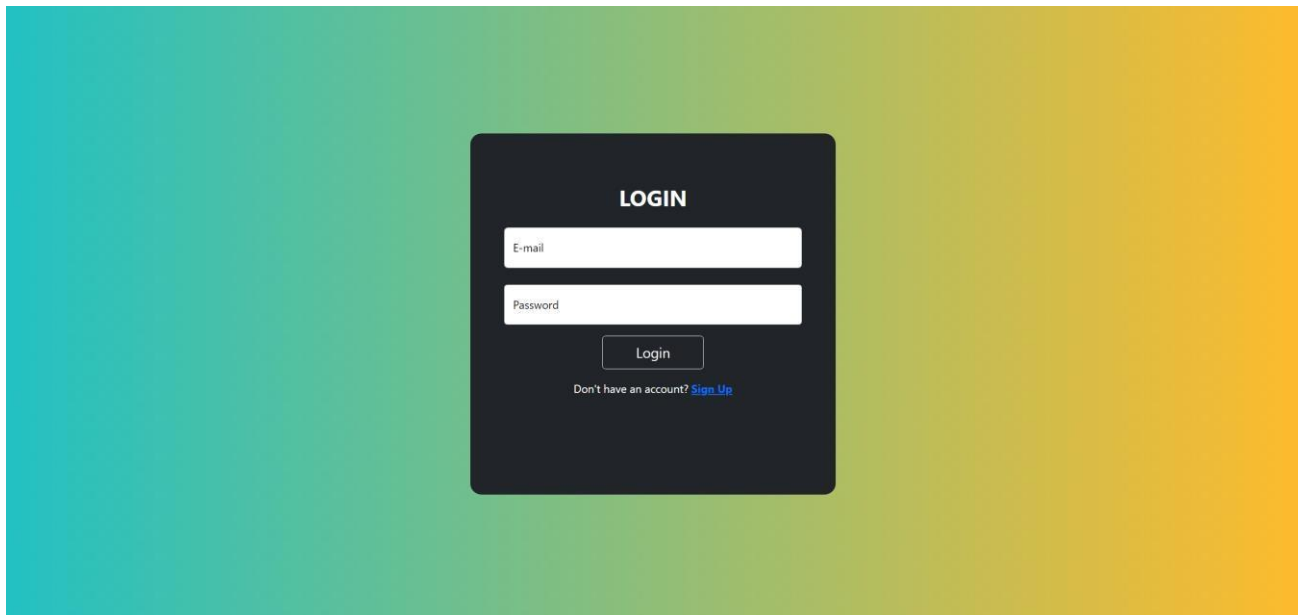
fastify.get('/', (req, reply) => { const sid = req.query.sid
let error = false let session = null
if (sid) { try {
session = jwt.verify(sid, jwtSecret)
} catch (err) {
error = 'Invalid session token'
}
}
reply.view('index.pug', {session, error})
})
fastify.get('/login', (req, reply) => { console.log(req.query)
reply.view('login.pug', {hasError: req.query.error})
})
fastify.post('/login', (req, reply) => { const {username, password} = req.body if (
username !== expectedUsername || password !== expectedPassword
){
return reply.redirect('/login?error=1')
}
// create jwt token
const jwtToken = jwt.sign({ sub: username, role: 'user' }, jwtSecret)
// redirect with parameter
return reply.redirect(`/?sid=${jwtToken}`)
})
next()
}

```

Algorithm:

1. User Login:
 - User provides valid credentials.
 - Server generates a JWT token containing user info.
 - Token is signed using a secret key.
2. Token Sending:
 - Server sends the token to the client.
3. Token Storage:
 - Client securely stores the token (e.g., local storage, cookie).
4. Subsequent Requests:
 - Client sends token with each request (usually in header).
5. Token Verification:
 - Server verifies token signature and expiration.
 - Allows request if token is valid; otherwise denies it.
6. Optional: Token Refresh:
 - Implement a mechanism to refresh tokens before expiry.
7. Optional: Logout:
 - Provide a way to invalidate tokens on logout.
8. Security Considerations:
 - Use HTTPS.
 - Keep secret key secure.
 - Protect against XSS and CSRF attacks.

Output:



The code generates a JWT containing the user's name and signs it. The token is verified while login to ensure that the user is authentic.