# S-AES 算法程序开发手册

## 1. 项目概述

S-AES（Simplified Advanced Encryption Standard）算法是一个用于教育和理解加密算法基本原理的简化版本，用于加密和解密 16 位二进制数据块。本项目提供了一个基于 C++ 的 S-AES 算法程序，用于演示和学习加密算法的基本原理。

## 2. 使用说明简介

（1）选择二进制加密或者 ACSII 字符串加密。
（2）输入 16 位的二进制明文或密文。
（3）输入 16 位的二进制密钥。
（4）选择加密或解密模式。
（5）程序将生成加密后的密文或解密后的明文。

## 3. 算法原理

### 3.1 分组长度

S-AES 使用 16 位数据块作为输入。

### 3.2 密钥长度

S-AES 使用 16 位密钥作为输入。

### 3.3 算法描述

#### 3.3.1 加密算法

S-AES 加密算法包括以下步骤：
（1）密钥扩展（Key Expansion）：使用密钥排列算法，将 16 位的主密钥扩展为两个轮密钥。
（2）初始轮（Initial Round）：将明文块与主密钥进行 XOR 操作。
（3）第一轮
①半字节代替（SubBytes）：将数据块中的每个字节分成两部分，分别通过 S-盒替代，然后合并。
②行位移（ShiftRows）：对数据块的行进行循环位移操作，以增加混淆度。
③列混淆（MixColumns）：对数据块的列进行混淆操作，通过矩阵乘法进行。
④轮密钥加（AddRoundKey）：将数据块与第一个轮密钥进行 XOR 操作。
（4）第二轮：
①半字节代替（SubBytes）：再次进行半字节代替操作。
②行位移（ShiftRows）：再次进行行位移操作。
③轮密钥加（AddRoundKey）：将数据块与第二个轮密钥进行 XOR 操作。
（5）输出密文（ciphertext）：最后的数据块即为密文，长度为 16 位。

加密算法使用 4 个不同的函数或变换：密钥加（$A_k$）、半字节代替（NS）、行移位（SR）和列混淆（MC）。将加密算法表示为一个复合函数：

$$A_{K_2} \circ \mathrm{SR} \circ \mathrm{NS} \circ A_{K_1} \circ \mathrm{MC} \circ \mathrm{SR} \circ \mathrm{NS} \circ A_{K_0}$$

### 3.3.2 解密算法

S-AES 解密算法包括以下步骤：

（1）初始轮（Initial Round）：将明文块与第二个轮密钥进行 XOR 操作。

（3）第一轮

①逆行位移（Inverse ShiftRows）：对数据块的行进行逆循环位移操作，与加密时相反。

②逆半字节代替（Inverse SubBytes）：将数据块中的每个字节分成两部分，分别通过逆 S-盒替代，然后合并。

③轮密钥加（AddRoundKey）：将数据块与第一个轮密钥进行 XOR 操作。

④逆列混淆（Inverse MixColumns）：对数据块的列进行逆混淆操作，通过逆矩阵乘法进行。

（4）第二轮：

①逆行位移（Inverse ShiftRows）：再次进行逆行位移操作，与加密时相反。

②逆半字节代替（Inverse SubBytes）：再次进行逆半字节代替操作。

③轮密钥加（AddRoundKey）：将数据块与主密钥进行 XOR 操作。

（5）输出明文（plaintext）：最后的数据块即为明文，长度为 16 位。

将解密算法表示为复合函数：

$$A_{K_0} \circ \mathrm{INS} \circ \mathrm{ISR} \circ \mathrm{IMC} \circ A_{K_1} \circ \mathrm{INS} \circ \mathrm{ISR} \circ A_{K_2}$$

### 3.3.3 密钥扩展

16 位初始密钥被分成两个 8 位字，算法如下。

$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \mathrm{RCON}(1) \oplus \mathrm{SubNib}(\mathrm{RotNib}(w_1))$$
$$w_3 = w_2 \oplus w_1$$
$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus \mathrm{RCON}(2) \oplus \mathrm{SubNib}(\mathrm{RotNib}(w_3))$$
$$w_5 = w_4 \oplus w_3$$

RCON 是一个轮常数，RCON(1)=10000000，RCON(2)=00110000。

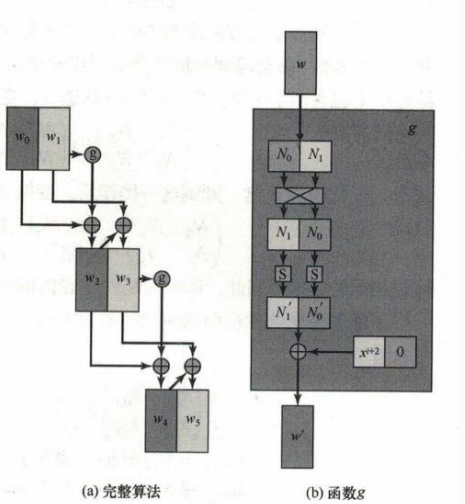


图 D.5　S-AES 密钥扩展

## 3.4 转换装置设定

### 3.4.1 S 盒

| | | j | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 10 | 11 |
| i | 00 | 9 | 4 | A | B |
| | 01 | D | 1 | 8 | 5 |
| | 10 | 6 | 2 | 0 | 3 |
| | 11 | C | E | F | 7 |

(a) S 盒

注意：阴影格中的是十六进制数，非阴影格中是二进制数。

### 3.4.2 逆 S 盒

| | | j | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 10 | 11 |
| i | 00 | A | 5 | 9 | B |
| | 01 | 1 | 7 | 8 | F |
| | 10 | 6 | 0 | 2 | 3 |
| | 11 | C | 4 | D | E |

(b) 逆 S 盒

# 4. 代码组件

## 4.1 主要函数

为了使代码模块化程度更高，逻辑更加清晰，编写多个核心函数来进行加解密。并且首先在外部定义加密过程中所需要的全局变量。

```
//S盒
const int s[4][4] = {
    {9, 4, 10, 11},
    {13, 1, 8, 5},
    {6, 2, 0, 3},
    {12, 14, 15, 7}};
//逆S盒
const int s_ni[4][4]= {
    {10,5,9,11},
    {1,7,8,15},
    {6,0,2,3},
    {12,4,13,14}};

const int tihuanwei[16][4] = {
    {0, 0, 0, 0},
    {0, 0, 0, 1},
    {0, 0, 1, 0},
    {0, 0, 1, 1},
    {0, 1, 0, 0},
    {0, 1, 0, 1},
    {0, 1, 1, 0},
    {0, 1, 1, 1},
    {1, 0, 0, 0},
    {1, 0, 0, 1},
    {1, 0, 1, 0},
    {1, 0, 1, 1},
    {1, 1, 0, 0},
    {1, 1, 0, 1},
    {1, 1, 1, 0},
    {1, 1, 1, 1}};
```

```
//* 定义轮常数
int rcon1[8] = {1, 0, 0, 0, 0, 0, 0, 0};
int rcon2[8] = {0, 0, 1, 1, 0, 0, 0, 0};
```

### 4.1.1 实现 x^nfx 的函数

```
void x_de_n_fang_cheng_fx(int xfx[4], int a[4]) //* xfx是结果，a是上一步的结果
{
    //! 注意要取模
    //! 既约多项式是 x^4 + x + 1
    //* 保存四次乘法的系数
    if (a[0] == 0)
    {
        for (int i = 0; i < 3; i++)
            xfx[i] = a[i + 1];
    }
    else
    {
        //! 如果乘数首项不为1就需要将 b1x^2+b0x 与 x+1 进行异或
        xfx[1] = a[2];
        xfx[2] = a[3] == 1 ? 0 : 1;
        xfx[3] = 1;
    }
}
```

### 4.1.2 乘法函数

```
int *chengfa(int a[4], int b[4])
{
    //* 储存结果的系数
    int *result = new int[4];
    for (int i = 0; i < 4; i++)
        result[i] = 0;

    //* 记录下x^nfx
    int xfx[4] = {0};
    x_de_n_fang_cheng_fx(xfx, a);
    int x2fx[4] = {0};
    x_de_n_fang_cheng_fx(x2fx, xfx);
    int x3fx[4] = {0};
    x_de_n_fang_cheng_fx(x3fx, x2fx);

    //* 现在需要根据多项式a和b开始异或
    if (b[0] == 1)
        for (int i = 0; i < 4; i++)
            result[i] ^= x3fx[i];
    if (b[1] == 1)
        for (int i = 0; i < 4; i++)
            result[i] ^= x2fx[i];
    if (b[2] == 1)
        for (int i = 0; i < 4; i++)
            result[i] ^= xfx[i];
    if (b[3] == 1)
        for (int i = 0; i < 4; i++)
            result[i] ^= a[i];
    return result;
}
```

### 4.1.3 逆多项式乘法函数

```
// 逆多项式乘法
int *ni_chengfa(int a[4], int b[4]) {
    int *result = new int[4];
    for (int i = 0; i < 4; i++)
        result[i] = 0;

    // 计算 x^3fx
    int x3fx[4];
    for (int i = 0; i < 4; i++) {
        x3fx[i] = a[i];
    }

    // 计算 x^2fx
    int x2fx[4];
    x_de_n_fang_cheng_fx(x2fx, x3fx);

    // 计算 xfx
    int xfx[4];
    x_de_n_fang_cheng_fx(xfx, x2fx);

    // 根据多项式 b 开始异或操作
    if (b[0] == 1) {
        for (int i = 0; i < 4; i++) {
            result[i] ^= x3fx[i];
        }
    }
    if (b[1] == 1) {
        for (int i = 0; i < 4; i++) {
            result[i] ^= x2fx[i];
        }
    }
    if (b[2] == 1) {
        for (int i = 0; i < 4; i++) {
            result[i] ^= xfx[i];
        }
    }
    if (b[3] == 1) {
        for (int i = 0; i < 4; i++) {
            result[i] ^= a[i];
        }
    }

    return result;
}
```

### 4.1.4 八位异或函数

```
int *yihuo8(int *a, int *b)//8位的异或
{
    int *t = new int[8];
    for (int i = 0; i < 8; i++)
        t[i] = a[i] ^ b[i];
    return t;
}
```

#### 4.1.5 四位异或函数

```
int *yihuo4(int *a, int *b)//4位的异或
{
    int *t = new int[4];
    for (int i = 0; i < 4; i++)
        t[i] = a[i] ^ b[i];
    return t;
}
```

#### 4.1.6 轮密钥加

```
void lunmiyaojia(int **mingwen, int **key)
{
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 8; j++)
            mingwen[i][j] ^= key[i][j];
}
```

#### 4.1.7 输出函数

```
//用于输出
void shuchu(int **a)
{
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 8; j++)
            cout << a[i][j] << ' ';
    cout << endl;
}
```

### 4.2 二进制加密函数

#### 4.2.1 S 盒替换

```
void s_he_tihuan(int *temp) //使用s盒替换的函数，8位换
{
    int t1 = 2 * temp[0] + temp[1];
    int t2 = 2 * temp[2] + temp[3];
    int t3 = 2 * temp[4] + temp[5];
    int t4 = 2 * temp[6] + temp[7];
    int tihuan1 = s[t1][t2]; //记录替换后的数字
    int tihuan2 = s[t3][t4];
    //* 四位四位进行替换
    for (int i = 0; i < 4; i++)
        temp[i] = tihuanwei[tihuan1][i];
    for (int i = 0; i < 4; i++)
        temp[i + 4] = tihuanwei[tihuan2][i];
}
```

#### 4.2.2 循环左移

```
void zuoyi(int **temp) //循环左移
{
    for(int i = 4;i < 8;i ++)
    {
        int t = temp[0][i];
        temp[0][i] = temp[1][i];
        temp[1][i] = t;
    }
}
```

### 4.2.3 列混淆

```
void liehunxiao(int **mingwen)
{
    int si_de2jinzhi[4] = {0, 1, 0, 0};
    int *m00 = new int[4];
    int *m10 = new int[4];
    int *m01 = new int[4];
    int *m11 = new int[4];
    for (int i = 0; i < 4; i++)
    {
        m00[i] = mingwen[0][i];
        m10[i] = mingwen[0][i + 4];
        m01[i] = mingwen[1][i];
        m11[i] = mingwen[1][i + 4];
    }
    int *n00 = new int[4];
    int *n10 = new int[4];
    int *n01 = new int[4];
    int *n11 = new int[4];
    n00 = yihuo4(m00, chengfa(si_de2jinzhi, m10));
    n10 = yihuo4(chengfa(si_de2jinzhi, m00), m10);
    n01 = yihuo4(m01, chengfa(si_de2jinzhi, m11));
    n11 = yihuo4(chengfa(si_de2jinzhi, m01), m11);
    for (int i = 0; i < 4; i++)
    {
        mingwen[0][i] = n00[i];
        mingwen[0][i + 4] = n10[i];
        mingwen[1][i] = n01[i];
        mingwen[1][i + 4] = n11[i];
    }
}
```

## 4.3 二进制解密

### 4.3.1 逆 S 盒替换

```
void s_ni_tihuan(int *temp) //使用逆s盒替换的函数，8位换
{
    int t1 = 2 * temp[0] + temp[1];
    int t2 = 2 * temp[2] + temp[3];
    int t3 = 2 * temp[4] + temp[5];
    int t4 = 2 * temp[6] + temp[7];
    int tihuan1 = s_ni[t1][t2]; //记录替换后的数字
    int tihuan2 = s_ni[t3][t4];
    //* 四位四位进行替换
    for (int i = 0; i < 4; i++)
        temp[i] = tihuanwei[tihuan1][i];
    for (int i = 0; i < 4; i++)
        temp[i + 4] = tihuanwei[tihuan2][i];
}
```

### 4.3.2 逆行变换

```
void ni_zuoyi(int **state) {
    for (int i = 4; i < 8; i++) {
        int t = state[1][i];
        state[1][i] = state[0][i];
        state[0][i] = t;
    }
}
```

### 4.3.3 逆列混淆

```
void Invliehunxiao(int** miwen)
{
    int two_2jinzhi[4] = { 0,0,1,0 };
    int nine_2jinzhi[4] = { 1,0,0,1 };
    int* m00 = new int[4];
    int* m10 = new int[4];
    int* m01 = new int[4];
    int* m11 = new int[4];
    for (int i = 0; i < 4; i++)
    {
        m00[i] = miwen[0][i];
        m10[i] = miwen[0][i + 4];
        m01[i] = miwen[1][i];
        m11[i] = miwen[1][i + 4];
    }
    int* n00 = new int[4];
    int* n10 = new int[4];
    int* n01 = new int[4];
    int* n11 = new int[4];
    n00 = yihuo4(chengfa(nine_2jinzhi,m00), chengfa(two_2jinzhi, m10));
    n10 = yihuo4(chengfa(two_2jinzhi, m00), chengfa(nine_2jinzhi,m10));
    n01 = yihuo4(chengfa(nine_2jinzhi,m01), chengfa(two_2jinzhi, m11));
    n11 = yihuo4(chengfa(two_2jinzhi, m01), chengfa(nine_2jinzhi,m11));
    for (int i = 0; i < 4; i++)
    {
        miwen[0][i] = n00[i];
        miwen[0][i + 4] = n10[i];
        miwen[1][i] = n01[i];
        miwen[1][i + 4] = n11[i];
    }
}
```

## 4.4 ASCII 码加/解密

由于加解密中涉及到 ASCII 字符串和二进制字符串之间的转换，编写两个函数来实现。

```
// 将ASCII字符串转换为二进制字符串
void asciiToBinaryString(const char *ascii, char *binary) {
    for (int i = 0; i < 8; i++) {
        char c = ascii[i];
        for (int j = 0; j < 8; j++) {
            binary[i * 8 + j] = ((c >> (7 - j)) & 1) ? '1' : '0';
        }
    }
}

// 将二进制字符串转换为ASCII字符串
void binaryStringToAscii(const char *binary, char *ascii) {
    for (int i = 0; i < 8; i++) {
        char c = 0;
        for (int j = 0; j < 8; j++) {
            c = (c << 1) | (binary[i * 8 + j] - '0');
        }
        ascii[i] = c;
    }
}

//异或
void XORBlocks(int* a, int* b) {
    for (int i = 0; i < 8; i++) {
        a[i] ^= b[i];
    }
}
```

本程序支持任意位数的 ASCII 码加解密，因此在最初使用 String 类型获取用户输入，同时取得用户输入的字符个数，再将其转换为定长的字符数组。通过 8bit 临时数组 cut 以及循环标识 flag 的使用，以 1byte 为一组进行加/解密。其中加解密过程的选择通过用户输入 choose1 和 if 语句来实现。

```
cout<<"请输入明/密文+密钥"<<endl;
cin>>mm;
int length =mm.length(); //输入的字符个数
cout<<"长度为："<<length<<endl;
char m[length]; //字符串转字符数组
strcpy(m, mm.c_str());
//将ASCII明文转换为二进制字符串
char binaryM[length*8];
asciiToBinaryString(m, binaryM); //现在总明文存在binaryM数组里，在之后需要切割成2个一组
cout<<"转换后的二进制数组为："<<endl;
 for(int i=0;i<length*8;i++)
 cout<<binaryM[i];
 cout<<endl;
for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
    cin >> key[i][j];

for (int i = 0; i < 2; i++)
    key1[i] = new int[8];
for (int i = 0; i < 2; i++)
    key2[i] = new int[8];

key1[0] = yihuo8(key[0], g(key[1], rcon1));
key1[1] = yihuo8(key1[0], key[1]);
key2[0] = yihuo8(key1[0], g(key1[1], rcon2));
key2[1] = yihuo8(key2[0], key1[1]);

while(flag<length*8)
{
for(int i=0;i<2;i++)
for(int j=0;j<8;j++)
{
    int temp=0;
    if(binaryM[k]=='0')temp=0;
    else temp=1;
    cout<<binaryM[k]<<" ";
     mingwen[i][j]=temp;
     cout<<mingwen[i][j]<<endl;
     k++;
} //切割数组存放16位
```

在切割以及转换完成后，按照二进制加解密过程执行即可。

## 4.5 中间相遇攻击

```
if(choose==9)//密钥全遍历

{
    bool get = false; //确认是否找到了密钥

    cout<<"请输入明文+密文"<<endl;
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
    cin >> mingwen[i][j];
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
    cin >> miwen[i][j];
    for (int i = 0; i < 2; i++)
    keyD[i] = new int[8];
    for (int i = 0; i < 2; i++)
    key[i] = new int[8];
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
    {
    key[i][j]=0; //初始化密钥
    keyD[i][j]=0;
    }

    int sum=0;
    int sumz=0;

while(sumz!=17) //keyD的所有可能和key1的所有可能组合
{
//初始化实验数组
for (int i = 0; i < 2; i++)
for (int j = 0; j < 8; j++)
mingwen_test[i][j] = mingwen[i][j];


bool shouldbreak = false;
for(int i=0;i<2 && !shouldbreak; i++)
for(int j=0;j<8;j++)
    {
        if(keyD[i][j]==0){
        keyD[i][j]=1;
        shouldbreak = true;
        break;
        }
        if(keyD[i][j]==1){
        keyD[i][j]=0;
        }

    } //keyD二进制数组全排列

    sumz=0;
    for(int i=0;i<2;i++)
    for(int j=0;j<8;j++)
    if(keyD[i][j]==1)sumz++;//统计1的个数
//      cout<<"----------------------------------kerD中1的个数为："<<sumz<<endl;


    //开始尝试，首先生成密钥keyD1,keyD2,给加密使用
         for (int i = 0; i < 2; i++)
         keyD1[i] = new int[8];
         for (int i = 0; i < 2; i++)
         keyD2[i] = new int[8];

keyD1[0] = yihuo8(keyD[0], g(keyD[1], rcon1));
keyD1[1] = yihuo8(keyD1[0], keyD[1]);
keyD2[0] = yihuo8(keyD1[0], g(keyD1[1], rcon2));
keyD2[1] = yihuo8(keyD2[0], keyD1[1]);
```

```
    sum=0;
    while(sum!=17)
    {
       //初始化实验数组
    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
    miwen_test[i][j] = miwen[i][j];

        bool shouldbreak2 = false;
          for(int i=0;i<2 && !shouldbreak2 ;i++)
          for(int j=0;j<8;j++)
       {
          if(key[i][j]==0){
          key[i][j]=1;
          shouldbreak2 = true;
          break;
          }
          if(key[i][j]==1){
          key[i][j]=0;
          }

       } //二进制数组全排列
       sum=0;
       for(int i=0;i<2;i++)
       for(int j=0;j<8;j++)
       {
          if(key[i][j]==1)sum++;
       }//统计1的个数

       //开始尝试，首先生成密钥key1,key2
         for (int i = 0; i < 2; i++)
         key1[i] = new int[8];
         for (int i = 0; i < 2; i++)
         key2[i] = new int[8];

key1[0] = yihuo8(key[0], g(key[1], rcon1));
key1[1] = yihuo8(key1[0], key[1]);
key2[0] = yihuo8(key1[0], g(key1[1], rcon2));
key2[1] = yihuo8(key2[0], key1[1]);


    //进行明密文对比
    int count=0;

    for (int i = 0; i < 2; i++)
    for (int j = 0; j < 8; j++)
       {
          if(mingwen_test[i][j]==miwen_test[i][j])count+=1;
       }

       if(count==16)
       {
          get= true;
          cout<<"破译成功!"<<endl<<"密钥K1为：";
       for (int i = 0; i < 2; i++)
       for (int j = 0; j < 8; j++)
       cout<<keyD[i][j];
       cout<<endl;
          cout<<"密钥K2为：";
       for (int i = 0; i < 2; i++)
       for (int j = 0; j < 8; j++)
       cout<<key[i][j];
       cout<<endl;

       if(sum==16)sum=17; //全排列结束条件

       }

    if(sumz==16)sumz=17; //全排列结束条件

    }


if(get==false) cout<<"无对应密钥";
//else cout<<"破译成功";
return 0;
}
```

## 4.6 其它函数

### 4.6.1 明密文块异或

```
//异或
void XORBlocks(int* a, int* b) {
    for (int i = 0; i < 8; i++) {
        a[i] ^= b[i];
    }
}
```

### 4.6.2 十六进制转二进制数组存放

```
//16转二进制数组存放iv
void toBinaryArray(uint16_t value, int numBits, int binaryArray[]) {
    for (int i = numBits - 1; i >= 0; i--) {
        binaryArray[i] = (value & (1 << i)) ? 1 : 0;
    }
}
```

### 4.6.3 随机生成初始向量 IV

```
// 随机生成初始向量 IV
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<uint16_t> dis(0, 0xFFFF);
uint16_t iv = dis(gen);
```