Nam : Shabbar Adamjee  
Roll No.: PB57  
PRN: 1032221508

*ICS LAB ASSIGNMENT 3*

*Simple AES*

*Code*

#include <algorithm>

#include <array>

#include <bitset>

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <map>

#include <sstream>

#include <string>

#include <vector>

std::array<std::array<int, 4>, 4> sBox{{

    {9, 4, 10, 11},

    {13, 1, 8, 5},

    {6, 2, 0, 3},

    {12, 14, 15, 7},

}};

std::array<std::array<int, 15>, 4> mixColumnTable{

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

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

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

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

std::map<int, int> mixColumnMapper{{1, 0}, {2, 1}, {4, 2}, {9, 3}};

std::bitset<8> gFunction(std::bitset<8> w) {

  std::string N0, N1;

  N0 = w.to\_string().substr(0, 4);

  N1 = w.to\_string().substr(4, 4);

  std::bitset<4> N0bits(N0);

  std::bitset<4> N1bits(N1);

  std::bitset<4> temp;

  temp = N0bits;

  N0bits = N1bits;

  N1bits = temp;

  int row1 = (N0bits[3] << 1) | N0bits[2]; // First 2 bits (from left to right)

  int col1 = (N0bits[1] << 1) | N0bits[0]; // Next 2 bits

  int row2 = (N1bits[3] << 1) | N1bits[2]; // First 2 bits (from left to right)

  int col2 = (N1bits[1] << 1) | N1bits[0]; // Next 2 bits

  N0bits = sBox[row1][col1];

  N1bits = sBox[row2][col2];

  unsigned long randomNum = rand() % 256;

  std::cout << "Random number in gFunction: " << randomNum << std::endl;

  std::bitset<8> randomNumBits{randomNum};

  std::bitset<8> final(N0bits.to\_string() + N1bits.to\_string());

  return (final ^ randomNumBits);

}

std::string expandKey(std::bitset<8> w0, std::bitset<8> w1) {

  gFunction(w1);

  std::bitset<8> w2 = w0 ^ gFunction(w1);

  std::bitset<8> w3 = w1 ^ w2;

  return (w2.to\_string() + w3.to\_string());

}

std::bitset<16> round1(std::string intermediate, std::string key1) {

  std::string nibble1 = intermediate.substr(0, 4);

  std::string nibble2 = intermediate.substr(4, 4);

  std::string nibble3 = intermediate.substr(8, 4);

  std::string nibble4 = intermediate.substr(12, 4);

  std::bitset<4> bNibble1(nibble1);

  std::bitset<4> bNibble2(nibble2);

  std::bitset<4> bNibble3(nibble3);

  std::bitset<4> bNibble4(nibble4);

  std::vector<std::bitset<4>> nibbles{bNibble1, bNibble2, bNibble3, bNibble4};

  for (int i = 0; i < nibbles.size(); i++) {

    int row = (nibbles[i][3] << 1) | nibbles[i][2];

    int col = (nibbles[i][1] << 1) | nibbles[i][0];

    nibbles[i] = sBox[row][col];

  }

  // Shift rows; swap 2nd and 4th

  std::bitset<4> temp = nibbles[1];

  nibbles[1] = nibbles[3];

  nibbles[1] = temp;

  // Mix columns

  std::array<std::array<int, 2>, 2> matrix;

  int count = 0;

  for (int i = 0; i < 2; i++) {

    for (int j = 0; j < 2; j++) {

      matrix[i][j] = nibbles[count].to\_ulong();

      count++;

    }

  }

  std::array<std::array<int, 2>, 2> mulMat{{{1, 4}, {4, 1}}};

  std::array<std::array<int, 2>, 2> result;

  // 0, 0 x 0, 0 XOR 0, 1 x 0, 1             1, 0 x 0, 0 XOR 1, 1 x 0, 1

  // 0, 0 x 1, 0 XOR 0, 1 x 1, 1             1, 0 x 1, 0 XIR 1, 1 x 1, 1

  result[0][0] =

      mixColumnTable[mixColumnMapper[mulMat[0][0]]][matrix[0][0] - 1] ^

      mixColumnTable[mixColumnMapper[mulMat[0][1]]][matrix[0][1] - 1];

  result[1][0] =

      mixColumnTable[mixColumnMapper[mulMat[1][0]]][matrix[0][0] - 1] ^

      mixColumnTable[mixColumnMapper[mulMat[1][1]]][matrix[0][1] - 1];

  result[0][1] =

      mixColumnTable[mixColumnMapper[mulMat[1][1]]][matrix[1][0] - 1] ^

      mixColumnTable[mixColumnMapper[mulMat[0][1]]][matrix[1][1] - 1];

  result[1][1] =

      mixColumnTable[mixColumnMapper[mulMat[1][0]]][matrix[1][0] - 1] ^

      mixColumnTable[mixColumnMapper[mulMat[1][1]]][matrix[1][1] - 1];

  count = 0;

  for (int i = 0; i < 2; i++) {

    for (int j = 0; j < 2; j++) {

      nibbles[count] = result[j][i];

      count++;

    }

  }

  std::string almost = "";

  for (int i = 0; i < nibbles.size(); i++) {

    almost += nibbles[i].to\_string();

  }

  std::bitset<16> almostBits(almost);

  std::bitset<16> key(key1);

  return (almostBits ^ key);

}

std::bitset<16> round2(std::string intermediate, std::string key2) {

  std::string nibble1 = intermediate.substr(0, 4);

  std::string nibble2 = intermediate.substr(4, 4);

  std::string nibble3 = intermediate.substr(8, 4);

  std::string nibble4 = intermediate.substr(12, 4);

  std::bitset<4> bNibble1(nibble1);

  std::bitset<4> bNibble2(nibble2);

  std::bitset<4> bNibble3(nibble3);

  std::bitset<4> bNibble4(nibble4);

  std::vector<std::bitset<4>> nibbles{bNibble1, bNibble2, bNibble3, bNibble4};

  for (int i = 0; i < nibbles.size(); i++) {

    int row = (nibbles[i][3] << 1) | nibbles[i][2];

    int col = (nibbles[i][1] << 1) | nibbles[i][0];

    nibbles[i] = sBox[row][col];

  }

  // Shift rows; swap 2nd and 4th

  std::bitset<4> temp = nibbles[1];

  nibbles[1] = nibbles[3];

  nibbles[1] = temp;

  std::string almost = "";

  for (int i = 0; i < nibbles.size(); i++) {

    almost += nibbles[i].to\_string();

  }

  std::bitset<16> almostBits(almost);

  std::bitset<16> key(key2);

  return (almostBits ^ key);

}

int main() {

  srand(time(0));

  std::string plainText;

  std::string key0;

  std::cout << "Enter plain text (space separated): ";

  std::getline(std::cin, plainText);

  std::cout << "Enter key 0 (space separated): ";

  std::getline(std::cin, key0);

  std::cout << std::endl;

  std::string temp, w0, w1;

  std::stringstream ss{key0};

  ss >> temp;

  w0 = temp;

  ss >> temp;

  w0 += temp;

  ss >> temp;

  w1 = temp;

  ss >> temp;

  w1 += temp;

  std::bitset<8> w0Bits(w0);

  std::bitset<8> w1Bits(w1);

  std::string key1 = expandKey(w0Bits, w1Bits);

  std::cout << "Key 1: " << key1 << std::endl << std::endl;

  std::bitset<8> w2Bits(key1.substr(0, 8));

  std::bitset<8> w3Bits(key1.substr(8, 8));

  std::string key2 = expandKey(w2Bits, w3Bits);

  std::cout << "Key 2: " << key2 << std::endl << std::endl;

  plainText.erase(std::remove(plainText.begin(), plainText.end(), ' '),

                  plainText.end());

  key0.erase(std::remove(key0.begin(), key0.end(), ' '), key0.end());

  std::bitset<16> ptBits(plainText);

  std::bitset<16> keyBits(key0);

  std::bitset<16> result = ptBits ^ keyBits;

  result = round1(result.to\_string(), key1);

  result = round2(result.to\_string(), key2);

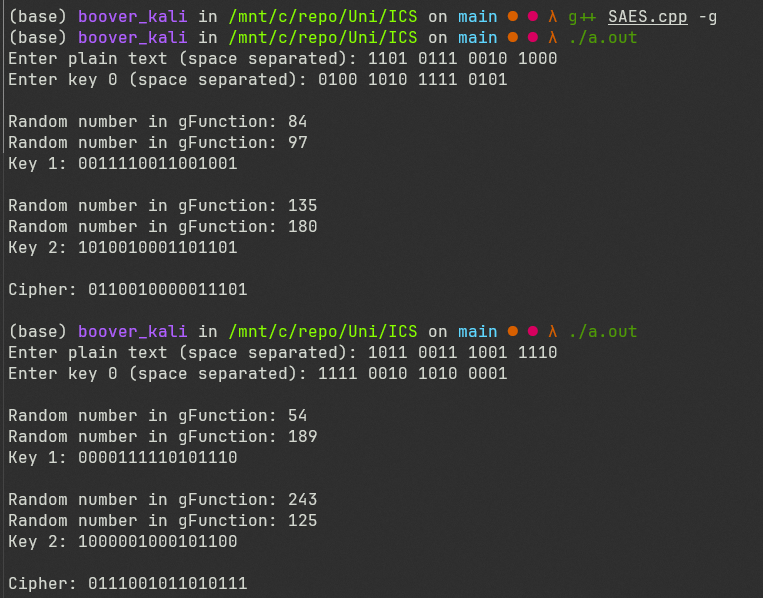
  std::cout << "Cipher: " << result << std::endl;

  std::cout << std::endl;

  return 0;

}

*Output*

**