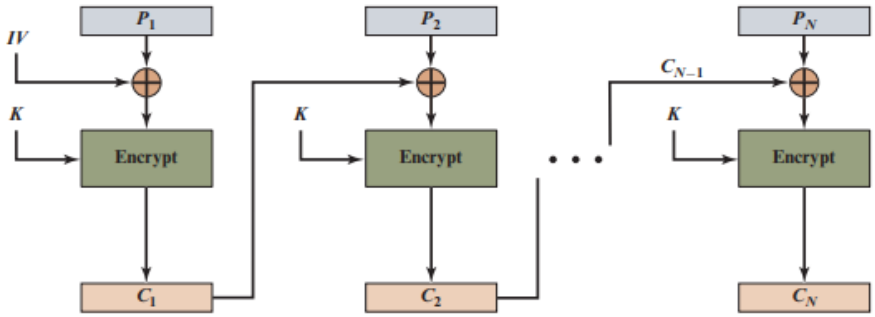
LAB SESSION 04

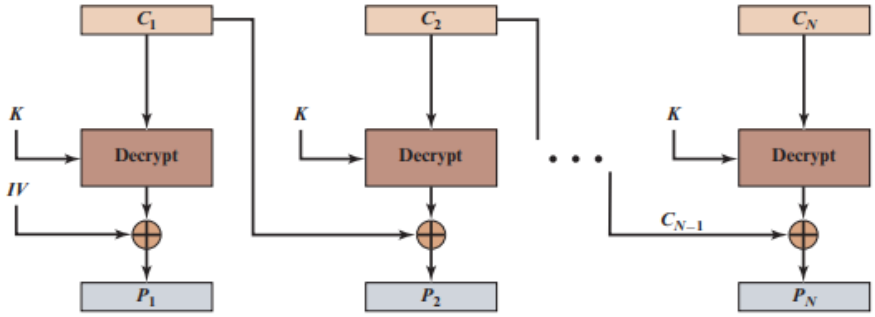
IMPLEMENTATION OF DES USING CIPHER BLOCK CHAINING MODE (CBC) ON TEXTS OF SIZE 512BITS, 1KB AND 10 KB)

BY: BHUVANA PRABHA B (22BCE1639)

ENCRYPTION MODE USED:



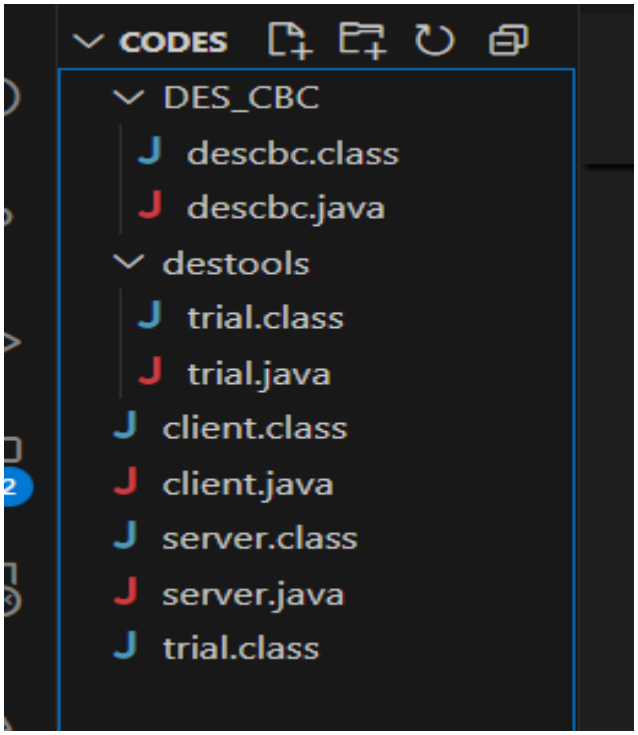
(a) Encryption



(b) Decryption

Figure 7.4 Cipher Block Chaining (CBC) Mode

ORGANIZATION OF CODE FILES:



- **The DES_CBC custom package contains the descbc.java and descbc.class source files of the Cipher block chain encryption mode (using des algorithm).**
- **The destools custom package contains the trial.class and trial.java source files which contain the entire structure of DES algorithm**
- **The rest files (client.java, client.class) and (server.java, server.class) follow the client-server-based architecture for the mode of communication.**

CODE:

Trial.java file (The des algorithm)

```
package destools;

import java.util.*;

//The DES Algorithm here is using 64 bits of plain text (as one block of plain
text).
//So each block should consider of 8 characters
//And the overall text length should be a multiple of 8.

/*
 * NOTE!!! :
 * I have specified my master key as "EFGHHFGE" in the server.java file. Use
the same master key when asked for input on the client side
```

```
 * Or you can have your own customised master key, but just make sure to
change into the same on the server side file
 * I have commented out the print statements everywhere
 * If you have doubt regarding any process, uncomment all or some of the print
statements accordingly
 */

public class trial{

    //Initial permutation over the plain text
    public static final int[] ip_table = {
        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
    };

    //Expansion permutation table(32 to 48 bit) for plain text
    public static int[] EP = {
            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
    };

        static final int[][][] S_BOXES = {
        {
          {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
          {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},
          {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
          {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
        },

        {
        {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
        {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
        {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
        {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
        },
```

```
        {
        {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
        {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
        {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
        {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
        },

        {
        {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
        {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
        {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
        {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
        },

        {
        {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
        {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
        {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
        {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
        },

        {
        {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
        {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
        {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
        {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13}
        },

        {
        {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
        {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
        {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
        {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
        },

        {
        {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
        {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
        {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
        {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
        }
    };

    //Straight P-Box permutation
    public static int[] P_BOX = {
        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 int[]  pc1_table = {
        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
    };

    //Compression permutation table(56 to 48 bit in key generation)
    public static int[] pc2_table = {
            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
        };

    //Final permutation
    public static int[] FP = {
    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
    };

    //shift schedule
    public static int[] shift_schedule = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
2, 2, 2, 1};

    //List to store all the 16 roundkeys
    public static List<String> roundKeys = new ArrayList<>();
    public static List<String> reverseRoundKeys;

        public static String initialKeyPerm(String key){
        //convert master key to 64-bit binary
        StringBuilder binaryKey = new StringBuilder();
        for(char c : key.toCharArray()){
```

```
            binaryKey.append(String.format("%8s",
Integer.toBinaryString(c)).replace(' ', '0'));
        }

        //System.out.println("64-bit Binary Key: " + binaryKey);


        //Perform permutation using PC1 table
        StringBuilder permutedKey = new StringBuilder();
        for(int position : pc1_table)
          permutedKey.append(binaryKey.charAt(position-1)); //Adjust for zero
based indexing

        //System.out.println("56-bit Key after PC1 permutation: " +
permutedKey);
        return permutedKey.toString();
    }//method end

    public static String leftCircularShift(String input, int shifts){
    return input.substring(shifts) + input.substring(0, shifts);
   }//method end

        public static void generateSubKey(String key){

        //Split the 56 bit key into te 28 bits parts
         String leftPart = key.substring(0, 28);
         String rightPart = key.substring(28);

         int cumulativeShifts = 0;
         for(int round = 0; round < 16; round ++){
            //Determine the number of shifts for this round
            cumulativeShifts += shift_schedule[round];

            //Perform left circular shift on both parts
            leftPart = leftCircularShift(leftPart, cumulativeShifts);
            rightPart = leftCircularShift(rightPart, cumulativeShifts);

            //Combine the two parts into a 56 bit key
            String combinedKey = leftPart + rightPart;

            //Apply PC2 to compress the key to 48 bits
            StringBuilder compressedKey = new StringBuilder();
            for(int position : pc2_table){
                compressedKey.append(combinedKey.charAt(position-
1));//Adjusting for the zero based indexing
            }//for end

            //Store the round key
            roundKeys.add(compressedKey.toString());
```

```
            //System.out.println("Round " + (round+1) + " key: " +
compressedKey);
         }//for end
    }//method end

    public static String initialTextMut(String text){

        //Convert 8-character length plaintext to binary(64 bits)
    StringBuilder binaryPlainText = new StringBuilder();
    for(char c : text.toCharArray()){
        binaryPlainText.append(String.format("%8s",
Integer.toBinaryString(c)).replace(' ', '0'));
    }

     //System.out.println("64-bit Binary Representation of plaintext: ");
     //System.out.println(binaryPlainText);
     return binaryPlainText.toString();
}//method end


    public static String applyPermutation(String input, int[] table){
    StringBuilder output = new StringBuilder();
    for(int index : table){
        output.append(input.charAt(index-1)); //0-based indexing
    }//for end

    return output.toString();
   }//method end

   public static String xorStrings(String a, String b){
    StringBuilder output = new StringBuilder();
    for(int i = 0; i < a.length(); i++)
      output.append(a.charAt(i) ^ b.charAt(i));

    //System.out.println("XOR Result:(with corresponding subkey)
"+output.toString());
    return output.toString();
   }//method end

   public static String applySBoxes(String input){
    StringBuilder output = new StringBuilder();

    //Divide input into 8 blocks of 6 bits each
    for(int i = 0; i < 8; i++){
        String block = input.substring(i*6, (i+1)*6);

        //Row: first and last bit, Column: the middle 4 bits
        int row = Integer.parseInt(""+block.charAt(0)+block.charAt(5), 2);
        int column = Integer.parseInt(block.substring(1, 5), 2);
```

```
        //Get value from S-box
        int value = S_BOXES[i][row][column];

        output.append(String.format("%4s",
Integer.toBinaryString(value)).replace(' ', '0'));
    }//for end

    return output.toString();
   }

    public static String feistalRounds(String plaintext, List<String>
roundKeys){

   //Divide the String into 2 equal halves
   String left = plaintext.substring(0, 32);
   String right = plaintext.substring(32);

   for(int round = 0; round < 16; round ++){

    //System.out.println("\nRound: " + (round+1));
    //System.out.println("Input to round: " + left + " " + right);

   //Expansion permutation
   String expandedRight = applyPermutation(right, EP);
   //System.out.println("Expanded right half:" + expandedRight);

   //XOR with corresp. round subkey
   String xorResult = xorStrings(expandedRight, roundKeys.get(round));

   //S-box substitution
   String sBoxOutput = applySBoxes(xorResult);
   //System.out.println("S-box output:" + sBoxOutput);

   //P-box permutation
   String pBoxOutput = applyPermutation(sBoxOutput, P_BOX);
   //System.out.println("P-box output:" + pBoxOutput);

   //Apply XOR operation with left 32 bit
   String output = xorStrings(pBoxOutput, left);

   //Switch the modified right half and left of the plain 54 bit text
   left = right;
   right = output;

   //System.out.println("After switching, the halves(output of round): " +
left + " " + right + " \n");
```

```
   }//for end - end of 16 rounds

   return left+right;

   }//method end

   //perform the final reversal of the (output of round 16) plain text as
RE16,LE16
   public static String finalReversal(String plaintext){

   //Divide the String into 2 equal halves
   String left = plaintext.substring(0, 32);
   String right = plaintext.substring(32);

   //System.out.println("\nfinal reversal: " + right + " " + left);

   return right+left;

   }//method end

   public static void keyGeneration(String masterKey){
        //Perform initial modifications on the master key
            masterKey = initialKeyPerm(masterKey);

            //Generate 48bit subkeys for all the 16 rounds
            generateSubKey(masterKey);

            //Store these generated round keys in reverse order for decryption
            reverseRoundKeys = new ArrayList<>(roundKeys);
            Collections.reverse(reverseRoundKeys);

            //System.out.println("\nThe reversed round keys for decryption");
            /*for(int i = 0; i < reverseRoundKeys.size(); i++){
                System.out.println("Round " + (i+1) + " key: " +
reverseRoundKeys.get(i));
            }//for end*/
   }//method end

   public static String encryption(String plaintext){
          //perform initial modifications on the text
          /*Commenting this out, since initial modifications will be performed
on the main text in the Cipher chaining mode
           * for each 64 bit block of plain text and the ending plain text
will be padded
           */
            //plaintext = initialTextMut(plaintext);

            //Perform initial Permutation
```

```
            plaintext = applyPermutation(plaintext, ip_table);
            //System.out.println("Plaintext after initial permutation: " +
plaintext);

            //perform the 16 round modifications on the text
            plaintext = feistalRounds(plaintext, roundKeys);

            //perform final reversal and apply IP_1(Ip inverse) permutation
            String cipherText = applyPermutation(finalReversal(plaintext),
FP);
            return cipherText;

   }//method end

   public static String toActualString(String binString){
    StringBuilder result = new StringBuilder();

    //Ensure that the binary string length is a multiple of 8
    if(binString.length() % 8 != 0){
        throw new IllegalArgumentException("Binary string length must be a
multiple of 8.");
    }//if end

    //Process each 8-bit chunk
    for(int i = 0; i < binString.length(); i+= 8){
        String byteString = binString.substring(i, i+8);//Get 8-bit chunk
        int charCode = Integer.parseInt(byteString, 2); //Convert to decimal
        result.append((char) charCode);
    }//for end

    return result.toString();
   }//method end

   public static String decryption(String cipherText){
            //Apply IP_1(Ip inverse) permutation
            cipherText = applyPermutation(cipherText, ip_table);
            //System.out.println("\nCipher text after applying Initial
permutation: "+cipherText);

            //Perform the reversal
            //cipherText = finalReversal(cipherText);

            //Perform the 16 round modifications on the text
            cipherText = feistalRounds(cipherText, reverseRoundKeys);

            //Perform the reversal
            cipherText = finalReversal(cipherText);
```

```
            //Perform the Initial permutation on the cipher text
            String decryptedText = applyPermutation(cipherText, FP);
            //System.out.println("\nCipher text after applying Final
permutation: " + decryptedText);

            //return toActualString(decryptedText);
            return decryptedText;
   }//method end

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

            //Enter the master key
            System.out.println("Enter a master key (8 characters): ");
            String masterKey = sc.nextLine();

            // Ensure the master key is exactly 8 characters
            while (masterKey.length() != 8) {
            System.out.println("Invalid input. Please enter exactly 8
characters: ");
            masterKey = sc.nextLine();
            }

            //Gemeration of keys method
            keyGeneration(masterKey);

          //Enter the plain text to send to the client
            System.out.println("Enter 8 characters(plaintext): ");
            String plaintext = sc.nextLine();

            //Ensure the input is exactly 8 characters long
            while(plaintext.length() != 8){
                System.out.println("Error: The input must be exactly 8
character (64 bits).");
                plaintext = sc.nextLine();
            }//while end

            //perform encryption
            String cipherText = encryption(plaintext);

            System.out.println("The encrypted text is: " + cipherText);

            System.out.println("Decryption of the text: ");

            //Perform the decryption of cipherText
            String decryptedText = decryption(cipherText);
            System.out.println("\nDecrypted text: " + decryptedText);
```

```
        sc.close();
    }//main end
}//class end
```

2. descbc.java source file(Cipher block chaining encryption mode)

```
package DES_CBC;
//import java.util.*;
import destools.trial;

public class descbc {

    public static String iv = "AAAAAAAA";
    //public static String IV = trial.initialTextMut(iv);
    //public static List<String> cipherTexts = new ArrayList<>();
    //ensure that the plaintext charcater length is a multiple of 8(else padd
it)
        public static String ensureMultipleOf8(String text){
        int length = text.length();
        int paddingNeeded = (8-(length%8)) % 8; //calculate the padding needed
        return text + "X".repeat(paddingNeeded);
    }//method end

    public static String cbc_Encrypt(String plaintext){
         plaintext = ensureMultipleOf8(plaintext);
         String operand = trial.initialTextMut(iv);
         StringBuilder cipherTexts = new StringBuilder();

        //Extract 8 bit character from the plain text
        for(int i = 0; i < plaintext.length(); i += 8){

            //Extract 8 bit character from the plain text
            String text = plaintext.substring(i, i+8);
            //System.out.println("Extracted plain text for encryption is: " +
text);

            //Perform the xor operation with the plaintext block and
iv(initially, will be later replaced with cipher text from previous
encryption)
            //First convert them into binary string using initalTextMut and
then apply the xorStrings...
            String sxor = trial.xorStrings(trial.initialTextMut(text),
operand);
```

```
            //Perform encryption on the XOR-ed output
            String cipher = trial.encryption(sxor);
            //Binary strings(encrpyted text) are stored in the cipherTexts
list(so must by of 64 bit sized blocks of encrypted (binary string) message)
            cipherTexts.append(cipher);

            //The second string involved in the XOR operatioon with the plain
text
            operand = cipher;
        }//for end

         return cipherTexts.toString();
    }//method end

     public static String cbc_decrypt(String ciphertext){

        String operand = trial.initialTextMut(iv);
        StringBuilder plainTexts = new StringBuilder();

        //Extracting 64 bits of the ciphered text or rather 8 character length
of cipher text block
        //64 bit cause the cipher text here in the function is of binaryString
form
        for(int i = 0; i < ciphertext.length(); i += 64){
            String cipher = ciphertext.substring(i, i+64);

            //decrypt the ciphered text
            String output = trial.decryption(cipher);

            //Perform the xor operation with the cipher text and iv(initially,
will be later replaced with cipher text from previous decryption)
            String text = trial.xorStrings(operand, output);

            plainTexts.append(trial.toActualString(text));
            operand = cipher;

        }//for end
        return plainTexts.toString();
     }//method end

}//class end
```

3. Client.java source file

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

```
import java.net.*;
import destools.trial;
import DES_CBC.descbc;
public class client {

    public static void main(String[] args){
       String serverAddress = "localhost";
       int port = 8081;

       try(Socket socket = new Socket(serverAddress, port)){

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

        //Send message to server
        Scanner sc = new Scanner(System.in);
        System.out.println("Connected to the server. Please type as
required...");
        //Enter the master key
            System.out.println("Enter a master key (8 characters): ");
            String masterKey = sc.nextLine();

            // Ensure the master key is exactly 8 characters
            while (masterKey.length() != 8) {
            System.out.println("Invalid input. Please enter exactly 8
characters: ");
            masterKey = sc.nextLine();
            }

            trial.keyGeneration(masterKey);

          //Enter the plain text to send to the server
            System.out.println("Enter the plain text to be sent to the server:
");
            String plaintext = sc.nextLine();


           //Inorder to measure the time taken to encrypt this text
           long startEncrypt = System.nanoTime();
           //ENCRYPTION PERFORMED
            String cipherText = descbc.cbc_Encrypt(plaintext);
           long endEncrypt = System.nanoTime();
           System.out.println("Time taken to encrypt the text: " + plaintext +
" is " + (endEncrypt - startEncrypt) + " nanoseconds");
```

```
        //System.out.println("The encrypted text is: " + cipherText);
        out.println(cipherText);

        //Read response from server
        String response = in.readLine();
        //System.out.println("Crypted message From the server: "+ response);

        System.out.println("Decryption of the text(sent from server)...");

            //Apply decryption
            String decryptedText = descbc.cbc_decrypt(response);

            System.out.println("\nDecrypted text: " + decryptedText);

        sc.close();

       } catch(IOException e){
        e.printStackTrace();
       }//try catch end

    }//main end
}//class end
```

4. Server.java source file

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

import DES_CBC.descbc;
import destools.trial;

//Assuming that the server knows the master key used already
public class server {

    public static void main(String[] args){
        int port = 8081; //port number where server listens

        try(ServerSocket serverSocket = new ServerSocket(port)){
            System.out.println("server is listening on port "+port);

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

            //Create input and output Streams
```

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

            //Assuming that the master key is already known.
            /* Please note!
             * Use the same master key value as provided by the user in the
client side
             */
            String masterKey = "EFGHHFGE";
            trial.keyGeneration(masterKey);

            //Read message from client
            String message = in.readLine();
            //System.out.println("Encrypted message Received: "+ message);

            System.out.println("Decryption of the text from the client...");


            //Measuring the time taken to decrypt the text
            long startDecrypt = System.nanoTime();
            //Apply decryption
            String decryptedText = descbc.cbc_decrypt(message);
            long endDecrypt = System.nanoTime();
            System.out.println("The time taken to decrypt the same text (which
was encrypted at the client side is): " + (endDecrypt - startDecrypt) + "
ns");
            System.out.println("\nDecrypted text: " + decryptedText);

            //Send response to client
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter a response to send to the client: ");

          //Enter the plain text to send to the client
            System.out.println("Enter 8 characters(plaintext): ");
            String plaintext = sc.nextLine();

            //perform initial modifications on the text
            String cipherText = descbc.cbc_Encrypt(plaintext);
            //System.out.println("The encrypted text is: " + cipherText);

            out.println(cipherText);

            sc.close();
            in.close();
            out.close();
            socket.close();
        } catch (IOException e){
            e.printStackTrace();
```

```
        }//try catch end

    }//main end

}//class end
```

OUTPUTS:

(The user first has to enter the master key value and the text to be sent on the client side. The decrypted text will be printed on the server side. Then. the user has to enter the text on the server side(I have used “over” for simplicity, since I am mainly focusing on the time parameter) which will be encrypted and sent to the client side, the client side decrypts this text and displays the decrypted text.)

NOTE: Partial plaintext blocks are appended with “x” character so that it becomes a block of 8 character size (or 64 bit size) to be used by the DES algorithm.

1. **512 bits of text:**

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes> java client
Connected to the server. Please type as required...
Enter a master key (8 characters):
EFGHHFGE
Enter the plain text to be sent to the server:
MISSIONOPERATIONEAGLESTRIKEAT0400HRSCOORDINATESSECTOR7G
Time taken to encrypt the text: MISSIONOPERATIONEAGLESTRIKEAT0400HRSCOORDINATESSECTOR7G is 56383100 na
noseconds
Decryption of the text(sent from server)...

Decrypted text: OVERXXXX
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes>
```

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes> java server
server is listening on port 8081
Client is connected
Decryption of the text from the client...
The time taken to decrypt the same text (which was encrypted at the client side is): 49241400 ns

Decrypted text: MISSIONOPERATIONEAGLESTRIKEAT0400HRSCOORDINATESSECTOR7GX
Enter a response to send to the client:
Enter 8 characters(plaintext):
OVER
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes>
```

2. **10 kb of text:**

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes> java client
Connected to the server. Please type as required...
ALPHA.COMMANDERAPRROVALREQUIREDFORTACTICALENGAGEMENT.OPERATIONGHOSTSHADOWREMAINSCLASSIFIED.PRIORITYNATIONALSECURITY
.ENDTRANSMISSION.
Time taken to encrypt the text: SECURETRANSMISSIONCODERED.ENEMYMOVEMENTSDETECTEDNEARBORDEROUTPOSTCHARLIE.SATELLITEI
MAGERYCONFIRMSHEAVYARTILLERYPRESENCE.AIRSTRIKEREQUESTPENDINGCONFIRMATION.TROOPSONHIGHALERT.ALLUNITSTOMAINTAINRADIOS
ILENCEUNTILFURTHERORDERS.SUPPLYFROPSSCHDULEDAT2200HRS.REINFORCEMENTSARRIVINGFROMBASEALPHA.COMMANDERAPRROVALREQUIRED
FORTACTICALENGAGEMENT.OPERATIONGHOSTSHADOWREMAINSCLASSIFIED.PRIORITYNATIONALSECURITY.ENDTRANSMISSION. is 110505600
nanoseconds
Decryption of the text(sent from server)...

Decrypted text: OVERXXXX
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes>
```

```
TERMINAL                                                    powershell
  The time taken to decrypt the same text (which was encrypted at the client side is): 135975800 ns

  Decrypted text: SECURETRANSMISSIONCODERED.ENEMYMOVEMENTSDETECTEDNEARBORDEROUTPOSTCHARLIE.SATELLITEIMAGERYCONFIRMSHE
  AVYARTILLERYPRESENCE.AIRSTRIKEREQUESTPENDINGCONFIRMATION.TROOPSONHIGHALERT.ALLUNITSTOMAINTAINRADIOSILENCEUNTILFURTH
  ERORDERS.SUPPLYFROPSSCHDULEDAT2200HRS.REINFORCEMENTSARRIVINGFROMBASEALPHA.COMMANDERAPRROVALREQUIREDFORTACTICALENGAG
  EMENT.OPERATIONGHOSTSHADOWREMAINSCLASSIFIED.PRIORITYNATIONALSECURITY.ENDTRANSMISSION.XX
  Enter a response to send to the client:
  Enter 8 characters(plaintext):
  OVER
  PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes>
```

3. 10KB of text

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes> java client
Connected to the server. Please type as required...
Enter a master key (8 characters):
EFGHHFGE
Enter the plain text to be sent to the server:
JEBFYBE VOHUF 8YVBHFR8VNYDNFIHVKEWYO8FVN HFE8YOENWVFHVNO8YERHNGFLKNODSKL;OIEWVNOYGRFNYEVNHRN8FVOBGYERNGOIUFNEMVDBNS
FMCOEWHRPOIwj bnvkvnfgiernbvferyngvrejfroerinhmcfhe9uhnvkergnvfirynohrygfvdlkjfhuo8ydhkjfnhergndvhfriuhfyrcgnohuroi
runvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyugrvhewpu9nhc
egxnoiurnvceyrgnrfeugncvuiuygrtnrenycxegydchgfrydfhfbrhg438yrvrygt875bvkjhgncorncsdh8fhbvhfyghbvaoshdufoyyrgbv7ryvh
dfbvbythbfhyhvbhjgfvdhfoiuhrubvgjdkhkfudhgjkjdtybvjdfkncvvfgvjdkfgfdnkvhguierhvhghhkbvcnhifdghhdfghdsnlv
gfyghdfkyfrbkvhncmbhfbgkydfghldnhuerhvgfkdvdhfc
efhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyugrvhewpu9nhcegxnoiurnvceyrgnrfeu
gncvuiuygrtnrenycxegydchgfrydfhfbrhg438yrvrygt875bvkjhgncorncsdh8fhbvhfyghbvaoshdufoyyrgbv7ryvhdfbvbythbfhyhvbhjgfv
dhfoiuhrubvgjdkhkfudhgjkjdtybvjdfkncvvfgvjdkfgfdnkvhguierhvhghhkbvcnhifdghhdfghdsnlvgfyghdfkyfrbkvhncmbhfbgkydfghld
nhuerhvgfkdvdhfcvkgkfdgkbvskdhsfhkhrkhgtykdhscyckghidufhfygerbfbdcyugrf7tghrbfrghrbr8g7ebfh47t6erbfjnv8gtjngbydbvh
gr8egdfjhbvyvodouhygoyghubrgdvbiugodhgyuerbyuggiyg7rhgvafyugfgbgfigyghbgyuggu7rtbygv8rbgjbgyifghbbvyrghiuuhhs7bgr7d
hbfgrghhnsor7ogh7br is 169178299 nanoseconds
Decryption of the text(sent from server)...

Decrypted text: HAPPYCODING!XXXX
```

```
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes> java server
server is listening on port 8081
yrcgnohuroirunvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyu
rvhewpu9nhcegxnoiurnvceyrgnrfeugncvuiuygrtnrenycxegydchgfrydfhfbrhg438yrvrygt875bvkjhgncorncsdh8fhbvhfyghbvaoshduf
yyrgbv7ryvhdfbvbythbfhyhvbhjgfvdhfoiuhrubvgjdkhkfudhgjkjdtybvjdfkncvvfgvjdkfgfdnkvhguierhvhghhkbvcnhifdghhdfghdsnl
gfyghdfkyfrbkvhncmbhfbgkydfghldnhuerhvgfkdvdhfcvkgkfdgkbvskdhsfhkhrkhgtykdhscyckghidufhfygerbfbdcyugrf7tghrbfrghr
yrcgnohuroirunvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyu
rvhewpu9nhcegxnoiurnvceyrgnrfeugncvuiuygrtnrenycxegydchgfrydfhfbrhg438yrvrygt875bvkjhgncorncsdh8fhbvhfyghbvaoshduf
yyrgbv7ryvhdfbvbythbfhyhvbhjgfvdhfoiuhrubvgjdkhkfudhgjkjdtybvjdfkncvvfgvjdkfgfdnkvhguierhvhghhkbvcnhifdghhdfghdsnl
yrcgnohuroirunvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyu
yrcgnohuroirunvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyu
yrcgnohuroirunvgrhenoforyegvnhrefhpewihrfneigdmoerhjcfnfrgfoiunhvuhgorwiehpnverouhncerhoiiuopwrnhocvyoerhcnygcnbyu
rvhewpu9nhcegxnoiurnvceyrgnrfeugncvuiuygrtnrenycxegydchgfrydfhfbrhg438yrvrygt875bvkjhgncorncsdh8fhbvhfyghbvaoshduf
yyrgbv7ryvhdfbvbythbfhyhvbhjgfvdhfoiuhrubvgjdkhkfudhgjkjdtybvjdfkncvvfgvjdkfgfdnkvhguierhvhghhkbvcnhifdghhdfghdsnl
gfyghdfkyfrbkvhncmbhfbgkydfghldnhuerhvgfkdvdhfcvkgkfdgkbvskdhsfhkhrkhgtykdhscyckghidufhfygerbfbdcyugrf7tghrbfrghr
r8g7ebfh47t6erbfjnv8gtjngbydbvhgr8egdfjhbvyvodouhygoyghubrgdvbiugodhgyuerbyuggiyg7rhgvafyugfgbgfigyghbgyuggu7rtbyg
8rbgjbgyifghbbvyrghiuuhhs7bgr7dhbfgrghhnsor7ogh7brXXXX
Enter a response to send to the client:
Enter 8 characters(plaintext):
HAPPYCODING!
PS C:\Users\Bhuvanaprabha\OneDrive\Desktop\codes>
```