11464: INFORMATION SYSTEMS SECURITY

12/5/2021

Chapter 4: Advanced Encryption Standard (AES)

2

Advanced Encryption Standard (AES)

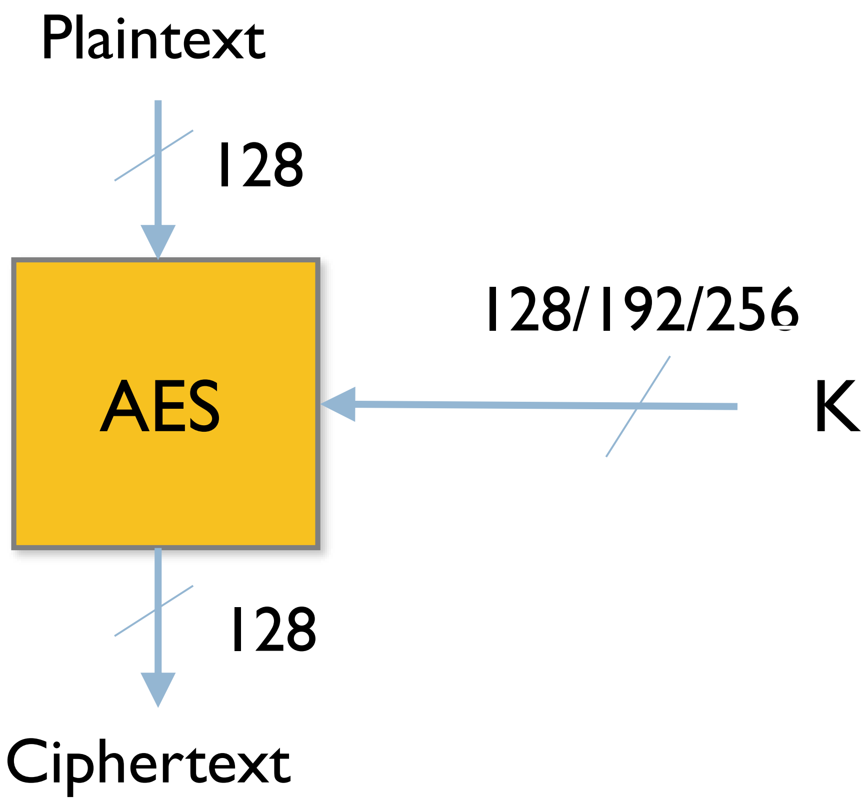
By

Mustafa Al-Fayoumi

Overview of the AES algorithm

- The Advanced Encryption Standard (AES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST) in December 2001.

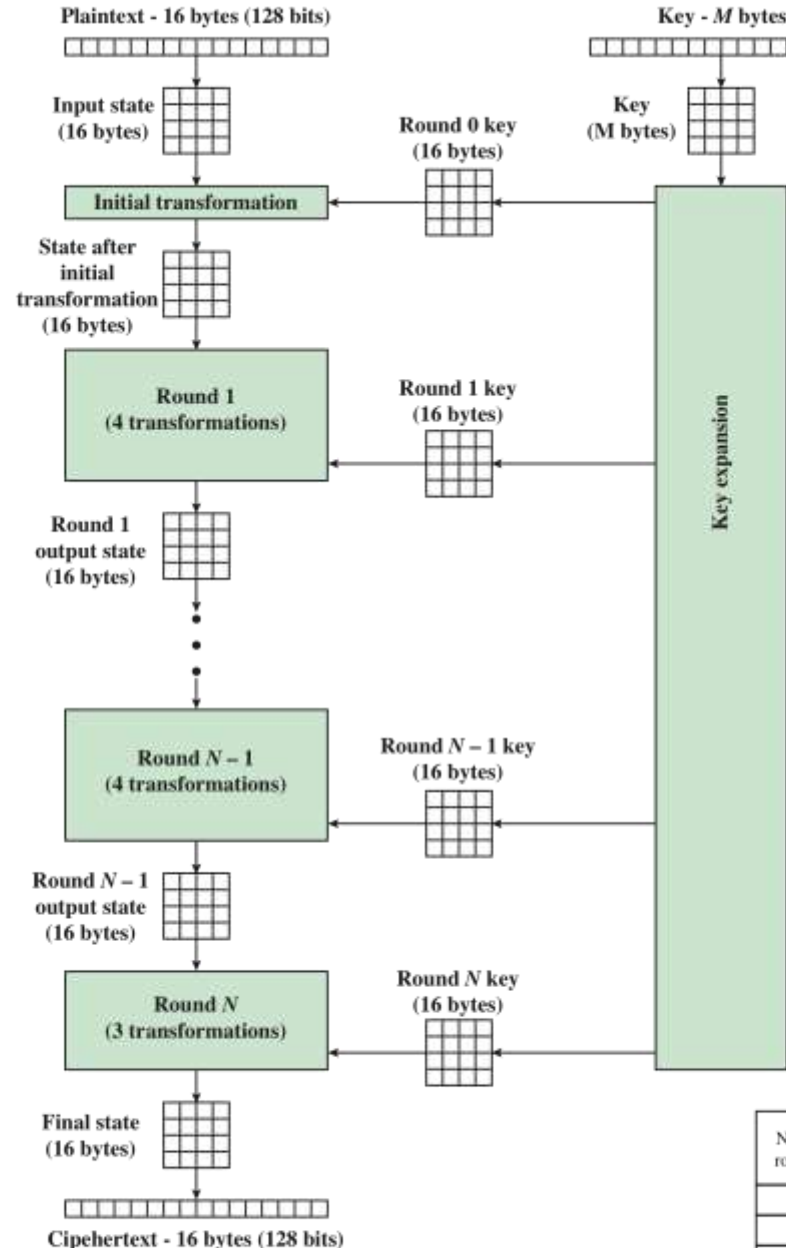
Overview of the AES algorithm



Overview of the AES algorithm

- The number of rounds depends on the chosen key length:
- In the Advanced Encryption Standard (AES) all operations are performed on 8-bit bytes
- The arithmetic operations of addition, multiplication, and division are performed over the finite field $GF(2^8)$
- A field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set
- Division is defined with the following rule:
  - $a / b = a (b^{-1})$
- An example of a finite field (one with a finite number of elements) is the set Z_p consisting of all the integers $\{0, 1, \ldots, p - 1\}$, where p is a prime number and in which arithmetic is carried out modulo p

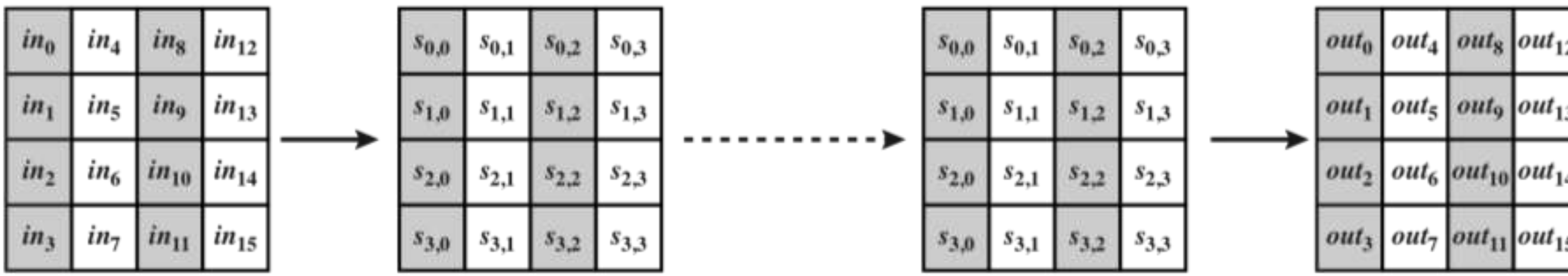
AES Encryption Process



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

Figure 7.1 AES Encryption Process

AES Data Structures



(a) Input, state array, and output

(b) Key and expanded key

Table 5.1 AES Parameters

- Block length (plaintext, cipher text) is limited to 128 bits
- The key size can be independently specified to 128, 192 or 256 bits
- Words = 32 bits = 4 byte

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

Rounds

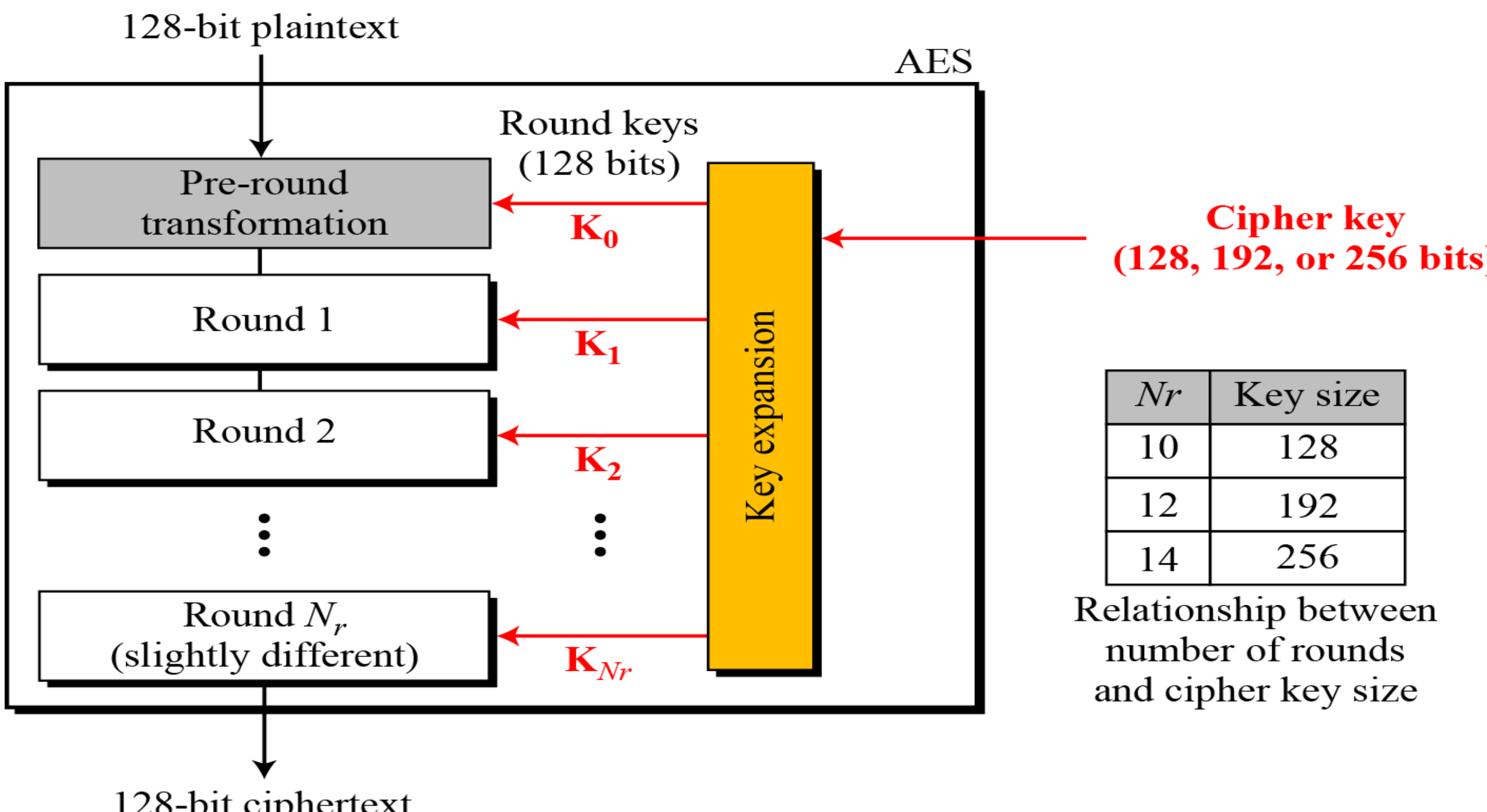
- AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Note

AES has defined three versions, with 10, 12, and 14 rounds.
Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.

Rounds

Figure 7.2 General design of AES encryption cipher

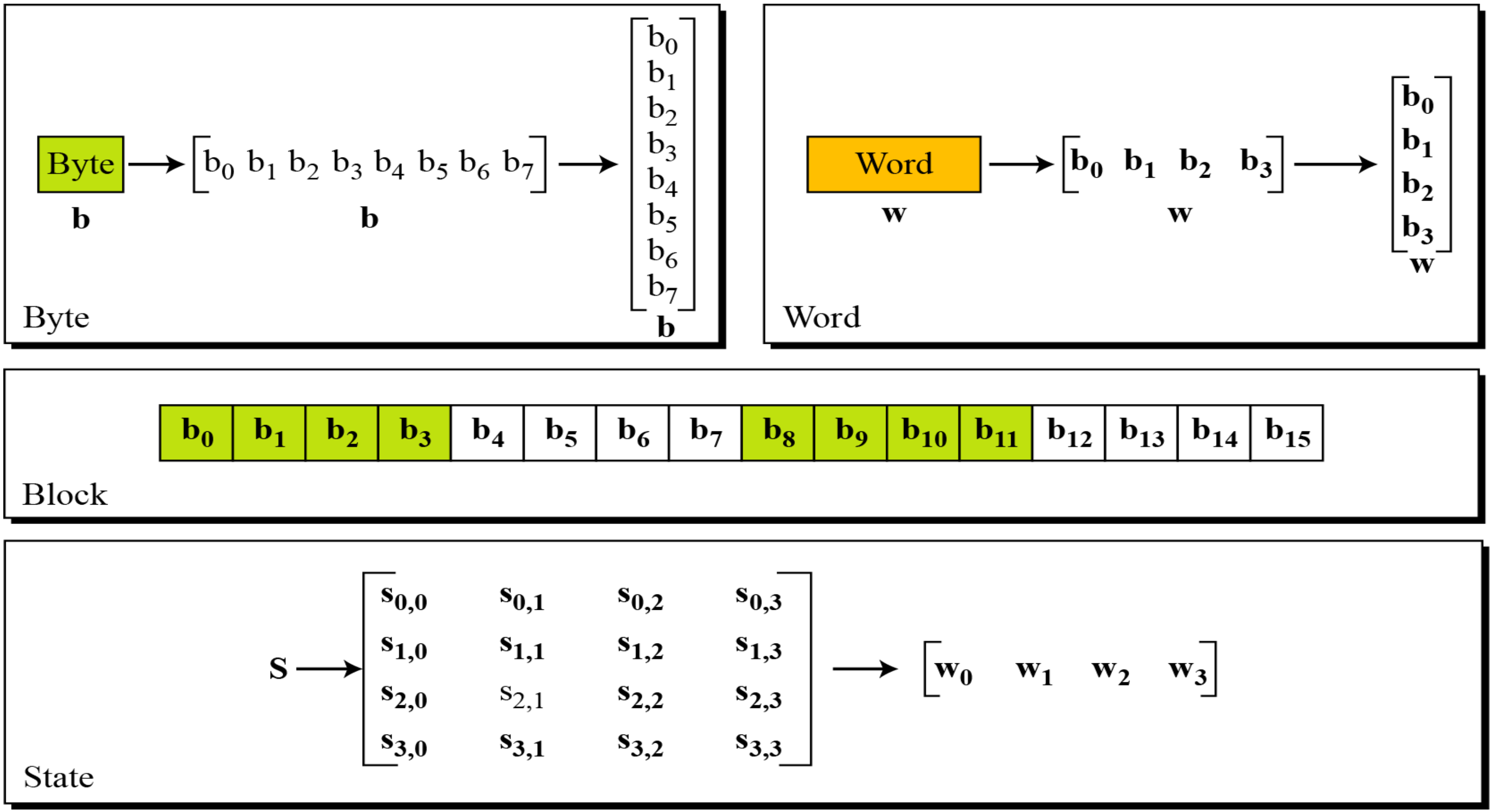


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

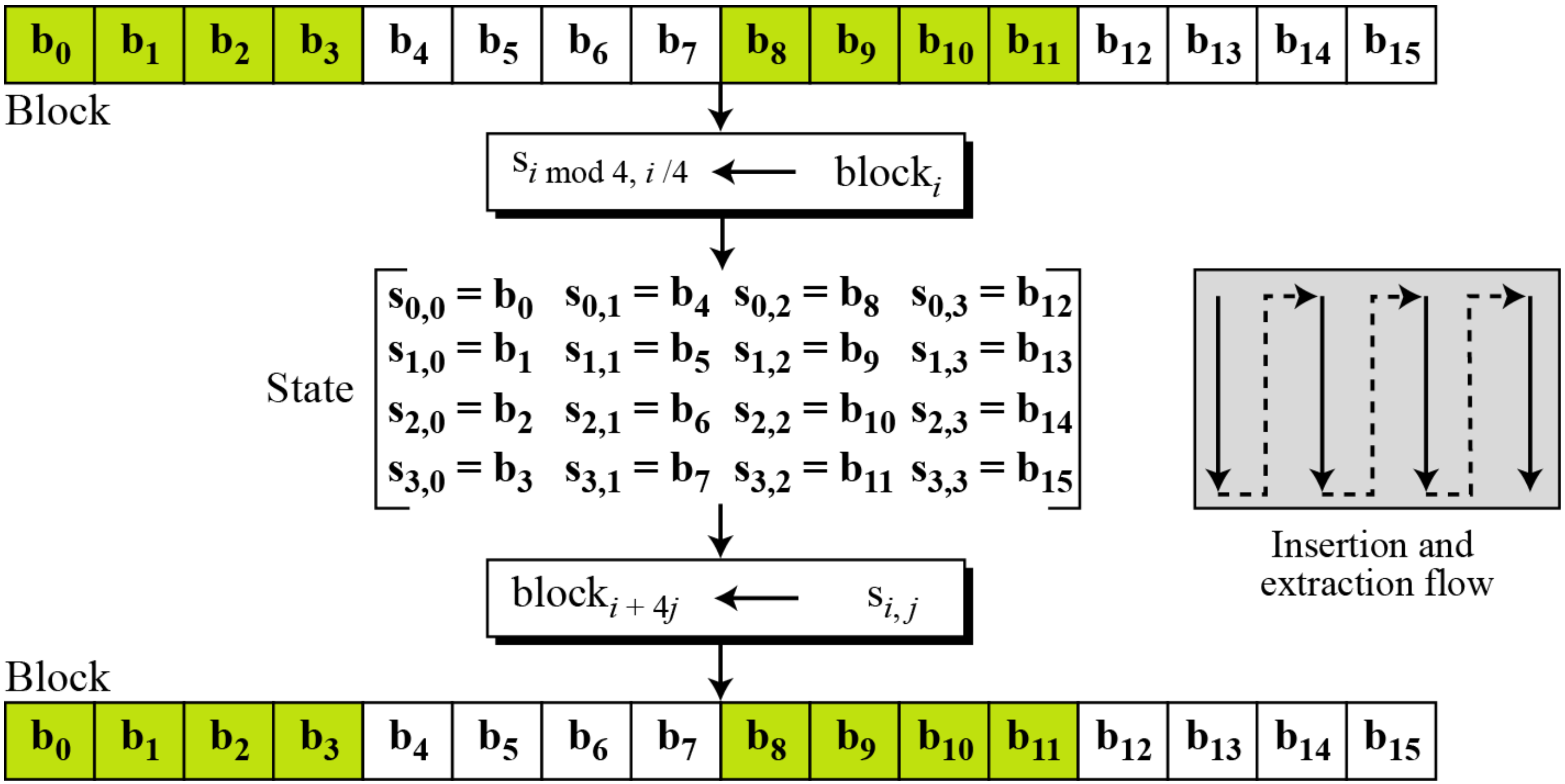
Data Units

Figure 7.3 Data units used in AES



Unit Transformation

Figure 7.4 Block-to-state and state-to-block transformation



Changing Plaintext to State

Example

- Block –state Example
- AES is a byte-oriented cipher

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

$$\begin{array}{cc} \begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix} & \text{State} \\ W_0 \quad W_1 \quad W_2 \quad W_3 & \end{array}$$

- We have 16 hexadecimal value * 8 bits for each value = 128 bits
- The state *A* (i.e., the 128-bit data path) can be arranged in a 4x4 matrix:
- **The first step** is convert block of plaintext to state as in above table Initial state is written as below

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

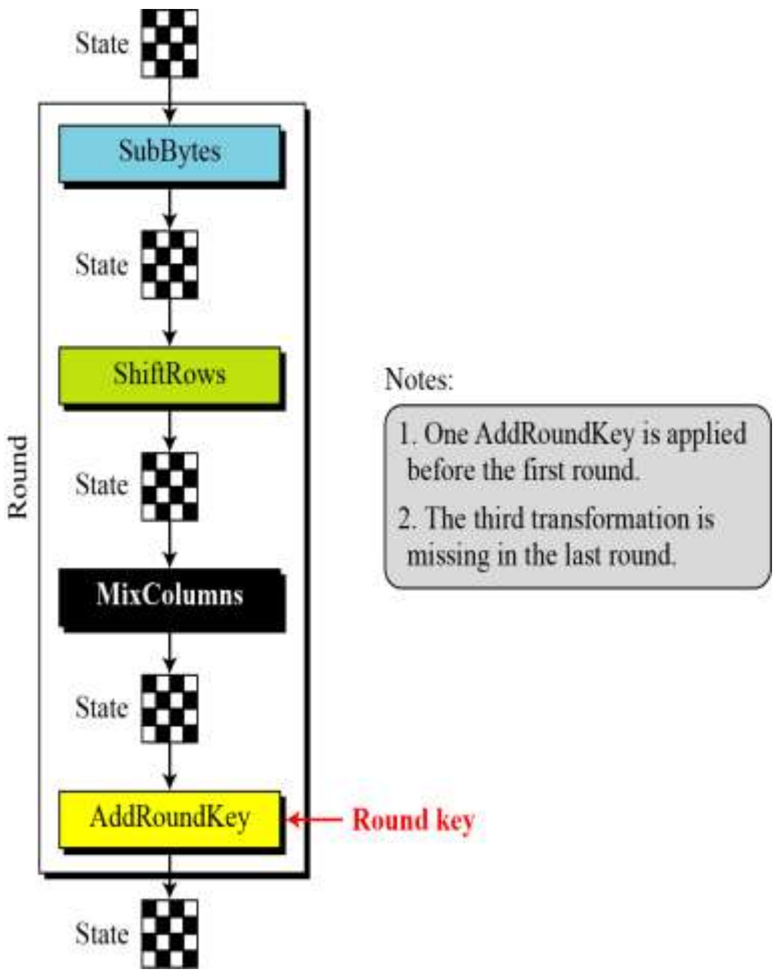
with A0,…, A15 denoting the 16-byte input of AES

Details of Each Round

- ADD ROUND KEY
- BYTE SUB
- SHIFT ROW
- MIX COLUMN

AES encryption cipher using a 16 byte key.

Round	Function
–	Add Round Key(State)
0	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
1	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
2	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
3	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
4	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
5	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
6	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
7	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
8	Add Round Key(Mix Column(Shift Row(Byte Sub(State))))
9	Add Round Key(Shift Row(Byte Sub(State)))



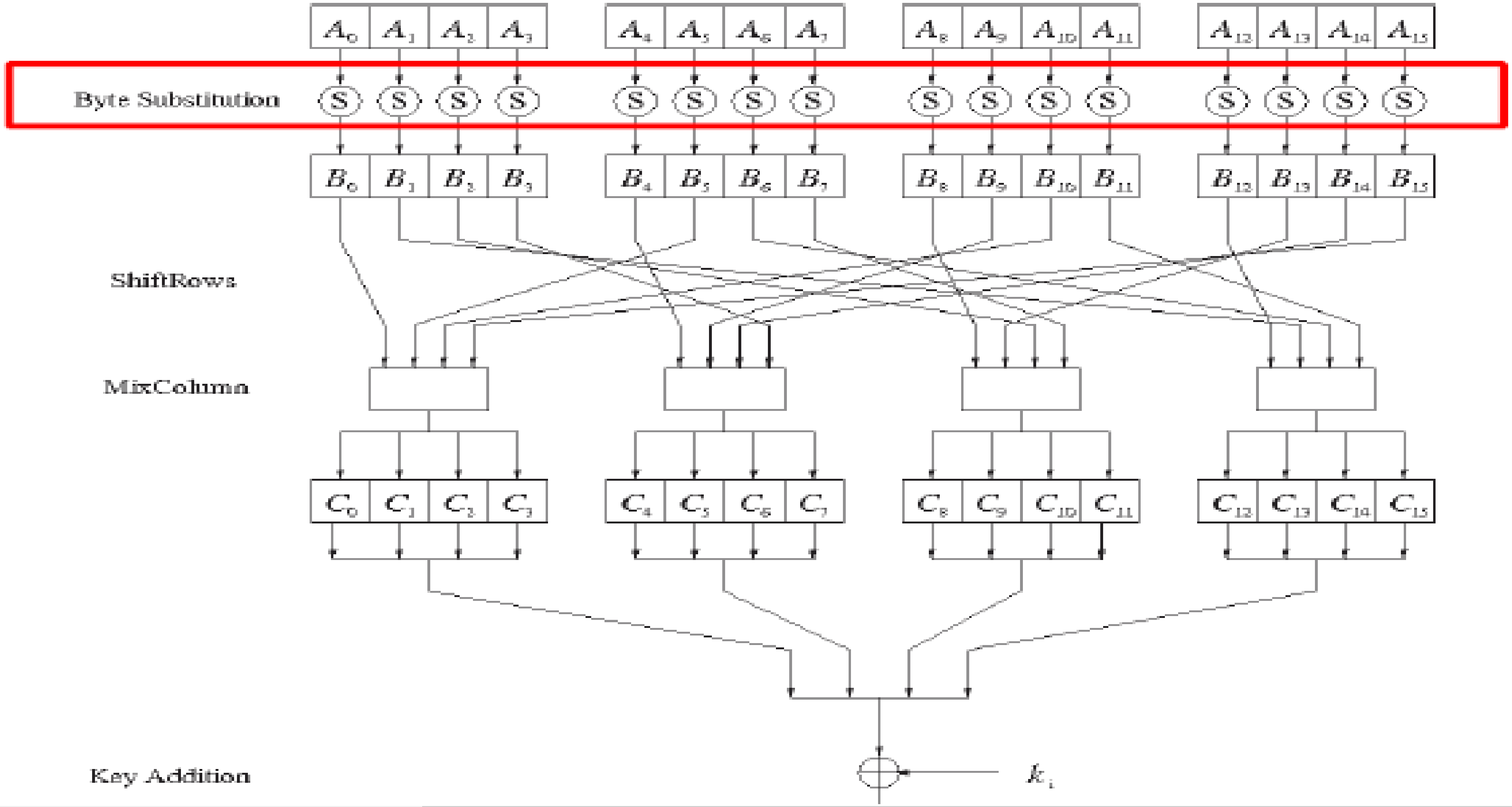
Step2: AddRoundKey for first state

- **The second step** is perform AddRoundKey (w0, w1, w2, w3). This means perform Xor between state and k0 (w0, w1, w2, w3)

- The input for round1 is the output of second step. The round 1 includes:
  - SubByte (i.e s-box 16*16)
  - ShiftRows
  - MixColumns
  - AddRoundKey

Step 3: Byte Substitution

- Byte Substitution Layer



Step 3: Byte Substitution

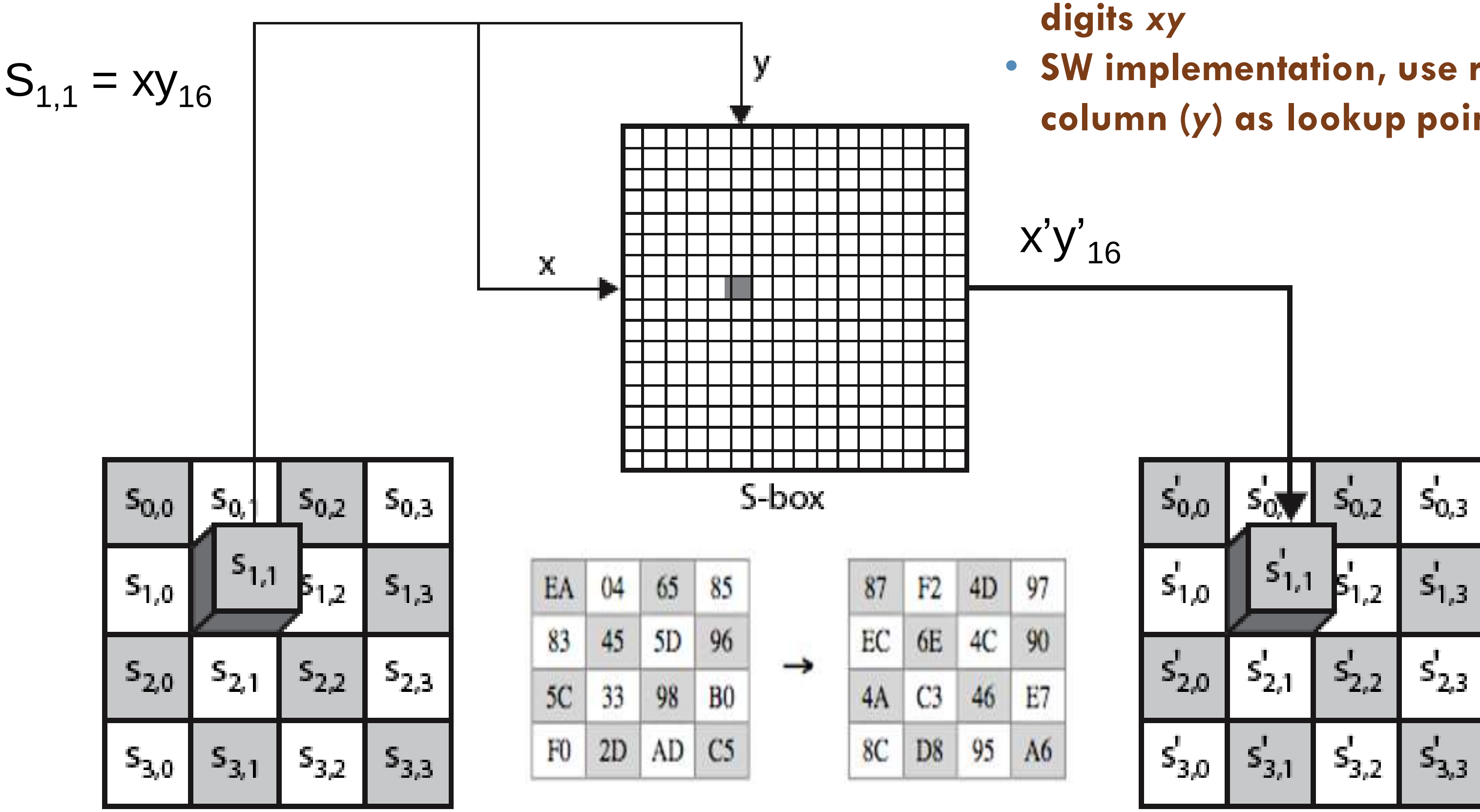
- AES, like DES, uses substitution. AES uses two invertible transformations.
- SubBytes
  - The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits.

Note

The SubBytes operation involves 16 independent byte-to-byte transformations.

SubBytes Operation

- **Interpret the byte as two hexadecimal digits *xy***
- **SW implementation, use row (*x*) and column (*y*) as lookup pointer**



SubBytes Table

- Implement by Table Lookup- AES S-Box Lookup Table
- During encryption each value of the state is replaced with the corresponding SBOX value
- For example HEX 19 would get replaced with HEX D4

		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

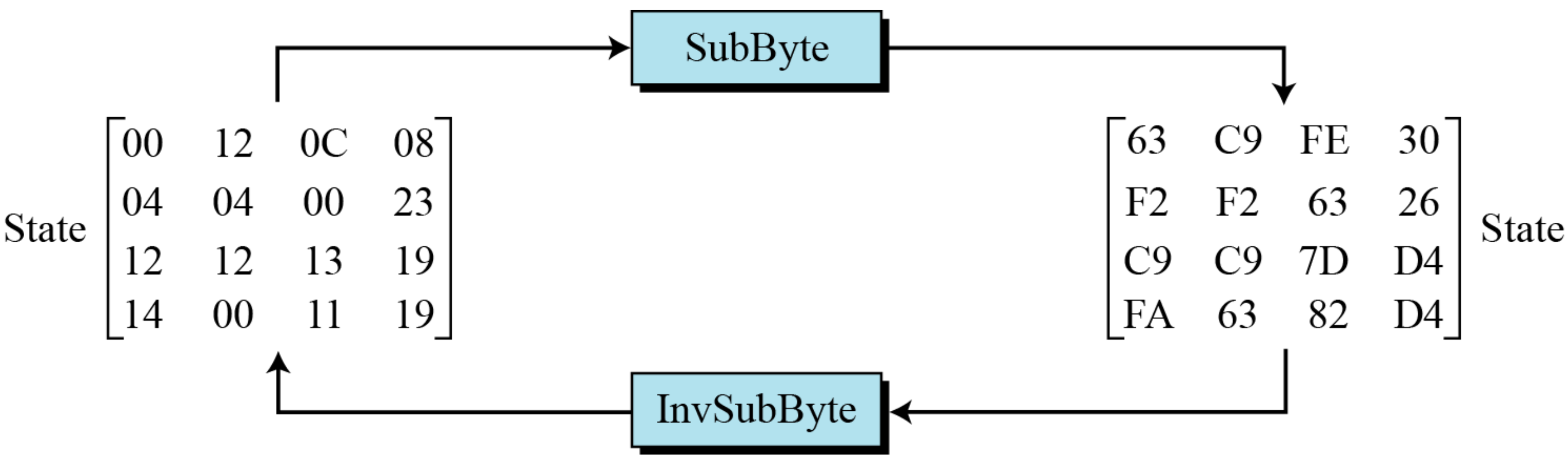
InvSubBytes Table

- During decryption each value in the state is replaced with the corresponding inverse of the SBOX
- For example HEX D4 would get replaced with HEX 19

		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

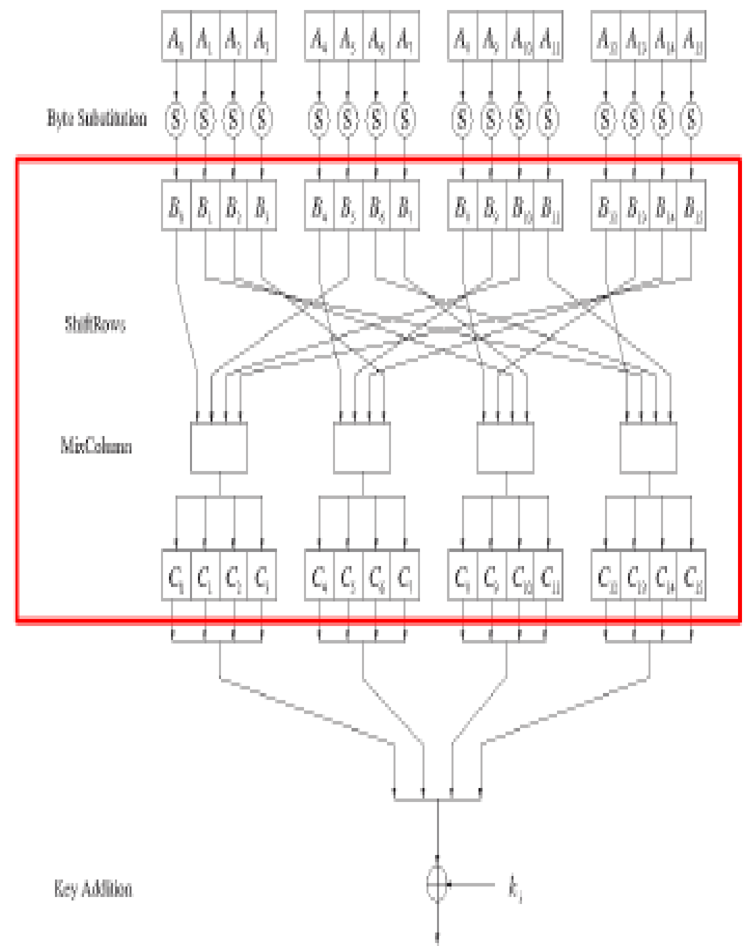
Sample SubByte Transformation

- The SubBytes and InvSubBytes transformations are inverses of each other.
- The following Figure shows how a state is transformed using the SubBytes transformation. The figure also shows that the InvSubBytes transformation creates the original one. Note that if the two bytes have the same values, their transformation is also the same.



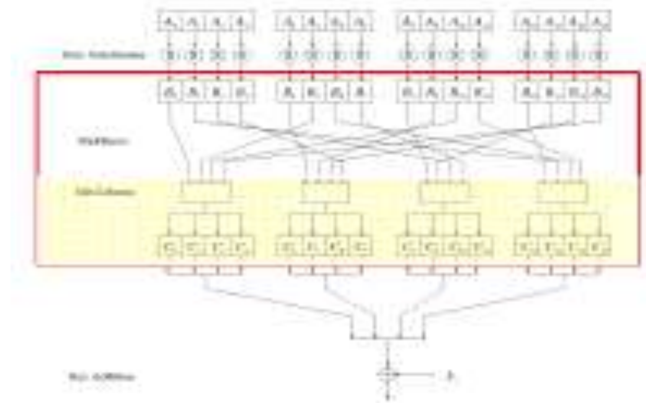
Diffusion Layer

- The Diffusion layer
  - provides diffusion over all input state bits
- consists of two sublayers:
  - **ShiftRows Sublayer**: Permutation of the data on a byte level
  - **MixColumn Sublayer**: Matrix operation which combines ("mixes") blocks of four bytes
- performs a linear operation on state matrices *A*, *B*, i.e.,
- DIFF(A) + DIFF(B) = DIFF(A + B)

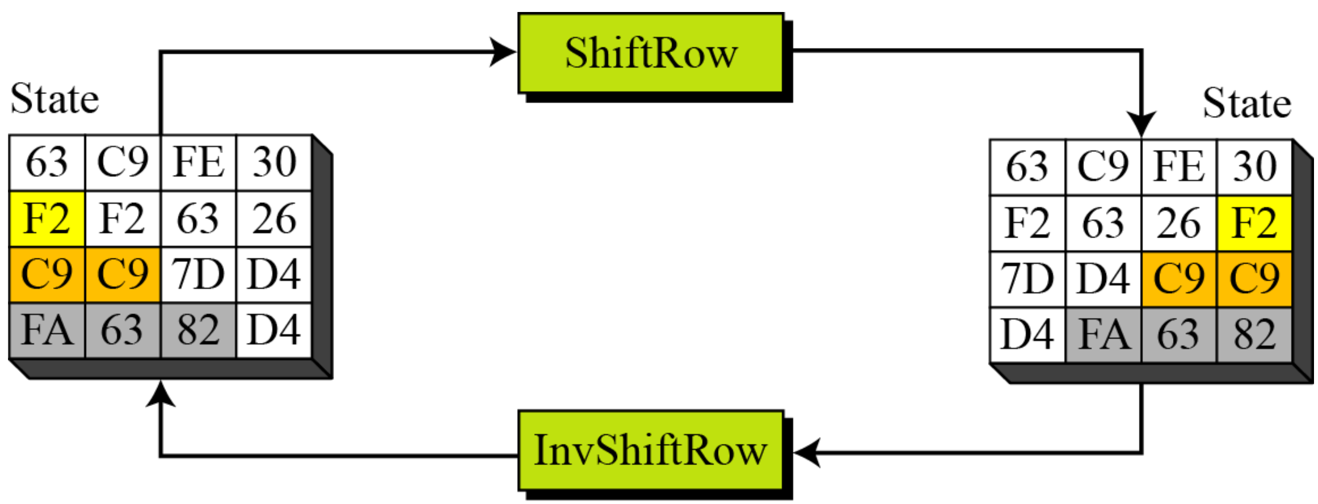


Step 4:ShiftRows

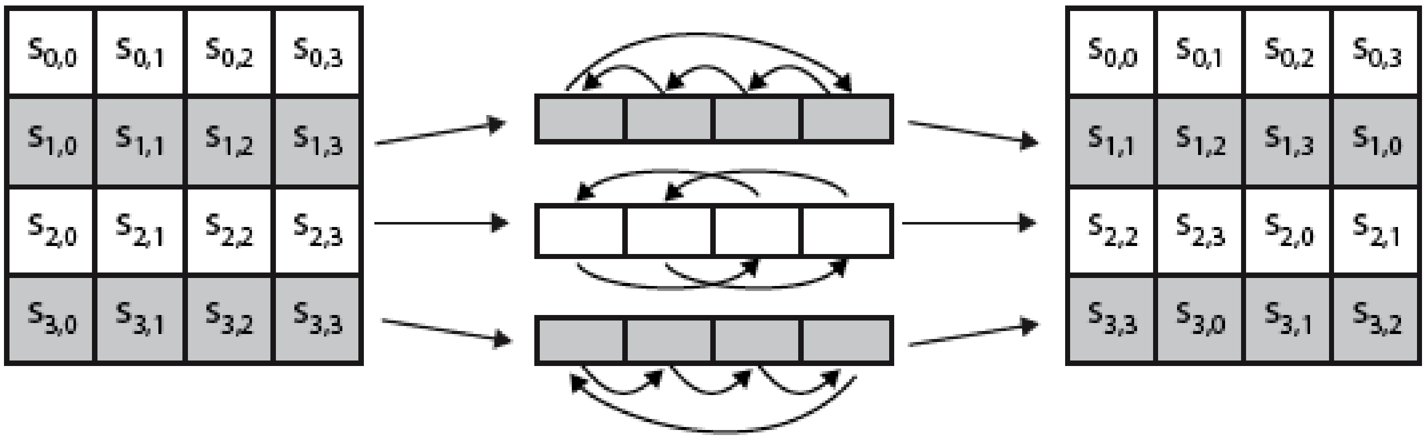
- Rows of the state matrix are shifted cyclically
- Shifting, which permutes the bytes.
- A circular byte shift in each each
  - 1[st] row is unchanged
  - 2[nd] row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left

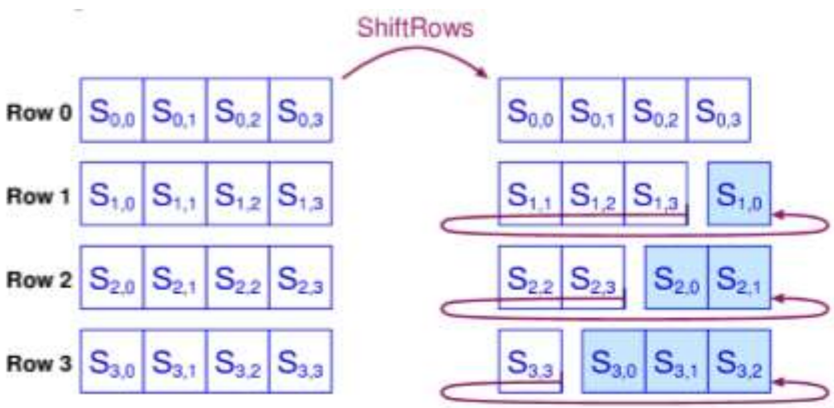


- In the encryption, the transformation is called ShiftRows
- In the decryption, the transformation is called InvShiftRows and the shifting is to the right



ShiftRows





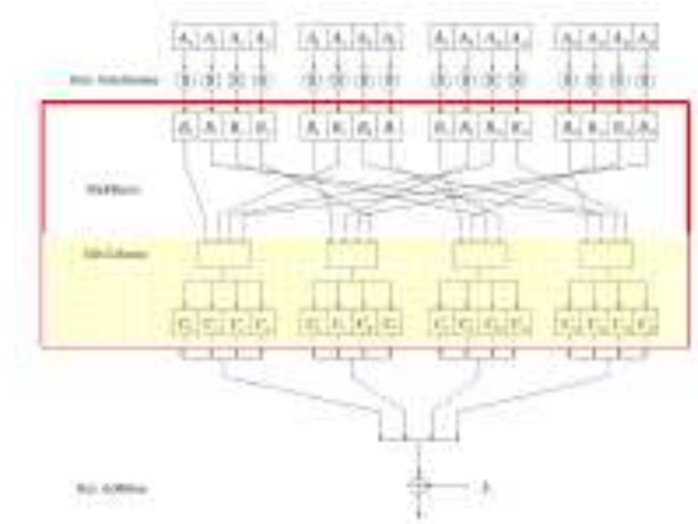
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

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

Step 5: Mix Column

- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Effectively a matrix multiplication in $GF(2^8)$ using prime poly $m(x) = x^8+x^4+x^3+x+1$



$$\begin{matrix} a\mathbf{x} + b\mathbf{y} + c\mathbf{z} + d\mathbf{t} \rightarrow \\ e\mathbf{x} + f\mathbf{y} + g\mathbf{z} + h\mathbf{t} \rightarrow \\ i\mathbf{x} + j\mathbf{y} + k\mathbf{z} + l\mathbf{t} \rightarrow \\ m\mathbf{x} + n\mathbf{y} + o\mathbf{z} + p\mathbf{t} \rightarrow \end{matrix} \begin{bmatrix} \blacksquare \\ \blacksquare \\ \blacksquare \\ \blacksquare \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \\ \mathbf{t} \end{bmatrix}$$

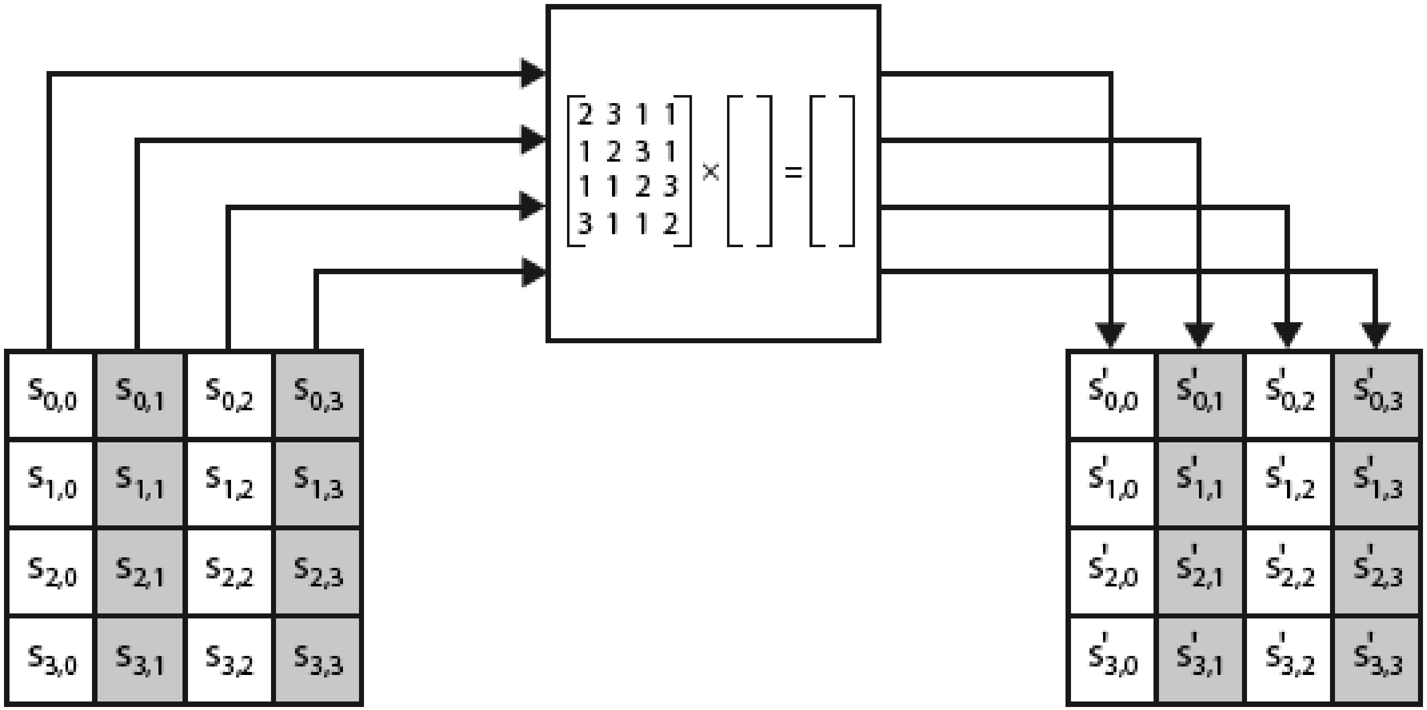
New matrix **Constant matrix** Old matrix

MixClumns Scheme

- Linear transformation which mixes each column of the state matrix
- Each 4-byte column is considered as a vector and multiplied by a fixed 4x4 matrix, e.g.,

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

MixClumns Scheme



- The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

MixColumn and InvMixColumn

- Constant multiplication matrix

$$\underset{C}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \xleftrightarrow{\text{Inverse}} \underset{C^{-1}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}}$$

MixClumns

- The first result byte is calculated by multiplying 4 values of the state column against 4 values of the first row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.

  b1 = (b1 * 2) XOR (b2*3) XOR (b3*1) XOR (b4*1)

- The second result byte is calculated by multiplying the same 4 values of the state column against 4 values of the second row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.

  b2 = (b1 * 1) XOR (b2*2) XOR (b3*3) XOR (b4*1)

- The third result byte is calculated by multiplying the same 4 values of the state column against 4 values of the third row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.

  b3 = (b1 * 1) XOR (b2*1) XOR (b3*2) XOR (b4*3)

- The fourth result byte is calculated by multiplying the same 4 values of the state column against 4 values of the fourth row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.

  b4 = (b1 * 3) XOR (b2*1) XOR (b3*1) XOR (b4*2)

- This procedure is repeated again with the next column of the state, until there are no more state columns

Multiplication Matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

16 byte State

b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15
b4	b8	b12	b16

MixClumns Scheme

Example

- Example assume we have state from ShiftRows as following

$$\text{State} \begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix}$$

- Multiplication of state with constant multiplication matrix

$$\begin{bmatrix} 63 & C9 & FE & 30 \\ F2 & F2 & 63 & 26 \\ C9 & C9 & 7D & D4 \\ FA & 63 & 82 & D4 \end{bmatrix} \text{x} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

AES Arithmetic in MixClumns Scheme

- First col with first row 63 * 02 + f2 * 03 + c9 * 01 + f1 * 01
- Convert the hexadecimal to binary
- 63 = 6 3 = 0110 0011
- F2 = f 2 = 1111 0010
- C9 = c 9 = 1100 1001
- F1 = f 1 = 1111 0001
- 63 * 02 = 0110 0011 * 02 (when the value multiplied by 02 look to the 1st bit from left if it zero then perform leftshift by 1 (i.e move 0 from left to right) 1100 0110 (Result 1)

AES Arithmetic in MixClumns Scheme

- f2 * 03 (03 means 02 + 01) → f2* 03 = f2 * (02+01) = f2*02 + f2* 01 = 1111 0010 * 02 + 1111 0010 * 01

  1111 0010 * 02 (check first bit from left is not zero it is one then two step is performed first remove the this bit (1st bit from left and insert zero in the right 1111 0010 become 1110 0100 Step two perform xor with 1B (0001 1011) 1110 0100 xor 0001 1011 = 1111 1111

  F2 * 01 = f2 = 1111 0010

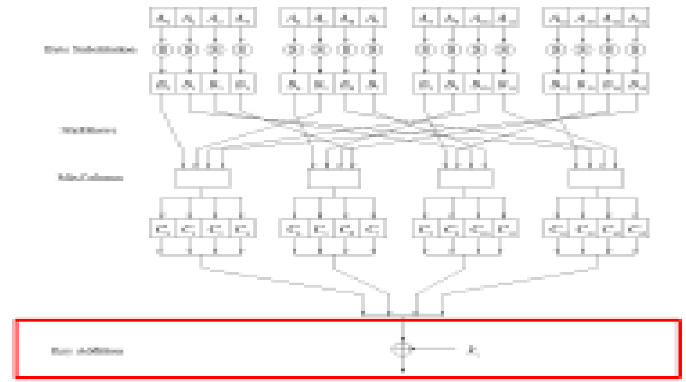
  So f2*03 = f2 * 2 + f2 * 01 = 1111 0010 * 02 + 1111 0010 * 01

  = 1111 1111 + 1111 0010 = 0000 1101 (Result 2)

AES Arithmetic in MixClumns Scheme

- 7D * 01 = 7D = 0111 1101 (Result 3)
- D4* 01 = D4 = 1101 0100 (Result 4)
- Perform Xor Between Result 1, Result 2, Result3 and Result 4
- 1100 0110 xor 0000 1101 xor 0111 1101 xor 1101 0100 = 0110 0010 = (62 Hex) this is first value in new state 1st row, 1st col
- In the same way compute second value, 3rd…….

Step 6: AddRoundKey



- Inputs:
  - 16-byte state matrix C
  - 16-byte subkey k_i
- Output: C $\oplus$ k_i
- XOR state with 128-bits of the round key
- AddRoundKey proceeds one column at a time.
  - adds a round key word with each state column matrix
  - the operation is matrix addition
- Inverse for decryption identical
  - since XOR own inverse, with reversed keys
- Designed to be as simple as possible
- The steps are repeated 10 times
- The subkeys are generated in the key schedule

Example

- Given the plaintext [0809 0A0B 0C0D 0E0F 0001 0203 0405 0607] and the key in hexadecimal [1010 1010 1010 1010 1010 1010 1010 1010].
  - Show the contents of **initial state**, displayed as a 4x4 matrix.
  - Show the contents of the first column of the first value of state after initial **AddRoundKey**.
  - Assume the state is now as follows: show the value of State after **SubBytes.**
  - Show the value of State after **ShiftRows** applied to the previous state

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

Show the contents of **initial state**, displayed as a 4x4 matrix.

State= $\begin{bmatrix} 08 & 0C & 00 & 04 \\ 09 & 0D & 01 & 05 \\ 0A & 0E & 02 & 06 \\ 0B & 0F & 03 & 07 \end{bmatrix}$ Key = $\begin{bmatrix} 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \end{bmatrix}$

Show the contents of the first column of the first value of state after initial **AddRoundKey**.

$$\begin{bmatrix} 08 & 0C & 00 & 04 \\ 09 & 0D & 01 & 05 \\ 0A & 0E & 02 & 06 \\ 0B & 0F & 03 & 07 \end{bmatrix} \oplus \begin{bmatrix} 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 \end{bmatrix} = \begin{bmatrix} 18 & 1C & 10 & 14 \\ 19 & 1D & 11 & 15 \\ 1A & 1E & 12 & 16 \\ 1B & 1F & 13 & 17 \end{bmatrix}$$

show the value of State after **SubBytes.**

$$\begin{bmatrix} AD & 9C & CA & FA \\ D4 & A4 & 82 & 59 \\ A2 & 72 & C9 & 47 \\ AF & CD & 7D & FD \end{bmatrix}$$

Show the value of State after **ShiftRows** applied to the previous state

$$\begin{bmatrix} AD & 9C & CA & FA \\ A4 & 82 & 59 & D4 \\ C9 & 47 & A2 & 72 \\ FD & AF & CD & 7D \end{bmatrix}$$

- Compare the AES to DES. For each of the following elementst of DES, indicate the comparable element in AES or explain why it is not needed in AES.
  - XOR of subkey material with the input to the function f function.
  - XOR of the f function output with left side of the block.
  - The f function.
  - Permutation P.
  - Swapping of halves of the block.

- **XOR of subkey material with the input to the function f function.**
  - The similar element in AES for XOR of subkey with the input to the function (that passes different stages before XORing) is the added round key stage in all the 10 rounds.
- **XOR of the f function output with left side of the block.**
  - There is no similar element in AES for XOR the f function output with left half side of the block, this is because AES structure is not a feistel structure. The entire block is processed in parallel (No two halves are using one half to modify the other half).
- **The f function.**
  - There is no single element that is similar to f function, but the four stages (Substitution bytes, shift rows, mix columns, added roundly) in each round do the same as f function.
- **Permutation P.**
  - The similar element for **P** is the shift rows in each of the 10 rounds.
- **Swapping of halves of the block.**
  - No similar element in AES this is because that AES structure not a feistel structure and no need to swap halves since work in parallel (No half needs to modify the other half).

AES Key Scheduling

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

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.
- Subkeys are derived recursively from the original 128/192/256-bit input key
- Each round has 1 subkey, plus 1 subkey at the beginning of AES

Key length (bits)	Number of subkeys
128	11
192	13
256	15

- Key whitening: Subkey is used both at the input and output of AES $\Rightarrow$ # subkeys = # rounds + 1

Key Schedule

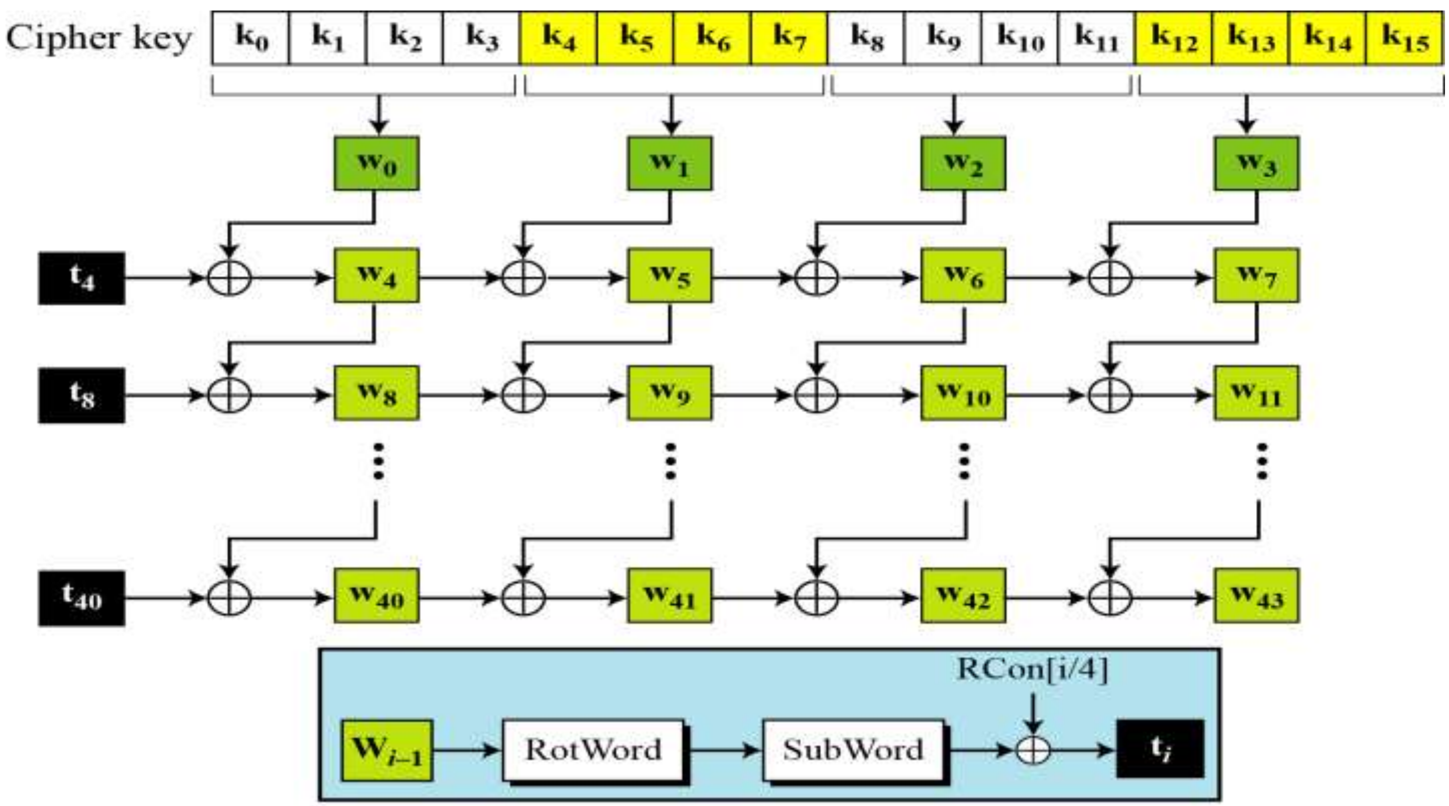
- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words

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	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$

Figure : Words for each round

Key Schedule

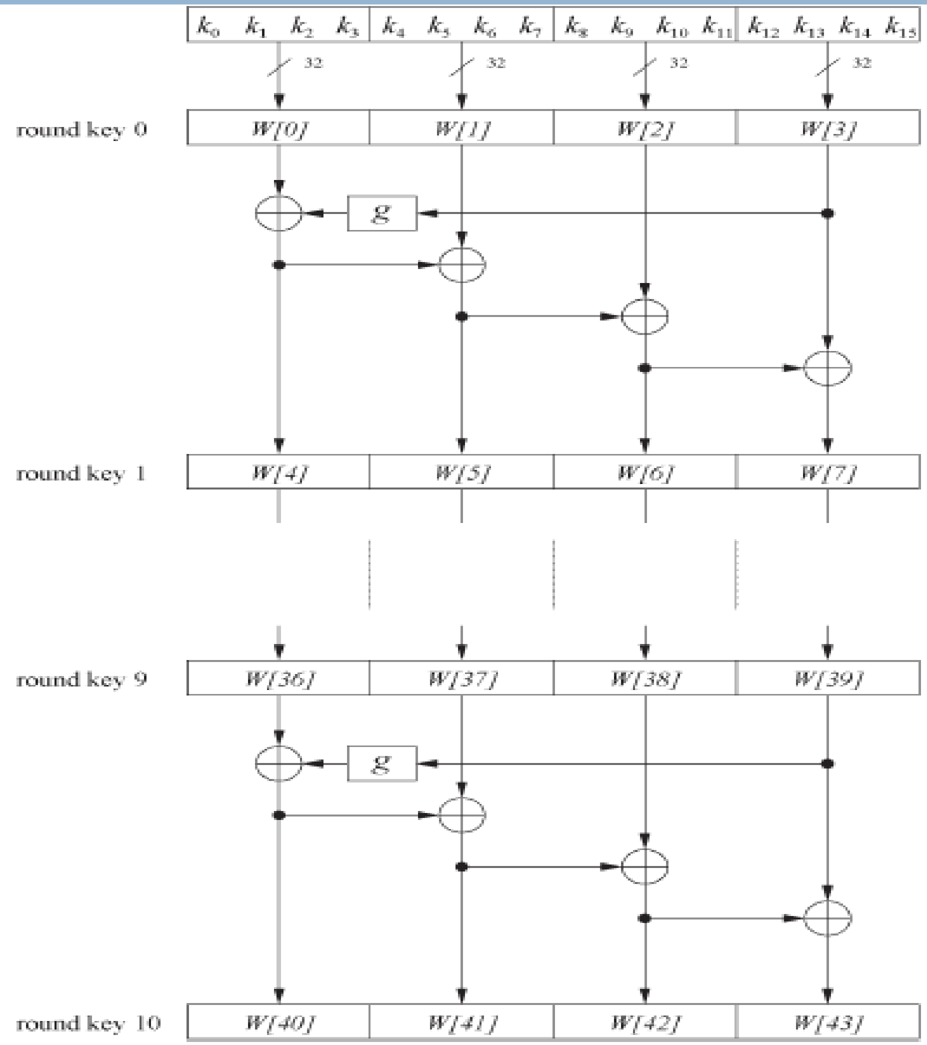
- Figure 7.16 *Key expansion in AES*



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

Key Schedule

- Example: Key schedule for 128-bit key AES
- Word-oriented: 1 word = 32 bits
- 11 subkeys are stored in *W[0]…W[3],W[4]…W[7], … ,W[40]…W[43]*
- First subkey *W[0]…W[3]* is the original AES key
- Function *g* rotates its four input bytes and performs a bytewise S-Box substitution ⇒ nonlinearity



Key Schedule

- The round coefficient *RC* is only added to the leftmost byte and varies from round to round:

$$RC[1] = x^0 = (00000001)_2$$
$$RC[2] = x^1 = (00000010)_2$$
$$RC[3] = x^2 = (00000100)_2$$
...
$$RC[10] = x^9 = (00110110)_2$$

- x^i represents an element in a Galois field

Key Expansion submodule

- **RotWord** performs a one byte circular left shift on a word For example:

  RotWord[b0,b1,b2,b3] = [b1,b2,b3,b0]

- **SubWord** performs a byte substitution on each byte of input word using the S-box

- **SubWord(RotWord(temp))** is XORed with RCon[j] – the round constant

Round Constant (RCon)

- RCON is a word in which the three rightmost bytes are zero
- It is different for each round and defined as:

  RCon[j] = (RCon[j],0,0,0)

  where RCon[1] =1 , RCon[j] = 2 * RCon[j-1]
- Multiplication is defined over GF(2^8) but can be implement in Table Lookup

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

AES Key Expansion

- takes 128-bits (16-bytes) key and expands into array of 44 32-bit words
- Key is a set of words, each word = 32 bits. For example assume the key is

0f 15 71 c9	47 d9 e8 59	0c b7 ad df	af 7f 67 98
w0	w1	w2	w3

- To perform key Expansion, there are two rules:
  - **Rule 1**→ K[n] : W[i] = K[n-1]: W[i] xor K[n]: w[i-1]
  - **Rule 2** → K[n]: W0 = K[n-1]: W0 xor SubByte (K[n-1] : W3 >>8) xor Rcon[n]
- Rule 2 is used to compute k[n] for W0 **only**
- Rule 1 is used to compute k[n] for W1, W2, W3
- K1:W1➔ read as k1 for W1 or W1 for k1

AES Key Expansion

- To find K1 we do the following
  - find K1: W0, we using Rule 2,
    - K1: W0 = K0:W0 Xor SubByte(K0: W3>>8) Xor Rcon[1]
  - find K1:W1, we using Rule 1,
    - K1:W1 = K0:W1 Xor K1:W0
  - Find K1:W2, we using Rule 1,
    - K1:W2 = K0:W2 Xor K1:W1
  - Find K1: W3, we using Rule 1,
    - K1: W3 = K0:W3 Xor K1:w2
- To Find K2 we find the following
  - K2: W0 = K1:W0 Xor SubByte(K1:W3>>8) Xor Rcon[2]
  - K2:W1 = K1:W1 Xor K2:W0
  - K2:W2 ---- and K2:W3 as so on

AES Key Expansion

- Example Assume the key is :

  0f 15 71 c9 47 d9 e8 59 0c b7 ad df af 7f 67 98

- First the key0 (K0) is

0f 15 71 c9	K0:W0
47 d9 e8 59	K0:W1
0c b7 ad df	K0:W2
af 7f 67 98	K0:W3

To find k1 for W0, W1, W2, W3 the rule 1 and rule 2 will be used:

K1:W0 = K0:W0 Xor SubByte (K0:W3>>8) Xor Rcon[1]

K1:W0 = 0f 15 71 c9 Xor SubByte(af 7f 67 98 >>8) Xor Rcon[1]

AES Key Expansion

K1:W0 =

0f 15 71 c9 Xor SubByte(af 7f 67 98 >>8) Xor Rcon[1]

(i.e >>8 means move the first 8bit from left and replace it in right this mean the first of 2 hex digit is moved right)

K1:W0 = 0f 15 71 c9 xor SubByte(7f 67 98 af) Xor Rcon[1]

(use subByte tale which used before in encryption)

K1:W0 =

0f 15 71 c9 Xor d2 85 46 af Xor Rcon[1]

(Rcon use constant Rcon table to find the value when I =1; Rcon[1] = 01000000)

K1:W0 =

0f 15 71 c9 Xor d2 85 46 af XOR 01000000

Convert the hex to binary and perform XOR the result is df 90 37 b0

K1: W0 = dc 90 37 b0

AES Key Expansion

Then find K1:W1, K1:W2 and K1:W3

K1:W1 = K0:W1 Xor K1:W0

= 47 d9 cd 59 xor df 90 37 b0 = 98 49 df c9

K1:W2 = K0:W2 Xor K1:W1

K1:W3 = K0. W3 Xor K1:W2

Homework

- Given the plaintext [0001 0203 0405 0607 0809 0A0B 0C0D 0E0F] and the key [0101 0101 0101 0101 0101 0101 01010101]
  a) Show the original contents of state, displayed as a 4x4 matrix.
  b) Show the value of state after initial AddRoundKey.
  c) Show the value of State after SubBytes.
  d) Show the value of State after ShiftRows.
  e) Show the value of State after MixColumns.

Summary

1) AES encrypts 128 bit blocks with 128-bit, 192-bit or 256-bit keys using 10, 12, or 14 rounds, respectively.
2) Is not a Feistel cipher →All 128 bits are encrypted
3) Each round = 4 steps of SubBytes, ShiftRows, MixColumns, and AddRoundKey.
4) Last round has only 3 steps. No MixColumns.
5) Decryption is not the same as encryption (as in DES). Decryption consists of inverse steps.