Name: Aryan Sharma
Reg. No.: 23BCE1320
Subject: Cryptography & Network Security Lab 2

Codes:

```
import java.util.*;

public class DES {
    private static final int[] IP = {
        58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
        62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
        57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
        61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7
    };
    private static final int[] IP_INV = {
        40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25
    };
    private static final int[] PC1 = {
        57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2,
        59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39,
        31, 23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37,
        29, 21, 13, 5, 28, 20, 12, 4
    };
    private static final int[] PC2 = {
        14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,
        26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,
        51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32
    };
    private static final int[] E = {
        32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13,
        12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
        24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1
    };
    private static final int[] P = {
        16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
        2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25
    };
    public static String encrypt(String plainText, String key) {
        return process(plainText, key, true);
    }

    public static String decrypt(String hexCipher, String key) {
        return process(hexCipher, key, false);
    }

    private static String process(String input, String key, boolean isEncrypt) {
        int[] keyBits = getBits(key);
        int[][] subKeys = generateKeys(keyBits);
        if (!isEncrypt) {
            int[][] temp = new int[16][48];
            for (int i = 0; i < 16; i++) temp[i] = subKeys[15 - i];
            subKeys = temp;
        }

        StringBuilder result = new StringBuilder();
        if (isEncrypt) {
            while (input.length() % 8 != 0) input += " ";
            for (int i = 0; i < input.length(); i += 8) {
                String block = input.substring(i, i + 8);
```

```
                int[] blockBits = getBits(block);
                int[] encryptedBits = processBlock(blockBits, subKeys);
                result.append(binToHex(encryptedBits));
            }
        } else {
            for (int i = 0; i < input.length(); i += 16) {
                String block = input.substring(i, i + 16);
                int[] blockBits = hexToBin(block);
                int[] decryptedBits = processBlock(blockBits, subKeys);
                result.append(bitsToString(decryptedBits));
            }
        }
        return result.toString().trim();
    }

    private static int[] processBlock(int[] blockBits, int[][] subKeys) {
        blockBits = permute(blockBits, IP);
        int[] L = Arrays.copyOfRange(blockBits, 0, 32);
        int[] R = Arrays.copyOfRange(blockBits, 32, 64);

        for (int i = 0; i < 16; i++) {
            int[] tempR = Arrays.copyOf(R, 32);
            int[] expandedR = permute(R, E);
            int[] xored = xor(expandedR, subKeys[i]);
            int[] sboxOutput = sBoxSubstitution(xored);
            int[] pOutput = permute(sboxOutput, P);
            R = xor(L, pOutput);
            L = tempR;
        }

        int[] combined = new int[64];
        System.arraycopy(R, 0, combined, 0, 32);
        System.arraycopy(L, 0, combined, 32, 32);
        return permute(combined, IP_INV);
    }

    private static int[][] generateKeys(int[] keyBits) {
        int[][] keys = new int[16][48];
        keyBits = permute(keyBits, PC1);
        int[] C = Arrays.copyOfRange(keyBits, 0, 28);
        int[] D = Arrays.copyOfRange(keyBits, 28, 56);

        for (int i = 0; i < 16; i++) {
            C = leftShift(C, SHIFTS[i]);
            D = leftShift(D, SHIFTS[i]);
            int[] combined = new int[56];
            System.arraycopy(C, 0, combined, 0, 28);
            System.arraycopy(D, 0, combined, 28, 28);
            keys[i] = permute(combined, PC2);
        }
        return keys;
    }

    private static int[] sBoxSubstitution(int[] in) {
        int[] out = new int[32];
        for (int i = 0; i < 8; i++) {
            int row = (in[i * 6] << 1) | in[i * 6 + 5];
            int col = (in[i * 6 + 1] << 3) | (in[i * 6 + 2] << 2) | (in[i * 6 + 3]
<< 1) | in[i * 6 + 4];
            int val = S_BOX[i][row][col];
            for (int j = 0; j < 4; j++) out[i * 4 + j] = (val >> (3 - j)) & 1;
        }
        return out;
    }

    private static int[] permute(int[] in, int[] table) {
```

```
        int[] out = new int[table.length];
        for (int i = 0; i < table.length; i++) out[i] = in[table[i] - 1];
        return out;
    }

    private static int[] leftShift(int[] bits, int n) {
        int[] out = new int[bits.length];
        System.arraycopy(bits, n, out, 0, bits.length - n);
        System.arraycopy(bits, 0, out, bits.length - n, n);
        return out;
    }

    private static int[] xor(int[] a, int[] b) {
        int[] out = new int[a.length];
        for (int i = 0; i < a.length; i++) out[i] = a[i] ^ b[i];
        return out;
    }

    private static int[] getBits(String s) {
        int[] bits = new int[s.length() * 8];
        for (int i = 0; i < s.length(); i++) {
            int val = s.charAt(i);
            for (int j = 0; j < 8; j++) bits[i * 8 + j] = (val >> (7 - j)) & 1;
        }
        return bits;
    }

    private static String bitsToString(int[] bits) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < bits.length; i += 8) {
            int val = 0;
            for (int j = 0; j < 8; j++) val = (val << 1) | bits[i + j];
            sb.append((char) val);
        }
        return sb.toString();
    }

    private static String binToHex(int[] bits) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < bits.length; i += 4) {
            int val = 0;
            for (int j = 0; j < 4; j++) val = (val << 1) | bits[i + j];
            sb.append(Integer.toHexString(val).toUpperCase());
        }
        return sb.toString();
    }

    private static int[] hexToBin(String hex) {
        int[] bits = new int[hex.length() * 4];
        for (int i = 0; i < hex.length(); i++) {
            int val = Integer.parseInt(String.valueOf(hex.charAt(i)), 16);
            for (int j = 0; j < 4; j++) bits[i * 4 + j] = (val >> (3 - j)) & 1;
        }
        return bits;
    }
}
```

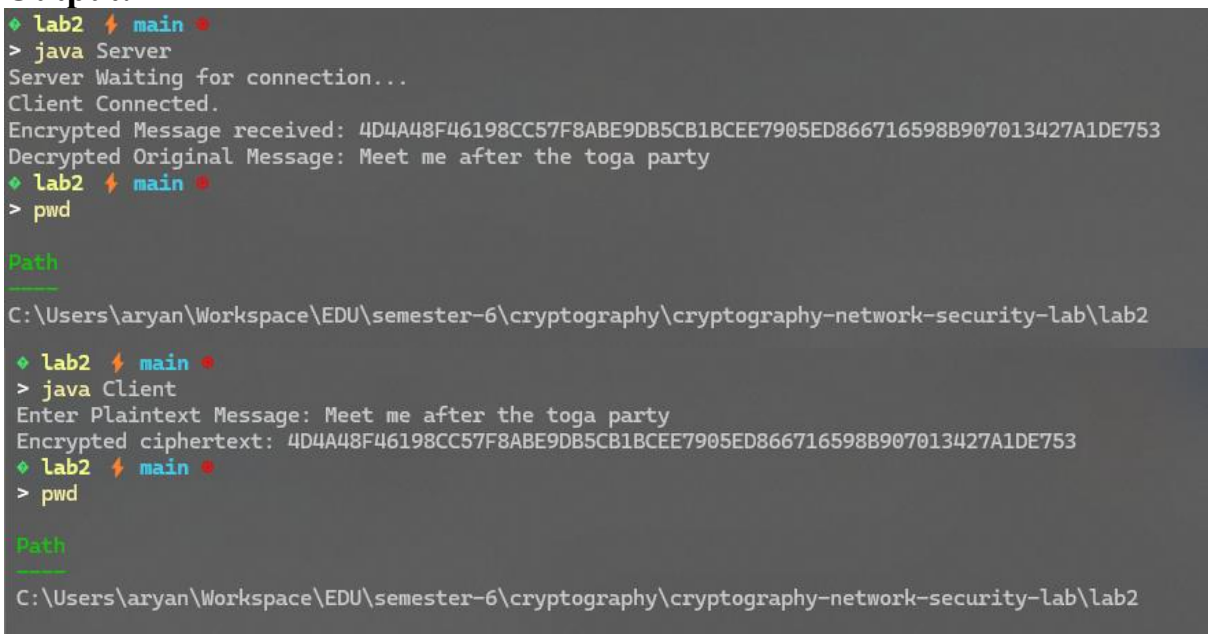
Client.java

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

public class Client {
    public static void main(String[] args) {
        try {
            String key = "SECRETKEY";
            if(key.length() != 8) key = "12345678";

            Socket socket = new Socket("localhost", 5000);
            DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
            Scanner sc = new Scanner(System.in);

            System.out.print("Enter Plaintext Message: ");
            String plainText = sc.nextLine();

            String encryptedMsg = DES.encrypt(plainText, key);
            System.out.println("Encrypted ciphertext: " + encryptedMsg);

            dos.writeUTF(encryptedMsg);

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

Server.java

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

public class Server {
    public static void main(String[] args) {
        try {
            String key = "SECRETKEY";
            if(key.length() != 8) key = "12345678";

            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server Waiting for connection...");

            Socket socket = serverSocket.accept();
            System.out.println("Client Connected.");

            DataInputStream dis = new DataInputStream(socket.getInputStream());
            String encryptedMsg = dis.readUTF();

            System.out.println("Encrypted Message received: " + encryptedMsg);

            String decryptedMsg = DES.decrypt(encryptedMsg, key);
            System.out.println("Decrypted Original Message: " + decryptedMsg);

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

Output:


lab2 main
java Server
Server Waiting for connection...
Client Connected.
Encrypted Message received: 4D4A48F46198CC57F8ABE9DB5CB1BCEE7905ED866716598B907013427A1DE753
Decrypted Original Message: Meet me after the toga party
lab2 main
pwd
Path
C:\Users\aryan\Workspace\EDU\semester-6\cryptography\cryptography-network-security-lab\lab2
lab2 main
java Client
Enter Plaintext Message: Meet me after the toga party
Encrypted ciphertext: 4D4A48F46198CC57F8ABE9DB5CB1BCEE7905ED866716598B907013427A1DE753
lab2 main
pwd
Path
C:\Users\aryan\Workspace\EDU\semester-6\cryptography\cryptography-network-security-lab\lab2