# AES 算法设计与实现

王越*

(南京大学 计算机科学与技术系，南京 210093)

# The design implementation of AES algorithm

Wang Yue*

(Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

**Abstract**: With the development of symmetric encryption,3DES is relatively slow with software , whose group length of 64 digits seem to not effective and safe enough. AES(Advanced Encryption Standard) became the new encryption standard of US.AES is an iterative and symmetric group key encryption encryption.This paper first introduces the mathematic foundation , and then elaborates the kernel of AES-the implementation process of password box. We also design the algorithms to inverse, multiply, divide polynomial. When finishing the AES system, we found that the encryption work was rather inefficient. And after analyzing the problem ,some means such as look-up table has been used to improve the efficiency.

**Key words**: AES; algorithm ; efficiency

**摘 要**: ：随着对称密码的发展，3DES用软件实现速度相对较慢，它使用的64位分组长度显得不够高效和安全的缺点使得需要一种新的高级加密标准来替代它。AES（Advanced Encryption Standard）于2002年成为新的联邦信息加密标准。AES是一个迭代的、对称密钥分组的密码。本文首先介绍AES算法的数学基础，并详细阐述了AES算法的核心-密码盒的详细实现过程,对实现过程中各环节的算法设计思想作了深入细致的介绍，提出多项式求逆、相乘、相除的计算算法并实现。同时，我们在实现的过程中发现严格按照原算法描述，加密速度很慢，在分析其原因之后，使用查表等手段降低冗余计算，极大的提高了计算时间。

关键词： AES;算法;加密速度

**中图法分类号**: TP301 **文献标识码**: A

## 1 引言

随着对称密码的发展，3DES 用软件实现速度相对较慢，它使用的 64 位分组长度显得不够高效和安全的缺点使得需要一种新的高级加密标准来替代它。AES 的全称是 Advanced Encryption Standard，即高级加密标准。该项目由美国国家标准技术研究所 (NIST)于 1997 年开始启动并征集算法，在 2000 年确定采用 Rijndael 作为其最终算法，并于 2001 年被美国商务部部长批准为新的联邦信息加密标准 (FIPS PUB 197)，该标准的

* 作者简介:

正式生效日期是 2002 年 5 月 26 日。2000 年 10 月 2 日，NIST 对 Rijndael 做 出了最终评估。

AES 是一个迭代的、对称密钥分组的密码，它可以使用 128、192 和 256 位密钥，并且用 128 位(16 字节)分组加密和解密数据。与公共密钥密码使用密钥对不同，对称密钥密码使用相同的密钥加密和解密数据。通过分组密码返回的加密数据的位数与输入数据相同。迭代加密使用一个循环结构，在该循环中重复置换(permutations)和替换(substitutions)输入数据。

本文首先介绍 AES 算法的数学基础，包括群，域和有限域 GF(2)等，因为 AES 加密的基本单元是字节和字（长 4 个字节），字节或字之间的运算主要是发生在有限域 GF(2)和 $GF(2^8)$。同时详细阐述 AES 的基本思路和算法，以及我们的实现过程。

## 2 AES 的数学基础

AES 算法中的许多运算是按字节定义的，一个字节为 8 位。AES 算法中还有一些运算是 4 个字节的字定义的，一个 4 字节的字为 32 位。一个字节可以看成是有限域 $GF(2^8)$中的一个元素。一个 4 字节的字可以看成是系数在 $GF(2^8)$中并且次数小于 4 的多项式。

### 2.1 群、环和域

**群：** 群包含一个集合 G 和一个定义在集合元素上的运算，这里表示为“+”：

$$+:\ G \times G \to G:(a,b) \to a+b$$

该运算必须满足以下条件：

1、封闭性：$\forall a,b \in G: a+b \in G$

2、结合性：$\forall a,b,c \in G: (a+b)+c = a+(b+c)$

3、零元素：$\exists 0 \in G, \forall a \in G: a+0 = a$

4、逆元素：$\forall a \in G, \exists b \in G: a+b = 0$

满足交换律的群称为交换群，也叫阿贝尔群。

5、交换律：$\forall a,b \in G, a+b = b+a$

一个典型的交换群是整数加群$<Z,+>$：整数集和加法运算。另一个例子是结构$<Z_n,+>$该集合包含从 0 到 n-1 的整数，其运算是模 n 下的加法，即$\forall a,b \in Z, a+b := (a+b) \bmod n$。

**环：**环$<R,+,\cdot>$。包含一个集合 R 和两个定义在 R 上元素的运算，分别用“+”和“。”表示，当然也可以用其他运算表示，环上的运算必须满足以下条件：

1、$<R,+>$是交换群

2、R 上的运算“·”满足封闭性和结合律。R 关于运算“·”存在零元。

3、运算“+”和“·”服从分配率。

$\forall a,b,c \in R: (a+b)\cdot c = (a\cdot c)+(b\cdot c)$

运算“·”下的零元素通常用 1 表示。若运算“·”具有可交换性，则环$<R,+,\cdot>$被称为交换环。

环的最常见的例子是$<Z,+,\cdot>$。该环是交换环。

**域：**结构$<F,+,\cdot>$称为一个域，如果满足以下两个条件：

1、$<F,+,\cdot>$是一个交换环。

2、$<F,+>$中的元素除了零元素 **0** 以为，F 中的其他所有元素关于“·”在 F 中均存在逆元素。

域上最常见的例子是具有加法和乘法运算的实数集。

### 2.2 有限域GF(2)

有限域是包含有限个元素的域。集合中元素的个数即为域的阶。M 阶域存在当且仅当 m 是某素数的幂，即存在某个整数 n 和素数 p，使得$m = p^n$。P 称为有限域的特征。具有相同阶的有限域是同构的，我们用$GP(p^n)$表示。

为了与一般意义上的“加法”和“乘法”区别，我们使用“$\oplus$”表示域中的加法，“$\otimes$”表示域中的乘法。

AES 算法中均使用以 2 为特征的有限域，我们首先介绍GP(2)。GP(2)中只有两个元素，定义其中的加法⊕：$\forall a,b \in GP(2), a \oplus b \coloneqq a\hat{}b$，其中符号“^”表示异或。定义乘法⊗：×，其中符号“×”表示一般意义上的乘法。事实上GP(2)只有两个元素{0,1}。

### 2.3 有限域上的多项式环

域 F 上的多项式形式如下

$$b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots b_2x^2 + b_1x + b_0$$

x 称为多项式的变元，$b_i \in F$是多项式的系数。域上次数小于 k 的多项式集合用F[X]|k表示。F[X]|k中的多项式可以通过将其 k 个系数作为一个字符串来实现有效存储。GP(2)上次数小于 8 的多项式可以用一个字节来存放。$b(x) \to b_7b_6 \dots b_0$。例如，例如多项式 $x^5 + x^2 + x + 1$可以用比特串 00100111 表示，也就是 16 进制表示 27。

多项式加法：多项式之和等于先对所有具有 x 次幂的系数求和然后各项再相加，特别要注意的是，各系数求和是在域 F 中进行的：

$c(x) = a(x) + b(x) \Leftrightarrow c_i = a_i \oplus b_i, 0 \le i < n$

多项式乘法：多项式乘法定义为模多想式乘积

$$c(x) = a(x) * b(x) \Leftrightarrow c(x) \equiv a(x) \times b(x) (\mathrm{mod}\ m(x))$$

其中 m（x）为约化多项式，所谓乘法和一般多项式的乘法相似，但系数的乘和加都是在域 F 上的。多项式乘法满足结合律，交换律和分配率。单位元为 1。结构< F[X]|k, +,∗>是一个交换环。

称 m（x）为 GF(p)既约多项式，当且仅当在域 GP（p）上不存在这样的多项式 a(x)和 b(x)，使得 $d(x) = a(x) \times b(x)$，这里 a(x)和 b(x)的次数均大于 0。

乘法逆元可以通过扩展的欧几里乘法求得(见[])，当仍要注意的是，求逆运算中设计的乘法，加法，减法，求商，求模等运算都是在域 F 中的运算，加法和乘法定义已经知道，那么减法，求商和求模是怎么定义的？减去一个元素等于加上它在加法下的逆元

$$\forall a, b \in F, a - b \coloneqq a + b_{\oplus}^{-1}$$

求商的定义类似多项式除法，但其中涉及到的减法为 F 中的减法，求商的过程结束时也就得到了模。

### 2.4 有限域GF($2^8$)

AES 算法中的许多运算是按字节定义的，一个字节为 8 位。一个字节对应GP(2)上次数小于 8 的多项式 $b_7b_6 \dots b_0 \to b(x)$。而 GF($2^8$)中的所有元素为所有系数在 GF(2)中并小于 8 的多项式。在按位模 2 加运算⊕和乘法⊗运算，一个字节的 256 种可能的取值构成 GF($2^8$)的所有元素。

既约多项式取$m(x) = x^8 + x^4 + x^3 + x + 1$。GF($2^8$)中的运算可用 GF(2)中并小于 8 的多项式的运算来定义。

GF(2)[x]|8 中加法⊕： $c(x) = a(x) + b(x) \Leftrightarrow c_i = a_i \oplus b_i, 0 \le i < n, \oplus$为 GF(2)中的加法，即两个位异或，再次映射到 GF($2^8$)中，即两个字节按位做异或，所以也用符号“⊕“表示。

GF(2) [x]|8 多项式乘法∗：多项式乘法定义为模多想式乘积

$$c(x) = a(x) * b(x) \Leftrightarrow c(x) \equiv a(x) \times b(x) (\mathrm{mod}\ m(x))$$

由数论的一个定理[]，对任意 GF($2^8$)元素 b 对应的多项式 b(x)，都存在 b(x)和 c(x)，使得

$$a(x)b(x) + c(x)m(x) = 1$$

两边模 m(x)有，$b^{-1}(x) = a(x) mod(m(x))$，由扩展的欧几里乘法可以求出a(x)。

可以证明 GF(2)[x]|8 的多项式关于上面定义的运算确实够了域 GF($2^8$)。

### 2.5 GF($2^8$)上的多项式GF($2^8$)[x]|4

AES 中也有一些运算是按 4 字节长度的字定义的。字对应 GF($2^8$)中的多项式，多项式的系数可以定义为 GF($2^8$)中的元素。通过这种方法，一个 4 字节的字对应于一个次数小于 4 的多项式。

GF($2^8$)[x]|4 加法⊕：对相应系数的简单相加可以实现多项式加法。由于 GF($2^8$)中的加法为按位模 2 加，

所以两个 4 字节的加法就是按位模 2 加，同样用符号“⊕“表示。

GF($2^8$)[x]|4 乘法**：设$a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0$，$a(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0$

$$c(x) = a(x) ** b(x) = c_6x^6 + c_5x^5 + \cdots + c_1x^1 + c_0$$

其中$c_i = \sum_{j=0}^{i} a_j * b_{i-j}$，即合并次数次数相同项的系数，其中的加法为 GF(2)[x]|8 中加法⊕，乘法为 GF(2)[x]|8 多项式乘法*。

C(x)不再可以表示一个 4 字节的字。通过对 c(x)模一个 4 次多项式求余可以得到一个次数小于 4 的多项式。在 AES 中，这一模多项式为$M(x) = x^4 + 1$。M(x)不是一个集约多项式，所以 GF($2^8$)[x]|4 的元素只能构成一个多项式环。

## 3 AES 设计

### 3.1 AES的结构

Rijndael 是一个密钥迭代分组密码，包含了轮变换对状态的重复作用。用 $N_r$ 表示轮数，它依赖于分组长度和密钥长度。

Rijndael 的加密过程包括一个初始密钥加法，记作 AddRoundKey，接着进行 $N_r - 1$ 次轮变换 Round，最后再使用一个轮变换 FinalRound。初始的密钥加法和每个轮变换均以状态 State 和一个轮密钥作为输入。第 i 轮的轮密钥记为 ExpandedKey[i]，初始密钥加法的输入记为 ExpandedKey[0]。从 CipherKey 导出 ExpandedKey 的过程记为 KeyExpansion。Rijndael 的高级语言伪 C 符号描述见列表 3.1。

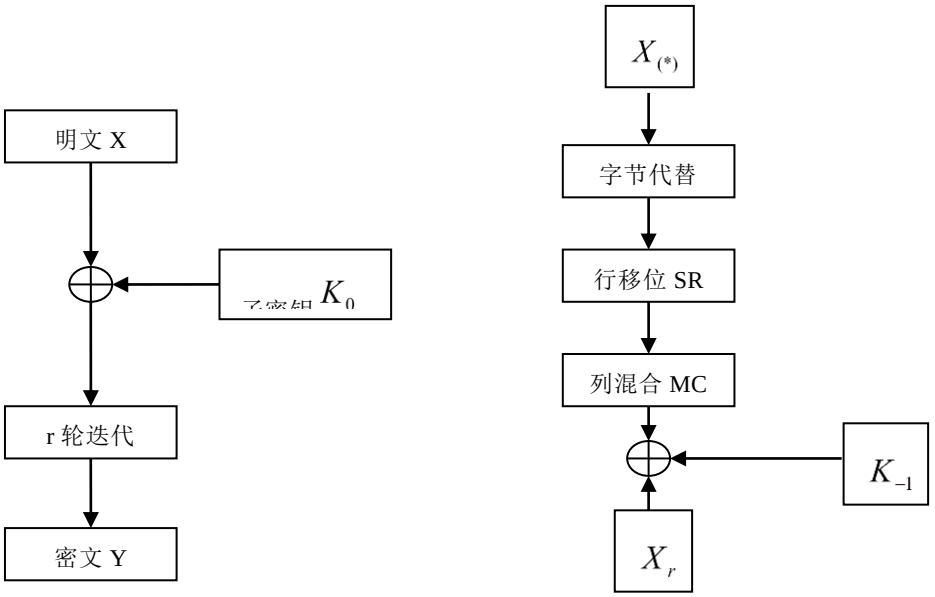
表 1　　使用 Rijndael 进行加密的高级算法

```
Rijndael （State, Cipherkey）
{
KeyExpansion ( CipherKey, ExpandedKey);
AddRoundKey ( State,ExpandedKey[0]);

For (i=1;i< Nr ;i++) Round ( State,ExpandedKey[i] );

FinalRound ( State, ExpandedKey[ Nr ] );

}
```

### 3.2 AES算法的框架描述

Rijndael 算法是一个可变数据块长和可变密钥长的分组迭代加密算法，数据块长和密钥长可分别为 128，192 或 256 比特，但为了满足 AES 的要求，分组长度为 128 比特，密钥长度为 128，192 或 256 比特。AES 密码算法采用的是代替一置换网络(SPN)结构，每一轮操作由 4 层组成：第 1 层(字节替换)为非线性层，用 S 盒对每一轮中的单个字节分别进行替换；第 2 层(行移位)和第 3 层(列混合)是线性混合层，对当前的状态阵按行移位，按列混合；第 4 层(密钥加层)用子密钥与当前状态阵进行字节上的异或。

具体算法结构如图 1 所示。




(a)AES 算法框　　(b)一轮 AES 结

图 1　AES 算法结构

图 1 中，(a)图给出了算法的整体结构，输入明文 x 与子密钥 I(0 异或，然后经过 r 轮迭代最终生成密文 Y，其中第 1 到 r 一 1 轮迭代结构为图(b)，第 r 轮与前面各轮稍微有点不同，缺少混合层。

**3.3** AES加、解密的输入/输出

Rijndael 的输入/输出可看作 8 位字节的一维数组。对加密来说，其输入是一个明文分组和一个密钥，输出是一个密文分组。对解密而言，输入是一个密文分组和一个密钥，而输出是一个明文分组。Rijndael 的轮变换及其每一步均作用在中间结果上，我们将该中间结果称为状态。状态可以形象地表示为一个矩形的字节数组，该数组共有 4 行。状态中的列数记为 $N_b$，它等于分组长度除以 32.将明文分组记为

$$P_0P_1P_2P_3\cdots P_{4\cdot N_b-1}$$

其中，$P_0$ 表示第一个字节，$P_{4\cdot N_b-1}$ 表示明文分组的最后一个字节。类似地，将密文分组记为

$$C_0C_1C_2C_3\cdots C_{4\cdot N_b-1}$$

将状态记为

$$a_{i,j},\ 0\le i<4,\ 0\le j<N_b$$

这里，$a_{i,j}$ 表示位于第 i 行第 i 列的字节。输入字节依次映射到状态字节 $a_{0,0}a_{1,0}a_{2,0}a_{3,0}a_{0,1}a_{1,1}a_{2,1}a_{3,1},\cdots$

上。

当加密是，输入是一个明文分组，映射是

$$a_{i,j} = P_{i+4j}，\ 0 \le i < 4，\ 0 \le j < N_b$$

当解密是，输入是一个密文分组，映射是

$$a_{i,j} = C_{i+4j}，\ 0 \le i < 4，\ 0 \le j < N_b$$

在加密结束时，密文分组以相同的顺序从状态字节中取出

$$C_i = a_{i \bmod 4, i/4}，\ 0 \le i < 4N_b$$

在解密结束时，明文分组按以下顺序从状态中得到

$$P_i = a_{i \bmod 4, i/4}，\ 0 \le i < 4N_b$$

类似地，密钥被映射到两维密码密钥上。密码密钥可以形象地表示为一个与状态类似的矩形数组，该数组也有 4 行。密码密钥的列数记为 $N_k$，它等于密钥长度除以 32。密钥的各字节被依次映射到密码密钥的各字节上：$k_{0,0}k_{1,0}k_{2,0}k_{3,0}k_{0,1}k_{1,1}k_{2,1}k_{3,1}k_{0,2},\cdots$。如果将密钥记为

$$Z_0 Z_1 Z_2 Z_3 \cdots Z_{4 \cdot N_k - 1}$$

那么

$$k_{i,j} = z_{i+4j}，\ 0 \le i < 4，\ 0 \le j < N_k$$

状态与密码密钥的表示以及明文-状态与密钥-密码密钥的映射，如图 2 所示

| | | | |
|---|---|---|---|
| $P_0$ | $P_4$ | $P_8$ | $P_{12}$ |
| $P_1$ | $P_5$ | $P_9$ | $P_{13}$ |
| $P_2$ | $P_6$ | $P_{10}$ | $P_{14}$ |
| $P_3$ | $P_7$ | $P_{11}$ | $P_{15}$ |

| | | | | | |
|---|---|---|---|---|---|
| $k_0$ | $k_4$ | $k_8$ | $k_{12}$ | $k_{16}$ | $k_{20}$ |
| $k_1$ | $k_5$ | $k_9$ | $k_{13}$ | $k_{17}$ | $k_{21}$ |
| $k_2$ | $k_6$ | … | $k_{14}$ | $k_{18}$ | $k_{22}$ |
| $k_3$ | $k_7$ | $k_{11}$ | $k_{15}$ | $k_{19}$ | $k_{23}$ |

图 2

当 $N_b$=4、 $N_k$=6 时状态和密码密钥的大致分布

## 4 AES 加密

### 4.1 轮变换

轮变换 Round 由 4 个变换组成，其中的每个变换称为步骤，如表 2 所示。该密码的最后一轮 FinalRound 稍有不同，也如列表 3.2 所示。在这个列表中，变换（Round，SubBytes，ShiftRows,…）作用在指针（State，ExpndedKey[i]）所指向的数组上。容易验证：若将 Round 变换中的 MixColumns 步骤去掉，则它等价于

FinalRound 变换。

表 2 Rijufael 的轮变换

```
Round （ State, ExpandedKey[i]）
{
   SubBytes ( State );
   ShiftRows ( State );
   MixColumns ( State );
   AddRoundKey ( State , ExpandedKey[i] );
}

FinalRound ( State , ExpandedKey[ Nr ] );

{
   SubBytes ( States );
   ShiftRows( States );

   AddRoundKEY( States, ExpandedKey[ Nr ] );

}
```

**4.2 密钥扩展**(Key Expansion)

为了防止已有的密码分析攻击，AES 使用了与轮相关的轮常量 (Rcon[j]，是一个字，这个字的右边三个字节总为 0)防止不同轮中产生的轮密钥的对称性或相似性。AES 在加密和解密算法使用了一个由种子密钥字节数组生成的密钥调度表，AES 规范中称之为密钥扩展例(KeyExpansion)。密钥扩展例程从一个原始密钥中生成多重密钥以代替使用单个密钥大大增加 了比特位的扩散，在 AES 密钥扩展算法的输入值是 4 字密钥 ，输出是一个 44 字的一一维线性数组。这足以为初始轮密钥扩展例程阶段和算法中的其他 10 轮中的每一轮提供 16 字节的轮密钥。

通过生成器产生 $N_r+1$ 轮轮密钥，每个轮密钥由 $N_b$ 个字组成，共有 $N_b(N_r+1)$ 个字 W[i]，i=0，1，…，$N_b(N_r+1)-1$。

在加密过程中，需要 $N_r+1$ 个子密钥，需要构造 $4(N_r+1)$ 个 32 位字。Rijndael 的密钥扩展方案的伪码描述如下：

```
KeyExpansion(byte key[4*Nk],word w[Nb*(Nr+1)],Nk)
{//Nk 代表以 32 位字为单位的密钥的长度，及 Nk=密钥长度/32
begin
   i=0
   while(I<Nk)
   w[i]=word[key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]]
   i=i+1
end while
i=Nk
while(i<Nb*(Nr+1))
    word temp=w[i-1]
```

```
    if (I mod Nk=0)
       temp=SubWord(RotWord(temp))xor Rcon[i/Nk]
    else if (Nk=8 and I mod Nk=4)
       temp=SubWord(temp)
end if
w[i]=w[i-Nk] xor temp
i=i+1
end while
end
}
```

其中，key[]和 w[]分别用于存储扩展前，扩展后的密钥。SubWord()、RotWord()分别是与 S 盒的置换和以字节为单位的循环移位。$Rcon[i]=(RC[i],'00','00','00')$，$RC[0]='01'$，$RC[i]=2\cdot(RC[i-1])$。

前 10 个轮常数 RC[i]的值（用十六进制表示）如表 5-14 所示，其对应的 Rcon[i]如表 5-15 所示。

表 3　RC[i]

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| RC[i] | 01 | 02 | 04 | 08 | 10 | 20 | 40 | 80 | 1b | 36 |

表 4　Rcon[i]

| I | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Rcon[i] | 01000000 | 02000000 | 04000000 | 08000000 | 10000000 |
| I | 6 | 7 | 8 | 9 | 10 |
| Rcon[i] | 20000000 | 40000000 | 80000000 | 1b000000 | 36000000 |

输入密钥直接被复制到扩展密钥数组的前四个字中，得到 w[0]、w[1]、w[2]、w[3]；然后每次用四个字填充扩展密钥数组余下的部分。在扩展密钥数组中，w[i]的值依赖于 w[i-1]和 w[i-4]（i ≥4）。

对 w 数组中下标不为 4 的倍数的元素，只是简单地异或，其逻辑关系为：$w[i]=w[i-1]\oplus w[i-4]$（i 不为 4 的倍数）

对 w 数组中下标为 4 的倍数的元素，采用如下的计算方法：

将一个字的四个字节循环左移一个字节，即将字$[b_0,b_1,b_2,b_3,]$变为$[b_0,b_1,b_2,b_3,]$；

基于 S 盒对输入字中的每个字节进行 S 代替；

将步骤 1 和步骤 2 的结果再与轮常量 Rcon[i]相异或。

### 4.3 字节替换(SubBytes)

AES 定义了一个 S 盒，State 中每个字节按照如下方式映射为一个新的字节：把该字节的高 4 位作为行值，低 4 位作为列值，然后取出 S 盒中对应行和列的元素作为输出。例如，十六进制数{84}。对应 S 盒的行是 8 列是 4，S 盒中该位置对应的值是{5F}。

S 盒是一个由 16x16 字节组成的矩阵，包含了 8 位值所能表达的 256 种可能的变换。S 盒按照以下方式构造：

逐行按照升序排列的字节值初始化 S 盒。第一行是{00}，{01}，{02}，…，{OF}；第二行是{10}，{l1}，…，{1F}等。在行 X 和列 Y 的字节值是{xy}。

把 S 盒中的每个字节映射为它在有限域 GF($2^k$)中的逆。GF 代表伽罗瓦域，GF($2^8$)由一组从 0x00 到 0xff

的 256 个值组成，加上加法和乘法。 $GF(2^8)=\frac{Z2[X]}{(x^8+x^4+x^3+x+1)}$ 。{00}被映射为它自身{00}。

把 S 盒中的每个字节记成 $(b_8,b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0)$ 。对 S 盒中每个字节的每位做如下变换：

$$bi'=b_i\oplus b_{(i+4)\bmod 8}\oplus b_{(i+5)\bmod 8}\oplus b_{(i+6)\bmod 8}\oplus b_{(i+7)\bmod 8}\oplus c_i$$

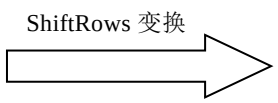
上式中 $c_i$ 是指值为{63}字节 C 第 i 位，即 $(c_8,c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0)=(01100011)$ 。 符号(')表示更新后的变量的值。AES 用以下的矩阵方式描述了这个变换：

$$\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}+\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

**4.4 行位移变换**(ShiftRows)

State 的第一行字节保持不变，State 的第二行字节循环左移一个字节，State 的第三行字节循环左移两个字节，State 的第四行循环左移三个字节。变化如图 2 所示。

| ShiftRows 之前 | | | |
|---|---|---|---|
| 14 | 0 | 5d | ab |
| 78 | 10 | c1 | fd |
| 31 | 9 | 11 | 3f |
| 28 | 0b | 2a | 45 |




| ShiftRows 之后 | | | |
|---|---|---|---|
| 14 | 0 | 5d | ab |
| 10 | c1 | fd | 78 |
| 11 | 3f | 31 | 9 |
| 45 | 28 | 0b | 2a |

图 5　ShiftRows 变换

**4.5 列混合变换**(MixColumns)

列混合变换是一个替代操作，是 AES 最具技巧性的部分。它只在 AES 的第 0，1，…，Nr 一 1 轮中使用，在第 N r 轮中不使用该变换。乘积矩阵中的每个元素都是一行和一列对应元素的乘积之和。在 MixColumns 变换中，乘法和加法都是定义在 GF( $2^8$ )上的。State 的每一列 ( $b_{i,j}$ ) i=0，…,3；J=0，…， $L_b$ 被理解为 GF( $2^8$ ) 上的多项式，该多项式与常数多项式 $a(x)=a_3x^3+a_2x^2+a_1x+a_0$ 相乘并模 $M(x)=x^4+1$ 约化。

这个运算需要做 GF( $2^8$ )上的乘法。但由于所乘的因子是三个固定的元素 02、03、01，所以这些乘法运算仍然是比较简单的（注意到乘法运算所使用的模多项式为 $m(x) = x^8 + x^4 + x^3 + x + 1$）。设一个字节为

b=(b7b6b5b4b3b2b1b0)，则

b×‘01’=b；

b×‘02’=(b6b5b4b3b2b1b00)+(000b7b70b7b7)；

b×‘03’=b×‘01’+b×’02’。

（请注意，加法为取模 2 的加法，即逐比特异或）

写成矩阵形式为：

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

**4.6 密钥加变换**(Add RoundKey)

Add RoundKey 称为轮密钥加变换，128 位的 State 按位与 128 位的密钥 XOR：$(b_{0j}, b_{1j}, b_{2j}, b_{3j}) \leftarrow (b_{0j}, b_{1j}, b_{2j}, b_{3j}) \oplus (k_{0j}, k_{1j}, k_{2j}, k_{3j})$对 j=0，…，L-1 轮密钥加变换很简单，却影响了 State 中的每一位。密钥扩展的复杂性和 AES 的其他阶段运算的复杂性，却确保了该算法的安全性。

**4.7** AES**解密**

解密算法可通过直接利用步骤 InvSubBytes、InvShiftRows、InvMixCloumns 和 AddRoundKey 的逆并倒置其次序而得到，此算法称为直接解密算法。在这个算法中，不仅步骤本身与加密不同，而且步骤出现的顺序也不相同。为了便于实现，通常将惟一的非线性步骤（SubBytes）放在轮变换的第一步（见第四章），我们在设计时已经考虑到这一点。Rijndael 的结构使得有可能定义一个等价的解密算法，其中所使用的步骤次序与加密相同，只是将每一步改成它的逆，并改变密钥编排方案。注意这种结构上的一致性不同于采用 Feistel 结构的许多密码中的组件和结构的一致性，IDEA 也是一样

## 5 接口设计和实现

该 AES 首先在使用 C++主要的算法接口，并导出为 dll，最后用 c#生成界面，界面如图 6。


图 6

为了区分加密解密，一般函数中参数 Mode 为加密模式，mode 为 1 时加密，mode 为 0 时解密。

### 5.1 组加密encript()

接口：

void encript(word8 state[][NB],const word8 cipherkey[])

对分组数据 state 加密，加密后的密文还是存放在 state 中

输入：state--组,4*NB 长度的字符数组，cipherkey--密匙，4*NB 长度的字符数组

加密步骤：

1）产生扩展密匙 keyExpasion(cipherKey,expandedKey);

2）对组进行一次密匙加法 addRoundKey(state,expandedKey,roundCount);

3）进行 NR-1 次轮变换 for(roundCount=1;roundCount<NR;roundCount++)//

round(state,expandedKey,roundCount);

4）最终轮变换 finalRound(state,expandedKey);

**5.2 密匙扩展**keyExpasion()

接口：

void keyExpasion(const word8 cipherKey[],word8 expandedKey[][(1+NR)*NB])

从密匙产生扩展密匙，扩展密匙的长度为 4*(1+NR)*NB，对应 NR-1 次轮变换（round（））和一个最终变换（finalRound （）），第 i 次轮变换使用的密匙为 expandedKey 数组中第 i 个 4*NB 部分

输入：cipherKey--4*NK 长度的字符数组，用户输入的密码

expandedKey--4*(1+NR)*NB 长度的字符数组，存放产生的扩展密匙

实现：

1）expandedKey 前 NK 列为密匙 cipherKey 的前 NK 列，

```
for(j=0;j<NK;j++)
        for(i=0;i<4;i++)
            expandedKey[i][j]=cipherKey[4*j+i];
```

2) 列 i>=NK 时，若 i 不是 NK 的倍数，则 expandedKey[i]是 expandedKey[i-NK]和 expandedKey[i-1]的按字节异或；若 i 是 NK 的倍数，则需把 expandedKey[i-1]列内的 字符列内移动后再字节变换，expandedKey[i-1]列的第一个数还得和一个轮常量 CR 异 或一下，轮常量 CR 的定义为递推形式：CR[1]=01，CR[2]=02，CR[i]=x*CR[i-1],注意

这里的乘法为域 L 中的乘法，规划多项式为 0x11b

```
if(j%NK==0)
        {

            expandedKey[0][j]=expandedKey[0][j-NK]^sTra(expandedKey[1][j-1])^CR;
            for(i=1;i<4;i++)
            {
            expandedKey[i][j]=expandedKey[i][j-NK]^sTra(expandedKey[(i+1)%4][j-1]);
            }

            if( HIGHESTBIT8(CR))
                CR=(CR<<1)^0x1b;
            else
                CR<<=1;
        }
        else
        {
            for(i=0;i<4;i++)
                expandedKey[i][j]=expandedKey[i][j-NK]^expandedKey[i][j-1];
        }
```

**5.3 轮变换**round()

接口：

void round(word8 state[][NB],word8 expandedKey[][(1+NR)*NB],int roundCount)

对分组 state 进行一次轮变换，变换的后字符依然存放在 state 中

输入：state--组,4*NB 长度的字符数组

expandedKey--4*(1+NR)*NB 长度的字符数组，存放扩展密匙

roundCount--int 型变量，表示正在进行的是第 i 次轮变换

实现：

轮变换包括四个步骤，依次是

1）字节变换

subBytes(state);

2）行移动

shiftRows(state);

3）列混合

mixColumns(state);

4）密匙加法

ddRoundKey(state,expandedKey,roundCount);

**5.4 字节变换**

接口：

void subBytes(word8 state[][NB],bool mode)

对分组 state 进行一次字节变换，state 中的字符 a，字节变换 $f(a(x)) = (u(x)**a^{-1}(x) + v(x)) \bmod (x^8 + 1)$，其中 $u(x) = x^7 + x^6 + x^5 + x^4 + 1$，$v(x) = x^7 + x^6 + x^2 + x$，乘法**和求逆都是有限域 GF( $2^8$ )中的运算，变换的后字符依然存放在 state 原位置处

输入：state--组,4*NB 长度的字符数组

Mode--bool 型变量，mode 为 1 时加密，mode 为 0 时解密

实现：

对 state 每一个字节 a 做字节变换

1）首先求 a 在限域 GF( $2^8$ )中求逆 a=inverseEle(a);

2）然后做仿射变换，a=affineTra(a,mode);

mode 为 0 时解密，则作相反的动作

a=affineTra(a,mode);

a=inverseEle(a);

**5.5 行移动**

接口：

void shiftRows(word8 state[][NB],bool mode)

对分组 state 进行一次行移动，对 state 的第 i 行循环左移动 Ci 个距离，变换的后字符依然存放在 state 中

输入：state--组,4*NB 长度的字符数组

Mode--bool 型变量，mode 为 1 时加密，mode 为 0 时解密

实现：

State 是一个 4 行 NB 列的数组，对 state 的第 i 行循环左移动 Ci 个距离，本算法中取 C0=0，C1=1，C2=2，C3=3。实现是使用一个有趣的算法，第 i 行循环左移动 Ci 个距 离时，首先第一步将第 i 行对称交换（第一个和最后一个交换，第二个和倒数第二个交 换，依次类推），第二步交换前 Ci 个元素，第三步交换后面剩下的元素。

1）exchange(state[i],0,NB-1);

2）exchange(state[i],0,NB-1-offset[i]);

3）exchange(state[i],NB-offset[i],NB-1);

mode 为 0 时解密，则作相反的动作

exchange(state[i],0,NB-1);

exchange(state[i],0,offset[i]-1);

exchange(state[i],offset[i],NB-1);

**5.6 混合变换**

接口：

void mixColumns(word8 state[][NB],bool mode)

对分组 state 进行一次列混合变换，列 $(t_1,t_2,t_3,t_4)^T$ 定义在有限域 GF( $2^8$ )的次数小于 4 的多项式，

$t(x)=t_3x^3+t_2x^2+t_1x+t_0$，列混合变换变换的后字符依然存放在 state 中

输入：state--组,4*NB 长度的字符数组

Mode--bool 型变量，mode 为 1 时加密，mode 为 0 时解密

实现：

列混合运算对一列（也就是四个字符进行变换），这里这四个字节看做定义在有限域 GF( $2^8$ )的次数小于 4 的多项式系数（字节变换中的字符看做 GF( 2 )中次数小于 8 的多项式系数，注意相区别）。列混合运算主要是将当前列与一个 GF( $2^8$ )中的多项式 $C(x)=03x^3+01x^2+01x+02$ 想乘即获得变换结果，这里的乘法***定义为 y1(x)***y2(x)：=y1(x)**y2(x)mod(N(x)),N(x)= $x^4+1$ 。实际实现时，因为 C（x）已经给出，可将结果多项式每一项对应的系数事先计算好存放在数组中，这样计算会更快一点,如下图。

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

定义字符数组 c[2][4][4]存放对应的表，c[mode]对应加密或解密的表，由 mode 决定，mode 为 1 时加密，为 0 时解密。

```
static word8 c[2][4][4]={
        {
        {0x0e,0x0b,0x0d,0x09},
        {0x09,0x0e,0x0b,0x0d},
        {0x0d,0x09,0x0e,0x0b},
        {0x0b,0x0d,0x09,0x0e}
        },
        {
        {0x02,0x03,0x01,0x01},
        {0x01,0x02,0x03,0x01},
        {0x01,0x01,0x02,0x03},
        {0x03,0x01,0x01,0x02}
        }
```

```
};
```

1）对第 i 列进行变换，首先将第 i 列临时保存，

```
for(j=0;j<4;j++)
    s[j]=state[j][i];
```

2）求第 i 列第 j 个元素，mul 就是上面所述的乘法***

```
state[j][i]=0;
for(k=0;k<4;k++)
{
    temp=mul( c[mode][j][k], s[k] ) ;
    state[j][i]^=temp;
}
```

逆元算时对应的表如下

$$\begin{bmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{bmatrix}$$

**5.7 密匙加法**addRoundKey()

接口：

void addRoundKey(word8 state[][NB],word8 expandedKey[][(1+NR)*NB],int roundCount)

对分组 state 进行一次密匙加法，将明文 state 对应的密匙想加，state：=state^expandedKey[][roundCount*NB:oundCount*NB-1]，变换的后字符依然存放在 state 中

输入：state--组,4*NB 长度的字符数组

expandedKey--4*(1+NR)*NB 长度的字符数组，存放扩展密匙

roundCount--int 型变量，表示正在进行的是第 roundCount-次轮变换

实现：

密匙加法的实现很简单，只要组 state 和第 roundCount 次轮变换对应的扩展密码的部分按字节按位异或。

```
for(i=0;i<4;i++)
    for(j=0;j<NB;j++)
        state[i][j]^=expandedKey[i][roundCount*NB+j];
```

**5.8 组解密**invEncript（）

接口：

void invEncript(word8 state[][NK],const word8 cipherKey[])

对分组数据 state 解密，加密后的密文还是存放在 state 中

输入：state--组,4*NB 长度的字符数组，cipherkey--密匙，4*NB 长度的字符数组

实现：

解密过程和加密过程的操作顺序刚好相反

1）密匙扩展 keyExpasion(cipherKey,expandedKey);

2）做 finalRound 的逆操作 invFinalRound(state,expandedKey);

3) 做 NR-1 次逆轮变换

```
for(roundCount=NR-1;roundCount>0;roundCount--)
    invRound(state,expandedKey,roundCount);
```

4）最后一次密匙加法 addRoundKey(state,expandedKey,0)；

### 5.9 逆轮变换

接口：

void invRound(word8 state[][NB],word8 expandedKey[][(1+NR)*NB],int roundCount)

输入：state--组,4*NB 长度的字符数组

expandedKey--4*(1+NR)*NB 长度的字符数组，存放扩展密匙

roundCount--int 型变量，表示正在进行的是第 i 次逆轮变换

实现：

操作的顺序和 round（）相反

1）密匙加法 addRoundKey(state,expandedKey,roundCount);

2）逆方向列混合变换 mixColumns(state,0);

3）逆方向行移动 shiftRows(state,0);

3）逆方向的字节变换 subBytes(state,0);

### 5.10 文件加密接口

void encriptFile(char inFile[],char outFile[],const word8 cipherkey[],const bool mode)

密匙长度为 16 个字节，使用该函数前请检验

mode==1 时加密，mode==0 时解密

inFile 为需要加密的文件名（包含路径）

outFile 为加密的后的文件名（包含路径）

## 6 几种重要运算的设计和实现

### 6.1 x乘xtime()

将 2 和 GF( $2^8$ )的元素 a 的乘法定义为 x 乘,2**a=2*a mod （0x11b），,就是将 a 的位向左移动以后得到 2*a，然后减去 0x11b，这里的减法为加法（按位异或）的逆运算，也还是按位异或，而 a 的最高位也就是左移出去的那一位 a(7)。

如果 a(7)=0，则 2*a 对应多项式的次数小于 8，2**a=2*a，

如果 a(7)=1，则 2*a 需和 0x11b 异或得到 2**a，2**a 对应多项式的次数一定是小于 8 的（2*a（8）^0x11b（8）=0），因此左移后的 a（7）已经不重要了，这样，2**a=2*a ^0x1b

GF( $2^8$ )是域，因此 GF( $2^8$ )的加法^和乘法**满足 a**（b^c）=(a**b)^(a**c)，这是域的性质

而 4**a=2**2**a=2**xtime(a)=xtime(xtime(a))，由归纳法，可以求出任意的 $2^k$ **x，只需 k 次的迭代 xtime(a)运算。

实现:

```
b=a & 0x80;//取得 a 的最高位
a<<=1;
if(b>0)
    a^=0x1b;
return a;
```

### 6.2 域GF( $2^8$ )中的乘法运算mul（a，b），取不可约多项式0x11b

给出 GF( $2^8$ )的元素 a、b（也就是字符），假设 a、b 在 GF( 2 )对应多项式为 a(x)、b(x)，

a**b 主要是通过移位和异或操作实现，一个字符的位数为 8，所以要做 8 次移位移位和异或操作实现。记 $b^{(i)}$ 这样一个字符，它的第 i 位和 b 的第 i 位相同，其余位为 0，$b = b^{(0)} \wedge b^{(1)} \wedge ... \wedge b^{(7)}$，又 GF( $2^8$ )是域，因此 GF( $2^8$ ) 的加法∧和乘法 ** 满足 a**（d^c）=(a**d)^(a**c)，这是域的性质，从而得出 a**b= $a**(b^{(0)} \wedge b^{(1)} \wedge ... \wedge b^{(7)}) = (a**b^{(0)}) \wedge (a**b^{(1)}) \wedge ... \wedge (a**b^{(7)})$ ,又 $b^{(i)} = 2^i$，这样 mul（a，b）转换为一系列的 xtime（）

实现：

```
word8 s=0,temp,flag;
int i,j;
for(i =0;i<8;i++)
{
    flag=b&0x1;//取 b 的第 i 位
    b>>=1;
    if(flag)//只有当第 i 位为 1 时才计算
    {
        temp=a;
        for(j=0;j<i;j++)//第 i 位迭代 i 次
            temp=xtime(temp);
        s^=temp;
    }

}

return s;
```

**6.3** 域GF(2)上的多项式乘法polyMul(a，b)

通过移位和异或可以实现 GF(2)域上 a(x)和 b(x)的乘法，这个很简单，不再赘述。

```
Word32 s=0,t=0,flag;
    int i;
    t^=a;
    for(i=0;i<16;i++)
    {
        flag=b&0x1;//取 b 的第 i 位
        b>>=1;
        if(flag)//b 的第 i 位为 1，s 和 2^i 异或
        {
            s^=t;
        }
        t<<=1;
    }
    return s;
```

**6.4** 域GF(2)上的多项式求商polyDiv(a，b)

算法实现时事先必须知道 a 和 b 最高非 0 位是哪一位，由函数 highestNZP（）实现，设 k1、k2 为 a 和 b 最高

非 0 位，当 k1>=k2 也就被除数 a 大于除数 b 时，计算 t=k1-k2，将 b 左移 t 位得到 temp，然后 a 减去 temp 也就是 a 和 temp 做异或，重复直到 k1<k2 也就被除数 a 小于除数 b，这和我们做除法的过程是一致的。

```
k1=highestNZP(a);
k2=highestNZP(b);
while( k1>=k2 )
{
    t=k1-k2;
    temp=b<<t;
    bu+=(1<<t);
    a^=temp;
    k1=highestNZP(a);
}
return bu;
```

**6.5** GF( $2^8$ )上求逆元inverseEle（）

GF( $2^8$ )如上文定义，乘法**和加法^也都是 GF( $2^8$ )中的运算。求 a 的逆元 b，b 满足 a**b=1，记 0X11B 对应的多项式为 M(x)，M(x)和 a(x),所以存在 b(x)和 c(x)，满足 a(x)**b(x)+c(x)**M(x)=1，则 a(x)**b(x)modM(x)=1，b(x)即为 a(x)的逆。

对于 a>b >0 成立：

a= b • q +r ， 0≤r < b， gcd(a, b ) = gcd ( b, r)。

$$
\begin{aligned}
&a(x) = c_a(x)b(x) + r_1(x)\\
&b(x) = c_b(x)r_1(x) + r_2(x)\\
&r_1(x) = c_1(x)r_2(x) + r_3(x)\\
&\cdots\\
&r_i(x) = c_i(x)r_{i+1}(x) + r_{r+2}(x)\\
&\cdots
\end{aligned}
\qquad (1)
$$

由上面的推导有

$$
\begin{aligned}
&a(x) = 1\cdot a(x) + 0\cdot b(x) = p_a(x)a(x) + q_a(x)b(x), \text{其中} p_a(x) = 1,\ \ q_a(x) = 0\\
&b(x) = 0\cdot a(x) + 1\cdot b(x) = p_b(x)a(x) + q_b(x)b(x), \text{其中} p_{\mathrm{b}}(x) = 0,\ \ q_b(x) = 1\\
&r_1(x) = a(x) - c_a(x)\cdot b(x) = p_1(x)a(x) + q_1(x)b(x)\\
&\text{其中} p_1(x) = p_a(x) - c_a(x)\cdot p_b(x),\ \ q_1(x) = q_a(x) - c_a(x)\cdot q_b(x)
\end{aligned}
$$

由归纳法可以得到 $r_{i-1}(x), r_i(x)$ 的关于 $a(x)$、$b(x)$ 的递推表达式 $r_i(x) = p_i(x)a(x) + q_i(x)b(x)$。假设 k<i+1 时

$r_k(x)$ 的表达式已求出，下求 $r_{i+1}(x)$

由条件

$$
\begin{aligned}
&r_{i-1}(x) = p_{i-1}(x)a(x) + q_{i-1}(x)b(x)\\
&r_i(x) = p_i(x)a(x) + q_i(x)b(x)\\
&r_{i-1}(x) = c_{i-1}(x)r_i(x) + r_{r+1}(x)
\end{aligned}
$$

则可得 $p_{i+1}(x) = p_{i-1}(x) - c_{i-1}(x)\cdot p_b(x)$，$q_{i+1}(x) = q_a(x) - c_{i-1}(x)\cdot q_b(x)$，其中 $c_{i-1}(x)$ 为多项式 $r_{i-1}(x)$ 除

$r_i(x)$ 的商。当 $a(x)$、$b(x)$ 互素时，存在 i 使得 $r_i(x)=1=p_i(x)a(x)+q_i(x)b(x)$，此时 $p_i(x)$ 即为 $a(x)$ 的逆元素。

将上面的算法推导应用到本问题中，则可以得到求 a(x)在 GF( $2^8$ )关于 M(x)的逆的算法。

现将算法描述如下,

记 0X11B 对应的多项式为 M(x)，M(x)和 a(x)对应的多项式

```
X1=1,X2=0,X3=0x11B;\\X3 为 M(X)，X1 为 M(x)的系数，X2 为 a(x)的系数
                    \\X3=X1**M+X2**a
Y1=0,Y2=1,Y3=a;\\Y3 为 a(X)，Y1 为 M(x)的系数，Y2 为 a(x)的系数
                \\Y3=Y1**M+Y2**a
T1,T2,T3,BU;
while(Y3!=1)
    {//迭代，知道 ri(x) ==1，此时 Y2 即 a(x)的系数为 a（x）的逆
        BU=polyDiv(X3,Y3);
        T1=X1^polyMul(BU,Y1);
        T2=X2^polyMul(BU,Y2);
        T3=X3^polyMul(BU,Y3);

        X1=Y1;X2=Y2;X3=Y3;
        Y1=T1;Y2=T2;Y3=T3;
    }
return  Y2
```

**6.6 仿射变换**affineTra()

接口：

word8 affineTra(word8 a,bool mode)

对字符 a 仿射变换，返回变换后的值

参数：a--需要做仿射变换的字符

mode--bool 型变量，mode 为 1 时加密，mode 为 0 时解密

实现：

仿射变换即将 a(x)和多项式 $u(x)=x^7+x^6+x^5+x^4+1$ 相乘再和 $v(x)=x^7+x^6+x^2+x$ 相加的结果返回，这是 GF( $2^8$ )中的乘法， $f(a(x))=(u(x)**a\ (x)+v(x))\bmod(x^8+1)$ 。但这里的乘法和上面有些不同，规约多项式取为 $x^8+1$，为了提高计算速度我们这里使用表的方法，需事先计算出多项式系数，如下图。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

```
U[2][8]={ {0xa4,0x49,0x92,0x25,0x4a,0x94,0x29,0x52},
          {0xf1,0xe3,0xc7,0x8f,0x1f,0x3e,0x7c,0xf8} };
V[2][1]={{0x5},{0x63}};
    int i,j;
    for(i=7;i>=0;i--)//计算 bi
    {
        b<<=1;
        temp=0;
        at=a;
        ut=u[i];
        for(j=0;j<8;j++)////计算 bi,U 表的第 i 行和 a 做乘法，实际上求 b（x）第 i 项的系数
        {
            temp^=HIGHESTBIT8(at)*HIGHESTBIT8(ut);
            at<<=1;
            ut<<=1;
        }
        if( temp>0 )
        {
            b|=0x1;
        }
    }
    return b^v;
```

## 7 算法运行时间的改进

### 7.1 运行时间的分析

该算法完成时，加密速度相当慢，分析其运算时间，主要的时间花费在字节变换、列混合变换。观察原算法描述，发现分组变换中有一个生成扩展密匙过程，但对整个文件加密时，文件被分割成为很多分组，这样每次对分组加密都会生成一次扩展密匙，实际上扩展密匙仅和输入密匙有关，多次生成是没有必要的，这个地方可以改进。

### 7.2 字节变换算法的改进

建议用长度为 256 的线性表。产生一个更高效且更常采用的方法是将字节变换变换做成查表的形式，一个字节有 8 位，共有 $2^8$ 的字符，且字节变换和密匙无关，所以一个确定的字符字节变换的结果是确定的。可

利用求逆函数（inverse()）和仿射变换函数（affineTra()）生成字节变换表 SBOX 和逆字节变换表 invSBOX。具体的过程是这样的，首先定义一个 256 个元素的表 SBOX，从小到大遍历所有的 unsigned char 型变量，不妨设其为 $a=a^{(7)}a^{(6)}...a$，$a^{(i)}$ 表示 a 二进制表示的第 i 位，则 a 变换后存放在表 SBOX 的 $a^{(7)}a^{(6)}a^{(5)}a^{(4)}*4+a^{(3)}a^{(2)}a^{(1)}a^{(0)}$ 位，a 高四位和低四位可通过移位获得，而 a 的变换为先求逆再做仿射变换，所以变换之后的 a：=affineTra(nverse(a))。逆字节变换表 invSBOX 也可以由类似的方法获得。这两个表的具体形式见附录，查表时取字节的高 4 位做行号，低四位做列号。改进后的字节变换如下

1）求列号 cNr=state[i][j]&0xf;

2）求行号 rNr=(state[i][j]>>4)&0xf;

3）字节变换 state[i][j]=SBOX[rNr*16+cNr];

### 7.3 列混合变换的改进

列混合变换也可以改进为查遍形式，一个直接的思路是遍历一个列生成一整个表，这样表的大小为 $2^{32}$ 个字节，明显太大。观察列混合变换的过程，记列混合的输入 **c**，输出为 **d**，则

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 03 & 01 \\ 03 & 01 & 01 & 02 \end{bmatrix} . \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}, 0 \le j \le N_b$$

上述矩阵运算可以看做是 4 个列向量的线性组合：

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} . c_{0,j} + \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} . c_{1j} + \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} . c_{2j} + \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} . c_{3,j}, 0 \le j \le N_b$$

定义 4 个 T 表，$T_0$、$T_1$、$T_2$ 以及 $T_3$

$$T_0[a] = \begin{bmatrix} 02 \bullet a \\ 01 \bullet a \\ 01 \bullet a \\ 03 \bullet a \end{bmatrix}, T_1[a] = \begin{bmatrix} 03 \bullet a \\ 02 \bullet a \\ 01 \bullet a \\ 01 \bullet a \end{bmatrix}, T_2[a] = \begin{bmatrix} 01 \bullet a \\ 03 \bullet a \\ 02 \bullet a \\ 01 \bullet a \end{bmatrix}, T_3[a] = \begin{bmatrix} 01 \bullet a \\ 01 \bullet a \\ 03 \bullet a \\ 02 \bullet a \end{bmatrix}$$

每个 T 表都有 256 个 4 字节的条目，从而需要 4KB 的存储空间，使用这些表，可以将 **d** 表示为

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0(c_{0,j}) + T_1(c_{1,j}) + T_2(c_{2,j}) + T_3(c_{3,j})$$

也就是说只要求出这个 4 个 T 表，就可以通过简单的查表和域上的加法（异或）就出一个列列混合变换之后的结果。生成 T 表使用域 GF( $2^8$ )中的乘法(mul（）)，$T_k[i][j] = mul(C[i][j], a)$，其中表 C 为：

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 03 & 01 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

列混合变换的对 state 第 j 列第 i 行元素的变换成为：
state[j][i]=T[0][j][s[0]]^T[1][j][s[1]]^T[2][j][s[2]]^T[3][j][s[3]]。

### 7.4 扩展密匙的改进

将扩展密匙的过程（keyExpasion()）从分组加密的过程移到问价加密的过程，在分组加密的接口增加参加参数 expandedKey，expandedKey 是生成的扩展密匙。改变后的分组加密接口：
void encript(word8 state[][NB],word8 expandedKey[][(1+NR)*NB])
对分组数据 state 加密，加密后的密文还是存放在 state 中
输入：state--组,4*NB 长度的字符数组，expandedKey--扩展密匙，4*NB 长度的字符数组。

### 7.5 该进前后的速度比较

表 1

| 加密时间/毫秒<br>文件大小/MB | 1.49 | 6.72 | 115 |
|---|---|---|---|
| 改进前 | 55372 | --- | --- |
| 改进后 | 431 | 1926 | 32956 |

## 8 结束语

本文从数学基础的知识上阐明了 AES 算法的四个步骤。介绍算法的主要步骤并实现该算法，特别了讨论一些域中的基本运算，同时对算法的效率做了有效改进。

**致谢** 在此，首先感谢南京大学计算机系黄浩教授，这篇论文是在他的指导下完成的，黄老师给出很多指导性的建议，并对文中的一些错误做了修改。同时感谢南京大学计算机系 427 实验的瞿飞同学，他完成加密软件界面部分的工作.

**References**:

[1] 王衍波，薛通．应用密码学[M]．北京：机械工业出版社，2003.

[2] 沈世镒，陈鲁生．信息论与编码理论[M]．北京：科学出版社，2002．114—126

[3] 谷大武，徐胜波译.高级加密标准(AE)s 算法—Rijndael 的设计.北京:清华大学出版社，2003:111 一 133 页

[4] 沉鲁生,沉世镒.现代密码学.北京：科学出版社,2002