CRYPTOGRAPHY LABORATORY FILE
CS-511

Submitted to:
Dr. Samayveer Singh
Assistant Professor
Department of Computer Science

Submitted by:
Shashi Shekhar Azad
Roll No.: 23203029
M. Tech. CSE 1st Semester

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DR. B. R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR

Assignment- 4

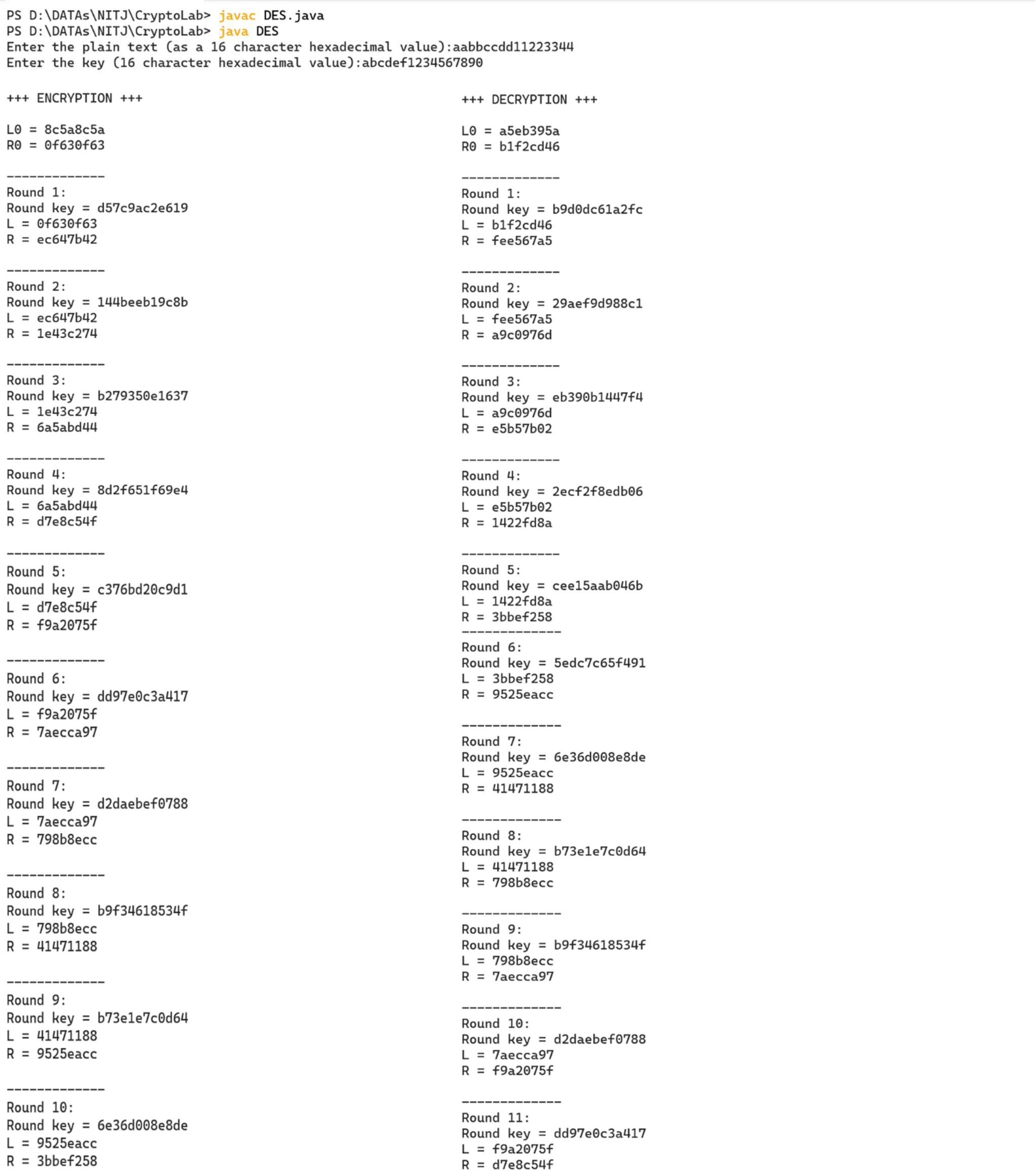
Write a program to implement the encryption and decryption process of Data Encryption Standard (DES).

Code:

```
import java.util.*;

class DES {

   private static final byte[] 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 byte[] 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
   };

   // Permuted Choice 2 table key compression table 56 bit to 48 bit
   private static final byte[] 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
};

// Array to store the number of circular left shift that are to be done on each round
private static final byte[] rotations = {
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};

// Expansion (aka P-box) table 32 bit to 48 bit of data
private static final byte[] 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
};

// S-boxes (i.e. Substitution boxes)
private static final byte[][] S = { {
    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
} };

// Permutation table
private static final byte[] 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
};

// Final permutation (aka Inverse permutation) table for data of 64 bit
private static final byte[] 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
};

// 28 bits each, used as storage in the KS (Key Structure) rounds to
// generate round keys (aka subkeys)
private static int[] C = new int[28];
private static int[] D = new int[28];
private static int[][] subkey = new int[16][48];

public static void main(String args[]) {
    System.out.println("Enter the input as a 16 character hexadecimal value:");
    String input = new Scanner(System.in).nextLine();
    int inputBits[] = new int[64];
    for(int i=0 ; i < 16 ; i++) {
        String s = Integer.toBinaryString(Integer.parseInt(input.charAt(i) + "", 16));

        while(s.length() < 4) {
            s = "0" + s;
        }
        for(int j=0 ; j < 4 ; j++) {
            inputBits[(4*i)+j] = Integer.parseInt(s.charAt(j) + "");
        }
    }

    System.out.println("Enter the key as a 16 character hexadecimal value:");
    String key = new Scanner(System.in).nextLine();
    int keyBits[] = new int[64];
    for(int i=0 ; i < 16 ; i++) {
        String s = Integer.toBinaryString(Integer.parseInt(key.charAt(i) + "", 16));
        while(s.length() < 4) {
            s = "0" + s;
        }
        for(int j=0 ; j < 4 ; j++) {
            keyBits[(4*i)+j] = Integer.parseInt(s.charAt(j) + "");
        }
    }

    System.out.println("\n+++ ENCRYPTION +++");
    int outputBits[] = permute(inputBits, keyBits, false);
    System.out.println("\n+++ DECRYPTION +++");
    permute(outputBits, keyBits, true);
}
```

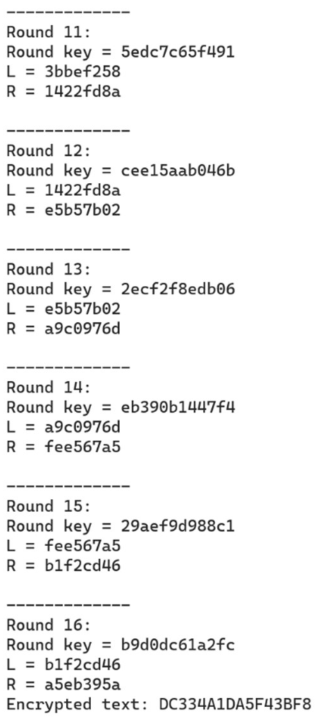
```
private static int[] permute(int[] inputBits, int[] keyBits, boolean isDecrypt) {

    int newBits[] = new int[inputBits.length];
    for(int i=0 ; i < inputBits.length ; i++) {
        newBits[i] = inputBits[IP[i]-1];
    }



    int L[] = new int[32];
    int R[] = new int[32];
    int i;

    for(i=0 ; i < 28 ; i++) {
        C[i] = keyBits[PC1[i]-1];
    }
    for( ; i < 56 ; i++) {
        D[i-28] = keyBits[PC1[i]-1];
    }
    System.arraycopy(newBits, 0, L, 0, 32);
    System.arraycopy(newBits, 32, R, 0, 32);
    System.out.print("\nL0 = ");
    displayBits(L);
    System.out.print("R0 = ");
    displayBits(R);
    for(int n=0 ; n < 16 ; n++) {
        System.out.println("\n-------------");
        System.out.println("Round " + (n+1) + ":");
        int newR[] = new int[0];
        if(isDecrypt) {
            newR = fiestel(R, subkey[15-n]);
            System.out.print("Round key = ");
            displayBits(subkey[15-n]);
        } else {
            newR = fiestel(R, KS(n, keyBits));
            System.out.print("Round key = ");
            displayBits(subkey[n]);
        }
        int newL[] = xor(L, newR);
        L = R;
        R = newL;
        System.out.print("L = ");
        displayBits(L);
        System.out.print("R = ");
        displayBits(R);
    }
    int output[] = new int[64];
    System.arraycopy(R, 0, output, 0, 32);
    System.arraycopy(L, 0, output, 32, 32);
    int finalOutput[] = new int[64];
    for(i=0 ; i < 64 ; i++) {
        finalOutput[i] = output[FP[i]-1];
    }

    String hex = new String();
    for(i=0 ; i < 16 ; i++) {
        String bin = new String();
        for(int j=0 ; j < 4 ; j++) {
            bin += finalOutput[(4*i)+j];
        }
        int decimal = Integer.parseInt(bin, 2);
        hex += Integer.toHexString(decimal);
    }
    if(isDecrypt) {
        System.out.print("Decrypted text: ");

    } else {
        System.out.print("Encrypted text: ");
    }
    System.out.println(hex.toUpperCase());
    return finalOutput;
}

private static int[] KS(int round, int[] key) {
    int C1[] = new int[28];
    int D1[] = new int[28];
    int rotationTimes = (int) rotations[round];
    C1 = leftShift(C, rotationTimes);
    D1 = leftShift(D, rotationTimes);
    int CnDn[] = new int[56];
    System.arraycopy(C1, 0, CnDn, 0, 28);
    System.arraycopy(D1, 0, CnDn, 28, 28);
    int Kn[] = new int[48];
    for(int i=0 ; i < Kn.length ; i++) {
        Kn[i] = CnDn[PC2[i]-1];
    }

    subkey[round] = Kn;
    C = C1;
    D = D1;
    return Kn;
}

private static int[] fiestel(int[] R, int[] roundKey) {
    int expandedR[] = new int[48];
    for(int i=0 ; i < 48 ; i++) {
        expandedR[i] = R[E[i]-1];
    }
    int temp[] = xor(expandedR, roundKey);
    int output[] = sBlock(temp);
    return output;
}

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

private static int[] sBlock(int[] bits) {
    int output[] = new int[32];
    for(int i=0 ; i < 8 ; i++) {
        int row[] = new int [2];
        row[0] = bits[6*i];
        row[1] = bits[(6*i)+5];
        String sRow = row[0] + "" + row[1];
        int column[] = new int[4];
        column[0] = bits[(6*i)+1];
        column[1] = bits[(6*i)+2];
        column[2] = bits[(6*i)+3];
        column[3] = bits[(6*i)+4];
        String sColumn = column[0] +""+ column[1] +""+ column[2] +""+ column[3];
        int iRow = Integer.parseInt(sRow, 2);
        int iColumn = Integer.parseInt(sColumn, 2);
        int x = S[i][(iRow*16) + iColumn];
        String s = Integer.toBinaryString(x);
        while(s.length() < 4) {
            s = "0" + s;
        }
        for(int j=0 ; j < 4 ; j++) {
            output[(i*4) + j] = Integer.parseInt(s.charAt(j) + "");
        }
    }
    int finalOutput[] = new int[32];
    for(int i=0 ; i < 32 ; i++) {
        finalOutput[i] = output[P[i]-1];
    }
    return finalOutput;
}

private static int[] leftShift(int[] bits, int n) {
    int answer[] = new int[bits.length];
    System.arraycopy(bits, 0, answer, 0, bits.length);
    for(int i=0 ; i < n ; i++) {
        int temp = answer[0];
        for(int j=0 ; j < bits.length-1 ; j++) {
            answer[j] = answer[j+1];
        }
        answer[bits.length-1] = temp;
    }
    return answer;
}

private static void displayBits(int[] bits) {

    for(int i=0 ; i < bits.length ; i+=4) {
        String output = new String();
        for(int j=0 ; j < 4 ; j++) {
            output += bits[i+j];
        }
        System.out.print(Integer.toHexString(Integer.parseInt(output, 2)));
    }
    System.out.println();
}
}
```

Output:

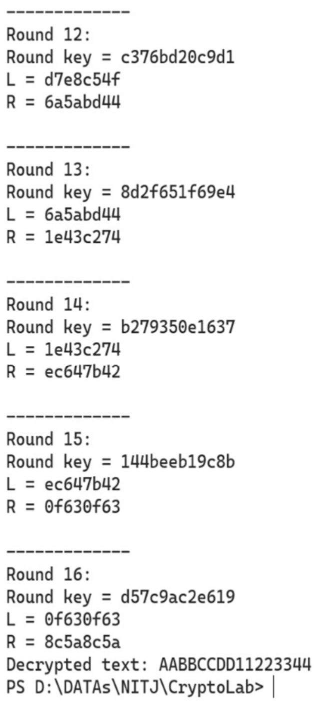


```
PS D:\DATAs\NITJ\CryptoLab> javac DES.java
PS D:\DATAs\NITJ\CryptoLab> java DES
Enter the plain text (as a 16 character hexadecimal value):aabbccdd11223344
Enter the key (16 character hexadecimal value):abcdef1234567890

+++ ENCRYPTION +++

L0 = 8c5a8c5a
R0 = 0f630f63

-------------
Round 1:
Round key = d57c9ac2e619
L = 0f630f63
R = ec647b42

-------------
Round 2:
Round key = 144beeb19c8b
L = ec647b42
R = 1e43c274

-------------
Round 3:
Round key = b279350e1637
L = 1e43c274
R = 6a5abd44

-------------
Round 4:
Round key = 8d2f651f69e4
L = 6a5abd44
R = d7e8c54f

-------------
Round 5:
Round key = c376bd20c9d1
L = d7e8c54f
R = f9a2075f

-------------
Round 6:
Round key = dd97e0c3a417
L = f9a2075f
R = 7aecca97

-------------
Round 7:
Round key = d2daebef0788
L = 7aecca97
R = 798b8ecc

-------------
Round 8:
Round key = b9f34618534f
L = 798b8ecc
R = 41471188

-------------
Round 9:
Round key = b73e1e7c0d64
L = 41471188
R = 9525eacc

-------------
Round 10:
Round key = 6e36d008e8de
L = 9525eacc
R = 3bbef258
```

```
+++ DECRYPTION +++

L0 = a5eb395a
R0 = b1f2cd46

-------------
Round 1:
Round key = b9d0dc61a2fc
L = b1f2cd46
R = fee567a5

-------------
Round 2:
Round key = 29aef9d988c1
L = fee567a5
R = a9c0976d

-------------
Round 3:
Round key = eb390b1447f4
L = a9c0976d
R = e5b57b02

-------------
Round 4:
Round key = 2ecf2f8edb06
L = e5b57b02
R = 1422fd8a

-------------
Round 5:
Round key = cee15aab046b
L = 1422fd8a
R = 3bbef258
-------------
Round 6:
Round key = 5edc7c65f491
L = 3bbef258
R = 9525eacc

-------------
Round 7:
Round key = 6e36d008e8de
L = 9525eacc
R = 41471188

-------------
Round 8:
Round key = b73e1e7c0d64
L = 41471188
R = 798b8ecc

-------------
Round 9:
Round key = b9f34618534f
L = 798b8ecc
R = 7aecca97

-------------
Round 10:
Round key = d2daebef0788
L = 7aecca97
R = f9a2075f

-------------
Round 11:
Round key = dd97e0c3a417
L = f9a2075f
R = d7e8c54f
```



```
-------------
Round 11:
Round key = 5edc7c65f491
L = 3bbef258
R = 1422fd8a

-------------
Round 12:
Round key = cee15aab046b
L = 1422fd8a
R = e5b57b02

-------------
Round 13:
Round key = 2ecf2f8edb06
L = e5b57b02
R = a9c0976d

-------------
Round 14:
Round key = eb390b1447f4
L = a9c0976d
R = fee567a5

-------------
Round 15:
Round key = 29aef9d988c1
L = fee567a5
R = b1f2cd46

-------------
Round 16:
Round key = b9d0dc61a2fc
L = b1f2cd46
R = a5eb395a
Encrypted text: DC334A1DA5F43BF8
```



```
-------------
Round 12:
Round key = c376bd20c9d1
L = d7e8c54f
R = 6a5abd44

-------------
Round 13:
Round key = 8d2f651f69e4
L = 6a5abd44
R = 1e43c274

-------------
Round 14:
Round key = b279350e1637
L = 1e43c274
R = ec647b42

-------------
Round 15:
Round key = 144beeb19c8b
L = ec647b42
R = 0f630f63

-------------
Round 16:
Round key = d57c9ac2e619
L = 0f630f63
R = 8c5a8c5a
Decrypted text: AABBCCDD11223344
PS D:\DATAs\NITJ\CryptoLab>
```