密码学系列讲座

第 1 课：对称加密与哈希函数

lynndell 博士
新火科技 密码学专家 lynndell2010@gmail.com

目录

对称加密与哈希函数的核心原理：为实现随机性

- **线性变换：每 bit 的输入发生变换，影响 50%输出，密文随机性高。**
- **非线性变换：S 盒子，抵抗解方程组攻击、延展攻击。**

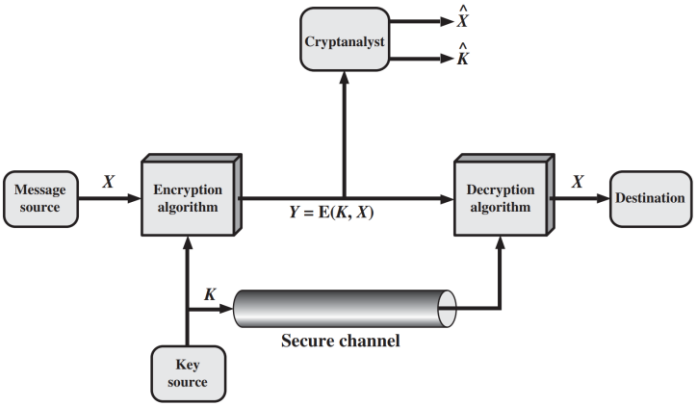
  延展性：基于已有的密文，计算新密文，且新密文能正确解密。

  举例：同态加密具有延展性：
  $$c_1 = Enc_{key}(m_1), c_2 = Enc_{key}(m_2)$$
  $$c_3 = c_1 + c_2 = Enc_{key}(m_1 + m_2)$$

- **轮密钥加/轮常量加：添加一些随机密钥或常量，提高信息商。**

1 对称加密

1.1 基本概念



初始化：双方共享一个保密随机数 K，或使用相同的设备生成一个相同的随机数 K。

加密：发送方使用设备生成随机数 K，对消息 X，如下计算

$$Y := K \oplus X$$

发送密文 Y。

解密：接收方接收密文 Y，使用设备生成随机数 K，如下计算

$$X := Y \oplus K$$

获得消息 X。

双方如何容易地共享保密随机数 K？信道是否一定要安全，不安全可以吗？

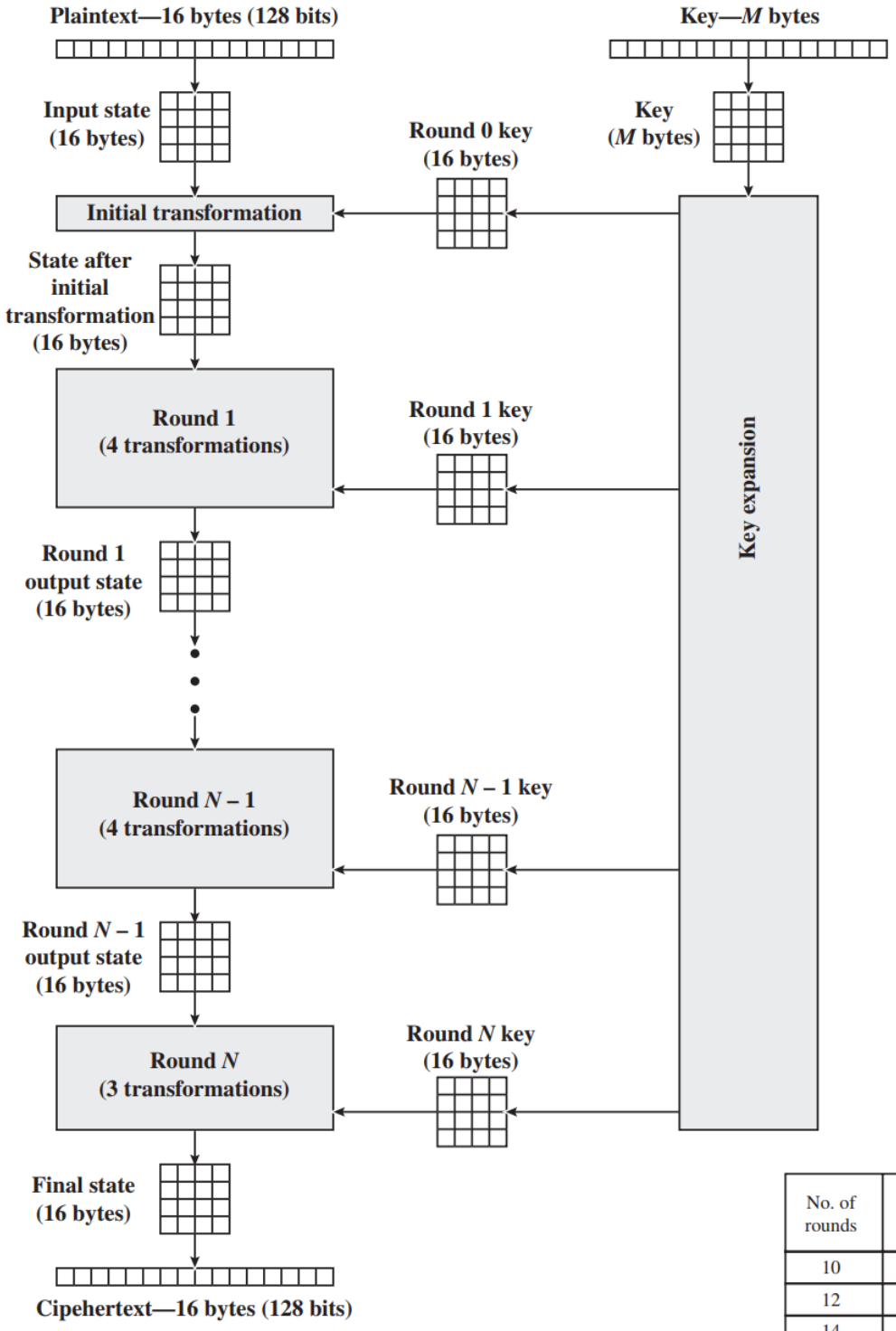
答：可以的，这是公钥密码学的任务。

1.2 高级加密标准 AES

数据加密标准（Data Encryption Standard, DES）的密钥长度是 56 比特，因此算法的理论安全强度是 2^{56}。但是，二十世纪中后期计算机飞速发展，元器件制造工艺的进步使得计算机的处理能力越来越强，DES 将不能提供足够的安全性。**AES 得到全世界很多密码工作者的响应。最终有 5 个候选算法进入最后一轮：Rijndael，Serpent，Twofish，RC6 和 MARS。最终经过安全性分析、软硬件性能评估等严格的步骤，Rijndael 算法获胜。**由比利时两位非常著名的密码学家 Joan Daemen 和 Vincent Rijmen 设计。

1.2.1 加密流程

AES 是一个分组密码，每组 128bit。



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 5.1 AES Encryption Process

轮函数和密钥扩展是 AES 最关键部分。

1.2.2 轮函数

轮函数包含 4 种操作：

- 字节代替（SubBytes）
- 行移位（ShiftRows）
- 列混淆（MixColumns）

- **轮密钥加（AddRoundKey）**

下图给出 AES 加解密的流程，从图中可以看出：**1）解密算法的每一步分别对应加密算法的逆操作，2）加解密所有操作的顺序正好是相反的。正是由于这几点（再加上加密算法与解密算法每步的操作互逆）保证了算法的正确性。**加解密中每轮的密钥分别由种子密钥经过**密钥扩展算法**得到。算法中 16 字节的明文、密文和轮子密钥都以一个 4x4 的矩阵表示。

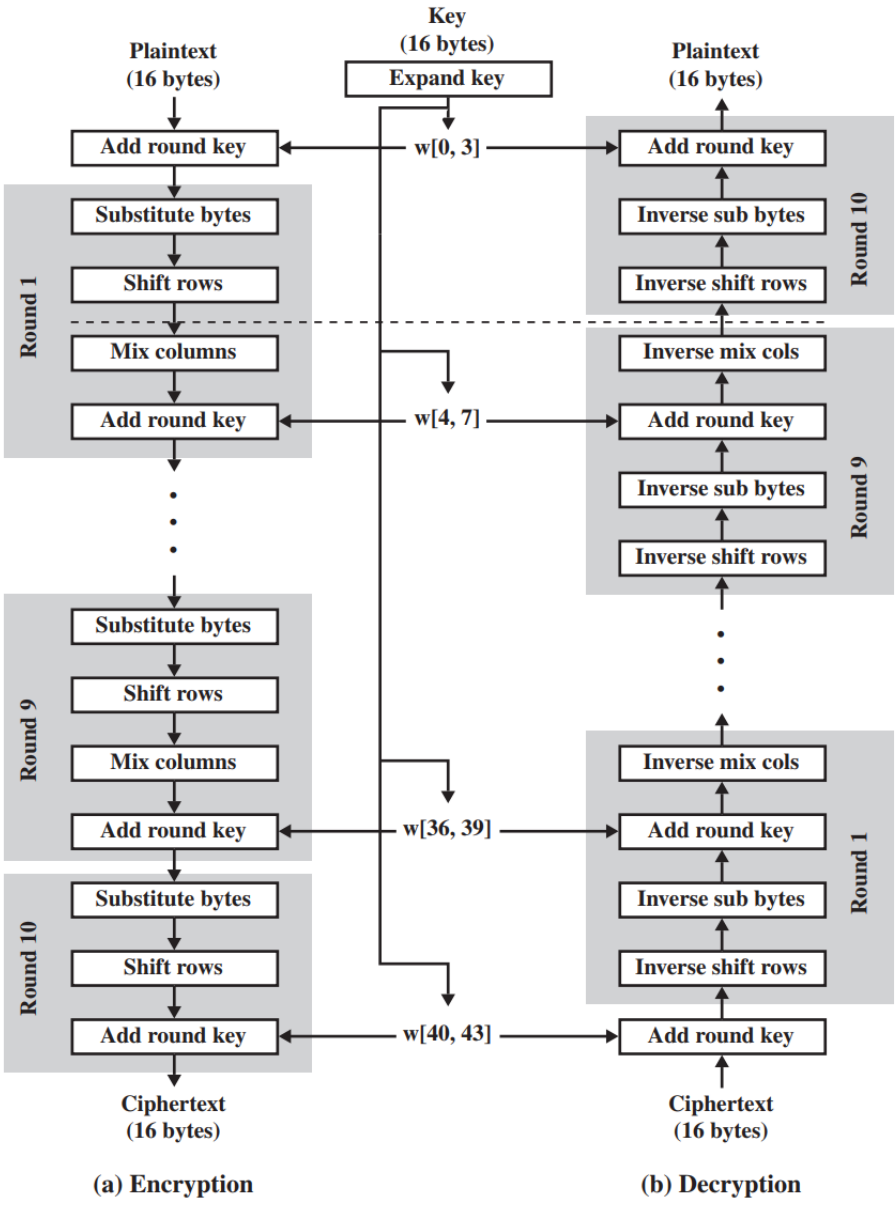


Figure 5.3 AES Encryption and Decryption

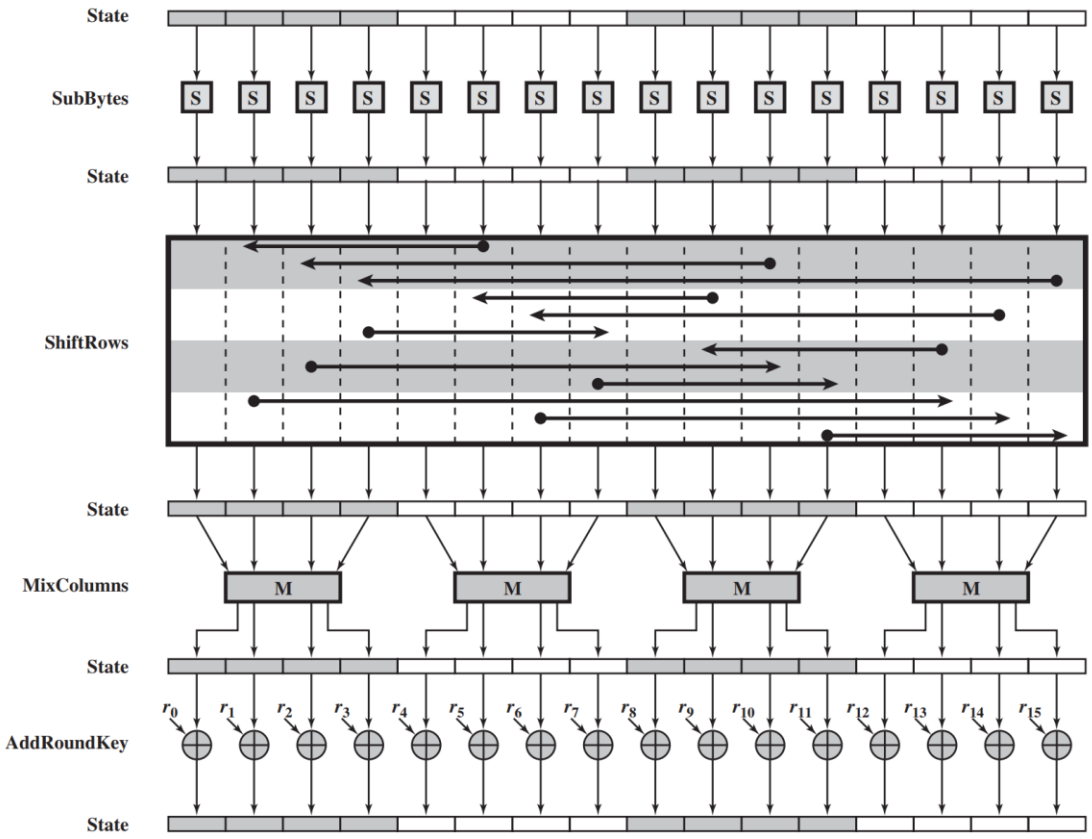


Figure 5.4 AES Encryption Round

字节替代（非线性变换）

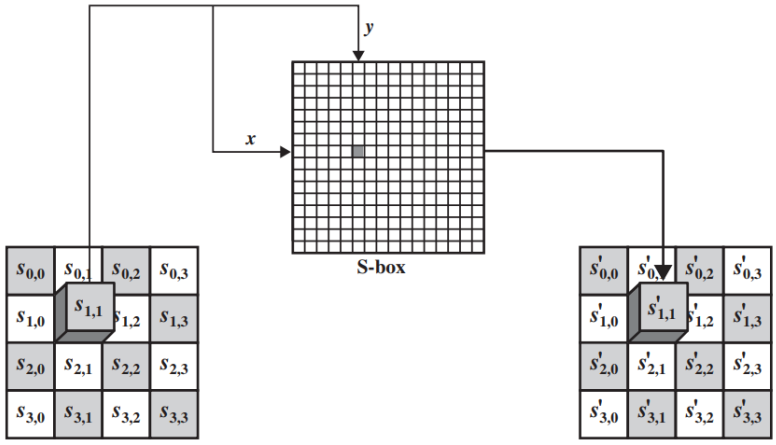
字节代替通过 **S 盒**完成一个字节到另外一个字节的映射。这里直接给出构造好的结果，下图(a)为 S 盒，图(b)为 S^{-1}（S 盒的逆）。S 盒用于提供密码算法的**混淆性**。

S 和 S^{-1} 分别为 16x16 的矩阵，完成一个 8 比特输入到 8 比特输出的映射，

输入的高 4-bit 对应的值作为行标，低 4-bit 对应的值作为列标。

输入字节值为 $a=a_7a_6a_5a_4a_3a_2a_1a_0$，输出值为 $S[a_7a_6a_5a_4][a_3a_2a_1a_0]$，$S^{-1}$ 的变换也同理。

例如：字节 00000000_B 替换后的值为（S[0][0]=）63_H，再通过 S^{-1} 即可得到替换前的值，（S^{-1} [6][3]=）00_H。



(a) Substitute byte transformation

		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

(a) S-box

		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

(b) Inverse S-box

等价描述

1. 按字节值的升序逐行初始化 S 盒子。第 1 行是{00},{01},…,{0F}；第 2 行是{10},{11},…,{1F}。因此，在 **y 行 x 列**的字节值是{yx}

2. S 盒子中每个字节映射为有限域$GF(2^8)$中的逆，其中{00}映射为{00}。

3. S 盒子中的每个字节的 8 位记为$(b_7, b_6, ..., b_0)$，对每个字节进行如下**变换**

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

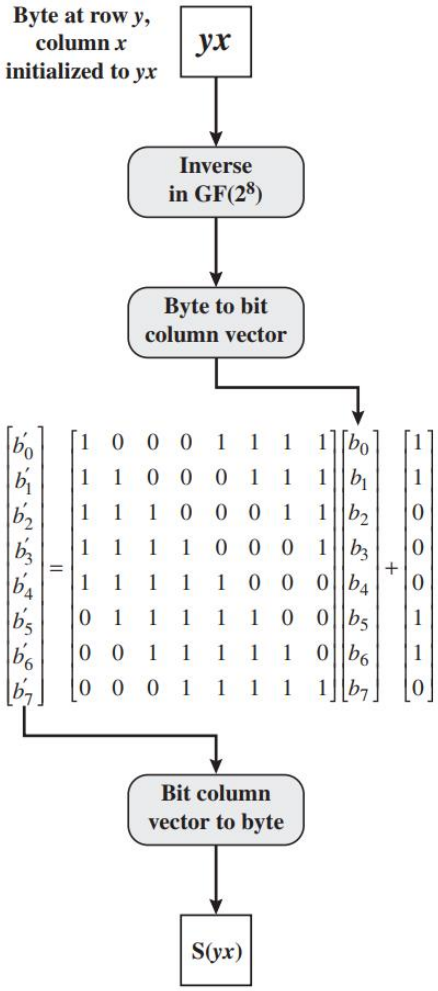
其中 c_i 是值{63}字节 c 的第 i 位 $(c_7, c_6, c_5, c_4, c_3, c_2, c_1) = (01100011)$

对应的**逆变换**为

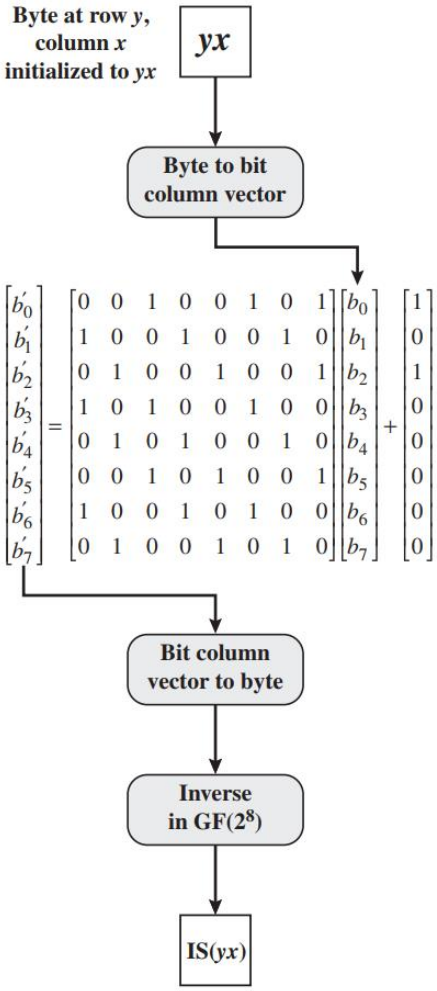
$$b_i' = b_{i+2 \bmod 8} \oplus b_{i+5 \bmod 8} \oplus b_{i+7 \bmod 8} \oplus d_i$$

其中字节 $d = \{05\}$ 或 00000101.

用**矩阵等价描述**



(a) Calculation of byte at row y, column x of S-box



(a) Calculation of byte at row y, column x of IS-box

行移位（线性变换）

行移位是一个 4x4 的矩阵内部字节之间的置换，用于提供算法的**扩散性**。

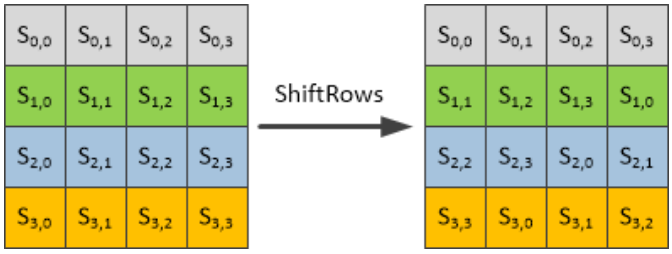
1) 正向行移位 16*8=128

正向行移位用于加密，其原理图如下。其中：

- **第一行保持不变，**
- **第二行循环左移 8 比特，**

- **第三行循环左移 16 比特，**
- **第四行循环左移 24 比特。**

对于矩阵 state，**线性表达：state'[i][j] = state[i][(j+i)%4]，其中 i、j 属于[0,3]**



2) 逆向行移位

逆向行移位即是相反的操作

- **第一行保持不变，**
- **第二行循环右移 8 比特，**
- **第三行循环右移 16 比特，**
- **第四行循环右移 24 比特。**

对于矩阵 state，**线性表达：state'[i][j] = state[i][(4+j-i)%4]；其中 i、j 属于[0,3]。**

列混淆（线性变换）

不可约多项式（本原多项式）：不能写成两个次数较低的多项式之乘积的多项式，即没有因子的多项式。

LIDL94 Lidl, R. and Niederreiter, H. *Introduction to Finite Fields and Their Applications.* Cambridge: Cambridge University Press, 1994.

- **规定 1：字节相加，是位异或运算；**
- **规定 2：字节相乘，有限域内的乘法运算，模不可约多项式 m(x)。**

列混淆计算方法 1：使用域 F_{2^8} 上的**不可约多项式**

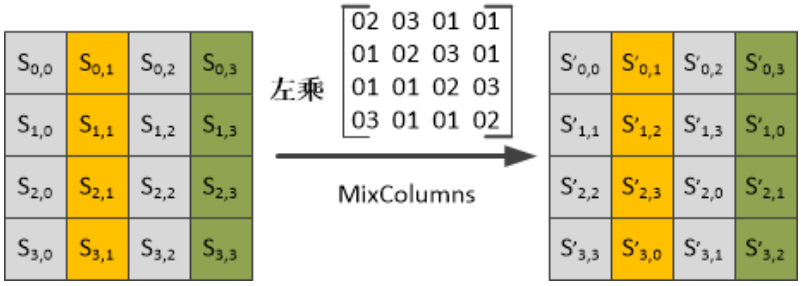
$$m(x) = x^8 + x^4 + x^3 + x + 1$$

举例：

$$\begin{aligned}
&\{02\} \bullet \{C9\} = (0000,0010) \bullet (1100,1001) \\
&= x \bullet (x^7 + x^6 + x^3 + 1) \bmod \big(m(x)\big) \\
&= x^8 + x^7 + x^4 + x - (x^8 + x^4 + x^3 + x + 1) \\
&= x^7 + x^3 + 1 = (1000,1001) = \{89\}
\end{aligned}$$

列混淆计算方法 2：

正向列混淆的原理图如下：



根据矩阵的乘法可知，在列混淆的过程中，每个字节对应的值只与该列的 4 个值有关系。**此处的乘法和加法都是定义在 GF(2^8)上的，**需要注意如下几点：

（1） 将某个字节所对应的值乘以 2，其结果就是将该值的二进制位左移一位。

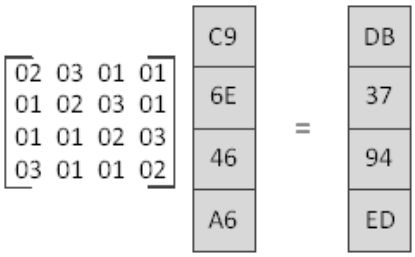
如果原始值的最高位为 1，则还需要将移位后的结果异或常量 00011011；

（2）乘法对加法满足分配率

$$07\ S_{0,0}=(01\oplus 02\oplus 04)\ S_{0,0}=(\ 01\ S_{0,0})\oplus\ (02\ S_{0,0})\ \oplus(04\ S_{0,0})$$

（3）加法是模 2^8 加法（异或运算）。

举例：



$$S'_{0,0}=(02\bullet C9)\oplus(03\bullet 6E)\oplus(01\bullet 46)\oplus(01\bullet A6)$$

其中：

$$02\bullet C9=02\bullet 11001001_B=10010010_B\oplus 00011011_B=10001001_B$$

$$03\bullet 6E=(01\oplus 02)\bullet 6E=01101110_B\oplus 11011100_B=10110010_B$$

$$01\bullet 46=01000110_B$$

$$01\bullet A6=10100110_B$$

则：

$$S'_{0,0}=10001001_B\oplus 10110010_B\oplus 01000110_B\oplus 10100110_B$$

$$=11011011_B=DB$$

第 1 个：在计算 02 与 C9 的乘积时，由于 C9 对应最左边的比特为 1，因此需要将 C9 左移一位后的值与(0001 1011)求异或。同理可以求出另外几个值。

第 2 个： $03\bullet 6E=(01\oplus 02)\bullet 6E=(01\bullet 6E)\oplus(02\bullet 6E)=0 1101110_B\oplus 1101110 0_B$

2) 逆向列混淆

逆向列混淆的原理图如下：

由于：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

两个矩阵互逆，经过一次逆向列混淆后即可恢复明文。

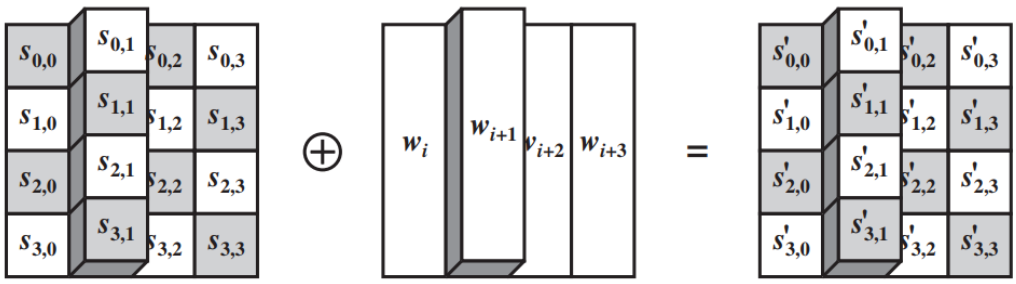
注意：AES 加密计算复杂度更低，AES 解密计算复杂度更高。

设计思想：CFB/OFB/CTR 模式仅使用 AES 加密算法，用于生成随机数，实现加密，所以需要加密矩阵的值更小，效率更高。

轮密钥加

加密：每轮的输入与轮子密钥异或一次，c=m⊕k

解密：再异或上该轮的轮子密钥即可恢复明文，m=c⊕k



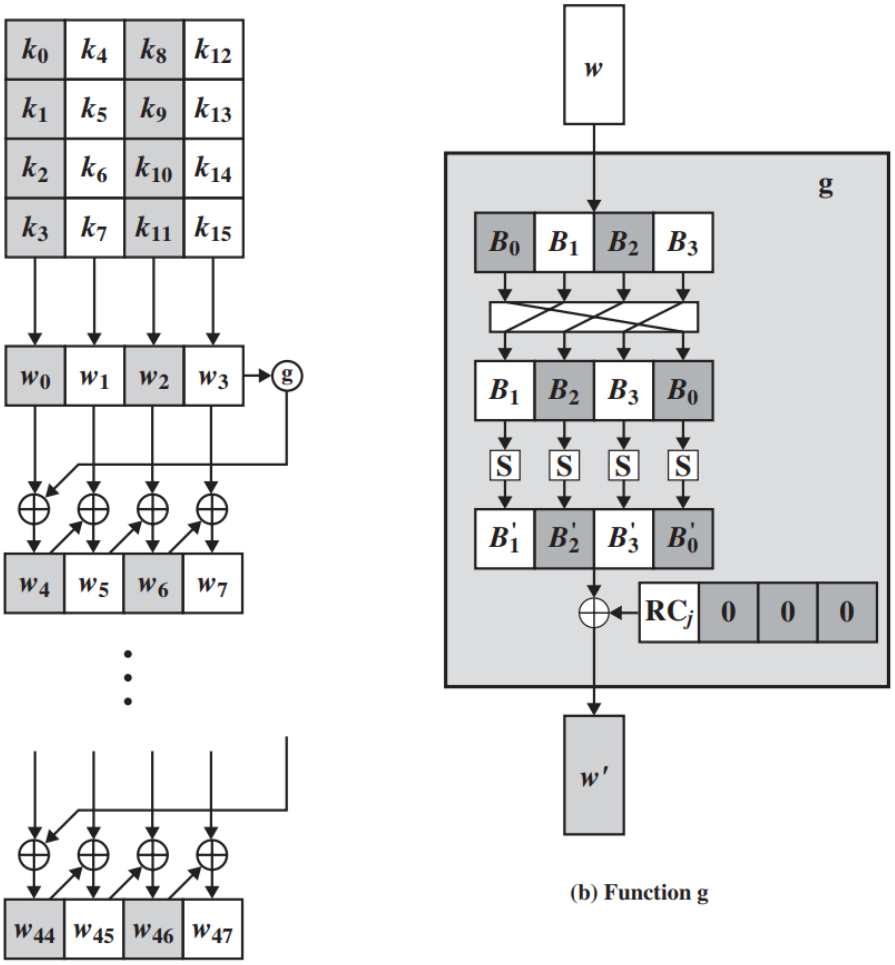
(b) Add round key transformation

1.2.3 密钥扩展算法

16byte*8=128bit

密钥串行扩展

g 函数内部：线性变换、非线性变换、轮常量加



(b) Function g

(a) Overall algorithm

Figure 5.9 AES Key Expansion

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    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++)
    {
     temp = w[i - 1];
     if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                                ⊕ Rcon[i/4];
     w[i] = w[i-4] ⊕ temp
    }
}
```

密钥扩展：将 16bytes 扩展为 43bytes

- 将种子密钥按图(a)的格式排列，k_0、k_1、……、k_{15} 依次表示种子密钥的一个字节；排列后用 4 个 32 比特的字表示，记为 w[0]、w[1]、w[2]、w[3]；4 组，每组 4 个字节
- 按照如下方式，**串行**求解 w[j]，其中 j 是整数并且属于[4,43]；

如果 j%4=0,则 w[j]=w[j-4]⊕g(w[j-1])，j=4,8,12,16,20,24,28,32,36,40;
否则 w[j]=w[j-4]⊕w[j-1]；j=5,6,7; 9,10,11; 13,14,15; 17,18,19; 21,22,23; 25,26,27; 29,30,31; 33,34,35; 37,38,39; 41,42,43;

函数 **g(x)原理：**

- 将 w 循环左移 8 比特；
- 分别对每个字节做 **S 盒置换；（使用上文的(a) S-box 盒子）**
- 与 32 比特的常量（RC[j/4],0,0,0）进行异或，RC 是一个一维数组，其值如下。

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

举例：

Key Words	Auxiliary Function
w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad d6 w3 = af 7f 67 98	RotWord (w3) = 7f 67 98 af = x1 SubWord (x1) = d2 85 46 79 = y1 Rcon (1) = 01 00 00 00 y1 ⊕ Rcon (1) = d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord (w7) = 81 15 a7 38 = x2 SubWord (x2) = 0c 59 5c 07 = y2 Rcon (2) = 02 00 00 00 y2 ⊕ Rcon (2) = 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord (w11) = ff d3 c6 e6 = x3 SubWord (x3) = 16 66 b4 83 = y3 Rcon (3) = 04 00 00 00 y3 ⊕ Rcon (3) = 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0	RotWord (w15) = ae 7e c0 b1 = x4 SubWord (x4) = e4 f3 ba c8 = y4 Rcon (4) = 08 00 00 00 y4 ⊕ Rcon (4) = ec f3 ba c8 = 4

Key Words	Auxiliary Function
w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord (w19) = 8c dd 50 43 = x5 SubWord (x5) = 64 c1 53 1a = y5 Rcon(5) = 10 00 00 00 y5 ⊕ Rcon (5) = 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord (w23) = 40 46 bd 4c = x6 SubWord (x6) = 09 5a 7a 29 = y6 Rcon(6) = 20 00 00 00 y6 ⊕ Rcon(6) = 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord (w27) = a5 a9 ef cf = x7 SubWord (x7) = 06 d3 bf 8a = y7 Rcon (7) = 40 00 00 00 y7 ⊕ Rcon(7) = 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord (w31) = 7d a1 4a f7 = x8 SubWord (x8) = ff 32 d6 68 = y8 Rcon (8) = 80 00 00 00 y8 ⊕ Rcon(8) = 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w32 = 3c be 0b 3	RotWord (w35) = be 0b 38 3c = x9 SubWord (x9) = ae 2b 07 eb = y9 Rcon (9) = 1B 00 00 00 y9 ⊕ Rcon (9) = b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord (w39) = 6b 41 56 f9 = x10 SubWord (x10) = 7f 83 b1 99 = y10 Rcon (10) = 36 00 00 00 y10 ⊕ Rcon (10) = 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	

密码算法要求是可逆的，解密算法才能恢复明文。

分组密码每次加密 128bit 的数据，如果数据为 128bit~300Gbit 数据，应该如何加密？
答：分组密码循环调用多次，实现对任意长数据的加密。

1.3 对称加密五种工作模式

模　式	描　述	典型应用
电码本（ECB）	用相同的密钥分别对明文分组独立加密	● 单个数据的安全传输（如一个加密密钥）
密文分组链接（CBC）	加密算法的输入是上一个密文组和下一个明文组的异或	● 面向分组的通用传输 ● 认证
密文反馈（CFB）	一次处理 s 位，上一块密文作为加密算法的输入，产生的伪随机数输出与明文异或作为下一单元的密文	● 面向数据流的通用传输 ● 认证
输出反馈（OFB）	与 CFB 类似，只是加密算法的输入是上一次加密的输出，且使用整个分组	● 噪声信道上的数据流的传输（如卫星通信）
计数器（CTR）	每个明文分组都与一个经过加密的计数器相异或。对每个后续分组计数器递增	● 面向分组的通用传输 ● 用于高速需求

1.3.1 ECB 电码本模式

ECB 模式是分组密码的基本工作方式。在该模式下,每个加密区块按顺序进行独立加密，得到独立的密文区块,每个加密区块的结果都不会被其他区块影响,用此方式，可用平行处理实

施加速加、解密运算,且在网络传输时任何一个区块出现错误,也不存在影响到其他区块传输的结果,这是该模式的好处。

ECB 模式的不足是易使明文的数据模式暴露。很多数据均存在固有模式，这主要是由数据结构与数据冗余导致的。若无任何措施，针对在需加密的文件里出现多回的明文，这部分明文如果刚好是加密区块的大小，则可能会得到一样的密文，且密文内容如果受到剪贴、代替，也很难被发现。

优点：能并行加密和解密。

缺点：如果 P_1=P_2，则 C_1=C_2 密文的随机性很差。

应用场景：仅加密 128bit 的数据。

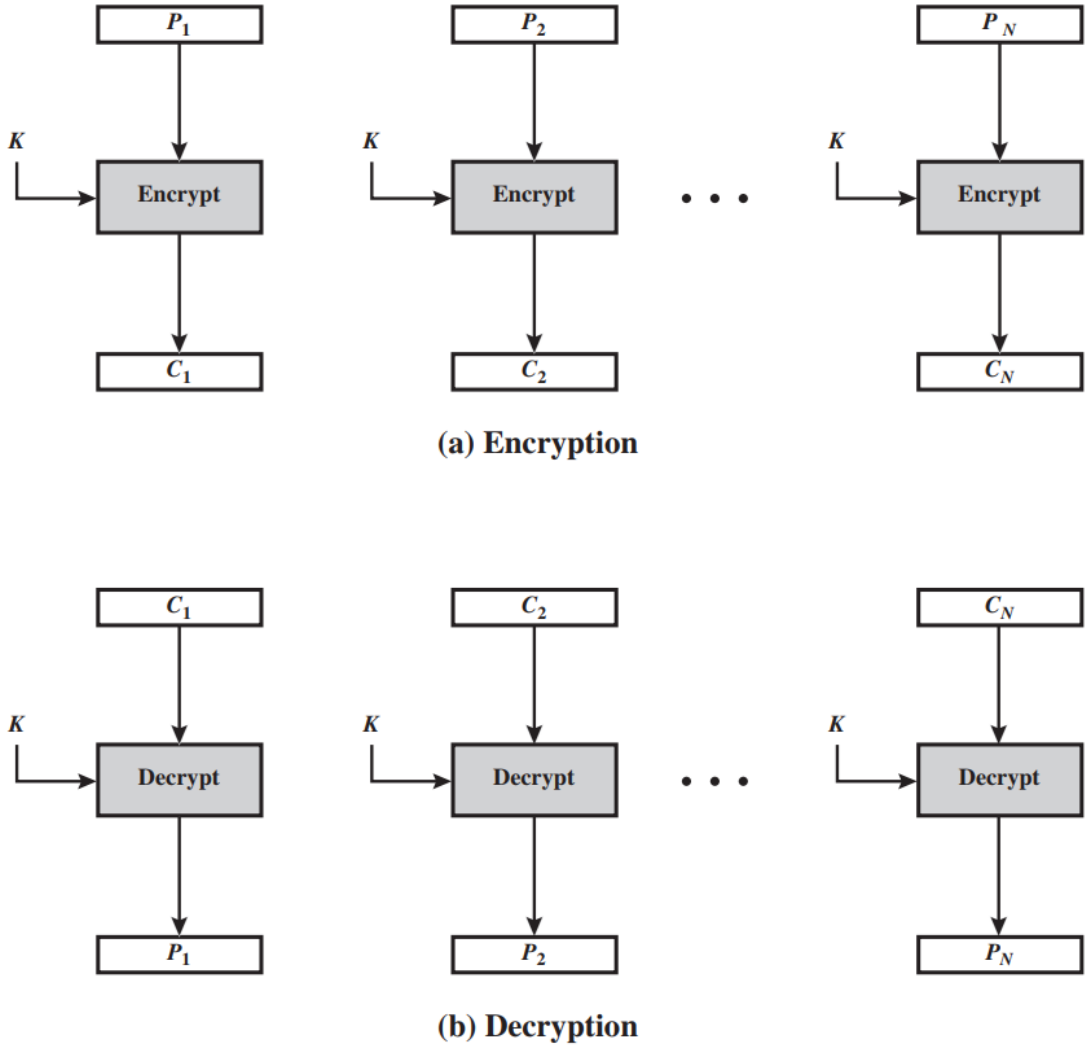


Figure 6.3 Electronic Codebook (ECB) Mode

1.3.2 CBC 密文分组链接模式

第一个加密区块先与初始向量做异或运算,再进行加密。其他每个加密区块在加密之前，必须与前一个加密区块的密文做一次异或运算，再进行加密。每个区块的加密结果都会被前面全部区块内容的影响，因此尽管在明文里出现多次一样的明文，也会得到不一样的密文。还有，密文内容如果遇到剪贴、替换、或于网络传输时出现错误，则它后面的密文会被破坏，不能顺利解密还原，这是这一模式的优点也是缺点。

其次，一定得选取 1 个初始向量来加密第 1 个区块，且加密作业时不能用平行处理加速加密运算，不过解密运算，做异或的加密区块结果已经有了，则还可用平行处理加速。

密文没有规律，引入雪崩效应：

即是优点（随机性极高） 也是缺点（传输错误，则影响后续解密正确性）。

最后一个分组需要填充

需要初始化向量

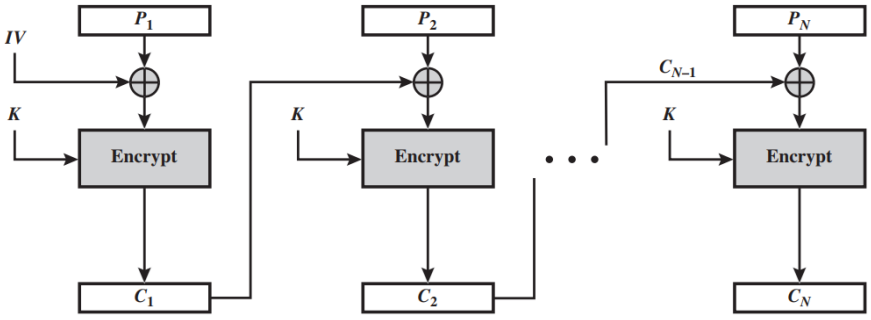
串行加密解密。

雪崩效应：1bit 传输错误，会影响后续解密。

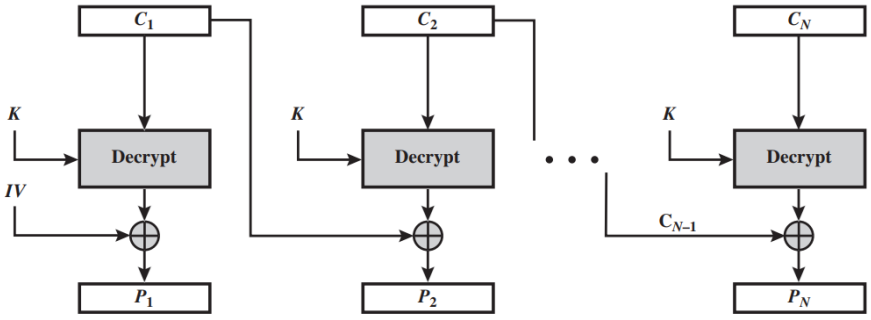
应用场景：

数据本地加密解密且数据量较小（或对速度无要求），则可用该方案。

远程传输则不使用该方案。



(a) Encryption



(b) Decryption

1.3.3 CFB 密文反馈模式

简单讲就是前一模块的密文输出，参与下一模块的加解密

密文没有规律，明文分组和一个数据流进行异或操作，生成密文

需要初始化向量

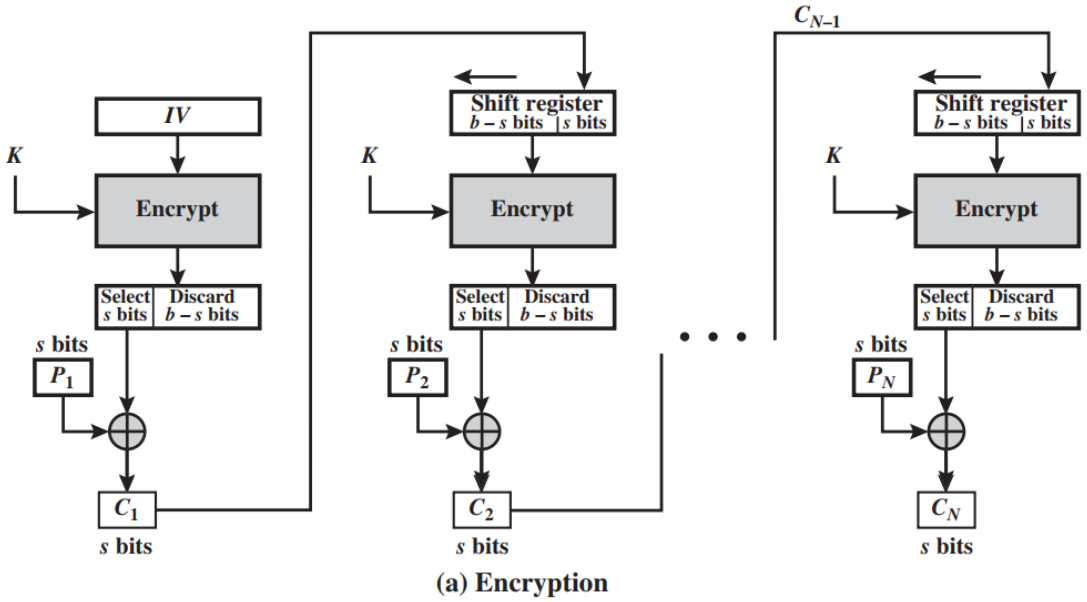
不需要填充

不能并行运算；

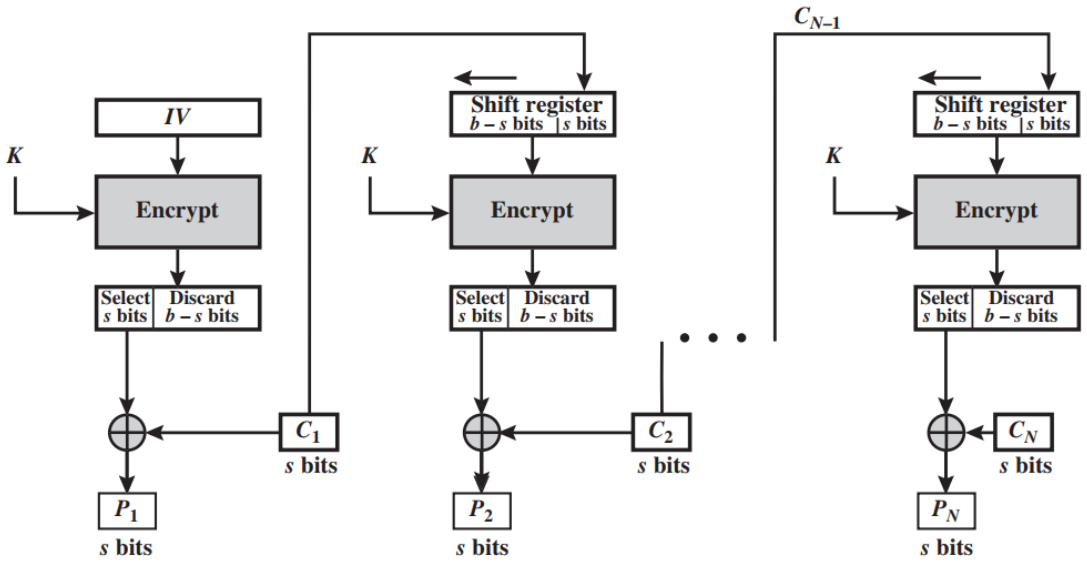
密文传输错误，则影响后续解密正确性。

每次加密 sbit，且串行，因此效率较低。

加密和解密过程均使用 AES 加密算法，计算效率较高。



(a) Encryption



(b) Decryption

1.3.4 OFB 输出反馈模式

初始化向量一次加密并参与各分组的加密过程

特点，密文没有规律，明文分组和一个数据流按位异或生成密文

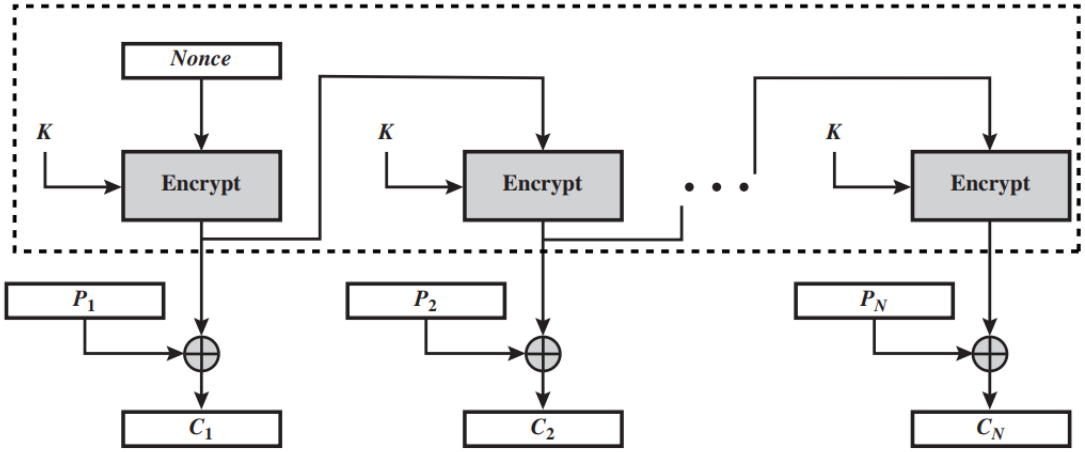
需要初始化向量

不需要填充

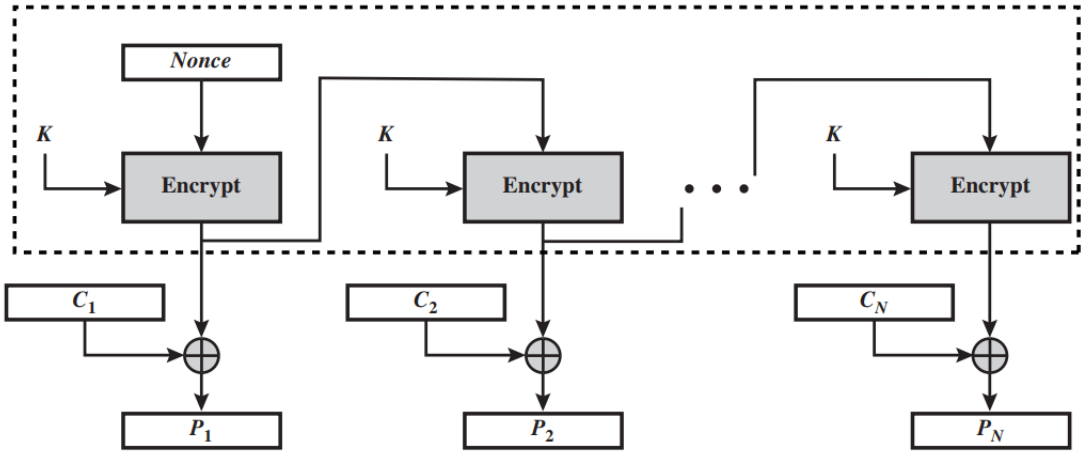
不能并行运算

加密和解密过程均使用 AES 加密算法，计算效率较高。

本质上是分组密码算法构造出流密码算法，生成任意长的随机数对消息加密。



(a) Encryption



(b) Decryption

256bit 的随机数 IV，需要保密传输

加密： $Y := K \oplus X$

解密： $X := Y \oplus K$

1.3.5 CTR 计数器模式

用计数器代替 OFB 模式中的向量

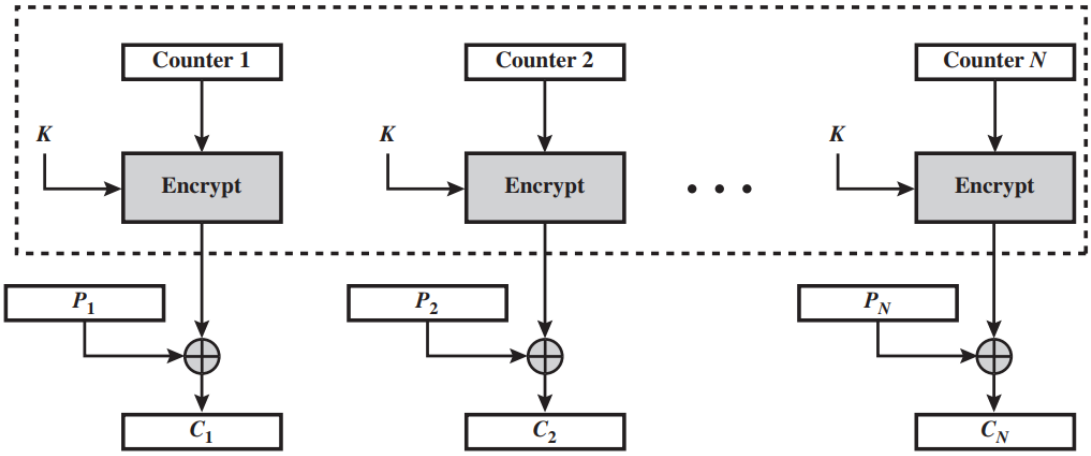
Counter1= IV，可公开传输

不需要填充，**独立性高，可并行运算**

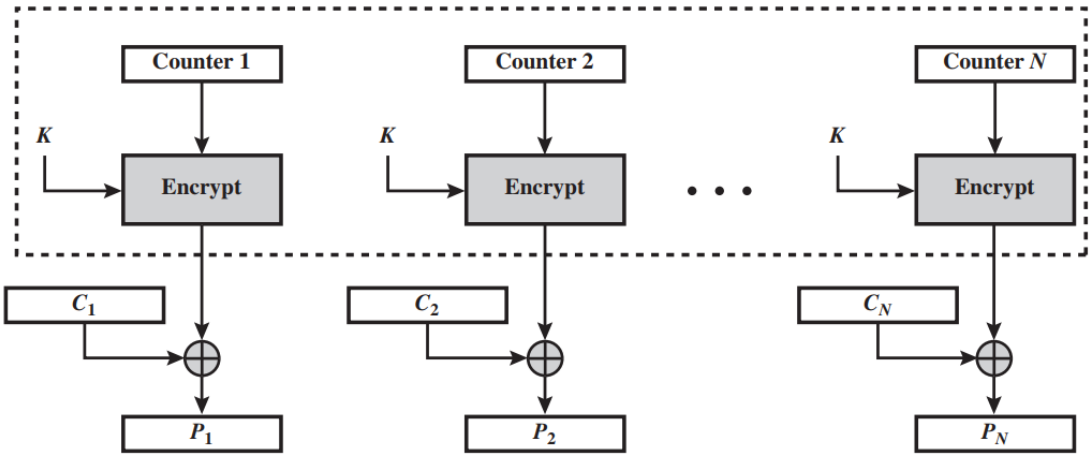
ECIES 加密方案中，调用 AES 对称加密时，就使用了该方案。

加密和解密过程均使用 AES 加密算法，计算效率较高。

本质上是分组密码算法构造流密码算法，生成任意长随机数 K 对消息 X 加密。



(a) Encryption



(b) Decryption

加密： $Y := K \oplus X$

解密： $X := Y \oplus K$

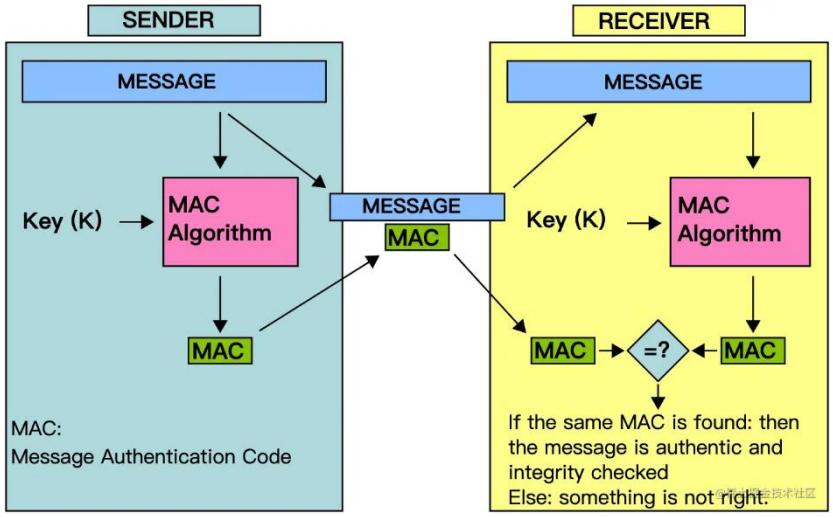
1.3.6 对比分析与应用场景

模式	名称	优点	缺点	备注
ECB	电子密码本模式 electronic codebock	支持并发加解密	重复明文反应在密文中 不能抵抗重放攻击	不推荐使用
CBC	密文分组链接模式 Cipher Block Chaining	抵御重放共计 可以并行解密	不能并行加密 错误的密文位影响后续密文解密	推荐使用
CFB	密文反馈模式 Cipher FeedBack	不需要填充 并行解密 解密分组间无依赖	不能并行加密 错误的密文位影响后续密文解密	不推荐
OFB	输出反馈模式 OutPut FeedBack	不需要填充 出现错误时仅对应明文出错	不支持并行计算	不推荐
CTR	计数器模式 Counter	不需要填充 支持并行计算	反转密文分组中部分bit时，对应明文也会反转	推荐

1.3.7 GCM 加密模式

目前最推荐的加密模式！GCM=MAC+CRT

消息认证码 (Message Authentication Code, MAC)

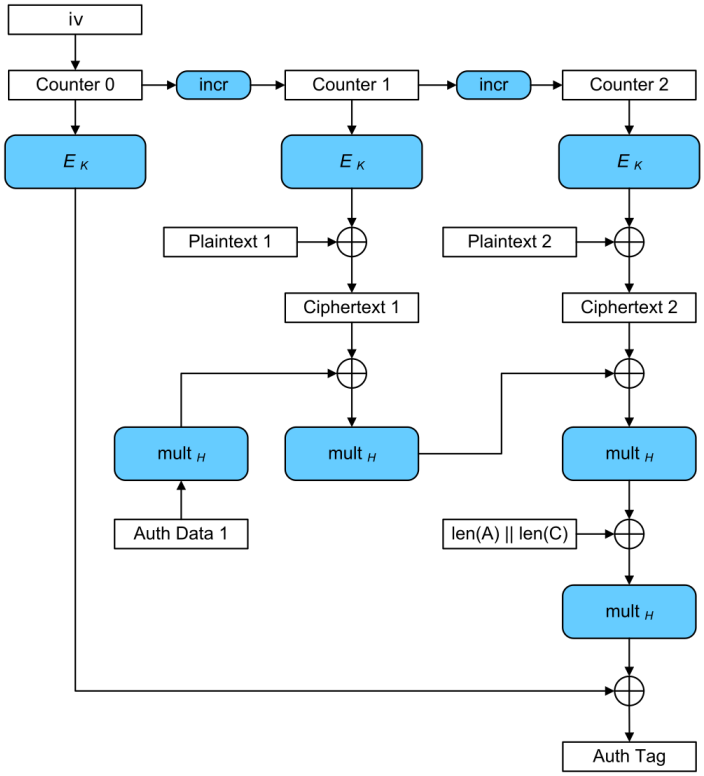


双方提前共享一个密钥，发送者使用共享密钥和**数据**计算**哈希值**，并随消息一起发送。**接收者通过共享密钥计算收到消息的 MAC 值，与接收的 MAC 值做比较，从而判断消息是否被改过(完整性)。**对于篡改者，由于没有密钥(认证)，也就无法对篡改后的消息计算 MAC 值。

GCM 是认证加密模式，将 MAC 和 CTR 计数器模式耦合，结合二者优点。

CTR 计数器模式：确保数据的**保密性、并行性、counter=IV 可公开等优点**。

MAC：提供附加消息的**完整性**校验。



发送方：

（1）CTR 模式下，先对块进行顺序编号，然后将该块编号与初始向量(IV)组合，并使用密钥 k，对输入做 AES 加密，然后，将加密的结果与明文进行 XOR 运算来生成密文。应该对每次加密使用不同的 IV。

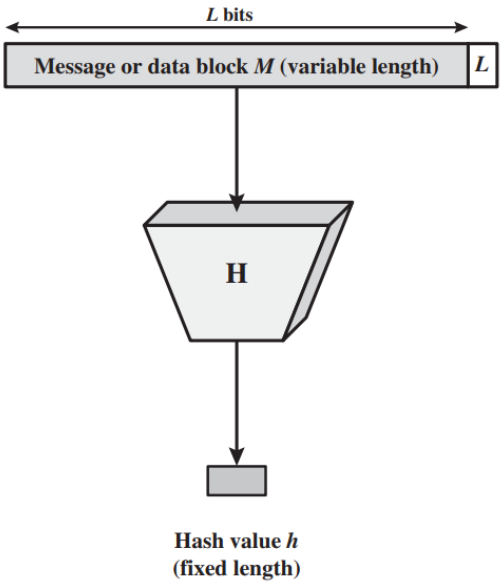
（2）对于附加消息（**共享秘钥**），使用密钥 H(由密钥 K 得出)，运行 GMAC，将结果与密文进行 XOR 运算，生成用于验证数据完整性的**身份验证标签**。

接收方：

密文接收者会收到一条完整的消息，包含密文、IV(计数器 CTR 的初始值)、身份验证标签(MAC 值)。

2.哈希函数

哈希函数：将任意长度的消息，映射为一个固定长（256bit）的随机数。这段随机数称为**消息摘要**。



关键性质：

[1] **单向性：已知哈希值 Y，无法在多项式时间内计算出哈希原象 X；**

[2] **弱抗碰撞性：已知(X, Y)，无法在多项式时间找到 X'，使得 Y=Hash(X')；**

[3] **强抗碰撞性：攻击者无法寻找 X, X'，满足 $X \neq X', hash(X) == hash(X')$；**

[4] **压缩性：通常是将 512bit 的数据压缩为 256bit；不足 512bit 则填充。**

[5] **随机性：输出的 Y 是 256bit 的 0/1 字符串是随机的；**

[6] **可重复性：如果输入 x1=x2，则输出的哈希值 Y1=Y2。函数的基本性质**

2.1 SHA2

2.1.1. 常量与基本运算

基础工具包括：8+64 个初始常量、信息预处理（数据填充）、轮函数 F

2.1.1.1 常量

8 个初值常量如下：（256bit = 8*8*4）

h0 := 0x**6a0**9e667

h1 := 0xbb67ae85

h2 := 0x3c6ef372

h3 := 0xa54ff53a

h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

来源：对自然数中前 8 个质数（2,3,5,7,11,13,17,19）的**平方根**的小数部分取前 32bit。
例如：$ \sqrt{2} $小数部分约为 0.414213562373095048

$$0.414213562373095048 \approx \mathbf{6}*16^{-1}+\mathbf{a}*16^{-2}+\mathbf{0}*16^{-3}+...$$

所以，质数 2 的平方根的小数部分**取前 32bit** 就对应出了 0x**6a0**9e667。
每个参数都有来源根据，没有后门。

64 个常量如下：

428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2

来源：与 8 个哈希初值类似，这些常量是对自然数中前 64 个质数(2,3,5,7,11,13,17,19,23,29,31,37, 41,43,47,53,59,61,67,71,73,79,83,89,97…)的**立方根的小数部分取前 32bit。**

2.1.1.2 基本运算

6 个逻辑运算：
前 4 个计算用于 64 轮循环（**用于数据非线性压缩**）
后 2 个计算用于 16 个字扩展为 64 个字（**用于数据非线性扩展**）
字=32 比特

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Ma(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma 0(x) = S^{2}(x) \oplus S^{13}(x) \oplus S^{22}(x)$$
$$\Sigma 1(x) = S^{6}(x) \oplus S^{11}(x) \oplus S^{25}(x)$$

$$\sigma 0(x) = S^{7}(x) \oplus S^{18}(x) \oplus R^{3}(x)$$
$$\sigma 1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

∧ 按位“与”
¬ 按位“补”
⊕ 按位“异或”
S^n 循环右移 n 个 bit
R^n 右移 n 个 bit

2.1.2 数据预处理

数据填充使整个消息长度和结构满足规定。
信息的预处理分为 2 步：附加填充比特和附加长度。

2.1.2.1 STEP1：填充比特

填充规则：先补第一个比特为 1，然后都补 k 个 0，直到长度满足对 512 取模后余数是 448。剩余 64bit 填充数据长度。

- **msg'= {msg, 1, 0,…0, msg-len-64bit}**
- **计算方法：(Len(msg)+1+k+64)/512=0，寻找最小自然数 k。**

需要注意的是，信息必须进行填充，也就是说，即使长度已经满足对 512 取模后余数是 448，补位也必须要进行，这时要填充 512 个比特。
因此，填充是至少补一位，最多补 512 位。

2.1.3. 计算摘要

SHA256 算法的主体部分，即消息摘要是如何计算的。
首先：将消息分解成 512-bit 大小的块。

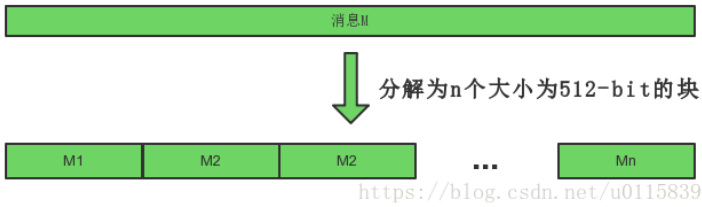


图 1：消息分为 n 个块，|M| = 512 * n

消息 M 可以被分解为 n 个块，整个算法需要完成 n 次迭代，n 次迭代的结果就是最终的哈希值，即 256bit 的数字摘要。

一个 256-bit 的摘要的**初始值 H0**，经过第一个数据块进行运算，得到 H1，即完成了第一次迭代 H1 经过第二个数据块得到 H2，……，依次处理，最后得到 Hn，Hn 即为最终的 256-bit 消息摘要将每次迭代进行的映射

$$H_i := Map(H_{i-1}, M_i)$$

于是迭代可以更形象的展示为：

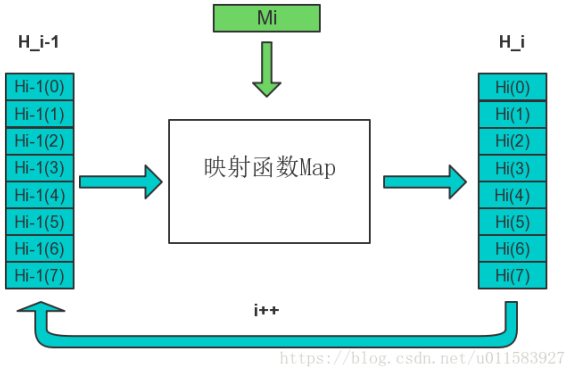


图 2：迭代过程

图中 256-bit 的 Hi 被描述 8 个小块，这是因为 **SHA256 算法中的最小运算单元称为“字”（Word），一个字是 32 位。**

第一次迭代中，映射的初值设置为前面介绍的 8 个哈希初值，如下图所示：

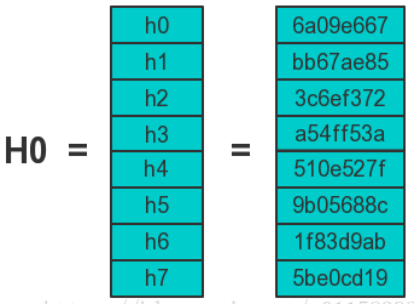


图 3：哈希初值

下面开始介绍每一次迭代的内容，即映射 Map(H_{i-1}) = H_i 的具体算法

Map 映射函数

也就是对称加密中的轮函数 Round Function

STEP1：扩展函数：输入：512bits (16 个字)，输出 2048bits(64 个字)；每个字 32bit

对于每个块 M(512bits=16*32)，分解为 **16 个 32-bit 的 big-endian 的字** w[0], …, w[15];

1. **起始状态：前 16 个字**直接由消息的第 1 个块分解得到：512bit= **w[0], …, w[15]**
2. **其余的 48 个字**由如下**迭代公式**得到：$W_t=\sigma_1(W_{t-2})+W_{t-7}+\sigma_0(W_{t-15})+W_{t-16}$

$W_{16}=\sigma_1(W_{14})+W_{11}+\sigma_0(W_1)+W_0$

$W_{17}=\sigma_1(W_{15})+W_{12}+\sigma_0(W_2)+W_1$

,…,

W_{64}

$\sigma0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$

$\sigma1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$

$\wedge$ 按位“与”

$\neg$ 按位“补”

$\oplus$ **按位“异或”**

S^n **循环**右移 n 个 bit

R^n 右移 n 个 bit

STEP2：压缩函数：进行 64 次循环；输入 64 个字和 64 个常量 k_i

映射函数 $H_i := Map(H_{i-1}, M_i)$ 包含 64 次循环

即进行 64 次循环即可完成一次迭代
每次加密循环可以由下图描述：
256 = 8 * 32

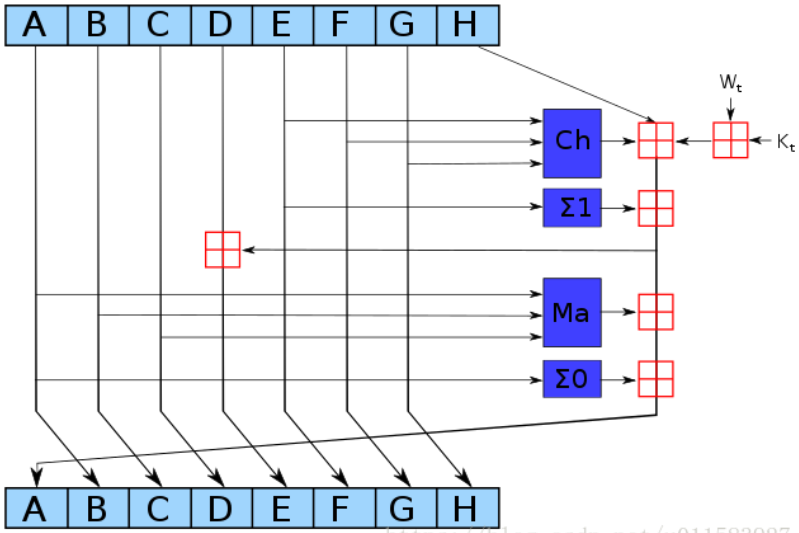


图 4：64 次循环 **轮函数**

- Ch(x, y, z) = (x ∧ y) ⊕ (¬x ∧ z)
- Ma(x, y, z) = (x ∧ y) ⊕ (x ∧ z) ⊕ (y ∧ z)
- Σ0(x) = S^2(x) ⊕ S^{13}(x) ⊕ S^{22}(x)
- Σ1(x) = S^6(x) ⊕ S^{11}(x) ⊕ S^{25}(x)

图中，ABCDEFGH 这 8 个字（word）在按照一定的规则进行更新，其中
深蓝色方块是事先定义好的非线性逻辑函数；
红色田字方块代是：相加后 mod2^{32}，其中一个红色方框是字与常量的模加

$$W_t + K_t \ (mod 2^{32})$$

其中，是 K_t 是 64 个常量。

ABCDEFGH 一开始的**初始值**分别为 H_{i-1}(0),H_{i-1}(1),…,H_{i-1}(7)。
Kt 是 64 个常量，每次循环使用 1 个常量。
Wt 是本区块产生第 t 个 word。原消息被切成固定长度 512-bit 的区块，对每一个区块，产生 64 个 word，通过重复运行循环 n 次对 ABCDEFGH 这八个字循环加密。
Add the compressed chunk to the current hash value:

h0’ := h0 + A
h1’ := h1 + B
h2’ := h2 + C
h3’ := h3 + D
h4’ := h4 + E
h5’ := h5 + F

h6’ := h6 + G

h7’ := h7 + H

最后一次循环所产生的八个字合起来即是第 i 个块对应到的哈希值 H_i。

256bits **hash**:= h0’ || h1’ || h2’ || h3’ || h4’ || h5’ || h6’ || h7’

MD5 和 SHA1 已经不安全，所以 NIST 于 2007 年 12 月发表公告，征集新 SHA3 算法。

2010 年 10 月，第 2 轮遴选结束，入选 5 个算法： BLAKE/Grøst1/JH/Keccak/Skein

2.2 SHA3-BLAKE

BLAKE 算法

由瑞士的 Aumasson 等人设计，采用了 HAIFA 算法迭代结构，其中的压缩算法基于 ChaCha 流密码，内部结构采用 Davies-Meyer 模式，**在压缩函数中，采用模余 322 加法运算与 XOR 运算，实现计算的非线性。**

针对 MD5 结构弱点，BLAKE 算法的压缩函数中加入随机盐(salt)与计数器

整个算法具有良好安全性，且是主流哈希算法中速度最快的。

BLAKE 的算法

- 消息填充：m’←{m,10000.....001,64bitsLen}
- **计算方法：：(Len(msg)+2+k+64)/512=0，寻找最小自然数 k。**

将填充后的消息以 512 位分块，然后依次输入**压缩函数**。

$$Input:(Block_i,s,l_i);$$
$$Output:h_L$$
$$let(h_0:=IV)$$
$$for(i=0,i++,i\le L-1)\{$$
$$h_{i+1}:=compress(h_i,Block_i,s,l_i)$$
$$\}$$

其中 s 为随机盐(salt)，由使用者决定；l_i 为当前已输入的消息长度；compress 表示 BLAKE 算法的压缩函数。

IV_0 = 0x6a09e667f3bcc908	// Frac(sqrt(2))
IV_1 = 0xbb67ae8584caa73b	// Frac(sqrt(3))
IV_2 = 0x3c6ef372fe94f82b	// Frac(sqrt(5))
IV_3 = 0xa54ff53a5f1d36f1	// Frac(sqrt(7))
IV_4 = 0x510e527fade682d1	// Frac(sqrt(11))
IV_5 = 0x9b05688c2b3e6c1f	// Frac(sqrt(13))
IV_6 = 0x1f83d9abfb41bd6b	// Frac(sqrt(17))
IV_7 = 0x5be0cd19137e2179	// Frac(sqrt(19))

常量：

链值 $ChainValue:h=h_0,...,h_7$，随机盐 $salt:s=s_0,s_1,s_2,s_3$，长度计数器 $t=t_0,t_1$，

每次压缩 512bit 消息 $M=M_0,...,M_{15},16\times32=512bit$，16 个字，每个字是 32bit。

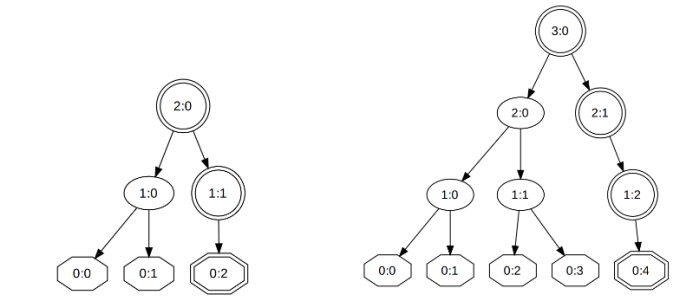
如果是最后一个**数据块**，则 $f_0=\{\text{FF...FF}\}$，否则 $f_0=\{\text{00...00}\}$。

如果是最后一个**节点**，则 $f_1=\{\text{FF...FF}\}$，否则 $f_1=\{\text{00...00}\}$。

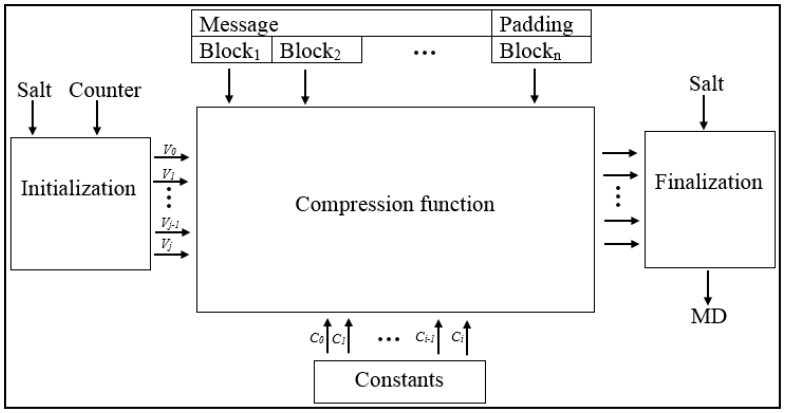
$input\{h_0,...,h_7;s_0,..,s_3;t_0,t_1;\ f_0,\ f_1;M_0,...,M_{15}\}$

$8+4+4+16=32\text{word}$

全部输入为 32 个字，每个字为 32bit。



(a) Hashing 3 blocks: the tree has depth 3. (b) Hashing 5 blocks: the tree has depth 4.



初始化：v0,…,v15，16 个字，每个字 32bit，16*32=512

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ IV_0 & IV_1 & IV_2 & IV_3 \\ t_0 \oplus IV_4 & t_1 \oplus IV_5 & f_0 \oplus IV_6 & f_1 \oplus IV_7 \end{pmatrix}$$

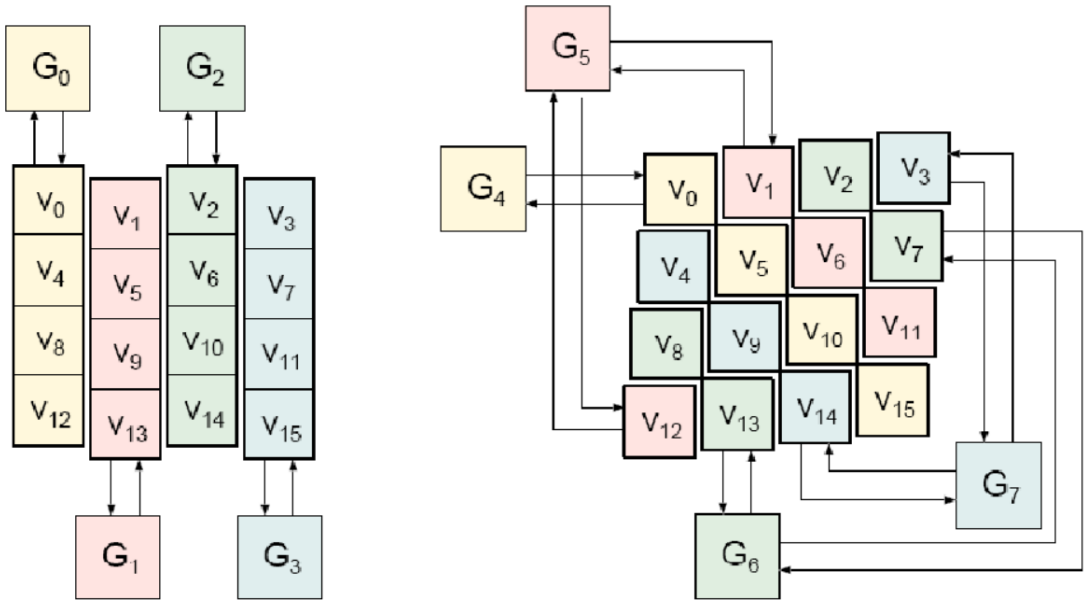
16*32bit=512bit

8 个轮函数 G0,…,G7，执行 10 轮

$$G_0(v_0,v_4,v_8,v_{12}),G_1(v_1,v_5,v_9,v_{13}),G_2(v_2,v_6,v_{10},v_{14}),G_3(v_3,v_7,v_{11},v_{15})$$
$$G_4(v_0,v_5,v_{10},v_{15}),G_5(v_1,v_6,v_{11},v_{12}),G_6(v_2,v_7,v_8,v_{13}),G_7(v_3,v_4,v_9,v_{14})$$

输入(v0~v15)16, 16*4*8=512bit，输出 512bit，所以称为**轮函数**

包括**线性变换、非线性变换、轮常量加。**



维基百科：Gi 函数，i=0,…,7

j ← σ[r%10][2×i] // Index computations
k ← σ[r%10][2×i+1]
a ← a + b + (**m[j]** ⊕ c[k]常量) // Step 1 (with input)
d ← (d ⊕ a) >>> 16
c ← c + d // Step 2 (no input)
b ← (b ⊕ c) >>> 12
a ← a + b + (**m[k]** ⊕ c[j] 常量) // Step 3 (with input)
d ← (d ⊕ a) >>> 8
c ← c + d // Step 4 (no input)
b ← (b ⊕ c) >>> 7

σ_r 位置置换函数， r 当前迭代轮数。

官方文档：Blake2b 和 Blake2s 的轮函数

$$
\begin{aligned}
a &\leftarrow a + b + m_{\sigma_r(2i)} \\
d &\leftarrow (d \oplus a) \ggg 32 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 24 \\
a &\leftarrow a + b + m_{\sigma_r(2i+1)} \\
d &\leftarrow (d \oplus a) \ggg 16 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 63
\end{aligned}
\qquad
\begin{aligned}
a &\leftarrow a + b + m_{\sigma_r(2i)} \\
d &\leftarrow (d \oplus a) \ggg 16 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 12 \\
a &\leftarrow a + b + m_{\sigma_r(2i+1)} \\
d &\leftarrow (d \oplus a) \ggg 8 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 7
\end{aligned}
$$

σ[0] = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
σ[1] = 14 10 4 8 9 15 13 6 1 12 0 2 11 7 5 3

σ[2] = 11 8 12 0 5 2 15 13 10 14 3 6 7 1 9 4
σ[3] = 7 9 3 1 13 12 11 14 2 6 5 10 4 0 15 8
σ[4] = 9 0 5 7 2 4 10 15 14 1 11 12 6 8 3 13
σ[5] = 2 12 6 10 0 11 8 3 4 13 7 5 15 14 1 9
σ[6] = 12 5 1 15 14 13 4 10 0 7 6 3 9 2 8 11
σ[7] = 13 11 7 14 12 1 3 9 5 0 15 4 8 6 2 10
σ[8] = 6 15 14 9 11 3 0 8 12 2 13 7 1 4 10 5
σ[9] = 10 2 8 4 7 6 1 5 15 11 9 14 3 12 13 0

$c_0 = 243F6A8885A308D3$ $c_1 = 13198A2E03707344$
$c_2 = A4093822299F31D0$ $c_3 = 082EFA98EC4E6C89$
$c_4 = 452821E638D01377$ $c_5 = BE5466CF34E90C6C$
$c_6 = C0AC29B7C97C50DD$ $c_7 = 3F84D5B5B5470917$
$c_8 = 9216D5D98979FB1B$ $c_9 = D1310BA698DFB5AC$
$c_{10} = 2FFD72DBD01ADFB7$ $c_{11} = B8E1AFED6A267E96$
$c_{12} = BA7C9045F12C7F99$ $c_{13} = 24A19947B3916CF7$
$c_{14} = 0801F2E2858EFC16$ $c_{15} = 636920D871574E69$

最后，压缩输出结果：
输入 512bit，输出 256bit

$$h_0' = h_0 \oplus s_0 \oplus v_0 \oplus v_8$$
$$h_1' = h_1 \oplus s_1 \oplus v_1 \oplus v_9$$
$$h_2' = h_2 \oplus s_2 \oplus v_2 \oplus v_{10}$$
$$h_3' = h_3 \oplus s_3 \oplus v_3 \oplus v_{11}$$
$$h_4' = h_4 \oplus s_0 \oplus v_4 \oplus v_{12}$$
$$h_5' = h_5 \oplus s_1 \oplus v_5 \oplus v_{13}$$
$$h_6' = h_6 \oplus s_2 \oplus v_6 \oplus v_{14}$$
$$h_7' = h_7 \oplus s_3 \oplus v_7 \oplus v_{15}$$

速度最快！

2.3 SHA3-Keccak

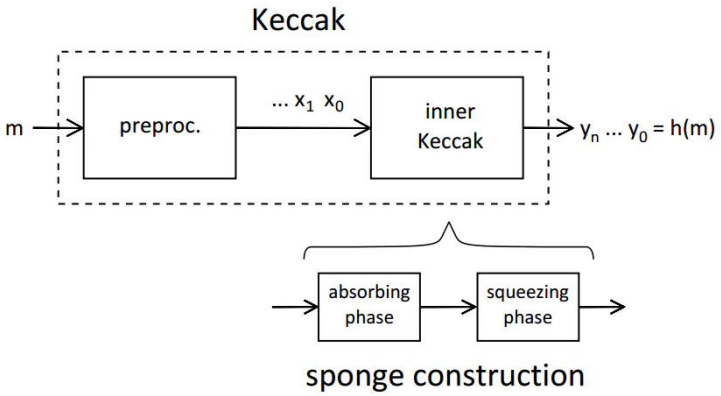
2.3.1.Keccak

采用**海绵结构（sponge construction）**，在预处理（padding 并分成大小相同的块）后，海绵结构主要分成两部分：

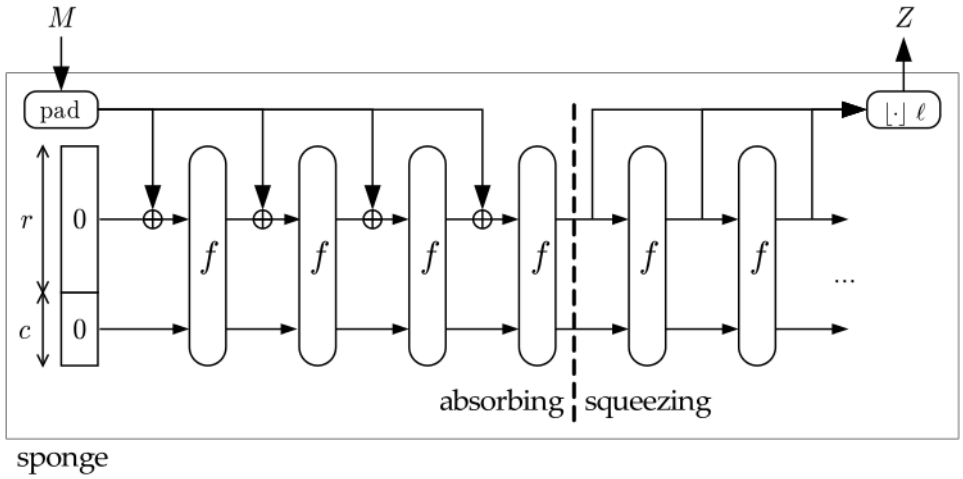
吸入阶段（absorbing phase）：将块 x_i 传入算法并处理。

挤出阶段（squeezing phase）：产生固定长度的输出。

Keccak 算法的整体结构如图：



在这两个阶段要是使用同一个**轮函数 Keccak-f**，下图展示了算法“吸入”一个块 x_i 并处理，最后挤出输出的过程：



M=r*n

r = 1088
c = 512
轮函数 f
吸入阶段：输入 1600bit，输出 1600bit，取 1088bit
挤出阶段：输入 1600bit，输出 r-bit 作为哈希值, r=256

2.3.1.1 吸入和挤出阶段

从图中我们可以归纳出大致的流程：

1. **数据填充：**对输入串 x 做 padding，使其长度能被 r=1088 整除，将 padding 后分割成长度为 r=1088bits 的块，即 $x=x_0||x_1||x_2||...||x_{t-1}$。
2. **初始化：**一个长度为 b=r+c bit 的全零向量。b=1088+512=1600bits
3. **输入：**块 x_i，将 x_i 和向量的前 r 个**异或运算**，然后输入到 Keccak-f 函数中处理。重复上一步，直至处理完 x 中的每个块。

4. **输出：长为 r 的块作为 y_0**，将向量输入到 Keccak-f 函数中处理，输出 y_1，以此类推。得到的 Hash 序列即为 y=y_0||y_1||y_2||...||y_u。在 Keccak-224/**256**/384/512 中，**只需要在 y_0 中取出对应长度的前缀即可。**

针对图中的参数，做出如下定义：

- r：**比特率（bit rate）**，其值为每个输入块的长度。
- c：**容量（capacity）**，其长度为输出长度的两倍。
- b：向量的长度，**b=r+c**。b 的值依赖于指数 L，即 b=25*(2^L),L=6，b=1600。
- **b = 1600; r = 1088 ; c = 512;**

在 Keccak-224/256/384/512 中，b、r、c 及输出长度的取值见下表。

b/比特	*r*/比特	*c*/比特	输出长度/比特	安全级别/比特
1600	1152	448	224	112
1600	1088	512	256	128
1600	832	768	384	192
1600	576	1024	512	256

Keccak算法的参数定义

2.3.1.2 填充

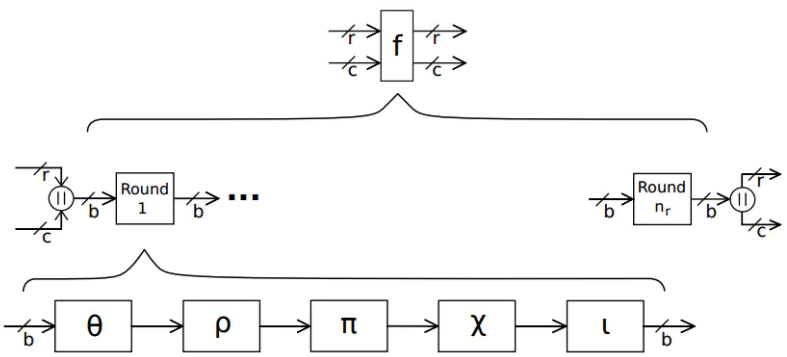
填充规则：m'={m,1000001}

要求：(Len(msg)+2+k)/r=0，寻找最小自然数 k。

2.3.2.轮函数

对称加密中的轮函数，对应此处的压缩函数。

压缩函数以 b=r+c 比特作为输入，b=r+c 比特作为输出。内部结构如下：



- **线性变换**
- **非线性变换**

- **轮密钥加/轮常量加**

Keccak-f 包含 n_r 轮。n_r 的取值与之前计算 b 时用到的指数 L 成线性关系

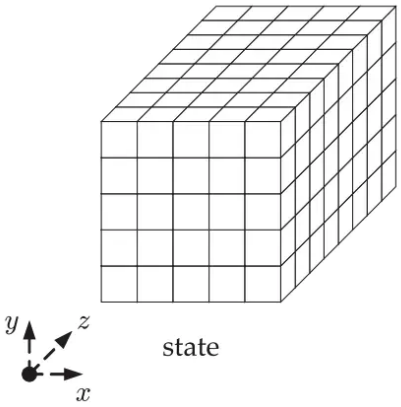
$$L = \log_2 W = \log_2 64 = 6$$
$$n_r = 12 + 2L = 12 + 2*6 = 24$$

Keccak-224/**256**/384/512 中，**W=64,则 L=6，因此 n_r=24 rounds**。在每一轮中，要以此执行五步，即 **θ(theta)、ρ(rho)、π(pi)、χ(chi)、ι(iota)**。在处理过程中，**把 b=1600 比特数据排列成一个 5*5*64 的立体**

如图：(x,y,z)=(5,5,64)。5*5*64=1600bits

slice(x,y)平面平移；plane(x,z)平面平移；sheet(y,z)平面平移；

row x 轴平移；column y 轴平移；lane z 轴平移；



- AES 加密将数据排为**平面矩阵**，轮函数包含 4 种操作：字节替代（S 盒子）、行移位、列混淆和轮密钥加。
- SHA3-Keccak 将数据排为**立体**，轮函数包括 5 个子函数：θ(theta)、ρ(rho)、π(pi)、χ(chi)、ι(iota)。

任意一个比特记为：$a[x][y][z]$，其中 x,y,z 的**模系数**分别为 5,5,64

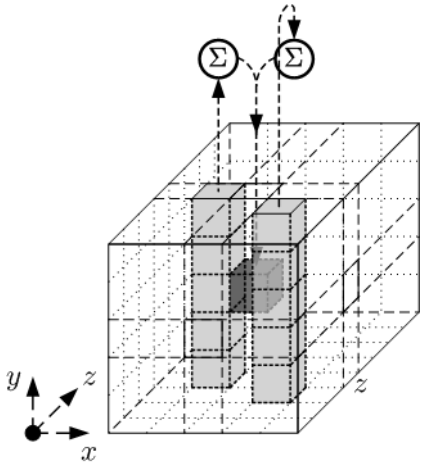
二进制异或运算就是二进制加法运算

Theta (θ) 函数

列与周围 2 列元素相加

$$\theta: a'[x][y][z] := a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$$

$$\theta: a'[3][3][3] := a[3][3][3] + \sum_{y'=0}^{4} a[2][y'][3] + \sum_{y'=0}^{4} a[4][y'][2]$$

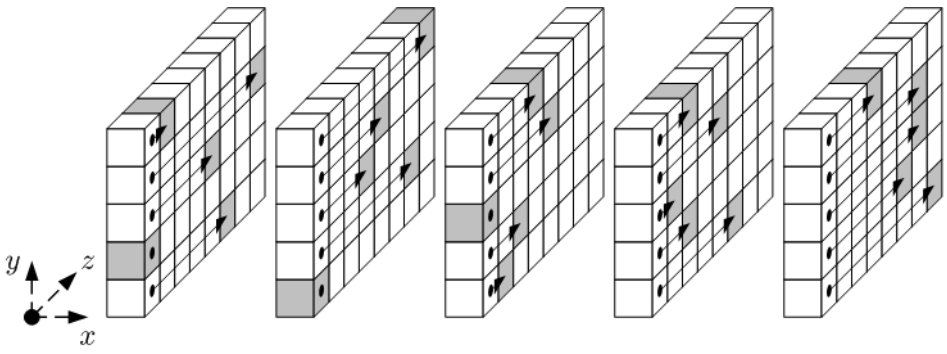


Rho (ρ) and Pi (π)函数：在深度方向循环移位

$$\rho: a'[x][y][z] := a[x][y][z-(t+1)(t+2)/2], t \in [0,...,24], (x,y) = (1,0)\begin{pmatrix} 0 & 2 \\ 1 & 3 \end{pmatrix}^t$$

$$t = 1, (x,y) = (0,2)$$

$$\rho: a'[0][2][z] := a[0][2][z-3]$$

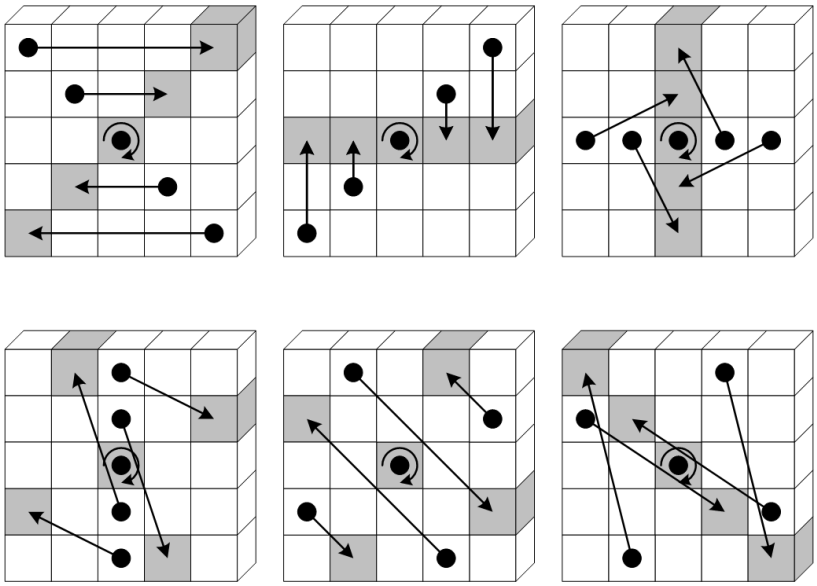


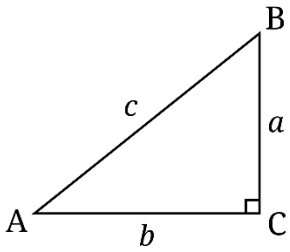
需要常量如下

```
const int R_CONS[MAT][MAT] = {
    {0, 36, 3, 41, 18},
    {1, 44, 10, 45, 2},
    {62, 6, 43, 15, 61},
    {28, 55, 25, 21, 56},
    {27, 20, 39, 8, 14}};
}
```

在平面层面旋转

$$\pi : a[x'][y'] := a[x][y], \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} x \\ y \end{pmatrix}, t = 0, \ldots, 23$$





直角三角形：从 A 点到 C 点，再到 B 点，等价于直接从 A 点到 B 点。

二者结合，等价表达为

- **B[y,2*x+3*y] = rot(A[x,y], r[x,y]), for (x,y) in (0…4,0…4)**

该步的输入为 **A 数组**，输出为 **B 数组**。

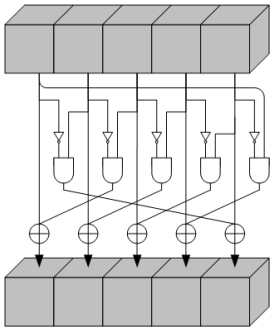
rot 含义同上一步，其中作为 **offset 的 r 数组**定义如下：

	x=3	**x=4**	**x=0**	**x=1**	**x=2**
y=2	25	39	3	10	43
y=1	55	20	36	44	6
y=0	28	27	0	1	62
y=4	56	18	14	2	61
y=3	21	8	41	45	15

Chi 函数（读开） (χ) 唯一的非线性变换

某一行数据与后面两行数据累加

For x=0,…,4{
    For y=0,…,4{
        a'[x][y]= a[x][y]⊕(NOT a[x+1][y]) AND a[x+2][y]
}}



iota (ι) 函数：a'[0][0]= a[0][0]⊕RC, z=0,…,23

异或轮常数，消除对称性

RC 值与轮数有关，RC 在 24 轮中的定义如下：

i_r	RC	i_r	RC	i_r	RC
0	0000000000000001	8	000000000000008A	16	8000000000008002
1	0000000000008082	9	0000000000000088	17	8000000000000080
2	800000000000808A	10	0000000080008009	18	000000000000800A
3	8000000080008000	11	000000008000000A	19	800000008000000A
4	000000000000808B	12	000000008000808B	20	8000000080008081
5	0000000080000001	13	800000000000008B	21	8000000000008080
6	8000000080008081	14	8000000000008089	22	0000000080000001
7	8000000000008009	15	8000000000008003	23	8000000080008008

2.3.3 输出

通过海绵结构中的挤出阶段，可以获得任意长度的输出。在 Keccak-224/256/384/512 中，我们只需要获得 y0 中的前 224/**256**/384/512 个 bit 作为输出即可。

输出长度为 1088 bits 的 y0，取前 256bit 就是哈希值。

zk unfrendly: SHA2/BLAKE/Keccak

zk frendly: Poseidon/Rescue

电路： $R1CS: a \cdot b = c$

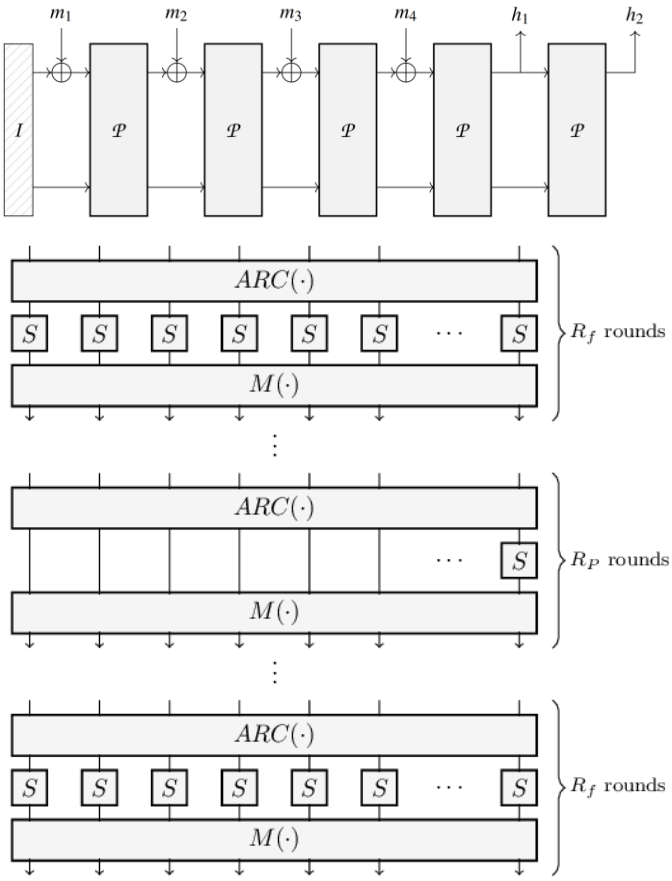
$PlonkGate: a \cdot b + a + b + c + constant = 0$

2.4 Poseidon 哈希函数

初始向量： $I = 0^r \,||\, 0^c$，r-bit 的零和 c-bit 的零。

将数据分为多个块，例如分为 4 块，$m = \{m_1 \,||\, m_2 \,||\, m_3 \,||\, m_4\}$

输出 $h = \{h_1 \,||\, h_2\}$



Poseidon 的轮函数包括：

- 轮密钥/常量加（添加常量）
- 非线性变换（S 盒子）$\text{S-box}(x) = x^5$
- 线性变换（MDS 矩阵/满秩）$\vec{m}' := A \cdot \vec{m}$

Poseidon 哈希函数在 64bit 域上进行加法运算、乘法运算，电路门数量较低。

对称加密与哈希函数：线性变换、非线性变换、轮密钥加/轮常量加。

lynndell 新火科技 密码学专家 lynndell2010@gmail.com