AES算法的实现

算法背景（时俊华）

1.发展过程

高级加密标准（Advanced Encryption Standard，AES），又称Rijndael加密法，是目前应用最广泛的对称加密算法之一。AES算法背后的历史可以追溯到20世纪70年代，当时IBM和美国国家标准技术研究所（NIST）共同研究了一种名为Lucifer的加密算法。Lucifer是一种基于分组密码的加密算法，它的分组长度为64位，密钥长度为128位。随着计算机技术的发展，这种加密算法逐渐被认为是不够安全的。

在1997年，比利时密码学家Joan Daemen和Vincent Rijmen共同设计了一种新的分组密码算法——Rijndael。这种算法支持多种分组和密钥长度，从而可以灵活地应用于不同的安全需求。随后，这种算法被提名为下一代加密标准，最终在2001年被选为AES加密算法。AES算法使用128位、192位或256位密钥，分组长度固定为128位。

2.原理介绍

AES算法的安全性基于其设计的核心原则——替代-置换网络（Substitution-Permutation Network，SPN）。SPN是一种常见的分组密码结构，通过交替执行替代（Substitution）和置换（Permutation）操作，实现对明文的加密和解密。AES算法使用了四个主要的操作——字节代替（SubBytes）、行移位（ShiftRows）、列混淆（MixColumns）和轮密钥加（AddRoundKey）。其中，字节代替操作通过将每个输入字节替换为另一个预定义的字节来实现替代操作；行移位操作将每一行移位一定的位置，使得每个字节在不同的列上进行混淆；列混淆操作则通过将每一列看作一个多项式，并乘以一个预定义的矩阵，实现置换操作。轮密钥加操作是一个简单的异或操作，用于将当前轮的密钥与当前轮的输出进行混合，从而增强安全性。

3.优点与不足

AES算法的安全性经过了广泛的测试和验证，并且已被广泛应用于各种领域，如金融、通信、网络安全等。由于其高效性、可靠性和安全性，AES算法已成为当前应用最广泛的对称加密算法之一。

除了其高效性、可靠性和安全性，AES算法还有一些其他优点。首先，AES算法的实现非常灵活，可以根据具体的应用需求选择不同的分组长度和密钥长度，从而实现不同的安全级别。其次，AES算法的加解密速度非常快，在现代计算机上可以实现高速加解密操作，从而适用于需要快速加密和解密的场景。此外，AES算法的安全性是公认的，它已经通过了广泛的安全性测试和验证，可以保证在目前的计算机技术条件下不被破解。

然而，AES算法也存在一些不足之处。例如，由于其采用的是对称加密算法，因此需要双方都知道密钥，而密钥的传递和管理可能会面临一些挑战。此外，由于AES算法的密钥长度较长，因此其加密和解密过程需要消耗较多的计算资源和存储资源，这可能会影响一些资源受限的设备和应用场景。

4.未来展望

虽然AES算法已经成为目前应用最广泛的对称加密算法之一，但是在未来，随着计算机技术的不断发展和加密需求的不断变化，AES算法也需要不断改进和发展：

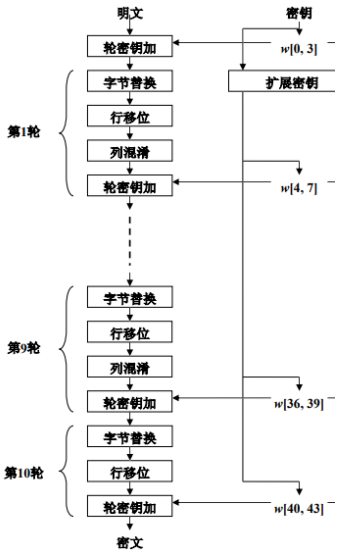
1. 提高安全性：随着计算机技术的发展，破解AES算法的难度可能会降低。因此，未来的AES算法需要进一步提高安全性，例如增加密钥长度、改进算法结构等。
2. 降低资源消耗：AES算法的加密和解密过程需要消耗较多的计算资源和存储资源，这可能会影响一些资源受限的设备和应用场景。因此，未来的AES算法需要考虑如何在保证安全性的前提下降低资源消耗。
3. 支持更多的应用场景：当前的AES算法主要适用于数据加密和保护，但未来可能需要更多的加密场景，例如物联网、人工智能等领域。因此，未来的AES算法需要支持更多的应用场景，并提供更灵活的加密方式。
4. 结合其他加密技术：AES算法可以结合其他加密技术，例如哈希函数、数字签名、公钥加密算法等，来实现更高的安全性和更灵活的加密方式。因此，未来的AES算法需要与其他加密技术进行结合，以提供更全面的加密解决方案。

算法流程（肖虹毅）

AES算法是一种高级加密标准算法，被广泛应用于网络通信、数据安全存储等领域。算法密钥长度可变，可分别为128位、192位和256位，数据长度位不一样，加密轮数也不一样，128位的是10轮，192位的是12轮，256位的是14轮。本文以128位密钥长度的AES算法为例进行讲解。算法流程如下：

1.加密流程

流程如图



1.密钥扩展

AES算法的密钥扩展是将初始密钥转化为更多的轮密钥，用于后续的运算。

首先，需要将初始密钥分为16个字节，然后利用S-盒对每个字节进行替代，再进行行移位和列混合操作。之后，根据轮常量和轮密钥周期性地对每个字节进行异或运算，直至得到所需的轮密钥。

扩展密钥是以4字节字为元素的一维阵列，表示为 $W[Nb*(Nr+1)]$ ，其中前Nk个字取为种子密钥，以后每个字按递归方式 定义。扩展算法根据Nk≤6和Nk>6有所不同，其区别在于：当i-4为Nk的整数倍时，须先将前一个字 W[i-1]经过S盒

2.字节替换

字节替代操作是将每个字节替换成S盒中对应的值，S盒是一个固定的256字节大小的表格，每个字节都有一个对应的8位输出值。

字节替代是非线性代换，独立地对状态的每个字节。S盒式可逆的，其算法步骤为：

1）将字节作为GF(2^8)上的元素映射到自己的逆元

2）将字节做GF(2)上的仿射变换

即

$$y = Ax^{-1} + B$$

其中A为GF（2）上的一个8*8可逆矩阵，B是GF（2）上一个八位列向量。

3.行移位

行移位操作是将矩阵的每行字节进行循环左移操作。

第0行不变，

第1行向左移动1位，

第三行向左移动2位，

第四行向左移动3位。

将状态阵列的各行进行循环移位，不同行的移位量不同

0行：不动

1行：循环左移C1字节

2行：循环左移C2字节

3行：循环左移C3字节

记为：ShiftRow(State)

N_b	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

4.列混淆

列混合操作是将矩阵的每一列进行混合操作，混合操作通常使用特定的固定矩阵进行计算。

将每列视为GF（2^8）上的多项式，与固定的多项式$c(x)$进行模x^4+1的乘法，$c(x)$模x^4+1可逆。

写为矩阵乘法为:

$$b(x) = c(x) \otimes a(x)$$

$$c(x) = '03'x^3 + '01'x^2 + '01'x + '02'$$

矩阵形式

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

5.轮密钥加

轮密钥加操作是将每个字节与轮密钥进行逐比特异或运算

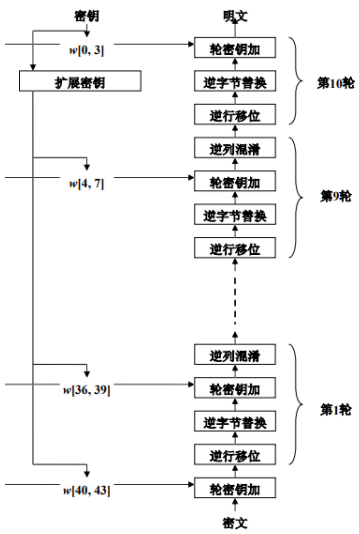
轮密钥长度与分组长度相同

经过九轮字节替换、行移位、列混淆、轮密钥加运算后，

第十轮进行字节替换、行移位、轮密钥加，最终得到密文

2.解密过程

解密过程与加密过程顺序相反，流程如下



1.逆行移位

逆行移位操作是将矩阵的每行字节进行循环右移操作。

第0行不变，

第1行向右移动1位，

第三行向右移动2位，

第四行向右移动3位。

2.逆字节替换

逆字节替代变换是字节替代变换的逆变换，在状态的每个字节上应用逆S盒

这是通过应用字节替代变换中的仿射变换的逆变换，再对所得结果应用 有限域的乘法逆运算得到即：

$$y = (A^{-1}(x - B))^{-1}$$

3.轮密钥加

与加密过程的流程相同，轮密钥加操作是将每个字节与轮密钥进行逐比特异或运算

轮密钥长度与分组长度相同

4.逆列混淆

逆列混淆变换是列混淆变换的逆，将状态矩阵中的每一列视为系数在 上的次数小于4的 多项式与同一个固定的多项式 $d(x)$ 相乘

$$\boldsymbol{d(x)=‘0B’x^3+‘0D’x^2+‘09’x+‘0E’}$$

$$b(x) = d(x) \otimes a(x)$$

矩阵形式

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_2 \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

密文也需要先进行九轮逆行移位、逆字节替换、轮密钥加、逆列混淆操作，

第十轮进行逆行移位、逆字节替换、轮密钥加操作，以此得到明文。

安全分析（周开轩）

AES作为对称加密算法，其算法设计采用了高度复杂的数学运算和结构，如有限域、置换、代换等，使得算法在加密过程中具有高度的混淆性和扩散性，从而增加了攻击者破解密文的难度。本文将从AES算法的安全性、攻击方式、防护措施等方面进行详细的分析。

一、AES算法的安全性

1.1 比较安全的密钥长度

AES算法的密钥长度分别为128位、192位和256位，其中256位密钥长度的算法安全性最高。由于AES算法的密钥长度越长，破解难度就越大，因此在实际应用中，建议使用更长的密钥长度来提高安全性。

1.2 复杂的算法结构

AES算法采用了多轮迭代结构，每轮包含四个步骤：字节代换、行移位、列混淆和密钥加。这种结构可以增加算法的安全性，同时也使得算法的加密速度变慢。AES算法的算法设计非常复杂，包含了多种数学运算和结构，这些都为算法的安全性提供了很好的保障。

1.3 抗击各种攻击方式

AES算法在加密过程中能够抗击各种攻击方式，如差分攻击、线性攻击、密文攻击等。差分攻击是一种针对分组密码的攻击方式，攻击者通过对明文和密文之间的差异进行分析，来获取密钥信息。线性攻击是一种基于线性逼近的攻击方式，攻击者利用密文和明文的线性关系来获取密钥信息。而密文攻击是一种通过分析密文来获得密钥信息的攻击方式。AES算法的设计可以有效的抵御这些攻击方式，从而保证算法的安全性。

二、AES算法的攻击方式

2.1 差分攻击

差分攻击是一种针对分组密码的攻击方式，攻击者通过对明文和密文之间的差异进行分析，来获取密钥信息。差分攻击的基本思想是构造一些差分（即两个明文之间的差异）和密文之间的关系，然后通过统计分析来推断密钥信息。对于AES算法，攻击者可以构造一些差分，然后利用这些差分来推断出密钥信息。

2.2 线性攻击

线性攻击是一种基于线性逼近的攻击方式，攻击者利用密文和明文的线性关系来获取密钥信息。线性攻击的基本思想是构造一些线性逼近，然后通过统计分析来推断密钥信息。对于AES算法，攻击者可以构造一些线性逼近，然后利用这些线性逼近来推断出密钥信息。

2.3 密文攻击

密文攻击是一种通过分析密文来获得密钥信息的攻击方式。对于AES算法，攻击者可以通过分析密文中的统计特性，来推断出密钥信息。例如，攻击者可以通过分析密文中的频率分布、自相关性等特

性，来推断出密钥信息。

三、AES算法的防护措施

3.1 物理屏蔽技术

物理屏蔽技术是一种通过物理隔离来保护AES算法的技术。例如，可以在加密设备周围加入金属屏蔽罩，来防止攻击者通过电磁波等方式获取AES算法中的密钥信息。

3.2 增加掩码

掩码技术是一种通过将明文或密钥与随机数进行异或运算，来混淆算法中的数据，从而防止攻击者获得明文或密钥相关的数据。增加掩码

实现过程（何优 、丁祎）

加密

AES加密函数中，首先进行密钥拓展，然后将128位长度的字符串读入4*4数组中作为状态矩阵，之后每轮都对数组进行修改即对状态矩阵进行混淆（字节代换、行移位、列混合、轮密钥加），执行完加密后最后把数组转回字符串，加密完成。

```
// AES加密
void aes(char *p, int plen, char *key){

    if(plen == 0 || plen % 16 != 0) {
        printf("明文字符长度必须为16的倍数! \n");
        exit(0);
    }

    int keylen = strlen(key);// 密钥长度
    if(!checkKeyLen(keylen)) {
        printf("密钥字符长度错误! 长度必须为16、24和32。当前长度为%d\n",keylen);
        exit(0);
    }

    extendKey(key);// 扩展密钥
    int pArray[4][4];

    for(int k = 0; k < plen; k += 16) {
        convertToIntArray(p + k, pArray);// 将字符串读入4*4数组中

        addRoundKey(pArray, 0);// 一开始的轮密钥加

        // 前9轮
        for(int i = 1; i < 10; i++){
```

```
25
26              subBytes(pArray);// 字节代换
27
28              shiftRows(pArray);// 行移位
29
30              mixColumns(pArray);// 列混合
31
32              addRoundKey(pArray, i);
33
34          }
35
36          // 第10轮
37          subBytes(pArray);// 字节代换
38
39          shiftRows(pArray);// 行移位
40
41          addRoundKey(pArray, 10);
42
43          convertArrayToStr(pArray, p + k);
44      }
45  }
```

密钥扩展

密钥扩展将传入密钥从字符串读取到w[0]到w[3]，循环将w[44]每个元素初始化完成密钥的扩展。注意T函数进行轮常量异或时需要轮数，这里将j作为轮数进行传递。

```
1  // 密钥对应的扩展数组
2  static int w[44];
3
4  // 密钥扩展
5  static void extendKey(char *key) {
6      // 将传入密钥先读取到w[0]到w[3]
7      for(int i = 0; i < 4; i++)
8          w[i] = getWordFromStr(key + i * 4);
9
10     // 扩展密钥，将w[44]中每个元素初始化
11     for(int i = 4, j = 0; i < 44; i++) {
12         if( i % 4 == 0) {
13             // T函数行为与轮数有关，需要传递轮数j
14             w[i] = w[i - 4] ^ T(w[i - 1], j);
15             j++;// 下一轮
16         }else {
17             w[i] = w[i - 4] ^ w[i - 1];
18         }
```

```
    }
}
```

以下是常量代换表的定义和T函数的实现。

```
// 常量代换表
static const int Rcon[10] = {
    0x01000000, 0x02000000,
    0x04000000, 0x08000000,
    0x10000000, 0x20000000,
    0x40000000, 0x80000000,
    0x1b000000, 0x36000000 };

// T函数
static int T(int num, int round) {
    int numArray[4];
    // 从32位数取出4字节放入数组中，方便移位、代换操作
    splitIntToArray(num, numArray);
    leftLoop4int(numArray, 1);// 字循环

    // 字节代换
    for(int i = 0; i < 4; i++)
        numArray[i] = getNumFromSBox(numArray[i]);

    // 将数组中的4字节重新转化为32位数
    int result = mergeArrayToInt(numArray);

    // 进行轮常量异或后返回
    return result ^ Rcon[round];
}
```

字节代换

字节代换只需将状态矩阵每个元素取8位数中高4位作为行值，低4位作为列值，查表完成代换。

```
// 根据索引从S盒中获得元素
static int getNumFromSBox(int index) {
    int row = getLeft4Bit(index);
    int col = getRight4Bit(index);
    return S[row][col];
}

// 字节代换
static void subBytes(int array[4][4]){
```

```
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            array[i][j] = getNumFromSBox(array[i][j]);
}
```

行移位

行移位时将2、3、4行从状态矩阵中读取出来，完成移位后再复制回状态矩阵即可。

```
// 将数组中元素循环左移step位
static void leftLoop4int(int array[4], int step) {
    int temp[4];
    // 复制数组
    for(int i = 0; i < 4; i++)
        temp[i] = array[i];

    int index = step % 4 == 0 ? 0 : step % 4;
    // 循环完成移位
    for(int i = 0; i < 4; i++){
        array[i] = temp[index];
        index++;
        index = index % 4;
    }
}

// 行移位
static void shiftRows(int array[4][4]) {
    int rowTwo[4], rowThree[4], rowFour[4];
    // 复制状态矩阵的第2,3,4行
    for(int i = 0; i < 4; i++) {
        rowTwo[i] = array[1][i];
        rowThree[i] = array[2][i];
        rowFour[i] = array[3][i];
    }
    // 循环左移相应的位数
    leftLoop4int(rowTwo, 1);
    leftLoop4int(rowThree, 2);
    leftLoop4int(rowFour, 3);

    //把左移后的行复制回状态矩阵中
    for(int i = 0; i < 4; i++) {
        array[1][i] = rowTwo[i];
        array[2][i] = rowThree[i];
        array[3][i] = rowFour[i];
    }
}
```

列混合

列混合时，将状态矩阵与列混合矩阵相乘即可完成对状态矩阵的混淆。需要注意的是，矩阵元素的加法和乘法都是定义在GF(2^8)上的二元运算，对于加法等价于两个字节异或，乘法则需要另外实现。

```
// 列混合矩阵
static const int colM[4][4] = {
    2, 3, 1, 1,
    1, 2, 3, 1,
    1, 1, 2, 3,
    3, 1, 1, 2 };

// GF乘2
static int GFMul2(int s) {
    int result = s << 1;
    int a7 = result & 0x00000100;

    if(a7 != 0) {
        result = result & 0x000000ff;
        result = result ^ 0x1b;
    }

    return result;
}

// GF乘3
static int GFMul3(int s) {
    return GFMul2(s) ^ s;
}

// GF上的乘法
static int GFMul(int n, int s) {
    int result;

    if(n == 1)
        result = s;
    else if(n == 2)
        result = GFMul2(s);
    else if(n == 3)
        result = GFMul3(s);
    else if(n == 0x9)
        result = GFMul9(s);
    else if(n == 0xb)//11
        result = GFMul11(s);
```

```
    else if(n == 0xd)//13
        result = GFMul13(s);
    else if(n == 0xe)//14
        result = GFMul14(s);

    return result;
}

// 列混合
static void mixColumns(int array[4][4]) {

    int tempArray[4][4];

    // 复制状态矩阵
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            tempArray[i][j] = array[i][j];

    // 将状态矩阵与混合矩阵相乘，得到混淆后的状态矩阵
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++){
            array[i][j] = GFMul(colM[i][0],tempArray[0][j]) ^ GFMul(colM[i][1],t
                ^ GFMul(colM[i][2],tempArray[2][j]) ^ GFMul(colM[i][3], tempArra
        }
}
```

轮密钥加

轮密钥加根据轮数将状态矩阵与相应的w[i]异或即可。

```
// 轮密钥加
static void addRoundKey(int array[4][4], int round) {
    int warray[4];
    for(int i = 0; i < 4; i++) {
        // 将32位数转为4字节放入数组，方便操作
        splitIntToArray(w[ round * 4 + i], warray);

        // 字节逐位异或
        for(int j = 0; j < 4; j++) {
            array[j][i] = array[j][i] ^ warray[j];
        }
    }
}
```

解密

AES解密与加密存在一些不同，具体应通过流程图来理解。另外，解密函数中调用的是加密函数中各轮操作的逆函数，逻辑大致相当，这里只给出代码而不再赘述了。

```
void deAes(char *c, int clen, char *key) {

    int keylen = strlen(key);
    if(clen == 0 || clen % 16 != 0) {
        printf("密文字符长度必须为16的倍数！现在的长度为%d\n",clen);
        exit(0);
    }

    if(!checkKeyLen(keylen)) {
        printf("密钥字符长度错误！长度必须为16、24和32。当前长度为%d\n",keylen);
        exit(0);
    }

    extendKey(key);// 扩展密钥
    int cArray[4][4];
    for(int k = 0; k < clen; k += 16) {
        convertToIntArray(c + k, cArray);// 将字符串读入4*4数组中

        addRoundKey(cArray, 10);// 解密开始的轮密钥加

        int wArray[4][4];
        // 前9轮
        for(int i = 9; i >= 1; i--) {

            deSubBytes(cArray);

            deShiftRows(cArray);

            deMixColumns(cArray);

            getArrayFrom4W(i, wArray);

            deMixColumns(wArray);

            addRoundTowArray(cArray, wArray);
        }
        // 第10轮
        deSubBytes(cArray);

        deShiftRows(cArray);
```

```
41
42          addRoundKey(cArray, 0);
43
44          convertArrayToStr(cArray, c + k);
45
46      }
47  }
```

逆字节变换

```
1  // 根据索引从逆S盒中获取值
2  static int getNumFromS1Box(int index) {
3          int row = getLeft4Bit(index);
4          int col = getRight4Bit(index);
5          return S2[row][col];
6  }
7
8  // 逆字节变换
9  static void deSubBytes(int array[4][4]) {
10         for(int i = 0; i < 4; i++)
11                 for(int j = 0; j < 4; j++)
12                         array[i][j] = getNumFromS1Box(array[i][j]);
13 }
```

逆行移位

```
1  // 把4个元素的数组循环右移step位
2  static void rightLoop4int(int array[4], int step) {
3          int temp[4];
4          for(int i = 0; i < 4; i++)
5                  temp[i] = array[i];
6
7          int index = step % 4 == 0 ? 0 : step % 4;
8          index = 3 - index;
9          for(int i = 3; i >= 0; i--) {
10                 array[i] = temp[index];
11                 index--;
12                 index = index == -1 ? 3 : index;
13         }
14 }
15
16 // 逆行移位
17 static void deShiftRows(int array[4][4]) {
```

```
        int rowTwo[4], rowThree[4], rowFour[4];
        for(int i = 0; i < 4; i++) {
                rowTwo[i] = array[1][i];
                rowThree[i] = array[2][i];
                rowFour[i] = array[3][i];
        }

        rightLoop4int(rowTwo, 1);
        rightLoop4int(rowThree, 2);
        rightLoop4int(rowFour, 3);

        for(int i = 0; i < 4; i++) {
                array[1][i] = rowTwo[i];
                array[2][i] = rowThree[i];
                array[3][i] = rowFour[i];
        }
}
```

逆列混合

```
// 逆列混合矩阵
static const int deColM[4][4] = { 0xe, 0xb, 0xd, 0x9,
        0x9, 0xe, 0xb, 0xd,
        0xd, 0x9, 0xe, 0xb,
        0xb, 0xd, 0x9, 0xe };

// 逆列混合
static void deMixColumns(int array[4][4]) {
        int tempArray[4][4];

        for(int i = 0; i < 4; i++)
                for(int j = 0; j < 4; j++)
                        tempArray[i][j] = array[i][j];

        for(int i = 0; i < 4; i++)
                for(int j = 0; j < 4; j++){
                        array[i][j] = GFMul(deColM[i][0],tempArray[0][j]) ^ GFMu
                                ^ GFMul(deColM[i][2],tempArray[2][j]) ^ GFMul(de
                }
}
```

其他杂项

```
// S盒
static const int S[16][16] = { 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,

// 逆S盒
static const int S2[16][16] = { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44,
        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b,
        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49,
        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc,
        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57,
        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05,
        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03,
        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce,
        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8,
        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e,
        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe,
        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f,
        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c,
        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,

// 获取整形数据的低8位的左4个位
static int getLeft4Bit(int num) {
        int left = num & 0x000000f0;
        return left >> 4;
}

// 获取整形数据的低8位的右4个位
static int getRight4Bit(int num) {
        return num & 0x0000000f;
}

```

```
// 把一个字符转变成整型
static int getIntFromChar(char c) {
        int result = (int) c;
        return result & 0x000000ff;
}


// 把16个字符转变成4X4的数组，
// 该矩阵中字节的排列顺序为从上到下，
// 从左到右依次排列。
static void convertToIntArray(char *str, int pa[4][4]) {
        int k = 0;
        for(int i = 0; i < 4; i++)
                for(int j = 0; j < 4; j++) {
                        pa[j][i] = getIntFromChar(str[k]);
                        k++;
                }
}

// 打印4X4的数组
static void printArray(int a[4][4]) {
        for(int i = 0; i < 4; i++){
                for(int j = 0; j < 4; j++)
                        printf("a[%d][%d] = 0x%x ", i, j, a[i][j]);
                printf("\n");
        }
        printf("\n");
}

// 打印字符串的ASSCI
// 以十六进制显示
static void printASSCI(char *str, int len) {
        for(int i = 0; i < len; i++)
                printf("0x%x ", getIntFromChar(str[i]));
        printf("\n");
}

// 把连续的4个字符合并成一个4字节的整型
static int getWordFromStr(char *str) {
        int one = getIntFromChar(str[0]);
        one = one << 24;
        int two = getIntFromChar(str[1]);
        two = two << 16;
        int three = getIntFromChar(str[2]);
        three = three << 8;
        int four = getIntFromChar(str[3]);
        return one | two | three | four;
```

```
95 }
96
97
98 // 把一个4字节的数的第一、二、三、四个字节取出，
99 // 放入一个4个元素的整型数组里面。
100 static void splitIntToArray(int num, int array[4]) {
101         int one = num >> 24;
102         array[0] = one & 0x000000ff;
103         int two = num >> 16;
104         array[1] = two & 0x000000ff;
105         int three = num >> 8;
106         array[2] = three & 0x000000ff;
107         array[3] = num & 0x000000ff;
108 }
109
110
111 // 把数组中的第一、二、三和四元素分别作为
112 // 4字节整型的第一、二、三和四字节，合并成一个4字节整型
113 static int mergeArrayToInt(int array[4]) {
114         int one = array[0] << 24;
115         int two = array[1] << 16;
116         int three = array[2] << 8;
117         int four = array[3];
118         return one | two | three | four;
119 }
120
121 // 把4X4数组转回字符串
122 static void convertArrayToStr(int array[4][4], char *str) {
123         for(int i = 0; i < 4; i++)
124                 for(int j = 0; j < 4; j++)
125                         *str++ = (char)array[j][i];
126 }
127
128 // 检查密钥长度
129 static int checkKeyLen(int len) {
130         if(len == 16)
131                 return 1;
132         else
133                 return 0;
134 }
135
136 // 把两个4X4数组进行异或
137 static void addRoundTowArray(int aArray[4][4],int bArray[4][4]) {
138         for(int i = 0; i < 4; i++)
139                 for(int j = 0; j < 4; j++)
140                         aArray[i][j] = aArray[i][j] ^ bArray[i][j];
141 }
```

```
142
143 // 从4个32位的密钥字中获得4X4数组，
144 // 用于进行逆列混合
145 static void getArrayFrom4W(int i, int array[4][4]) {
146         int index = i * 4;
147         int colOne[4], colTwo[4], colThree[4], colFour[4];
148         splitIntToArray(w[index], colOne);
149         splitIntToArray(w[index + 1], colTwo);
150         splitIntToArray(w[index + 2], colThree);
151         splitIntToArray(w[index + 3], colFour);
152
153         for(int i = 0; i < 4; i++) {
154                 array[i][0] = colOne[i];
155                 array[i][1] = colTwo[i];
156                 array[i][2] = colThree[i];
157                 array[i][3] = colFour[i];
158         }
159
160 }
```

应用（杨俊贤）

静态页面AES加密的实现

引入纯JavaScript的加密类库：`crypto-js`

```
1 <script type="text/javascript"src="https://cdn.jsdelivr.net/npm/crypto-
  js@4.0.0/crypto-js.min.js"></script>
```

获取试探密文：

```
1 temp = CryptoJS.AES.encrypt("XV33233", "password").toString();
```

获得试探密文为：`U2FsdGVkX19YCTT5erGX2S7lDy1LY325JwRZXHLefzk=`

搭建前端页面：

```
1 <div style="border: 2px solid black;padding: 10px;">
2         <div data-origin="XV33233" data-now="U2FsdGVkX19YCTT5erGX2S7lDy1LY325JwR
3      class="cryptoText">
4             U2FsdGVkX184cvZYgKlIx0x0IBdt0ECrcMGZUHuJ0/U=
```

```
        </div>
        请输入密码查看隐藏内容：<input type="password">
        <input type="submit" onclick="crypto(this)"></button>
    </div>
```

JS部分：

```
<script>
        function crypto(sub) {
        const e = sub.parentNode.firstElementChild; //密文部分
        const key = sub.previousElementSibling.value; //用户输入的密钥
        const origin = e.getAttribute("data-origin"); //试探明文
        const now = e.getAttribute("data-now"); //试探密文
        const res = CryptoJS.AES.decrypt(now, key).toString(CryptoJS.enc.Utf8);
        if (res == origin) {
        //如果密钥正确
        const t = e.innerHTML.replace(/\s/g, "");
        sub.parentNode.innerHTML = CryptoJS.AES.decrypt(t, key).toString(CryptoJ
        }
        else {
        alert("密码错误！");
        }
        }
    </script>
```

成品：

请输入密码查看隐藏内容： 提交

CryptoPre

密钥为：`password`