BAHRIA UNIVERSITY
Discovering Knowledge

Advanced Encryption Standard (AES)

Data Encryption & Security (CEN-451)

Spring 2025 (BSE-8A&B)

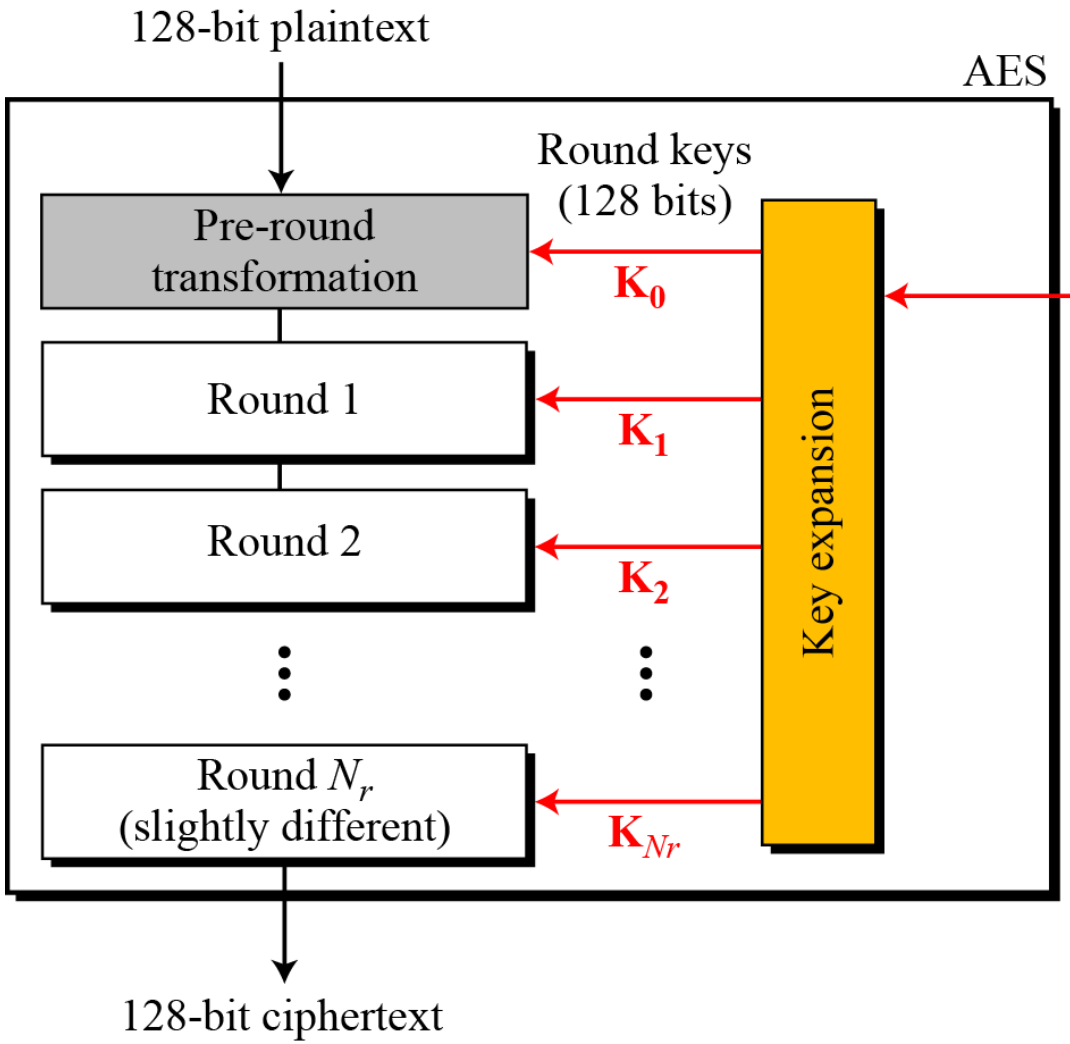
Overview

- **AES** was published in 2001, and still in wide spread use.
- A **symmetric block** cipher that is intended to replace **3-DES**, where **3-DES** will be disallowed from **2024**.
- Currently, **3-DES** has been superseded by **AES** in most applications. E.g. US government agencies have been using it to secure sensitive but unclassified material.
- Uses **128-bit block size** and **128, 192 and 256 bits keys**. Larger block size and longer keys than **3-DES**, hence more secure.
- Outperforms **3-DES** both in software and hardware implementations.

AES Structure

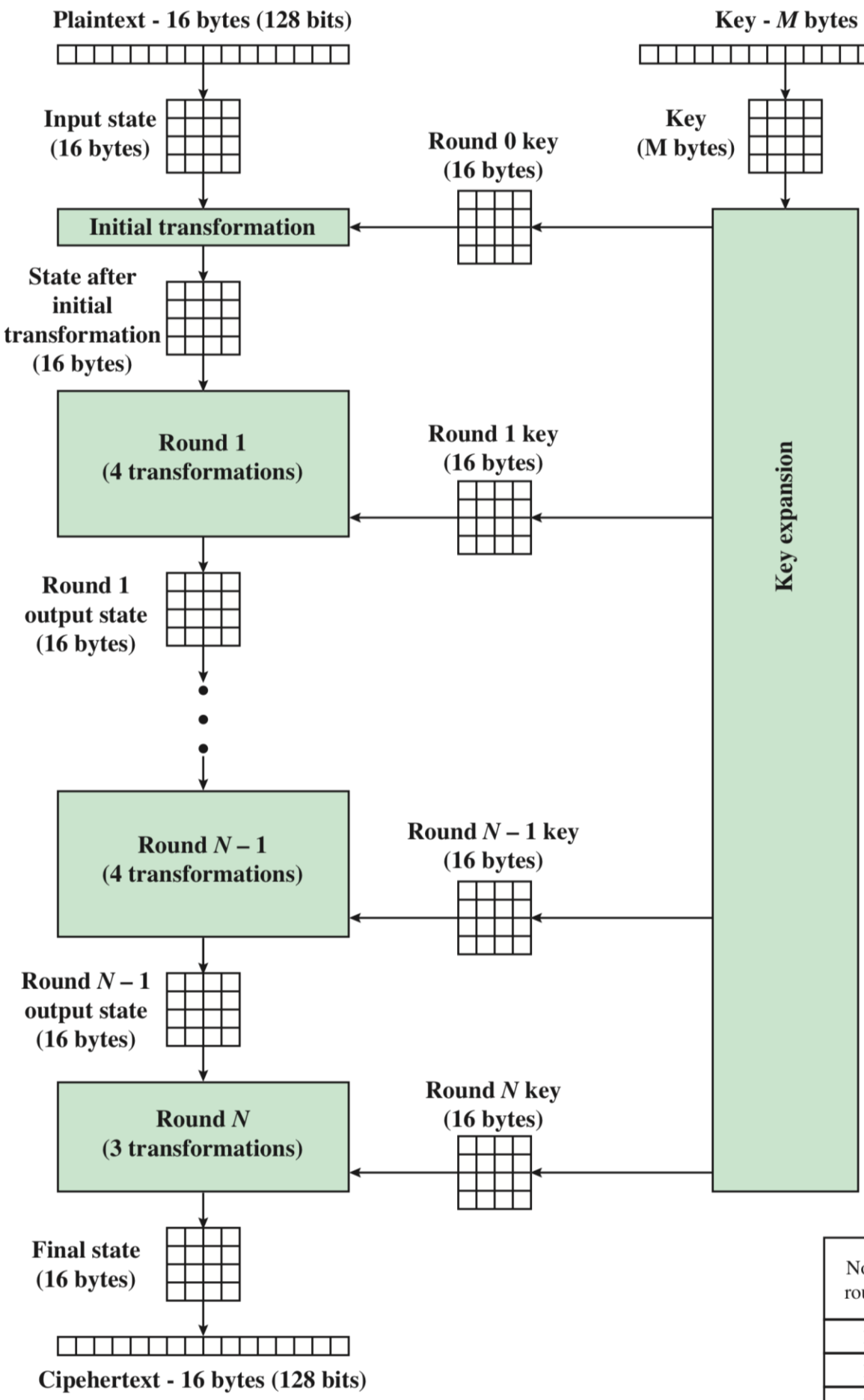
- **AES** encrypts and decrypts a data block of 128 bits.
- **AES** has defined three versions:
  - 10 rounds → key size 128-bits.
  - 12 rounds → key size 192-bits.
  - 14 rounds → key size 256-bits.
- However, the round keys are always **128 bits**.

AES Structure (Cont.)



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size



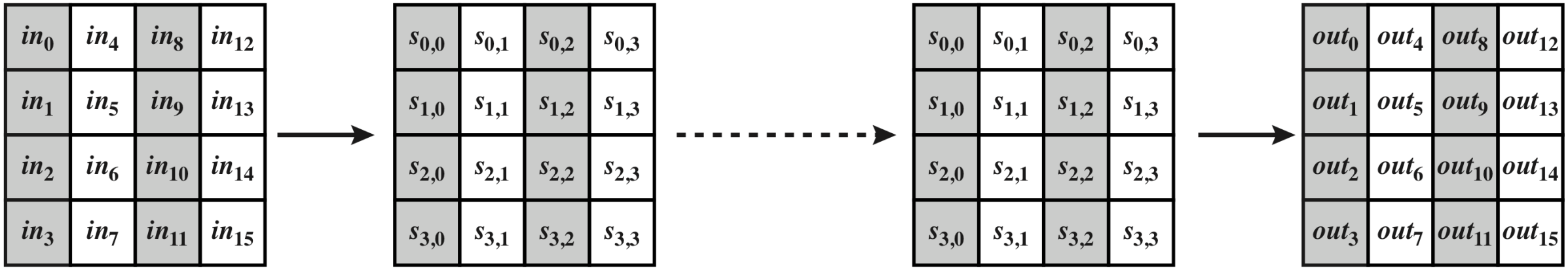
No. of rounds	Key Length (bytes)
10	16
12	24
14	32

AES Encryption: General Structure

AES Encryption: General Structure (Cont.)

- The cipher takes a plaintext block size of 128 bits or 16 bytes.
- The input is depicted as a **4 × 4 square matrix** of bytes.
- Ordering of bytes within a matrix is **by column**.
- The input is copied into the **State** array, which is modified at each stage of encryption or decryption.
- After the final stage, the **State** is copied to an output matrix.

AES Encryption: General Structure (Cont.)



The input, state array and output

AES Encryption: General Structure (Cont.)

- The key length can be 16, 24 or 32 bytes (i.e. 128, 192 or 256 bits).
- The algorithm is referred to as AES-128, AES-192 or AES-256, i.e. depending on key length.
- The key is **"expanded"** into an array of **words**.

The Key and Expanded Key **(AES-128)**

AES Encryption: General Structure (Cont.)

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

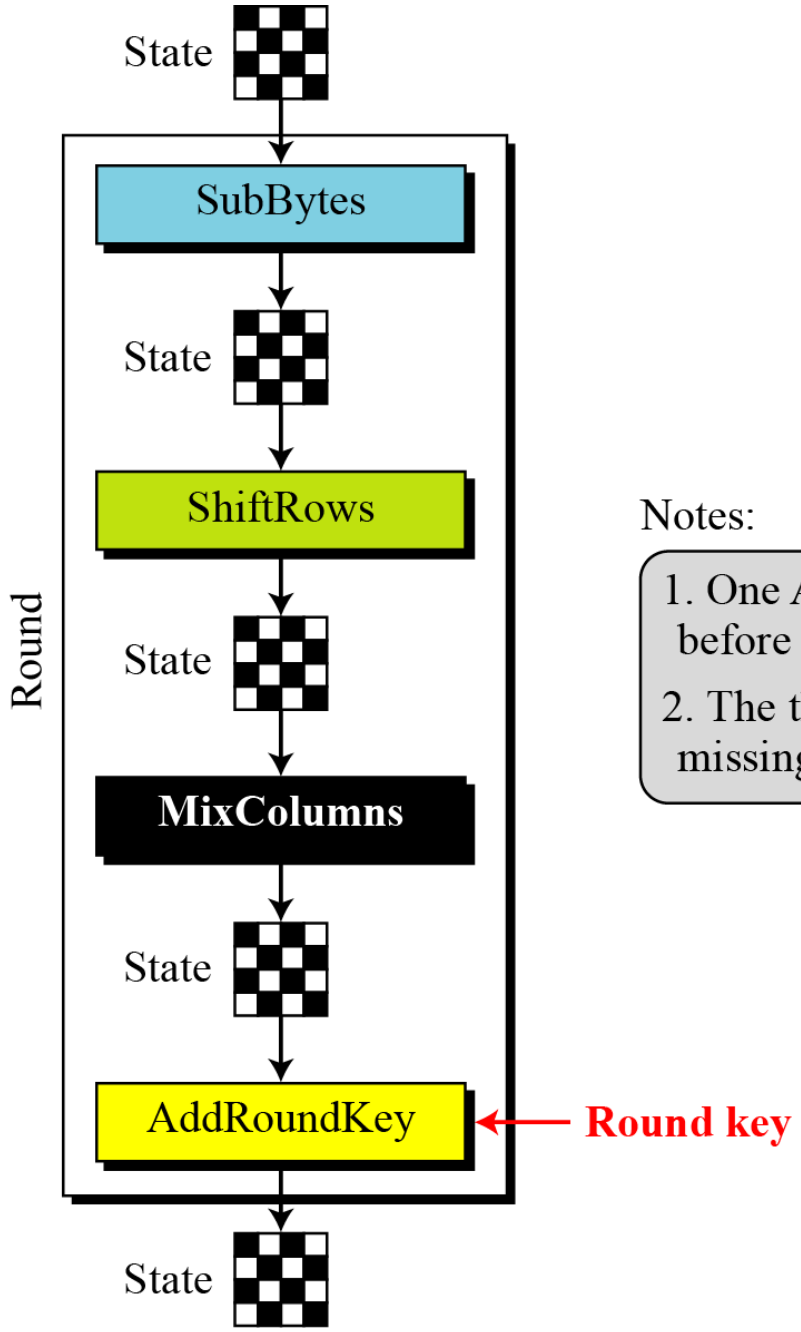
AES Encryption: General Structure (Cont.)

- The cipher consists of ***N* rounds**, where the number of rounds depends on the key length.
- There exists an initial single transformation **(AddRoundKey)** before the first round, which can be considered **Round 0**.
- The first *N* - 1 rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns and AddRoundKey.
- The final round contains only three transformations.

AES Encryption: General Structure (Cont.)

- Each transformation takes **4 × 4** matrices as input and produces a **4 × 4** matrix as output.
- The output of each round is a **4 × 4** matrix, with output of the final round being the ciphertext.
- The key expansion function generates $N + 1$ round keys, each of which is a distinct **4 × 4** matrix.
- Each round key serves as one of the inputs to the AddRoundKey transformation in each round.

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 Detailed Structure

AES Detailed Structure

General points on **AES** structure *(considering 128-bit key)*:

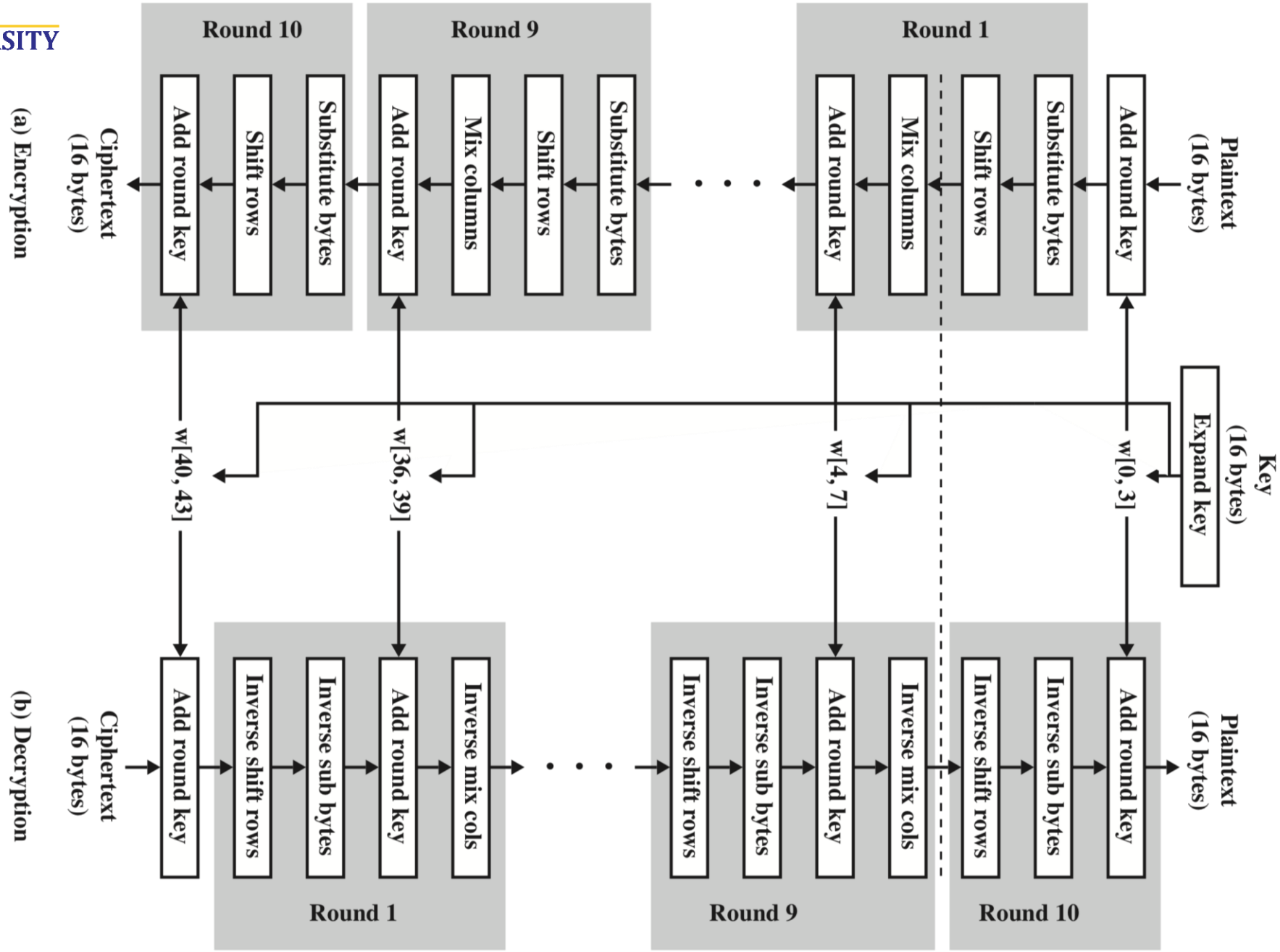
1. It is not a **Feistel** structure, where AES processes the entire data block as a single matrix.
2. Input key is expanded into an array of **forty-four** words, *w[i]*.
3. **Four** distinct words serve as a round key for each round.
4. In each round *(except for last)* four different stages are used.
5. In both encryption and decryption, the cipher begins with **AddRoundKey**, followed by **nine** rounds that each includes all **four stages**, followed by a **tenth** round of **three stages**.

AES Detailed Structure (Cont.)

6. Only **AddRoundKey** makes use of key.
7. The cipher begins and ends with an **AddRoundKey**, where any other stage applied at beginning or end is reversible.
8. The other three stages provide **confusion**, **diffusion** and **nonlinearity**, but by themselves would provide no security.
9. We can view the cipher as alternating operations of XOR **(AddRoundKey)** and scrambling **(other three stages)**.

AES Detailed Structure (Cont.)

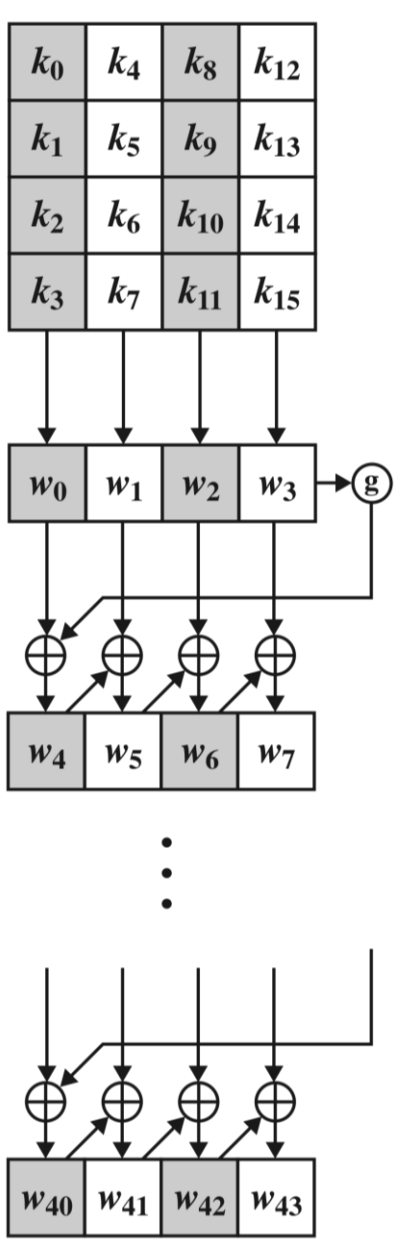
10. **Each stage is easily reversible**.
    a. For **Substitute Byte**, **ShiftRows** and **MixColumns**, an inverse function is used in decryption.
    b. For **AddRoundKey**, inverse is achieved by XORing the round key to block.
11. Decryption algorithm is **not identical** to encryption algorithm, ***which is a consequence of the particular structure of AES***.
12. The final round of both encryption and decryption consists of three stages. ***This is a consequence of the particular structure of AES and is required to make the cipher reversible***.


Round 10
Round 9
Round 1
Add round key
Shift rows
Substitute bytes
Add round key
Mix columns
Shift rows
Substitute bytes
Add round key
Mix columns
Shift rows
Substitute bytes
Ciphertext
(16 bytes)
Plaintext
(16 bytes)
(a) Encryption
w[40, 43]
w[36, 39]
w[4, 7]
w[0, 3]
Key
(16 bytes)
Expand key
Add round key
Inverse shift rows
Inverse sub bytes
Add round key
Inverse mix cols
Inverse shift rows
Inverse sub bytes
Add round key
Inverse mix cols
Inverse shift rows
Inverse sub bytes
Add round key
Ciphertext
(16 bytes)
Plaintext
(16 bytes)
(b) Decryption
Round 1
Round 9
Round 10

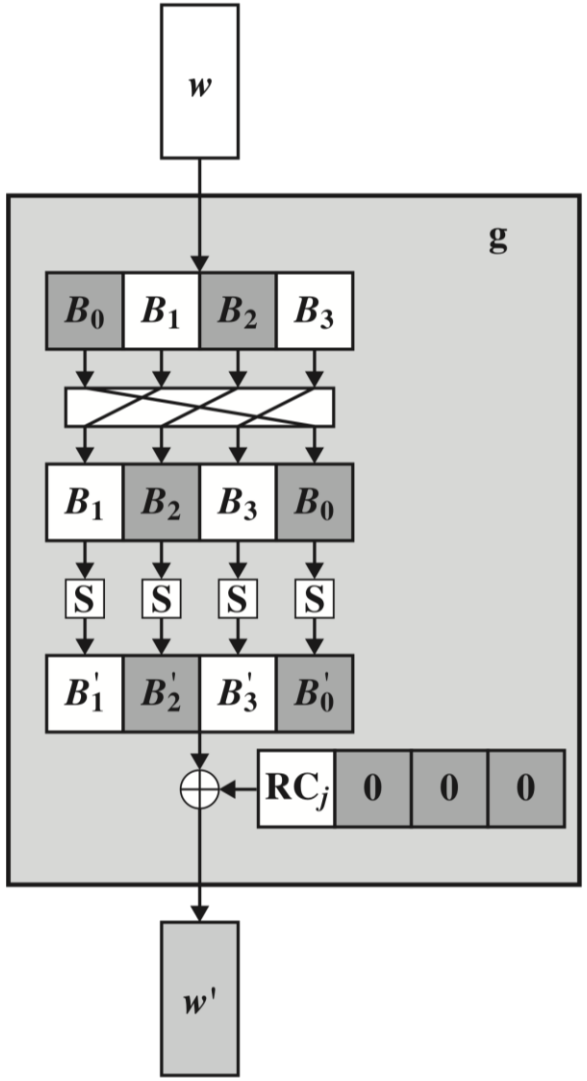
AES Key Expansion

AES Key Expansion

- Considering **128-bit** key input, **AES** key expansion algorithm takes input of **4 words (16-byte)** key and produces an expanded key of **44 words (176 bytes)**.
- The input key is copied into the first 4 words of expanded key, while remaining expanded key is ***filled four words at a time***.
- Each added word **w[i]** depends on the immediately preceding word **w[i - 1]** and the word four positions back **w[i - 4]**.
- In three out of four cases, a simple XOR is used.
- For a word whose position in the **w** array is a **multiple of 4**, a more complex function **g** is used.



(a) Overall algorithm



(b) Function g

AES Key Expansion (Cont.)

Function **g** consists of the following sub-functions:

1. **RotWord** performs a **one-byte** circular left shift on a word.
2. **SubWord** performs a **byte substitution** on each byte of its input word, using the **S-box**.
3. Result of steps 1 & 2 is **XORed** with a round constant **Rcon[j]**
   a. Round constant is a word in which the three rightmost bytes are always 0.
   b. The round constant is different for each round.
   c. Values of **RC[j]** in hexadecimal:

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

AES Key Expansion (Cont.)

		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

S-box

AES Key Expansion (Cont.)

- **Example:** suppose that round key for round 8 is as given below. Calculate the round key for the next round.

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

- **Solution:** first 4 bytes **(1st word)** of round 9 key are calculated using **g** function, since its index is **multiple of 4** ***(i = 36)***
- Last word of round 8 key = **7F 8D 29 2F**
- Apply **RotWord** = **8D 29 2F 7F**
- Apply **SubWord** = **5D A5 15 D2**

AES Key Expansion (Cont.)

- **Example (Cont.):**
- Given that, **RC[9] = 1B 00 00 00**
- **SubWord ⊕ RC[9] = 5D A5 15 D2 ⊕ 1B 00 00 00 = 46 A5 15 D2**
- We have, **w[i – 4] = EA D2 73 21 ⊕ 46 A5 15 D2 = AC 77 66 F3**
- Hence, w_{36} **= AC 77 66 F3**

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i − 4]	w[i] = temp ⊕ w[i − 4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

AES Key Expansion (Cont.)

- **Example (Cont.):** remaining three words are calculated using simple **XOR** operations between immediately preceding word **w[i - 1]** and the word four positions back **w[i - 4]**.
- w_{37} = **AC 77 66 F3** $\oplus$ **B5 8D BA D2** = **19 FA DC 21**
- w_{38} = **19 FA DC 21** $\oplus$ **31 2B F5 60** = **28 D1 29 41**
- w_{39} = **28 D1 29 41** $\oplus$ **7F 8D 29 2F** = **57 5C 00 6E**
- Hence, round 9 key is:

AC 77 66 F3 19 FA DC 21 28 D1 29 41 57 5C 00 6E

AES Transformation Functions

Substitute Bytes Transformation

- Forward substitute byte is a simple **table lookup**.
- AES defines a **16 × 16** matrix of byte values, called an **S-box**.
- The **S-box** contains a permutation of all possible 256 8-bit values.
- Each byte of **State** is mapped into a new byte by:
  - **Leftmost 4 bits** of the byte are used as a **row value** and **rightmost 4 bits** are used as a **column value**.
  - The row and column values serve as indexes into the **S-box** to select a unique 8-bit output value.

Substitute Bytes Transformation (Cont.)

		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

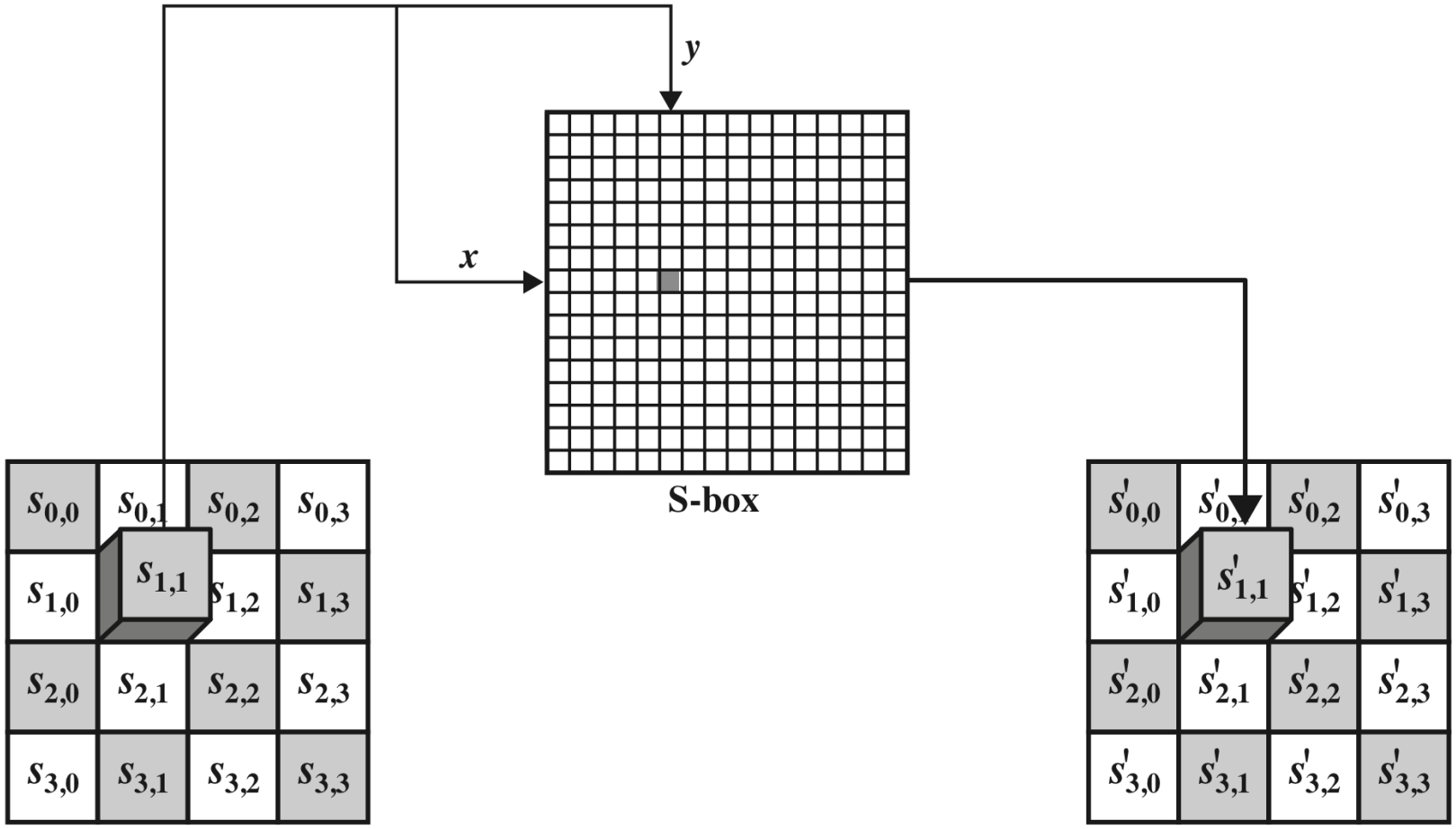
S-box

Substitute Bytes Transformation (Cont.)

		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 S-box

Substitute Bytes Transformation (Cont.)



(a) Substitute byte transformation

Substitute Bytes Transformation (Cont.)

- An example of the **SubBytes** transformation:

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

→

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

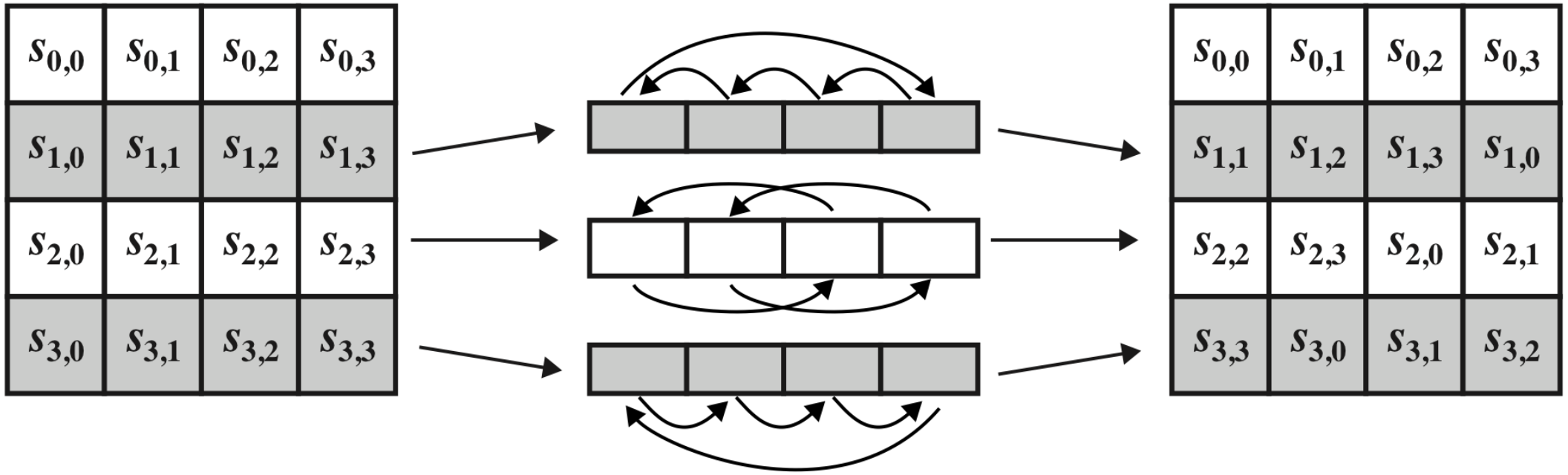
Substitute Bytes Transformation (Cont.)

- **S-box** is designed to be resistant to known **cryptanalytic attacks**.
- Developers sought a design that has **low correlation** between input bits and output bits.
- Developers sought a design that output is **not a linear mathematical function** of the input.

ShiftRows Transformation

- The *forward shift row transformation* is called **ShiftRows**.
- The **first row** of **State** is not altered.
- For **second row**, a 1-byte circular left shift is performed.
- For **third row**, a 2-bytecircular left shift is performed.
- For **fourth row**, a 3-byte circular left shift is performed.

ShiftRows Transformation (Cont.)



(a) Shift row transformation

ShiftRows Transformation (Cont.)

- The following is an example of **ShiftRows**.

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

→

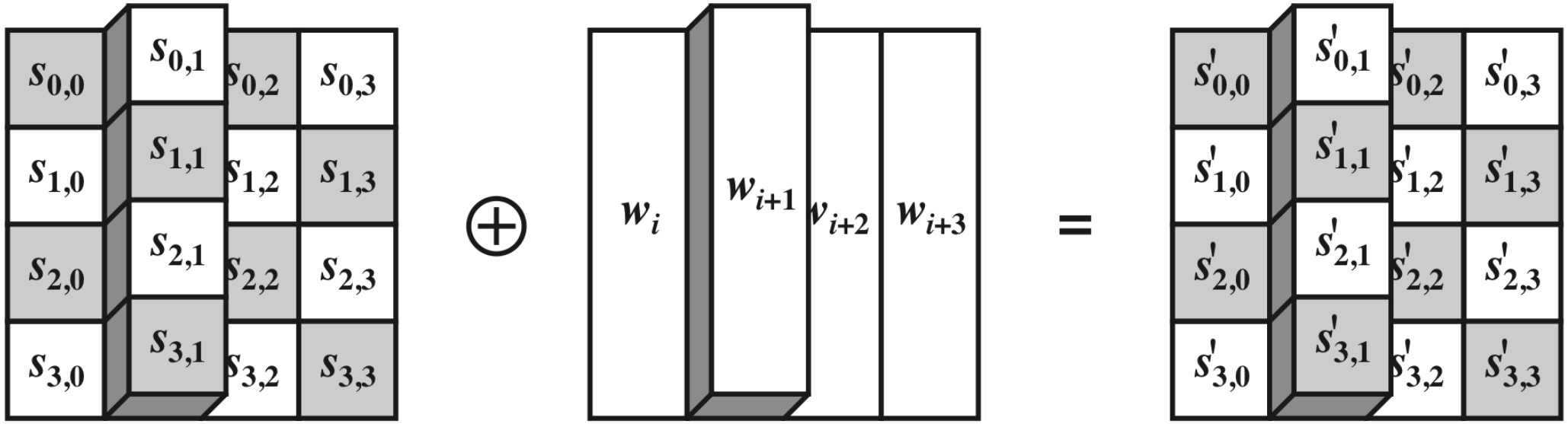
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

- The *inverse shift row transformation*, called **InvShiftRows**, performs the circular shifts in the opposite direction for each of the last three rows (1-byte circular right shift for second row, and so on…).

AddRoundKey Transformation

- In *forward add round key transformation*, called **AddRoundKey**, the 128 bits of **State** are **XOR** with the 128 bits of the **round key**.
- The operation is viewed as a **column-wise** operation between the 4 bytes of a State column and one word of the round key *(it can also be viewed as a byte-level operation)*.
- The *inverse add round key transformation* is identical to the *forward add round key transformation*, because **XOR** is its own inverse.

AddRoundKey Transformation (Cont.)



(b) Add round key Transformation

AddRoundKey Transformation (Cont.)

- The following is an example of AddRoundKey, where the first matrix is **State**, and the second matrix is the **round key**:

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

$=$

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

MixColumns Transformation

- **AES** uses arithmetic in the finite field **GF(2^8)** for mix column transformation.
- Consider two elements $A = (a_7 a_6 \dots a_1 a_0)$ and $B = (b_7 b_6 \dots b_1 b_0)$.
- The sum $A + B = (c_7 c_6 \dots c_1 c_0)$, where $c_i = a_i \oplus b_i$.
- The multiplication **{02}** . A equals $(a_6 a_5 \dots a_0 0)$ if $a_7 = 0$ and equals $(a_6 a_5 \dots a_0 0) \oplus (00011011)$ if $a_7 = 1$.
- The multiplication **{03}** . A equals $A \oplus$ ({02} . A)

MixColumns Transformation (Cont.)

- The forward mix column transformation, called **MixColumns**, operates on each column individually.
- **Each byte of a column** is mapped into a new value that is a function of **all four bytes in that column**.
- Transformation is defined by a matrix multiplication on **State**.

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

MixColumns Transformation (Cont.)

- The MixColumns transformation on a single column of **State** can be expressed as:

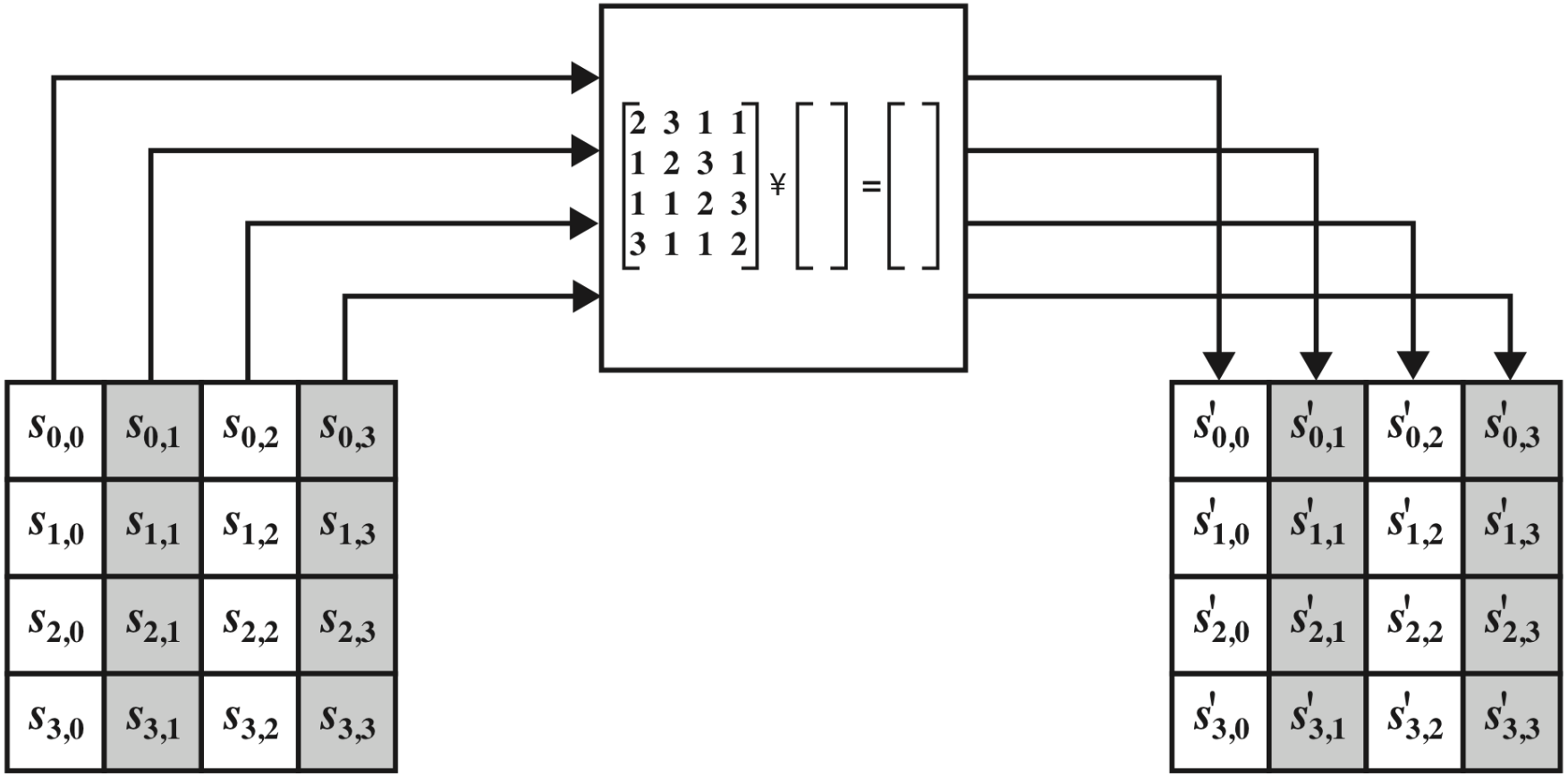
$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j}$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$

$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

MixColumns Transformation (Cont.)



(b) Mix column transformation

MixColumns Transformation (Cont.)

- The following is an example of **MixColumns**:

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

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

- In this example, we will verify the first column only while the rest is left as an exercise for students.

MixColumns Transformation (Cont.)

- To verify **MixColumns** transformation on the first column, we need to show that:

$$
\begin{aligned}
(\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\
\{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} &= \{37\} \\
\{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) &= \{94\} \\
(\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) &= \{ED\}
\end{aligned}
$$

MixColumns Transformation (Cont.)

- For the first equation **(in previous slide)**:
  - **{02} . {87}**

a. Convert 87 into its binary format, i.e. 1000 0111.

b. Check MSB for 0 or 1:

- ✓ If 0, remove MSB and insert 0 in LSB.
- ✓ If 1, remove MSB and insert 0 in LSB and XOR with **0001 1011**
- ✓ Hence, **{02} . {87}** = (0000 1110) $\oplus$ **(0001 1011)** = (0001 0101)

MixColumns Transformation (Cont.)

c. **{03} . {6E}** = {6E} ⊕ ({02} . {6E}) = (0110 1110) ⊕ (1101 1100) = **(1011 0010)**

{02} . {87} =	0001 0101
{03} . {6E} =	1011 0010
{46} =	0100 0110
{A6} =	1010 0110

0100 0111 = {47}

- ***The other three equations can be similarly verified!***

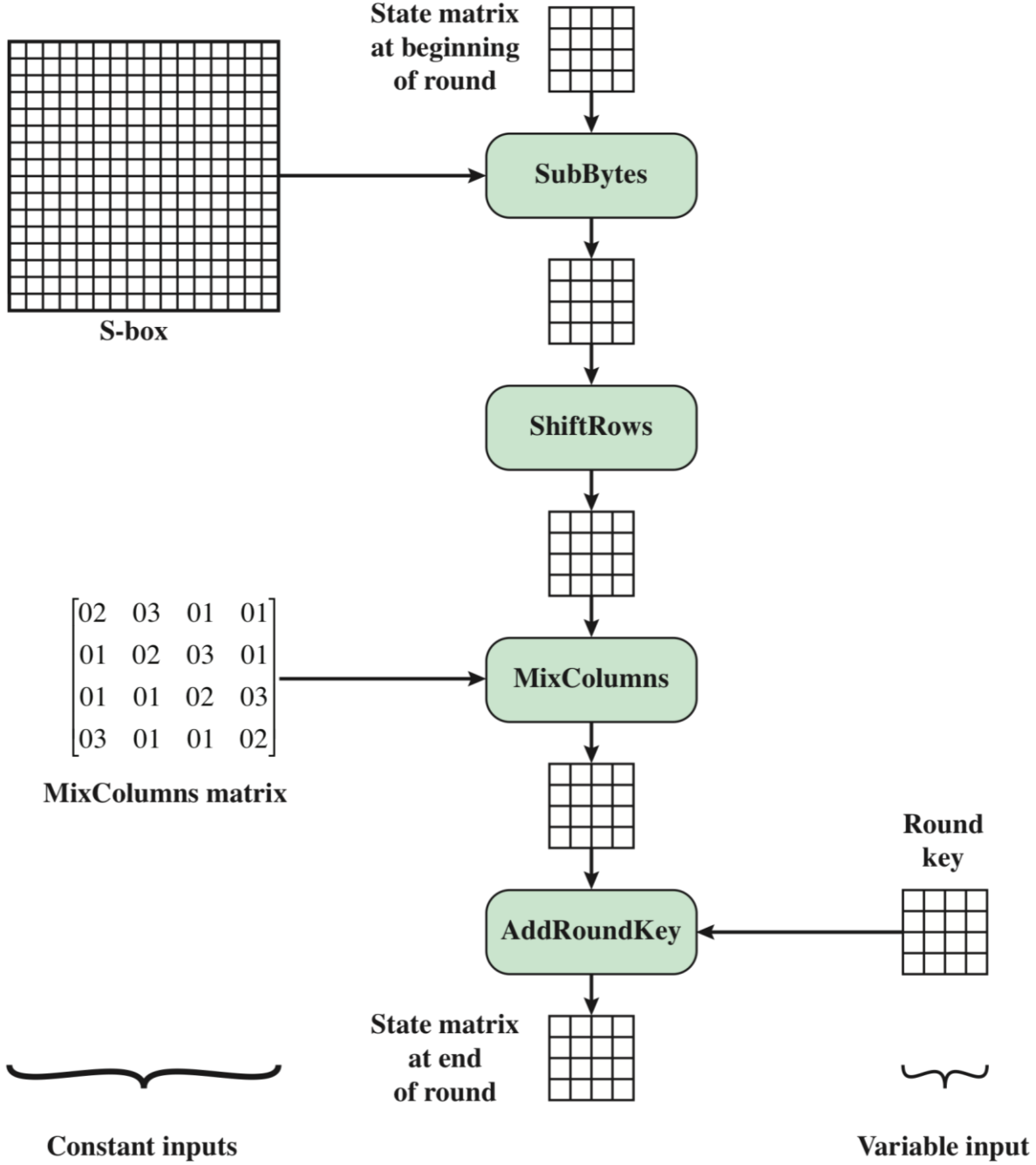
MixColumns Transformation (Cont.)

- The inverse mix column transformation, called **InvMixColumns**, is defined by:

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

- ***Details of the above matrix multiplication is beyond the scope of this course.***

Inputs for Single AES Round



Thank You!