Cryptography and Network Security Lab
Ex - 5
AES Encryption

Name: *Deepansh Parmar*
Reg No: *21BCE1812*

Aim:
To implement AES encryption for 128 bit key.

Language used: *C++*

Code:

```
#include<iostream>
#include<string.h>
#include<string>
#include<vector>
#include <bitset>
#include <sstream>

using namespace std;

vector<string> words;
string initialState[4][4];
string roundKeys[11][4][4];

int hextodec(char str)
{
   if(str == '0')
      return 0;
   else if(str == '1')
      return 1;
   else if(str == '2')
      return 2;
   else if(str == '3')
      return 3;
   else if(str == '4')
      return 4;
```

```
    else if(str == '5')
        return 5;
    else if(str == '6')
        return 6;
    else if(str == '7')
        return 7;
    else if(str == '8')
        return 8;
    else if(str == '9')
        return 9;
    else if(str == 'a' || str == 'A')
        return 10;
    else if(str == 'b' || str == 'B')
        return 11;
    else if(str == 'c' || str == 'C')
        return 12;
    else if(str == 'd' || str == 'D')
        return 13;
    else if(str == 'e' || str == 'E')
        return 14;
    else if(str == 'f' || str == 'F')
        return 15;
}
string binaryToHex(const string& binaryString) {
    string hextobin = "";
    string s1 = "";
    for(int i = 0; i < binaryString.size(); i+=4)
    {
        s1 = binaryString.substr(i, 4);
        int s = stoi(s1);
        switch(s)
        {
            case 0:
            hextobin+="0";
            break;
        case 1:
            hextobin+="1";
            break;
        case 10:
            hextobin+="2";
            break;
        case 11:
            hextobin+="3";
            break;
```

```
        case 100:
            hextobin+="4";
            break;
        case 101:
            hextobin+="5";
            break;
        case 110:
            hextobin+="6";
            break;
        case 111:
            hextobin+="7";
            break;
        case 1000:
            hextobin+="8";
            break;
        case 1001:
            hextobin+="9";
            break;
        case 1010:
            hextobin+="a";
            break;
        case 1011:
            hextobin+="b";
            break;
        case 1100:
            hextobin+="c";
            break;
        case 1101:
            hextobin+="d";
            break;
        case 1110:
            hextobin+="e";
            break;
        case 1111:
            hextobin+="f";
            break;
        default:
            cout<<"Invalid"<<endl;
        }
        s1 = "";
    }
    return hextobin;
}
```

```
string HexBin(string hexdec)
{
    size_t i = (hexdec[1]=='x'||hexdec[1]=='X')?2:0;
    string hextobin="";
    while(hexdec[i])
    {
        switch(hexdec[i])
        {
            case '0':
                hextobin+="0000";
                break;
            case '1':
                hextobin+="0001";
                break;
            case '2':
                hextobin+="0010";
                break;
            case '3':
                hextobin+="0011";
                break;
            case '4':
                hextobin+="0100";
                break;
            case '5':
                hextobin+="0101";
                break;
            case '6':
                hextobin+="0110";
                break;
            case '7':
                hextobin+="0111";
                break;
            case '8':
                hextobin+="1000";
                break;
            case '9':
                hextobin+="1001";
                break;
            case 'A':case 'a':
                hextobin+="1010";
                break;
            case 'B':case 'b':
                hextobin+="1011";
                break;
```

```
            case 'C':case 'c':
               hextobin+="1100";
               break;
            case 'D':case 'd':
               hextobin+="1101";
               break;
            case 'E':case 'e':
               hextobin+="1110";
               break;
            case 'F':case 'f':
               hextobin+="1111";
               break;
            default:cout<<"\nInvalidhexadecimaldigit"<<hexdec[i];
         }
         i++;
      }
      return hextobin;
}

string Xor(string a, string b){
        string result = "";
        int size = b.size();
        for(int i = 0; i < size; i++){
                if(a[i] != b[i]){
                        result += "1";
                }
                else{
                        result += "0";
                }
        }
        return result;
}
string XOR(string a, string b)
{
   string ans = "";
   for(int i = 0; i < a.size(); ++i)
   {
      if(a[i] == b[i])
      {
         ans+="0";
      }
      else
      {
         ans+="1";
```

```
        }
    }
    return ans;
}

void lcs(string word)
{
    word = word.substr(2, 6) + word.substr(0, 2);
}

string complexFunction(string word, int ind)
{
    string sbox[16][16] = {{"63", "7c", "77", "7b", "f2", "6b", "6f", "c5", "30", "01", "67",
"2b", "fe", "d7", "ab", "76"}, {"ca", "82", "c9", "7d", "fa", "59", "47", "f0", "ad", "d4", "a2",
"af", "9c", "a4", "72", "c0"}, {"b7", "fd", "93", "26", "36", "3f", "f7", "cc", "34", "a5", "e5",
"f1", "71", "d8", "31", "15"}, {"04", "c7", "23", "c3", "18", "96", "05", "9a", "07", "12",
"80", "e2", "eb", "27", "b2", "75"}, {"09", "83", "2c", "1a", "1b", "6e", "5a", "a0", "52",
"3b", "d6", "b3", "29", "e3", "2f", "84"}, {"53", "d1", "00", "ed", "20", "fc", "b1", "5b",
"6a", "cb", "be", "39", "4a", "4c", "58", "cf"}, {"d0", "ef", "aa", "fb", "43", "4d", "33", "85",
"45", "f9", "02", "7f", "50", "3c", "9f", "a8"}, {"51", "a3", "40", "8f", "92", "9d", "38", "f5",
"bc", "b6", "da", "21", "10", "ff", "f3", "d2"}, {"cd", "0c", "13", "ec", "5f", "97", "44", "17",
"c4", "a7", "7e", "3d", "64", "5d", "19", "73"}, {"60", "81", "4f", "dc", "22", "2a", "90",
"88", "46", "ee", "b8", "14", "de", "5e", "0b", "db"}, {"e0", "32", "3a", "0a", "49", "06",
"24", "5c", "c2", "d3", "ac", "62", "91", "95", "e4", "79"}, {"e7", "c8", "37", "6d", "8d",
"d5", "4e", "a9", "6c", "56", "f4", "ea", "65", "7a", "ae", "08"}, {"ba", "78", "25", "2e",
"1c", "a6", "b4", "c6", "e8", "dd", "74", "1f", "4b", "bd", "8b", "8a"}, {"70", "3e", "b5",
"66", "48", "03", "f6", "0e", "61", "35", "57", "b9", "86", "c1", "1d", "9e"}, {"e1", "f8",
"98", "11", "69", "d9", "8e", "94", "9b", "1e", "87", "e9", "ce", "55", "28", "df"}, {"8c",
"a1", "89", "0d", "bf", "e6", "42", "68", "41", "99", "2d", "0f", "b0", "54", "bb", "16"}};

    string intermediate_key = "";


    string rkey[] = {"01000000", "02000000", "04000000", "08000000", "10000000",
"20000000", "40000000", "80000000", "1B000000", "36000000"};

    word = word.substr(2, 6) + word.substr(0, 2);

    for(int i = 0; i < 7; ++i)
    {
        intermediate_key += sbox[hextodec(word[i+0])][hextodec(word[i+1])];
        ++i;
    }
```

```
    string ret = XOR(HexBin(intermediate_key), HexBin(rkey[ind]));

    return ret;

}

void keyGenerator(string key)
{

    words.push_back(key.substr(0, 8));
    words.push_back(key.substr(8, 8));
    words.push_back(key.substr(16, 8));
    words.push_back(key.substr(24, 8));

    for(int i = 4; i < 44; ++i)
    {
        if(i%4 == 0)
        {
            words.push_back(binaryToHex((XOR(complexFunction(words[i-1],
(i/4)-1),HexBin(words[i-4])))));
        }
        else
        {
            words.push_back(binaryToHex(XOR(HexBin(words[i-1]),
HexBin(words[i-4]))));
        }
    }

    for(int i = 0; i < 44; ++i)
    {
        cout<<i+1<<" "<<words[i]<<endl;
    }
}

void substituteBytes(string mat[4][4])
{
    string sbox[16][16] = {{"63", "7c", "77", "7b", "f2", "6b", "6f", "c5", "30", "01", "67",
"2b", "fe", "d7", "ab", "76"}, {"ca", "82", "c9", "7d", "fa", "59", "47", "f0",
"ad", "d4", "a2", "af", "9c", "a4", "72", "c0"}, {"b7", "fd", "93", "26", "36", "3f", "f7", "cc",
"34", "a5", "e5", "f1", "71", "d8", "31", "15"}, {                    "04", "c7", "23", "c3",
"18", "96", "05", "9a", "07", "12", "80", "e2", "eb", "27", "b2", "75"}, {"09", "83", "2c",
"1a", "1b", "6e", "5a", "a0",                    "52", "3b", "d6", "b3", "29", "e3", "2f",
"84"}, {"53", "d1", "00", "ed", "20", "fc", "b1", "5b", "6a", "cb", "be", "39", "4a", "4c",
"58", "cf"}, {                    "d0", "ef", "aa", "fb", "43", "4d", "33", "85", "45", "f9",
```

```
"02", "7f", "50", "3c", "9f", "a8"}, {"51", "a3", "40", "8f", "92", "9d", "38", "f5",
"bc", "b6", "da", "21", "10", "ff", "f3", "d2"}, {"cd", "0c", "13", "ec", "5f", "97", "44", "17",
"c4", "a7", "7e", "3d", "64", "5d", "19", "73"}, {                    "60", "81", "4f", "dc",
"22", "2a", "90", "88", "46", "ee", "b8", "14", "de", "5e", "0b", "db"}, {"e0", "32", "3a",
"0a", "49", "06", "24", "5c",                    "c2", "d3", "ac", "62", "91", "95", "e4",
"79"}, {"e7", "c8", "37", "6d", "8d", "d5", "4e", "a9", "6c", "56", "f4", "ea", "65", "7a",
"ae", "08"}, {                    "ba", "78", "25", "2e", "1c", "a6", "b4", "c6", "e8", "dd",
"74", "1f", "4b", "bd", "8b", "8a"}, {"70", "3e", "b5", "66", "48", "03", "f6", "0e",
"61", "35", "57", "b9", "86", "c1", "1d", "9e"}, {"e1", "f8", "98", "11", "69", "d9", "8e",
"94", "9b", "1e", "87", "e9", "ce", "55", "28", "df"}, {                    "8c", "a1", "89",
"0d", "bf", "e6", "42", "68", "41", "99", "2d", "0f", "b0", "54", "bb", "16"}
                    };
   for(int i = 0; i < 4; ++i)
   {
      for(int j = 0; j < 4; ++j)
      {
         mat[i][j] = sbox[hextodec(mat[i][j][0])][hextodec(mat[i][j][1])];
      }
   }

   cout<<"after substituting bytes"<<endl;
   for(int i = 0; i < 4; ++i)
   {
      for(int j = 0; j < 4; ++j)
      {
         cout<<mat[i][j]<<" ";
      }
      cout<<endl;
   }
}

void lcs2(string& word, int index)
{
   if(index == 1)
      word = word.substr(2, 6) + word.substr(0, 2);
   else if(index == 2)
      word = word.substr(4, 4) + word.substr(0, 4);
   else if(index == 3)
      word = word.substr(6, 2) + word.substr(0, 6);
}

void shiftRows(string mat[4][4])
{
   string temp = "";
```

```
    for(int i = 1; i < 4; ++i)
    {
        temp = mat[i][0]+mat[i][1]+mat[i][2]+mat[i][3];
        lcs2(temp, i);
        mat[i][0] = temp.substr(0, 2);
        mat[i][1] = temp.substr(2, 2);
        mat[i][2] = temp.substr(4, 2);
        mat[i][3] = temp.substr(6, 2);
    }
    cout<<"after shifting rows:<"<<endl;
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
        {
            cout<<mat[i][j]<<" ";
        }
        cout<<endl;
    }
}

string multiply(string num1, string num2) {
    int flag = 0;

    if (num1 == "01") {
        return num2;
    }
    else if (num1 == "02")
    {
        string hexnum = HexBin(num2);
        char ch = hexnum[0];

        hexnum = hexnum.substr(1, 7) + "0";

        if (ch == '1') {
            hexnum = XOR(hexnum, HexBin("1b"));
        }
        hexnum = binaryToHex(hexnum);
        if(hexnum.size() == 1)
            hexnum = "0" + hexnum;
        return hexnum;
    }
    else {
        string hexnum = HexBin(num2);
        char ch = hexnum[0];
```

```
        hexnum = hexnum.substr(1, 7) + "0";

        if (ch == '1') {
            hexnum = XOR(hexnum, HexBin("1b"));
        }
        if(hexnum.size() == 1)
            hexnum = "0" + hexnum;

        string hexnum2 = binaryToHex(XOR(hexnum, HexBin(num2)));
        if(hexnum2.size() == 1)
            hexnum2 = "0" + hexnum2;
        return hexnum2;
    }
}
void mixColumns(string mat[4][4])
{
    string MColumn[4][4] = {
    {"02", "03", "01", "01"},
    {"01", "02", "03", "01"},
    {"01", "01", "02", "03"},
    {"03", "01", "01", "02"}
    };

    string result[4][4];

    string temp = "";

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            temp = "00";


            for (int k = 0; k < 4; k++) {
                result[i][j] = binaryToHex(XOR(HexBin(temp),
HexBin(multiply(MColumn[i][k], mat[k][j]))));
                temp = result[i][j];
            }
        }
    }
    cout<<"after mixing columns:"<<endl;
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
```

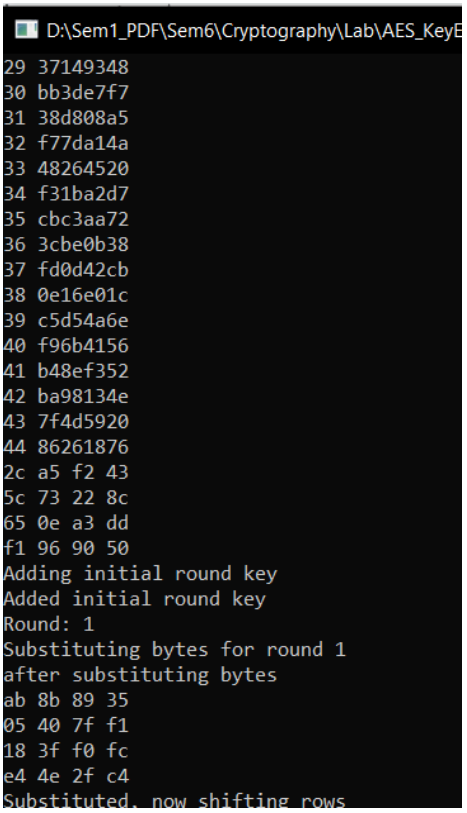
```
        {
            mat[i][j] = result[i][j];
            cout<<mat[i][j]<<" ";
        }
        cout<<endl;
    }
}

void addRoundKey(string mat[4][4], string key[4][4])
{
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
        {
            mat[i][j] = binaryToHex(XOR(HexBin(mat[i][j]), HexBin(key[i][j])));
        }
    }
}

void encryptAES(string pt)
{
    int c = 0, c1 = 0, c2 = 0;
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
        {
            initialState[j][i] = pt.substr(c, 2);
            c = c + 2;
        }
    }
    for(int k = 0; k < 11; ++k)
    {
        for(int i = 0; i < 4; ++i)
        {
            for(int j = 0; j < 4; ++j)
            {
                roundKeys[k][j][i] = words[c1].substr(c2, 2);
                c2+=2;
            }
            ++c1;
            c2 = 0;
        }
    }
```

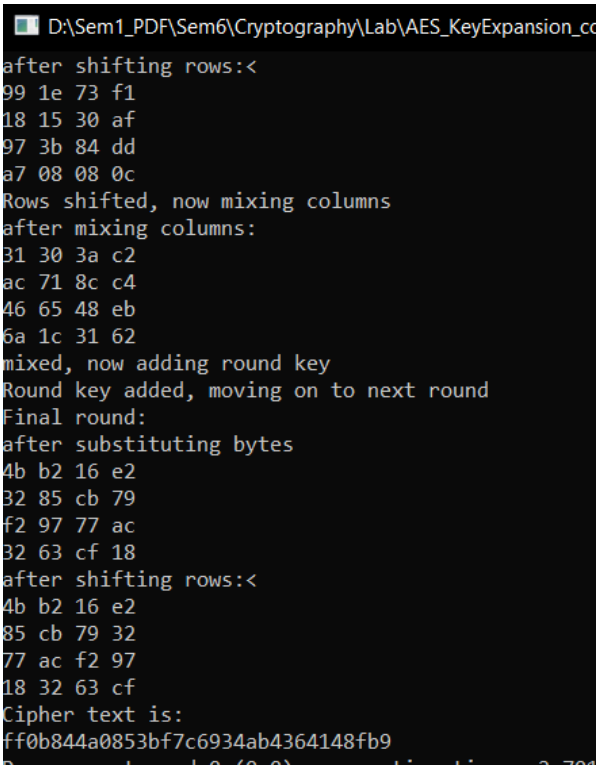
```
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
        {
            cout<<roundKeys[4][i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<"Adding initial round key"<<endl;

    addRoundKey(initialState, roundKeys[0]);

    cout<<"Added initial round key"<<endl;

    for(int i = 1; i < 10; ++i)
    {
        cout<<"Round: "<<i<<endl;
        cout<<"Substituting bytes for round "<<i<<endl;
        substituteBytes(initialState);
        cout<<"Substituted, now shifting rows"<<endl;
        shiftRows(initialState);
        cout<<"Rows shifted, now mixing columns"<<endl;
        mixColumns(initialState);
        cout<<"mixed, now adding round key"<<endl;
        addRoundKey(initialState, roundKeys[i]);
        cout<<"Round key added, moving on to next round"<<endl;
    }
    cout<<"Final round: "<<endl;

    substituteBytes(initialState);
    shiftRows(initialState);
    addRoundKey(initialState, roundKeys[10]);

    cout<<"Cipher text is: "<<endl;
    for(int i = 0; i < 4; ++i)
    {
        for(int j = 0; j < 4; ++j)
        {
            cout<<initialState[j][i];
        }
    }
}
int main()
{
```

```
    //string key = "00112233445566778899aabbccddeeff";
    string key = "0f1571c947d9e8590cb7add6af7f6798";
    string plainText = "0123456789abcdeffedcba9876543210";
    cout<<"Key is: "<<key<<endl;

    keyGenerator(key);

    encryptAES(plainText);

    return 0;
}
```

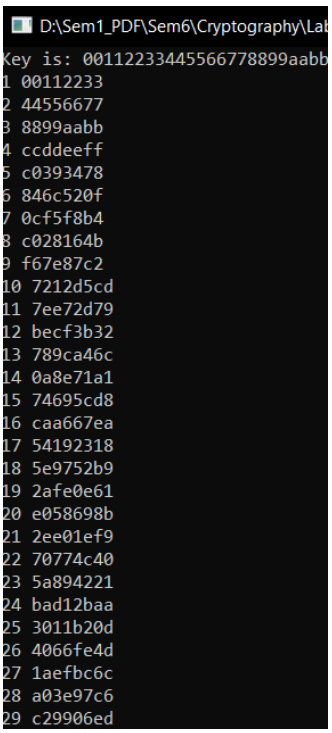
Output:

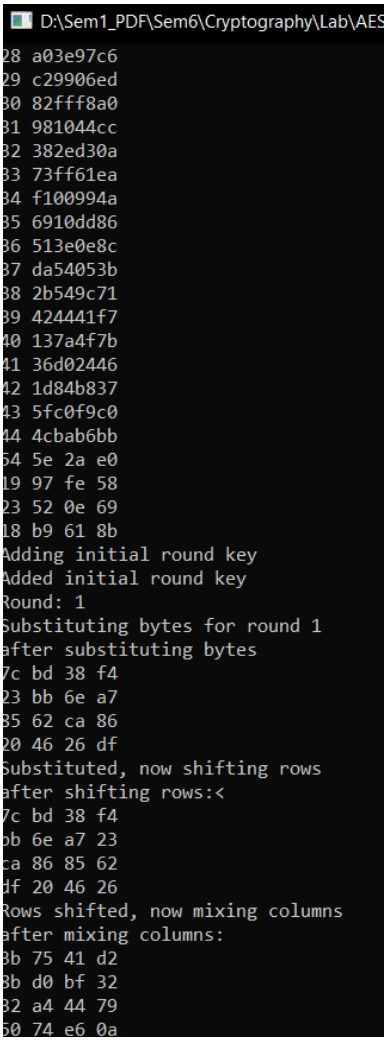
```
Key is: 0f1571c947d9e8590cb7add6af7f6798
1 0f1571c9
2 47d9e859
3 0cb7add6
4 af7f6798
5 dc9037b0
6 9b49dfe9
7 97fe723f
8 388115a7
9 d2c96bb7
10 4980b45e
11 de7ec661
12 e6ffd3c6
13 c0afdf39
14 892f6b67
15 5751ad06
16 b1ae7ec0
17 2c5c65f1
18 a5730e96
19 f222a390
20 438cdd50
21 589d36eb
22 fdee387d
23 0fcc9bed
24 4c4046bd
25 71c74cc2
26 8c2974bf
27 83e5ef52
28 cfa5a9ef
29 37149348
```

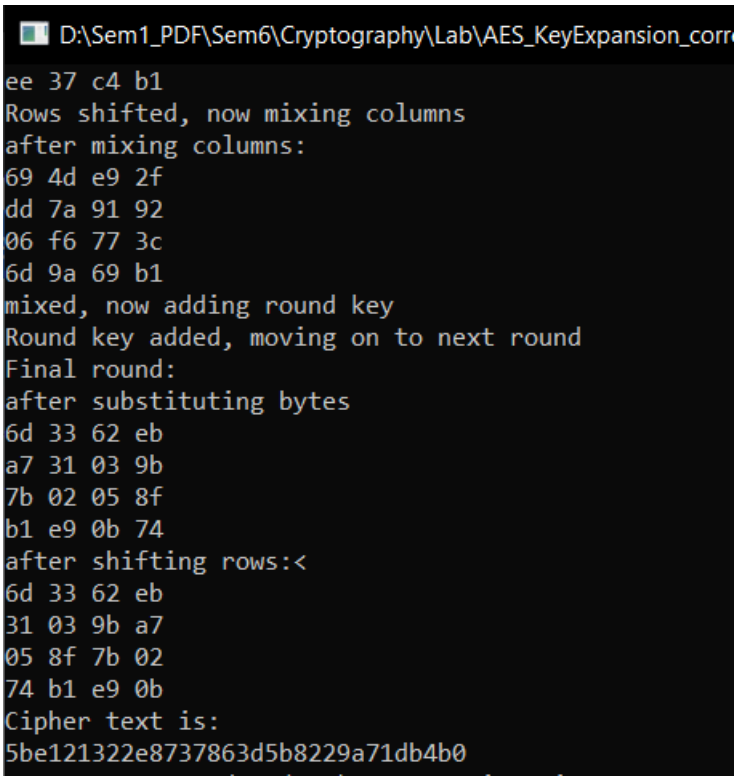




Above output is for the plaintext: 0123456789abcdeffedcba9876543210
And key: 0f1571c947d9e8590cb7add6af7f6798


D:\Sem1_PDF\Sem6\Cryptography\Lat
Key is: 00112233445566778899aabb
1 00112233
2 44556677
3 8899aabb
4 ccddeeff
5 c0393478
6 846c520f
7 0cf5f8b4
8 c028164b
9 f67e87c2
10 7212d5cd
11 7ee72d79
12 becf3b32
13 789ca46c
14 0a8e71a1
15 74695cd8
16 caa667ea
17 54192318
18 5e9752b9
19 2afe0e61
20 e058698b
21 2ee01ef9
22 70774c40
23 5a894221
24 bad12baa
25 3011b20d
26 4066fe4d
27 1aefbc6c
28 a03e97c6
29 c29906ed


D:\Sem1_PDF\Sem6\Cryptography\Lab\AES
28 a03e97c6
29 c29906ed
30 82fff8a0
31 981044cc
32 382ed30a
33 73ff61ea
34 f100994a
35 6910dd86
36 513e0e8c
37 da54053b
38 2b549c71
39 424441f7
40 137a4f7b
41 36d02446
42 1d84b837
43 5fc0f9c0
44 4cbab6bb
54 5e 2a e0
19 97 fe 58
23 52 0e 69
18 b9 61 8b
Adding initial round key
Added initial round key
Round: 1
Substituting bytes for round 1
after substituting bytes
7c bd 38 f4
23 bb 6e a7
35 62 ca 86
20 46 26 df
Substituted, now shifting rows
after shifting rows:<
7c bd 38 f4
bb 6e a7 23
ca 86 85 62
df 20 46 26
Rows shifted, now mixing columns
after mixing columns:
3b 75 41 d2
3b d0 bf 32
32 a4 44 79
50 74 e6 0a



Above output is for plaintext: 0123456789abcdeffedcba9876543210
And key: 00112233445566778899aabbccddeeff

Result:

AES was successfully implemented and output verified using C++ program.