第 2 章：分组密码体制

2.5 高级数据加密标准

赵俊舟

junzhou.zhao@xjtu.edu.cn

2025 年 3 月 5 日

AES 的起源

- DES 不安全，建议用 3DES，密钥 168 位，抵御密码分析攻击，但是 3DES 用软件实现速度较慢，分组仅 64 位。
- 美国国家标准技术协会 NIST 在 1997 年征集新标准，要求明文分组长度 128 位，密钥长 128、192 或 256 位。
- 1998 年 6 月，15 种候选算法通过了第一轮评估；1999 年 8 月，仅有 5 个候选算法通过了第二轮评估。
- 2000 年 10 月，NIST 选择 Rijndael 算法作为 AES 算法，Rijndael 的作者是比利时的密码学家 Joan Daemen 博士和 Vincent Rijmen 博士。
- 2001 年 11 月 NIST 发布了 AES 的最终标准 FIPS PUB 197。

AES 的评估准则

- **安全性**：指密码分析方法分析一个算法所需的代价；
- **成本**：期望 AES 能够广泛应用于各种实际应用，计算效率要高；
- **算法和执行特征**：算法灵活性、适合于多种硬件和软件方式的实现、简洁性，便于分析安全性。

Rijndael 的评估结果

- 一般安全性：依赖于密码学界的公共安全分析
- 软件实现：软件执行速度，跨平台执行能力及密钥长度改变时速度变化
- 受限空间环境：在诸如智能卡中的应用
- 硬件实现：硬件执行提高执行速度或缩短代码长度
- 对执行的攻击：抵御密码分析攻击
- 加密与解密
- 密钥灵活性：快速改变密钥长度的能力
- 其他的多功能性和灵活性
- 指令级并行执行的潜力

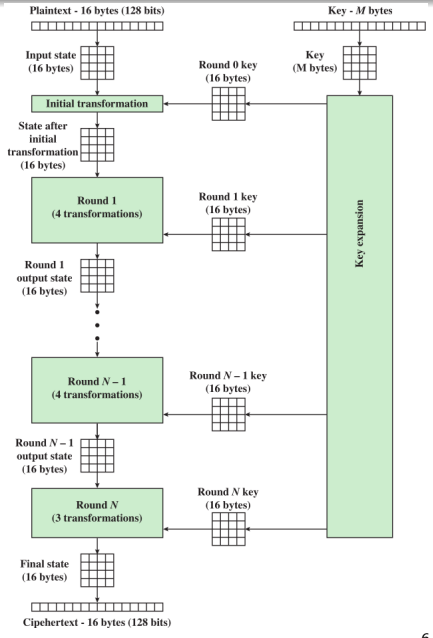
AES 参数选择

- AES 的分组长度为 128 位。
- 密钥长度可以是 128/192/256 的任意一种，根据使用的密钥长度，分为 AES-128，AES-192 或 AES-256。
- 接下来主要以 AES-128 为例进行讲述。

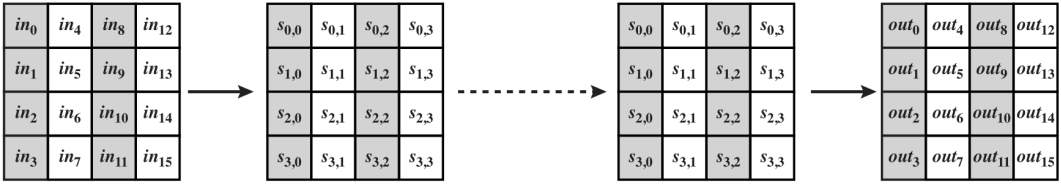
Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

AES 加密过程总体结构

- AES 采用非 Feistel 结构。
- 输入 128 位分组，表示为 4×4 字节方阵，称为状态数组。
- 密钥被表示为 4×4 密钥方阵，扩展为密钥字阵列，共包含 $N + 1$ 个密钥方阵。
- 加密由 N 轮构成，前 $N - 1$ 轮由 4 种不同变换组成，最后一轮仅包含 3 种变换。
- 每一轮修改状态数组，最后一轮输出密文。

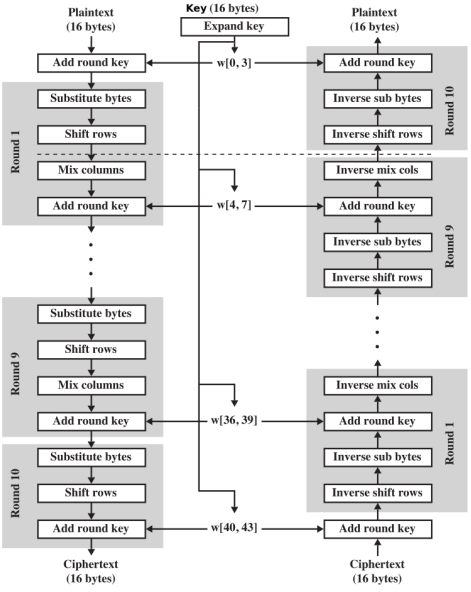


状态数组的修改及密钥扩展



AES 详细结构

- 输入密钥被扩展为 44 字数组 $w[i]$。
- 4 种操作（1 个代替，3 个置换）：
  - 字节代替：S 盒完成字节代替
  - 行移位：一个简单置换操作
  - 列混淆：域 $\mathrm{GF}(2^8)$ 上的算术
  - 轮密钥加：当前分组与扩展密钥一部分按位异或
- 每种操作均可逆，仅仅在轮密钥加阶段使用密钥。
- AES 的加解密算法不同，加解密的最后一轮均只包含 3 种操作。



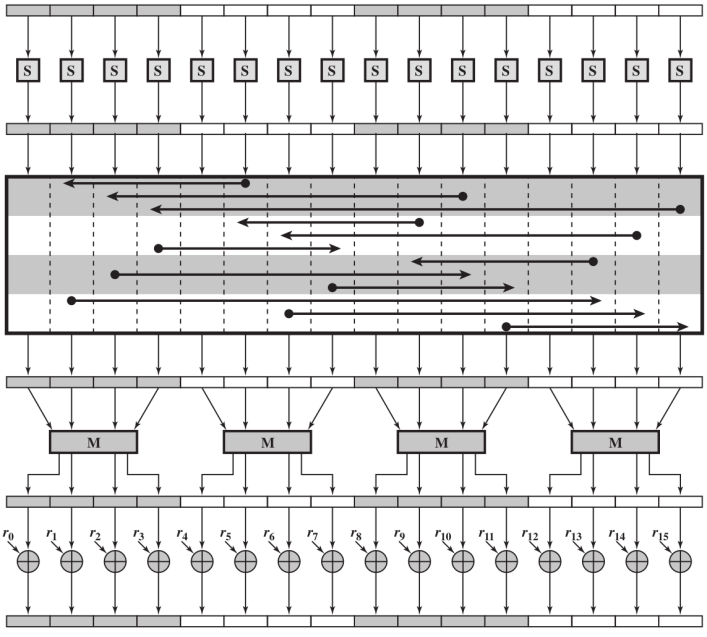
AES 的一轮加密过程

字节代替

行移位

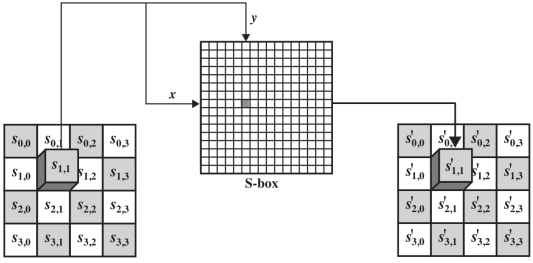
列混淆

轮密钥加



字节代替变换

- 字节代替变换是一个查表操作，实现分组字节到字节的代替。
- S 盒是一个 16×16 字节表，包含了所有 8 位的置换值。
- 每个字节的左 4 位选择行，右 4 位选择列来查表替换。



EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

S 盒

		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

逆 S 盒

		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

S 盒的构造方法

- 按字节值的升序逐行初始化 S 盒。第 x 行 y 列的值为 $\{xy\}$。
- 把 S 盒中的每个字节映射为它在域 $\mathrm{GF}(2^8)$ 中的逆，使用素多项式 $m(x) = x^8 + x^4 + x^3 + x + 1$，其中 $\{00\}$ 映射为 $\{00\}$。
- 把 S 盒中的每个字节表示为 $b_7 b_6 \cdots b_0$，对每个字节的每位做如下变换

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

  其中 c_i 指常数 $\{63\}$ 的第 i 位，即 $c = \{63\} = (01100011)$。
- 可以表示为下式，注意行和列相乘后再进行按位异或。

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

逆 S 盒的构造方法

- 求上一变换的逆变换，然后再求其在 $\mathrm{GF}(2^8)$ 上的乘法逆：

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

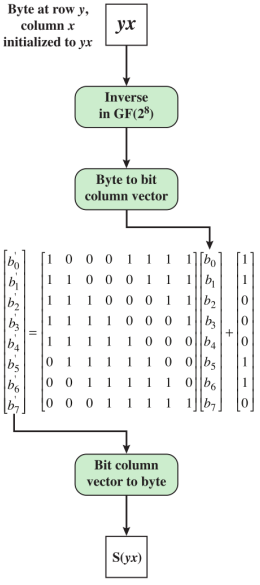
其中字节 $d = \{05\} = (00000101)$。表示为矩阵形式：

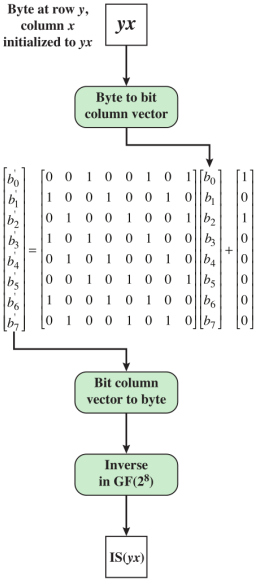
$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- 令字节代替变换和逆变换中的矩阵分别为 X 和 Y，常量 c、d 的向量表示为 C 和 D。
- $B' = XB \oplus C \Rightarrow YB' \oplus D = Y(XB \oplus C) \oplus D = YXB \oplus YC \oplus D = B$
- 可以发现 $YX = I$，$YC = D$，故 $YB' \oplus D = B$

S 盒和逆 S 盒

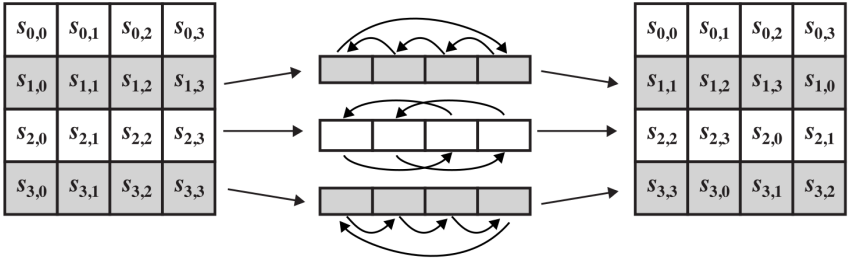
- S 盒被设计成能防止已知的各种密码分析攻击。
- 输入和输出的相关性很低，输出不是输入的线性函数，非线性度的产生是由于使用了乘法逆。
- 常量的选择使得 S 盒中没有不动点，即 $S[a] = a$，也没有“反不动点”，即 $S[a] = \bar{a}$，$\bar{a}$ 与 a 逐位取反。
- S 盒可逆，即 $S^{-1}[S[a]] = a$，但不自逆，即 $S[a] \neq S^{-1}[a]$。





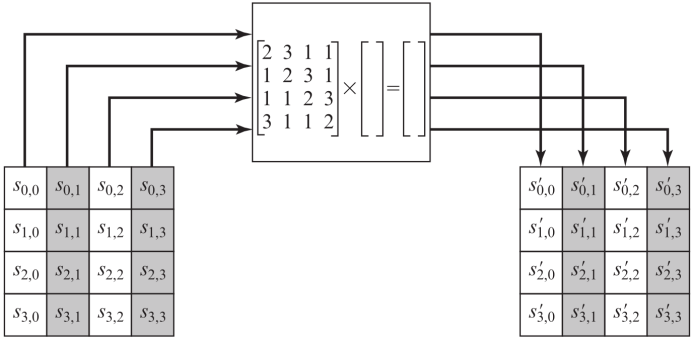
行移位变换

- 正向行移位变换：第 1 行保持不变。第 2 行循环左移 1 个字节。第 3 行循环左移 2 个字节。第 4 行循环左移 3 个字节。
- 逆向行移位变换进行相反的移位以实现解密。
- 因为状态矩阵是按列处理的，行移位变换就把字节在列之间进行了置换。



列混淆变换

- 正向列混淆变换对每列独立进行操作。
- 每列中的每个字节被映射为一个新值，由该列中的 4 个字节通过函数变换得到。
- 可以用状态矩阵乘法表示，使用素多项式 $m(x) = x^8 + x^4 + x^3 + x + 1$，加法和乘法都定义在 $\mathrm{GF}(2^8)$ 上。



列混淆变换

- 写成矩阵形式为

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix}$$

  例如 $s'_{0j} = 2s_{0j} \oplus 3s_{1j} \oplus s_{2j} \oplus s_{3j}$

- 逆向列混淆变换也由矩阵乘法定义：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix}$$

轮密钥加变换

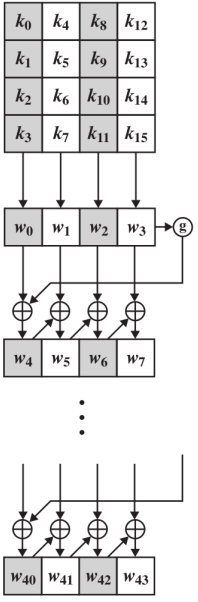
- 正向轮密钥加变换中 128 位的状态矩阵按位与 128 位的密钥异或。
- 都是基于状态矩阵列的操作，即把状态矩阵的一列中的 4 个字节与轮密钥的 1 个字进行异或。
- 逆向轮密钥加变换与正向轮密钥加变换相同，因为异或操作是其本身的逆。
- 轮密钥加变换非常简单，却能影响状态矩阵中的每一位。

密钥扩展算法

- AES 密钥扩展算法的输入是 4 字，输出 44 字。
- 输入密钥直接被复制到扩展密钥数组的前 4 个字。
- 然后每次用 4 字填充扩展密钥数组余下的部分：

$$w[i] = \begin{cases} w[i-4] \oplus w[i-1] & i \bmod 4 \neq 0 \\ w[i-4] \oplus g(w[i-1]) & i \bmod 4 = 0 \end{cases}$$

其中 g 函数对一个字进行复杂变换，输出另外一个字。

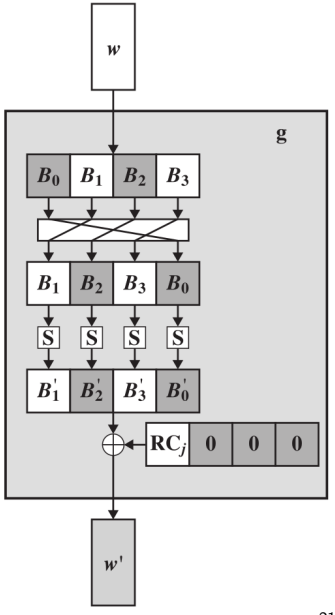


密钥扩展算法：g 函数

- 首先，字循环使一个字中的 4 个字节循环左移一个字节，即将输入字 $[B_0, B_1, B_2, B_3]$ 变为 $[B_1, B_2, B_3, B_0]$。
- 然后，字代替利用 S 盒对输入字中的每个字节进行字节代替。
- 最后，将前两步的结果与轮常量 $\mathrm{Rcon}_j = [\mathrm{RC}_j, 0, 0, 0]$ 相异或。
  - 轮常量是一个字，字的最右边 3 个字节总为 0
  - 轮常量第一个字节定义为（$\mathrm{GF}(2^8)$ 上的乘法）

$$\mathrm{RC}_1 = 1, \quad \mathrm{RC}_j = 2 \cdot \mathrm{RC}_{j-1}$$

j	1	2	3	4	5	6	7	8	9	10
RC_j	01	02	04	08	10	20	40	80	1B	36



密钥扩展算法

```
KeyExpansion(byte key[16], word w[44]){
  word tmp;
  for(i=0; i<4; ++i)
    w[i] = (key[4*i], key[4*i+1], key[4*i+2],
            key[4*i+3]);
  for(i=4; i<44; ++i){
    tmp = w[i-1];
    if(i mod 4 ==0)
      tmp = SubWord(RotWord(tmp))^Rcon[i/4];
    w[i] = w[i-4]^tmp;
  }
}
```

AES 举例

Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key
01 89 fe 76 23 ab dc 54 45 cd ba 32 67 ef 98 10				0f 47 0c af 15 d9 b7 7f 71 e8 ad 67 c9 59 d6 98
0e ce f2 d9 36 72 6b 2b 34 25 17 55 ae b6 4e 88	ab 8b 89 35 05 40 7f f1 18 3f f0 fc e4 4e 2f c4	ab 8b 89 35 40 7f f1 05 f0 fc 18 3f c4 e4 4e 2f	b9 94 57 75 e4 8e 16 51 47 20 9a 3f c5 d6 f5 3b	dc 9b 97 38 90 49 fe 81 37 df 72 15 b0 e9 3f a7
65 0f c0 4d 74 c7 e8 d0 70 ff e8 2a 75 3f ca 9c	4d 76 ba e3 92 c6 9b 70 51 16 9b e5 9d 75 74 de	4d 76 ba e3 c6 9b 70 92 9b e5 51 16 de 9d 75 74	8e 22 db 12 b2 f2 dc 92 df 80 f7 c1 2d c5 1e 52	d2 49 de e6 c9 80 7e ff 6b b4 c6 d3 b7 5e 61 c6
5c 6b 05 f4 7b 72 a2 6d b4 34 31 12 9a 9b 7f 94	4a 7f 6b bf 21 40 3a 3c 8d 18 c7 c9 b8 14 d2 22	4a 7f 6b bf 40 3a 3c 21 c7 c9 8d 18 22 b8 14 d2	b1 c1 0b cc ba f3 8b 07 f9 1f 6a c3 1d 19 24 5c	c0 89 57 b1 af 2f 51 ae df 6b ad 7e 39 67 06 c0
71 48 5c 7d 15 dc da a9 26 74 c7 bd 24 7e 22 9c	a3 52 4a ff 59 86 57 d3 f7 92 c6 7a 36 f3 93 de	a3 52 4a ff 86 57 d3 59 c6 7a f7 92 de 36 f3 93	d4 11 fe 0f 3b 44 06 73 cb ab 62 37 19 b7 07 ec	2c a5 f2 43 5c 73 22 8c 65 0e a3 dd f1 96 90 50
f8 b4 0c 4c 67 37 24 ff ae a5 c1 ea e8 21 97 bc	41 8d fe 29 85 9a 36 16 e4 06 78 87 9b fd 88 65	41 8d fe 29 9a 36 16 85 78 87 e4 06 65 9b fd 88	2a 47 c4 48 83 e8 18 ba 84 18 27 23 eb 10 0a f3	58 fd 0f 4c 9d ee cc 40 36 38 9b 46 eb 7d ed bd

AES 举例

72 ba cb 04 1e 06 d4 fa b2 20 bc 65 00 6d e7 4e	40 f4 1f f2 72 6f 48 2d 37 b7 65 4d 63 3c 94 2f	40 f4 1f f2 6f 48 2d 72 65 4d 37 b7 2f 63 3c 94	7b 05 42 4a 1e d0 20 40 94 83 18 52 94 c4 43 fb	71 8c 83 cf c7 29 e5 a5 4c 74 ef a9 c2 bf 52 ef
0a 89 c1 85 d9 f9 c5 e5 d8 f7 f7 fb 56 7b 11 14	67 a7 78 97 35 99 a6 d9 61 68 68 0f b1 21 82 fa	67 a7 78 97 99 a6 d9 35 68 0f 61 68 fa b1 21 82	ec 1a c0 80 0c 50 53 c7 3b d7 00 ef b7 22 72 e0	37 bb 38 f7 14 3d d8 7d 93 e7 08 a1 48 f7 a5 4a
db a1 f8 77 18 6d 8b ba a8 30 08 4e ff d5 d7 aa	b9 32 41 f5 ad 3c 3d f4 c2 04 30 2f 16 03 0e ac	b9 32 41 f5 3c 3d f4 ad 30 2f c2 04 ac 16 03 0e	b1 1a 44 17 3d 2f ec b6 0a 6b 2f 42 9f 68 f3 b1	48 f3 cb 3c 26 1b c3 be 45 a2 aa 0b 20 d7 72 38
f9 e9 8f 2b 1b 34 2f 08 4f c9 85 49 bf bf 81 89	99 1e 73 f1 af 18 15 30 84 dd 97 3b 08 08 0c a7	99 1e 73 f1 18 15 30 af 97 3b 84 dd a7 08 08 0c	31 30 3a c2 ac 71 8c c4 46 65 48 eb 6a 1c 31 62	fd 0e c5 f9 0d 16 d5 6b 42 e0 4a 41 cb 1c 6e 56
cc 3e ff 3b a1 67 59 af 04 85 02 aa a1 00 5f 34	4b b2 16 e2 32 85 cb 79 f2 97 77 ac 32 63 cf 18	4b b2 16 e2 85 cb 79 32 77 ac f2 97 18 32 63 cf		b4 ba 7f 86 8e 98 4d 26 f3 13 59 18 52 4e 20 76
ff 08 69 64 0b 53 34 14 84 bf ab 8f 4a 7c 43 b9				

AES 的实现

- 可以在 8 位处理器上非常有效地实现：
  - 字节代替在字节级别上操作，只要求一个 256 字节的表。
  - 行移位是简单的移字节操作。
  - 轮密钥加是按位异或操作。
  - 列混淆变换要求在域 $\mathrm{GF}(2^8)$ 上的乘法，所有的操作都是基于字节的，只要简单地查表即可。
- 可以在 32 位处理器上非常有效地实现：
  - 将操作定义在 32 位的字上。
  - 事先准备好 4 张 256 字的表。
  - 每一轮通过查表并加上 4 次异或即可计算每一列的值。
  - 需要 $4KB$ 空间来存储这 4 张表。

AES 在 32 位处理器上的高效实现

每一轮的四种变换可以归纳为以下运算：

1. 字节代替

$$b_{i,j} = S[a_{i,j}]$$

2. 行移位

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$$

3. 列混淆

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

4. 轮密钥加

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES 在 32 位处理器上的高效实现

可以写成一个式子：

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j-2}] \right) \oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES 在 32 位处理器上的高效实现

- 定义 4 个 256 字的表 $T_i, i = 0, 1, 2, 3$。
- 每个表输入一个 $0 \sim 255$ 范围内的数 x，输出一个字 $T_i[x]$。
- 四个表可以提前计算好，需要存储空间 4KB。

$$T_0[x] = \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \qquad T_1[x] = \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x]$$

$$T_2[x] = \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \qquad T_3[x] = \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x]$$

AES 在 32 位处理器上的高效实现

- 对状态数组一列进行一轮运算可以简化为 4 次查表操作和 4 次异或运算：

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

- 这种紧凑、高效的实现方式是 Rijndael 能被选为 AES 的重要原因之一。