高级加密标准(AES)

目　录

1．引言.....4

2．定义.....4

2.1 术语和缩写词表.....4
2.2 算法参数、符号和函数.....5

3．符号和惯例.....6

3.1 输入和输出.....6
3.2 字节(Bytes).....6
3.3 字节数组.....7
3.4 状态(State).....8
3.5 将状态看作列数组.....9

4．数学基础.....9

4.1 加法.....9
4.2 乘法.....9
4.2.1 乘 x.....10
4.3 系数在 GF(2^8)中的多项式.....10

5．算法说明.....12

5.1 加密.....13
5.1.1 字节替代(SubBytes( ))变换.....13
5.1.2 行移位(ShiftRows( ))变换.....15
5.1.3 列混合(MixColumns( ))变换.....16
5.1.4 轮密钥加(AddRoundKey( ))变换.....16
5.2 密钥扩展.....17
5.3 解密.....18
5.3.1 逆行移位(InvShiftRows( ))变换.....19
5.3.2 逆字节替代(InvSubBytes( ))变换.....20
5.3.3 逆列混合(InvMixColumns( ))变换.....20
5.3.4 轮密钥加(InvMixColumns( ))变换的逆变换.....21
5.3.5 等价的解密变换.....21

6．实现方面的问题.....22

6.1 密钥长度要求.....22
6.2 密钥限制.....22
6.3 密钥长度，分组大小和轮数的参数化.....23
6.4 针对不同平台的实现建议.....23

图表目录

图表 1　比特类型的 16 进制表示.....7
图表 2　字节和比特编号.....8
图表 3　状态矩阵、输入和输出.....8
图表 4　Key-Block-Round 组合.....12
图表 5　加密算法伪代码.....13
图表 6　SubBytes（）：作用在状态的单个字节上.....14
图表 7　S 盒：字节 xy（16 进制格式）的替代值.....15
图表 8　ShiftRows（）在状态的后三行上循环移位.....15
图表 9　MixColumns（）在状态的列上运算.....16
图表 10　AddRoundKey（） 将密钥编排得到的字异或到状态的每一列上.....17
图表 11　密钥扩展伪代码.....18
图表 12　解密算法的伪代码.....19
图表 13　InvShiftRows（）在状态的后三行上循环移位.....20
图表 14　S 盒的逆：字节 xy 的替代值（16 进制格式）.....20
图表 15　等价解密算法的伪代码.....22

1. 引言

该标准详细说明了 Rijndael 算法([3]和[4])，一个对称分组密码算法。可以处理 128 bits 数据分组，使用的密钥长为 128，192 和 256 bits。Rijndael 的设计还可以允许处理其它的分组长度和密钥长度，然而该标准中没有采用。

在本标准的其余部分，本文说明的算法均被称为“AES 算法”。如上所述，该算法可以使用 3 种不同的密钥长度，分别称为“AES-128”、“AES-192”、“AES-256”。

该说明包括下列部分：

2．术语定义，缩写词，算法参数、符号和函数；

3．算法说明中使用的符号和惯例，包括 bits，bytes，words 的排序和编号方式；

4．对理解算法有帮助的数学工具；

5．算法说明，包括密钥扩展、加密和解密程序；

6．实现方面，如密钥长度支持、密钥限制和其它的分组/密钥/轮大小。

该标准以几个附录做为结尾，包括密钥扩展和加密的每一步的示例、加密和解密的示例向量以及参考文献列表（本翻译略去，详见 Fips-197 文档）。

2. 定义

2.1 术语和缩写词表

下面是文中经常出现的术语：

AES	高级加密标准
Affine Transformation	仿射变换，由一个线性变换和一个向量加变换复合而成
Array	包含一定个数的相同实体的集合(如字节数组)。
Bit	一个二进制数（比特），0 或 1。
Block	一个比特序列，如输入、输出、状态和轮密钥。该序列的长度即是其包含的比特个数。分组也可以解释为字节数组。
Byte	字节，一个 8 比特序列。
Cipher	使用密钥将明文变换为密文的一系列变换。
Cipher Key	密码所使用的秘密密钥，在密钥扩展程序中用来生成一系列轮密钥；这些轮密钥可以表示成以字节为元素的矩阵形式，包含 4 行 Nk 列。

Ciphertext	加密算法得到的输出或解密算法的输入。
Inverse Cipher	使用密钥将密文变换为明文的一系列变换。
Key Expansion	密钥扩展算法：将秘密密钥扩展为一系列轮子密钥。
Plaintext	输入到加密算法的数据或从解密算法得到的输出。
Rijndael	该高级加密标准（AES）中描述的密码算法的名字。
Round Key	秘密密钥经过密钥扩展算法得到的一系列子密钥;它们将被应用到加密和解密中的状态上。
State	加密算法的中间结果，可以表示为字节的矩阵数组，有 4 行 Nb 列。
S-box	一个非线性替代表： 在字节替代变换和密钥扩展程序中用于执行字节替变换。
Word	一个 32 比特的比特串，也可以看作包含 4 个字节的数组。

2.2 算法参数、符号和函数

下列算法参数、符号和函数将贯穿于该标准始终：

AddRoundKey()	加密和解密中使用的变换，将一个轮密钥异或到状态上。轮密钥的长度等于状态的大小(即对于 Nb=4，轮密钥长度等于 128bits/16bytes)。
InvMixColumns()	解密中使用的变换，MixColumns()的逆变换。
InvShiftRows()	解密中使用的变换，ShiftRows()的逆变换。
InvSubBytes()	解密中使用的变换，SubBytes()的逆变换。
K	密码所使用的秘密密钥
MixColumns()	加密中使用的变换，以状态的每一列作为输入，混合每一列的数据(彼此独立的)得到新的列。
Nb	状态包含的列(32-bit 字)的个数。对于 AES，Nb＝4 (参见 6.3.)
Nk	密钥包含的 32-bit 字的个数。对于该标准，Nk＝4,6 或 8。(参见 6.3.)

Nr	轮数，是 Nk 和 Nb(固定的)的函数。对于该标准，Nr＝10,12 或 14。(参见 6.3.)
Rcon[]	密钥扩展算法中用到的轮常数。
RotWord()	密钥扩展算法中使用的函数，对 4-byte 字进行循环移位。
ShiftRows()	加密中使用的变换，将状态的最后 3 行循环移动不同的位移量。
SubBytes()	加密中使用的变换，利用一个非线性字节替代表(S 盒)，独立地对状态的每个字节进行操作。
SubWord()	密钥扩展算法中使用的函数，它以 4-byte 字作为输入，对于 4 字节中的每一字节分别应用 S 盒，得到一个输出字。
XOR	异或运算
$\oplus$	异或运算
$\otimes$	两个多项式(每一个的度(degree)均小于 4)相乘再模 x^4+1。
•	有限域上的乘法

3. 符号和惯例

3.1 输入和输出

AES 算法的**输入**和**输出**均为一个长度为 128 的比特串（值为 0 或 1）。这些序列也可以称为**分组**(blocks)，其中含有的位(bits)数也称为分组长度。AES 算法的**密钥**是 128,192 或 256 比特的序列。本标准中不允许其它的输入、输出长度和密钥长度。

序列中比特的编号从 0 开始，至序列长度(分组长度或密钥长度)减 1。将该比特的编号 i 称为其索引。根据分组长度和密钥长度决定其值属于下列范围中的哪一个：$0 \le i$ <128, $0 \le i$ <192 或 $0 \le i$ <256。

3.2 字节(Bytes)

AES 算法中基本的运算单位是**字节**(byte)，即视为一个整体的 8 比特序列。3.1 节中描述的输入、输出和密钥比特序列将以字节为单位进行运算，将这些序列分成 8 个连续比特的组

形成字节数组(参见 3.3)。对于记为 a 的输入、输出或密钥，数组中的字节可以用如下两种形式表示：a_n 或 $a[n]$。n 可以属于如下范围：

Key length=128 bits, $0 \le n < 16$ Block length=128 bits, $0 \le n < 16$

Key length=192 bits, $0 \le n < 24$

Key length=256 bits, $0 \le n < 32$

AES 算法中的所有字节都可以表示成夹在大括号中间的值为 0 或 1 的比特联成的串，顺序如下$\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$。这些字节可以看作以多项式表示的有限域上的元素：

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (3.1)$$

例如，{01100011}表示有限域上的元素 $x^6 + x^5 + x + 1$。

也可以方便的用 16 进制符号来表示字节值，将每 4 比特表示成如图 1 所示的一个符号。

Bit Pattern	Character
0000	0
0001	1
0010	2
0011	3

Bit Pattern	Character
0000	4
0001	5
0010	6
0011	7

Bit Pattern	Character
0000	8
0001	9
0010	a
0011	b

Bit Pattern	Character
0000	c
0001	d
0010	e
0011	f

图 1　16 进制

因此，元素{01100011}可以表示为{63}，代表较高位 4 比特的符号仍在左边。

一些有限域运算将包括 8-bit 字节左侧的一个比特(b_8)。 当该比特出现时，它将立即以{01}的形式出现在 8-bit 字节前；例如，一个 9-bit 序列将表示为{01}{1b}。

3.3 字节数组

字节数组将表示为如下形式：

$$a_0 a_1 a_2 ... a_{15}$$

128-bit 输入序列划分成字节时，字节和字节内的比特排序按照如下方式：

$$input_0 input_1 input_2 ... input_{126} input_{127}$$

$$a_0 = \{input_0, input_1, ..., input_7\};$$

$$a_1 = \{input_8, input_9, ..., input_{15}\};$$

$$\bullet$$

$$\bullet$$

$$\bullet$$

$$a_{15} = \{input_{120}, input_{121}, ..., input_{127}\}.$$

该形式可以扩展到更长的序列(如 192-和 256-bit 密钥)，一般的有：

$$a_n = \{input_{8n}, input_{8n+1}, ..., input_{8n+7}\}. \qquad (3.2)$$

结合第 3.2 节和第 3.3 节，图 2 表示了字节中的每一比特如何编号。

Input bit sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Byte number	0								1								2								...
Bit numbers in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

图 2 字节和比特编号

3.4 状态(State)

AES 算法的运算都是在一个称为**状态**的二维字节数组上进行。一个状态由四行组成，每一行包括 Nb 个字节，Nb 等于分组长度除以 32。用 s 表示一个状态矩阵，每一个字节的位置由行号 r (范围是 $0 \le r < 4$)和列号 c (范围是 $0 \le c < Nb$)唯一确定，记为 $s_{r,c}$ 或 s[r,c]。在该标准中 Nb=4，即 $0 \le c < 4$(参见 6.3 节)。

如第 5 节所示，在加密和解密的初始阶段将输入字节数组 $in_0, in_1, ..., in_{15}$ 复制到如图 3 所示的状态矩阵中。加密或解密的运算都在该状态矩阵上进行，最后的结果将被复制到输出字节数组 $out_0, out_1, ..., out_{15}$。

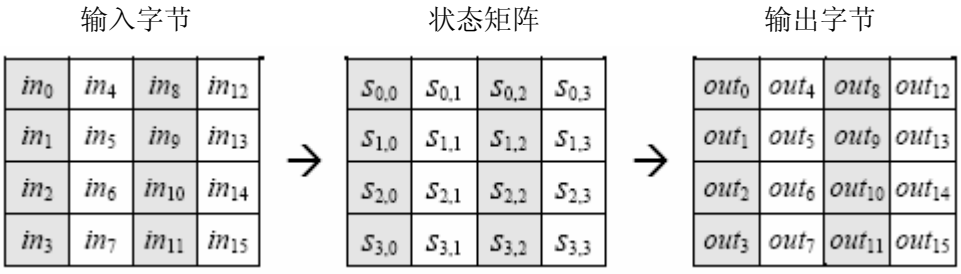


图 3 状态矩阵、输入和输出

在加密和解密的初始阶段，输入数组 in 按照下述规则复制到状态矩阵中：

s[r,c]=in[r+4c] for $0 \le r < 4$ 且 $0 \le c < Nb$ (3.3)

在加密和解密的结束阶段，状态矩阵将按照下述规则被复制到输出数组 out 中：

out[r+4c]= s[r,c] for $0 \le r < 4$ 且 $0 \le c < Nb$ (3.4)

3.5 将状态看作列数组

状态矩阵中每一列的四个字节可以看作一个 32-bit 字，行号 r 可以作为每一个字中四个字节的索引。因此状态可以看作 32-bit 字(列), $w_0 ... w_3$ 的一维数组，列号 c 是该数组的索引。因此对于图 3 中的例子，该状态可以看作 4 个字组成的数组，如下所示：

$$w_0 = s_{0,0} s_{1,0} s_{2,0} s_{3,0} \qquad w_2 = s_{0,2} s_{1,2} s_{2,2} s_{3,2}$$

$$w_1 = s_{0,1} s_{1,1} s_{2,1} s_{3,1} \qquad w_3 = s_{0,3} s_{1,3} s_{2,3} s_{3,3} \qquad (3.5)$$

4. 数学基础

AES 算法中的所有字节都将按照 3.2 节的记号表示有限域中的元素。有限域元素可以进行加法和乘法运算，但是这些运算不同于代数中使用的运算。下面将介绍一些第 5 章节要用到的基本数学概念。

4.1 加法

有限域中两个元素的加法定义为其多项式表示的相应系数的“加法”。此处加法是异或运算(记为⊕)，即模 2 加，1⊕1＝0，1⊕0＝1，0⊕0＝0。因此，多项式减法与多项式加法的规则相同。

有限域元素的加法也可以表示成字节中相应比特的模 2 加。对于两个字节 { $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$ } 和 { $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$ }，其和为 { $c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0$ }，

$c_i = a_i \oplus b_i$ (即 $c_7 = a_7 \oplus b_7$，$c_6 = a_6 \oplus b_6, \ldots, c_0 = a_0 \oplus b_0$)。

例如，下述表达式彼此相等：

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$ (多项式记法)；

{01010111} ⊕ {10000011} = {11010100} (二进制记法)；

{57} ⊕ {83} = {d4} {十六进制记法}。

4.2 乘法

在多项式表示中，有限域 GF(2^8)上的乘法(记为•)定义为多项式的乘积模一个次数为 8 的不可约多项式：

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad (4.1)$$

用十六进制表示该多项式为{01}{1b}。

例如，{57}•{83}={c1}，因为

$(x^6+x^4+x^2+x+1)(x^7+x+1)$

$=x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1$

$=x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$

而 $x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \quad \text{modulo} \quad (x^8+x^4+x^3+x+1) \quad = \quad x^7+x^6+1$

模 m(x)确保了所得结果是次数小于 8 的二元多项式，因此可以用一个字节表示。不像加法，乘法在相应的字节级别并没有简单运算。

上述定义的乘法具有结合性，元素{01}是乘法单位元。对任意次数小于 8 的非零二元多项式 b(x)，其乘法逆元记为 $b^{-1}(x)$，可通过下述方法找到：使用扩展欧几里德算法计算多项式 a(x)和 c(x)使得

$$b(x)a(x)+m(x)c(x)=1 \qquad (4.2)$$

因此 $a(x) \bullet b(x) \bmod m(x)=1$ 意味着

$$b^{-1}(x)=a(x) \bmod m(x). \qquad (4.3)$$

而且，对该域上的任意 a(x)，b(x)和 c(x)均有下式成立

$$a(x) \bullet (b(x)+c(x))=a(x) \bullet b(x)+a(x) \bullet c(x)$$

由此可见，由所有 256 个可能的字节值组成的集合，使用异或作为加法以及上述定义的乘法运算，构成有限域 $GF(2^8)$。

4.2.1 乘 x

用多项式 x 乘以等式(3.1)中定义的二元多项式将得到

$$b_7x^8+b_6x^7+b_5x^8+b_4x^8+b_3x^4+b_2x^3+b_1x^2+b_0x \qquad (4.4)$$

由等式(4.1)的定义知，将上述结果模 m(x)即可得到 $x \bullet b(x)$的结果。如果 $b_7=0$，则该结果已经是模运算后的形式。如果 $b_7=1$，则模运算需要异或多项式 m(x)完成。乘 x (即{00000010}或{02})可通过字节级别的左移和与{1b}进行有条件的按位异或来实现。将该操作记为 xtime( )。x 的高次幂的乘法可以通过重复应用 xtime( )实现。通过将中间结果相加，可以实现任意常数的乘法。

例如，{57}•{13}={fe} 因为

{57}•{02}=xtime({57})={ae}

{57}•{04}=xtime({ae})={47}

{57}•{08}=xtime({47})={8e}

{57}•{10}=xtime({8e})={07}

因此

{57}•{13}={57}•({01}⊕{02}⊕{10})

={57}⊕{ae}⊕{07}

={fe}

4.3 系数在 $GF(2^8)$中的多项式

考虑含有 4 个项、且系数为有限域元素的多项式，即

$$a(x)=a_3x^3+a_2x^2+a_1x+a_0 \qquad (4.5)$$

它可以表示为如下形式[a_0, a_1, a_2, a_3]的字(word)。注意本节中的多项式与有限域元素定义中使用的多项式操作起来是不同的，即使这两类多项式均使用相同的变量 x。该节中的系数本身就是有限域元素，即字节(bytes)而不是比特(bits)。同时，该 4 项多项式的乘法使用了一个不同的模多项式，将在下面定义。

为说明加法和乘法运算，令

$$b(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0 \qquad (4.6)$$

为另一个 4 项多项式。加法是对 x 相应次数项的有限域系数进行相加运算。该加法对应于相应字节间的异或运算。换句话说，即整个字值的异或。

因此，使用等式(4.5)和(4.6)，

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0) \qquad (4.7)$$

乘法要用两步完成。第一步，对多项式相乘的结果$c(x) = a(x) \bullet b(x)$进行代数扩展：

$$c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 \qquad (4.8)$$

其中

$$c_0 = a_0 \bullet b_0 \qquad c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \qquad c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \qquad c_6 = a_3 \bullet b_3$$

$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \qquad (4.9)$$

所得结果 c(x)并没有表示为一个 4 – 字节的字。因此，乘法的第二步是模一个 4 次多项式来化简 c(x)，使得结果化简为一个次数小于 4 的多项式。在 AES 算法中，这一模多项式取为$x^4 + 1$，由于

$$x^j \mod (x^4+1) = x^{j \bmod 4} \qquad (4.10)$$

则 a(x)和 b(x)取模的乘积记为$a(x) \otimes b(x)$，表示为下述的 4 项多项式 d(x)，即

$$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0 \qquad (4.11)$$

其中

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \qquad (4.12)$$

当 a(x)是一个固定多项式时，等式(4.11)中定义的运算可以写成矩阵形式，如：

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad (4.13)$$

由于 x^4+1 不是 GF(2^8)上的不可约多项式，因此被一个固定的 4 次多项式相乘的乘法不一定可逆。然而，在 AES 算法中选择了一个固定的 4 项多项式的乘法，它按照乘法运算有逆元(见 5.1.3 节和 5.3.3 节)：

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad (4.14)$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \qquad (4.15)$$

AES 算法中用到的另一个多项式(见 5.2 节中的 **RotWord( )** 函数)为 $a_0 = a_1 = a_2 = \{00\}$，$a_3 = \{01\}$，即多项式 x^3。通过对上述等式(4.13)的观察，可以发现其效果是将输入字中的字节循环移位来得到输出字。即 $[b_0, b_1, b_2, b_3]$ 将变换为 $[b_1, b_2, b_3, b_0]$。

5. 算法说明

对于 AES 算法，输入分组、输出分组、状态长度均为 128 比特。Nb=4 ，该值反应了状态中 32-bit 字的个数(列数)。

对于 AES 算法，密钥 K 的长度是 128、192 或 256 bits。密钥长度表示为 Nk=4、6 或 8，反应了密钥中 32-bit 字的个数(列数)。

对于 AES 算法，算法的轮数依赖于密钥长度。将轮数表示为 Nr，当 Nk=4 时 Nr＝10；当 Nk=6 时 Nr＝12；当 Nk=8 时 Nr＝14。

符合该标准的一切密钥长度 - 分组长度 - 轮数的组合如图 4 所示。

与密钥长度、分组大小和轮数有关的实现方面的问题见 6.3 节。

	密钥长度 (Nk words)	分组大小 (Nb words)	轮数 (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

图 4 Key-Block-Round 组合

对于加密和解密变换，AES 算法使用的轮函数由 4 个不同的以字节为基本单位的变换复合而成：1）字节替代，利用一个替代表(即 S 盒)，2)将状态矩阵的每一行循环移位不同的位移量，3)将状态矩阵中每一列的数据进行混合，4)将轮密钥加到状态上。这些变换(及其逆变换)将在 5.1.1－5.1.4 和 5.3.1－5.3.4 节描述。

加密和解密分别在 5.1 节和 5.3 节描述，密钥扩展算法在 5.2 节描述。

5.1 加密

加密开始时，使用 4.3 节描述的惯例将输入复制到状态矩阵中。经过初始轮子密钥加后，通过执行 10、12 或 14 次轮函数来变换状态矩阵(依赖于密钥长度)，最后一轮与前 Nr －1 轮略有不同。最后状态将如 3.4 节中所述被复制到输出。

轮函数通过密钥扩展算法进行参数化，密钥编排由 5.2 节中描述的密钥扩展程序得到的一维 4-byte 字数组组成。

加密算法由图 5 中的伪代码表示，下面的每一个个变换—**SubBytes**（），**ShiftRows**（），**MixColumns**（），和 **AddRoundKey**（）— 都作用在状态上，将在下面的章节中说明。在图 5 中，数组 w[ ]中包含了密钥编排得到的密钥，这些将在 5.2 节中说明。

如图 5 所示，除了最后一轮，所有的轮变换均相同。最后一轮不包括 MixColumns（）变换。

附录 B 给出了该加密算法的一个例子，列出了每一轮变换之前的状态矩阵值以及应用了下面几节描述的 4 个变换后的值。

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

图 5 加密算法伪代码[1]

5.1.1 字节替代(SubBytes( ))变换

字节替代(SubBytes（）)变换是一个非线性的字节替代，它独立地将状态中的每个字节

[1] 作用于状态矩阵上的各种变换（如 SubBytes（），ShiftRows（）等等）由‘state’指针指向其地址。AddRoundKey（）使用额外的指针指向轮密钥的地址。

利用替代表(S 盒)进行运算。该 S 盒(图 7)是可逆的，由两个变换复合而成：

1. 选取 4.2 节所描述的有限域 GF(2^8)上的乘法逆运算，其中，元素{00}映射到它自身。
2. 应用下面定义的(GF(2)上的)仿射变换：

$$b_i^{'} = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \qquad (5.1)$$

对于$0 \le i < 8$，b_i 是字节的第 i 比特，c_i 是值为{63}或{01100011}的字节 c 的第 i 比特。在此处和其它地方，在变量的右上角作标记（如 b'）表示该变量将用右侧的值更新。

以矩阵的形式，S 盒的仿射变换可以表示为：

$$\begin{bmatrix} b_0^{'} \\ b_1^{'} \\ b_2^{'} \\ b_3^{'} \\ b_4^{'} \\ b_5^{'} \\ b_6^{'} \\ b_7^{'} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \qquad (5.2)$$

图 6 画图说明了 SubBytes（）变换在状态上的作用效果。

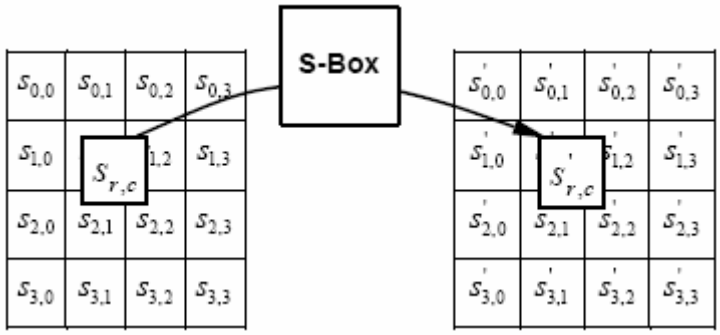


图 6 SubBytes（）作用在状态的单个字节上

以 16 进制表示的 SubBytes（）变换对应的 S 盒如图 7 所示。

例如，如果 $s_{1,1}$ ={53}，则替代后的值将由图 7 中行标为 5 列标为 3 的交集决定。即 $s'_{1,1_{1,1}}$ 的值为{ed}。

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

图 7　S 盒：字节 xy 的替代值（16 进制格式）

5.1.2 行移位(ShiftRows( ))变换

在行移位(ShiftRows( ))变换中，状态的最后 3 行循环移位不同的位移 r。第一行中 r = 0，即保持不变。

具体地，行移位变换按照下述方法进行：

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb} \qquad 0 < r < 4 \text{且} 0 \le c < Nb \qquad (5.3)$$

移位量 shift(r，Nb)依赖于行号 r，如下所示(记着 Nb=4)：

shift(1，4)=1；　Shift(2，4)=2；　Shift(3，4)=3；　(5.4)

其作用效果是将行中的字节移向较低位(即给定行中 c 的较低值)，最低位的字节将交换至行的最高位(即给定行中 c 的较高值)。

图 8 描述了行移位变换。

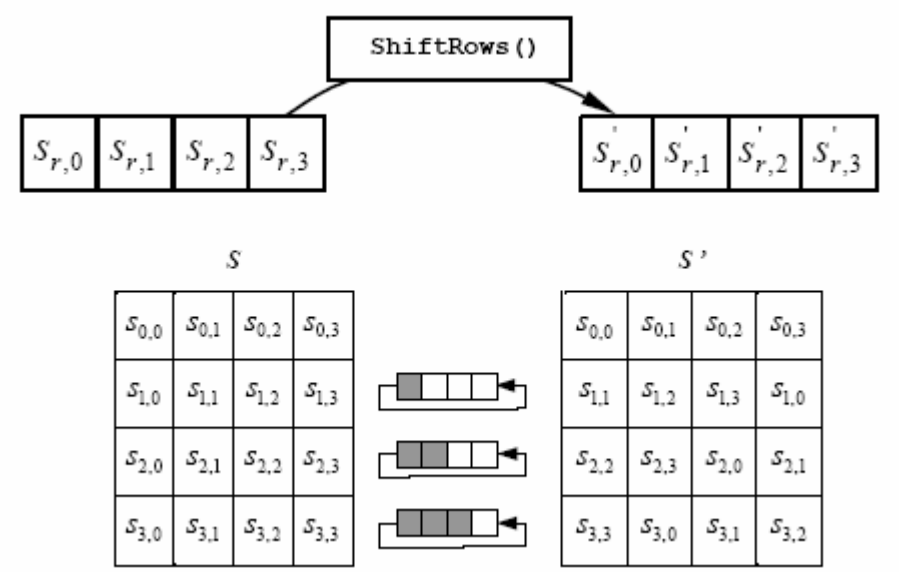


图 8 ShiftRows（）在状态的后三行上循环移位

5.1.3 列混合(MixColumns( ))变换

列混合(MixColumns( ))变换在状态上按照每一列进行运算，并将每一列看作 4.3 节中描述的4次多项式，即将状态的列看作 GF(2^8)上的多项式且被一个固定的多项式 $a(x)$ 模 x^4+1 乘， $a(x)$ 为：

$$a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\} \tag{5.5}$$

正如 4.3 节中所述，这可以写成矩阵乘法。令 $s'(x)=a(x)\otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad 0 \le c < Nb \tag{5.6}$$

经过该乘法计算后，一列中的 4 个字节将由下述结果取代：

$$s'_{0,c}=(\{02\}\bullet s_{0,c})\oplus(\{03\}\bullet s_{1,c})\oplus s_{2,c}\oplus s_{3,c}$$

$$s'_{1,c}=s_{0,c}\oplus(\{02\}\bullet s_{1,c})\oplus(\{03\}\bullet s_{2,c})\oplus s_{3,c}$$

$$s'_{2,c}=s_{0,c}\oplus s_{1,c}\oplus(\{02\}\bullet s_{2,c})\oplus(\{03\}\bullet s_{3,c})$$

$$s'_{3,c}=(\{03\}\bullet s_{0,c})\oplus s_{1,c}\oplus s_{2,c}\oplus(\{02\}\bullet s_{3,c})$$

图 9 描述了 MixColumns（）变换。

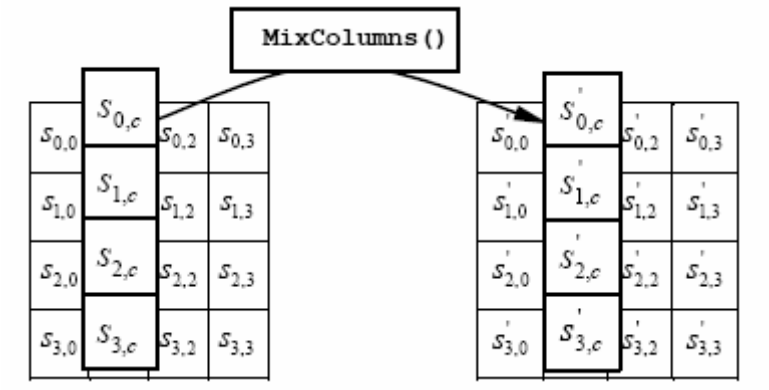


图 9　MixColumns（）在状态的列上运算

5.1.4 轮密钥加(AddRoundKey( ))变换

在轮密钥加变换中，用简单的比特异或将一个轮密钥作用在状态上。每一个轮密钥由

通过密钥编排得到的 Nb 个字组成(5.2 节描述)。将这 Nb 个字异或到状态的列上，即

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \qquad 0 \le c < Nb \qquad (5.7)$$

其中的 $[w_i]$ 是 5.2 节中描述的密钥编排得到的字，*round* 是如下范围内的值 $0 \le round \le Nr$。在加密算法中，当 *round*=0，在应用第一个轮函数之前进行初始轮密钥加。当 $1 \le round \le Nr$ 时，AddRoundKey（）变换应用到加密算法的 Nr 轮上。

该运算如图 10 所示，其中 *l=round*Nb*。密钥编排得到的字中的字节编号如 3.1 节所述。

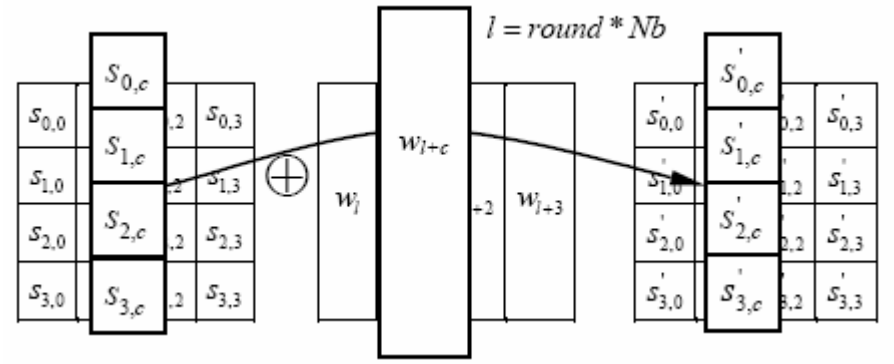


图 10 AddRoundKey（） 将密钥编排得到的字异或到状态的每一列上

5.2 密钥扩展算法

AES 算法接受密码的秘密密钥 K，运行密钥扩展程序来生成密钥编排的结果。密钥扩展总共生成 Nb(Nr+1)个字：该算法需要一个 Nb 个字组成的初始集合，Nr 轮中的每一轮需要 Nb 个字的密钥数据。密钥编排结果由一个 4-byte 字的线性数组组成，记为$[w_i]$，其中 $0 \le i < Nb(Nr+1)$。

根据图 11 中的伪代码将输入密钥扩展成密钥编排结果。

SubWord（）函数将接受一个 4-byte 输入字，对每一个字节应用 S 盒(5.1.1 节图 7)得到输出字。函数 **RotWord**（）接受字$[a_0, a_1, a_2, a_3]$作为输入，执行循环移位后返回字 $[a_1, a_2, a_3, a_0]$。轮常数字数组 **Rcon[i]**，包含了由$[x^{i-1}, \{00\}, \{00\}, \{00\}]$给定的值。如 4.2 节所述(注意到 i 从 1 开始，而不是 0)，x^{i-1}是有限域 GF(2^8)上 x(x 记为{02})的指数幂。

由图 11 可见，第一个 **Nk** 字由密码的秘密密钥直接填充。接下来的每个字 **w[i]**，等于其前一个字 **w[i－1]**与 **Nk** 个位置之前的字 **w[i－Nk]**的异或(XOR)。对于 **Nk** 的整数倍位置的字，在异或之前，要对 **w[i－1]**进行一次变换，该变换先进行一次字的字节循环移位(RotWord（）)，然后再做一次字节替代变换(SubWord（）)，即对字中的 4 个字节应用查表。再异或一个轮常数。

需要注意 256-bit 密钥(**Nk＝8**)的密钥扩展程序与 128-bit 和 192-bit 密钥的稍有不同。如果 **Nk＝8** 且 i-4 是 **Nk** 的整数倍，异或之前对 **w[i－1]**要做一次字节替代(**SubWord**（）)变换。

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   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)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented;
they are all included in the conditional statement above for
conciseness.   Specific implementation requirements for the
Cipher Key are presented in Sec. 6.1.
```

图 11 密钥扩展伪代码[2]

附录 A 中给出了密钥扩展的例子。

5.3 解密

对于 AES 算法，可以将 5.1 节中的加密变换逆转，然后以逆序执行即可直接得到解密算法。解密算法中使用的变换－**InvShiftRows**（），**InvSubBytes**（），**InvMixColumns**（）和 **AddRoundKey**（）－作用在状态上，将在下面的章节中说明。

图 12 描述了解密算法的伪代码。在图 12 中，如前面 5.2 节中所描述的数组 **w[ ]**包含了密钥编排得到的密钥。

[2] 函数 SubWord（）和 RotWord（）会返回一个结果值，即是对函数输入的变换，然而加密和解密中的变换（如 ShitfRows（），SubBytes（）等等）是对由‘state’指针指向的状态矩阵的变换。

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

图 12　解密算法的伪代码[3]

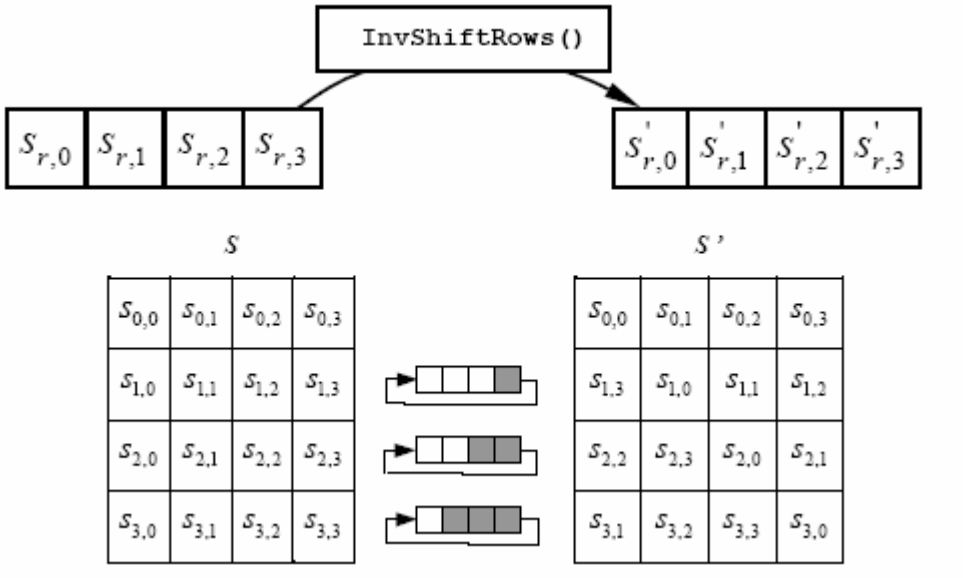
5.3.1 逆行移位(InvShiftRows( ))变换

逆行移位(**InvShiftRows**（）)变换是行移位(**ShiftRows**（）)变换的逆变换。状态最后 3 行中的字节循环移位不同的字节数(位移量)。第一行中 r = 0，即不移位。下面的三行将循环移动 **Nb**－shift(r,Nb)字节，其中位移量 shift(r,Nb)依赖于行号并由等式(5.4)(见 5.1.2 节)给出。

明确的，逆行移位变换将按照下述方法进行：

$$s'_{r,(c+shift(r,Nb)) \bmod N} = s_{r,c} \qquad 0 < r < 4 \text{且} 0 \le c < Nb \qquad (5.8)$$

图 13 描述了逆行移位(**InvShiftRows**（）)变换。



[3]作用于状态矩阵上的各种变换（如 SubBytes（），ShiftRows（）等等）由‘state’指针指向其地址。AddRoundKey（）使用额外的指针指向轮密钥的地址。

图 13　InvShiftRows（）在状态的后三行上循环移位

5.3.2 逆字节替代(InvSubBytes( ))变换

逆字节替代(**InvSubBytes**（）)变换是字节替代(**SubBytes**（）)变换的逆变换，在状态的每个字节上应用 S 盒的逆。这是通过应用仿射变换(5.1)的逆，再接着应用 GF(2^8)中的乘法逆得到的。

逆字节替代(**InvSubBytes**（）)变换中使用的 S 盒的逆如图 14 所示：

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

图 14　S 盒的逆：字节 xy 的替代值（16 进制格式）

5.3.3 逆列混合(InvMixColumns( ))变换

逆列混合(**InvMixColumns**（）)变换是列混合(**MixColumns**（）)变换的逆变换。逆列混合(**InvMixColumns**（）)变换在状态上对每一列进行运算，将每一列看作 4.3 节中描述的 4 次多项式。将状态的列看作 GF(2^8)上的多项式且被一个固定的多项式 $a^{-1}(x)$ 模 x^4+1 乘，$a^{-1}(x)$ 为：

$$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\} \tag{5.9}$$

正如 4.3 节中所述，这可以写成矩阵乘法。令 $s'(x)=a^{-1}(x)\otimes s(x)$：

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix}=\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \qquad 0\le c<Nb \tag{5.10}$$

经过该乘法计算后，一列中的 4 个字节将由下述结果取代：

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

5.3.4 轮密钥加(InvMixColumns( ))变换的逆变换

5.1.4 节中所述的轮密钥加(**InvMixColumns**（）)变换，其逆变换就是它本身，因为其中只应用了异或(XOR)运算。

5.3.5 等价的解密变换

在 5.3 节和图 12 中描述的直接得到的解密算法中，其各个变换的操作顺序与加密算法不同，但是加密和解密算法中的密钥编排形式相同。然而，AES 算法的若干性质允许构造一个等价的解密算法，解密时各个变换的操作顺序与加密(由逆变换取代原来的变换)相同。这是通过改变密钥编排完成的。

允许该等价解密变换的二个关键性质如下：

1. 字节替代(**SubBytes**（）)变换和行移位(**ShiftRows**（）)变换的顺序不影响结果。即，先进行字节替代(**SubBytes**（）)变换紧接着进行行移位(**ShiftRows**（）)变换等价于先进行行移位(**ShiftRows**（）)变换紧接着再进行字节替代(**SubBytes**（）)变换。对于逆变换 **InvSubBytes**（）和 **InvShiftRows**（），该性质也成立。
2. 列混合运算－**MixColumns**（）和 **InvMixColumns**（）－是关于列输入的线性变换，即意味着

   InvMixColumns（**state XOR Round Key**）＝

   InvMixColumns（**state**） **XOR InvMixColumns**（**Round Key**）

这些性质使得 **InvSubBytes**（）变换和 **InvShiftRows**（）变换可以交换顺序。**AddRoundKey**（）变换和 **InvMixColumns**（）变换的顺序也可以交换，只要将解密中密钥编排得到的密钥列（字）应用 **InvMixColumns**（）变换进行修改即可。

等价的解密算法如图 12 所示，定义为将 **InvSubBytes**（）变换和 **InvShiftRows**（）变换的顺序反转过来，同时当利用 **InvMixColumns**（）变换先修改了 1 到 Nr-1 轮的解密密钥编排结果后，将“轮循环”中使用的 **AddRoundKey**（）变换和 **InvMixColumns**（）变换的顺序反转。解密密钥编排结果的第一和最后 Nb 字将不使用该方式进行修改。

给出的这些改变，使得等价的解密算法比 5.3 节和图 12 中描述的解密算法具有更有效的结构。图 15 中给出了等价解密算法的伪代码。（字数组 dw[ ]中包含了修改过的解密密钥编排结果。图 15 中也给出了对密钥扩展程序的修改。）

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvMixColumns(state)
      AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
   end for

   InvSubBytes(state)
   InvShiftRows(state)
   AddRoundKey(state, dw[0, Nb-1])

   out = state
end


For the Equivalent Inverse Cipher, the following pseudo code is added at
the end of the Key Expansion routine (Sec. 5.2):
   for i = 0 step 1 to (Nr+1)*Nb-1
      dw[i] = w[i]
   end for


   for round = 1 step 1 to Nr-1
      InvMixColumns(dw[round*Nb, (round+1)*Nb-1])     //  note  change  of
type
   end for

Note that, since InvMixColumns operates on a two-dimensional array of bytes
while the Round Keys are held in an array of words, the call to
InvMixColumns in this code sequence involves a change of type (i.e. the
input to InvMixColumns() is normally the State array, which is considered
to be a two-dimensional array of bytes, whereas the input here is a Round
Key computed as a one-dimensional array of words).
```

图 15　等价解密算法的伪代码

6. 实现方面的问题

6.1 密钥长度要求

AES 算法的实现至少需要支持第 5 节中描述的 3 种密钥长度：128，192 或 256 bits（即 Nk＝4，6 或 8 分别的）。实现可以选择支持两种或三种密钥长度，这将促进算法执行的互用性。

6.2 密钥限制

对于 AES 算法没有发现弱密钥或半弱密钥，所以对密钥选取没有限制。

6.3 密钥长度，分组大小和轮数的参数化

该标准明确地定义了密钥长度（Nk），分组大小（Nb）和轮数（Nr）允许的取值如图 4 所示。然而，该标准的未来版本可能包括对这些参数允许取值的改变或增加。因此，当实现者设计 AES 的实现时可以选择将未来的变化考虑在内。

6.4 针对不同平台的实现建议

很多情况下，实现的可变性可能会提供更好的性能或其它优势。当给定相同的输入密钥和数据（明文或密文）时，任意与该标准说明的算法得到相同输出（密文或明文）的实现都是可以接受的 AES 实现。

参考文献【3】和参考【1】中列出的其它文章都包含了一些关于如何在不同平台上有效地实现 AES 算法的建议。