Symmetric Cipher

PREPARED BY: DR. REEMA PATEL

Types of symmetric key ciphers

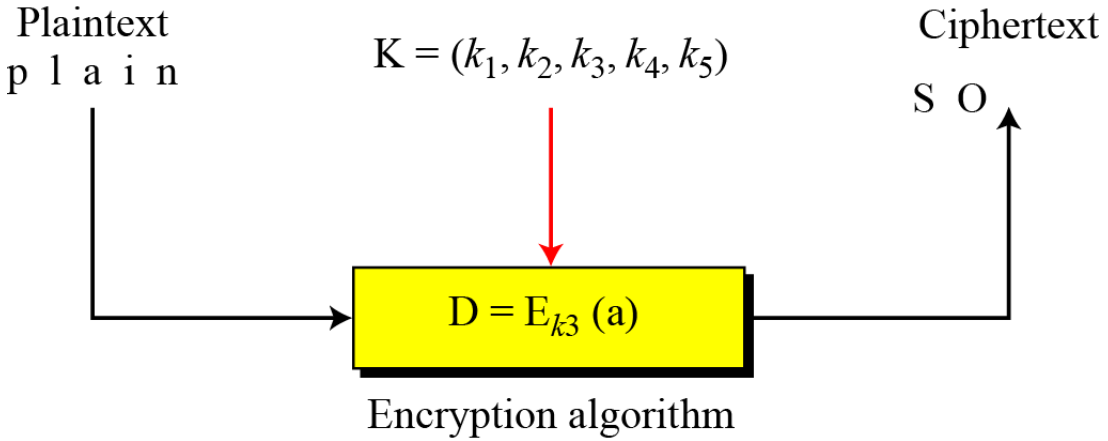
- The literature divides the symmetric ciphers into two broad categories:
  - stream ciphers and block ciphers.

Stream Cipher

- Stream Cipher:
  - Encrypt digital data stream of plaintext message one bit or one character at a time
  - Call the plaintext stream P, the ciphertext stream C, and the key stream K.

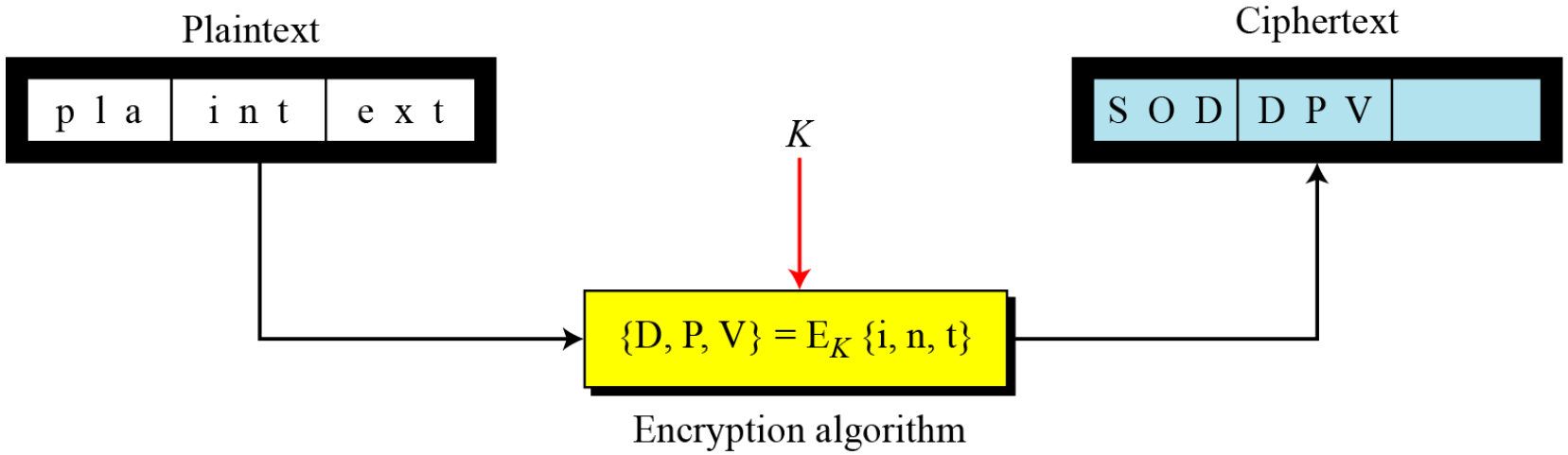
$$P = P_1P_2P_3, \ldots \qquad C = C_1C_2C_3, \ldots \qquad K = (k_1, k_2, k_3, \ldots)$$

$$C_1 = E_{k1}(P_1) \qquad C_2 = E_{k2}(P_2) \qquad C_3 = E_{k3}(P_3) \ldots$$



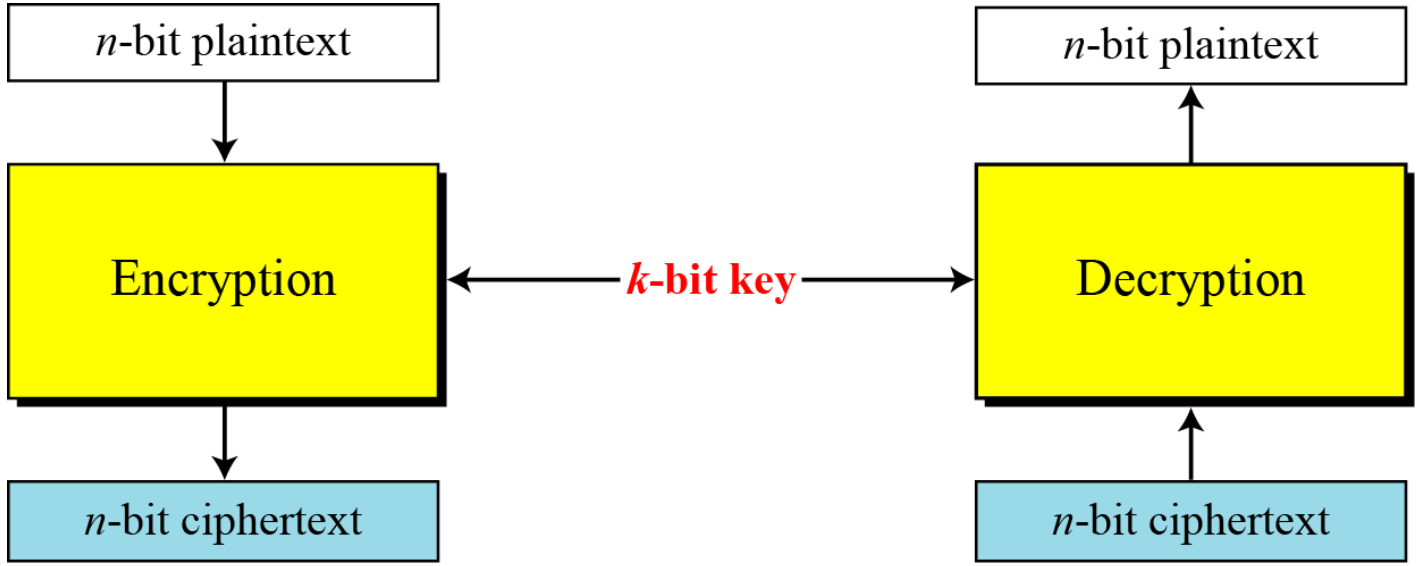
Block Cipher

- Block cipher:
  - Processes the plaintext in relatively large blocks (e.g. n≥64 bits)
  - Creating a group of cipher text of the same size n
  - A single key is used to encrypt the whole block even if the key is made of multiple values.



Introduction to modern Symmetric Key Ciphers

- Modern Block Ciphers
  - A symmetric-key modern block cipher encrypts an n-bit block of plaintext or decrypts an n-bit block of ciphertext.
  - The encryption or decryption algorithm uses a k-bit key.



Block Cipher

- A symmetric key modern cipher encrypts an n bit block of plaintext or decrypts an n bit block of cipher-text (n: block length)

- Padding:
  - If the message has fewer than n bits, padding must be done to make it n bits.
  - If the message size is not a multiple of n, then it should be divided into n bit blocks and the last block should be padded.

Transposition Cipher

- A modern block cipher can be designed to act as a
  - substitution cipher or a transposition cipher.

- Transposition Ciphers:
  - Involves rearrangement of bits, without changing the value
  - Consider an n bit cipher
  - How many such rearrangements are possible?
    - n!
  - How many key bits are necessary?
    - ceil[$\log_2$ (n!)]

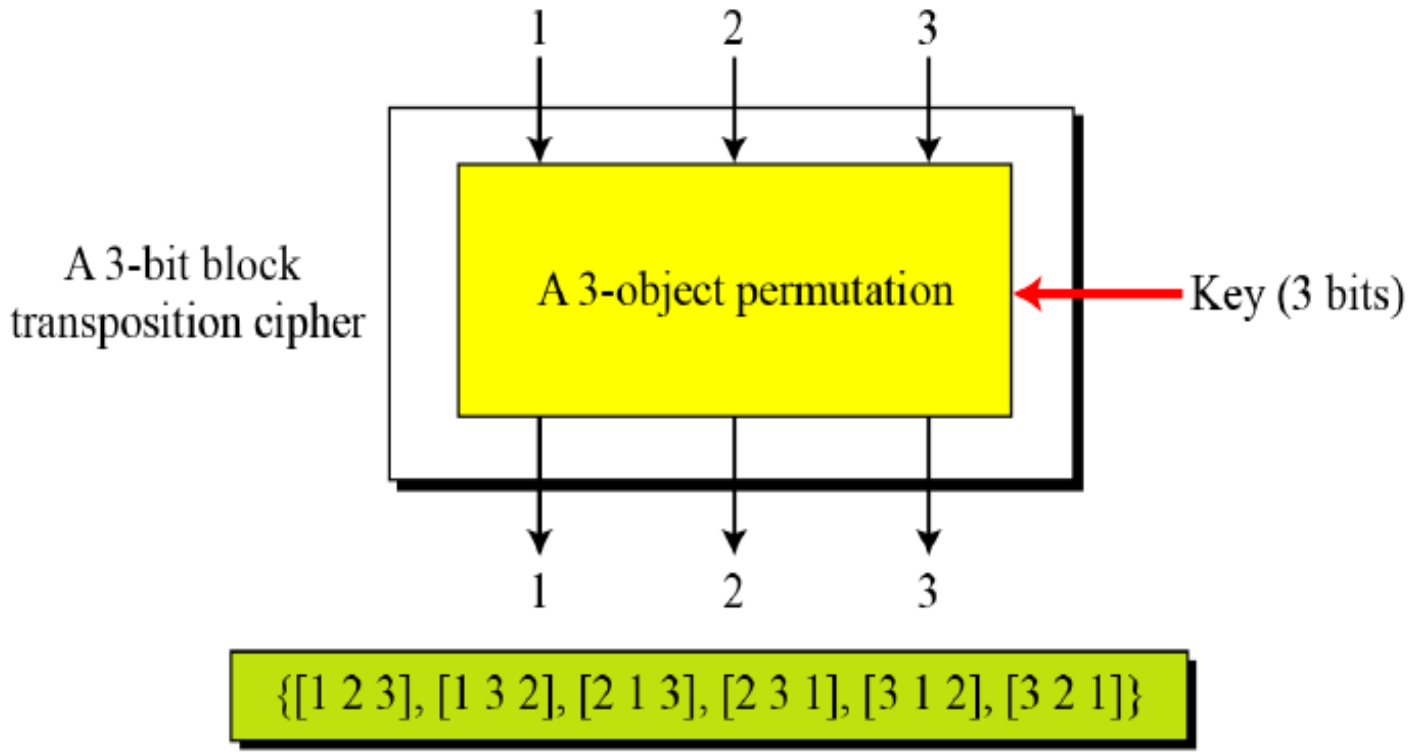
Transposition Cipher

- Show the model and the set of permutation tables for a 3-bit block transposition cipher where the block size is 3 bits.

- Solution :
  - The set of permutation tables has 3! = 6 elements.
  - Key : ceil($\log_2 6$)=3 bits

Transposition Cipher

A transposition block cipher modeled as a permutation



The set of permutation tables with 3! = 6 elements

Substitution Cipher

- Substitution Ciphers:
  - It does not transpose bits, but substitutes values
  - Can we model this as a permutation?
  - Yes. The n bit inputs and outputs can be represented as 2^n bit sequences, with one 1 and the rest 0's.
  - This can be thus modelled as a transposition.

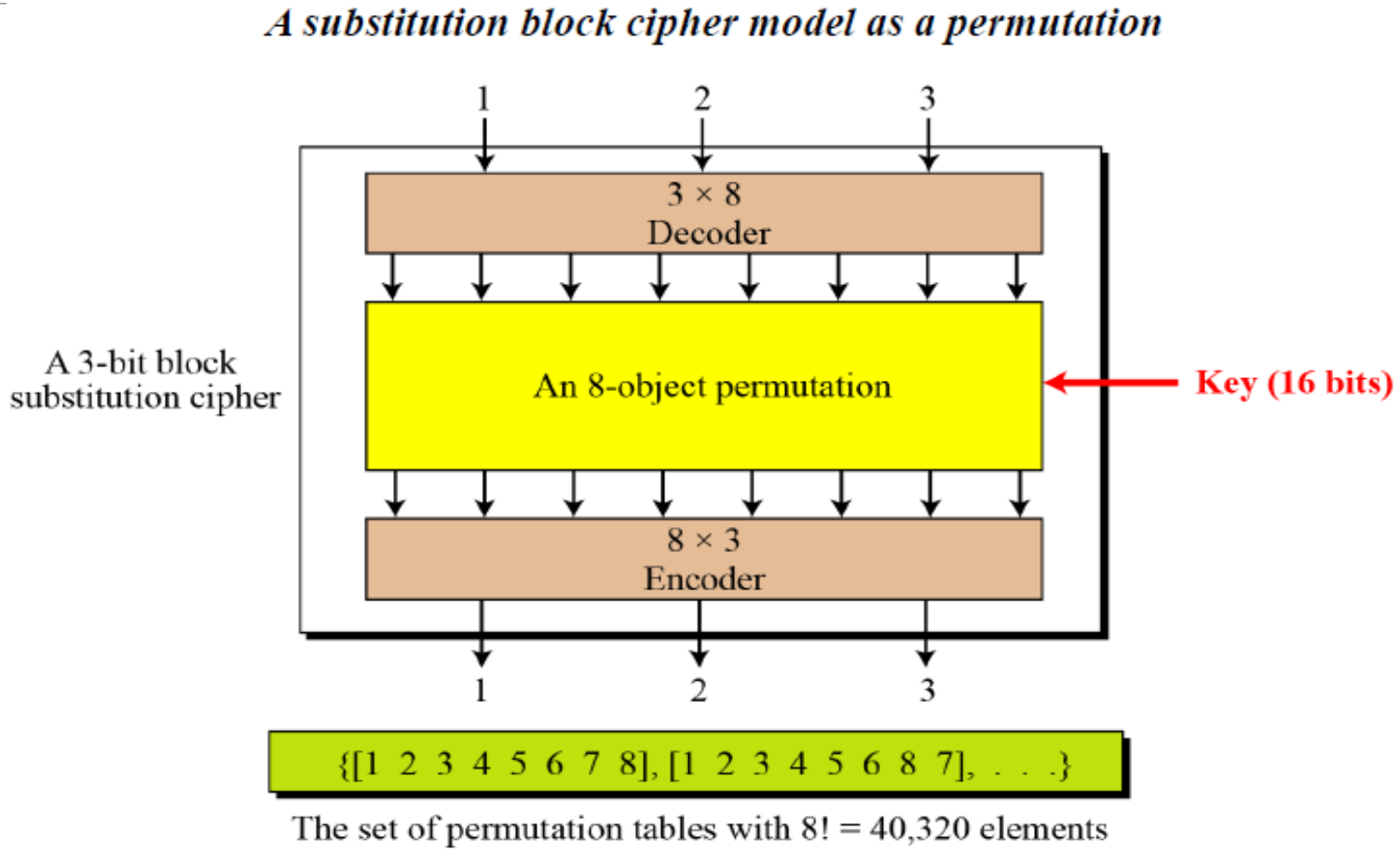
- Thus it is a permutation of 2^n values, thus needs
  - Key - ceil[$\log_2(2^n!)$] bits.

Substitution Cipher

- Show the model and the set of permutation tables for a 3-bit block substitution cipher.

- Solution:
- Figure next shows the model and the set of permutation tables.
- The key is also much longer, $[\log_2(40,320)] = 16$ bits.

Substitution Cipher



Block Cipher

- A modern block cipher can be designed to act as a substitution cipher or a transposition cipher

- To be resistant to exhaustive-search attack, a modern block cipher needs to be designed as a substitution cipher.

Example

- Suppose that we have a block cipher where n = 64. If there are 10 1's in the ciphertext, how many trial-and error tests does Eve need to do to recover the plaintext from the intercepted ciphertext in each of the following cases?

  a. The cipher is designed as a substitution cipher.
  b. The cipher is designed as a transposition cipher

Example

- Solution

- In the first case, Eve has no idea how many 1's are in the plaintext. Eve needs to try all possible 2^{64} 64-bit blocks to find one that makes sense.

- In the second case, Eve knows that there are exactly 10 1's in the plaintext. Eve can launch an exhaustive-search attack using only those 64-bit blocks that have exactly 10 1's.

Components of a Modern Block Cipher

- Most important components:
  - PBox: It is a key-less fixed transposition cipher
  - SBox: It is a key-less fixed substitution cipher

- **Diffusion:** it hides the relationship between the ciphertext and the plaintext
  - **Diffusion** refers to the property that redundancy in the statistics of the plaintext is "dissipated" in the statistics of the ciphertext.
  - Changing a single bit in a block of plain text will have the effect of changing each bit of cipher text with probability of 0.5
  - However, this changes are scattered across the block of cipher text.
  - Transposition enhances diffusion!!!

Components of a Modern Block Cipher

- **Confusion:** it hides the relationship between the ciphertext and the key
- **Confusion** refers to making the relationship between the key and the ciphertext as complex and as involved as possible
- a single bit change in key - changes roughly half of the bits in corresponding cipher text,
- moreover positions of changed (flipped) bits are random!
- Substitution enhances confusion!!

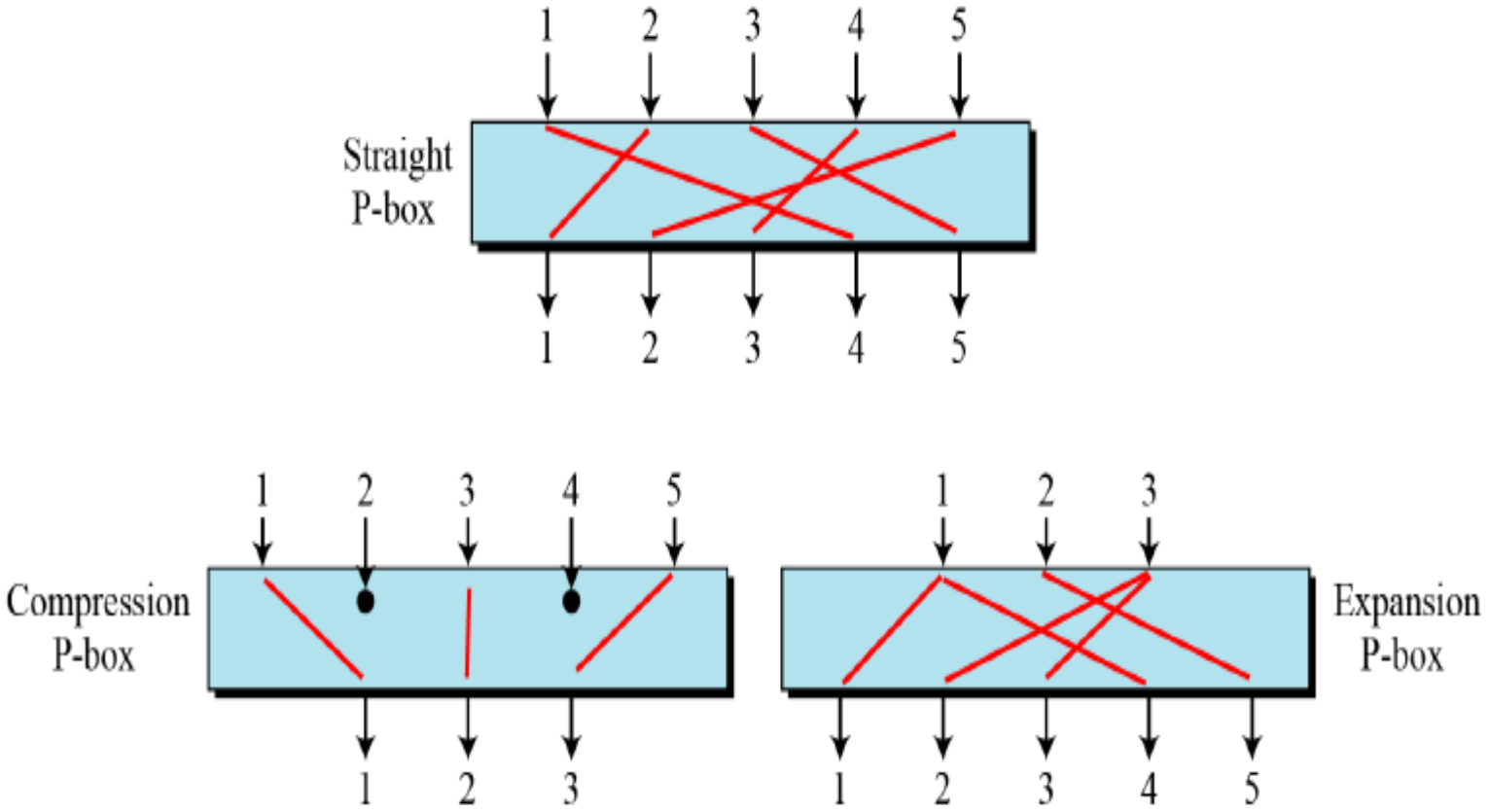
Principle of Confusion and Diffusion

- The design principles of Block Cipher depends on these properties
- The S-Box is used to provide **confusion**,
  - as it is dependent on the unknown key
- The P-Box is fixed, and there is no confusion due to it
  - But it provides **diffusion**
- Properly combining these is necessary.

Permutation Box - P Box

- Performs permutation or rearrangement of the bits in the input

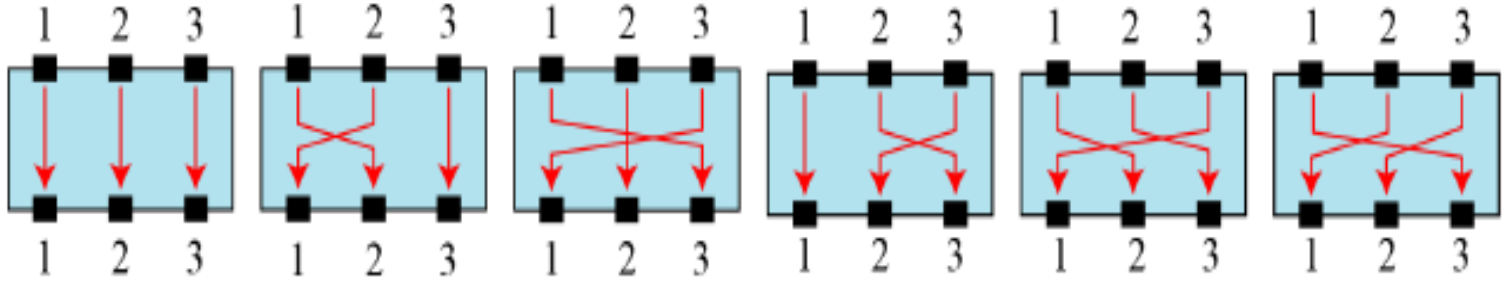
Three types of P-Boxes



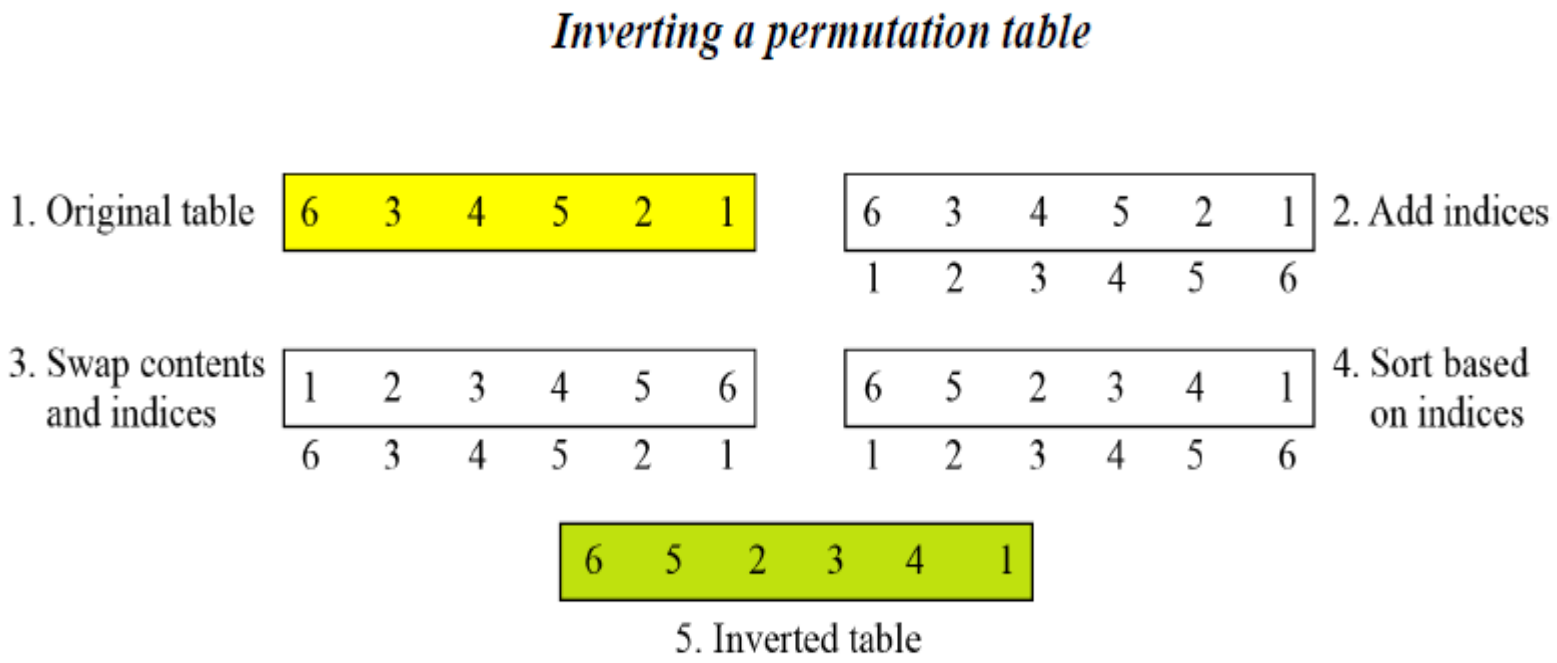
Example

- Figure below shows all 6 possible mappings of a 3 × 3 P-box

The possible mappings of a 3 × 3 P-box

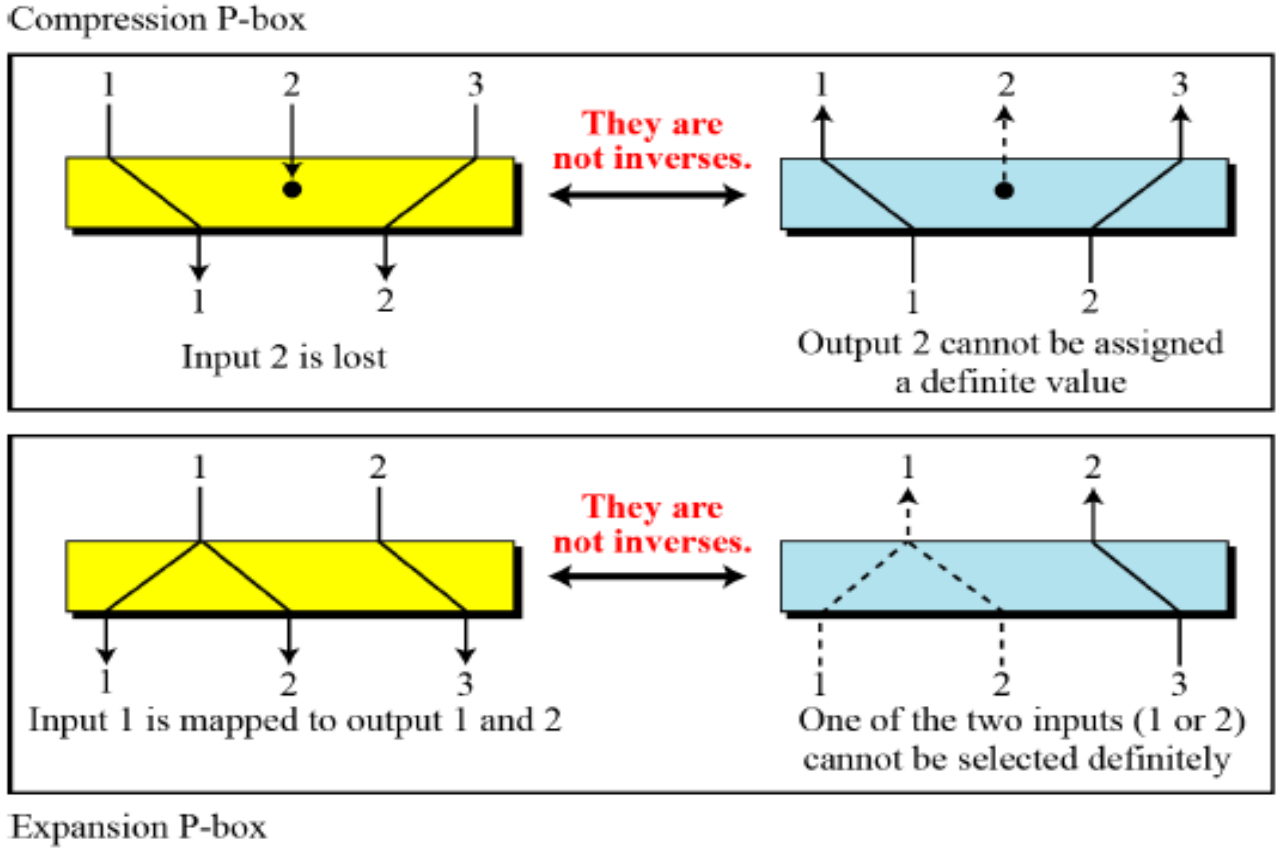


Example


Inverting a permutation table
1. Original table
6 3 4 5 2 1
6 3 4 5 2 1
1 2 3 4 5 6
2. Add indices
3. Swap contents and indices
1 2 3 4 5 6
6 3 4 5 2 1
6 5 2 3 4 1
1 2 3 4 5 6
4. Sort based on indices
6 5 2 3 4 1
5. Inverted table

Example

Compression and expansion P-boxes are non-invertible



Cannot invert expansion box if two <u>different</u> outputs would map to the same input

SBox

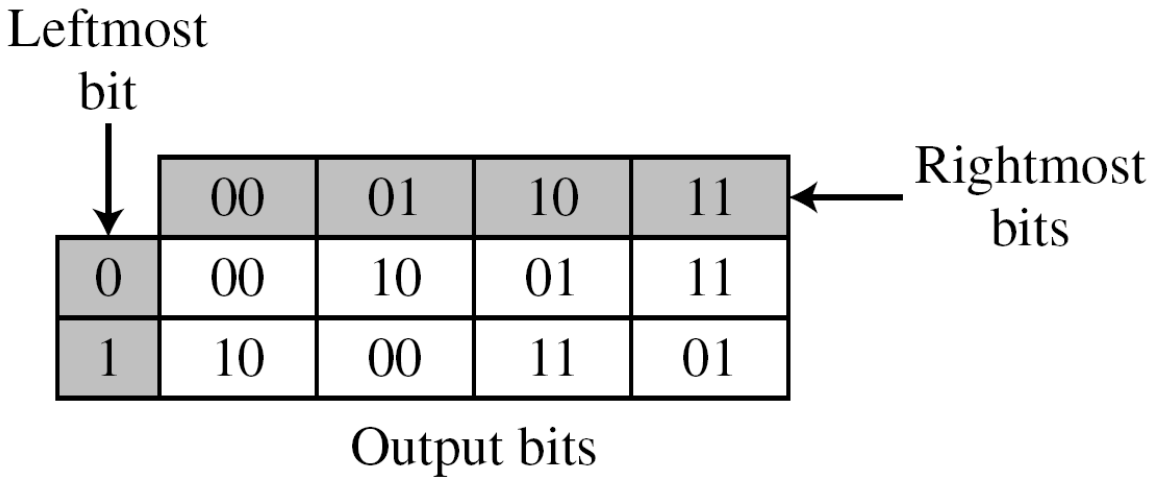
- An SBox (substitution box) is an nxm substitution box,
  - where m and n are not necessarily same

- Each output bit is a Boolean function of the inputs

$$y_1 = f_1(x_1, x_2, \ldots, x_n)$$
$$y_2 = f_2(x_1, x_2, \ldots, x_n)$$
$$\ldots$$
$$y_m = f_m(x_1, x_2, \ldots, x_n)$$

SBox - Example

- The following table defines the input/output relationship for an S-box of size 3 × 2 (3 bits input, 2 bits output). The leftmost bit of the input defines the row; the two rightmost bits of the input define the column. The two output bits are values on the cross section of the selected row and column.



- Based on the table, an input of 010 yields the output 01. An input of 101 yields the output of 00.

SBox – Example (6×4-bit)

- Selecting the row using the outer two bits (the first and last bits), and the column using the inner four bits.
- For example, an input "**0**11011**1**" has outer bits "**01**" and inner bits "1101";
  - the corresponding output would be "1001

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

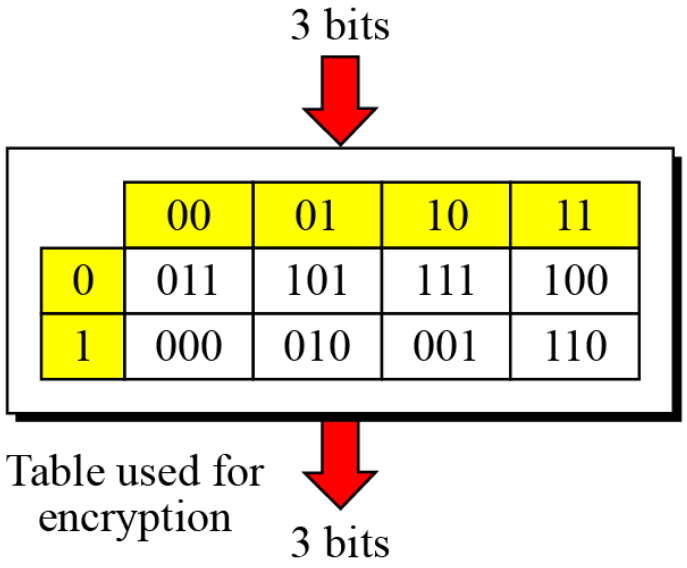
SBox

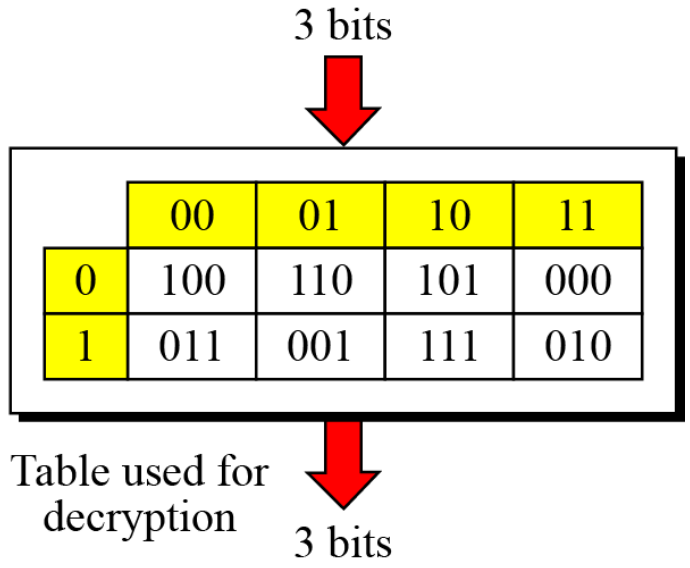
- An S-box may or may not be invertible.

- In an invertible S-box, the number of input bits should be the same as the number of output bits.

SBox - Example

- Example shows invertible S-box.
- Encryption : If the input to the left box is 001, the output is 101.
- Decryption : The input 101 in the right table creates the output 001, which shows that the two tables are inverses of each other.





Exclusive-Or (XOR)

- An important component in most block ciphers is the exclusive-or operation.
- The inverse of a component in a cipher
  - if the component represents a unary operation (one input and one output).
- keyless P-box or a keyless S-box can be made invertible because they have one input and one output.

Exclusive-Or (XOR)

- An XOR is a binary operation.
  - The inverse of an XOR is possible
  - Only if one of the inputs is fixed (is the same in encryption and decryption).

- If one of the inputs is the key,
  - which is the same in encryption and decryption,
  - XOR self-invertible

Properties of XOR

Ex-or is a binary operator, which results in 1 when both the inputs have a different logic. Otherwise, it computes 0.

Symbol: $\oplus$

Closure: Result of exoring two n bit numbers is also n bits.

Associativity: Allows to use more than one '$\oplus$'s in any order:

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z$$

Commutavity: $x \oplus y = y \oplus x$

Identity: The identity element is the n bit 0, represented by

$$(00...0)=0^n$$

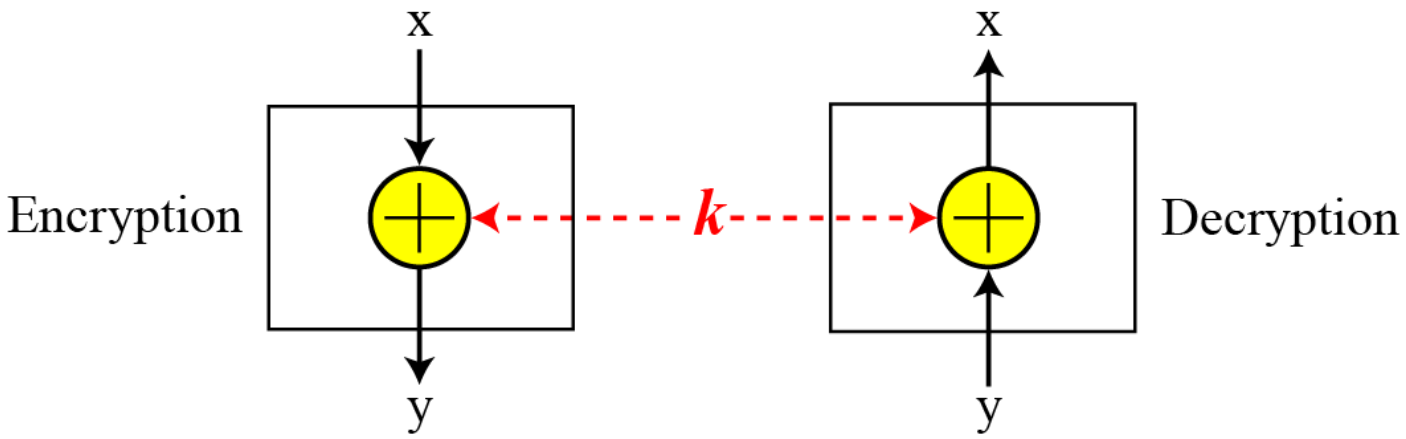
Thus, $x \oplus 0^n = x$

Inverse: Each word is the additive inverse of itself.

Thus, $x \oplus x = 0^n$

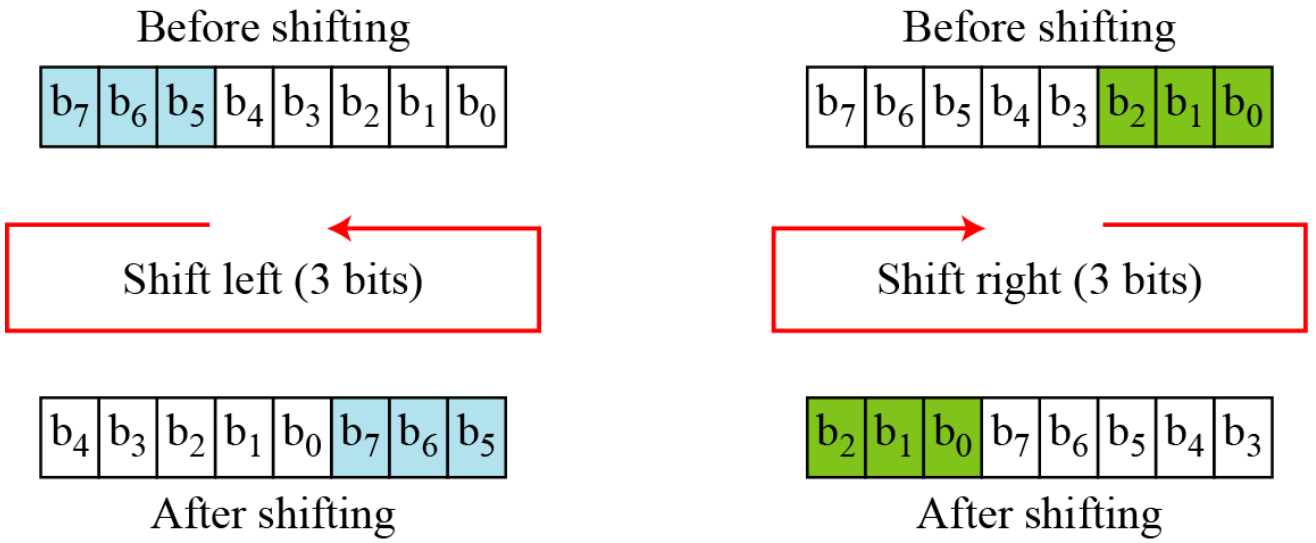
Exclusive-Or (XOR)

- Invertibility of the exclusive-or operation



Circular Shift

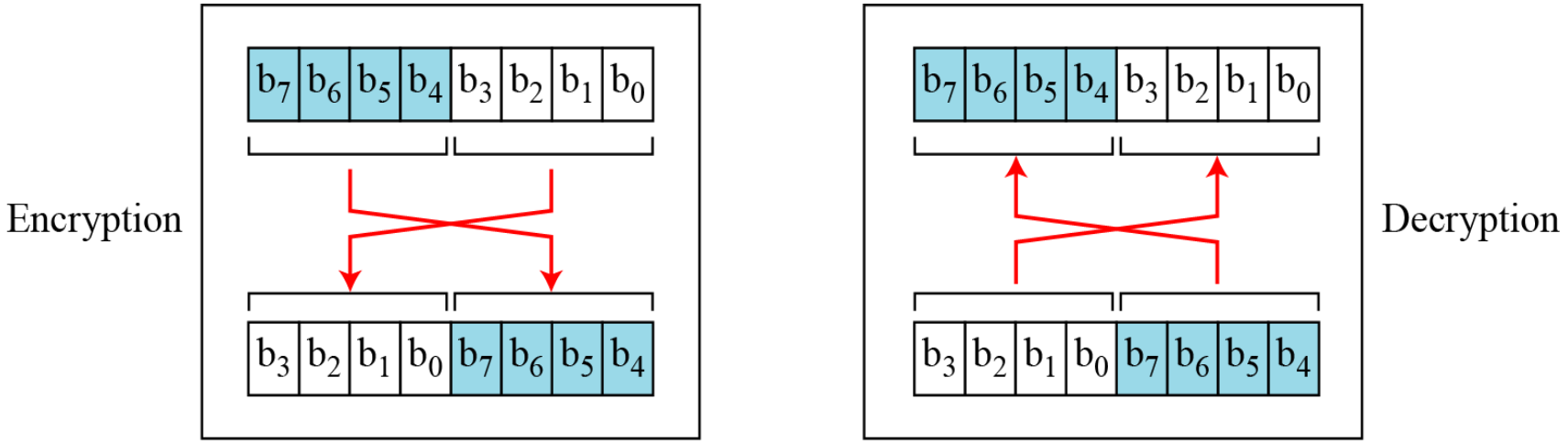
- It shifts each bit in an n-bit word k positions to the left.
  - The leftmost k bits become the rightmost bits.
  - Invertible Transformation



Circular Shift

- Swap:
  - A special type of shift operation where k=n/2

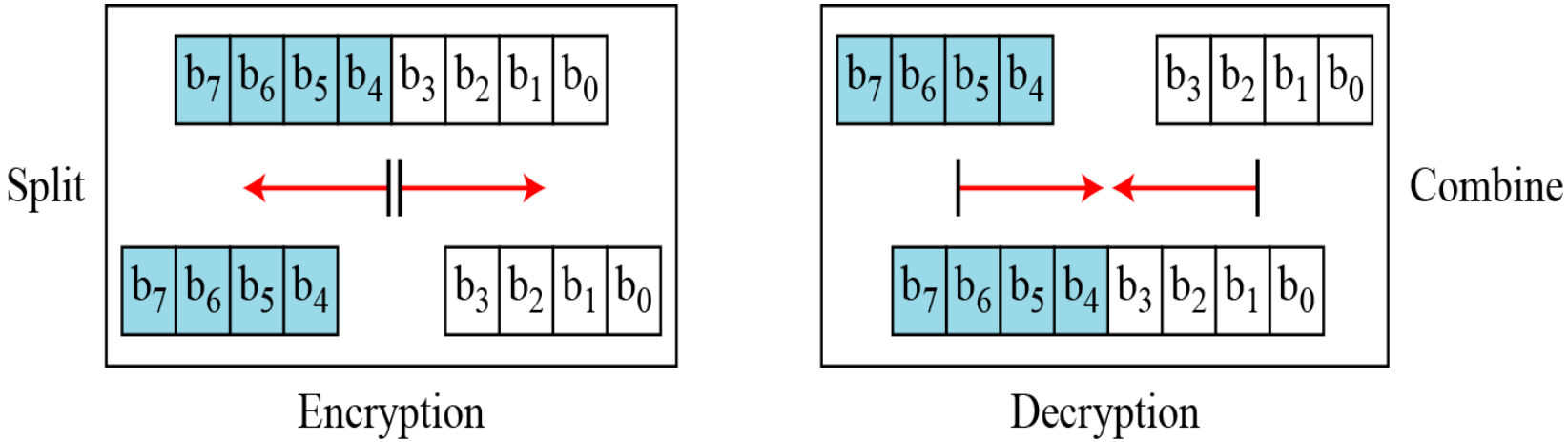
- Example : Swap operation on an 8-bit word



Circular Shift

- Two other operations found in some block ciphers are split and combine.

- Example: Split and combine operations on an 8-bit word



Improving Security

- Two key ideas:

1. **Product Ciphers:**
   - Shannon introduced the concept of a product cipher.
   - A product cipher is a complex cipher combining substitution, permutation, and other components (e.g., shift, exclusive OR, etc.)
   - combines two or more transformations
     - resulting cipher is more secure than the individual components
   - If S1: A → B and S2: B → C are ciphers,
   - so S = S1 × S2 : A → C

Improving Security

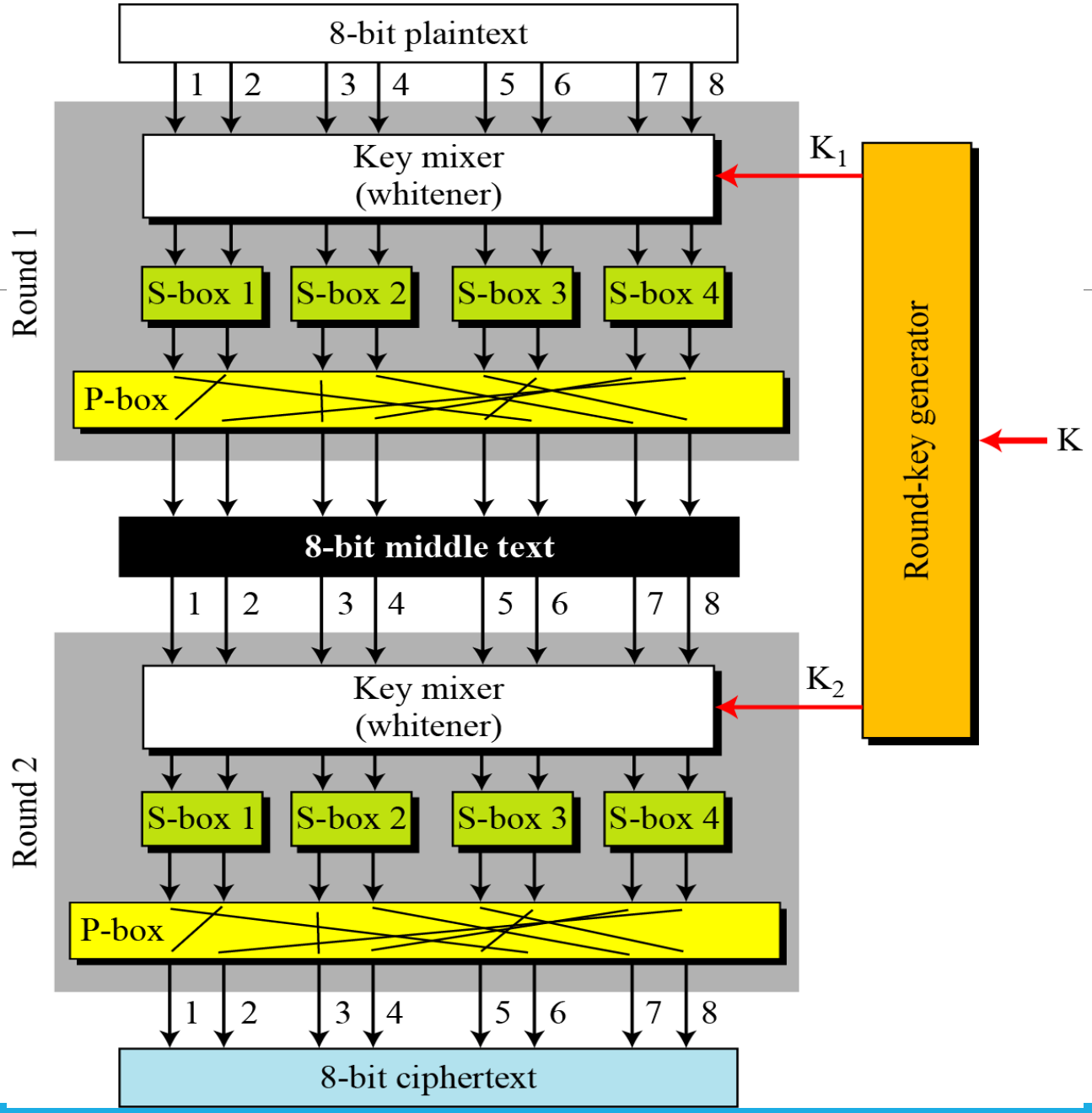
2. Iterated Cipher:

- a block cipher involving the sequential repetition of an internal function called a round function controlled by some parameters
- If $S : A \rightarrow A$ is a cipher, consider
  - $S^k = S \times S \times \ldots \times S : A \rightarrow A$
- A product of S with itself is denoted S x S or S^2
- As k increases, S^k is expected to have higher security

Improving Security

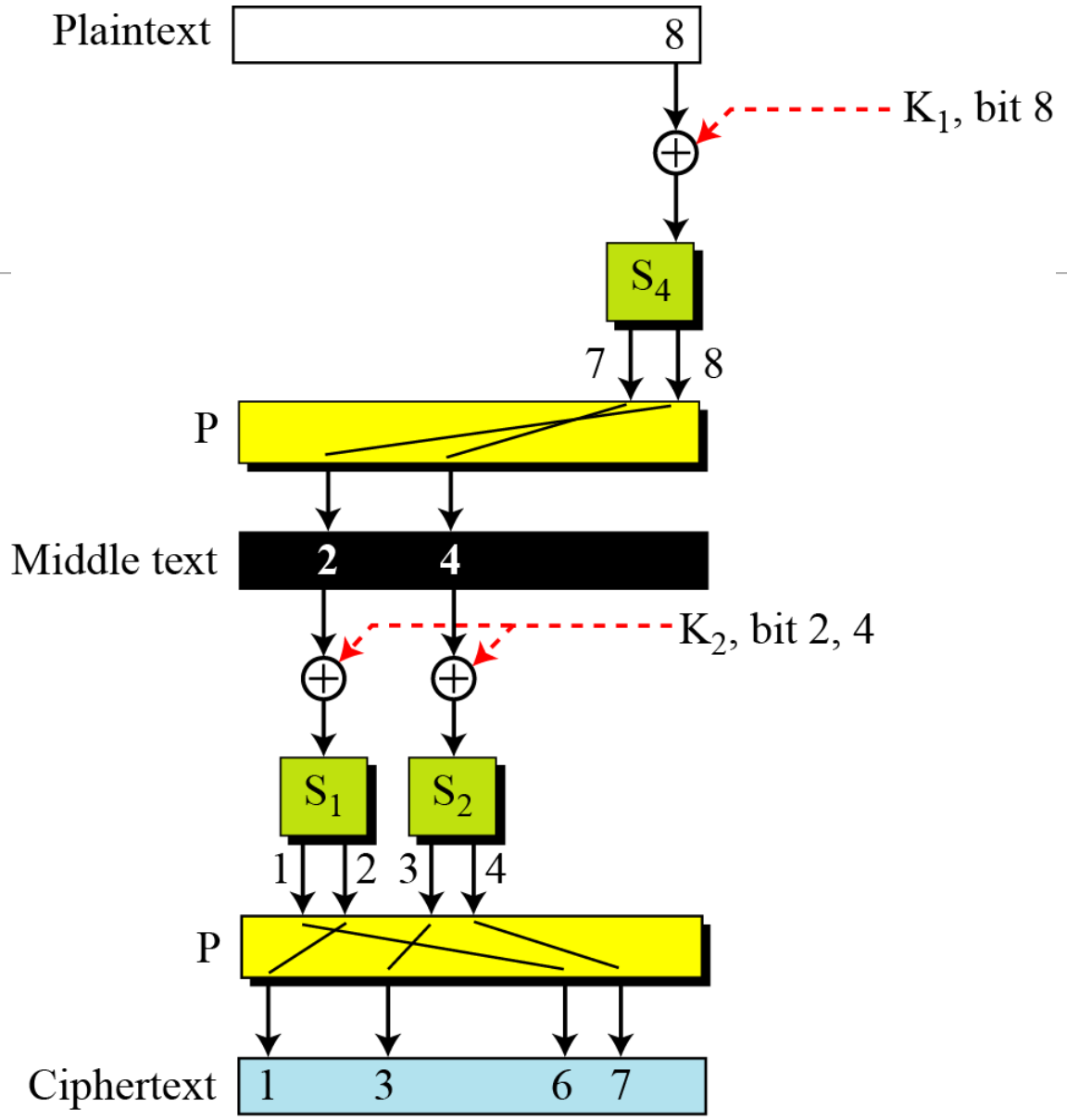
- Diffusion and confusion can be achieved using iterated product ciphers where each iteration is a combination of S-boxes, P-boxes, and other components.

A product cipher made of two rounds



A product cipher made of two rounds

- Diffusion and confusion in a block cipher
- **Avalanche effect:**
  - A small change in a plaintext or key will produces a large change in the cipher-text



Practical Ciphers

- Large data blocks
- More S-Boxes
- More rounds
- These help to improve the diffusion and confusion in the cipher.

Two Classes of Product Ciphers

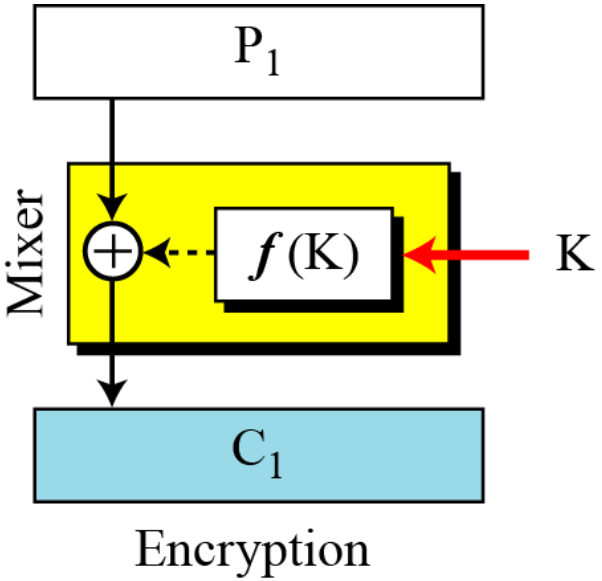
- Modern block ciphers are all product ciphers, but they are divided into two classes.

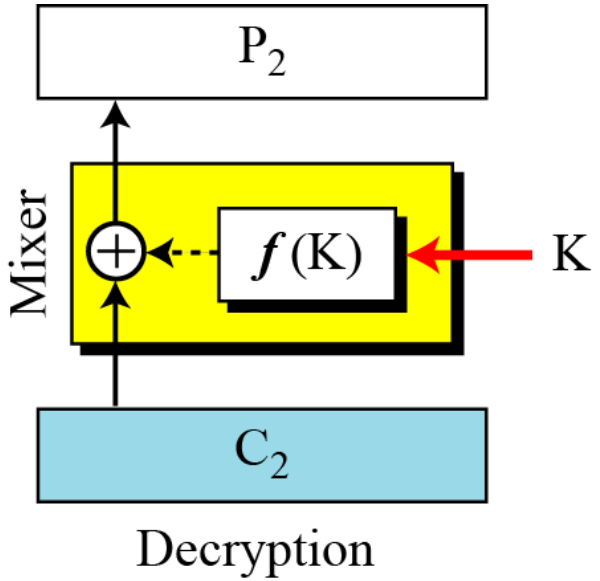
1. Feistel ciphers
   - Ex. DES (Data Encryption Standard)

2. Non-Feistel ciphers (Substitution Permutation Networks)
   - Ex. AES (Advanced Encryption System)

Feistel Cipher

- **Feistel cipher** refers to a type of block cipher design, not a specific cipher





Feistel Cipher - Example

- plaintext and cipher-text - 4 bits long
- key - 3 bits long.
- Function **f(k):**
  - takes the first and third bits of the key,
  - interprets these two bits as a decimal number,
  - squares the number,
  - interprets the result as a 4-bit binary pattern

Feistel Cipher - Example

- Show the results of encryption and decryption
  - if the original plaintext is 0111 and the key is 101

Feistel Cipher - Example

- Show the results of encryption and decryption
  - if the original plaintext is 0111 and the key is 101

- The function extracts the first and third bits of key: 11 in binary or 3 in decimal.

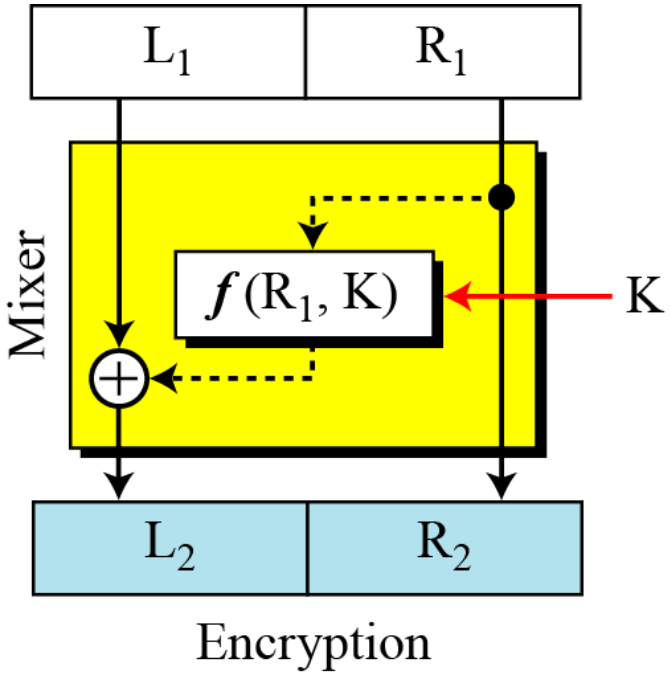
- The result of squaring is 9, which is 1001 in binary.

$$\textbf{Encryption: } C = P \oplus f(K) = 0111 \oplus 1001 = 1110$$

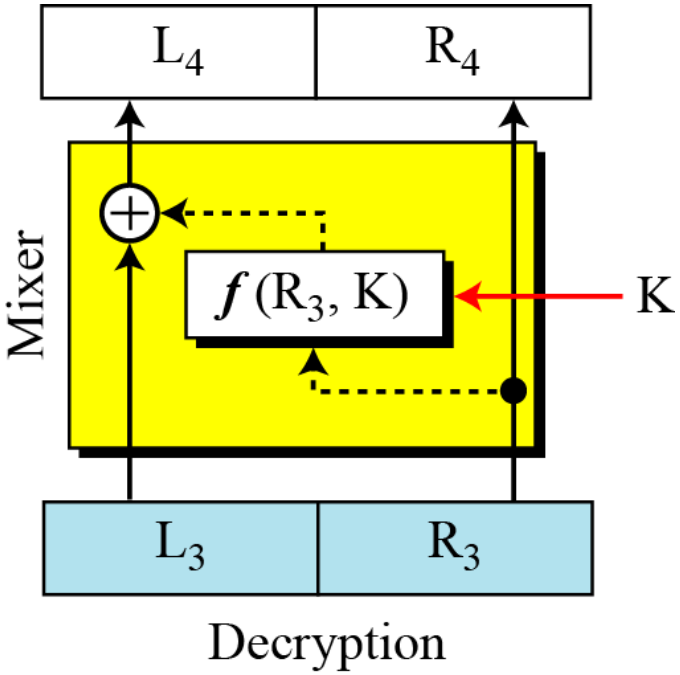
$$\textbf{Decryption: } P = C \oplus f(K) = 1110 \oplus 1001 = 0111$$

Feistel Cipher

- Improvement of the previous Feistel design

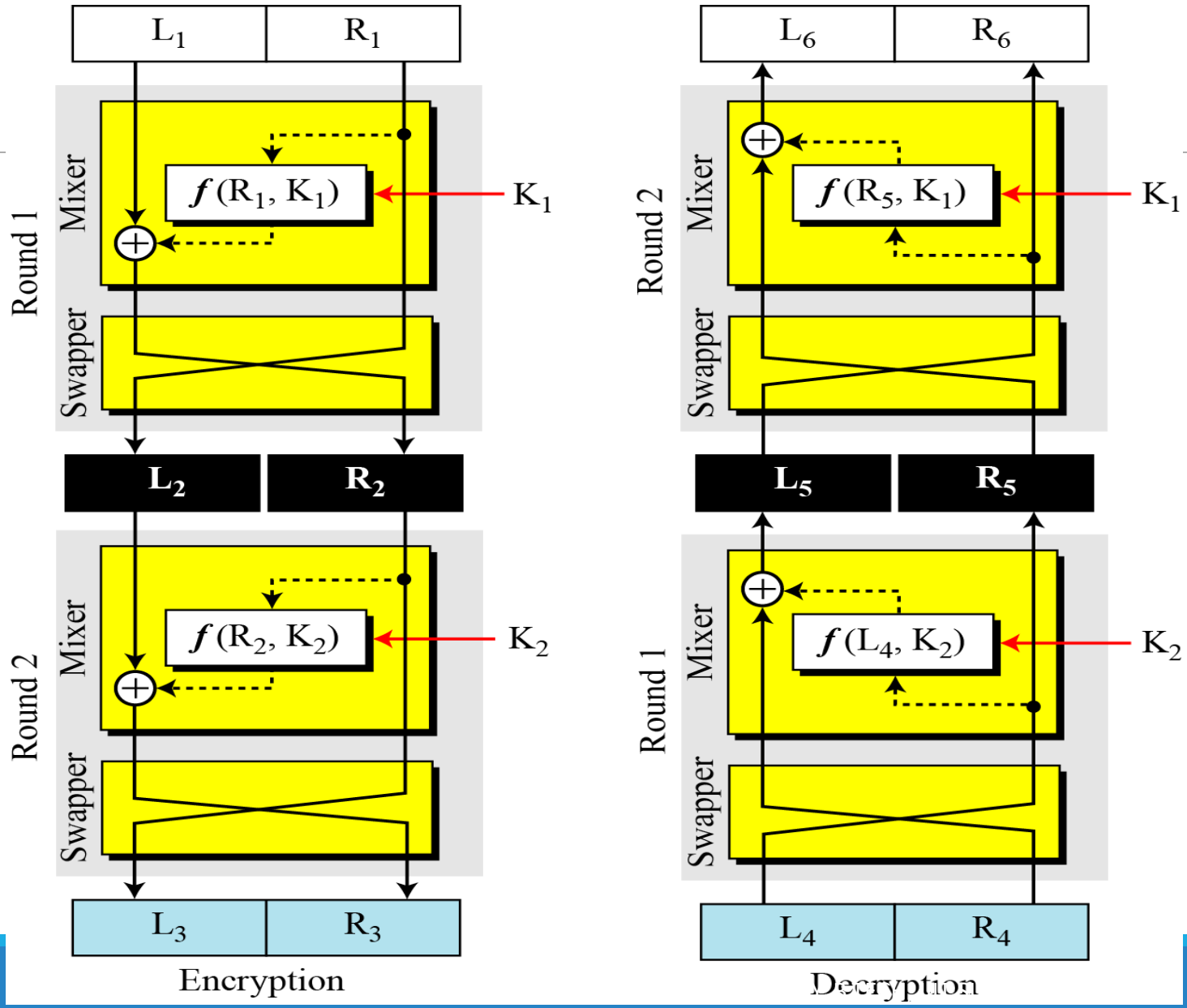


Encryption



Decryption

Final design of a Feistel cipher with two rounds



Final design of a Feistel cipher

- **Encryption:**
- Split plaintext block into left and right halves:
  - Plaintext = (L0,R0)
- For each round i=1,2,...,n, compute
  - **Li= Ri-1**
  - **Ri= Li-1 $\oplus$ f(Ri-1,Ki)**
- where f is **round function** and Ki is **subkey**
  - Ciphertext = (Ln,Rn)

Final design of a Feistel cipher

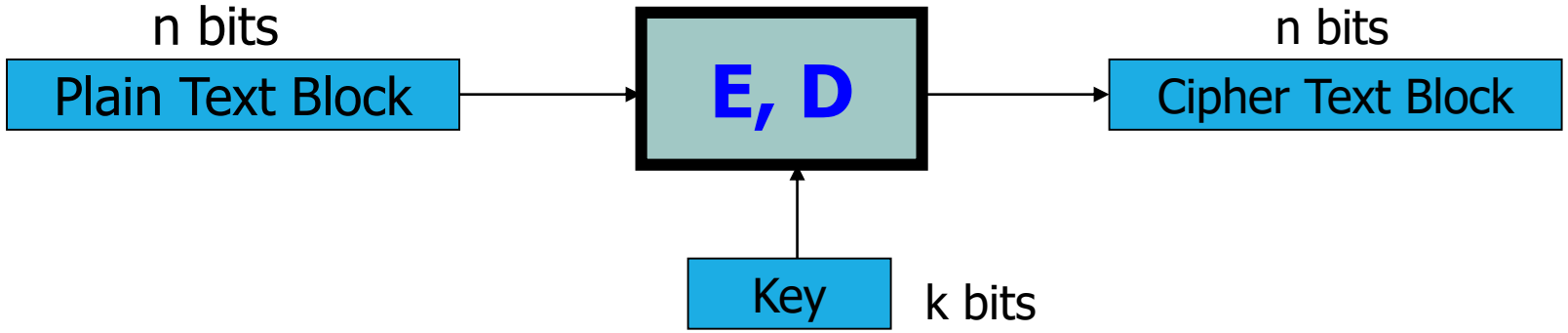
- Decryption: Ciphertext = (Ln,Rn)

- For each round i=n,n-1,…,1, compute
  - **Ri-1 = Li**
  - **Li-1 = Ri $\oplus$ f(Ri-1,Ki)**

- where f is round function and Ki is subkey
  - Plaintext = (L0,R0)

AES - Advanced Encryption Standard

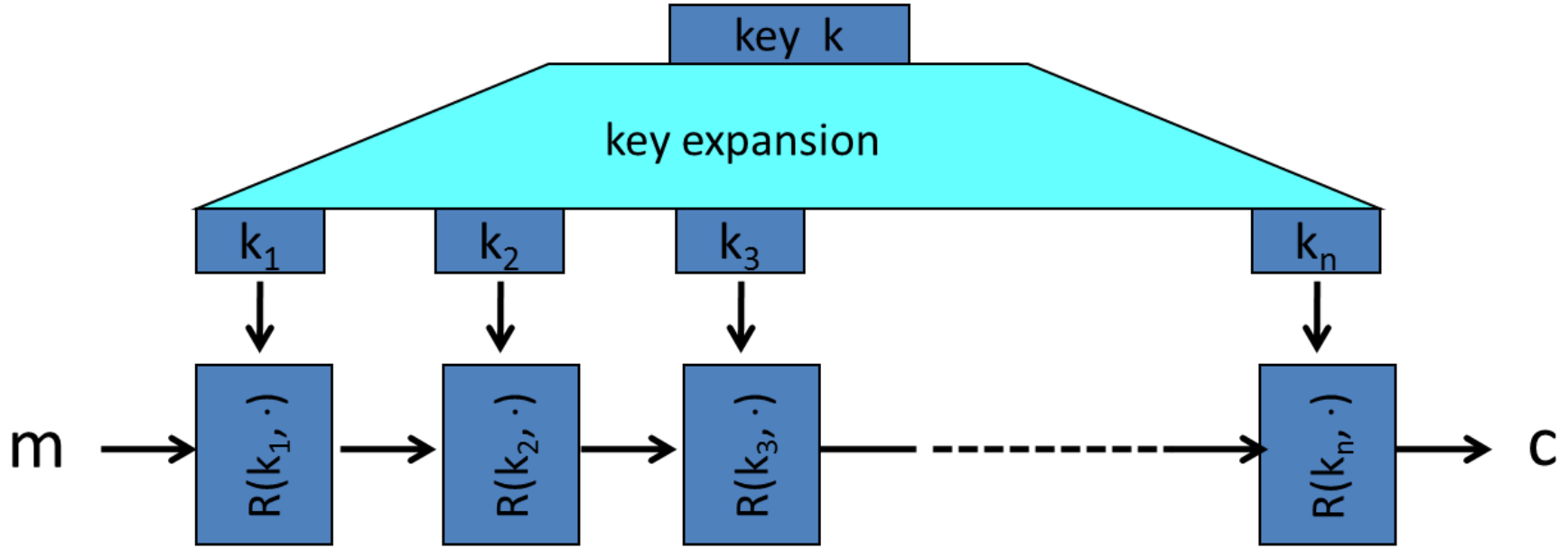
Block Ciphers



- Canonical examples:

1. 3DES: n= 64 bits, k = 168 bits
2. AES: n=128 bits, k = 128, 192, 256 bits

Block Ciphers Built by Iteration



- R(k,m) is called a round function

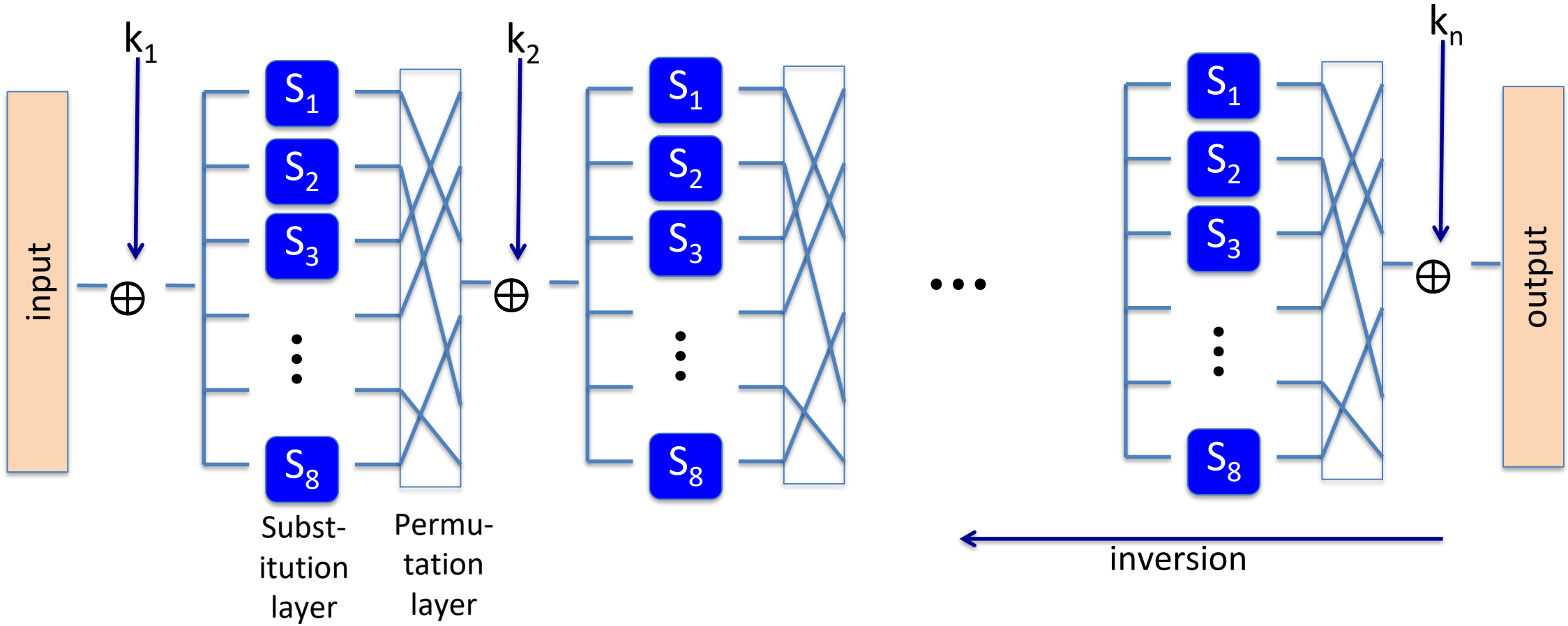
for 3DES (n=48), for AES-128 (n=10) (n = number of rounds)

Advanced Encryption Standard

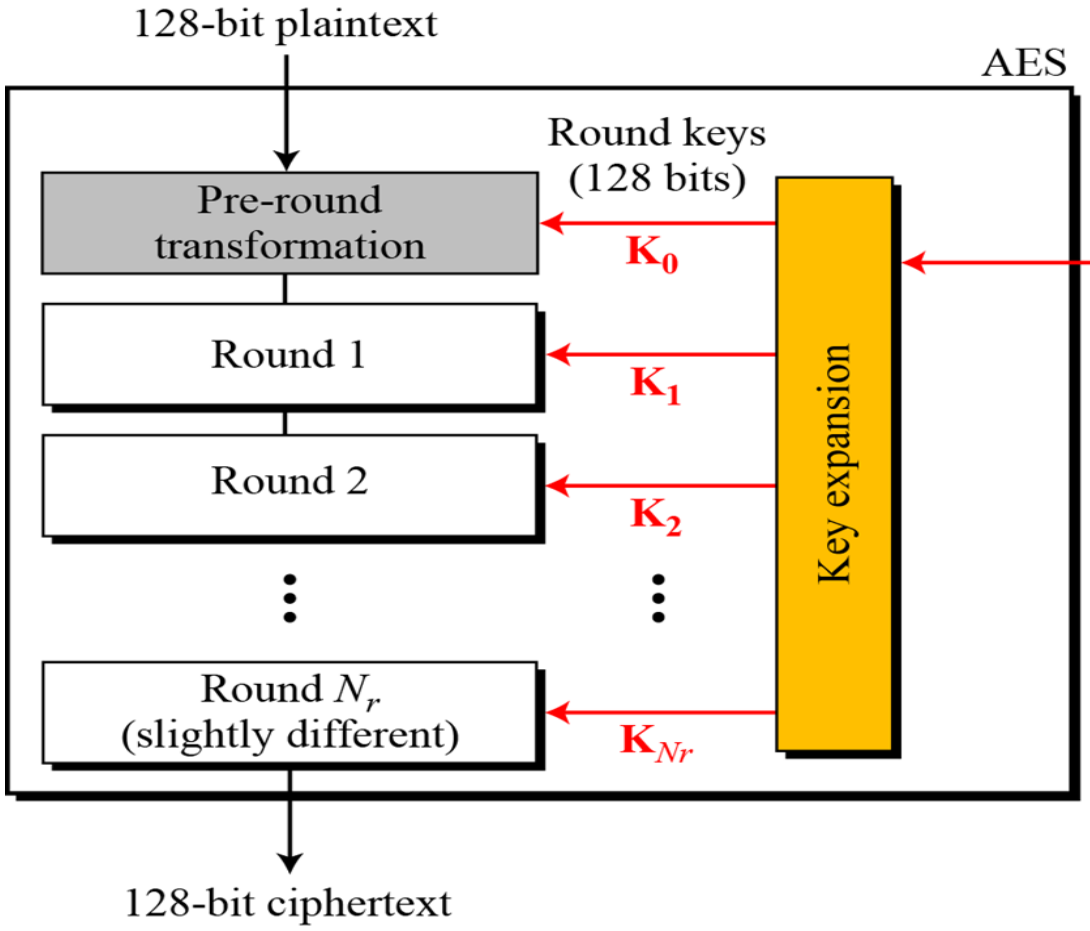
- 1997: NIST publishes request for proposal (Call for papers)
- 1998: 15 submissions. Five claimed attacks.
- 1999: NIST chooses 5 finalists (Rijndael, Serpent, 2fish, RC6, MARS)
- 2000: NIST chooses Rijndael as AES (designed in Belgium)

Key sizes: 128, 192, 256 bits. Block size: 128 bits

AES is a Substitution-Permutation Network (not Feistel)



AES General Design

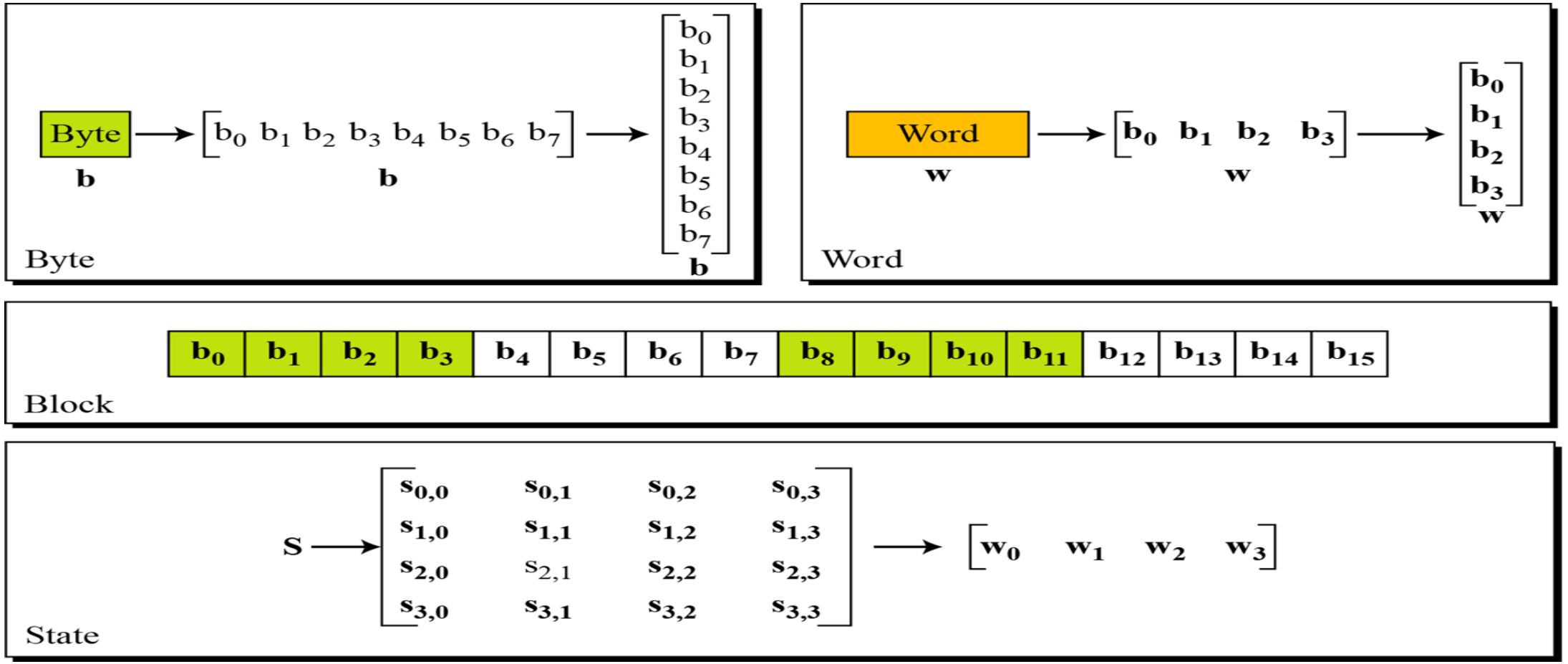


Cipher key
(128, 192, or 256 bits)

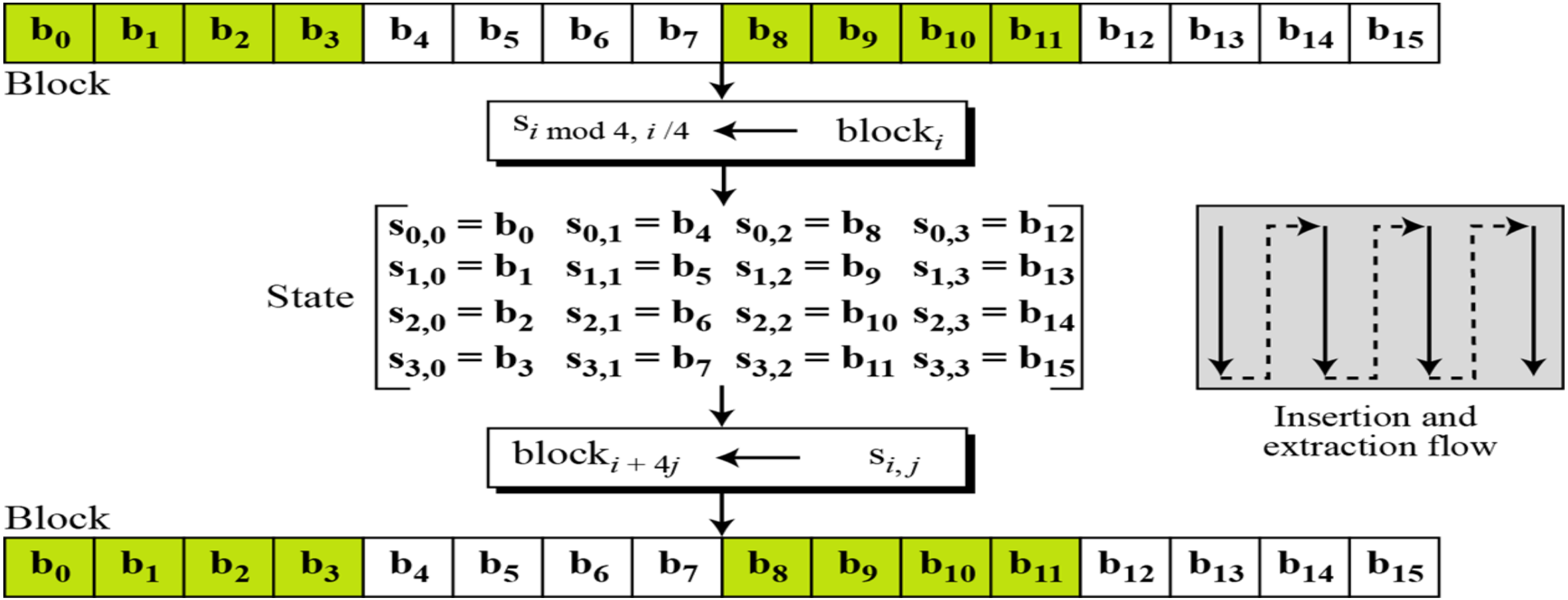
Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

AES (Data Units)



AES (Data Units)



Insertion and extraction flow

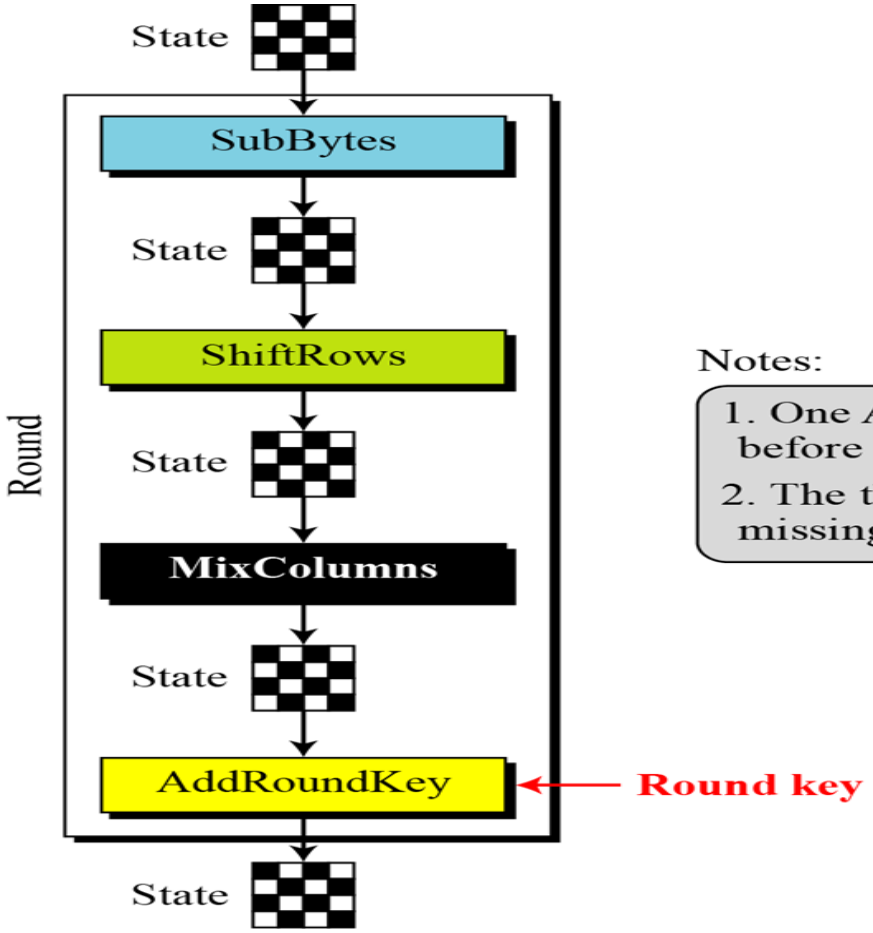
AES (Data Units)

- Changing Plain Text to State

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

Structure of Each Round



Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

- AES Transformations:

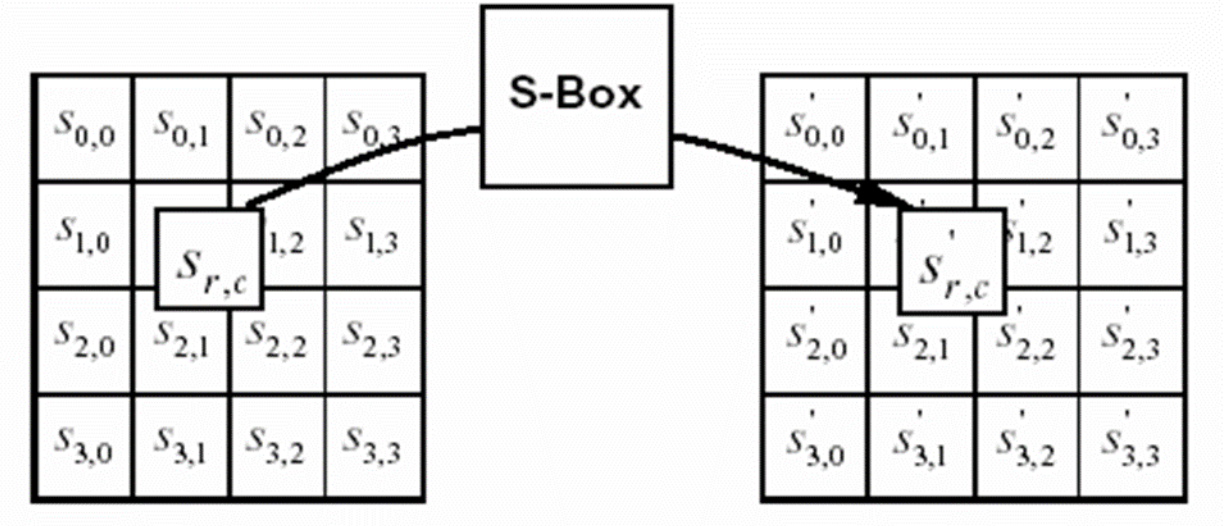
1. Substitution
2. Permutation
3. Mixing
4. Key-adding

Sub Bytes (Substitution)

- A simple substitution of each byte
- Uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- Each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
- S-box constructed using defined transformation of values in GF(2^8)

Sub Byte

- Non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box, invertible).



Sub Byte

19	A0	9a	E9
3d	F4	C6	F8
E3	e2	8d	48
Be	2b	2a	08

1 9

1 – Row

9 - Column

hex	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

x

Sub Byte Transformation

Table 7.1 *SubBytes transformation table*

	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

Sub Byte Transformation

Table 7.1 *SubBytes transformation table (continued)*

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
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	CB	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

InvSub Byte Transformation

Table 7.2 *InvSubBytes transformation table*

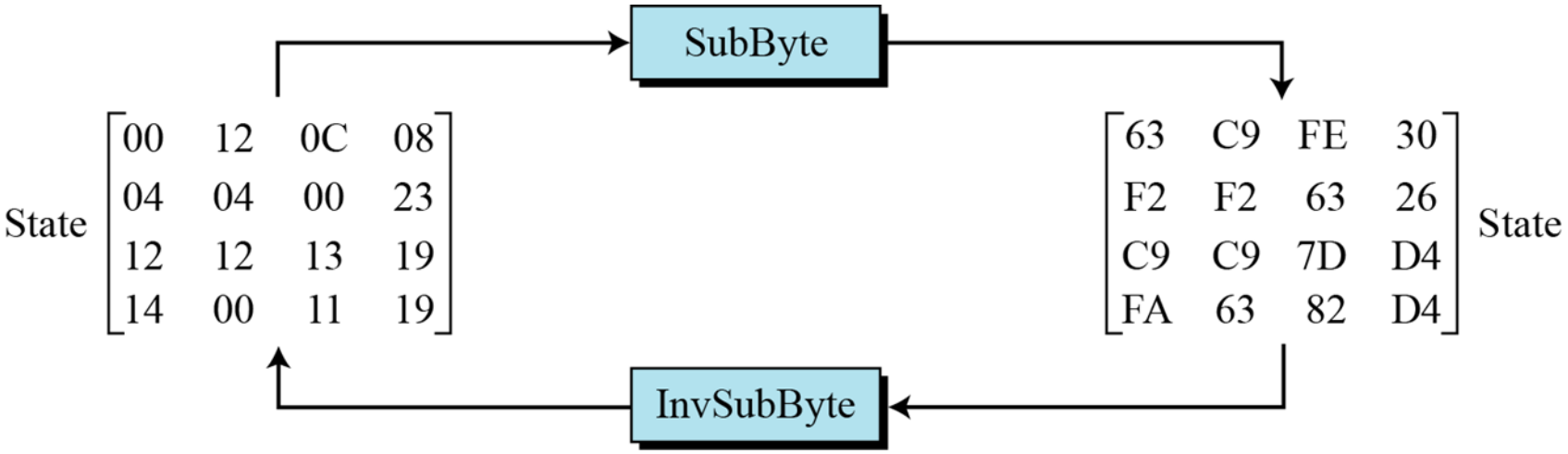
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	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

InvSub Byte Transformation

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

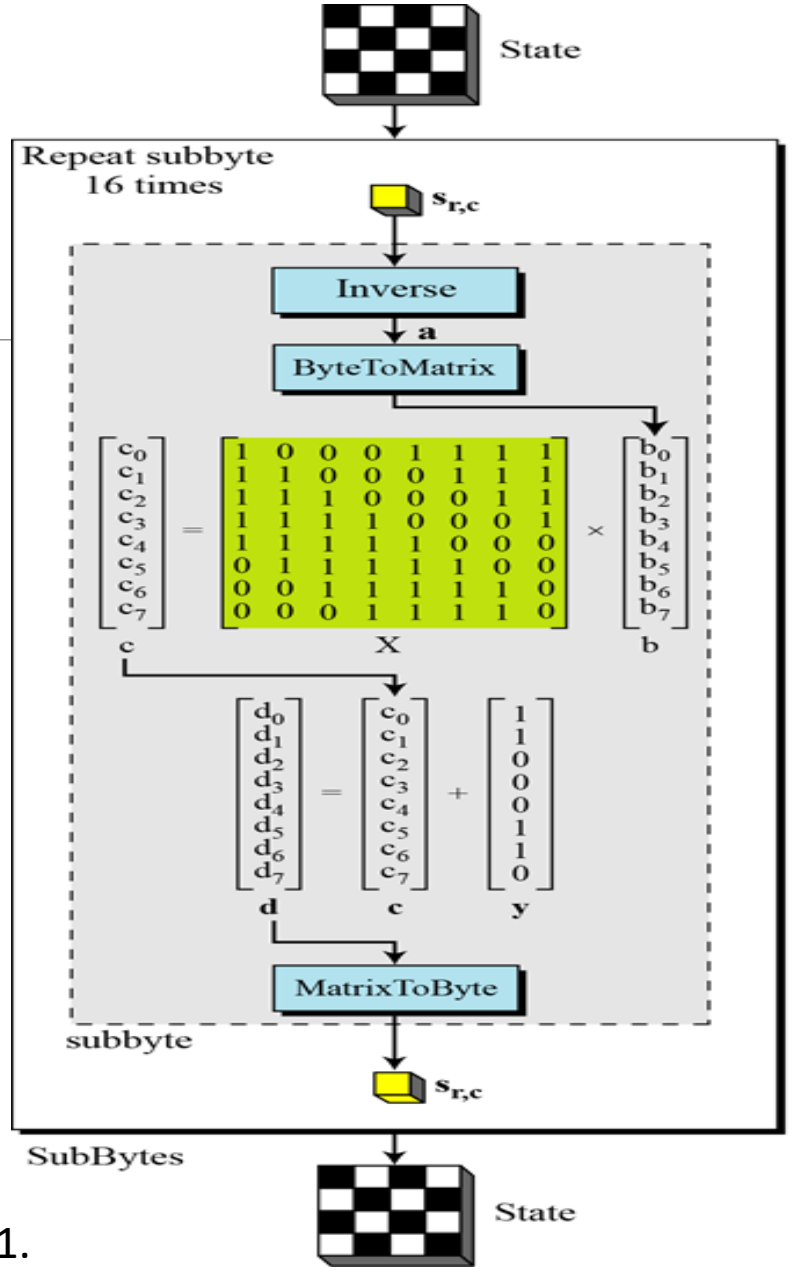
Substitution

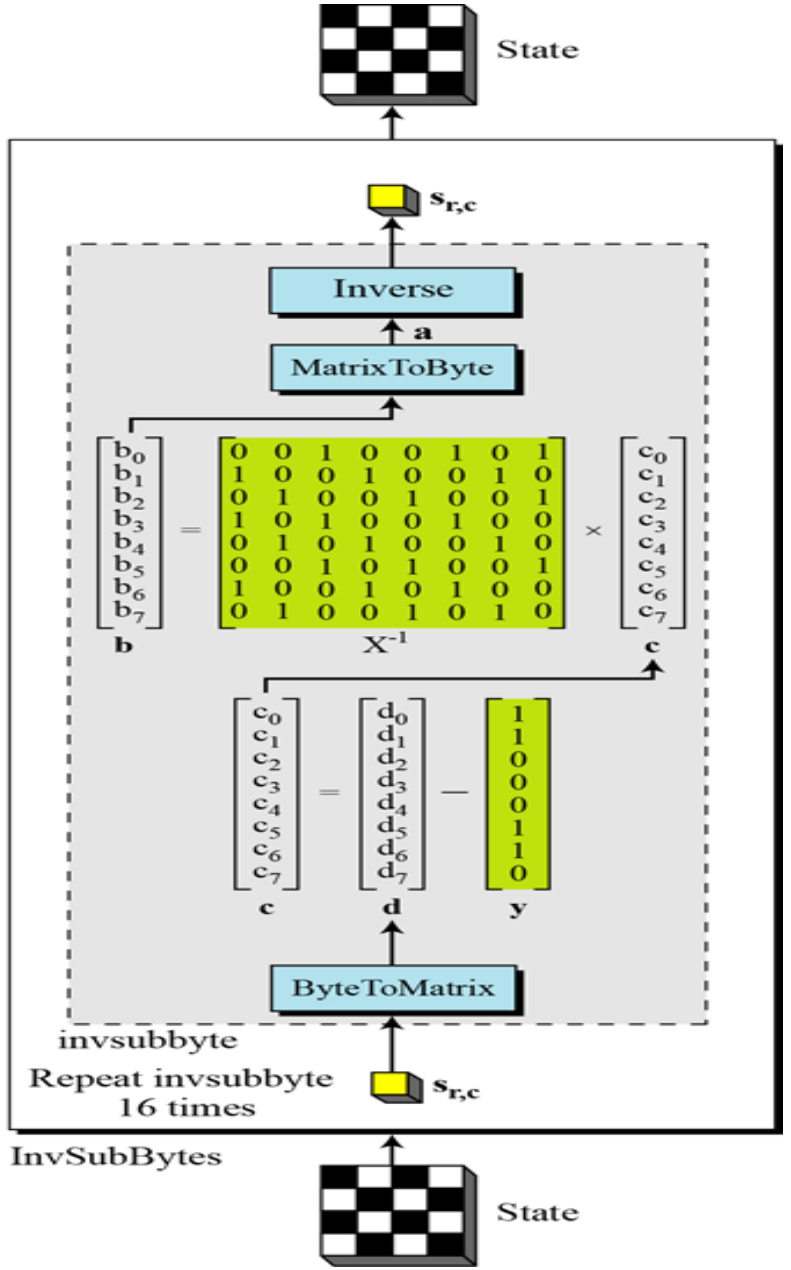
- state is transformed using the SubBytes transformation.
- InvSubBytes transformation creates the original one.
- If the two bytes have the same values, their transformation is also the same.



Substitution

- *SubBytes and InvSubBytes processes*





In X Matrix, In last column last bit is 1.

Example

- Let us show how the byte 0C is transformed to FE by subbyte routine and transformed back to 0C by the invsubbyte routine.

1. *subbyte:*
   a. The multiplicative inverse of 0C in GF(2^8) field is B0, which means **b** is (10110000).
   b. Multiplying matrix **X** by this matrix results in **c** = (10011101)
   c. The result of XOR operation is **d** = (11111110), which is FE in hexadecimal.

2. *invsubbyte:*
   a. The result of XOR operation is **c** = (10011101)
   b. The result of multiplying by matrix $\mathbf{X}^{-1}$ is (11010000) or B0
   c. The multiplicative inverse of B0 is 0C.

Example

- Multiplying matrix X by given matrix:
- Matrix
- 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

Input: b is 10110000

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row0 = 1 0 0 0 1 1 1 1
Bit 0 = 0 0 0 0 1 1 0 1 = 1 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row1 = 1 1 0 0 0 1 1 1
Bit 1 = 0 0 0 0 0 1 0 1 = 0 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row2 = 1 1 1 0 0 0 1 1
Bit 2 = 0 0 0 0 0 0 0 1 = 1 (XOR)

Output: c is 1001110**1 (LSB first)**

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row3 = 1 1 1 1 0 0 0 1
Bit 3 = 0 0 0 0 0 0 0 1 = 1 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row4 = 1 1 1 1 1 0 0 0
Bit 4 = 0 0 0 0 1 0 0 0 = 1 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row5 = 0 1 1 1 1 1 0 0
Bit 5 = 0 0 0 0 1 1 0 0 = 0 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row6 = 0 0 1 1 1 1 1 0
Bit 6 = 0 0 0 0 1 1 0 0 = 0 (XOR)

Input = 0 0 0 0 1 1 0 1 (LSB First)
Row7 = 0 0 0 1 1 1 1 1
Bit 7 = 0 0 0 0 1 1 0 1 = 1 (XOR)

Example

- Multiplication Output: c is 10011101
- Next Step XOR : 11000110 and c
- 11000110 and 10111001 (LSB first) – 01111111 – Output – 11111110 (LSB first) – FE in hexadecimal

Substitution

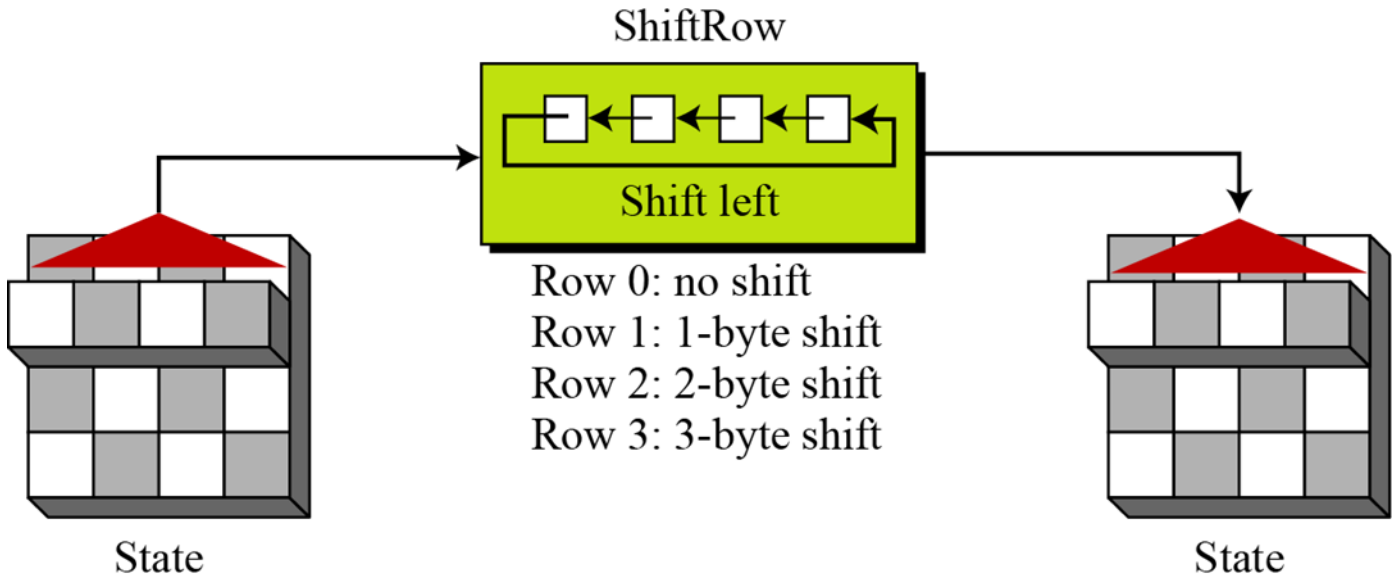
- Transformation Using the GF(2^8) Field
- AES also defines the transformation algebraically using the GF(2^8) field with the irreducible polynomials ($x^8 + x^4 + x^3 + x + 1$)

$$\text{subbyte:} \quad \rightarrow \quad \mathbf{d} = \mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y}$$
$$\text{invsubbyte:} \quad \rightarrow \quad [\mathbf{X}^{-1}(\mathbf{d} \oplus \mathbf{y})]^{-1} = [\mathbf{X}^{-1}(\mathbf{X}\,(s_{r,c})^{-1} \oplus \mathbf{y} \oplus \mathbf{y})]^{-1} = [(s_{r,c})^{-1}]^{-1} = s_{r,c}$$

- The SubBytes and InvSubBytes transformations are inverse of each other.

Permutation

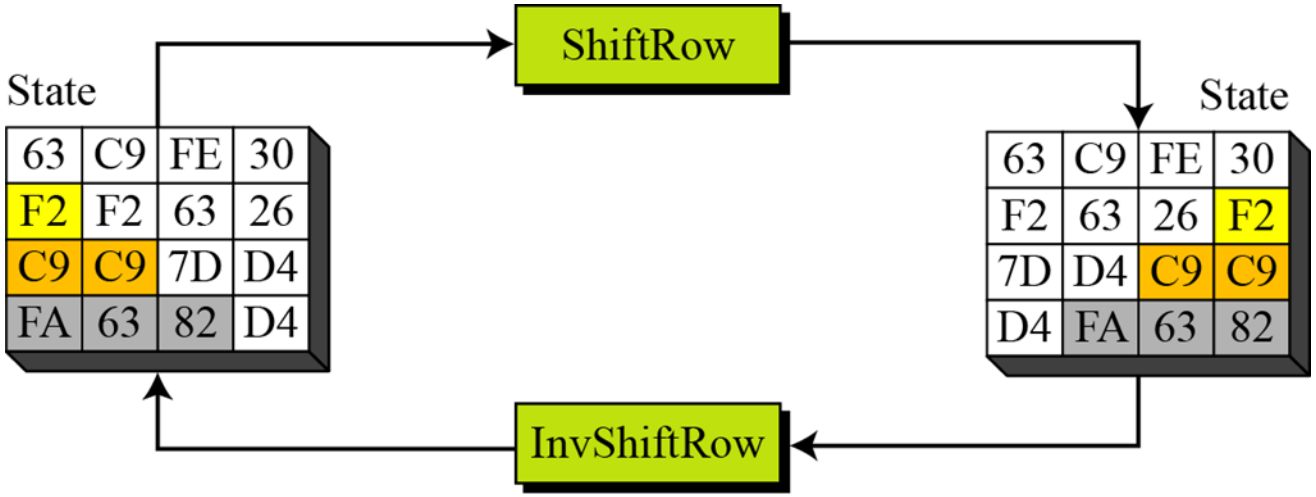
- In encryption, transformation is called **Shift Rows**
- A circular byte shift in each
- decryption inverts using shifts to right



- since state is processed by columns, this step permutes bytes between the columns

Permutation

- Example



Shift Row Example

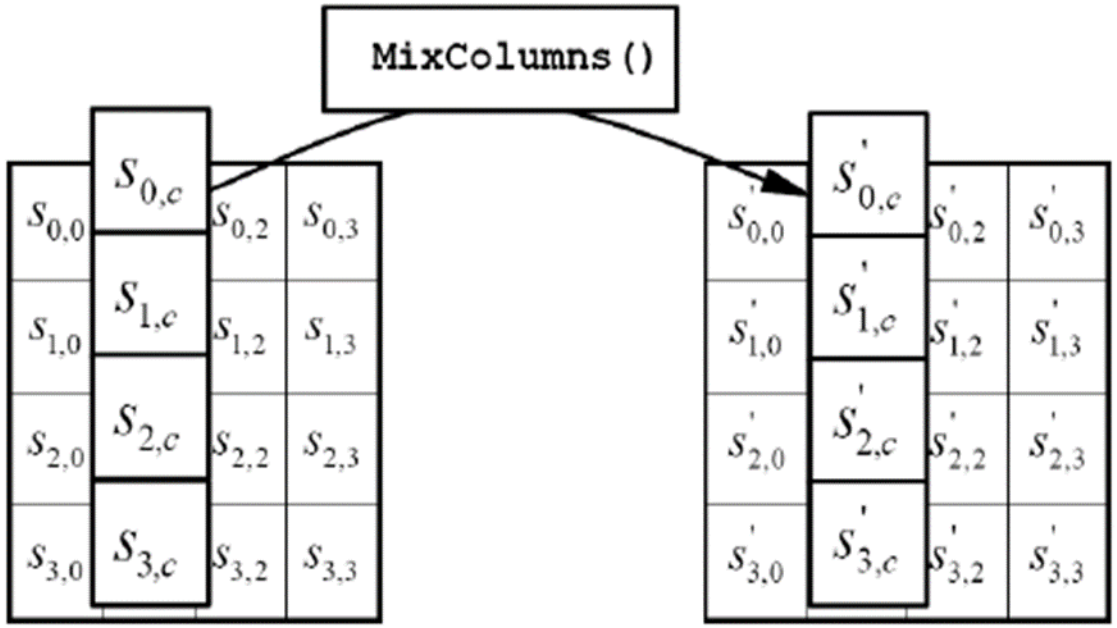
d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

Input for Shift Row

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

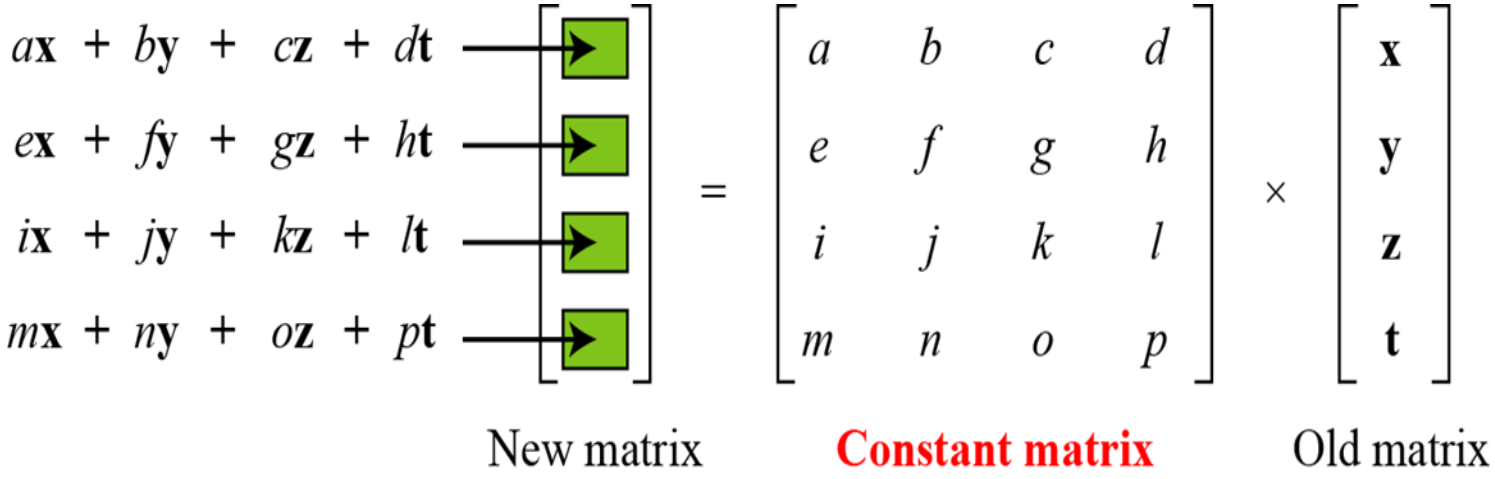
Output of Shift Row

Mix Columns Transformation

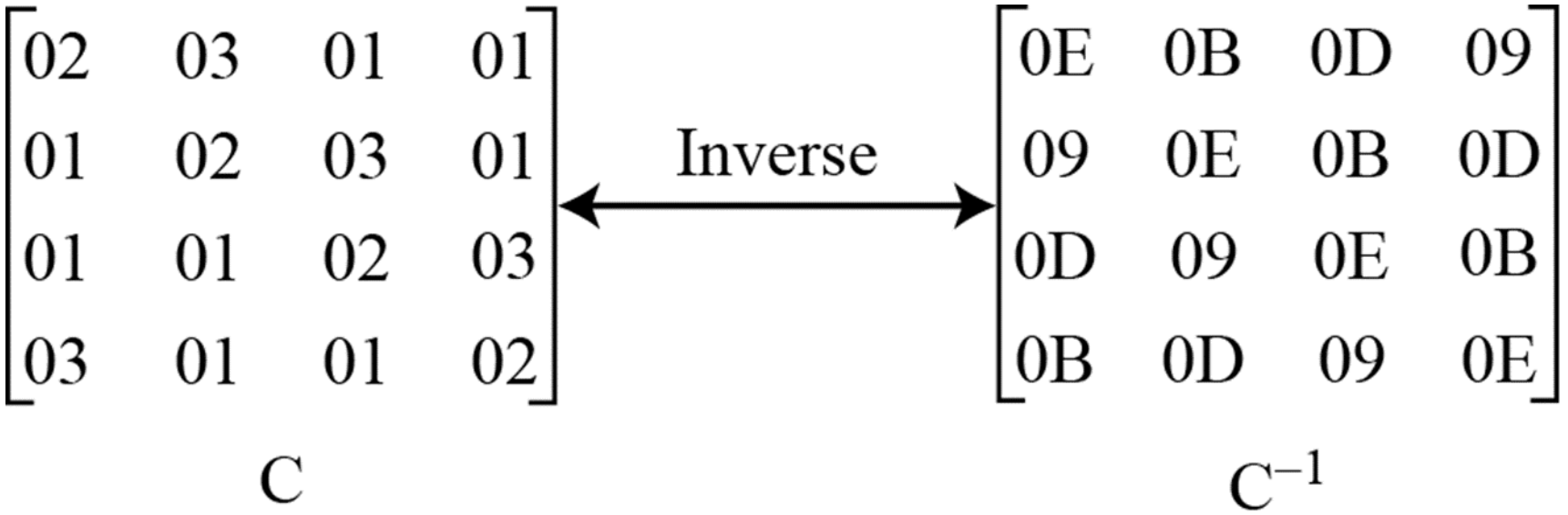


Mix Columns Transformation

- An interbyte transformation changes the bits inside a byte, based on the bits inside the neighboring bytes.
  - We need to mix bytes to provide diffusion at the bit level.

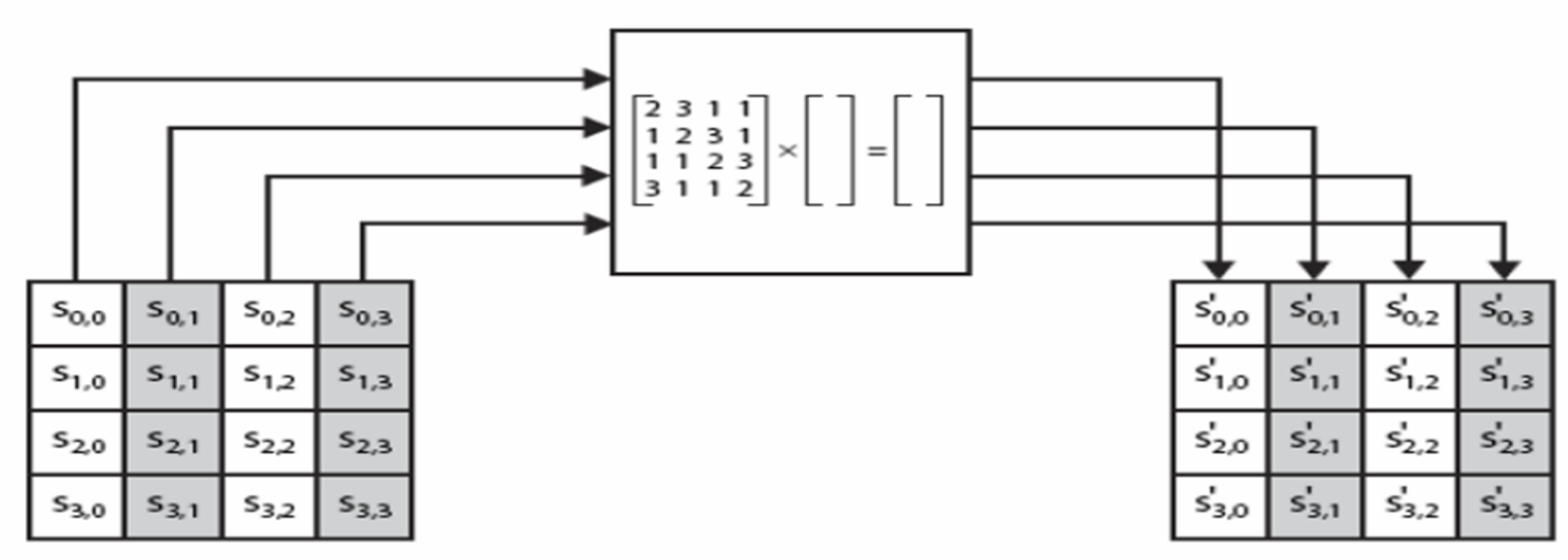


Mix Columns Transformation



Constant matrices used by MixColumns and InvMixColumns

Mix Columns Transformation



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

Mix Columns Transformation

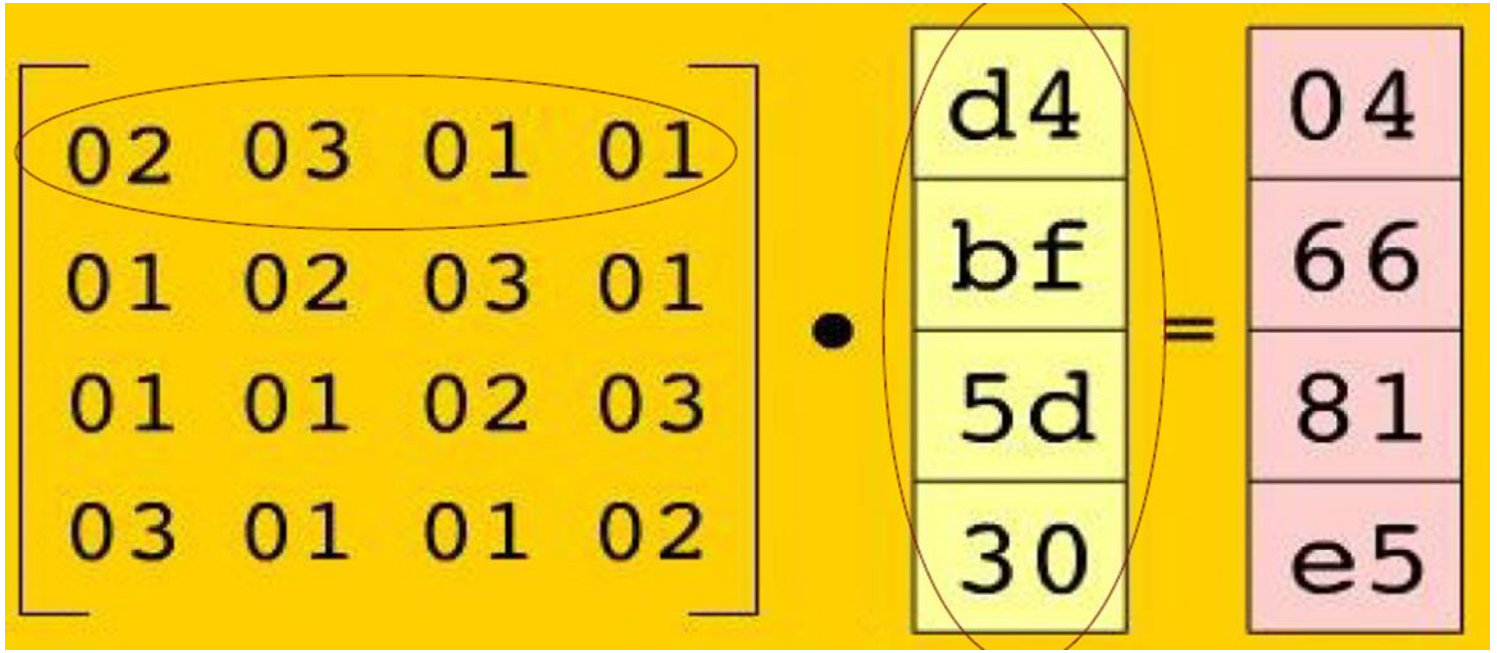
$$
\begin{aligned}
s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\
s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})
\end{aligned}
$$

Mix Columns Transformation - Example

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

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

Mix Columns Transformation - Example



Mix Columns Transformation - Example

- r0 = {02.d4} + {03.bf} + {01.5d} + {01.30}

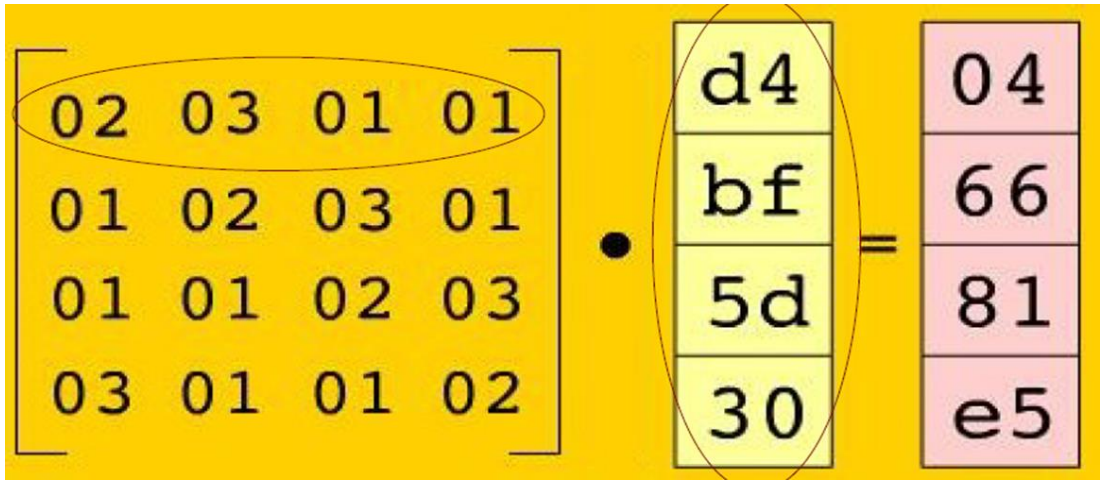
- {02.d4}

- Multiplication of a value by x (ie. by 02) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (00011011) if the leftmost bit of the original value (before the shift) is 1.

Mix Columns Transformation - Example

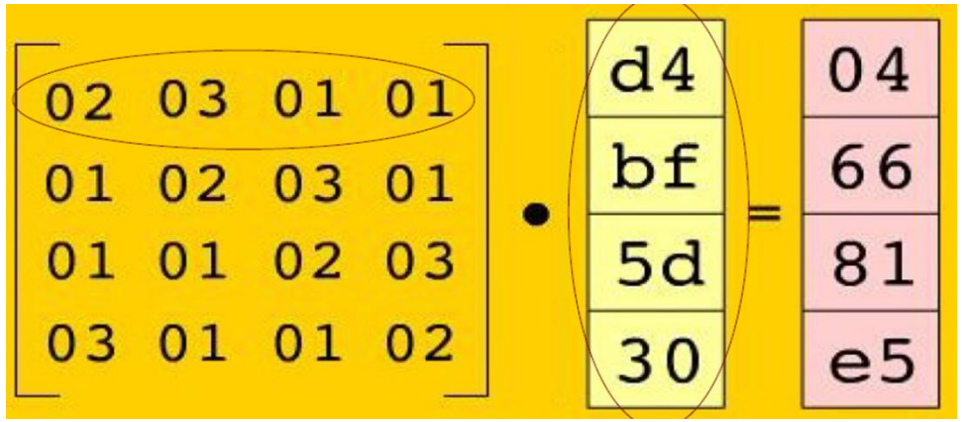
- d4 = 1101 0100

- {d4}.{02} = 1101 0100 << 1 (<< is left shift, 1 is the number of shift done, pad on with 0's)
- = 1010 1000 XOR 0001 1011 (because the leftmost is 1 before shift)
- = **1011 0011**
- **=Ans1**



Mix Columns Transformation - Example

- {03.bf}



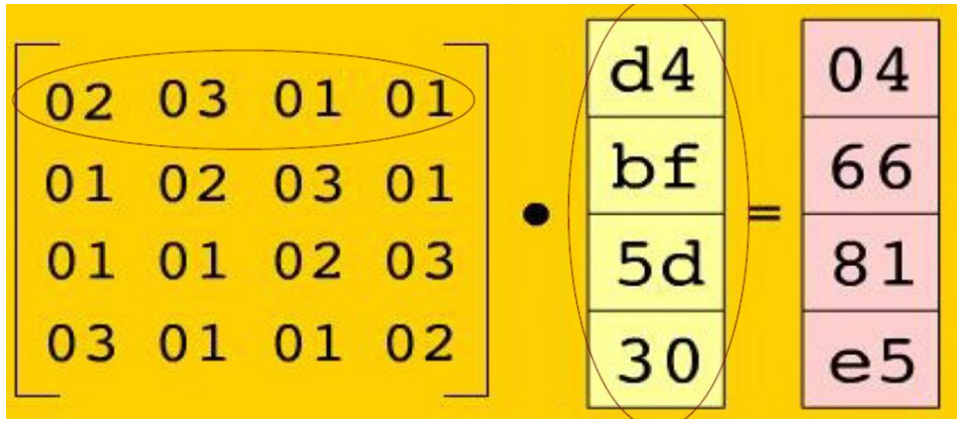
- bf = 1011 1111
- bf XOR ({02} . {bf})
- 1011 1111 XOR (0111 1110 [left shift by 1 bit])

- 1011 1111 XOR (0111 1110 XOR 0001 1011 (because the leftmost is a 1 before shift))

- 1011 1111 XOR 0111 1110 XOR 0001 1011
- **1101 1010**
- **=Ans2**

Mix Columns Transformation - Example

- 5d = 0101 1101
- 30 = 0011 0000



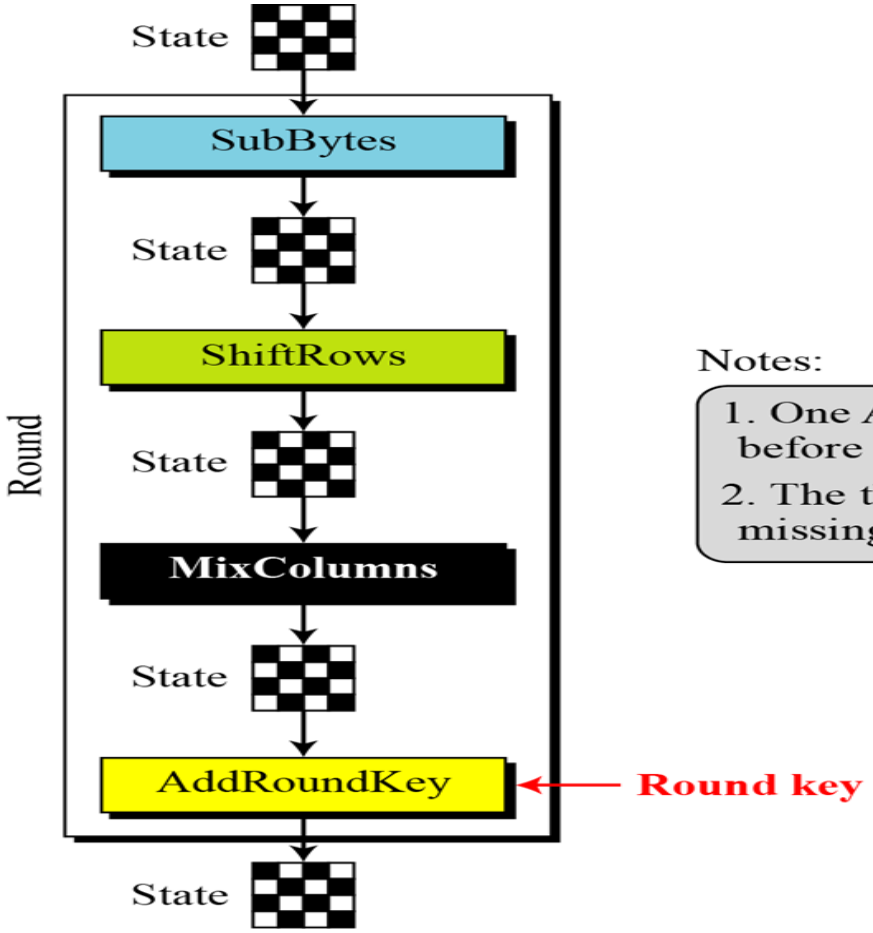
- r0 = {02.d4} + {03.bf} + {01.5d} + {01.30}

= **1011 0011 XOR 1101 1010 XOR 0101 1101 XOR 0011 0000**

= **0000 0100**

= **04 (in Hex)**

Structure of Each Round



Notes:

1. One AddRoundKey is applied before the first round.
2. The third transformation is missing in the last round.

- AES Transformations:

1. Substitution
2. Permutation
3. Mixing
4. Key-adding

AddRoundKey

- AddRoundKey proceeds one column at a time.
- AddRoundKey adds a round key word with each state column matrix; the operation in AddRoundKey is matrix addition.
- In the AddRoundKey() transformation, a Round Key is added to the State by a simple bitwise XOR operation.

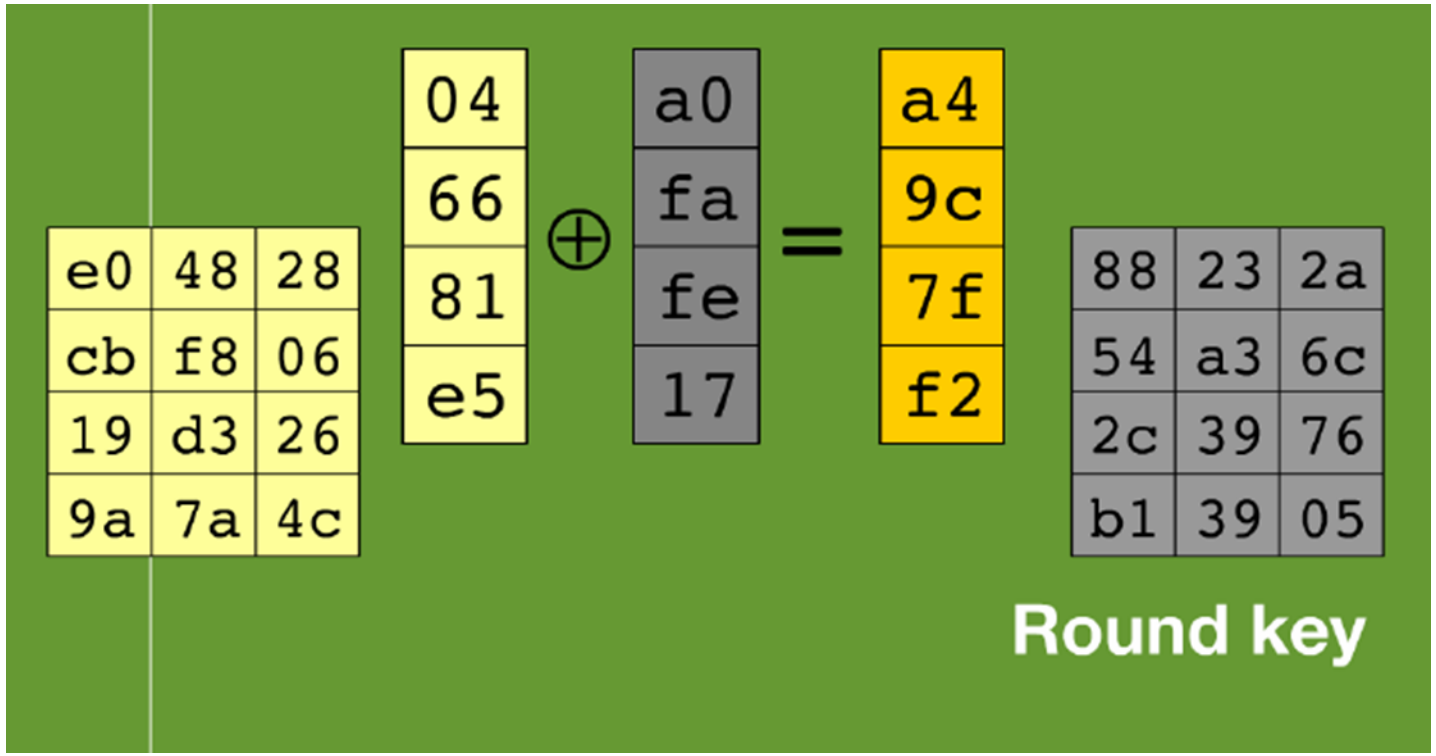
AddRoundKey

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

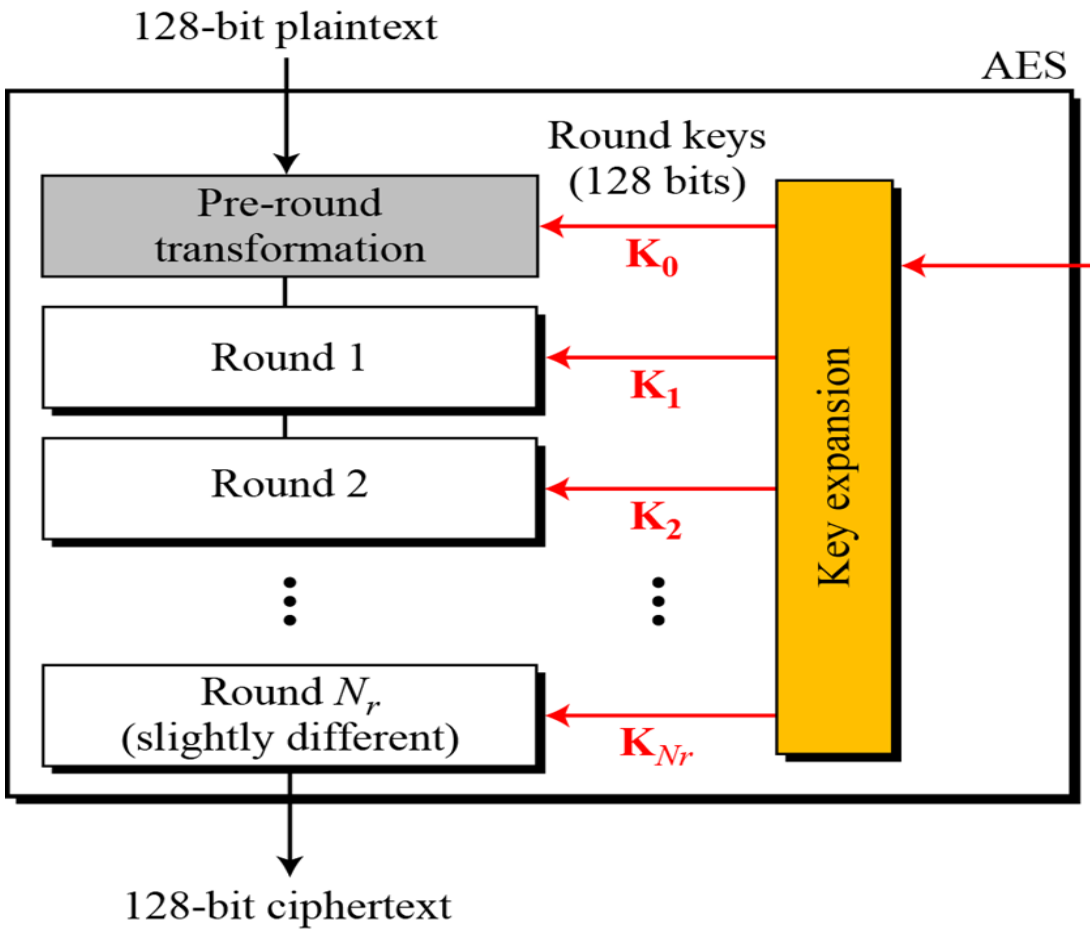
a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

Round Key

AddRoundKey



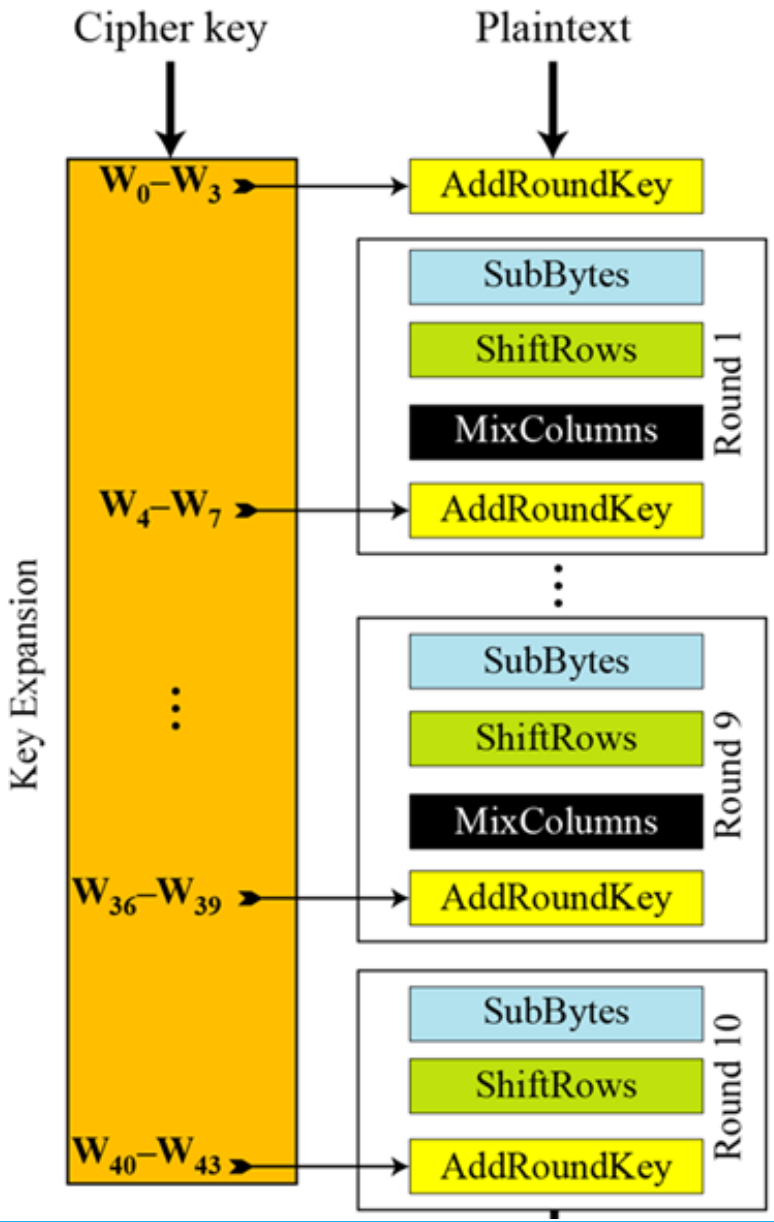
AES Key Expansion



Cipher key
(128, 192, or 256 bits)

Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size


Cipher key
Plaintext
Key Expansion
W_0–W_3
W_4–W_7
W_{36}–W_{39}
W_{40}–W_{43}
AddRoundKey
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round 1
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round 9
SubBytes
ShiftRows
AddRoundKey
Round 10

AES – Cipher Text Generation

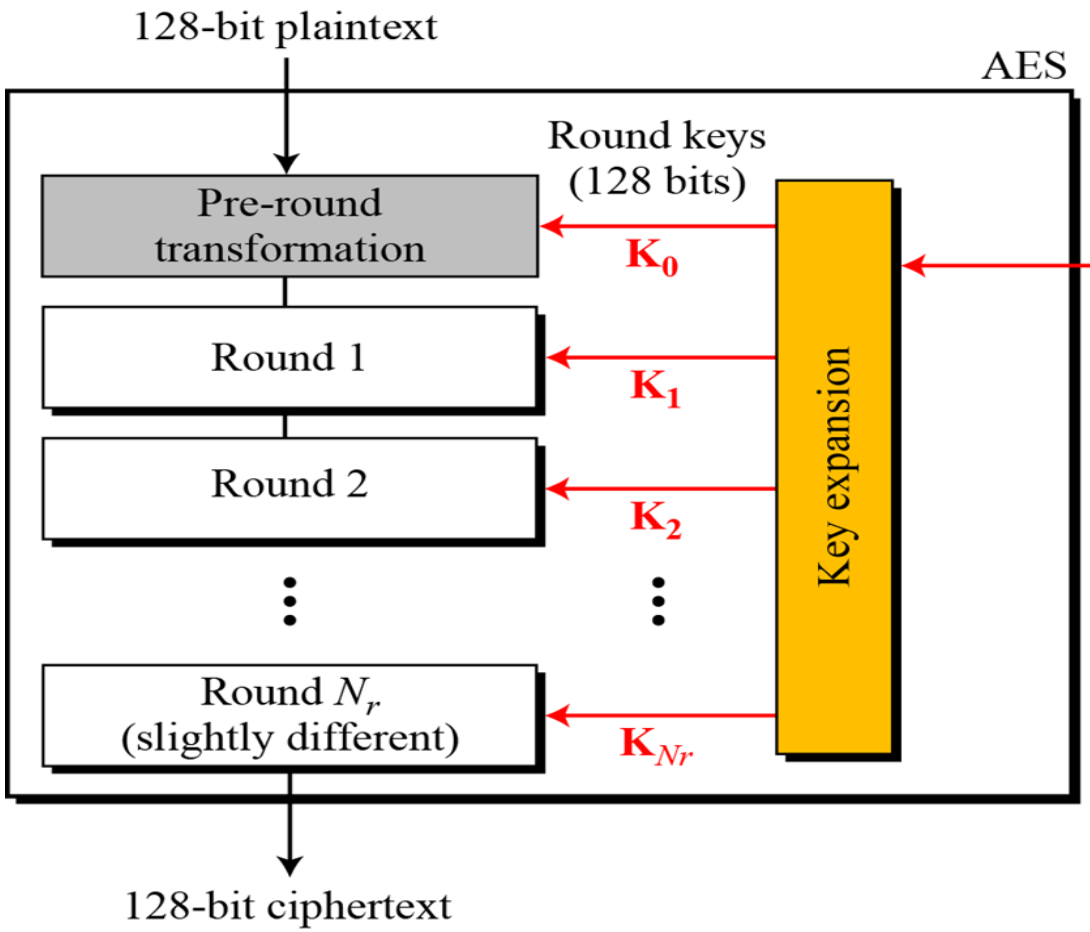
AES – Cipher Text Generation

AES Key Expansion

- To create round keys for each round, AES uses a key-expansion process.

- If the number of rounds is Nr , the key expansion routine creates Nr + 1 128-bit round keys from one single 128-bit cipher key.

AES Key Expansion



Cipher key
(128, 192, or 256 bits)

Nr	Key size
10	128
12	192
14	256

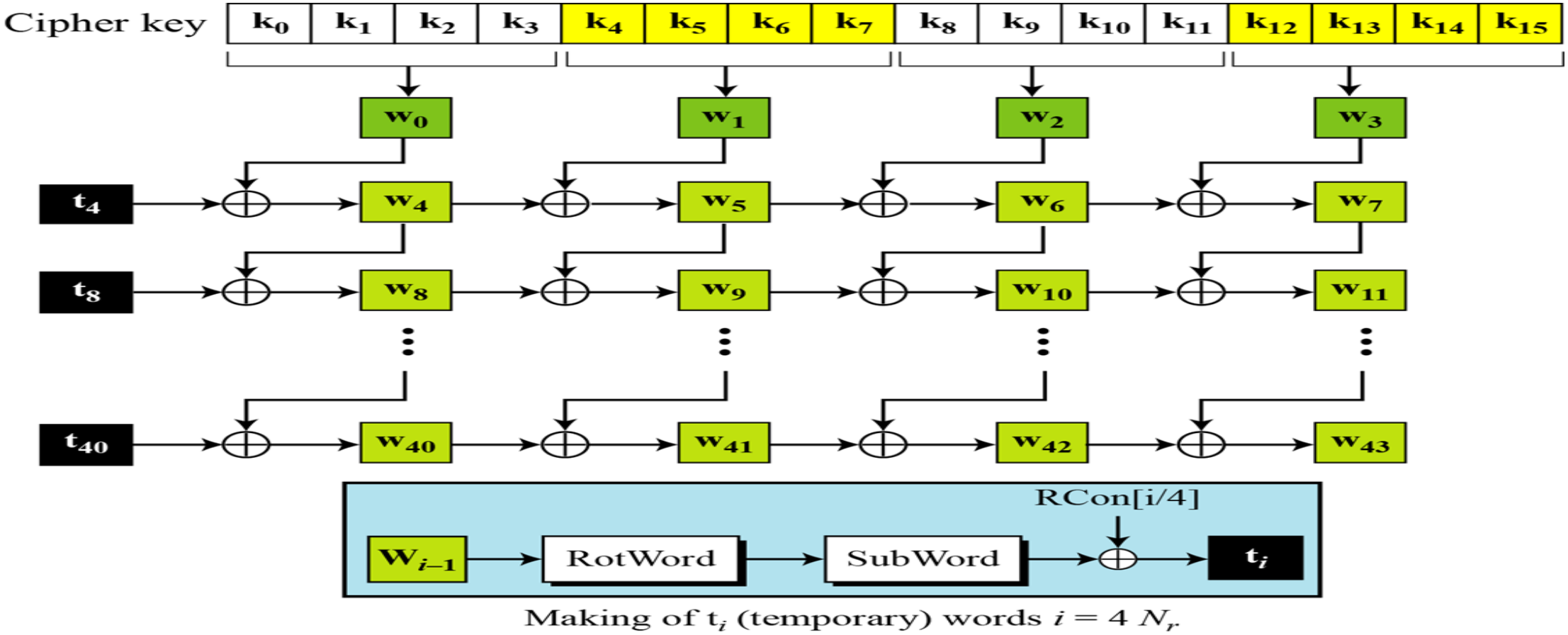
Relationship between number of rounds and cipher key size

AES Key Expansion

Table 7.3 *Words for each round*

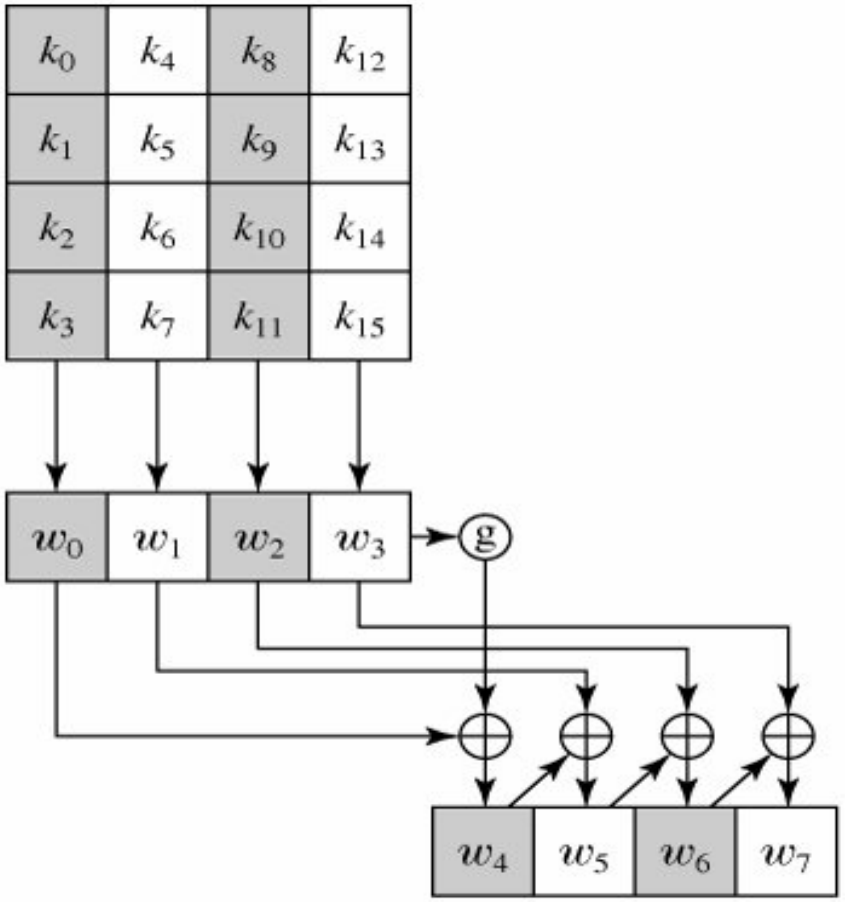
Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

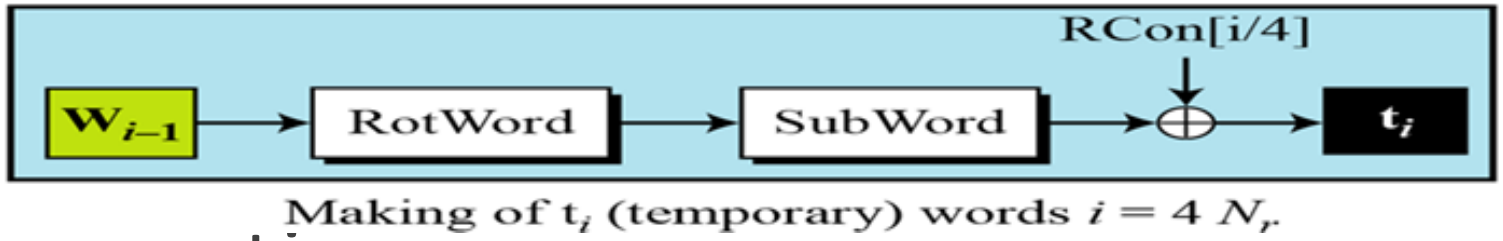
AES – Key Expansion



Making of t_i (temporary) words $i = 4\ N_r$

AES Key Expansion





Making of t_i (temporary) words $i = 4\ N_r$.

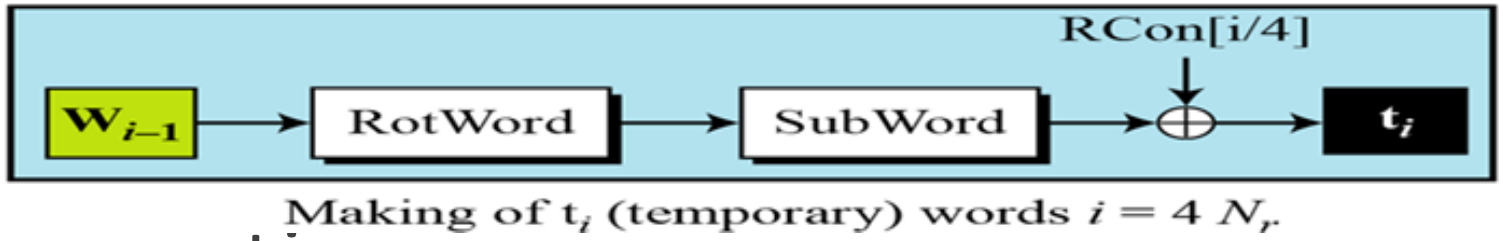
AES Key schedule generation

2b	28	ab	09												
7e	ae	f7	cf												
15	d2	15	4f												
16	a6	88	3c												

. . .

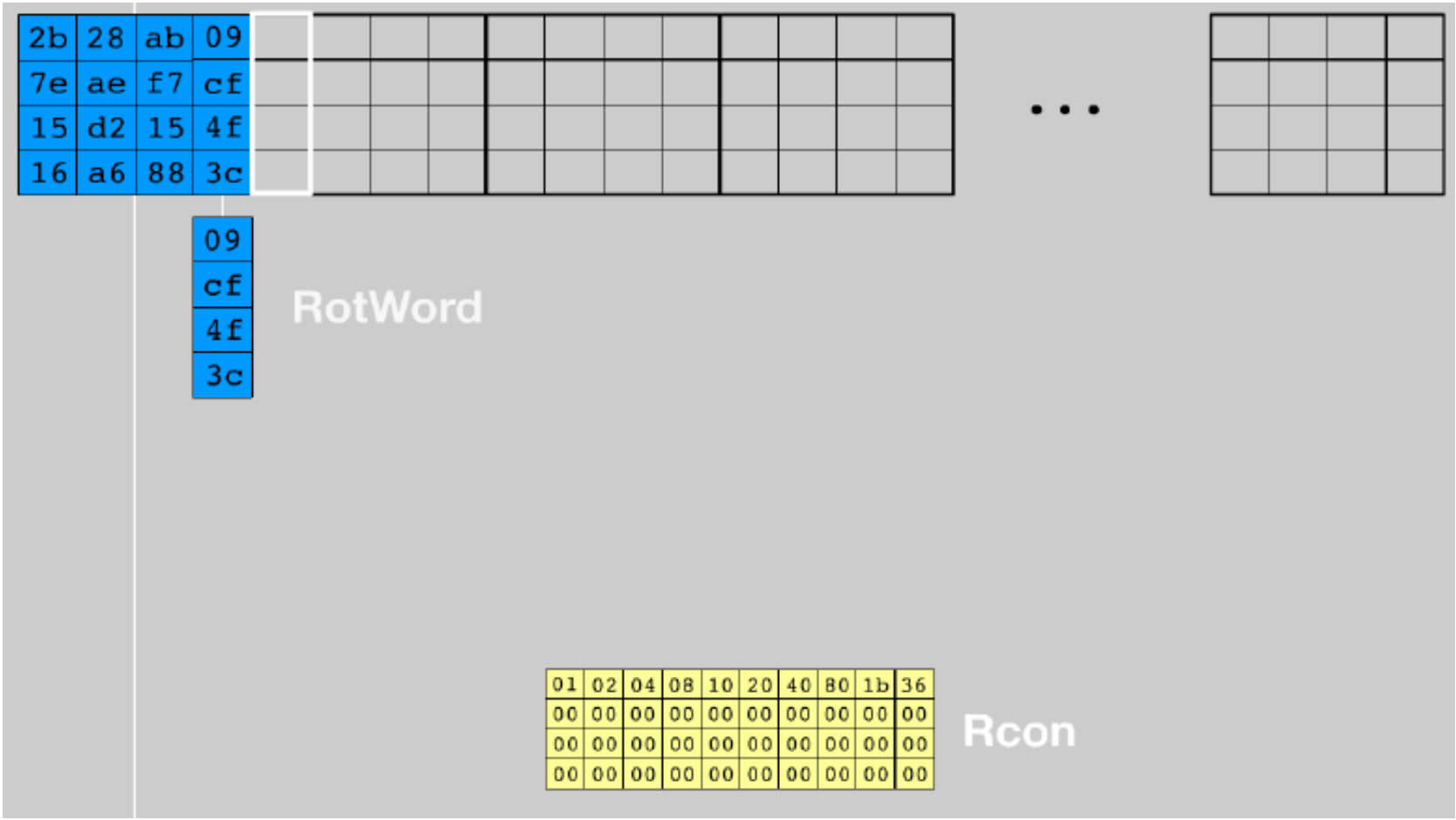
01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

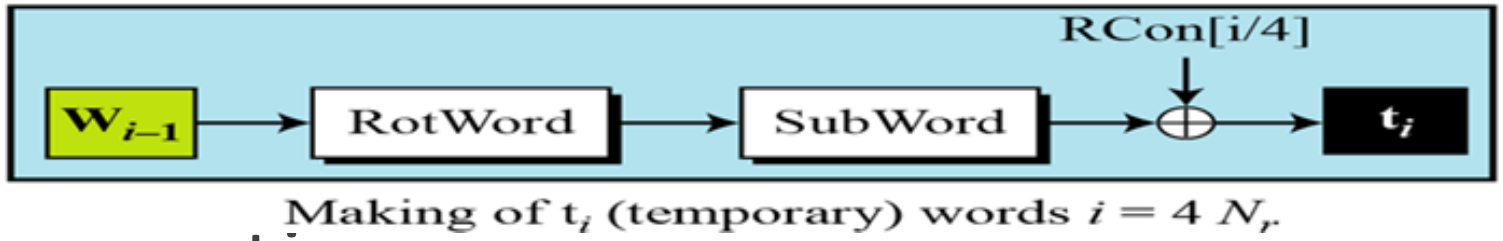
Rcon



Making of t_i (temporary) words $i = 4\ N_r$.

AES Key schedule generation





Making of t_i (temporary) words $i = 4\ N_r$.

AES Key schedule generation

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

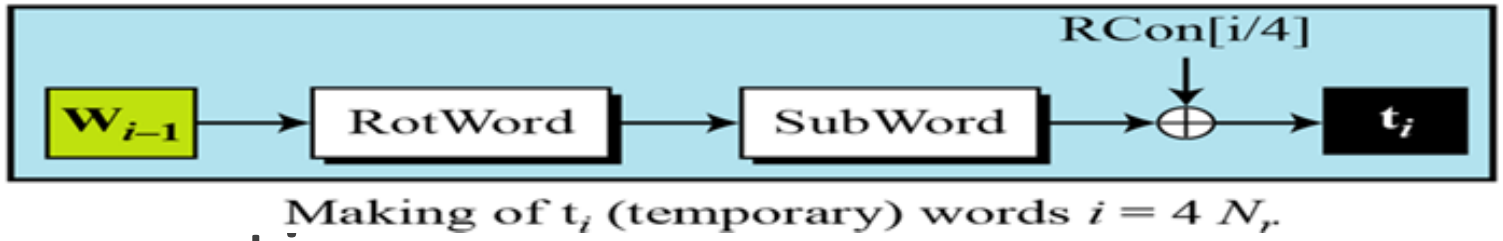
cf
4f
3c
09

SubBytes

S-BOX

01	02	04	08	10	20	40	80	1b	36	
00	00	00	00	00	00	00	00	00	00	
00	00	00	00	00	00	00	00	00	00	
00	00	00	00	00	00	00	00	00	00	

Rcon



Making of t_i (temporary) words $i = 4\ N_r$.

AES Key schedule generation

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

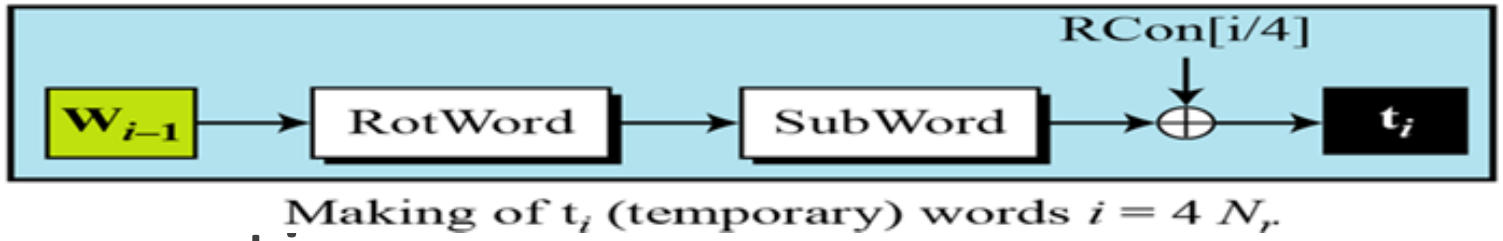
8a
84
3c
09

SubBytes

S-BOX

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon



Making of t_i (temporary) words $i = 4\ N_r$.

AES Key schedule generation

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

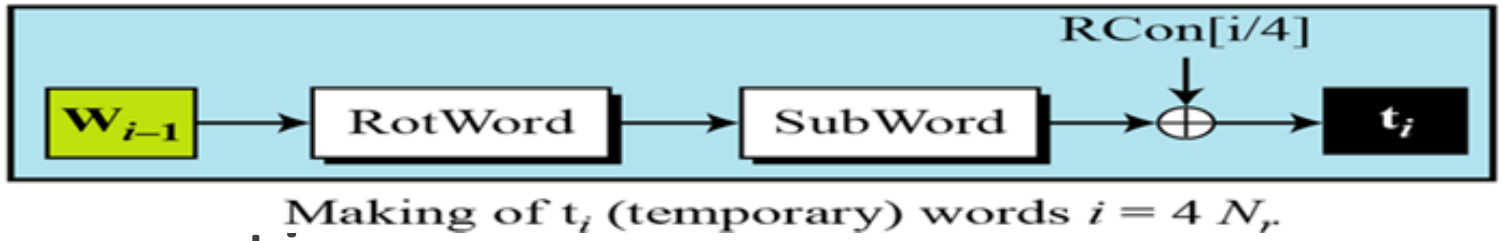
8a
84
eb
01

SubBytes

S-BOX

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon



Making of t_i (temporary) words $i = 4\ N_r$.

AES Key schedule generation

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

2b	⊕	8a	⊕	01
7e		84		00
15		eb		00
16		01		00

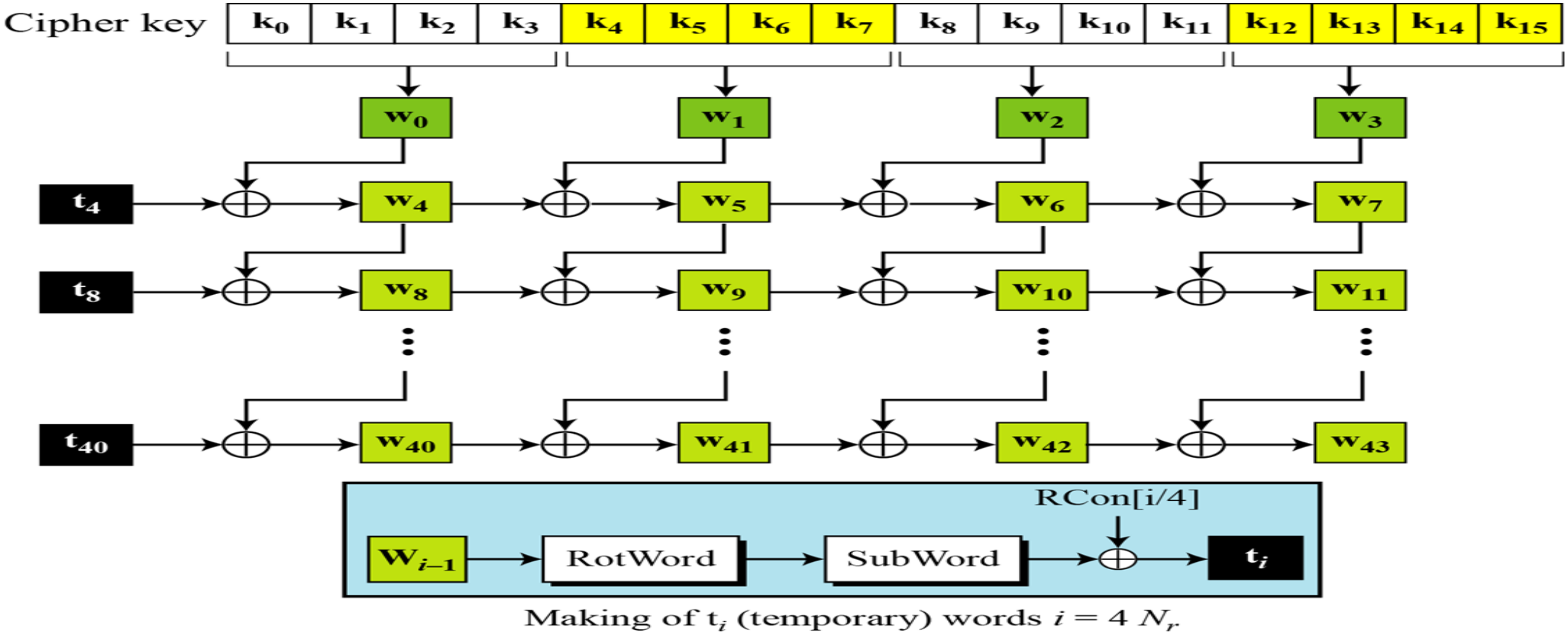
Rcon(4)

= a0 fa fe 17

02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00

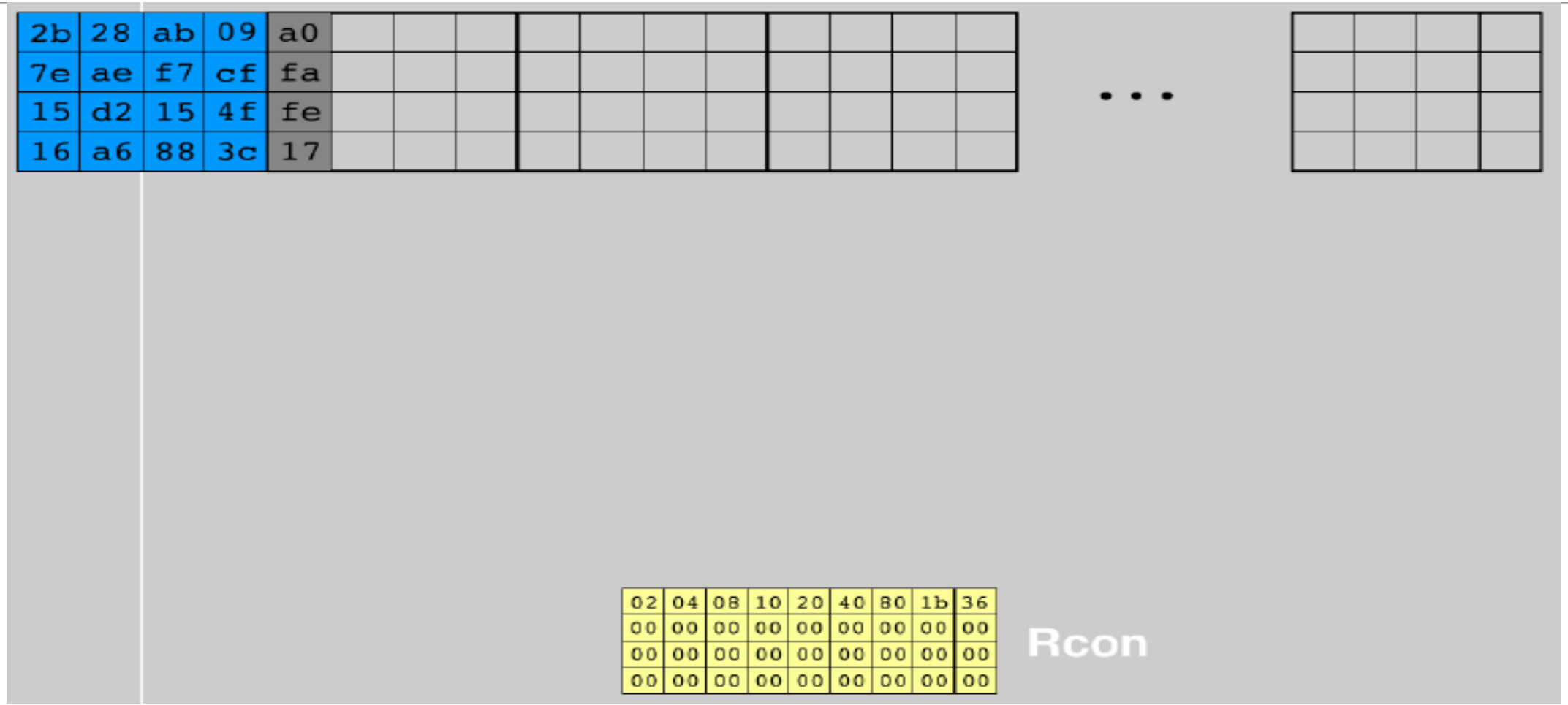
Rcon

AES – Key Expansion

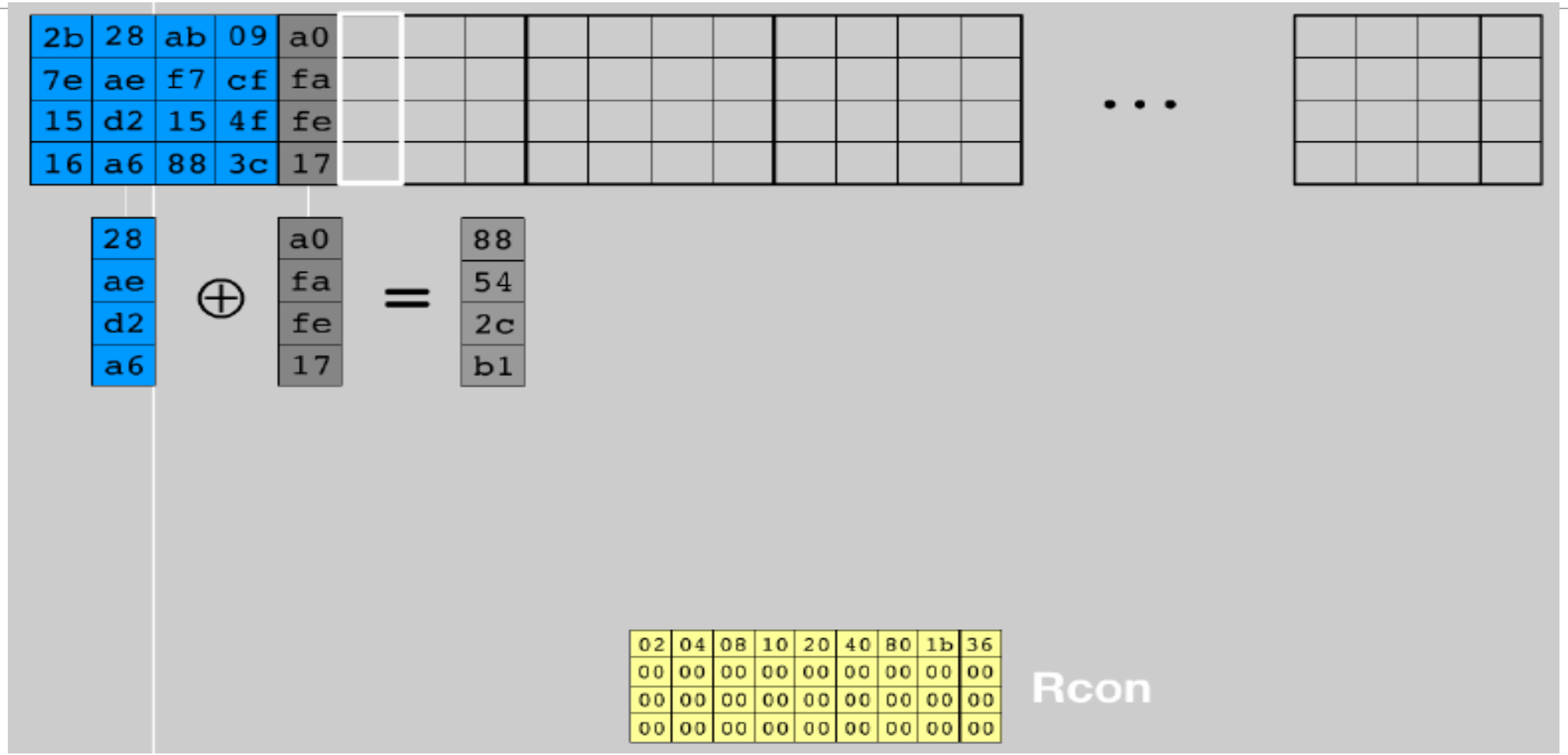


Making of t_i (temporary) words $i = 4\ N_r$

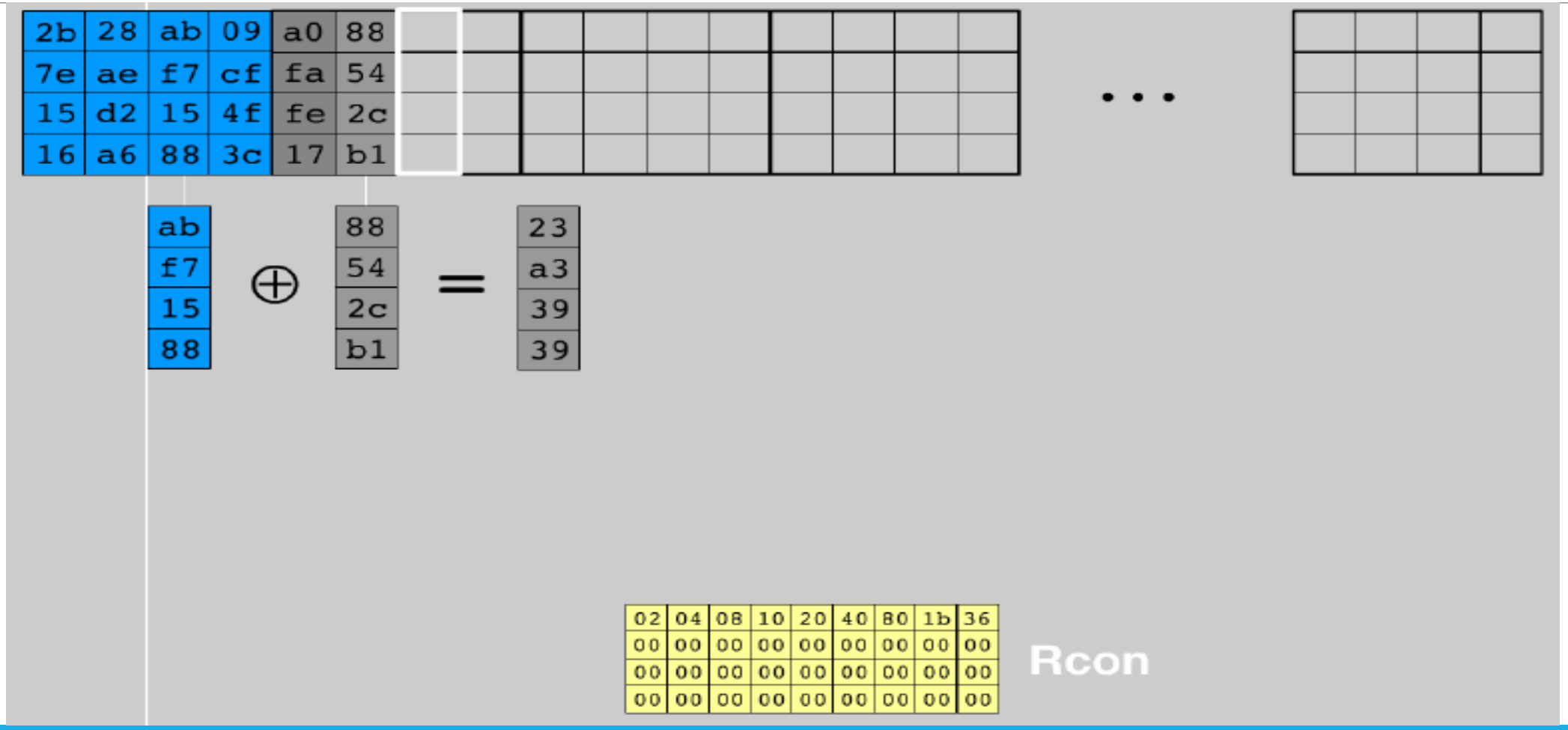
AES Key schedule generation



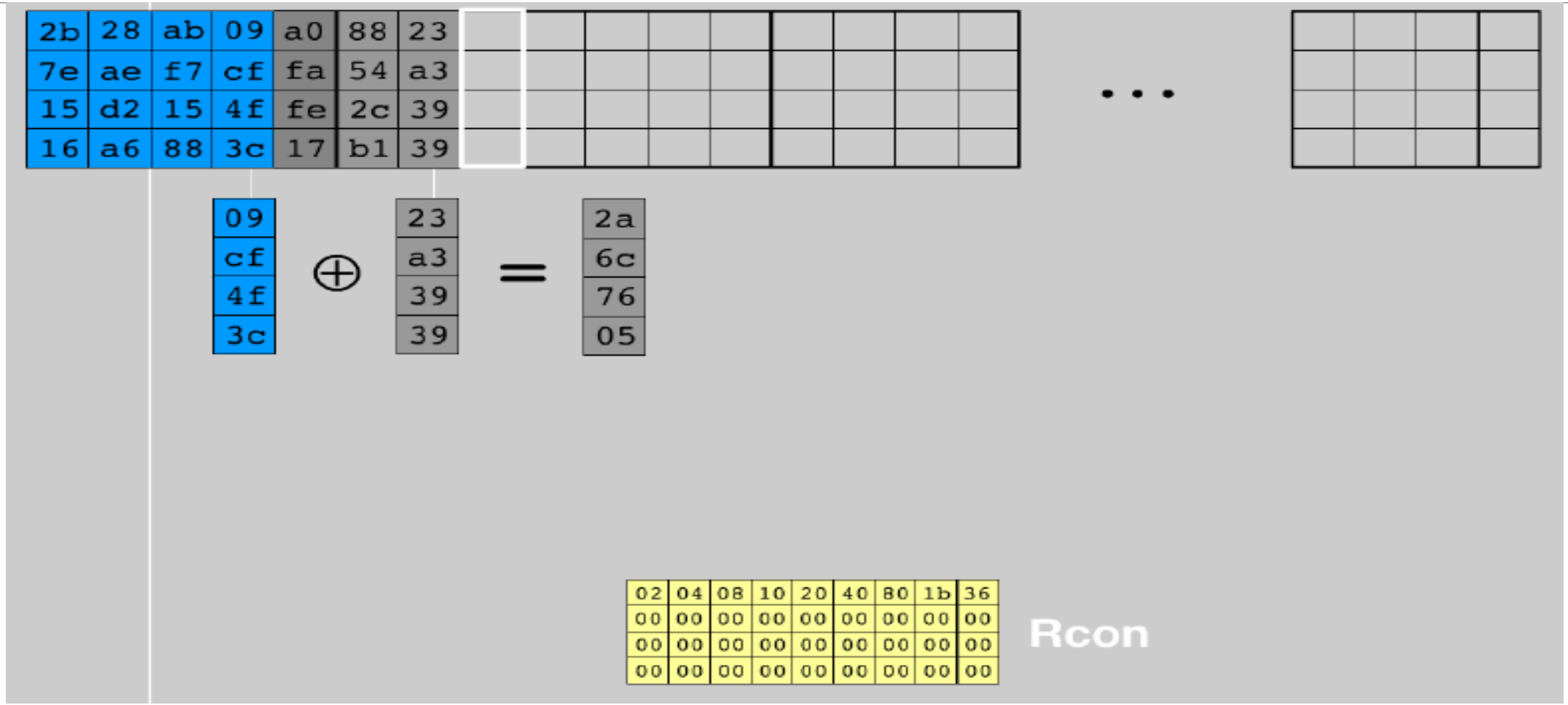
AES Key schedule generation



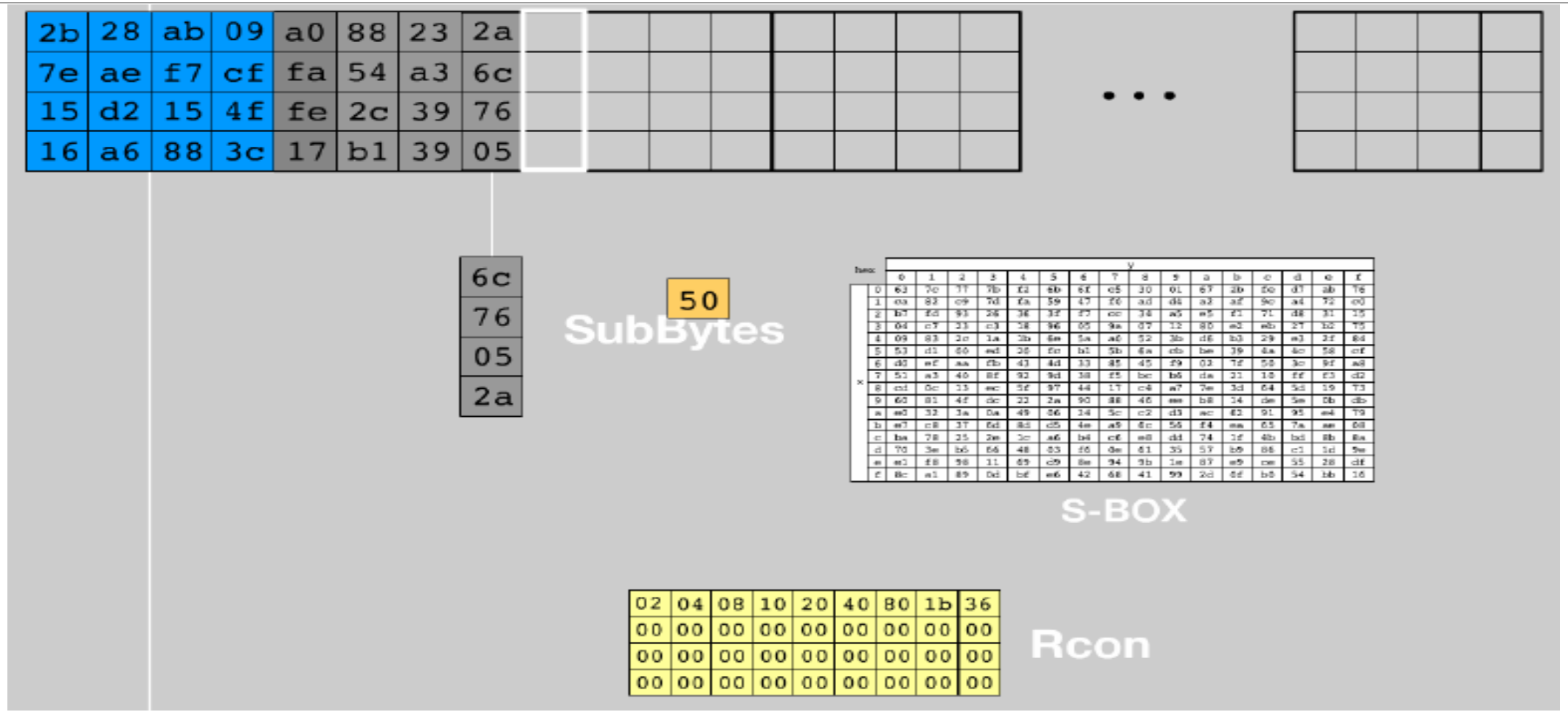
AES Key schedule generation



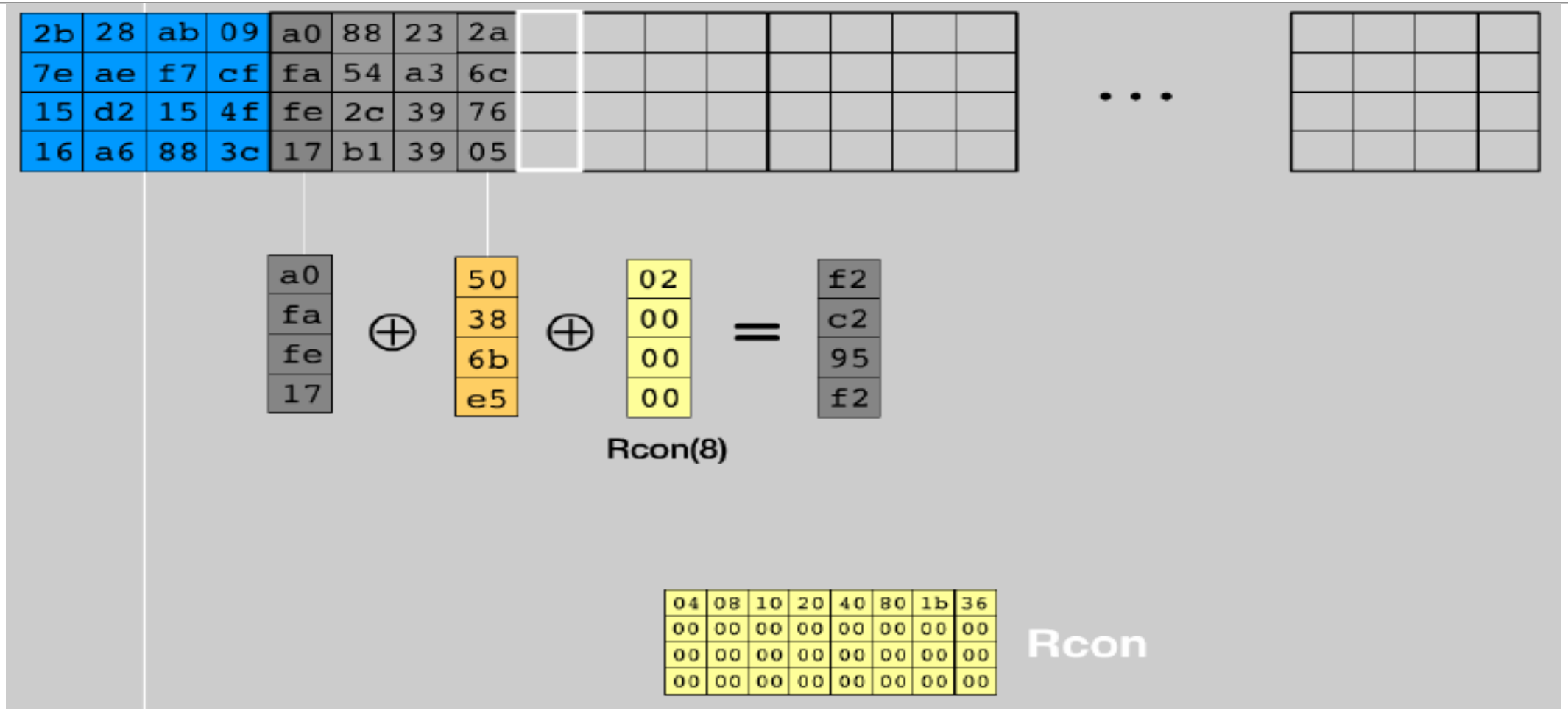
AES Key schedule generation



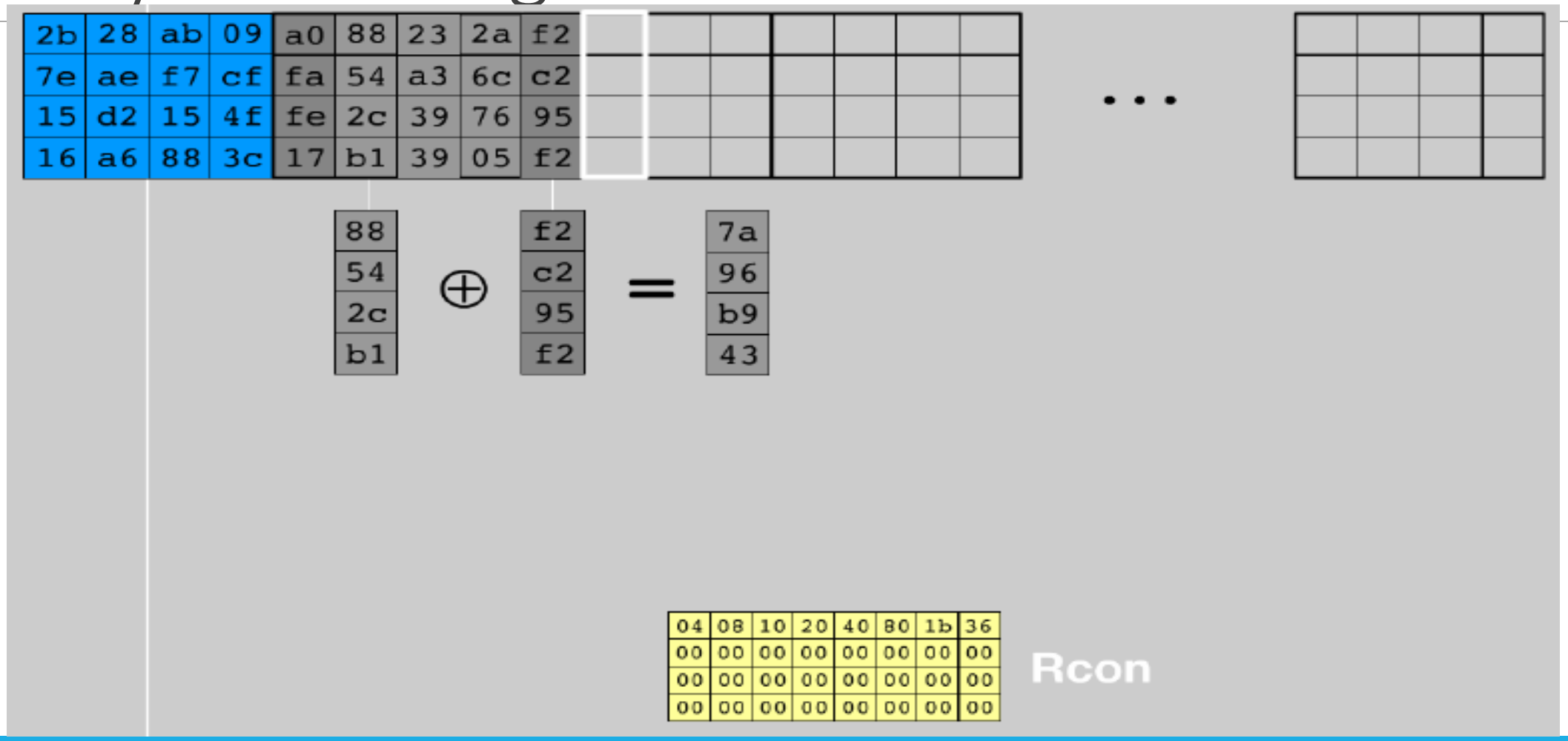
AES Key schedule generation



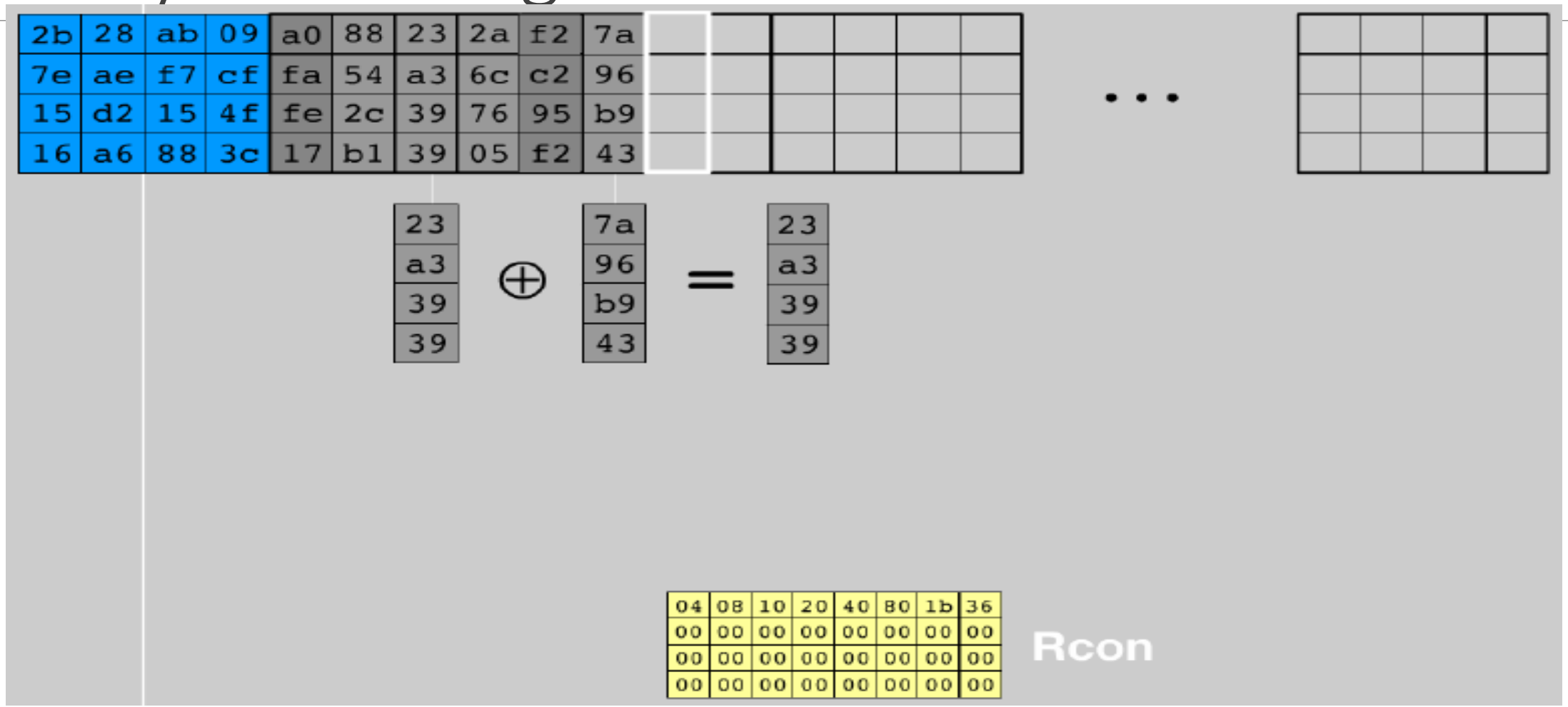
AES Key schedule generation



AES Key schedule generation



AES Key schedule generation

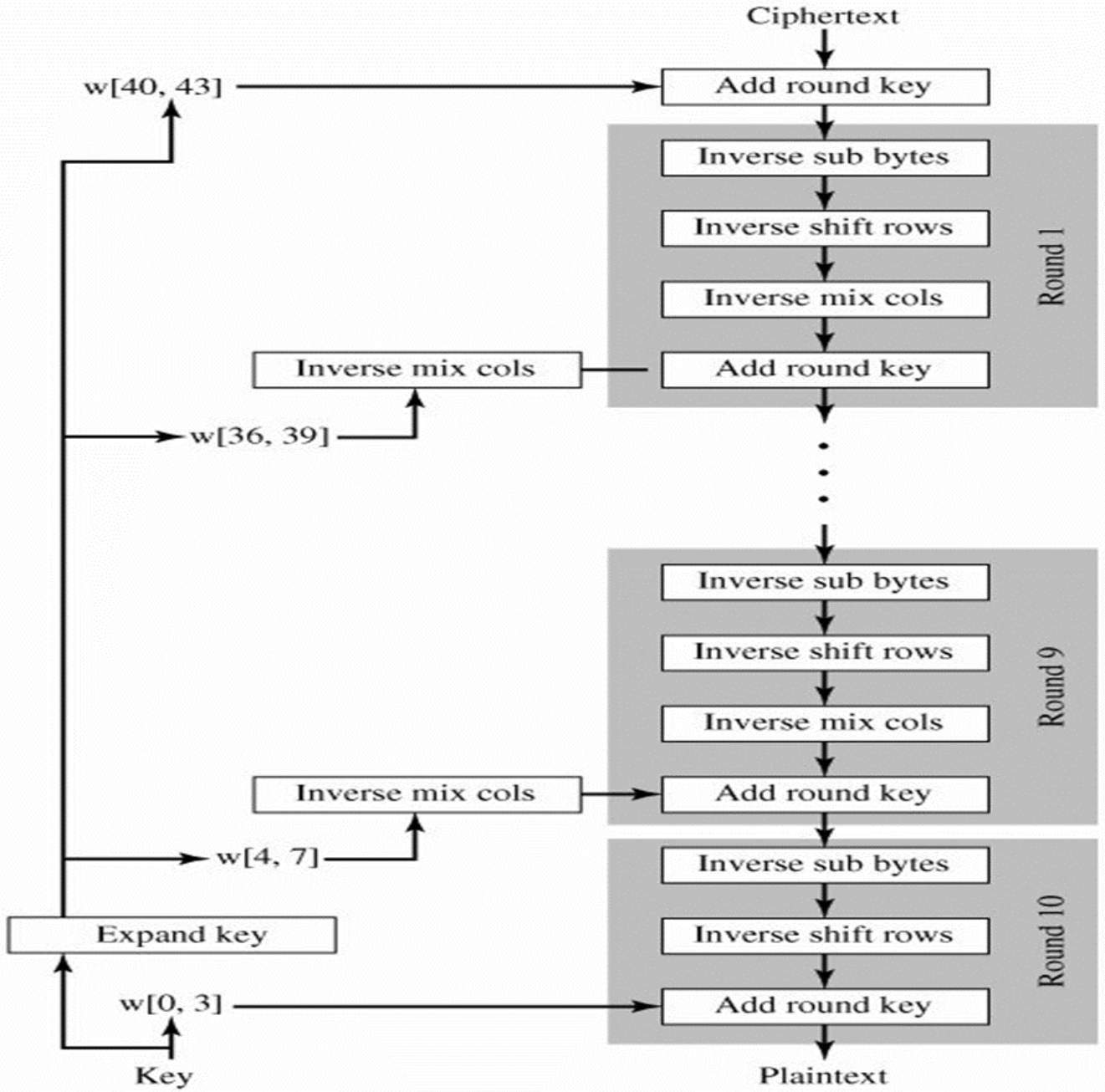


AES Key schedule generation

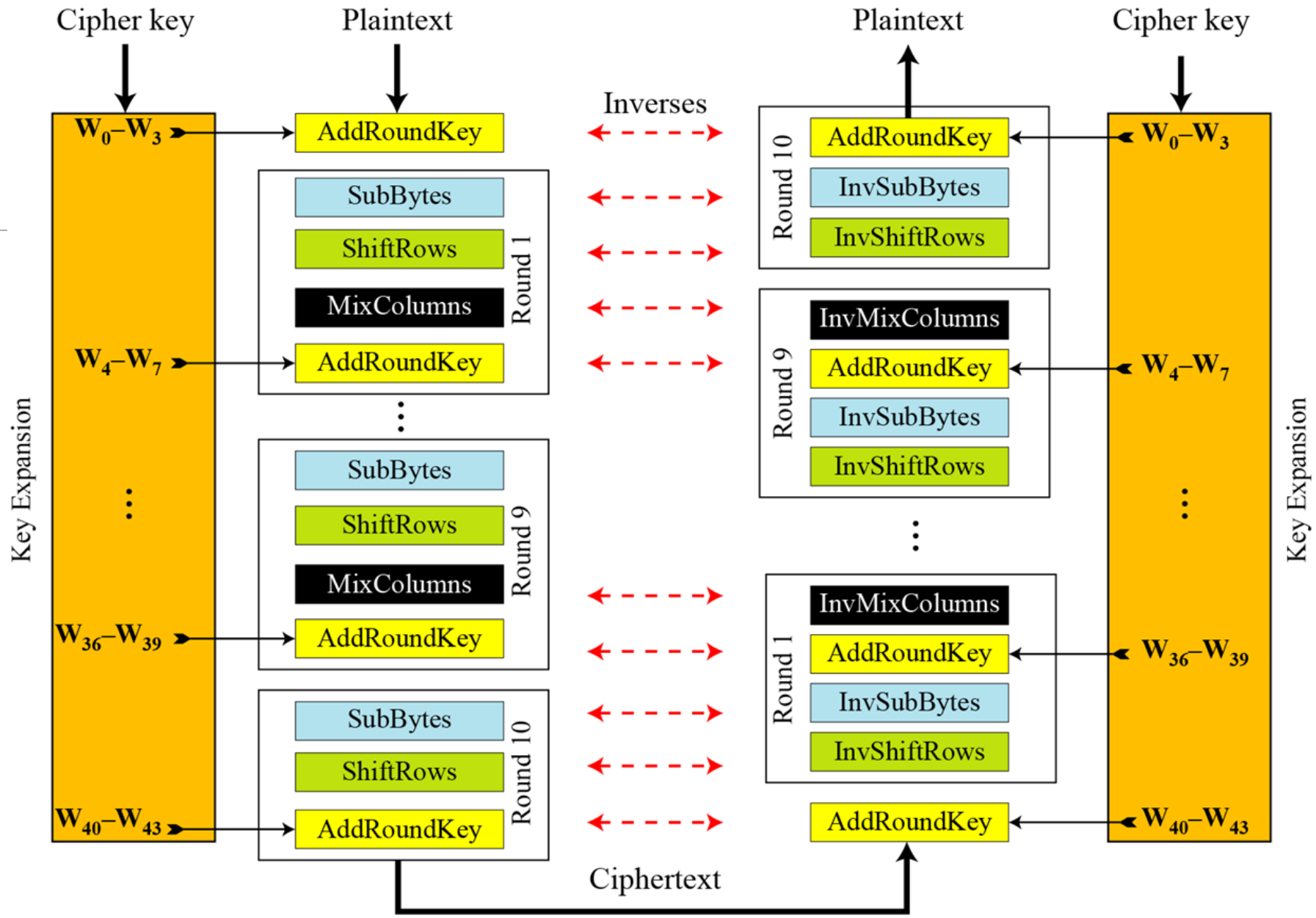
AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when swap byte substitution & shift rows
- swap mix columns & add (tweaked) round key

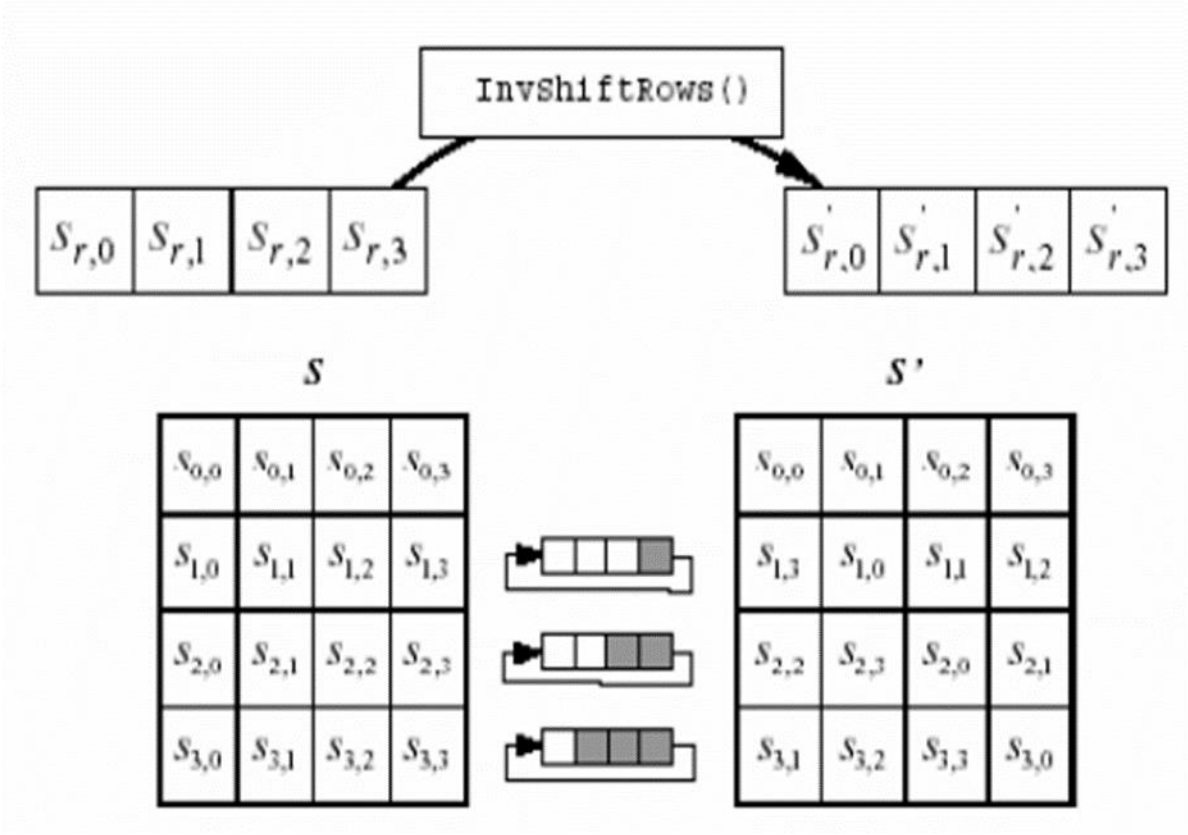
AES Decryption



AES Cipher


Cipher key
Plaintext
Plaintext
Cipher key
Inverses
W_0–W_3
AddRoundKey
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round 1
W_4–W_7
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round 9
W_{36}–W_{39}
SubBytes
ShiftRows
AddRoundKey
Round 10
W_{40}–W_{43}
Key Expansion
Round 10
AddRoundKey
InvSubBytes
InvShiftRows
Round 9
InvMixColumns
AddRoundKey
InvSubBytes
InvShiftRows
Round 1
InvMixColumns
AddRoundKey
InvSubBytes
InvShiftRows
AddRoundKey
W_0–W_3
W_4–W_7
W_{36}–W_{39}
W_{40}–W_{43}
Key Expansion
Ciphertext

InvShiftRows( )



Inverse S-box

(b) Inverse 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

Inverse Mix Column Transformation

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Proof : First matrix is inverse of matrix used for encryption .

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Analysis of AES

- The result of encryption when the plaintext is made of all 0s.

Plaintext:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Cipher Key:	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Ciphertext:	63	2C	D4	5E	5D	56	ED	B5	62	04	01	A0	AA	9C	2D	8D

Analysis of AES

- The avalanche effect.
  - Change only one bit in the plaintext and compare the results.
- changed only one bit in the last byte.
  - Changing a single bit in the plaintext has affected many bits in the ciphertext.

Plaintext 1:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Plaintext 2:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01
Ciphertext 1:	63	2C	D4	5E	5D	56	ED	B5	62	04	01	A0	AA	9C	2D	8D
Ciphertext 2:	26	F3	9B	BC	A1	9C	0F	B7	C7	2E	7E	30	63	92	73	13

Analysis of AES

- The effect of using a cipher key in which all bits are 0s.

Plaintext:	00	04	12	14	12	04	12	00	0c	00	13	11	08	23	19	19
Cipher Key:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Ciphertext:	5A	6F	4B	67	57	B7	A5	D2	C4	30	91	ED	64	9A	42	72

Analysis of AES

- Brute Force Attack
  - AES is definitely more secure than DES due to the larger-size key.
- Statistical Attacks
  - Numerous tests have failed to do statistical analysis of the ciphertext.
- Differential and Linear Attacks
  - There are no differential and linear attacks on AES as yet.

Implementation

- AES can be implemented in software, hardware, and firmware. The implementation can use table lookup process or routines that use a well-defined algebraic structure.

- The algorithms used in AES are so simple that they can be easily implemented using cheap processors and a minimum amount of memory.

AES Example

Plaintext:	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19
Cipher Key:	24	75	A2	B3	34	75	56	88	31	E2	12	00	13	AA	54	87
Ciphertext:	BC	02	8B	D3	E0	E3	B1	95	55	0D	6D	FB	E6	F1	82	41

AES Example Contd.

Table 7.7 *Example of encryption*

Round	*Input State*	*Output State*	*Round Key*
Pre-round	00 12 0C 08 04 04 00 23 12 12 13 19 14 00 11 19	24 26 3D 1B 71 71 E2 89 B0 44 01 4D A7 88 11 9E	24 34 31 13 75 75 E2 AA A2 56 12 54 B3 88 00 87
1	24 26 3D 1B 71 71 E2 89 B0 44 01 4D A7 88 11 9E	6C 44 13 BD B1 9E 46 35 C5 B5 F3 02 5D 87 FC 8C	89 BD 8C 9F 55 20 C2 68 B5 E3 F1 A5 CE 46 46 C1
2	6C 44 13 BD B1 9E 46 35 C5 B5 F3 02 5D 87 FC 8C	1A 90 15 B2 66 09 1D FC 20 55 5A B2 2B CB 8C 3C	CE 73 FF 60 53 73 B1 D9 CD 2E DF 7A 15 53 15 D4

AES Example Contd.

3	1A 90 15 B2 66 09 1D FC 20 55 5A B2 2B CB 8C 3C	F6 7D A2 B0 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	FF 8C 73 13 89 FA 4B 92 85 AB 74 0E C5 96 83 57
4	F6 7D A2 B0 1B 61 B4 B8 67 09 C9 45 4A 5C 51 09	CA E5 48 BB D8 42 AF 71 D1 BA 98 2D 4E 60 9E DF	B8 34 47 54 22 D8 93 01 DE 75 01 0F B8 2E AD FA
5	CA E5 48 BB D8 42 AF 71 D1 BA 98 2D 4E 60 9E DF	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	D4 E0 A7 F3 54 8C 1F 1E F3 86 87 88 98 B6 1B E1
6	90 35 13 60 2C FB 82 3A 9E FC 61 ED 49 39 CB 47	18 0A B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	86 66 C1 32 90 1C 03 1D 0B 8D 0A 82 95 23 38 D9

AES Example Contd.

7	18 0A B9 B5 64 68 6A FB 5A EF D7 79 8E B2 10 4D	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	62 04 C5 F7 83 9F 9C 81 3E B3 B9 3B B6 95 AD 74
8	01 63 F1 96 55 24 3A 62 F4 8A DE 4D CC BA 88 03	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	EE EA 2F D8 61 FE 62 E3 AC 1F A6 9D DE 4B E6 92
9	2A 34 D8 46 2D 6B A2 D6 51 64 CF 5A 87 A8 F8 28	0A D9 F1 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	E4 0E 21 F9 3F C1 A3 40 E3 FC 5A C7 BF F4 12 80
10	0A D9 F1 3C 95 63 9F 35 2A 80 29 00 16 76 09 77	BC E0 55 E6 02 E3 0D F1 8B B1 6D 82 D3 95 F8 41	DB D5 F4 0D F9 38 9B DB 2E D2 88 4F 26 D2 C0 40