#include <iostream>

#include <vector>

#include <string>

#include <map>

#include <sstream>

#include <cstring>

#include <bitset>

#include <random>

#include <iterator>

using namespace std;

string random\_string(size\_t length)

{

    const string CHARACTERS = "0123456789abcdefghijklmnopqrstuvwxyz";

    random\_device random\_device;

    mt19937 generator(random\_device());

    uniform\_int\_distribution<> distribution(0, CHARACTERS.size() - 1);

    string random\_string;

    for (size\_t i = 0; i < length; ++i)

    {

        random\_string += CHARACTERS[distribution(generator)];

    }

    return random\_string;

}

struct blockStruct

{

    // block declared inside structure

    string matrix[4][4];

};

struct blockStruct shiftRows(string mat[4][4], int shiftCount, int row)

{

    struct blockStruct BS;

    for (int i = 0; i < shiftCount; i++)

    {

        string lastColumn = mat[row][3];

        for (int j = 3; j > 0; j--)

        {

            mat[row][j] = mat[row][j - 1];

        }

        mat[row][0] = lastColumn;

    }

    for (int i = 0; i <= 3; i++)

        for (int j = 0; j <= 3; j++)

            BS.matrix[i][j] = mat[i][j];

    return BS;

}

struct blockStruct mixColumns(string mat[4][4])

{

    struct blockStruct BS;

    string mixed\_matrix[4][4];

    int pre\_defined\_matrix[4][4] = {

        {2, 3, 1, 1},

        {1, 2, 3, 1},

        {1, 1, 2, 3},

        {3, 1, 1, 2}};

    for (int z = 0; z < 4; z++)

    {

        for (int i = 0; i < 4; i++)

        {

            bitset<10> collected\_results[4]; // to be used after getting all 4 results then XORed to get main result of one cell in final matrix

            bitset<10> xor\_each\_cell;        // this is where Xor results will be processed

            for (int j = 0; j < 4; j++)

            {

                stringstream stream1;

                stream1 << hex << mat[j][z];

                unsigned n;

                stream1 >> n;

                bitset<10> cell\_value(n);

                bitset<10> result;

                if (pre\_defined\_matrix[i][j] == 2)

                {

                    result[0] = 0;

                    for (int t = 0; t < 10; t++)

                    {

                        result[t + 1] = cell\_value[t];

                    }

                }

                else if (pre\_defined\_matrix[i][j] == 3)

                {

                    result[0] = 0;

                    for (int t = 0; t < 10; t++)

                    {

                        result[t + 1] = cell\_value[t];

                    }

                    // XOR each bit for multiplication of 3 from the predefined matrix

                    for (int t = 0; t < 10; t++)

                    {

                        result[t] = (cell\_value[t] ^ result[t]);

                    }

                }

                else

                    result = cell\_value;

                // checking the resultant number of bits, if it is greater than 8 bit, we will need to apply the irreducible polynomial theorem

                int highest\_bit\_power = 0;

                for (int t = 0; t < 10; t++)

                {

                    if (result[t] == 1)

                        highest\_bit\_power = t;

                }

                string irr\_pol1 = "100011011";

                string irr\_pol2 = "1000110110";

                bitset<10> x8\_x4\_x3\_x\_1(irr\_pol1);

                bitset<10> x9\_x5\_x4\_x2\_x(irr\_pol2);

                if (highest\_bit\_power == 8)

                {

                    for (int t = 0; t < 10; t++)

                    {

                        result[t] = (x8\_x4\_x3\_x\_1[t] ^ result[t]);

                    }

                }

                else if (highest\_bit\_power == 9)

                {

                    for (int t = 0; t < 10; t++)

                    {

                        result[t] = (x9\_x5\_x4\_x2\_x[t] ^ result[t]);

                    }

                }

                collected\_results[j] = result;

            }

            xor\_each\_cell = (collected\_results[0] ^ collected\_results[1] ^ collected\_results[2] ^ collected\_results[3]);

            stringstream stream2;

            stream2 << hex << uppercase << xor\_each\_cell.to\_ulong();

            mixed\_matrix[i][z] = stream2.str();

        }

    }

    for (int i = 0; i <= 3; i++)

    {

        for (int j = 0; j <= 3; j++)

        {

            BS.matrix[i][j] = mixed\_matrix[i][j];

        }

    }

    return BS;

}

int main()

{

    map<char, map<char, string>> SBox;

    SBox['0']['0'] = "52";

    SBox['0']['1'] = "09";

    SBox['0']['2'] = "6a";

    SBox['0']['3'] = "d5";

    SBox['0']['4'] = "30";

    SBox['0']['5'] = "36";

    SBox['0']['6'] = "a5";

    SBox['0']['7'] = "38";

    SBox['0']['8'] = "bf";

    SBox['0']['9'] = "40";

    SBox['0']['a'] = "a3";

    SBox['0']['b'] = "9e";

    SBox['0']['c'] = "81";

    SBox['0']['d'] = "f3";

    SBox['0']['e'] = "d7";

    SBox['0']['f'] = "fb";

    SBox['1']['0'] = "7c";

    SBox['1']['1'] = "e3";

    SBox['1']['2'] = "39";

    SBox['1']['3'] = "82";

    SBox['1']['4'] = "9b";

    SBox['1']['5'] = "2f";

    SBox['1']['6'] = "ff";

    SBox['1']['7'] = "87";

    SBox['1']['8'] = "34";

    SBox['1']['9'] = "8e";

    SBox['1']['a'] = "43";

    SBox['1']['b'] = "44";

    SBox['1']['c'] = "c4";

    SBox['1']['d'] = "de";

    SBox['1']['e'] = "e9";

    SBox['1']['f'] = "cb";

    SBox['2']['0'] = "54";

    SBox['2']['1'] = "7b";

    SBox['2']['2'] = "94";

    SBox['2']['3'] = "32";

    SBox['2']['4'] = "a6";

    SBox['2']['5'] = "c2";

    SBox['2']['6'] = "23";

    SBox['2']['7'] = "3d";

    SBox['2']['8'] = "ee";

    SBox['2']['9'] = "4c";

    SBox['2']['a'] = "95";

    SBox['2']['b'] = "0b";

    SBox['2']['c'] = "42";

    SBox['2']['d'] = "fa";

    SBox['2']['e'] = "c3";

    SBox['2']['f'] = "4e";

    SBox['3']['0'] = "08";

    SBox['3']['1'] = "2e";

    SBox['3']['2'] = "a1";

    SBox['3']['3'] = "66";

    SBox['3']['4'] = "28";

    SBox['3']['5'] = "d9";

    SBox['3']['6'] = "24";

    SBox['3']['7'] = "b2";

    SBox['3']['8'] = "76";

    SBox['3']['9'] = "5b";

    SBox['3']['a'] = "a2";

    SBox['3']['b'] = "49";

    SBox['3']['c'] = "6d";

    SBox['3']['d'] = "8b";

    SBox['3']['e'] = "d1";

    SBox['3']['f'] = "25";

    SBox['4']['0'] = "72";

    SBox['4']['1'] = "f8";

    SBox['4']['2'] = "f6";

    SBox['4']['3'] = "64";

    SBox['4']['4'] = "86";

    SBox['4']['5'] = "68";

    SBox['4']['6'] = "98";

    SBox['4']['7'] = "16";

    SBox['4']['8'] = "d4";

    SBox['4']['9'] = "a4";

    SBox['4']['a'] = "5c";

    SBox['4']['b'] = "cc";

    SBox['4']['c'] = "5d";

    SBox['4']['d'] = "65";

    SBox['4']['e'] = "b6";

    SBox['4']['f'] = "92";

    SBox['5']['0'] = "6c";

    SBox['5']['1'] = "70";

    SBox['5']['2'] = "48";

    SBox['5']['3'] = "50";

    SBox['5']['4'] = "fd";

    SBox['5']['5'] = "ed";

    SBox['5']['6'] = "b9";

    SBox['5']['7'] = "da";

    SBox['5']['8'] = "5e";

    SBox['5']['9'] = "15";

    SBox['5']['a'] = "46";

    SBox['5']['b'] = "57";

    SBox['5']['c'] = "a7";

    SBox['5']['d'] = "8d";

    SBox['5']['e'] = "9d";

    SBox['5']['f'] = "84";

    SBox['6']['0'] = "90";

    SBox['6']['1'] = "d8";

    SBox['6']['2'] = "ab";

    SBox['6']['3'] = "00";

    SBox['6']['4'] = "8c";

    SBox['6']['5'] = "bc";

    SBox['6']['6'] = "d3";

    SBox['6']['7'] = "0a";

    SBox['6']['8'] = "f7";

    SBox['6']['9'] = "e4";

    SBox['6']['a'] = "58";

    SBox['6']['b'] = "05";

    SBox['6']['c'] = "b8";

    SBox['6']['d'] = "b3";

    SBox['6']['e'] = "45";

    SBox['6']['f'] = "06";

    SBox['7']['0'] = "d0";

    SBox['7']['1'] = "2c";

    SBox['7']['2'] = "1e";

    SBox['7']['3'] = "8f";

    SBox['7']['4'] = "ca";

    SBox['7']['5'] = "3f";

    SBox['7']['6'] = "0f";

    SBox['7']['7'] = "02";

    SBox['7']['8'] = "c1";

    SBox['7']['9'] = "af";

    SBox['7']['a'] = "bd";

    SBox['7']['b'] = "03";

    SBox['7']['c'] = "01";

    SBox['7']['d'] = "13";

    SBox['7']['e'] = "8a";

    SBox['7']['f'] = "6b";

    SBox['8']['0'] = "3a";

    SBox['8']['1'] = "91";

    SBox['8']['2'] = "11";

    SBox['8']['3'] = "41";

    SBox['8']['4'] = "4f";

    SBox['8']['5'] = "67";

    SBox['8']['6'] = "dc";

    SBox['8']['7'] = "ea";

    SBox['8']['8'] = "97";

    SBox['8']['9'] = "f2";

    SBox['8']['a'] = "cf";

    SBox['8']['b'] = "ce";

    SBox['8']['c'] = "f0";

    SBox['8']['d'] = "b4";

    SBox['8']['e'] = "e6";

    SBox['8']['f'] = "73";

    SBox['9']['0'] = "96";

    SBox['9']['1'] = "ac";

    SBox['9']['2'] = "74";

    SBox['9']['3'] = "22";

    SBox['9']['4'] = "e7";

    SBox['9']['5'] = "ad";

    SBox['9']['6'] = "35";

    SBox['9']['7'] = "85";

    SBox['9']['8'] = "e2";

    SBox['9']['9'] = "f9";

    SBox['9']['a'] = "37";

    SBox['9']['b'] = "e8";

    SBox['9']['c'] = "1c";

    SBox['9']['d'] = "75";

    SBox['9']['e'] = "df";

    SBox['9']['f'] = "6e";

    SBox['a']['0'] = "47";

    SBox['a']['1'] = "f1";

    SBox['a']['2'] = "1a";

    SBox['a']['3'] = "71";

    SBox['a']['4'] = "1d";

    SBox['a']['5'] = "29";

    SBox['a']['6'] = "c5";

    SBox['a']['7'] = "89";

    SBox['a']['8'] = "6f";

    SBox['a']['9'] = "b7";

    SBox['a']['a'] = "62";

    SBox['a']['b'] = "0e";

    SBox['a']['c'] = "aa";

    SBox['a']['d'] = "18";

    SBox['a']['e'] = "be";

    SBox['a']['f'] = "1b";

    SBox['b']['0'] = "fc";

    SBox['b']['1'] = "56";

    SBox['b']['2'] = "3e";

    SBox['b']['3'] = "4b";

    SBox['b']['4'] = "c6";

    SBox['b']['5'] = "d2";

    SBox['b']['6'] = "79";

    SBox['b']['7'] = "20";

    SBox['b']['8'] = "9a";

    SBox['b']['9'] = "db";

    SBox['b']['a'] = "c0";

    SBox['b']['b'] = "fe";

    SBox['b']['c'] = "78";

    SBox['b']['d'] = "cd";

    SBox['b']['e'] = "5a";

    SBox['b']['f'] = "f4";

    SBox['c']['0'] = "1f";

    SBox['c']['1'] = "dd";

    SBox['c']['2'] = "a8";

    SBox['c']['3'] = "33";

    SBox['c']['4'] = "88";

    SBox['c']['5'] = "07";

    SBox['c']['6'] = "c7";

    SBox['c']['7'] = "31";

    SBox['c']['8'] = "b1";

    SBox['c']['9'] = "12";

    SBox['c']['a'] = "10";

    SBox['c']['b'] = "59";

    SBox['c']['c'] = "27";

    SBox['c']['d'] = "80";

    SBox['c']['e'] = "ec";

    SBox['c']['f'] = "5f";

    SBox['d']['0'] = "60";

    SBox['d']['1'] = "51";

    SBox['d']['2'] = "7f";

    SBox['d']['3'] = "a9";

    SBox['d']['4'] = "19";

    SBox['d']['5'] = "b5";

    SBox['d']['6'] = "4a";

    SBox['d']['7'] = "0d";

    SBox['d']['8'] = "2d";

    SBox['d']['9'] = "e5";

    SBox['d']['a'] = "7a";

    SBox['d']['b'] = "9f";

    SBox['d']['c'] = "93";

    SBox['d']['d'] = "c9";

    SBox['d']['e'] = "9c";

    SBox['d']['f'] = "ef";

    SBox['e']['0'] = "a0";

    SBox['e']['1'] = "e0";

    SBox['e']['2'] = "3b";

    SBox['e']['3'] = "4d";

    SBox['e']['4'] = "ae";

    SBox['e']['5'] = "2a";

    SBox['e']['6'] = "f5";

    SBox['e']['7'] = "b0";

    SBox['e']['8'] = "c8";

    SBox['e']['9'] = "eb";

    SBox['e']['a'] = "bb";

    SBox['e']['b'] = "3c";

    SBox['e']['c'] = "83";

    SBox['e']['d'] = "53";

    SBox['e']['e'] = "99";

    SBox['e']['f'] = "61";

    SBox['f']['0'] = "17";

    SBox['f']['1'] = "2b";

    SBox['f']['2'] = "04";

    SBox['f']['3'] = "7e";

    SBox['f']['4'] = "ba";

    SBox['f']['5'] = "77";

    SBox['f']['6'] = "d6";

    SBox['f']['7'] = "26";

    SBox['f']['8'] = "e1";

    SBox['f']['9'] = "69";

    SBox['f']['a'] = "14";

    SBox['f']['b'] = "63";

    SBox['f']['c'] = "55";

    SBox['f']['d'] = "21";

    SBox['f']['e'] = "0c";

    SBox['f']['f'] = "7d";

    string msg = "this is text xyz"; // The message

    cout << "Please, specify the encryption type:"

         << "\n"

         << "1. AES 128 bit"

         << "\n"

         << "2. AES 256 bit"

         << "\n"

         << "3. AES 512 bit"

         << endl;

    int bytes;

    int rounds\_count;

    int key\_chage\_round;

    int key\_change\_round\_check = 0;

    bool check = false;

    int col\_count;

    string Rcon\_table[10] = {"00000001",

                             "00000010",

                             "00000100",

                             "00001000",

                             "00010000",

                             "00100000",

                             "01000000",

                             "10000000",

                             "00011011",

                             "00110110"};

    int Rcon\_index = 0;

    while (!check)

    {

        cout << "Please choose the selection number from 1,2 or 3." << endl;

        string input\_AES\_Type;

        cin >> input\_AES\_Type;

        if (input\_AES\_Type == "1")

        {

            bytes = 16;

            rounds\_count = 10;

            key\_chage\_round = 1;

            break;

        }

        else if (input\_AES\_Type == "2")

        {

            bytes = 32;

            rounds\_count = 14;

            key\_chage\_round = 2;

            break;

        }

        else if (input\_AES\_Type == "3")

        {

            bytes = 64;

            rounds\_count = 22;

            key\_chage\_round = 4;

            break;

        }

        else

        {

            cout << "Wrong entry." << endl;

        }

    }

    string initial\_key = random\_string(bytes);

    string add\_round\_key;

    string current\_key;

    for (int rnd = 0; rnd < rounds\_count; rnd++)

    {

        if (rnd == 0)

        {

            add\_round\_key = initial\_key;

        }

        else if (key\_change\_round\_check == key\_chage\_round)

        {

            col\_count = key\_chage\_round \* 4;

            // string round\_key[4][col\_count];

            map<int, map<int, string>> round\_key;

            map<int, map<int, string>> next\_key;

            // string next\_key[4][col\_count];

            int key\_row = 0;

            int key\_col = 0;

            for (char character : add\_round\_key)

            {

                int asciiValue = (int)character; // Get the ASCII value for each character

                stringstream key\_stream;

                key\_stream << hex << asciiValue; // convert to hex

                string res(key\_stream.str());

                round\_key[key\_row][key\_col] = res;

                if (key\_col >= 3)

                {

                    key\_col = 0;

                    key\_row += 1;

                }

                else

                    key\_col += 1;

                /\*for (int i = 0; i <= 3; i++)

                {

                    for (int j = 0; j <= col\_count - 1; j++)

                        cout << round\_key[i][j] << "\t ";

                    cout << "\n";

                }\*/

            }

            next\_key[0][0] = round\_key[1][col\_count];

            next\_key[1][0] = round\_key[2][col\_count];

            next\_key[2][0] = round\_key[3][col\_count];

            next\_key[3][0] = round\_key[0][col\_count];

            for (int i = 0; i <= 3; i++)

            {

                char SBOX\_Row = next\_key[i][0][0];

                char SBOX\_Col = next\_key[i][0][1];

                string conv = SBox[SBOX\_Row][SBOX\_Col];

                next\_key[i][0] = conv;

                // cout << next\_key[i][j] << "\t ";

            }

            for (int i = 0; i <= 3; i++)

            {

                stringstream stream\_keys\_comp1;

                stringstream stream\_keys\_comp2;

                stream\_keys\_comp1 << hex << round\_key[i][0];

                stream\_keys\_comp2 << hex << next\_key[i][0];

                unsigned n;

                unsigned n2;

                stream\_keys\_comp1 >> n;

                stream\_keys\_comp2 >> n2;

                bitset<8> coloumn1(n);

                bitset<8> coloumn2(n2);

                bitset<8> coloumn3(Rcon\_table[Rcon\_index]);

                Rcon\_index++;

                bitset<8> xor\_columns = (coloumn1 ^ coloumn2 ^ coloumn3);

                stringstream stream\_value;

                stream\_value << hex << uppercase << xor\_columns.to\_ulong();

                next\_key[i][0] = stream\_value.str();

            }

            for (int i = 1; i <= col\_count - 1; i++)

            {

                for (int j = 0; j <= 3; j++)

                {

                    stringstream stream\_keys\_comp1;

                    stringstream stream\_keys\_comp2;

                    stream\_keys\_comp1 << hex << round\_key[j][i];

                    stream\_keys\_comp2 << hex << next\_key[j][i - 1];

                    unsigned n;

                    unsigned n2;

                    stream\_keys\_comp1 >> n;

                    stream\_keys\_comp2 >> n2;

                    bitset<8> coloumn1(n);

                    bitset<8> coloumn2(n2);

                    bitset<8> xor\_columns = (coloumn1 ^ coloumn2);

                    stringstream stream\_value;

                    stream\_value << hex << uppercase << xor\_columns.to\_ulong();

                    next\_key[j][i] = stream\_value.str();

                }

            }

            string bit\_to\_string = "";

            for (int i = 0; i <= 3; i++)

            {

                for (int j = 0; j <= col\_count - 1; j++)

                {

                    stringstream ks;

                    ks << hex << next\_key[i][j];

                    unsigned n;

                    ks >> n;

                    bitset<8> char\_count(n);

                    stringstream stream\_value\_ascii;

                    stream\_value\_ascii << char(char\_count.to\_ulong());

                    bit\_to\_string += stream\_value\_ascii.str();

                }

            }

            add\_round\_key = bit\_to\_string;

            // cout << "this is the new add round key string" << add\_round\_key << endl;

            key\_change\_round\_check = 0;

        }

        cout << endl

             << "---------------------" << endl

             << "The current block"

             << endl;

        current\_key = initial\_key.substr((key\_change\_round\_check \* 15) + key\_change\_round\_check, 15);

        // cout << "this is the current key string" << add\_round\_key << endl;

        key\_change\_round\_check++;

        string block[4][4];

        int row = 0;

        int col = 0;

        for (char character : msg)

        {

            int asciiValue = (int)character; // Get the ASCII value for each character

            stringstream ss;

            ss << hex << asciiValue; // convert to hex

            string res(ss.str());

            block[row][col] = res;

            if (col >= 3)

            {

                col = 0;

                row += 1;

            }

            else

                col += 1;

        }

        int i, j;

        for (i = 0; i <= 3; i++)

        {

            for (j = 0; j <= 3; j++)

                cout << block[i][j] << "\t ";

            cout << "\n";

        }

        cout << endl

             << "---------------------" << endl

             << "Sub byte block"

             << endl;

        for (i = 0; i <= 3; i++)

        {

            for (j = 0; j <= 3; j++)

            {

                char SBOX\_Row = block[i][j][0];

                char SBOX\_Col = block[i][j][1];

                string conv = SBox[SBOX\_Row][SBOX\_Col];

                block[i][j] = conv;

                cout << block[i][j] << "\t ";

            }

            cout << "\n";

        }

        cout << endl

             << "---------------------" << endl

             << "Shift Rows block"

             << endl;

        cout << block[0][0] << "\t " << block[0][1] << "\t " << block[0][2] << "\t " << block[0][3] << "\n";

        struct blockStruct bl;

        for (i = 1; i <= 3; i++)

        {

            bl = shiftRows(block, i, i);

            for (j = 0; j <= 3; j++)

            {

                block[i][j] = bl.matrix[i][j];

                cout << block[i][j] << "\t ";

            }

            cout << "\n";

        }

        cout << "-----------------------" << endl

             << "Mixcolumn block"

             << endl;

        struct blockStruct bl\_mixColumns;

        bl\_mixColumns = mixColumns(block);

        for (i = 0; i <= 3; i++)

        {

            for (j = 0; j <= 3; j++)

            {

                block[i][j] = bl\_mixColumns.matrix[i][j];

                cout << block[i][j] << "\t ";

            }

            cout << "\n";

        }

        cout << "-----------------------" << endl

             << "Round Key"

             << endl;

        string round\_key[4][4];

        int key\_row = 0;

        int key\_col = 0;

        for (char character : current\_key)

        {

            int asciiValue = (int)character; // Get the ASCII value for each character

            stringstream key\_stream;

            key\_stream << hex << asciiValue; // convert to hex

            string res(key\_stream.str());

            round\_key[key\_row][key\_col] = res;

            if (key\_col >= 3)

            {

                key\_col = 0;

                key\_row += 1;

            }

            else

                key\_col += 1;

        }

        for (i = 0; i <= 3; i++)

        {

            for (j = 0; j <= 3; j++)

                cout << round\_key[i][j] << "\t ";

            cout << "\n";

        }

        cout << "-----------------------" << endl

             << rnd + 1 << " round Encrypted block"

             << endl;

        for (i = 0; i <= 3; i++)

        {

            for (j = 0; j <= 3; j++)

            {

                stringstream stream3;

                stringstream stream4;

                stream3 << hex << block[i][j];

                stream4 << hex << round\_key[i][j];

                unsigned m;

                stream3 >> m;

                bitset<8> block\_cell\_value(m);

                stream4 >> m;

                bitset<8> key\_cell\_value(m);

                bitset<8> roundkey\_xor;

                for (int t = 0; t < 10; t++)

                {

                    roundkey\_xor[t] = (key\_cell\_value[t] ^ block\_cell\_value[t]);

                }

                stringstream stream5;

                stream5 << hex << uppercase << roundkey\_xor.to\_ulong();

                block[i][j] = stream5.str();

                cout << block[i][j] << "\t ";

            }

            cout << "\n";

        }

        cout << endl

             << "start over to next round" << endl;

    }

    cout << endl;

}