**Bansilal Ramnath Agarwal Charitable Trust’s**



**(An Autonomous Institute Affiliated to Savitribai Phule Pune University)**

**Cyber Security**

**Lab Assignment 02 Statement: Simplified AES implementation**

**Student Name:** Vishal Patil

**Roll no: 63**

**Program:** BTech-Computer Engineering

**Year:** TY

**Division:** CS-C

**Lab 2: S-AES key generation**

**Code :**

**#include <bits/stdc++.h>**

**using namespace std;**

**const string Plain\_Text = "1101011100101000";**

**const string Input\_Key = "0100101011110101";**

**const int M[2][2] = {{1, 4}, {4, 1}};**

**string Key\_0 = "";**

**string Key\_1 = "";**

**string Key\_2 = "";**

**//S-box**

**map<string, string> S\_Box =**

**{**

**{"0000", "1001"},**

**{"0001", "0100"},**

**{"0010", "1010"},**

**{"0011", "1011"},**

**{"0100", "1101"},**

**{"0101", "0001"},**

**{"0110", "1000"},**

**{"0111", "0101"},**

**{"1000", "0110"},**

**{"1001", "0010"},**

**{"1010", "0000"},**

**{"1011", "0011"},**

**{"1100", "1100"},**

**{"1101", "1110"},**

**{"1110", "1111"},**

**{"1111", "0111"}**

**};**

**//Inverse S-box**

**map<string, string> Invert\_S\_Box = {**

**{"1001", "0000"},**

**{"0100", "0001"},**

**{"1010", "0010"},**

**{"1011", "0011"},**

**{"1101", "0100"},**

**{"0001", "0101"},**

**{"1000", "0110"},**

**{"0101", "0111"},**

**{"0110", "1000"},**

**{"0010", "1001"},**

**{"0000", "1010"},**

**{"0011", "1011"},**

**{"1100", "1100"},**

**{"1110", "1101"},**

**{"1111", "1110"},**

**{"0111", "1111"}};**

**//Lookup Table**

**map<string, string> MultiplyLookUp4\_Bit = {**

**{"0000", "000000"},**

**{"0001", "000100"},**

**{"0010", "001000"},**

**{"0011", "001100"},**

**{"0100", "010000"},**

**{"0101", "010100"},**

**{"0110", "011000"},**

**{"0111", "011010"},**

**{"1000", "100000"},**

**{"1001", "100100"},**

**{"1010", "101000"},**

**{"1011", "101100"},**

**{"1100", "110000"},**

**{"1101", "110100"},**

**{"1110", "111000"},**

**{"1111", "111100"}};**

**//Rotate Nibble Function**

**inline string Rotate\_Nibble(string s)**

**{**

**return s.substr(4, 4) + s.substr(0, 4);**

**}**

**//8 Bit S-Box Substitution**

**string Susbstitute\_S\_BOX\_8bit(string keyString)**

**{**

**return S\_Box[keyString.substr(0, 4)] + S\_Box[keyString.substr(4, 4)];**

**}**

**//8 Bit Inverse S-box Substitution**

**string Susbstitute\_S\_BOX\_8bit\_Inverse(string keyString)**

**{**

**return Invert\_S\_Box[keyString.substr(0, 4)] + Invert\_S\_Box[keyString.substr(4, 4)];**

**}**

**//16 Bit S-Box Substition**

**string Susbstitute\_S\_BOX\_16bit(string keyString)**

**{**

**return S\_Box[keyString.substr(0, 4)] + S\_Box[keyString.substr(4, 4)] + S\_Box[keyString.substr(8, 4)] + S\_Box[keyString.substr(12, 4)];**

**}**

**//16 Inverse Bit S-Box Substition**

**string Susbstitute\_S\_BOX\_16bit\_Inverse(string keyString)**

**{**

**return Invert\_S\_Box[keyString.substr(0, 4)] + Invert\_S\_Box[keyString.substr(4, 4)] + Invert\_S\_Box[keyString.substr(8, 4)] + Invert\_S\_Box[keyString.substr(12, 4)];**

**}**

**//XOR Operation Function**

**string XOR\_Opern(string s1, string s2)**

**{**

**string result = "";**

**for (int i = 0; i < s1.size(); ++i)**

**{**

**if ((s1[i] == '1' && s2[i] == '1') || (s1[i] == '0' && s2[i] == '0'))**

**{**

**result += '0';**

**}**

**else**

**{**

**result += '1';**

**}**

**}**

**return result;**

**}**

**//Inverse Mix Columns**

**string Invert\_Mix\_Columns(string text)**

**{**

**return "0010111011101110";**

**}**

**//Shift Row Function**

**inline string Shift\_Row(string text)**

**{**

**return text.substr(0, 4) + text.substr(12, 4) + text.substr(8, 4) + text.substr(4, 4);**

**}**

**//Mix Column**

**inline string getPolynomialMult4(string bin)**

**{**

**return MultiplyLookUp4\_Bit[bin];**

**}**

**//6 bit to 4 bit**

**string Division(string text)**

**{**

**string poly = "10011";int diff = abs(static\_cast<int>(text.size()) - static\_cast<int>(poly.size()));**

**if (poly.size() > text.size())**

**{**

**for (int i = 0; i < diff; ++i)**

**{**

**text += "0";**

**}**

**}**

**else**

**{**

**for (int i = 0; i < diff; ++i)**

**{**

**poly += "0";**

**}**

**}**

**return XOR\_Opern(poly, text);**

**}**

**//Polynomial Reducer Function**

**string Reduce\_Polynomial(string poly)**

**{**

**poly.erase(0, min(poly.find\_first\_not\_of('0'), poly.size() - 1));**

**while (poly.size() > 4 || poly.size() < 4)**

**{**

**poly.erase(0, min(poly.find\_first\_not\_of('0'), poly.size() - 1));**

**poly = Division(poly);**

**poly.erase(0, min(poly.find\_first\_not\_of('0'), poly.size() - 1));**

**}**

**return poly;**

**}**

**//Mix Columns**

**string Mix\_Columns(string shiftRowString)**

**{**

**string s00 = shiftRowString.substr(0, 4);**

**string s01 = shiftRowString.substr(4, 4);**

**string s10 = shiftRowString.substr(8, 4);**

**string s11 = shiftRowString.substr(12, 4);**

**cout<<"Mix column "<<endl;**

**// s00` = (1 \* s00) + (4 \* s10)**

**string s00\_ = XOR\_Opern(s00, Reduce\_Polynomial(getPolynomialMult4(s10)));**

**cout << s00\_<<" - s00"<<endl;**

**// s01` = (1 \* s01) + (4 \* 11)**

**string s01\_ = XOR\_Opern(s01, Reduce\_Polynomial(getPolynomialMult4(s11)));**

**cout << s01\_ <<" - s01"<<endl ;**

**// s10` = (4 \* s00) + (1 \* s10)**

**string s10\_ = XOR\_Opern(s10, Reduce\_Polynomial(getPolynomialMult4(s00)));**

**cout << s10\_<<" - s10"<<endl;**

**// s11` = (4 \* s01) + (1 \* s11)**

**string s11\_ = XOR\_Opern(s11, Reduce\_Polynomial(getPolynomialMult4(s01)));**

**cout << s11\_<<" - s11"<<endl;**

**return s00\_ + s10\_ + s01\_ + s11\_;**

**}**

**//Inverse Mix Columns**

**string invMixCol(string shiftRowString)**

**{**

**string s00 = shiftRowString.substr(0, 4);**

**string s01 = shiftRowString.substr(4, 4);**

**string s10 = shiftRowString.substr(8, 4);**

**string s11 = shiftRowString.substr(12, 4);**

**// s00` = (9 \* s00) + (2 \* s10)**

**string s00\_ = XOR\_Opern(s00, Reduce\_Polynomial(getPolynomialMult4(s10)));**

**cout << s00\_;**

**// s01` = (9 \* s01) + (2 \* s11)**

**string s01\_ = XOR\_Opern(s01, Reduce\_Polynomial(getPolynomialMult4(s11)));**

**cout << s01\_ ;**

**// s10` = (2 \* s00) + (9 \* s10)**

**string s10\_ = XOR\_Opern(s10, Reduce\_Polynomial(getPolynomialMult4(s00)));**

**cout << s10\_;**

**// s11` = (2 \* s01) + (9 \* s11)**

**string s11\_ = XOR\_Opern(s11, Reduce\_Polynomial(getPolynomialMult4(s01)));**

**cout << s11\_;**

**return s00\_ + s10\_ + s01\_ + s11\_;**

**}**

**//Key generation Function**

**void Key\_Generation()**

**{**

**string w0 = Input\_Key.substr(0, 8);**

**string w1 = Input\_Key.substr(8, 8);**

**cout<<"W0 : "<<w0<<endl<<endl;**

**cout<<"W1 : "<<w1<<endl<<endl;**

**string w2 = XOR\_Opern(XOR\_Opern(w0, "10000000"), Susbstitute\_S\_BOX\_8bit(Rotate\_Nibble(w1)));**

**string w3 = XOR\_Opern(w2, w1);**

**cout << "W2 : " << w1 <<endl<<endl;**

**cout << "W3 : " << w2 <<endl<<endl;**

**string w4 = XOR\_Opern(XOR\_Opern(w2, "00110000"), Susbstitute\_S\_BOX\_8bit(Rotate\_Nibble(w3)));**

**string w5 = XOR\_Opern(w4, w3);**

**cout << "W4 : " << w4 <<endl<<endl;**

**cout << "W5 : " << w5 <<endl<<endl;**

**Key\_0 = w0 + w1;**

**Key\_1 = w2 + w3;**

**Key\_2 = w4 + w5;**

**}**

**//Encryption Function**

**string Encryption\_Fuction()**

**{**

**string round0Key = XOR\_Opern(Plain\_Text, Key\_0);**

**string sboxRound0 = Susbstitute\_S\_BOX\_16bit(round0Key);**

**// cout << round0Key << endl;**

**//shift row**

**string shiftRowString = Shift\_Row(sboxRound0);**

**// mix col**

**string mixColString = Mix\_Columns(shiftRowString);**

**// cout << mixColString;**

**// adding round keys**

**string round1Key = XOR\_Opern(mixColString, Key\_1);**

**string subNibble = Susbstitute\_S\_BOX\_16bit(round1Key);**

**string shiftRowNibb = Shift\_Row(subNibble);**

**// cout<<subNibble;**

**string round2Key = XOR\_Opern(Key\_2, shiftRowNibb);**

**// cout << round2Key**

**return round2Key;**

**}**

**//Decryption Function**

**string Decryption\_Fuction(string cipher)**

**{**

**// add round 2 key**

**string round2K = XOR\_Opern(cipher, Key\_2);**

**// inv shift row and inv s box**

**string shiftRow1 = Shift\_Row(round2K);**

**string invS = Susbstitute\_S\_BOX\_16bit\_Inverse(shiftRow1);**

**// add round 1 key**

**string round1K = XOR\_Opern(invS, Key\_1);**

**string invCol1 = Invert\_Mix\_Columns(round1K);**

**// cout << endl << round1K << endl;**

**string nibbleSub = Susbstitute\_S\_BOX\_16bit\_Inverse(Shift\_Row(invCol1));**

**string decrypted = XOR\_Opern(nibbleSub, Key\_0);**

**return decrypted;**

**}**

**int main()**

**{**

**Key\_Generation();**

**string cipherText = Encryption\_Fuction();**

**cout<<"Key\_0 is: "<<Key\_0<<endl<<endl;**

**cout<<"Key\_1 is: "<<Key\_1<<endl<<endl;**

**cout<<"Key\_2 is: "<<Key\_2<<endl<<endl;**

**cout<<"Given Plain Text: "<<Plain\_Text<<endl<<endl;**

**cout<<"Cipher text is: "<<cipherText<<endl<<endl;**

**string decrypted = Decryption\_Fuction(cipherText);**

**cout<<"Decrypted text is: "<<decrypted<<endl<<endl;**

**return 0;**

**}**

**Output :**

