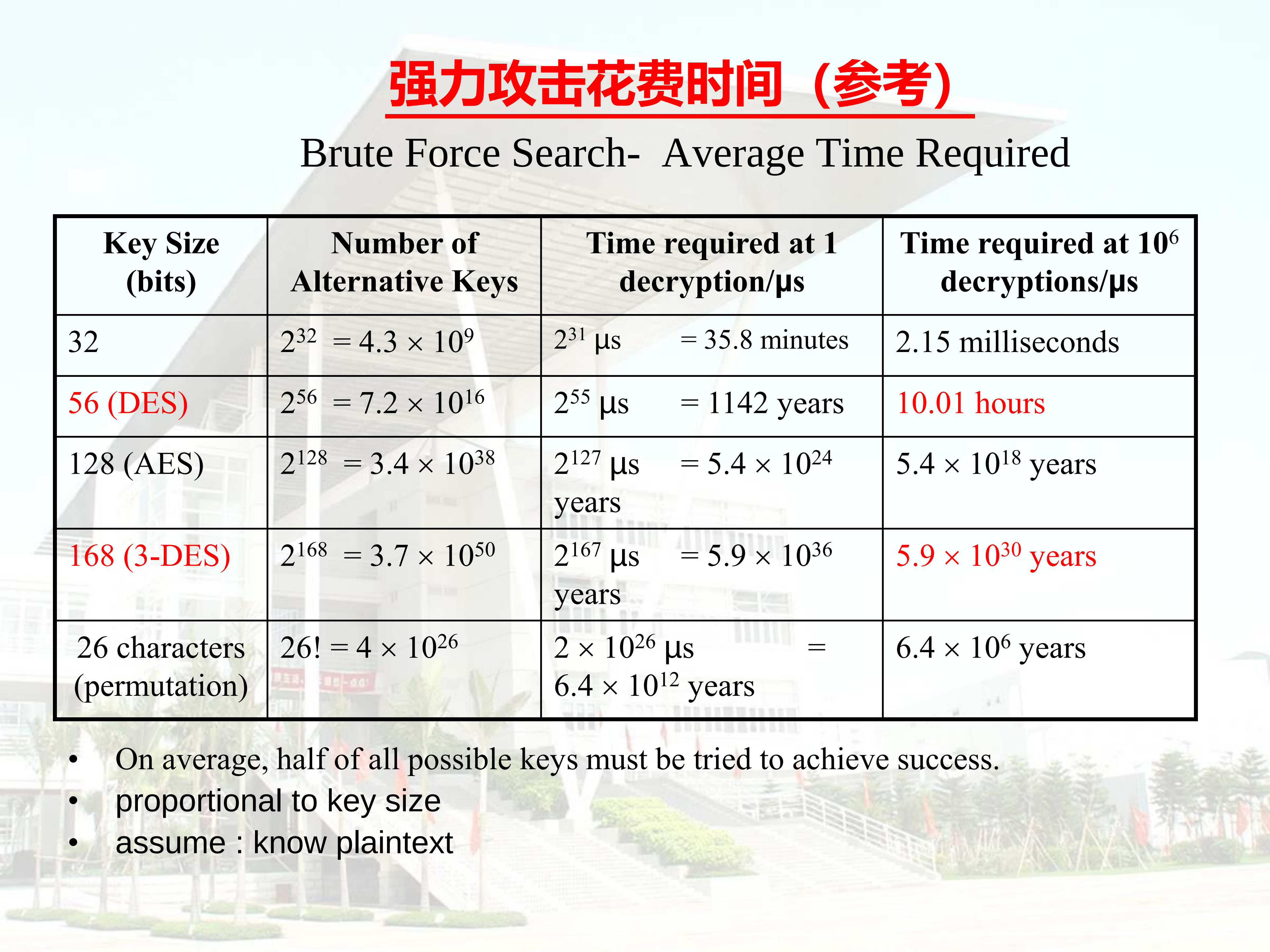


强力攻击花费时间（参考）

Brute Force Search- Average Time Required

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/µs	Time required at 10^6 decryptions/µs
32	$2^{32} = 4.3 \times 10^9$	2^{31} µs = 35.8 minutes	2.15 milliseconds
56 (DES)	$2^{56} = 7.2 \times 10^{16}$	2^{55} µs = 1142 years	10.01 hours
128 (AES)	$2^{128} = 3.4 \times 10^{38}$	2^{127} µs = 5.4×10^{24} years	5.4×10^{18} years
168 (3-DES)	$2^{168} = 3.7 \times 10^{50}$	2^{167} µs = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} µs = 6.4×10^{12} years	6.4×10^6 years

- On average, half of all possible keys must be tried to achieve success.
- proportional to key size
- assume : know plaintext

Advanced Encryption Standard （AES)
高级加密标准

- 1997美国国家标准技术局(NIST）发起征集AES的活动

⇩

National Institute of Standards and Technology

原因及目的：取代 DES

1） DES -》 3DES

2） 3DES软件实现时，速度慢

Block size 为 64 bits -》效率低

3) 安全考虑

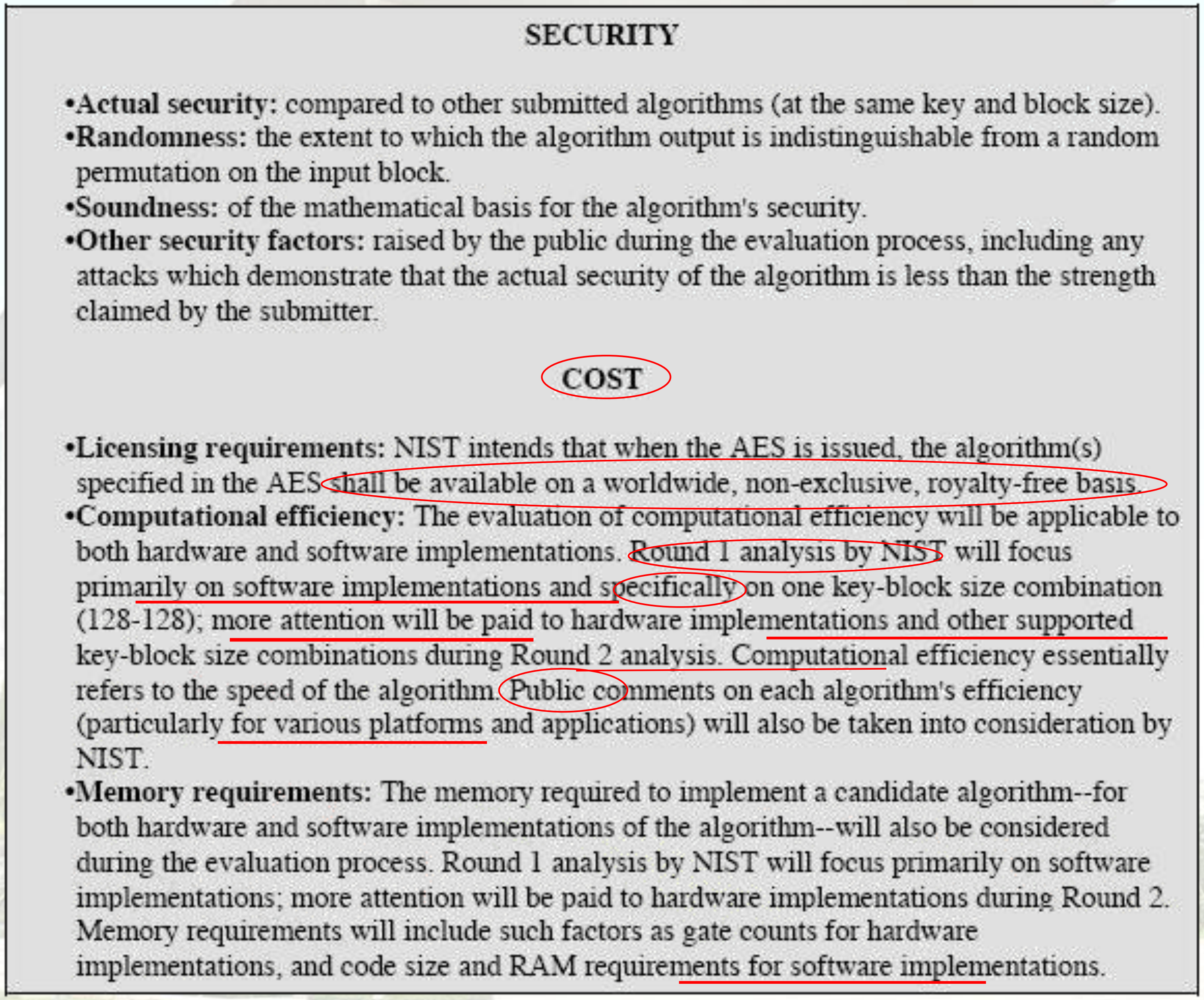
Advanced Encryption Standard （AES)
高级加密标准

- 1998年第一次AES候选大会，公布了15个候选算法
- 1999年第二次AES候选大会，选出5个候选算法
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- 2000年第三次AES候选大会。
- 2000年公布Rijndael算法作为候选算法。（fips-197）

比利时的Joan Daemen, Vincent Rijmen 设计：

Rijndael 算法 - 为AES算法

Table 5.1 NIST Evaluation Criteria for AES (September 12, 1997) (page 1 of 2)



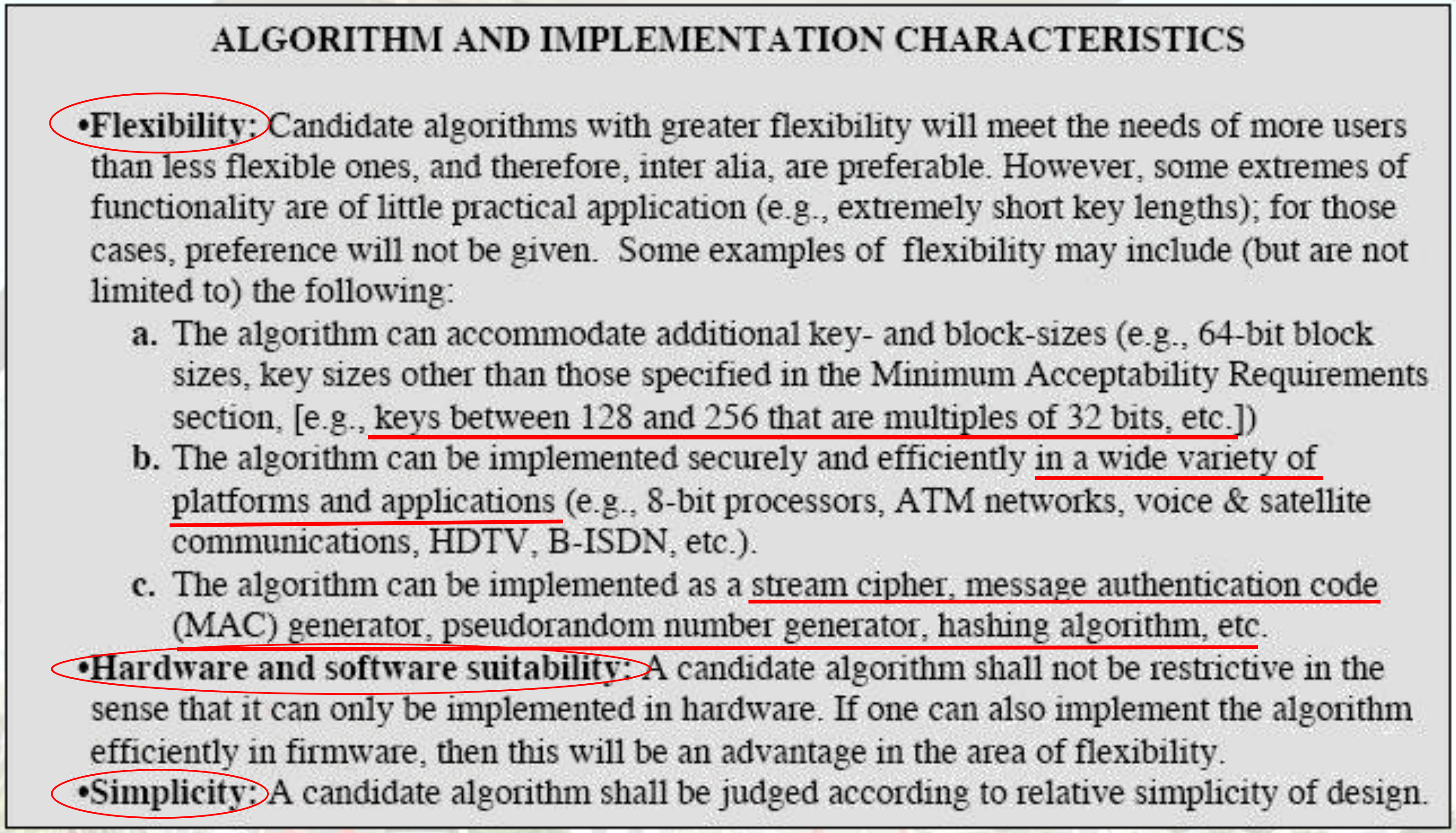
SECURITY

•**Actual security:** compared to other submitted algorithms (at the same key and block size).
•**Randomness:** the extent to which the algorithm output is indistinguishable from a random permutation on the input block.
•**Soundness:** of the mathematical basis for the algorithm's security.
•**Other security factors:** raised by the public during the evaluation process, including any attacks which demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.

COST

•**Licensing requirements:** NIST intends that when the AES is issued, the algorithm(s) specified in the AES shall be available on a worldwide, non-exclusive, royalty-free basis.
•**Computational efficiency:** The evaluation of computational efficiency will be applicable to both hardware and software implementations. Round 1 analysis by NIST will focus primarily on software implementations and specifically on one key-block size combination (128-128); more attention will be paid to hardware implementations and other supported key-block size combinations during Round 2 analysis. Computational efficiency essentially refers to the speed of the algorithm. Public comments on each algorithm's efficiency (particularly for various platforms and applications) will also be taken into consideration by NIST.
•**Memory requirements:** The memory required to implement a candidate algorithm--for both hardware and software implementations of the algorithm--will also be considered during the evaluation process. Round 1 analysis by NIST will focus primarily on software implementations; more attention will be paid to hardware implementations during Round 2. Memory requirements will include such factors as gate counts for hardware implementations, and code size and RAM requirements for software implementations.

Table 5.1 NIST Evaluation Criteria for AES (September 12, 1997) (page 2 of 2)



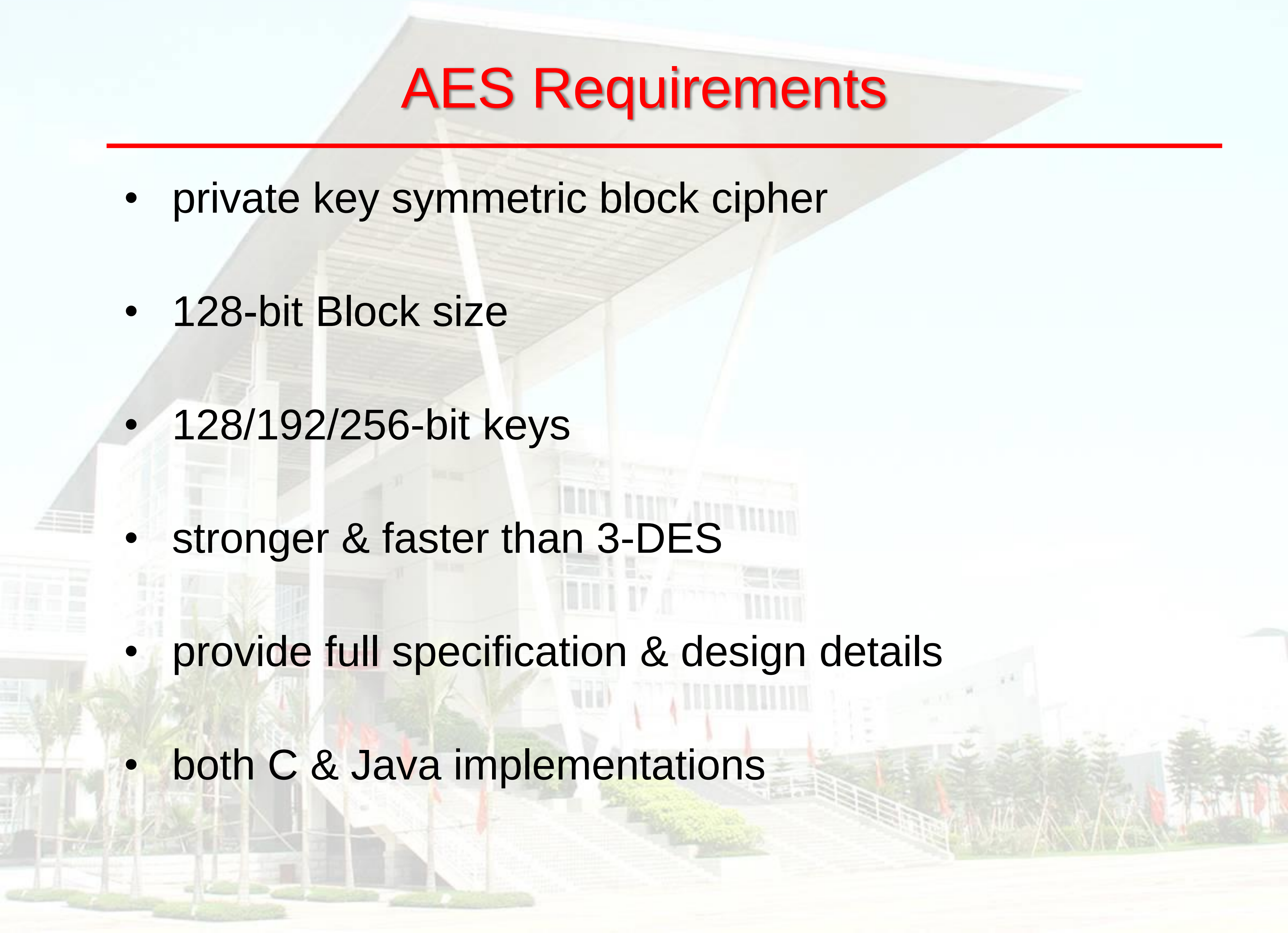
ALGORITHM AND IMPLEMENTATION CHARACTERISTICS

•**Flexibility:** Candidate algorithms with greater flexibility will meet the needs of more users than less flexible ones, and therefore, inter alia, are preferable. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preference will not be given. Some examples of flexibility may include (but are not limited to) the following:

- **a.** The algorithm can accommodate additional key- and block-sizes (e.g., 64-bit block sizes, key sizes other than those specified in the Minimum Acceptability Requirements section, [e.g., keys between 128 and 256 that are multiples of 32 bits, etc.])
- **b.** The algorithm can be implemented securely and efficiently in a wide variety of platforms and applications (e.g., 8-bit processors, ATM networks, voice & satellite communications, HDTV, B-ISDN, etc.).
- **c.** The algorithm can be implemented as a stream cipher, message authentication code (MAC) generator, pseudorandom number generator, hashing algorithm, etc.

•**Hardware and software suitability:** A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility.

•**Simplicity:** A candidate algorithm shall be judged according to relative simplicity of design.



AES Requirements

- private key symmetric block cipher
- 128-bit Block size
- 128/192/256-bit keys
- stronger & faster than 3-DES
- provide full specification & design details
- both C & Java implementations

AES Cipher

64 X 2/3/4

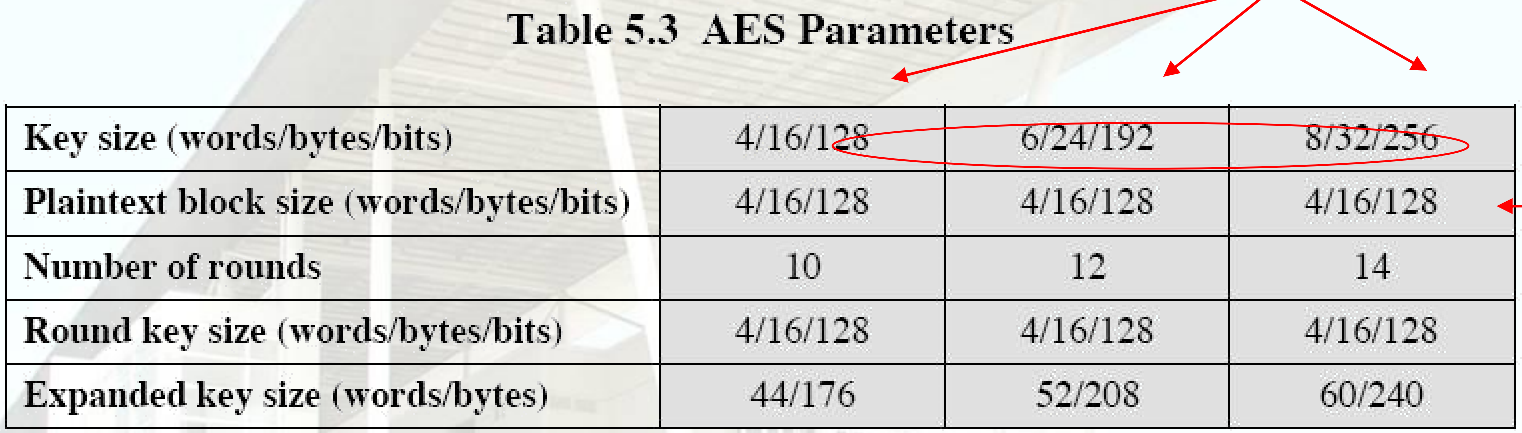


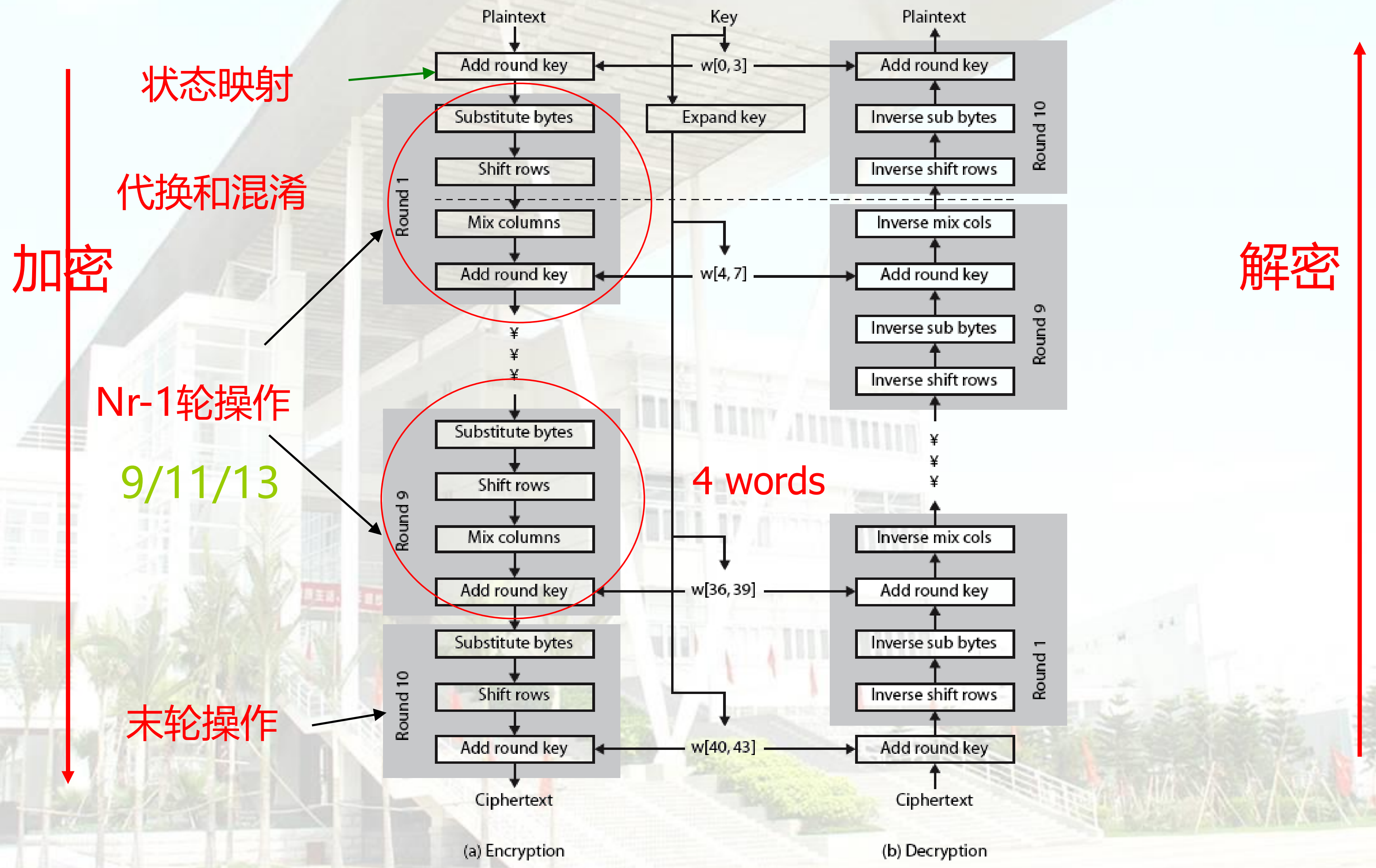
Table 5.3 AES Parameters

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

128固定

- an **iterative** (迭代) rather than **feistel structure**
  - operates on entire (整个) data block in every round
- designed to be:
  - resistant against known attacks （抗攻击）
  - speed and code compactness （紧凑） on many Platforms
  - design simplicity （简单）

Algorithm Structure

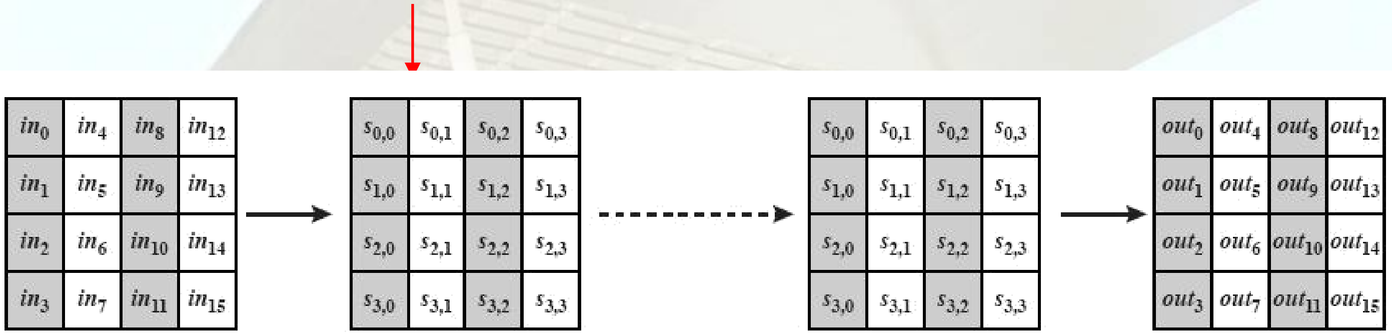


(a) Encryption

(b) Decryption

AES算法描述（一）

- x→state　　16字节明文变成4 * 4矩阵



in_0 in_4 in_8 in_{12}
in_1 in_5 in_9 in_{13}
in_2 in_6 in_{10} in_{14}
in_3 in_7 in_{11} in_{15}
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
out_0 out_4 out_8 out_{12}
out_1 out_5 out_9 out_{13}
out_2 out_6 out_{10} out_{14}
out_3 out_7 out_{11} out_{15}

(a) Input, state array, and output

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

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

AES Key Expansion（扩展）

key is expanded to array of words：

将 128-bit (16-byte，4-word) 密钥扩展为 44-word

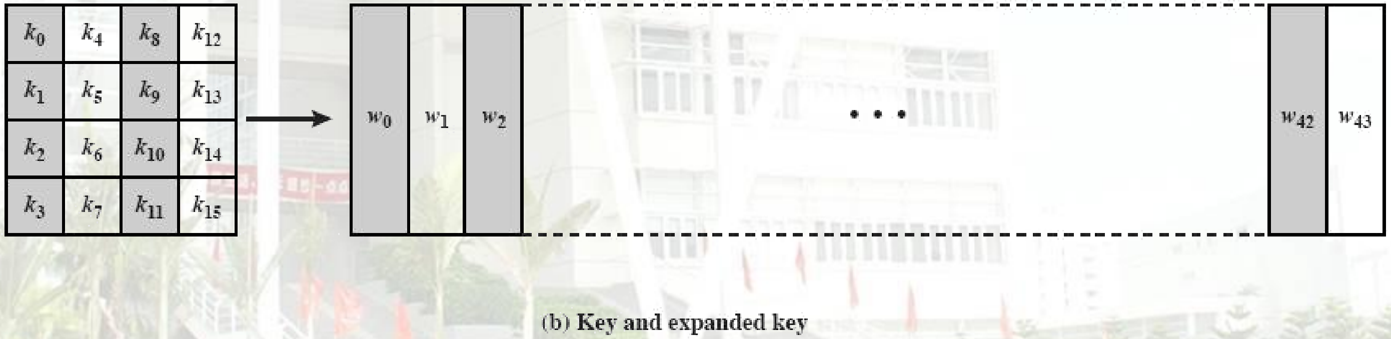

k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 • • • w_{42} w_{43}
(b) Key and expanded key

Figure 5.2 AES Data Structures

AES Key Expansion（扩展）

128bits 16bytes 4words

扩展至↓

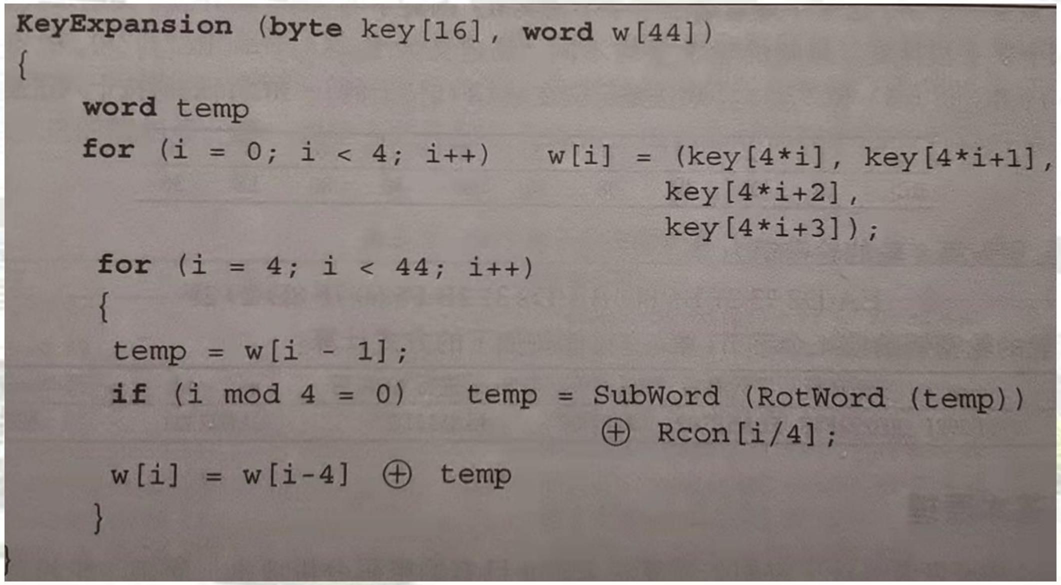
subkey in w[0…43]

初始AddRoundKey w[0,1,2,3]

9轮循环

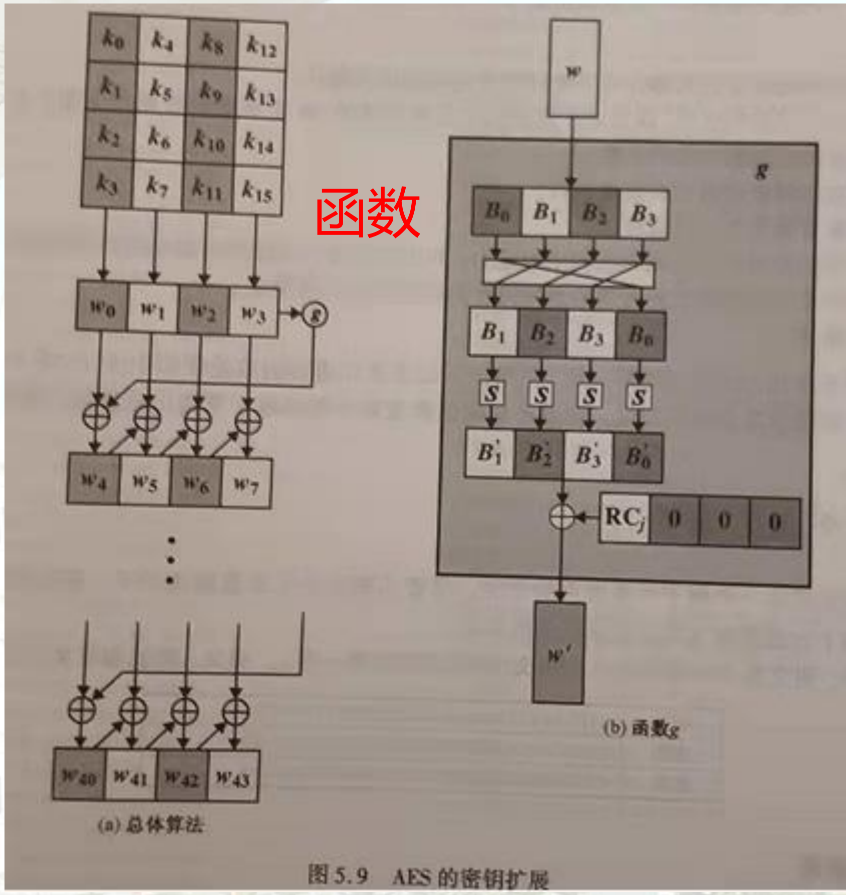
w[4,5,6,7]…w[36,37,38,39]

最后1轮 w[40,41,42,43]



```
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
     }
}
```

函数



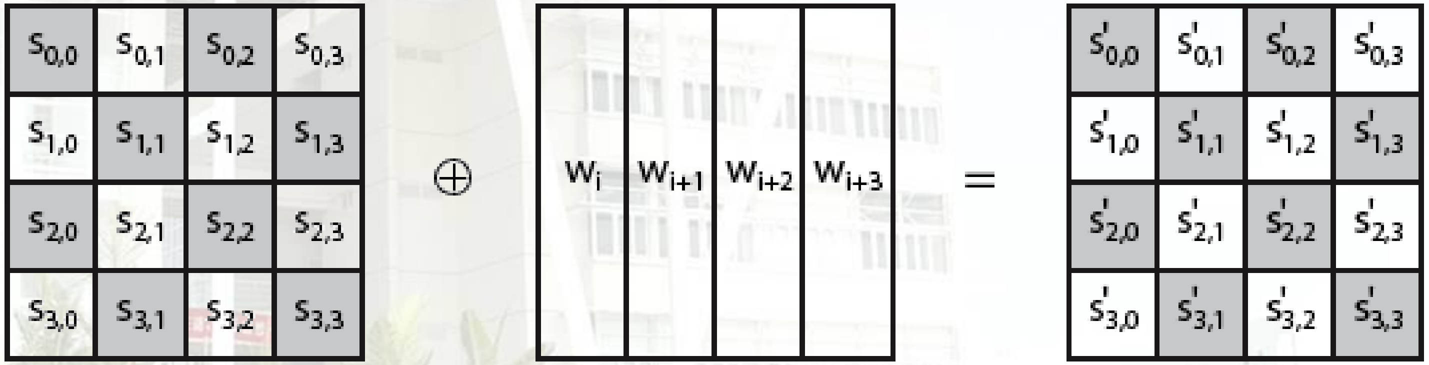
k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3 g
w_4 w_5 w_6 w_7
w_{40} w_{41} w_{42} w_{43}
(a) 总体算法
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B_1' B_2' B_3' B_0'
RC_j 0 0 0
w'
(b) 函数g

图 5.9 AES 的密钥扩展

Add Round Key － state矩阵与轮密钥异或

轮密钥异或，即state XOR RoundKey （AddRoundKey）

XOR state with 128-bits of the round key



$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline & & & \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ & & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ \hline s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ \hline s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ \hline s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \\ \hline \end{array}$$

AES算法描述（二）

- 对前N_r-1轮操作

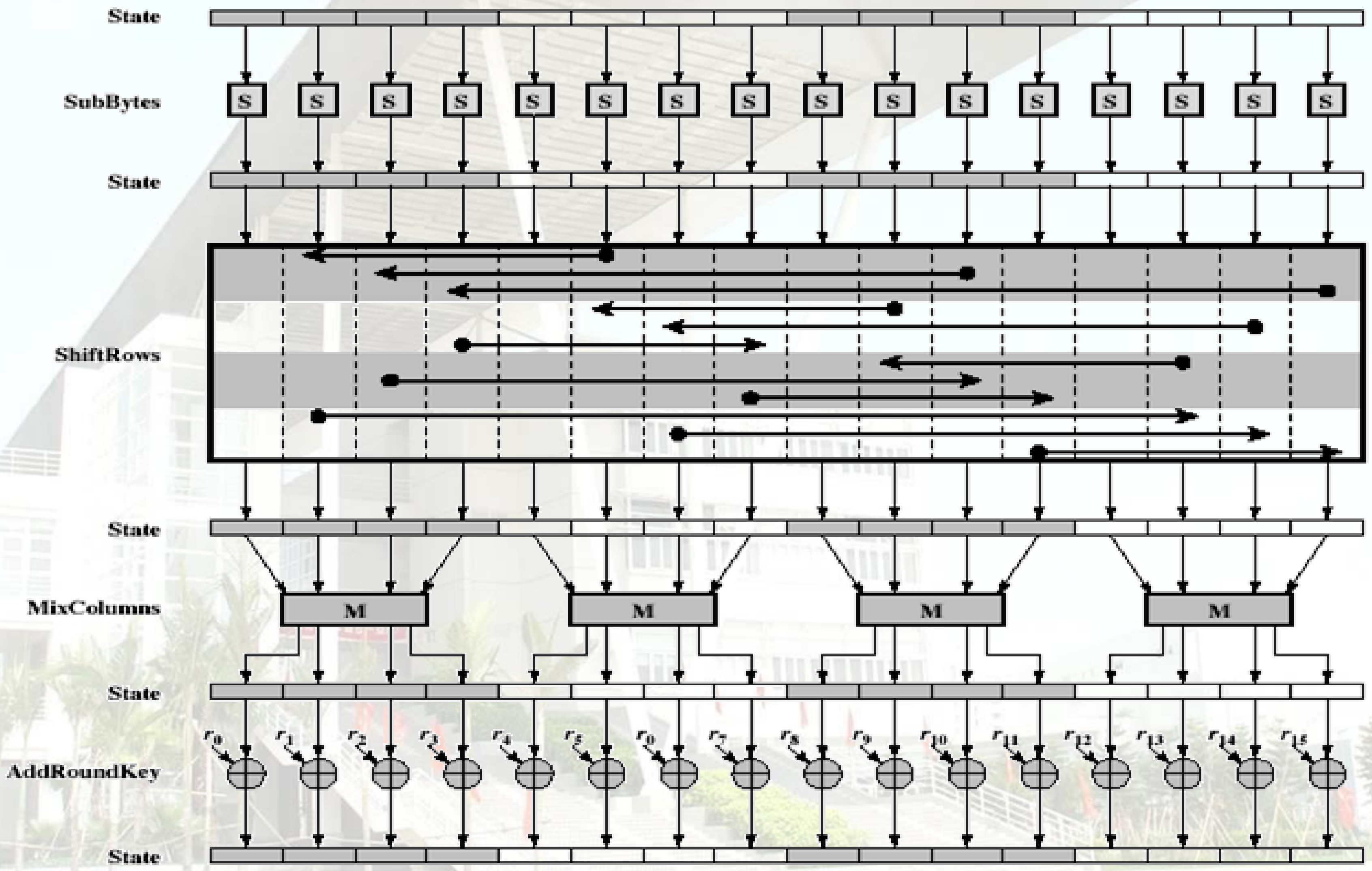
S: SubBytes state矩阵的每元素字节进行S盒代换
P: ShiftRows state矩阵每行逐次循环左移
MixColumns state矩阵每列进行置换
AddRoundKey state矩阵与轮密钥异或

最后1轮

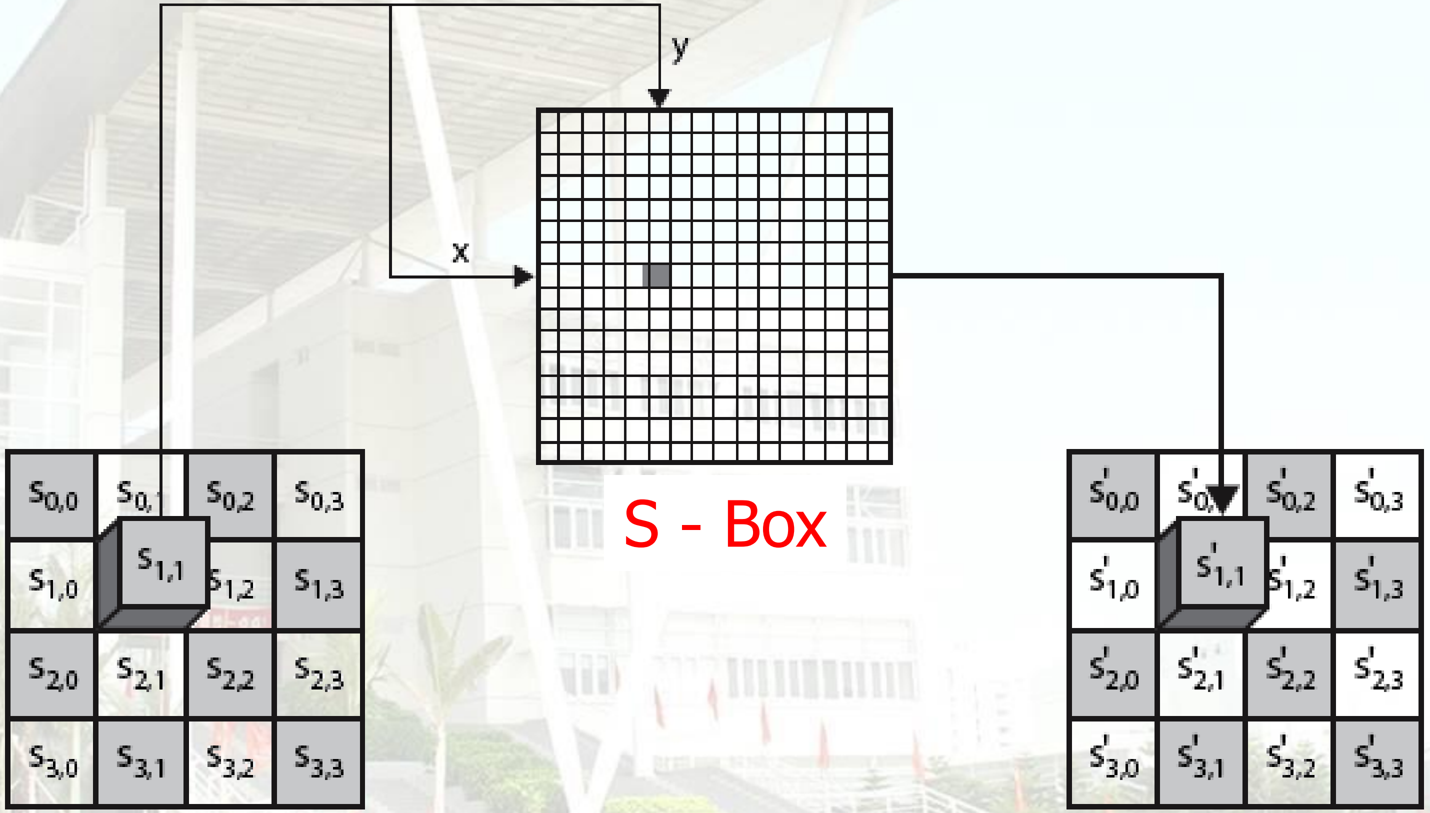
S: SubBytes
P: ShiftRows
AddRoundKey

state→y

AES 一轮处理过程


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

Byte Substitution – state矩阵的每元素字节进行S盒代换


y
x
S - Box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

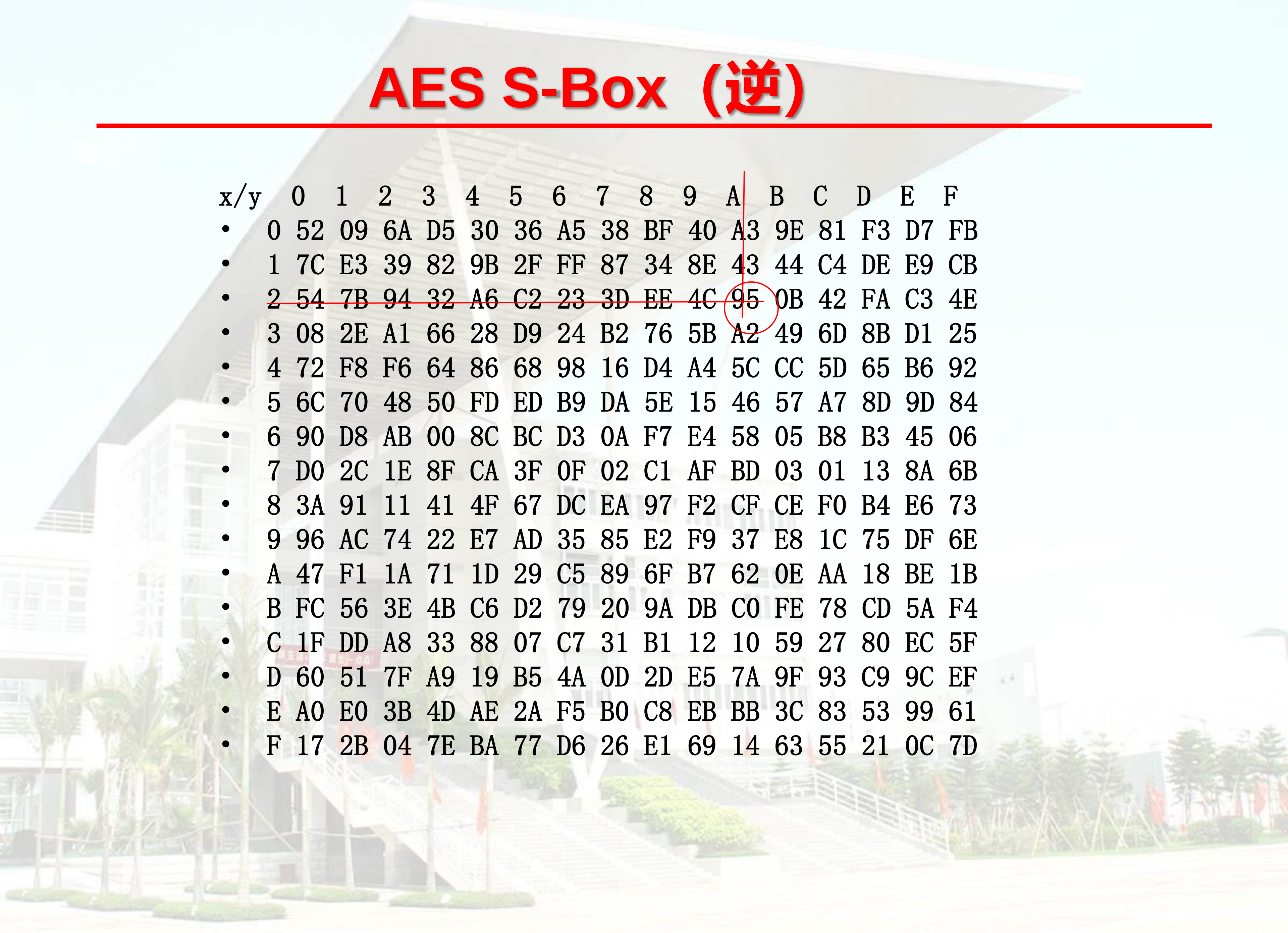
S-Box

each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)

eg. byte {95} is replaced by byte in row 9 column 5 —》 {2A}

x/y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

S-box constructed using defined transformation of values in GF(28)



AES S-Box（逆）

x/y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

Shift Rows（行移位） state矩阵每行逐次循环左移

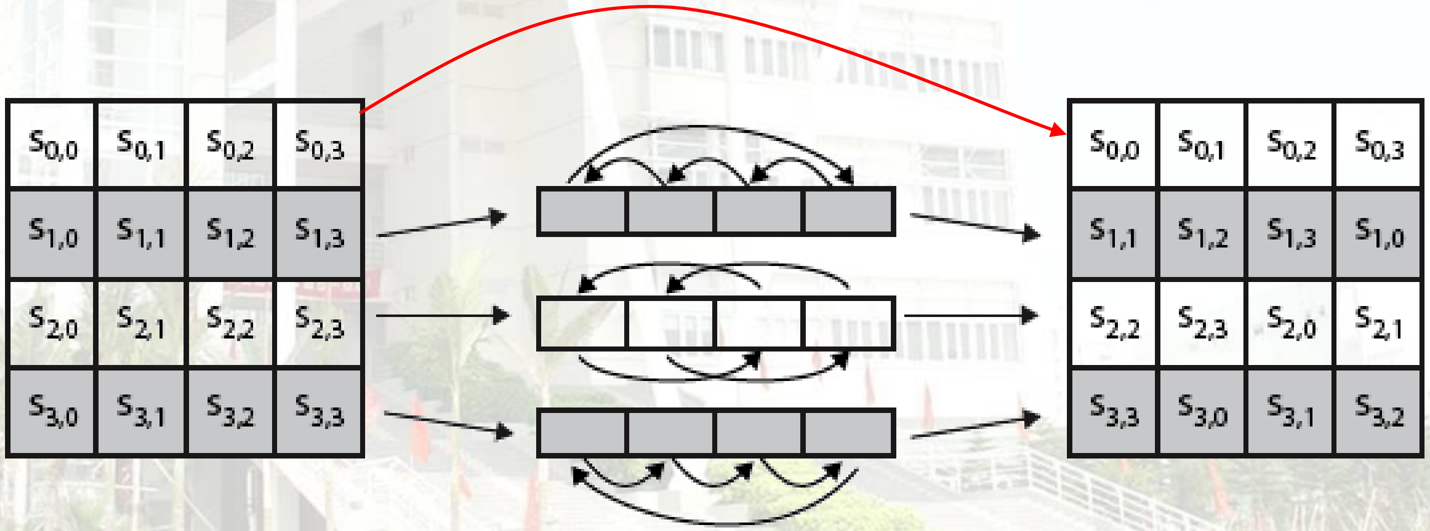
1） 1st row is unchanged （首行不变）

2） 2nd row does 1 byte circular shift to left （循环左移 1 byte）

3） 3rd row does 2 byte circular shift to left （循环左移 2- byte）

4) 4th row does 3 byte circular shift to left （循环左移 3- byte）

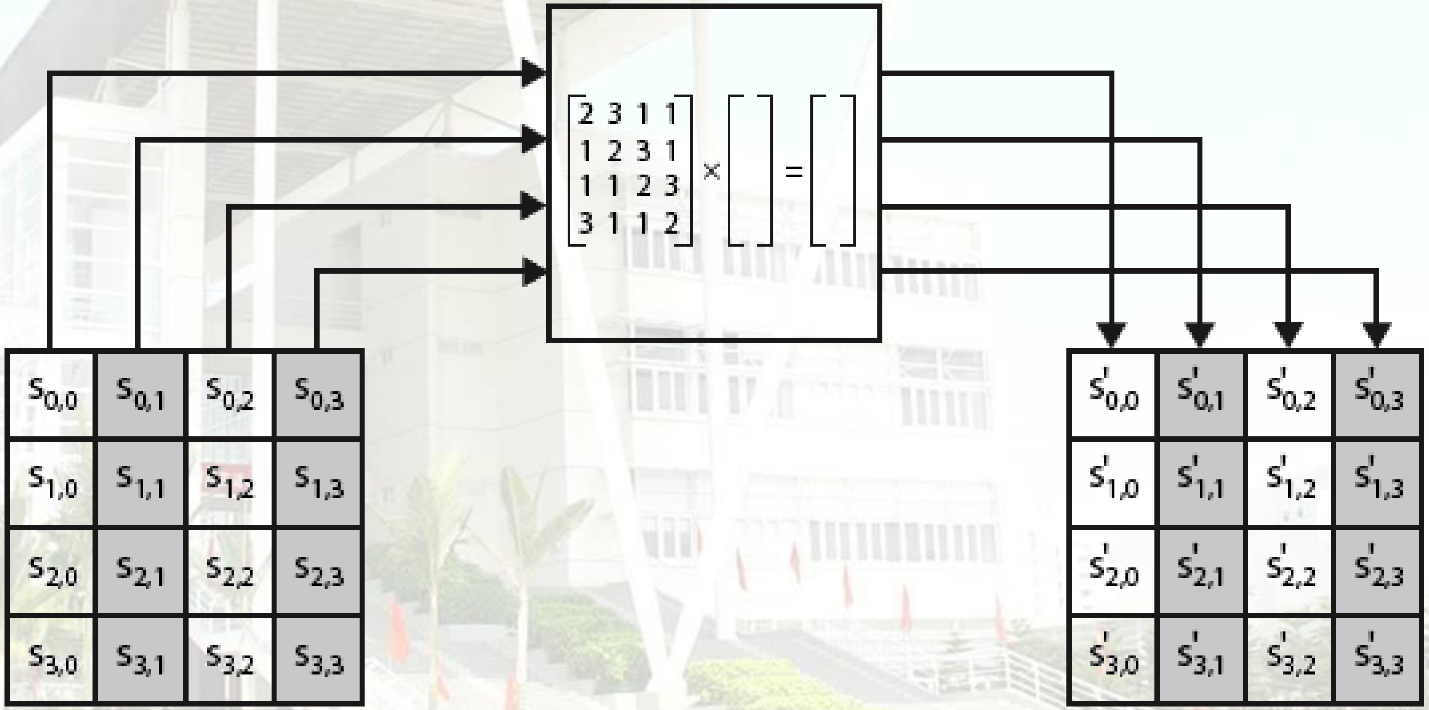
decrypt inverts using shifts to right



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

Mix Columns (列混淆） state矩阵每列进行置换

- each column is processed separately （逐列单独处理）
- each byte is replaced by a value dependent on all 4 bytes in the column



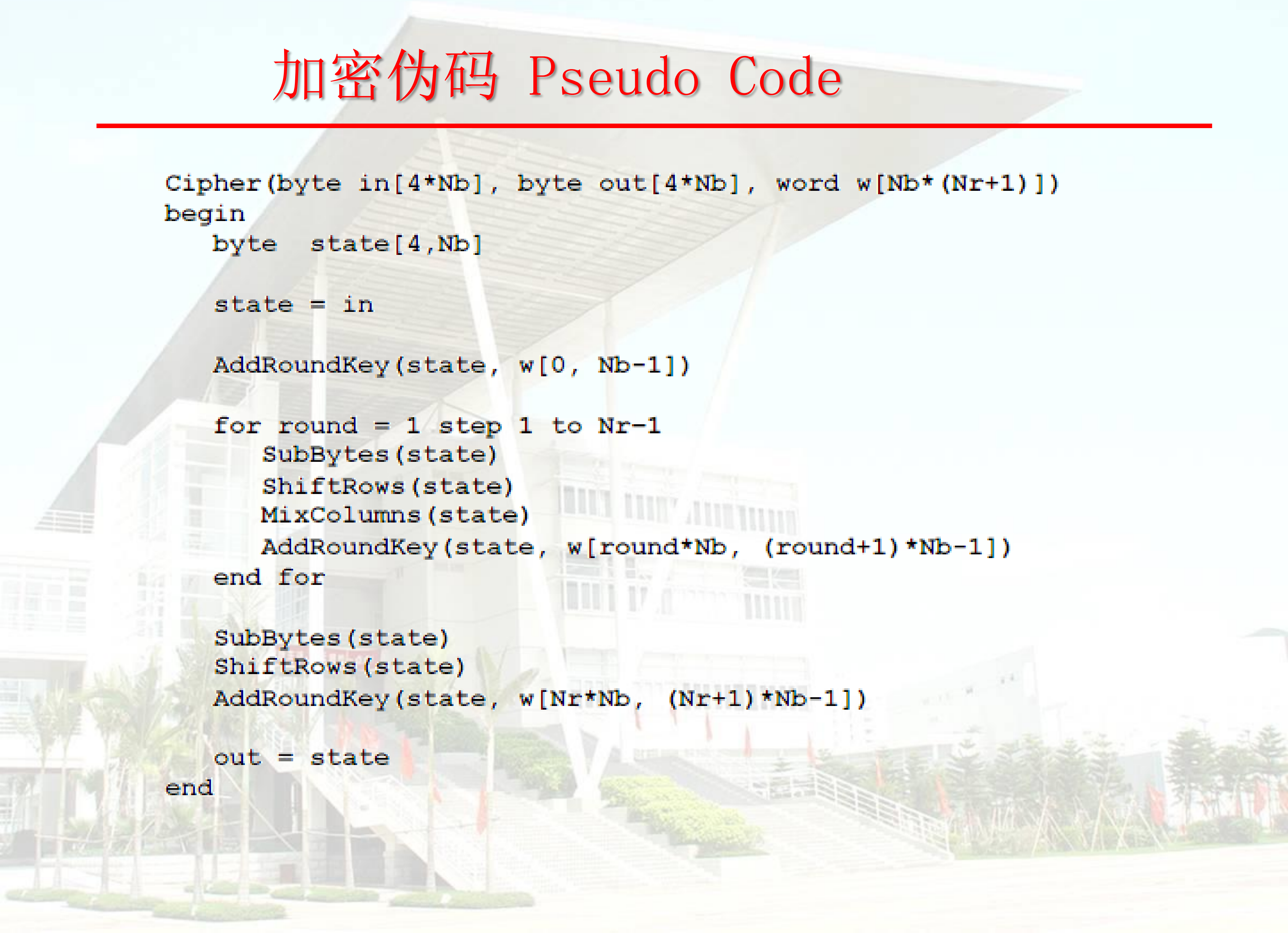
$\begin{bmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{bmatrix} \times \begin{bmatrix}\ \end{bmatrix} = \begin{bmatrix}\ \end{bmatrix}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Mix Columns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

加密伪码 Pseudo Code

```
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])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      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
```



密钥扩展伪码 KeyExpansion

```
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
```

AES 解密伪码 Pseudo Code

```
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
```

Key Exp

```
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])

end for
```

有限域　(Finite Gield)

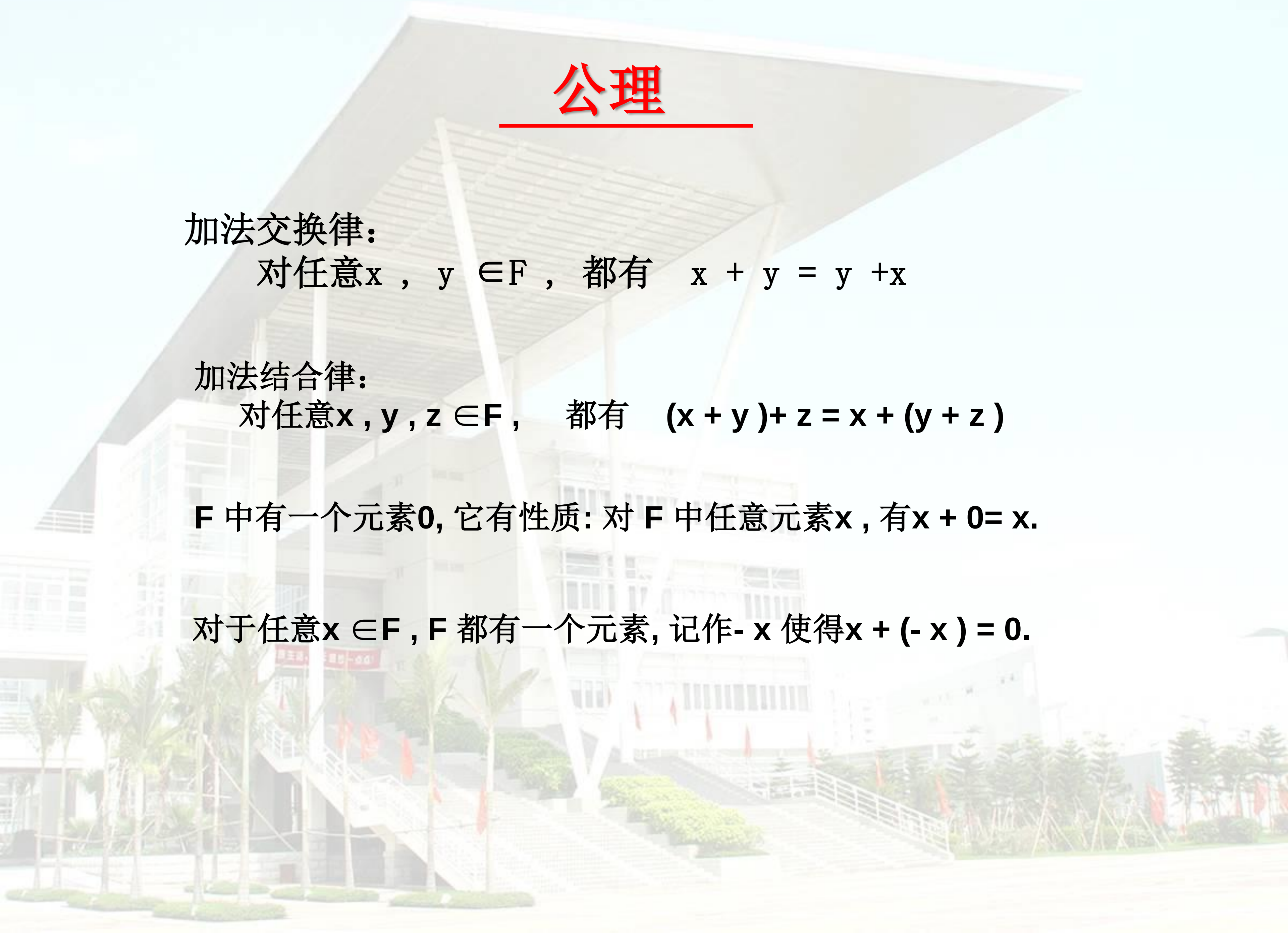
Galois field(伽罗华域)

1832 年, Galois 提出

(1811-1832年）
法国数学家

全体有理数的集合是一个域, 称为有理数域;
全体实数的集合也是一个域称为实数域,
 但这些域都是无限域, 即含无限多个元素的域.

有限域是只含有限个元素的域, 最简单的有限域就是只含两个元素0和1 的域



公理

加法交换律：

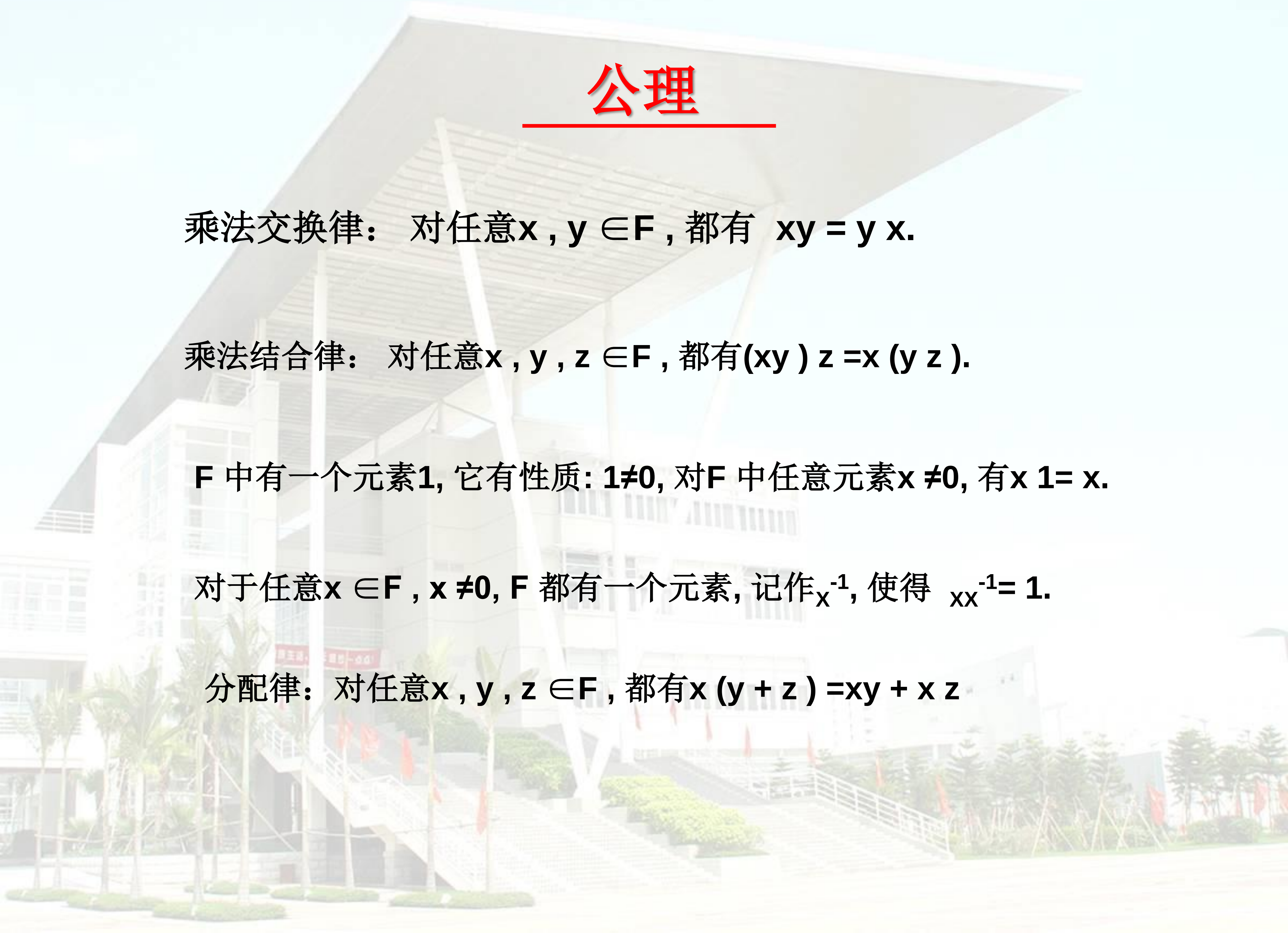
对任意x，$y \in F$，都有 $x + y = y + x$

加法结合律：

对任意x, y, $z \in F$，都有 $(x + y) + z = x + (y + z)$

F 中有一个元素0, 它有性质: 对 F 中任意元素x , 有$x + 0 = x$.

对于任意$x \in F$, F 都有一个元素, 记作$-x$ 使得$x + (-x) = 0$.



公理

乘法交换律： 对任意$x , y \in F$, 都有 $xy = y x$.

乘法结合律： 对任意$x , y , z \in F$, 都有$(xy) z =x (y z)$.

F 中有一个元素1, 它有性质: $1\neq 0$, 对F 中任意元素$x \neq 0$, 有$x 1= x$.

对于任意$x \in F$, $x \neq 0$, F 都有一个元素, 记作x^{-1}, 使得 $xx^{-1}= 1$.

分配律：对任意$x , y , z \in F$, 都有$x (y + z) =xy + x z$