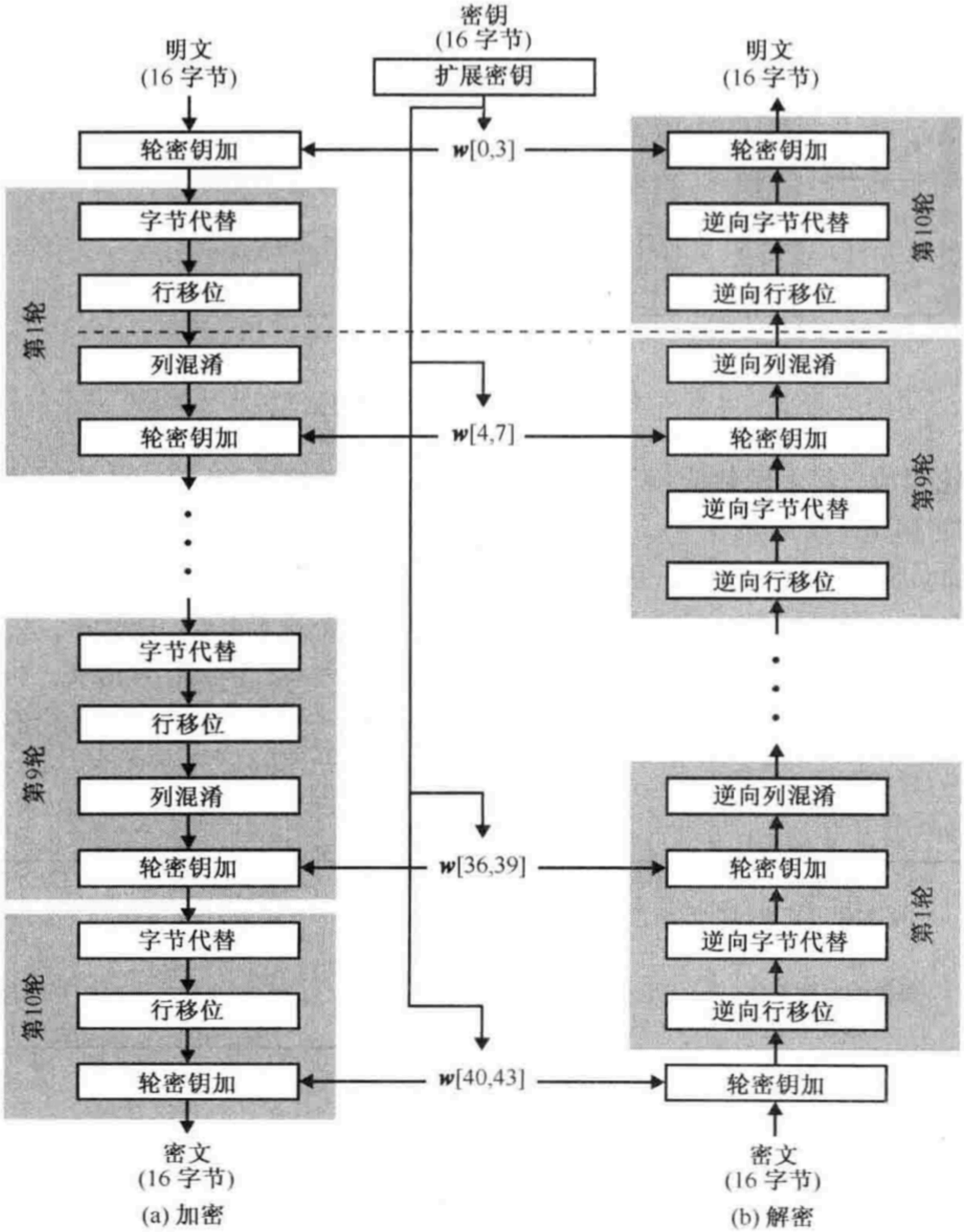
AES加密实验报告

姓名	学号	联系方式
张家豪	16337303	994328597@qq.com

一、实验内容

1. 完成AES的加密和解密
2. 制作GUI

二、算法流程



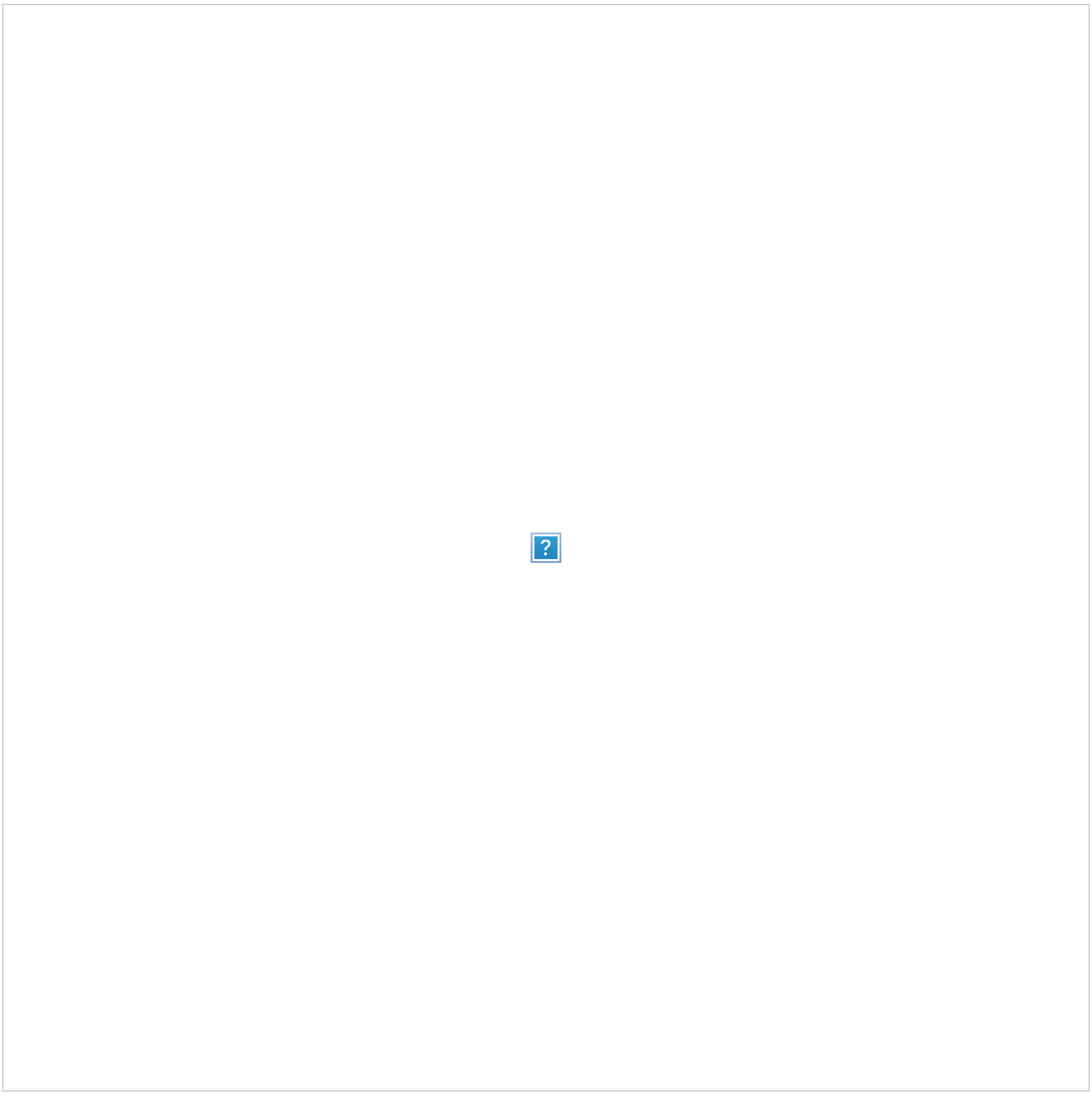
三、算法原理

AES算法属于对称加密算法，而且加密和解密的算法和密钥也是相同。

1、字节替代

AES的字节替代是一个置换过程，需要用到S盒，但是置换的原理和不同于DES的位置换，AES用到字节置换，字节替代的输入是16字节数据，对于每一个字节，字节的高4位作为行号，低4位作为列号，取出对应行列的字节替代原来的字节。以下是S盒

逆初始置换是初始置换的逆变换，通过逆置换可以获得原来的字节，逆置换S盒如下



2、行位移

将16字节输入表示成 4×4 的矩阵。行位移的过程：将第 i 行循环左移 $i-1$ 个字节。行位移变换的一个例子如下所示。

至于逆向行位移则是将第 i 行循环右移 $i-1$ 个字节。这样就能恢复到原来的矩阵。

3、列混淆变换

列混淆是基于GF(2^8)上的矩阵相乘，而不是整数域上的矩阵相乘，这一点是我开始时搞错的一点。列混淆计算过程如下图所示，在本次实验中，用于列混淆的矩阵是固定的。

对于以上的矩阵，我们可以推导出它在GF(2^8)上的逆矩阵，通过左乘该逆矩阵就能得到原来的字节矩阵。

4、轮密钥加变换

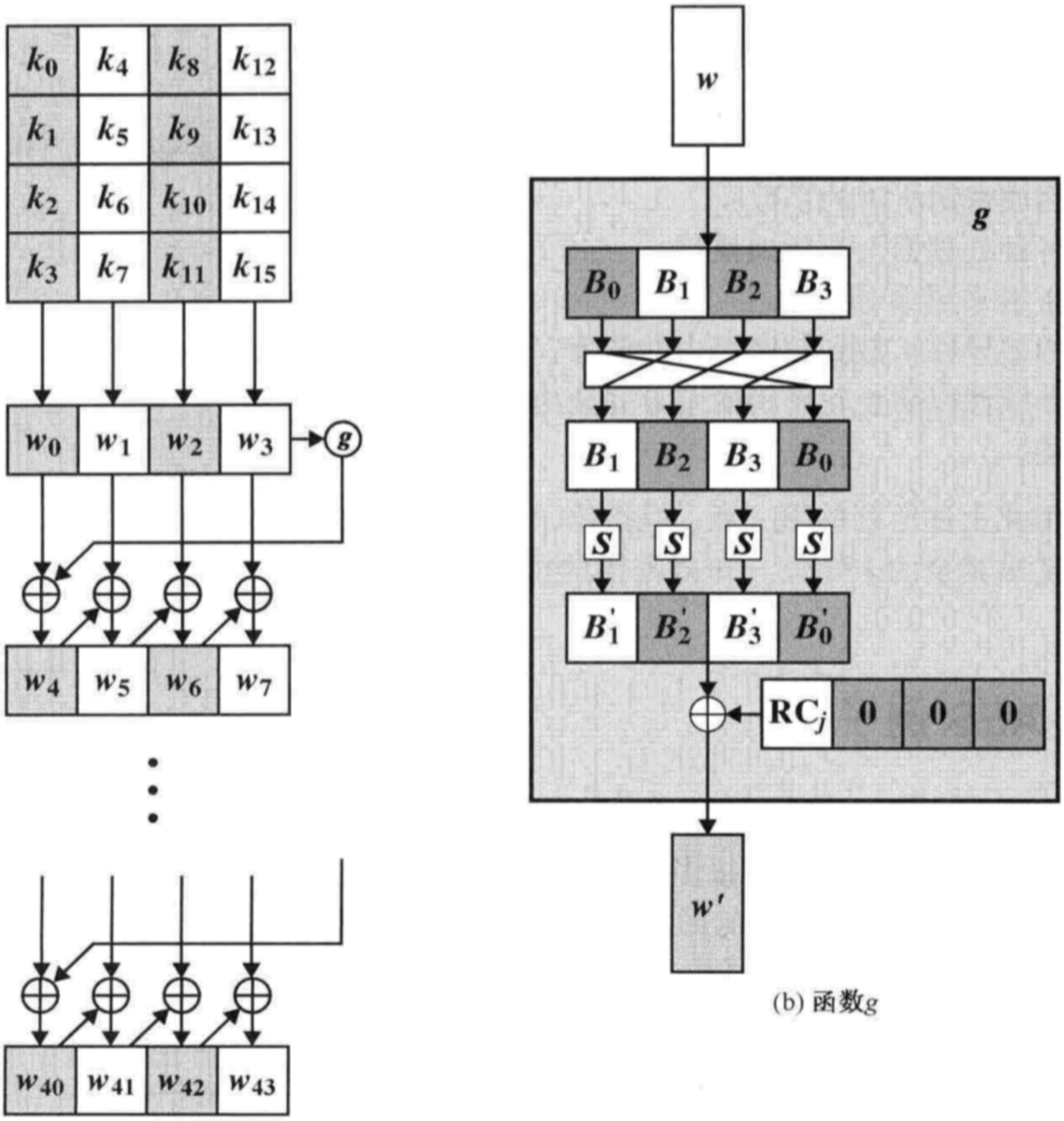
轮密钥加就是在每轮加密中将输入与轮密钥相异或即可。

5、密钥扩展算法

本次实验做的是AES128的加密，所以密钥是16字节的，我们要将16字节的密钥扩展成44字节，用于做11次轮密钥加变换。在密钥扩展过程中，除了前4字节是初始字节外（以0为初始字节），第 i 个字节与 第 $i-4$ 和 第 $i-1$ 个字节有关，

- 将初始密钥以列为主，转化为4字节的字，分别记为w[0...3]；
- 按照如下方式，依次求解w[j]，其中j是整数并且属于[4,43]；
- 若 $j \equiv 0(mod\ 4)$，则 $w[j] = w[j-4] \oplus g(w[j-1])$，否则 $w[j] = w[j-4] \oplus w[j-1]$；
- 函数g的流程说明：
    - 将 w 循环左移一个字节；
    - 分别对每个字节按S盒进行映射；
    - 与32 bits的常量（RC[j/4],0,0,0）进行异或，RC是一个一维数组，其值如下。（RC的值只需要有10个，而此处用了11个，实际上RC[0]在运算中没有用到，增加RC[0]是为了便于程序中用数组表示。由于j的最小取值是4，j/4的最小取值则是1，因此不会产生错误。）其中 RC = {00, 01, 02, 04, 08, 10, 20, 40, 80, 1B, 36}。

具体过程如下图所示


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w_{40} w_{41} w_{42} w_{43}
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B_1' B_2' B_3' B_0'
RC_j 0 0 0
w'

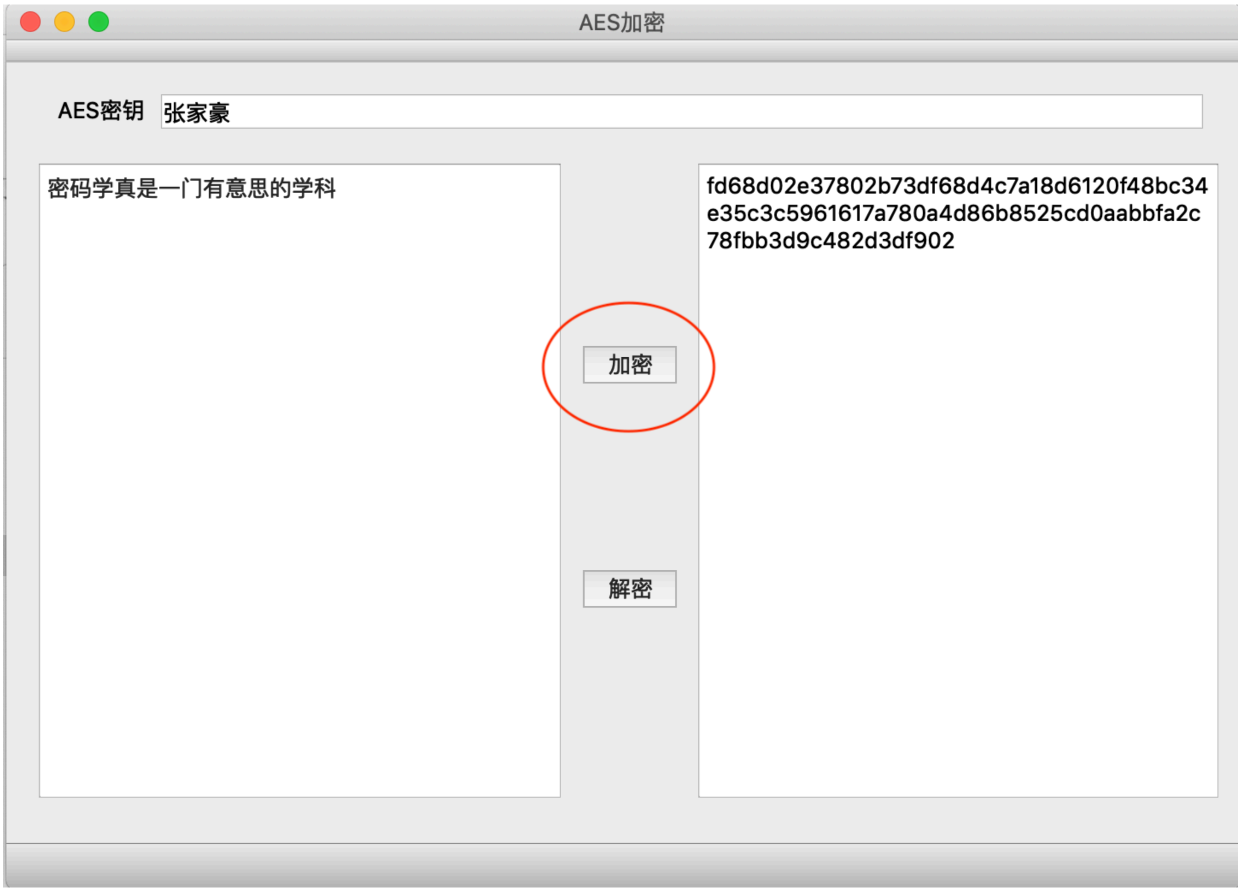
(a) 总体算法

(b) 函数g

四、实验结果

1、加密过程

- 密钥为张家豪
- 明文为**密码学真是一门有意思的学科**
- 加密密文为**fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902**



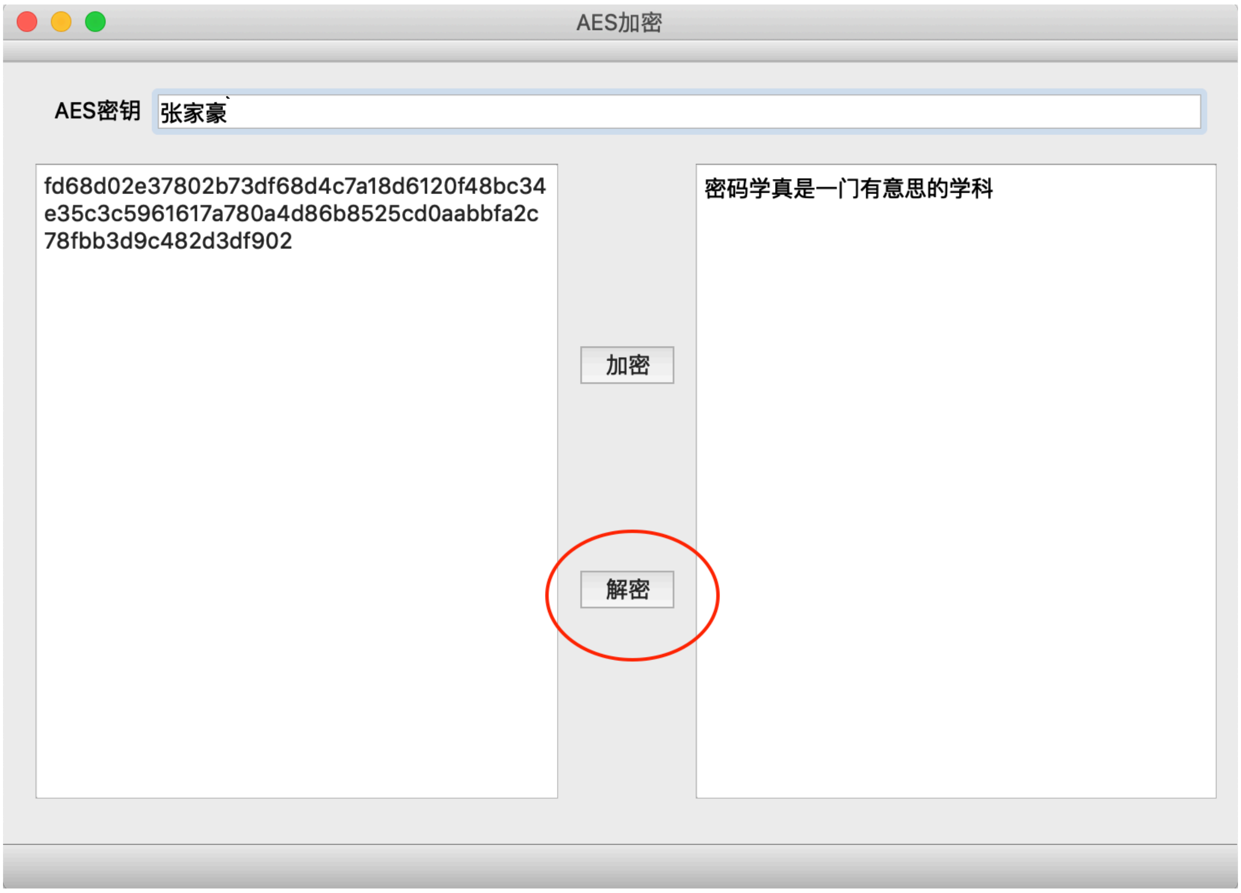
AES密钥 张家豪

密码学真是一门有意思的学科

fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902

2、解密过程

- 密钥为张家豪
- 密文为**fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902**
- 解密的明文为**密码学真是一门有意思的学科**



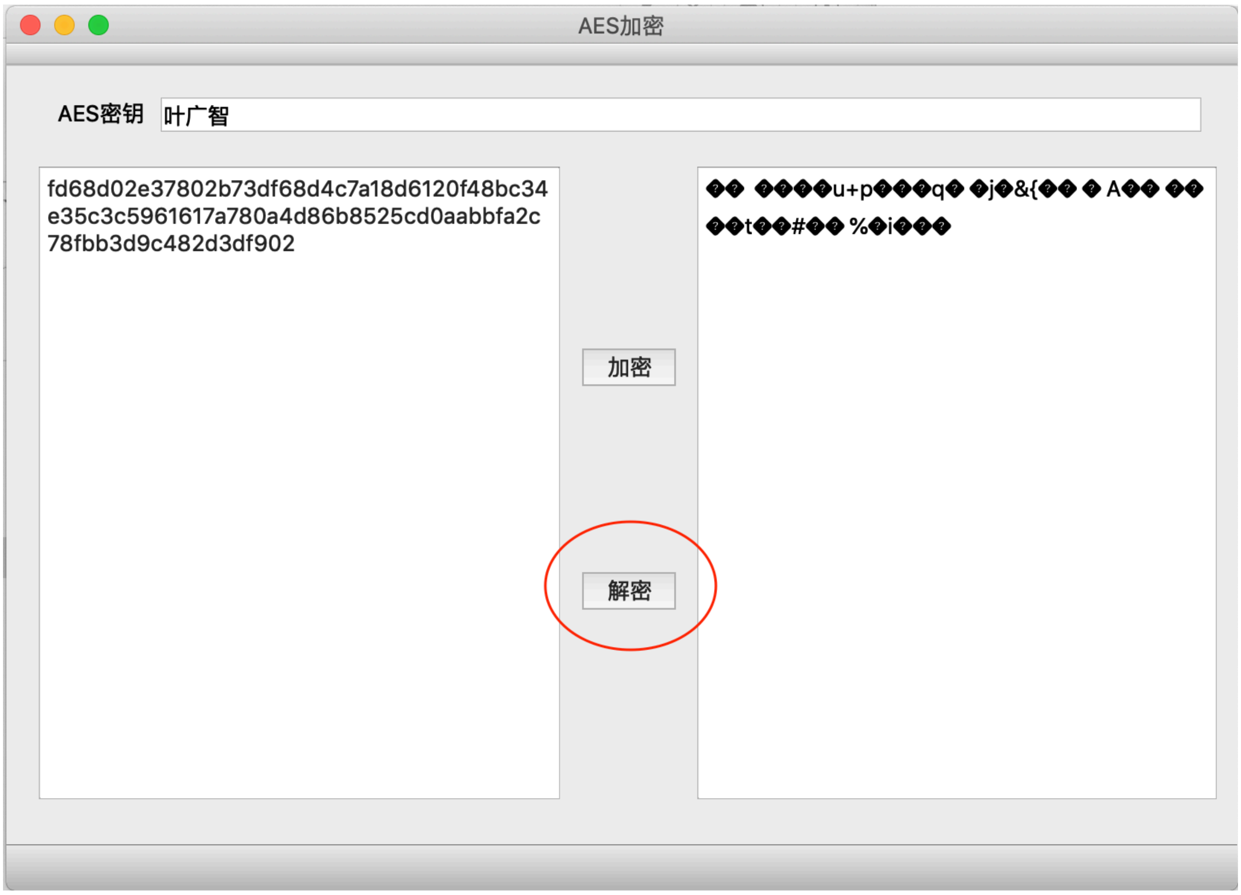
AES密钥 张家豪

fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902

密码学真是一门有意思的学科

3、错误解密过程

- 密钥为**叶广智**
- 密文为**fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902**
- 解密的明文为**������u+p���q��j�&{���A������t��#��%�i���**



AES密钥 叶广智

```
fd68d02e37802b73df68d4c7a18d6120f48bc34
e35c3c5961617a780a4d86b8525cd0aabbfa2c
78fbb3d9c482d3df902
```

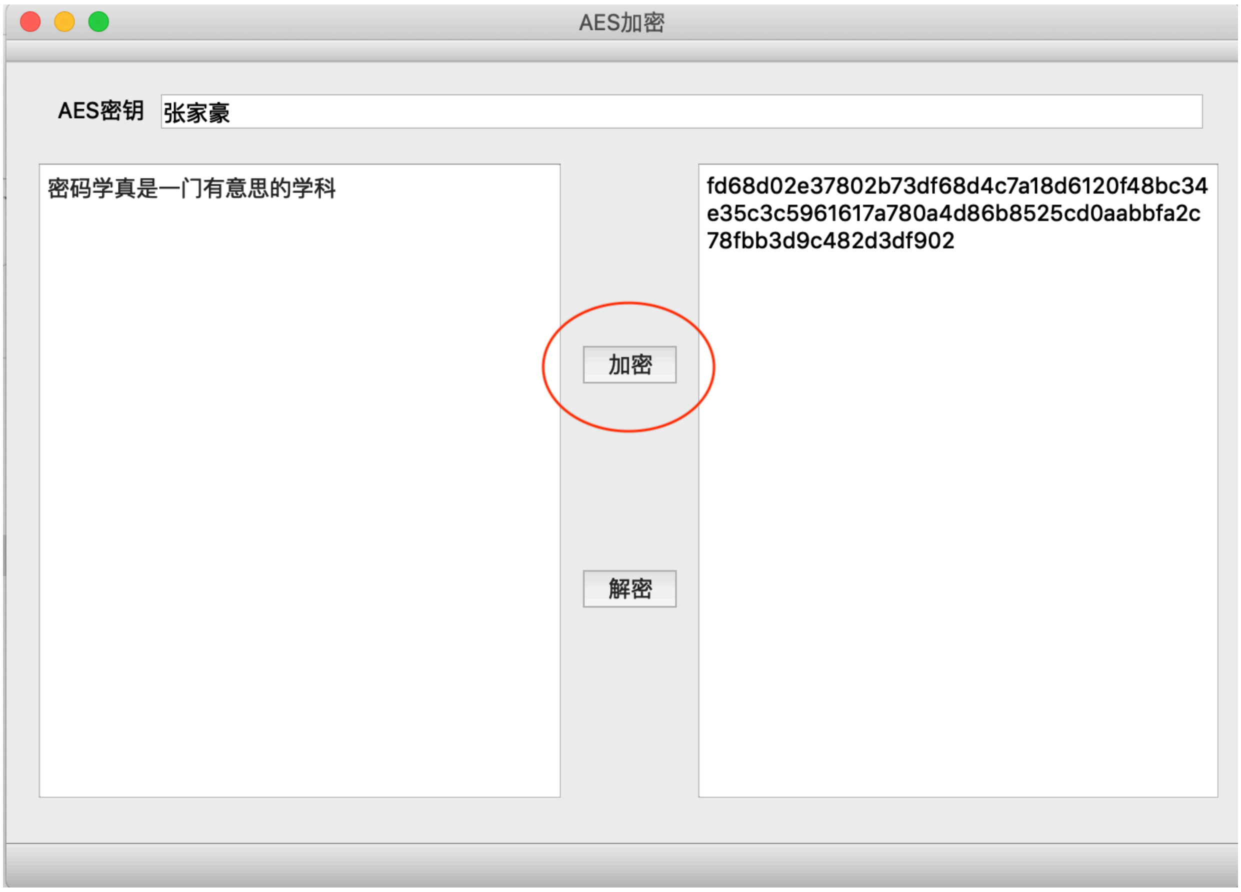
```
�� ����u+p���q� �j�&{�� � A�� ��
��t��#�� %�i���
```

可见如果没有用正确的密钥无法得出原来的明文。

4、雪崩效应

4.1 改变密钥

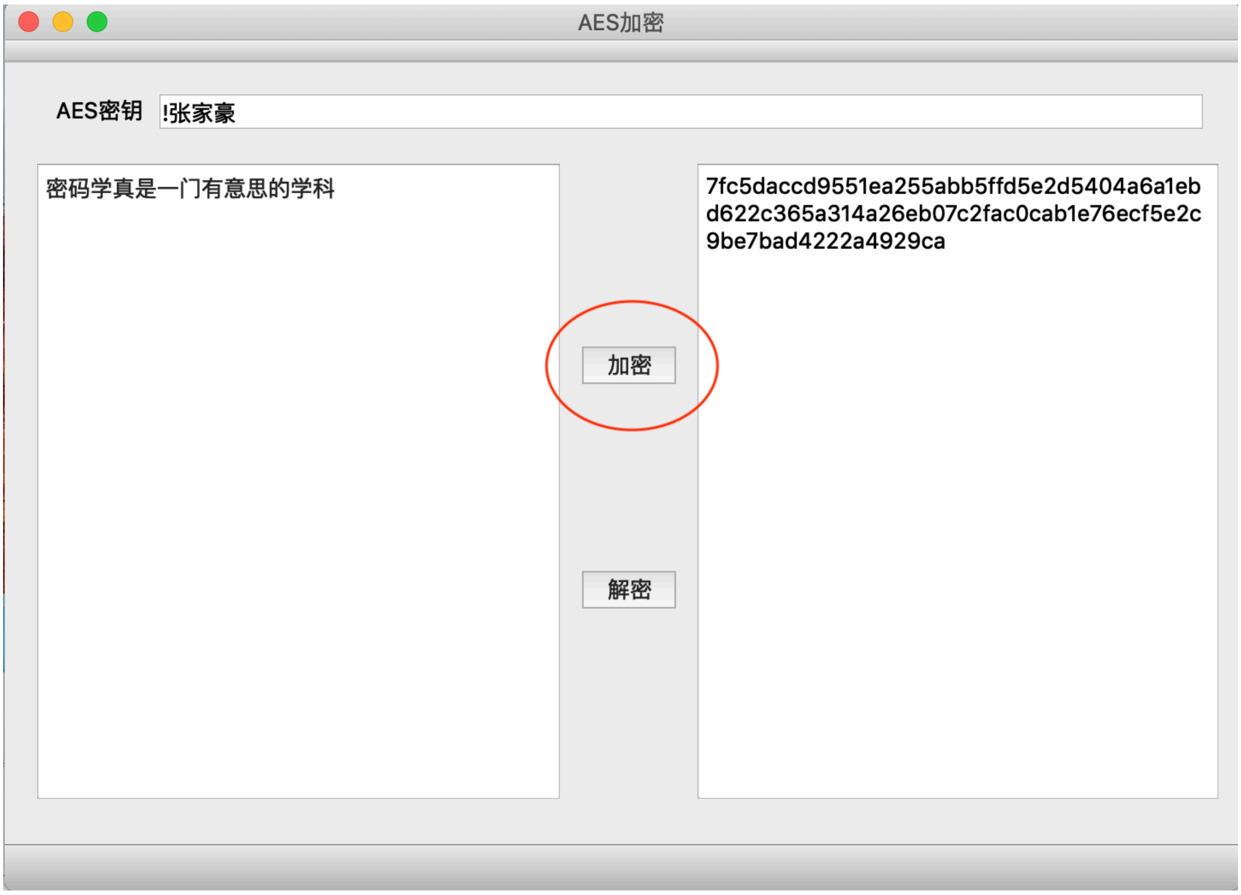
当密钥分别为 **张家豪** 和 **!张家豪** 时，加密后的密文具有很大的差异。



AES密钥 张家豪

密码学真是一门有意思的学科

```
fd68d02e37802b73df68d4c7a18d6120f48bc34
e35c3c5961617a780a4d86b8525cd0aabbfa2c
78fbb3d9c482d3df902
```

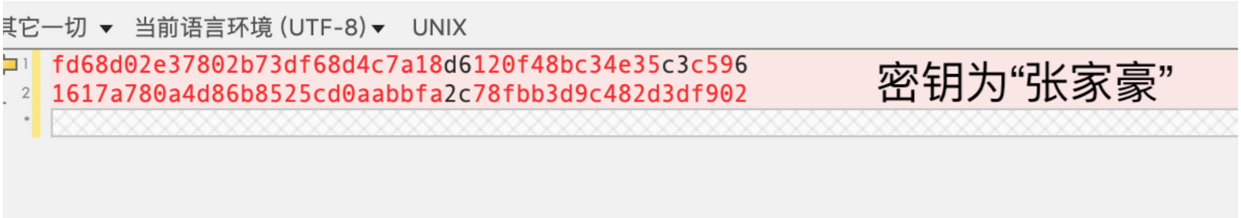


AES密钥 !张家豪

密码学真是一门有意思的学科

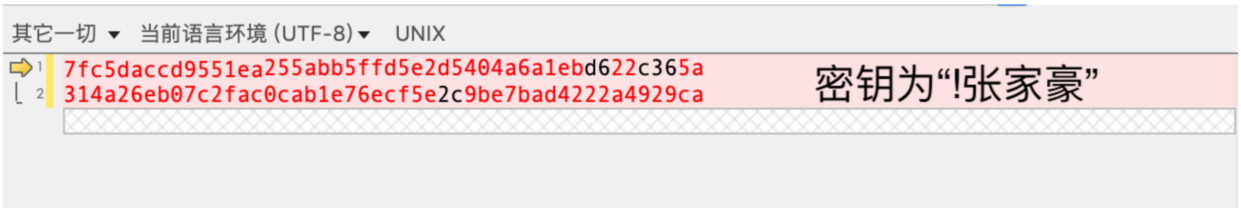
```
7fc5daccd9551ea255abb5ffd5e2d5404a6a1eb
d622c365a314a26eb07c2fac0cab1e76ecf5e2c
9be7bad4222a4929ca
```

diff结果：(红色为密文中不同的字段，黑色为相同的字段)



```
fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c596
1617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902
```

密钥为“张家豪”

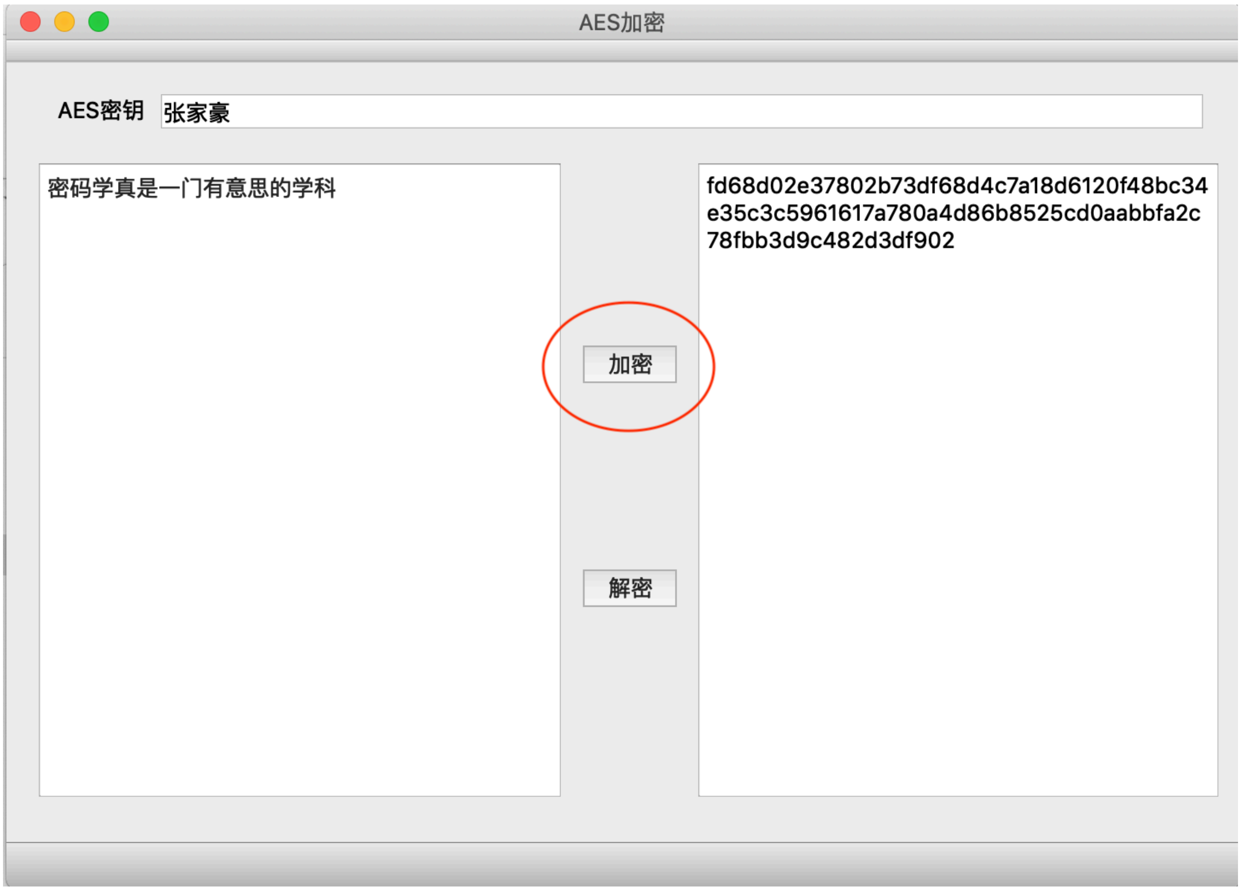


```
7fc5daccd9551ea255abb5ffd5e2d5404a6a1ebd622c365a
314a26eb07c2fac0cab1e76ecf5e2c9be7bad4222a4929ca
```

密钥为“!张家豪”

4.2 改变密文

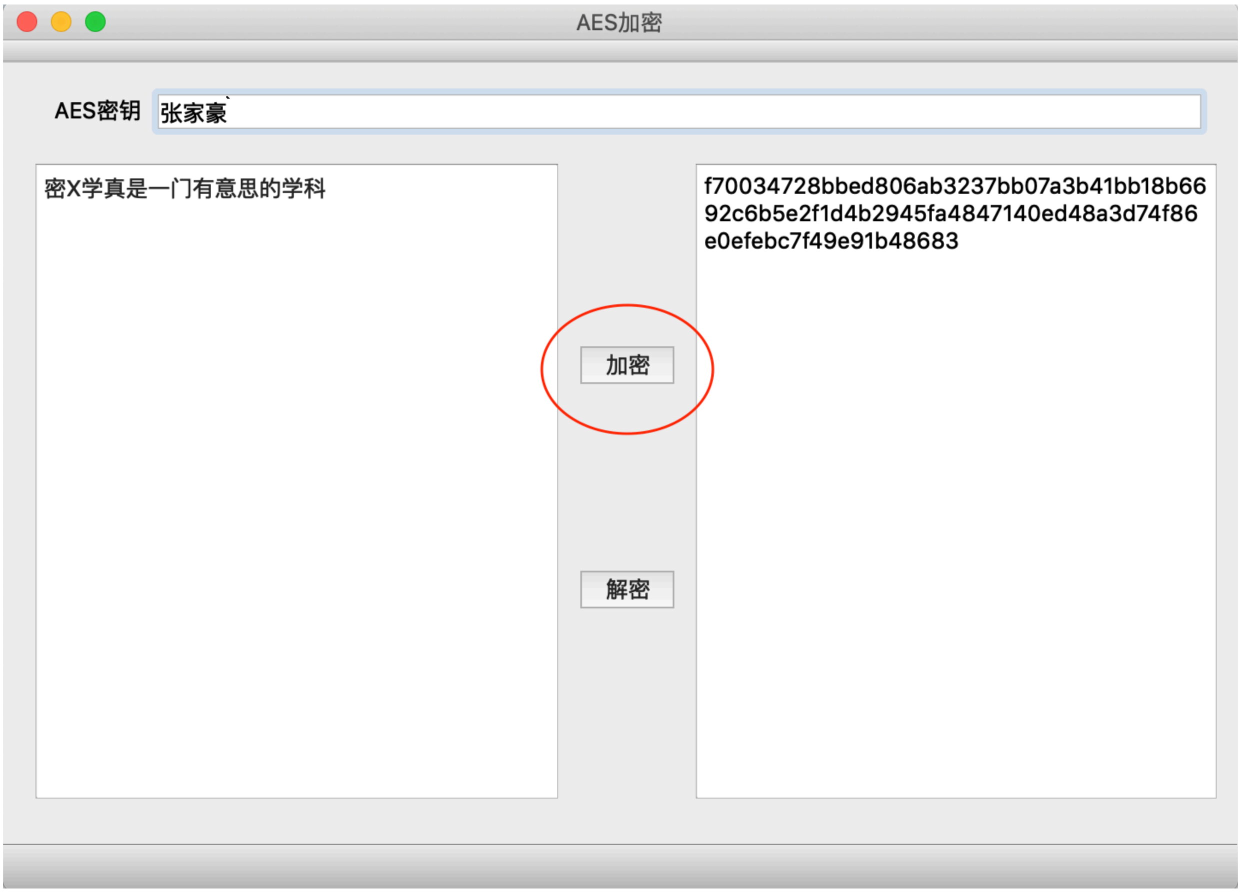
将明文中的“码”字换成“X”之后，密文几乎没有重复的字段。



AES密钥 张家豪

密码学真是一门有意思的学科

fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c5961617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902

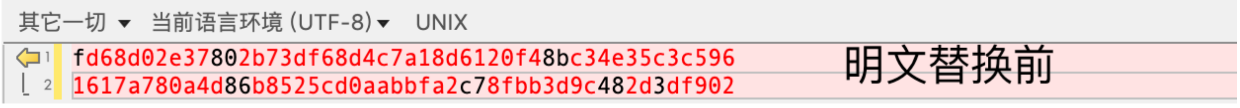


AES密钥 张家豪

密X学真是一门有意思的学科

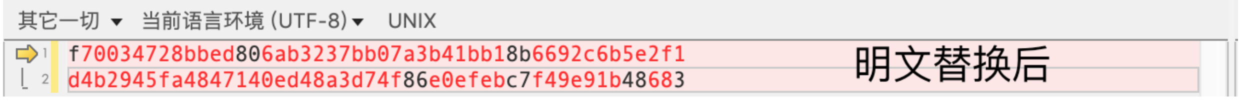
```
f70034728bbed806ab3237bb07a3b41bb18b66
92c6b5e2f1d4b2945fa4847140ed48a3d74f86
e0efebc7f49e91b48683
```

diff结果：(红色为密文中不同的字段，黑色为相同的字段)



```
fd68d02e37802b73df68d4c7a18d6120f48bc34e35c3c596
1617a780a4d86b8525cd0aabbfa2c78fbb3d9c482d3df902
```

明文替换前



```
f70034728bbed806ab3237bb07a3b41bb18b6692c6b5e2f1
d4b2945fa4847140ed48a3d74f86e0efebc7f49e91b48683
```

明文替换后

五、代码展示

1、GUI模块

1. mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QApplication>


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

2. mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "aes.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_2_clicked()//加密
{
    std::string key_ = (ui->lineEdit->text()).toStdString();
    std::string message_ = (ui->plainTextEdit->toPlainText()).toStdString();
    AES mes(message_, key_);
    QString cripher_ = QString::fromStdString(mes.encrypt());
    ui->textBrowser->setText(cripher_);
}

void MainWindow::on_pushButton_clicked()//解密
{
    std::string key_ = (ui->lineEdit->text()).toStdString();
    std::string cripher_ = (ui->plainTextEdit->toPlainText()).toStdString();
    AES cri(cripher_, key_);
    QString cripher = QString::fromStdString(cri.decrypt());
    ui->textBrowser->setText(cripher);
}
```

2、AES算法模块

1. aes.h(有些调试用的代码没有删除)

```
//
//  AES.h
//  AES
//
//  Created by 张家豪 on 2018/12/7.
//  Copyright © 2018 张家豪. All rights reserved.
//

#ifndef AES_H
#define AES_H
#include <iostream>
#include <vector>
#include <string>
#include "table.h"


typedef std::vector<unsigned> state_t;
typedef std::vector<state_t>  mes_t;

state_t string2state(std::string, int type);
state_t substitude(state_t);
state_t inv_substitude(state_t);
state_t row_shift(state_t);
state_t column_confuse(state_t);
state_t inv_shift(state_t);
state_t inv_confuse(state_t);
state_t operator ^(state_t, state_t);
std::string state2string(state_t, int type);
state_t encrypt(state_t, std::vector<state_t>);
state_t decrypt(state_t, std::vector<state_t>);


std::ostream& operator <<(std::ostream & out, state_t input){
    for(int i = 0, size = input.size(); i < size; i++){
        out << std::hex << input[i] << ' ';
        if((i+1) % 4 == 0)
            out << std::endl;
    }
    return out;
}

std::ostream& operator <<(std::ostream & out, mes_t input){
    for(int i = 0, size = input.size(); i < size; i++){
        out << input[i] << std::endl;
    }
    return out;
}

class AES{
private:
    std::string message;
    state_t key;

public:
    AES(std::string m, std::string k);
    std::string encrypt();
    std::string decrypt();
};

state_t gen_key(state_t key, int round){
    state_t res(16,0);
    for(int i = 0; i < 4; i++){
        res[i*4] = s_box[key[(i * 4+ 7)%16]] ^ key[i*4]; //第一列
        if(i == 0)
            res[0] = res[0] ^ RC[round];
    }
    for(int i = 1; i < 4; i++){
        for(int j = 0; j < 4; j++){//j是行数
            res[j*4 + i] = res[j*4 + i-1] ^ key[j*4 + i];
        }
    }
    return res;
}

std::vector<state_t> expand_key(state_t key){
    //    std::cout << key;
    std::vector<state_t> res;
    res.push_back(key);
    for(int i = 0; i < 10; i++){
        key = gen_key(key, i);
        //                std::cout << i+1 << "次： " << std::endl;
        //                std::cout << key << std::endl;
        res.push_back(key);
    }
    return res;
}
```

```

state_t substitude(state_t input){
    state_t res;
    for(int i = 0,size = input.size(); i < size; i++)
        res.push_back(s_box[input[i]]);
    return res;
}

state_t inv_substitude(state_t input){
    state_t res;
    for(int i = 0,size = input.size(); i < size; i++)
        res.push_back(inv_s_box[input[i]]);
    return res;
}

state_t row_shift(state_t input){
    state_t res;
    for(int i = 0,size = input.size(); i < size; i++){
        int row = i/4;
        res.push_back(input[row*4 + (i+row)%4]);
    }
    return res;
}

state_t inv_shift(state_t input){
    state_t res;
    for(int i = 0,size = input.size(); i < size; i++){
        int row = i/4;
        //        std::cout << i << " " << row << " " << row*4 + (i+row)%4 << " " << input[row*i + (i+row)%4] << std::endl;
        //
        res.push_back(input[row*4 + (i-row)%4]);
    }
    return res;
}

state_t column_confuse(state_t input){
    state_t res;
    for(int i = 0,size = 16; i < size; i++){
        unsigned sum = 0;
        int row = i / 4;
        int column = i % 4;
        for(int j = 0; j < 4; j++){
            if(mix_box[row * 4 + j] == 2 && input[4*j + column] >= 128)
                sum ^= (2*input[4*j + column]) ^ 0x1b;
            else if(mix_box[row * 4 + j] == 3){
                unsigned tmp = (2 *input[4*j + column]) % 256;
                if(input[4*j + column] >= 128)
```

```
                    tmp ^= 0x1b;
                sum ^= tmp ^ input[4*j + column];
            }
            else
                sum ^= (mix_box[row * 4 + j] * input[4*j + column]) % 256;
            //            sum ^= (mix_box[row * 4 + j] * input[4*j + column]) % 0x11b;
        }
        res.push_back(sum % 256);
    }
    return res;
}

unsigned mut(unsigned a, unsigned b){
    unsigned sum = 0;
    std::vector<unsigned> parts;
    parts.push_back(b);
    int t = 1;
    int i = 0;
    while(t*2 < a){
        unsigned tmp = parts[i++];
        if(tmp >= 128)
            parts.push_back( ((tmp<<1) ^ 0x1b) % 256);
        else
            parts.push_back((tmp << 1) % 256);
        t = t*2;
    }
    i = 0;
    while(a > 0){
        if(a&1){
            sum ^= parts[i];
        }
        a >>= 1;
        i++;
    }
    return sum;
}

state_t inv_confuse(state_t input){
    state_t res;
    for(int i = 0,size = 16; i < size; i++){
        unsigned sum = 0;
        int row = i / 4;
        int column = i % 4;
        for(int j = 0; j < 4; j++){
            sum ^= mut(inv_mix_box[row * 4 + j],input[4*j + column]);
        }
        res.push_back(sum % 256);
    }
    return res;
}

state_t operator ^(state_t a, state_t b){
    state_t res;
    for(int i = 0,size = a.size(); i < size; i++)
        res.push_back((a[i] ^ b[i]) % 256);
    return res;
}



state_t encrypt_(state_t mes, std::vector<state_t> keys){
    mes = mes ^ keys[0];
    for(int i = 1; i <= 9; i++){
        mes = substitude(mes);
        mes = row_shift(mes);
        mes = column_confuse(mes);
        mes = mes ^ keys[i];
    }
    mes = substitude(mes);
    mes = row_shift(mes);
    mes = mes ^ keys[10];
    return mes;
}

state_t decrypt_(state_t mes, std::vector<state_t> keys){
    mes = mes ^ keys[10];
    for(int i = 9; i > 0; i--){
        mes = inv_shift(mes);
        mes = inv_substitude(mes);
        mes = mes ^ keys[i];
        mes = inv_confuse(mes);
    }
    mes = inv_shift(mes);
    mes = inv_substitude(mes);
    mes = mes ^ keys[0];
    return mes;
}



char int2char(int input){
    if(input < 10)
        return input + '0';
    else
        return input-10+'a';
```

```
}

std::string bytes2string_c(state_t input){ //用于密文转换
    std::string res;
    for(int i = 0, size = input.size(); i < size; i++){
        //        std::cout << input[i] << " ";
        res += int2char((input[i] >> 4) % 16);
        res += int2char(input[i] % 16);
        //        std::cout << res << std::endl;
    }
    return res;
}

unsigned char2int(char input){
    unsigned res = 0;
    unsigned tmp = 1;
    for(int i = 0;i < 8; i++){
        if(input & 1 == 1)
            res += tmp;
        tmp <<= 1;
        input >>= 1;
    }
    return res;
}

std::vector<state_t> string2bytes(std::string mes){ //用于明文
    std::vector<state_t> res;
    for(int i = 0; i < 15; i++)
        mes += char(0);
    //    mes += "               ";//补长
    for(int i = 0, size = mes.size(); i < size - 16; i = i + 16){
        state_t tmp(16,0);
        for(int j = 0; j < 16; j++){
            tmp[j] = char2int(mes[i+j]);
        }
        res.push_back(tmp);
    }
    return res;
}

unsigned hex2bytes(char top, char low){
    int a = (top <= '9' && top >= '0')? top - '0' : top - 'a' + 10;
    int b = (low <= '9' && low >= '0')? low - '0' : low - 'a' + 10;
    return (a << 4) + b;
}

std::vector<state_t> string2bytes_c(std::string mes){ //密文转换
```

```
    std::vector<state_t> res;
    for(int i = 0, size = mes.size(); i < size; i = i + 32){
        state_t tmp(16,0);
        for(int j = 0; j < 32; j = j + 2){
            tmp[j/2] = hex2bytes(mes[i+j], mes[i+j+1]);
        }
        res.push_back(tmp);
    }
    return res;
}

std::string bytes2string(state_t input){
    std::string res;
    for(int i = 0, size = input.size(); i < size; i++)
        res += input[i];
    return res;
}

AES::AES(std::string mes, std::string k){//key 128bits
    key = string2bytes(k)[0];
    message = mes;
}

std::string AES::encrypt(){
    std::vector<state_t> mes = string2bytes(message);
    std::string cripher;
    mes_t keys = expand_key(key);
    for(int i = 0, size = mes.size(); i < size; i++){
        cripher += bytes2string_c(encrypt_(mes[i], keys));
        //        std::cout << cripher << std::endl;
    }
    return cripher;
}

std::string AES::decrypt(){
    std::vector<state_t> mes = string2bytes_c(message);
    std::string res;
    mes_t keys = expand_key(key);
    for(int i = 0, size = mes.size(); i < size; i++){
        res += bytes2string(decrypt_(mes[i], keys));
    }
    return res;
}
#endif // AES_H
```

3、置换表

1. table.h

```
//
//  table.h
//  table
//
//  Created by 张家豪 on 2018/12/7.
//  Copyright © 2018 张家豪. All rights reserved.
//

#ifndef TABLE_H
#define TABLE_H
static unsigned s_box[256] = {
    // 0     1     2     3     4     5     6     7     8     9     a     b     c     d     e     f
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, // 0
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, // 1
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, // 2
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, // 3
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, // 4
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, // 5
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, // 6
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, // 7
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, // 8
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, // 9
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, // a
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, // b
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, // c
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, // d
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, // e
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16};// f


static unsigned inv_s_box[256] = {
    // 0      1      2      3      4      5      6      7      8      9      a      b      c      d      e      f
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, // 0
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, // 1
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e, // 2
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, // 3
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, // 4
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84, // 5
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, // 6
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, // 7
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, // 8
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, // 9
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, // a
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4, // b
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, // c
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, // d
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61, // e
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d};// f

static unsigned mix_box[16] = {0x02, 0x03, 0x01, 0x01,
                               0x01, 0x02, 0x03, 0x01,
                               0x01, 0x01, 0x02, 0x03,
                               0x03, 0x01, 0x01, 0x02
};

static unsigned inv_mix_box[16] = {0x0e, 0x0b, 0x0d, 0x09,
                                   0x09, 0x0e, 0x0b, 0x0d,
                                   0x0d, 0x09, 0x0e, 0x0b,
                                   0x0b, 0x0d, 0x09, 0x0e
};

static unsigned RC[10] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};
```

```

#endif // TABLE_H
```