**AES\_Process.h**

#include <string>

#include <map>

#include <iostream>

#include <vector>

#include <sstream>

#include <cstring>

#include <fstream>

#include <bitset>

#include <random>

#include <iterator>

using namespace std;

struct blockStruct

{

    // block declared inside structure

    string matrix[4][4];

};

struct blockStruct subBytes(string mat[4][4], map<char, map<char, string>> SBox)

{

    struct blockStruct BS;

    for (int i = 0; i <= 3; i++)

    {

        for (int j = 0; j <= 3; j++)

        {

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

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

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

            mat[i][j] = conv;

        }

    }

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

{

    struct blockStruct BS;

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

    {

        string lastColumn = mat[row][0];

        // string lastColumn = mat[row][3];

        // for (int j = 3; j > 0; j--)

        for (int j = 0; j < 3; j++)

        {

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

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

        }

        // mat[row][0] = lastColumn;

        mat[row][3] = 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 << xor\_each\_cell.to\_ulong();

            if (stream2.str().size() == 1)

                mixed\_matrix[i][z] = "0" + stream2.str();

            else

                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;

}

struct blockStruct addRoundKey(string mat[4][4], string key[4][4])

{

    struct blockStruct BS;

    for (int i = 0; i <= 3; i++)

    {

        for (int j = 0; j <= 3; j++)

        {

            stringstream stream3;

            stringstream stream4;

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

            stream4 << hex << key[i][j];

            unsigned m;

            stream3 >> m;

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

            unsigned m2;

            stream4 >> m2;

            bitset<8> key\_cell\_value(m2);

            bitset<8> roundkey\_xor;

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

            mat[i][j] = roundkey\_xor.to\_string();

        }

    }

    for (int i = 0; i <= 3; i++)

        for (int j = 0; j <= 3; j++)

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

    return BS;

}

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;

}

**AES.cpp**

#include "AES\_Process.h"

int main()

{

    string msg\_full = ""; // The message

    string encrypted\_msg = "";

    string encrypted\_snippet = "";

    string txt = "";

    ifstream MyReadFile("test.txt");

    while (getline(MyReadFile, txt))

    {

        msg\_full += txt;

    }

    MyReadFile.close();

    string msg = "";

    int msg\_size = msg\_full.size();

    int msg\_blocks\_qty;

    if (msg\_size <= 16)

        msg\_blocks\_qty = 16 - msg\_size;

    else

        msg\_blocks\_qty = 16 - (msg\_size % 16);

    for (int u = 0; u < msg\_blocks\_qty; u++)

        msg\_full += "x";

    cout << "message blocks qty: " << msg\_blocks\_qty << endl;

    msg\_size = msg\_full.size();

    int current\_msg\_block = 0;

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

         << "\n"

         << "1. AES 128 bit"

         << "\n"

         << "2. AES 256 bit"

         << "\n"

         << "3. AES 512 bit"

         << "\n"

         << "4. AES 1024 bit"

         << endl;

    int bytes;

    int rounds\_count;

    int key\_chage\_round;

    bool check = false;

    int col\_count;

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

    string initial\_key;

    string key\_txt = "";

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

                             "00000010",

                             "00000100",

                             "00001000",

                             "00010000",

                             "00100000",

                             "01000000",

                             "10000000",

                             "00011011",

                             "00110110"};

    int Rcon\_index = 0;

    string input\_AES\_Type;

    while (!check)

    {

        cout << "Please choose the selection number from 1,2,3 or 4." << endl;

        cin >> input\_AES\_Type;

        if (input\_AES\_Type == "1")

        {

            bytes = 16;

            rounds\_count = 10;

            key\_chage\_round = 1;

            ifstream MyReadFile2("My\_AES128\_Key.key");

            while (getline(MyReadFile2, key\_txt))

            {

                initial\_key += key\_txt;

            }

            MyReadFile2.close();

            break;

        }

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

        {

            bytes = 32;

            rounds\_count = 14;

            key\_chage\_round = 2;

            ifstream MyReadFile2("My\_AES256\_Key.key");

            while (getline(MyReadFile2, key\_txt))

            {

                initial\_key += key\_txt;

            }

            MyReadFile2.close();

            break;

        }

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

        {

            bytes = 64;

            rounds\_count = 22;

            key\_chage\_round = 4;

            ifstream MyReadFile2("My\_AES512\_Key.key");

            while (getline(MyReadFile2, key\_txt))

            {

                initial\_key += key\_txt;

            }

            MyReadFile2.close();

            break;

        }

        else if (input\_AES\_Type == "4")

        {

            bytes = 128;

            rounds\_count = 33;

            key\_chage\_round = 8;

            ifstream MyReadFile2("My\_AES1024\_Key.key");

            while (getline(MyReadFile2, key\_txt))

            {

                initial\_key += key\_txt;

            }

            MyReadFile2.close();

            break;

        }

        else

        {

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

        }

    }

    // Changing the initial key ASCII to hex

    //--------------------------------------

    string initial\_key\_hex = "";

    for (char character : initial\_key)

    {

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

        stringstream key\_stream;

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

        // initial\_key\_hex += key\_stream.str();

        if (key\_stream.str().size() == 1)

            initial\_key\_hex += "0" + key\_stream.str();

        else if (key\_stream.str().size() > 2)

            initial\_key\_hex += key\_stream.str().substr(key\_stream.str().size() - 2, 2);

        else

            initial\_key\_hex += key\_stream.str();

    }

    //++++++++++++++++FOR TESTIGN+++++++++++++++++++++//

    // initial\_key\_hex = "2b28ab097eaef7cf15d2154f16a6883c";

    //++++++++++++++++FOR TESTIGN+++++++++++++++++++++//

    //--------------------------------------

    // Building the S-Box table from csv file to a map variable

    //--------------------------------------

    ifstream fin("SBOX.csv", ios::in);

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

    string word;

    char r[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

    char c[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

    int row\_index = 0;

    int col\_index = 0;

    while (getline(fin, word, ','))

    {

        if (word.find('\n') != std::string::npos)

        {

            SBox[r[row\_index]][c[col\_index]] = word[0];

            SBox[r[row\_index]][c[col\_index]] += word[1];

            row\_index++;

            col\_index = 0;

            SBox[r[row\_index]][c[col\_index]] = word[3];

            SBox[r[row\_index]][c[col\_index]] += word[4];

            col\_index++;

            continue;

        }

        SBox[r[row\_index]][c[col\_index]] = (word);

        col\_index++;

    }

    fin.close();

    //--------------------------------------

    for (int block\_index = 0; block\_index < msg\_size / 16; block\_index++)

    {

        Rcon\_index = 0;

        msg = msg\_full.substr(current\_msg\_block, 16);

        current\_msg\_block += 16;

        int key\_change\_round\_check = 0;

        if (input\_AES\_Type == "1")

            key\_change\_round\_check = 1;

        cout << "The full message: " << msg\_full << endl;

        cout << "The message block: " << msg << endl;

        string add\_round\_key;

        string current\_key;

        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;

        }

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

        // for (int rnd = 0; rnd < 1; rnd++)

        {

            if (rnd == 0)

            {

                add\_round\_key = initial\_key\_hex;

            }

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

            {

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

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

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

                int key\_row = 0;

                int key\_col = 0;

                int e = 0;

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

                {

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

                    {

                        round\_key[i][j] = add\_round\_key[e];

                        round\_key[i][j] += add\_round\_key[e + 1];

                        e += 2;

                    }

                }

                // Shifting the previous key block last column's rows

                //-------------------------------------------------

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

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

                next\_key[2][0] = round\_key[3][col\_count - 1];

                next\_key[3][0] = round\_key[0][col\_count - 1];

                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 << "shifting: " << next\_key[i][0] << "\n ";

                }

                //-------------------------------------------------

                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]);

                    bitset<8> xor\_columns;

                    if (i == 0)

                        xor\_columns = (coloumn1 ^ coloumn2 ^ coloumn3);

                    else

                        xor\_columns = (coloumn1 ^ coloumn2);

                    stringstream stream\_value;

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

                    if (stream\_value.str().size() == 1)

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

                    else

                        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 << xor\_columns.to\_ulong();

                        if (stream\_value.str().size() == 1)

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

                        else

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

                    }

                }

                string bit\_to\_string = "";

                add\_round\_key = "";

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

                {

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

                    {

                        add\_round\_key += next\_key[i][j];

                    }

                }

                Rcon\_index++;

                key\_change\_round\_check = 0;

            }

            if (rnd == 0 && input\_AES\_Type == "1")

            {

                current\_key = initial\_key\_hex.substr(0, 32);

                cout << "The round full key: " << initial\_key\_hex << endl;

            }

            else if (rnd < key\_chage\_round - 1)

            {

                current\_key = initial\_key\_hex.substr((key\_change\_round\_check \* 31) + key\_change\_round\_check, 32);

                cout << "The round full key: " << initial\_key\_hex << endl;

            }

            else

            {

                current\_key = add\_round\_key.substr((key\_change\_round\_check \* 31) + key\_change\_round\_check, 32);

                cout << "The round full key: " << add\_round\_key << endl;

            }

            key\_change\_round\_check++;

            cout << "The 4x4 key: " << current\_key << endl;

            cout << endl

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

                 << "The current block"

                 << endl;

            int i, j;

            int Hi = 0;

            string test\_hex[16] = {"32", "88", "31", "e0", "43", "5a", "31", "37", "f6", "30", "98", "07", "a8", "8d", "a2", "34"};

            string test\_hex2[16] = {"00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00"};

            for (i = 0; i <= 3; i++)

            {

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

                {

                    if (rnd == 0)

                        block[i][j] = test\_hex[Hi];

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

                    Hi++;

                }

                cout << "\n";

            }

            if (rnd == 0)

            {

                cout << endl

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

                     << "The initial key"

                     << endl;

                string round\_key1[4][4];

                int e = 0;

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

                {

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

                    {

                        round\_key1[i][j] = current\_key[e];

                        round\_key1[i][j] += current\_key[e + 1];

                        cout << round\_key1[i][j] << "\t ";

                        e += 2;

                    }

                    cout << "\n";

                }

                cout << endl

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

                     << "Block state after initial key"

                     << endl;

                struct blockStruct bl\_RndKey;

                bl\_RndKey = addRoundKey(block, round\_key1);

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

                {

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

                    {

                        stringstream stream5;

                        bitset<8> roundkey\_xor(bl\_RndKey.matrix[i][j]);

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

                        if (stream5.str().size() == 1)

                            block[i][j] = "0" + stream5.str();

                        else

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

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

                    }

                    cout << "\n";

                }

            }

            cout << endl

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

                 << "Sub byte block"

                 << endl;

            struct blockStruct blSB;

            blSB = subBytes(block, SBox);

            for (i = 0; i <= 3; i++)

            {

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

                {

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

                    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";

            }

            if (rnd != rounds\_count - 1)

            {

                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\_key1[4][4];

            int e = 0;

            if (rnd == 0 && input\_AES\_Type == "1")

                current\_key = add\_round\_key.substr(0, 32);

            else if (rnd < key\_chage\_round - 1)

            {

                current\_key = initial\_key\_hex.substr((key\_change\_round\_check \* 31) + key\_change\_round\_check, 32);

                key\_change\_round\_check++;

            }

            for (i = 0; i <= 3; i++)

            {

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

                {

                    round\_key1[i][j] = current\_key[e];

                    round\_key1[i][j] += current\_key[e + 1];

                    cout << round\_key1[i][j] << "\t ";

                    e += 2;

                }

                cout << "\n";

            }

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

                 << rnd + 1 << " --> " << rounds\_count << " round Encrypted block"

                 << endl;

            encrypted\_snippet = "";

            struct blockStruct bl\_RndKey;

            bl\_RndKey = addRoundKey(block, round\_key1);

            for (i = 0; i <= 3; i++)

            {

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

                {

                    stringstream stream5;

                    bitset<8> roundkey\_xor(bl\_RndKey.matrix[i][j]);

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

                    if (stream5.str().size() == 1)

                        block[i][j] = "0" + stream5.str();

                    else

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

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

                }

                cout << "\n";

            }

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

            {

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

                {

                    encrypted\_snippet += block[j][i];

                }

            }

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

                 << "the encrypted block in this round: " << encrypted\_snippet << endl;

            cout << endl

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

        }

        encrypted\_msg += encrypted\_snippet;

    }

    cout << "the fully encrypted block: " << encrypted\_msg << endl;

    ofstream Encrypted\_file;

    Encrypted\_file.open("Message\_Enc.dat");

    Encrypted\_file << encrypted\_msg;

    Encrypted\_file.close();

    string stop\_running;

    cin >> stop\_running;

    cout << endl;

}