上海理工大学光电信息与计算机工程学院

**《信息安全》实验报告**

****

**专　　业 计算机科学与技术**

**学生姓名　　许耀永**

**学　 号 1712480131**

**年　　级 2017级**

**指导教师 刘亚**

**成 绩：**

**教师签字：**

报告格式要求

1. 正文字体中文为宋体，五号，行距为固定值18磅，西文为Times New Rome, 五号，行距为固定值18磅。
2. 章节标题为加粗宋体，小四号，段前段后各0.5行，行距为固定值18磅。
3. 打印时需双面打印。

**实验一** **AES密码**

**一、 实验目的**

实现AES的加密(Encryption)和解密(Decryption)

**二、AES基础**

(1)AES的所有操作都是针对于一个二维数组state

一个state数组由 4行Nb列组成

(2)AES基于下列5个基本操作

1.异或(Exclusive Disjunction/XOR):当两个digit不同时输出为真

2.SubByte/InverseSubByte : 一个byte被另一个byte替代

加密时根据S\_Box查表，解密时根据Si\_Box查表

(每个byte都可以表示为两个十六进制的digit，

其中第一个代表S\_Box的行，第二个代表S\_Box的列)

3.Shift Rows(Rotation)/Inverse Shift Rows:

重排bytes，以不同的偏移量循环左移后三行

Row 0不移动

Row 1 移动一个byte

Row 2 移动两个byte

Row3 移动三个byte

解密时循环右移相对应的byte

4.MixColumns/InverseMixColumns:

二维数组state的每一列上的每一个byte映射为一个新的值

a.将state的每一列左乘一个4 \* Nb的矩阵，所得结果逐个异或而产生一个byte

b. a的乘法操作可以用两个查找表L\_Table和E\_Table来实现

step1 :一个byte，其表示为两个十六进制的digit，

而第一个digit用作L\_Table的行数，第二个digit用作L\_Table的列数

step2 : 将两个从L\_Table获得的值相加，得到一个新的byte

step3 : 在E\_Table查看这个相加的结果(按digit来查)

Note:任何数与1相乘的结果都是它本身

5.AddRoundKey:state数组的每个byte都和round key的每个byte相异或(XOR)

(3) 对于128比特的key,执行10 rounds(轮)

对于192比特的key,执行 12 rounds(轮)

对于 256比特的key,执行 14 rounds(轮)

(4) AES的框架

从第一轮到倒数第二轮，均执行:

a.SubBytes/InverseSubBytes

b.ShiftRows/InverseShiftRows

c.MixColumns/InverseMixColumns

d.AddRoundKey

最后一轮 不执行 MixColumns/InverseMixColumns

(6) Key Expansion

将一个长度为Nk的输入key生成一个长度为Nb \* (Nr + 1)的线性key

Nb : state数组的列数

Nr: 轮的个数

Nk: key的长度

记W[i]为expanded key的第i个word

**三、程序的数据结构设计和算法描述**

(1) Key Expansion

Begin

word temp;

for i <- 0 to (Nk - 1)

w[i] <- ((unsigned char) key [ 4 \* i ] << 24)

| ((unsigned char) key [ 4 \* i + 1 ] << 16)

| ((unsigned char) key [ 4 \* i + 2 ] << 8)

| ((unsigned char) key [ 4 \* i + 3 ] );

end for

for i <- (Nk - 1) to Nb \* Nr

temp = w[i - 1];

if (i mod Nk == 0)

temp = SubWord ( RotWord (temp))

XOR (Rcon [ i / Nk ] << 24) ;

else if ( Nk > 6 and ( i mod Nk ) == 4 )

temp = SubWord(temp);

end if

w[i] = w[i - Nk] XOR temp;

end for

End

(2) Encryption

Begin

state = plaintext

1.KeyExpansion

2.AddRoundkey(state,Expandedkey[0])

3.for r <- 1 to (Nr - 1)

a. SubBytes (state , S-Box)

b. ShiftRows (state)

c. MixColumns(state)

d. AddRoundKey(state,ExpandedKey[r])

4. SubBytes (state , S-Box)

ShiftRows (state)

AddRoundKey(state,ExpandedKey[Nr])

Out = cipherText

End

(3) Decryption

Begin

State = cipherText

1.Key Expansion

2.AddRoundKey(state,ExpandedKey[0])

3.for r <- (Nr - 1) to 1

a. InverseShiftRows(state)

b. InverseSubBytes(state,S-Box)

c. AddRoundKey(State,ExpandedKey[r])

4. InverseSubBytes(state,S-Box)

InverseShiftRows(state)

AddRoundKey(state,ExpandedKey[Nr])

Out = plaintext

End

**四、程序代码**

//AES.h

#define word unsigned int

#define byte unsigned char

#include<iostream>

#include<vector>

#include<fstream>

#include<string>

#include<sstream>

using namespace std;

class AES

{

vector<word> ExpandedKey;

int Nk; //width of key block

int Nr; //number of round

int Nb; //block size

static const byte S\_Box[256];

static const byte Si\_Box[256];

static const byte Rcon[30];

static const byte ColMixMatrix[4][4];

static const byte InvColMixMatrix[4][4];

static const byte AlogTable[256];

static const byte LogTable[256];

string cipherText;

byte state[4][4];

byte Mul(byte a,byte b);

void MixColumns();

void ShiftRows();

void SubBytes();

byte SubByte(byte oneByte);

word SubWord(word val);

word RotWord(word val);

void InvMixColumns();

void InvShiftRows();

byte InvSubByte(byte oneByte);

void InvSubBytes();

void AddRoundKey(int roundNo);

void KeyExpansion(string key);

public:

enum KeySize {AES128 = 128,AES192 = 192,AES256 = 256};

AES(string key,int bitSize);

~AES();

void Encrypt(string plainText);

string GetCipherText();

void Decrypt(string cipherText);

string ToString();

};

//AES.cpp

#include "AES.h"

byte AES::Mul(byte a,byte b){

if(a && b){

return AlogTable[((unsigned char)LogTable[a] + (unsigned char)LogTable[b]) % 255];

}

return 0;

}

//Encrypt

void AES::MixColumns(){

byte temp[4];

for(int c = 0;c < Nb;c++){

//4 rows and Nb columns to store temp mix col value

for(int r = 0;r < 4;r++){

temp[r] = Mul(ColMixMatrix[r][0],(state[0][c]))

^Mul(ColMixMatrix[r][1],(state[1][c]))

^Mul(ColMixMatrix[r][2],(state[2][c]))

^Mul(ColMixMatrix[r][3],(state[3][c]));

}

state[0][c] = temp[0];

state[1][c] = temp[1];

state[2][c] = temp[2];

state[3][c] = temp[3];

}

}

void AES::ShiftRows(){

//row always 4

for(int r = 0;r < 4;r++){

byte temp[4];

for(int c = 0;c < Nb;c++){

temp[c] = state[r][(r + c) % Nb];

}

state[r][0] = temp[0];

state[r][1] = temp[1];

state[r][2] = temp[2];

state[r][3] = temp[3];

}

}

byte AES::SubByte(byte oneByte){

//one byte represent in hex (xy)

//x is row index AND y is column index

return S\_Box[oneByte];

}

void AES::SubBytes(){

for(int i = 0;i < 4;i++){

for(int j = 0;j < Nb;j++){

state[i][j] = SubByte(state[i][j]);

}

}

}

void AES::AddRoundKey(int roundNo){

for(int col = 0;col < Nb;col++){

word roundKeyVal = ExpandedKey[(roundNo \* Nb) + col];

for(int row = 3;row >= 0;row--){

state[row][col] ^= (roundKeyVal & 0xFF);

roundKeyVal = roundKeyVal >> 8;

}

}

}

//destructor

AES::~AES(){}

//constructor

AES::AES(string key,int bitSize){

Nr = bitSize / 32 + 6;

Nk = bitSize / 32;

Nb = 4; //always 4

ExpandedKey.resize(Nk \* (Nr + 1));

KeyExpansion(key);

}

word AES::SubWord(word val){

byte oneByte;

word res = 0;

for(int i = 0;i < 4;i++){

res = res << 8;

oneByte = (val >> 24) & 0xFF;

res = res | SubByte(oneByte);

val = val << 8;

}

return res;

}

word AES::RotWord(word val){

word res = val << 8;

res = res | (val >> 24);

return res;

}

void AES::KeyExpansion(string key){

word temp;

for(int i = 0;i < Nk;i++){

ExpandedKey[i] = ((unsigned char) key[4 \* i] << 24) |

((unsigned char) key[4 \* i + 1] << 16) |

((unsigned char) key[4 \* i + 2] << 8) |

((unsigned char) key[4 \* i + 3]);

}

for(int i = Nk - 1;i < Nb \* Nr + 1;i++){

temp = ExpandedKey[i - 1];

if(i % Nk == 0){

temp = (SubWord(RotWord(temp)))

^(Rcon[i / Nk] << 24);

}else if(Nk > 6 && (i % Nk == 4)){

temp = SubWord(temp);

}

ExpandedKey[i] = ExpandedKey[i - Nk] ^ temp;

}

}

void AES::Encrypt(string plainText){

if((plainText.length() % (4 \* Nb)) != 0)

plainText.append((4 \* Nb) - (plainText.length() % (4 \* Nb)),'\0');

int count = 0;

while(count < (plainText.length())){

//copy one block into state

for(int c = 0;c < Nb;c++){

for(int r = 0;r < 4;r++){

state[r][c] = plainText[count + (c \* Nb) + r];

}

}

AddRoundKey(0);

int i;

for(i = 1;i < Nr;i++){

SubBytes();

ShiftRows();

MixColumns();

AddRoundKey(i);

}

//finally

SubBytes();

ShiftRows();

AddRoundKey(Nr);

cipherText = cipherText + ToString();

count += 4 \* Nb;

}

}

string AES::GetCipherText(){

return cipherText;

}

string AES::ToString(){

string str;

for(int c = 0;c < Nb;c++){

for(int r = 0;r < Nb;r++){

str.push\_back(state[r][c]);

}

}

return str;

}

//Decrypt

void AES::InvMixColumns(){

byte temp[4];

for(int c = 0;c < Nb;c++){

//4 rows and Nb columns to store temp mix col value;

for(int r = 0;r < 4;r++){

temp[r] = Mul(InvColMixMatrix[r][0],(state[0][c]))

^ Mul(InvColMixMatrix[r][1],(state[1][c]))

^ Mul(InvColMixMatrix[r][2],(state[2][c]))

^ Mul(InvColMixMatrix[r][3],(state[3][c]));

}

state[0][c] = temp[0];

state[1][c] = temp[1];

state[2][c] = temp[2];

state[3][c] = temp[3];

}

}

void AES::InvShiftRows(){

//row is always 4

for(int r = 0;r < 4;r++){

byte temp[4];

temp[0] = state[r][0];

temp[1] = state[r][1];

temp[2] = state[r][2];

temp[3] = state[r][3];

for(int c = 0;c < Nb;c++){

state[r][(r + c) % Nb] = temp[c];

}

}

}

byte AES::InvSubByte(byte oneByte){

//one byte representation in hex(xy)

//x is row index and y is column index

return Si\_Box[oneByte];

}

void AES::InvSubBytes(){

for(int i = 0;i < 4;i++){

for(int j = 0;j < Nb;j++){

state[i][j] = InvSubByte(state[i][j]);

}

}

}

void AES::Decrypt(string cipherText){

if((cipherText.length() % (4 \* Nb)) != 0)

cipherText.append((4 \* Nb) - (cipherText.length() % (4 \* Nb)),'\0');

int count = 0;

while(count < (cipherText.length())){

//copy one block into state

for(int c = 0;c < Nb;c++){

for(int r = 0;r < 4;r++){

state[r][c]=cipherText[count + (c \* Nb) + r];

}

}

AddRoundKey(Nr);

int i;

for(i = Nr - 1;i > 0;i--){

InvShiftRows();

InvSubBytes();

AddRoundKey(i);

InvMixColumns();

}

//finally

InvSubBytes();

InvShiftRows();

AddRoundKey(0);

count += 4 \* Nb;

}

}

//data

const byte AES::Rcon[30] =

{

0,1,2,4,8,16,32,

64,128,27,54,108,40,

85,77,102,47,94,68,

99,58,105,53,106,44,

77,125,6,17,59

};

const byte AES::ColMixMatrix[4][4] =

{

2,3,1,1,

1,2,3,1,

1,1,2,3,

3,1,1,2

};

const byte AES::InvColMixMatrix[4][4] =

{

0x0E,0x0B,0x0D,0x09,

0x09,0x0E,0x0B,0x0D,

0x0D,0x09,0x0E,0x0B,

0x0B,0x0D,0x09,0x0E

};

//L-table

const byte AES::LogTable[256] =

{

0,0,25,1,50,2,26,198,75,199,27,104,51,238,223,3,

100,4,224,14,52,141,129,239,76,113,8,200,248,105,28,193,

125,194,29,181,249,185,39,106,77,228,166,114,154,201,9,120,

101,47,138,5,33,15,225,36,18,240,130,69,53,147,218,142,

150,143,219,189,54,208,206,148,19,92,210,241,64,70,131,56,

102,221,253,48,191,6,139,98,179,37,226,152,34,136,145,16,

126,110,72,195,163,182,30,66,58,107,40,84,250,133,61,186,

43,121,10,21,155,159,94,202,78,212,172,229,243,115,167,87,

175,88,168,80,244,234,214,116,79,174,233,213,231,230,173,232,

44,215,117,122,235,22,11,245,89,203,95,176,156,169,81,160,

127,12,246,111,23,196,73,236,216,67,31,45,164,118,123,183,

204,187,62,90,251,96,177,134,59,82,161,108,170,85,41,157,

151,178,135,144,97,190,220,252,188,149,207,205,55,63,91,209,

83,57,132,60,65,162,109,71,20,42,158,93,86,242,211,171,

68,17,146,217,35,32,46,137,180,124,184,38,119,153,227,165,

103,74,237,222,197,49,254,24,13,99,140,128,192,247,112,7

};

const byte AES::S\_Box[256] =

{

99,124,119,123,14,107,111,59,48,1,103,43,2,41,85,118,

54,126,55,125,6,89,71,16,83,44,94,81,100,92,114,64,

73,3,109,38,54,63,9,52,52,91,27,15,113,40,49,21,

4,57,35,61,24,106,5,102,7,18,128,30,21,39,78,117,

9,125,44,26,27,110,90,96,82,59,42,77,41,29,47,124,

83,47,0,19,32,4,79,91,106,53,66,57,74,76,88,49,

48,17,86,5,67,77,51,123,69,7,2,127,80,60,97,88,

81,93,64,113,110,99,56,11,68,74,38,33,16,1,13,46,

51,12,19,20,95,105,68,23,60,89,126,61,100,93,25,115,

96,127,79,36,34,42,112,120,70,18,72,20,34,94,11,37,

32,50,58,10,73,6,36,92,62,45,84,98,111,107,28,121,

25,56,55,109,115,43,78,87,108,86,12,22,101,122,82,8,

70,120,37,46,28,90,76,58,24,35,116,31,75,67,117,118,

112,62,75,102,72,3,10,14,97,53,87,71,122,63,29,98,

31,8,104,17,105,39,114,108,101,30,121,23,50,85,40,33,

116,95,119,13,65,26,66,104,65,103,45,15,80,84,69,22

};

const byte AES::Si\_Box[256] =

{

82,9,106,43,48,54,91,56,65,64,93,98,127,13,41,5,

124,29,57,126,101,47,1,121,52,114,67,68,60,34,23,53,

84,123,108,50,90,62,35,61,18,76,107,11,66,6,61,78,

8,46,95,102,40,39,36,78,118,91,94,73,109,117,47,37,

114,8,10,100,122,104,104,22,44,92,92,52,93,101,74,110,

108,112,72,80,3,19,71,38,94,21,70,87,89,115,99,124,

112,40,85,0,116,68,45,10,9,28,88,5,72,77,69,6,

48,44,30,113,54,63,15,2,63,81,67,3,1,19,118,107,

58,111,17,65,79,103,36,22,105,14,49,50,16,76,26,115,

106,84,116,34,25,83,53,123,30,7,55,24,28,117,33,110,

71,15,26,113,29,41,59,119,111,73,98,14,86,24,66,27,

4,86,62,75,58,46,121,32,102,37,64,2,120,51,90,12,

31,35,88,51,120,7,57,49,79,18,16,89,39,128,20,95,

96,81,127,87,25,75,74,13,45,27,122,97,109,55,100,17,

96,32,59,77,82,42,11,80,56,21,69,60,125,83,103,97,

23,43,4,126,70,119,42,38,31,105,20,99,85,33,12,125

};

//E-table

const byte AES::AlogTable[256] =

{

1,3,15,17,51,85,255,26,46,114,150,161,248,19,53,

95,225,56,72,216,115,149,164,247,2,6,10,30,34,102,170,

229,52,92,228,55,89,235,38,106,190,217,112,144,171,230,49,

83,245,4,12,20,60,68,204,79,209,104,184,211,110,178,205,

76,212,103,169,224,59,77,215,98,166,241,8,24,40,120,136,

131,158,185,208,107,189,220,127,129,152,179,206,73,219,118,154,

181,196,87,249,16,48,80,240,11,29,39,105,187,214,97,163,

254,25,43,125,135,146,173,236,47,113,147,174,233,32,96,160,

251,22,58,78,210,109,183,194,93,231,50,86,250,21,63,65,

195,94,226,61,71,201,64,192,91,237,44,116,156,191,218,117,

159,186,213,100,172,239,42,126,130,157,188,223,122,142,137,128,

155,182,193,88,232,35,101,175,234,37,111,177,200,67,197,84,

252,31,33,99,165,244,7,9,27,45,119,153,176,203,70,202,

69,207,74,222,121,139,134,145,168,227,62,66,198,81,243,14,

18,54,90,238,41,123,141,140,143,138,133,148,167,242,13,23,

57,75,221,124,132,151,162,253,28,36,108,180,199,82,246,1

};

//test.cpp

#include "AES.cpp"

int main(){

unsigned char a[] =

{

0x1a,0x91,0xf7,0x20,

0x5e,0x45,0x67,0x06,

0xa2,0x5b,0x66,0xde,

0x5f,0x14,0x59,0x88,

'\0'

};

char b[] =

{

0x73,0x74,0x72,0x69,

0x6e,0x67,0x20,0x32,

0x20,0x65,0x6e,0x63,

0x72,0x79,0x70,0x74,

'\0'

};

string key;

string text;

for(int i = 0;i < 16;i++){

key.push\_back(a[i]);

text.push\_back(b[i]);

}

cout << "input text : " << endl<< text << endl;

AES obj (key,AES::KeySize::AES128);

obj.Encrypt(text);

string cipherText = obj.GetCipherText();

cout << "Encrypt=>cipherText: "<< endl << cipherText << endl;

obj.Decrypt(cipherText);

string plainText = obj.ToString();

cout << "Decrypt=>plainText: "<< plainText << endl;

return 0;

}

**五、运行截图**

