National Chung Cheng University
Department of Information Management

Advanced Encryption Standard

Pei-Ju (Julian) Lee

National Chung Cheng University
Information Security
pjlee@mis.ccu.edu.tw

Fall, 2016

Advanced Encryption Standard (AES)

- The AES was published by the NIST in 2001
- AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications
  - Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex
- All operations are performed on 8-bit bytes
- All arithmetic operations (addition, multiplication, and division) are performed over GF(2^8)

  - AES is based on the Rijndael cipher that developed by Belgian cryptographers, Joan Daemen and Vincent Rijmen

Review: Finite Field Arithmetic

- 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, ... , p - 1}, where *p* is a prime number and in which arithmetic is carried out modulo *p*

Cont'd

If one of the operations used in the algorithm is division, then we need to work in arithmetic defined over a field

- Division requires that each nonzero element have a multiplicative inverse

For convenience we would like to work with integers that fit exactly into a given number of bits with no wasted bit patterns

- Integers in the range 0 through $2^n - 1$, which fit into an *n*-bit word

The set of such integers, Z_{2^n}, using modular arithmetic, is not a field

- For example, the integer 2 has no multiplicative inverse in Z_{2^n}, that is, there is no integer *b*, such that $2b \bmod 2^n = 1$

A finite field containing 2^n elements is referred to as $GF(2^n)$

- Every polynomial in $GF(2^n)$ can be represented by an n-bit number

Cont'd

- A polynomial in GF(2^n) can be uniquely represented by its n binary coefficients $(a_{n-1}a_{n-2}\ldots a_0)$. Therefore, every polynomial in GF(2^n) can be represented by an n-bit number.
- Addition is performed by taking the bitwise XOR of the two n-bit elements.
- Multiplication of two bytes is defined as multiplication in the finite field GF(2^8), with the irreducible polynomial m(x).
  - Example

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

$$(a_6 \ldots a_1 a_0 0) \oplus (00011011)$$

Review: Make *S* a Finite Field

- With the appropriate definition of arithmetic operations, each such set *S* is a finite field:

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.
2. Arithmetic on the coefficients is performed modulo p. That is, we use the rules of arithmetic for the finite field Z_p.
3. If multiplication results in a polynomial of degree greater than $n - 1$, then the polynomial is reduced modulo some irreducible polynomial $m(x)$ of degree n. That is, we divide by $m(x)$ and keep the remainder. For a polynomial $f(x)$, the remainder is expressed as $r(x) = f(x) \bmod m(x)$.

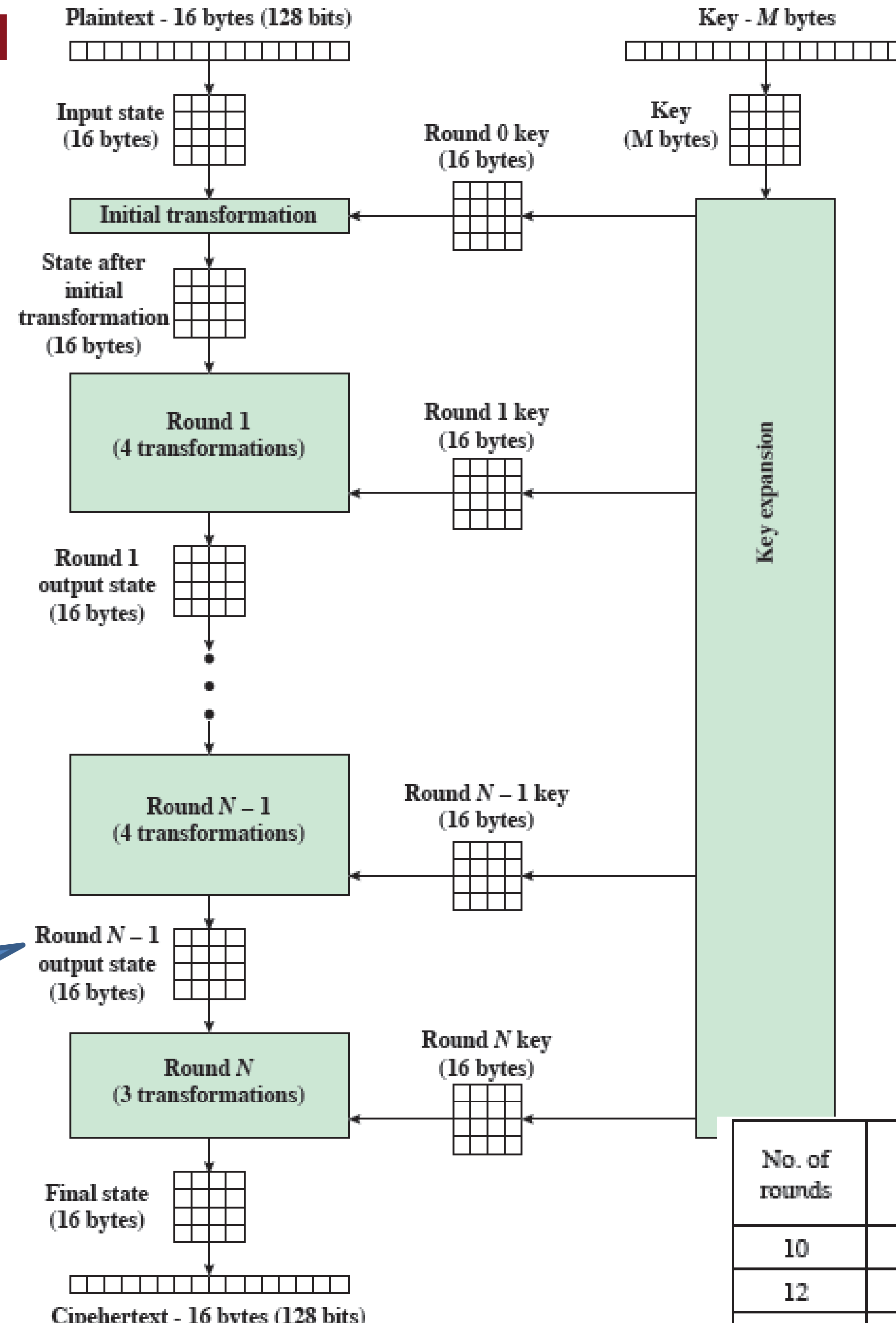
- To construct the finite field GF(2^n) so we can:
  - Represent integers/polynomials using n-bit number
  - Performing any operation without leaving the set
- Therefore, we can use:
  - The irreducible/prime polynomial
  - Find the multiplicative inverse

AES Encryption Process

- *P* block size: 128bits(16 Bytes)
- Key length: 16, 24, or 32 Bytes (128, 192, or 256 bits)
  - AES-128, AES-192, or AES-256

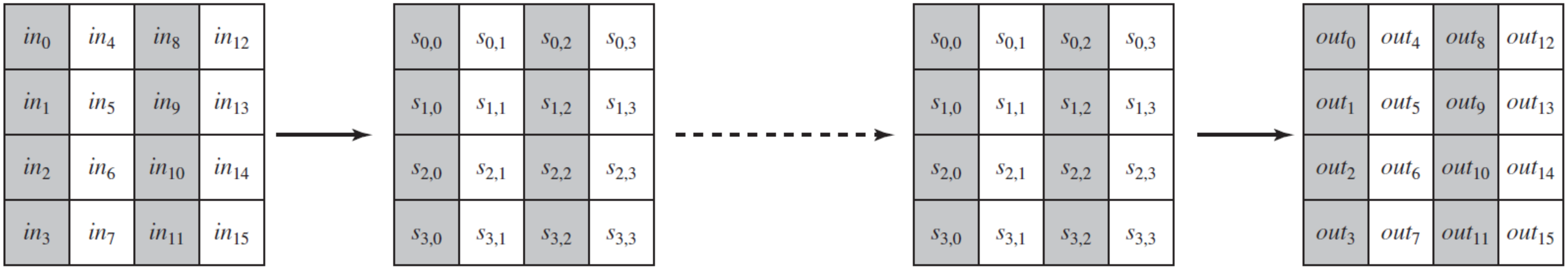
State array: modified at each stage of E or D

  - Number of rounds depends on the key length



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

AES Data Structures



(a) Input, state array, and output

(b) Key and expanded key

- 128-bits key / 16Bytes
  -> 44 words for 10 rounds
  (1 word is for 4 Bytes)

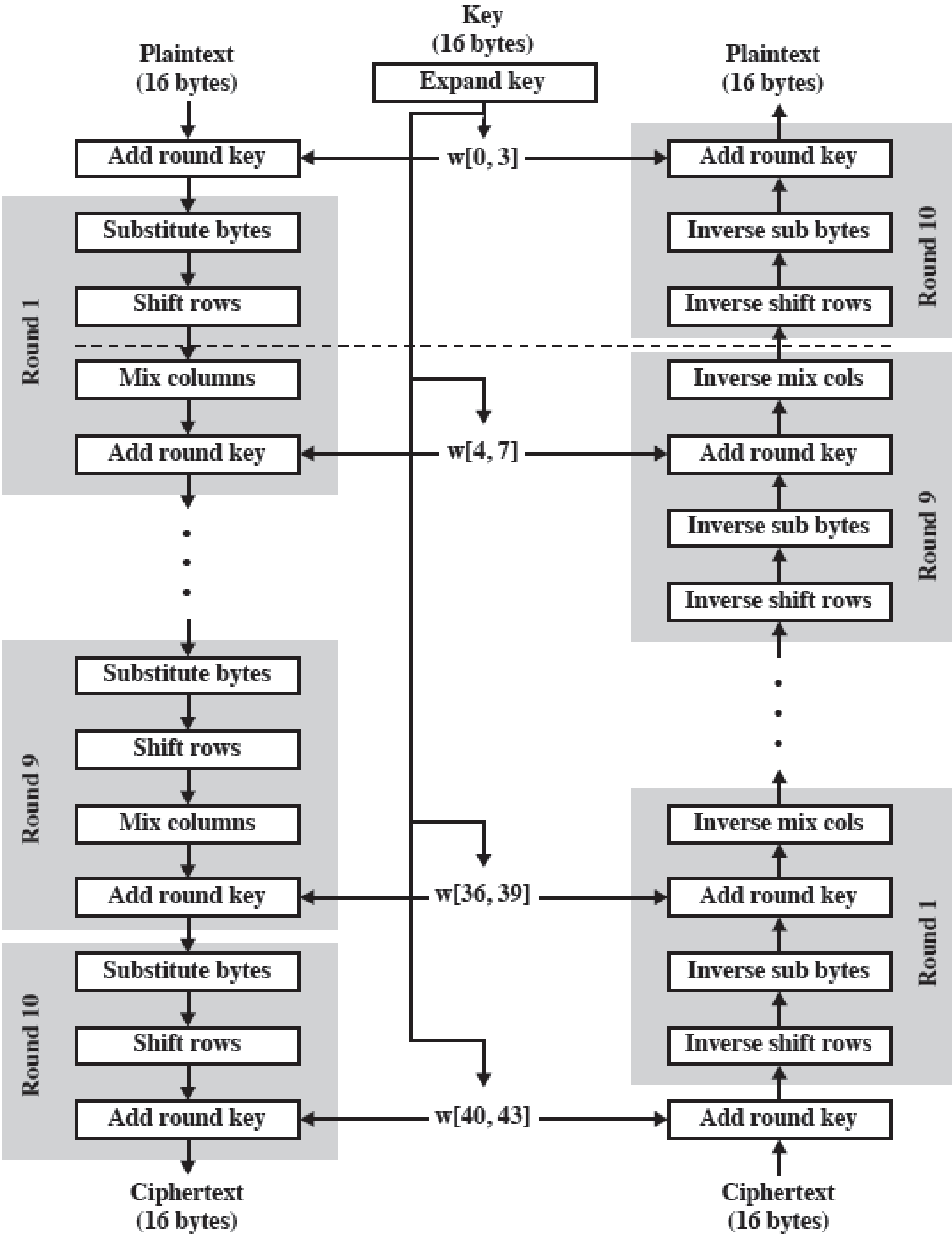
AES Parameters

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

AES Encryption and Decryption

- Initial transformation/Round0: AddRoundKey
- The first N-1 round consists of 4 transformation functions: SubBytes, ShiftRows, MixColumns, AddRoundKey
- Final round: SubBytes, ShiftRows, AddRoundKey

Each transformation takes one or more 4 * 4 matrices as input and produces a 4 * 4 matrix as output.



(a) Encryption (b) Decryption

Detailed Structure

- 1. Processes the entire data block as a single matrix during each round using substitutions and permutation (Feistel: half)
- 2. The key is expanded into an array of 44 32-bit words, w[i]. 4 words serve as a round key.
- 3. Four stages transformation: (1 of permutation, 3 of substitution)
  - Substitute bytes – uses an S-box to perform a byte-by-byte substitution of the block
  - ShiftRows – a simple permutation
  - MixColumns – a substitution that makes use of arithmetic over $GF(2^8)$
  - AddRoundKey – a simple bitwise XOR of the current block with a portion of the expanded key
- 4. The cipher begins and ends with an AddRoundKey stage (the only stage makes use the key, other stages would add no security)
- 5. Each stage is easily reversible

Cont’d

- 6. **State** is the same for both encryption and decryption(dash line in the fig.)
- 7. Each stage is easily reversible
- 8. The decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm
- 9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. At each horizontal point of encryption and decryption, State is the same for both encryption and decryption.
- 10. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

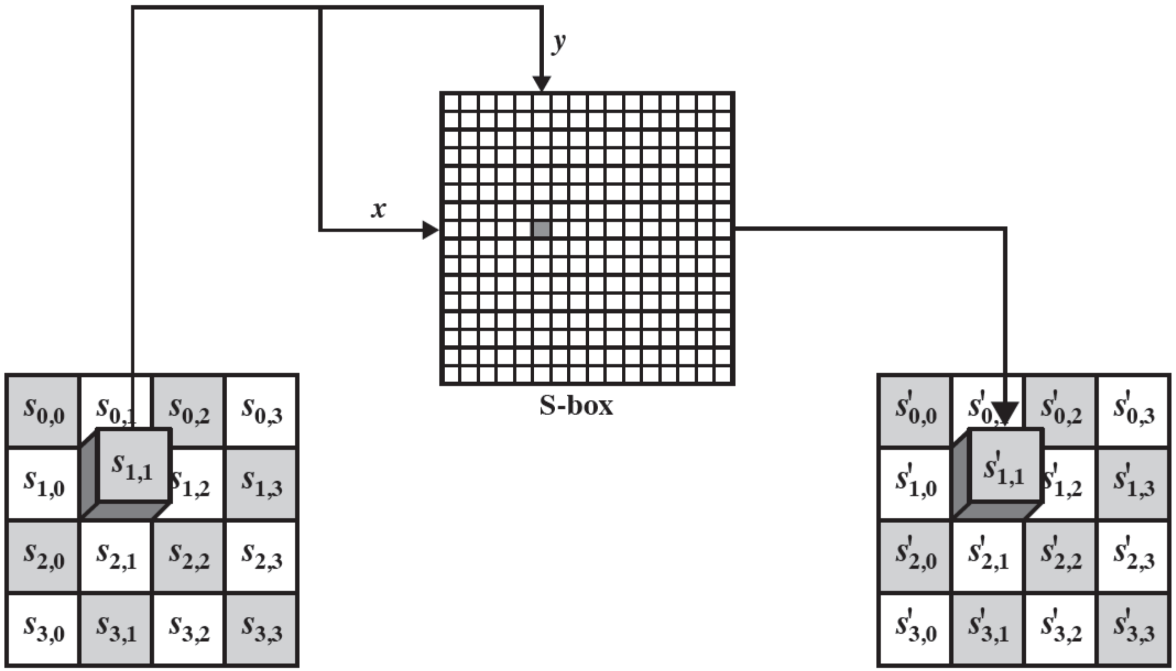
AES Transformation Function

- A discussion of each of the four transformations used in AES.
- For each stage, describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage.
  - Substitute Byte Transformation
  - ShiftRows Transformation
  - MixColumns Transformation
  - AddRoundKey Transformation

1.Substitute Byte Transformation

- The forward substitute byte transformation , called SubBytes

- Each individual byte of State is mapped into a new byte using a 16 * 16 matrix of byte values of S-box
- Ex.
  $S_{1,1}$ = {95} (Hexadecimal)
  ->S-box: row 9, col 5
  -> $S'_{1,1}${2A}



(a) Substitute byte transformation

S-box

- Ex. value {95} is mapped into the value {2A}.

		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

Inverse S-box

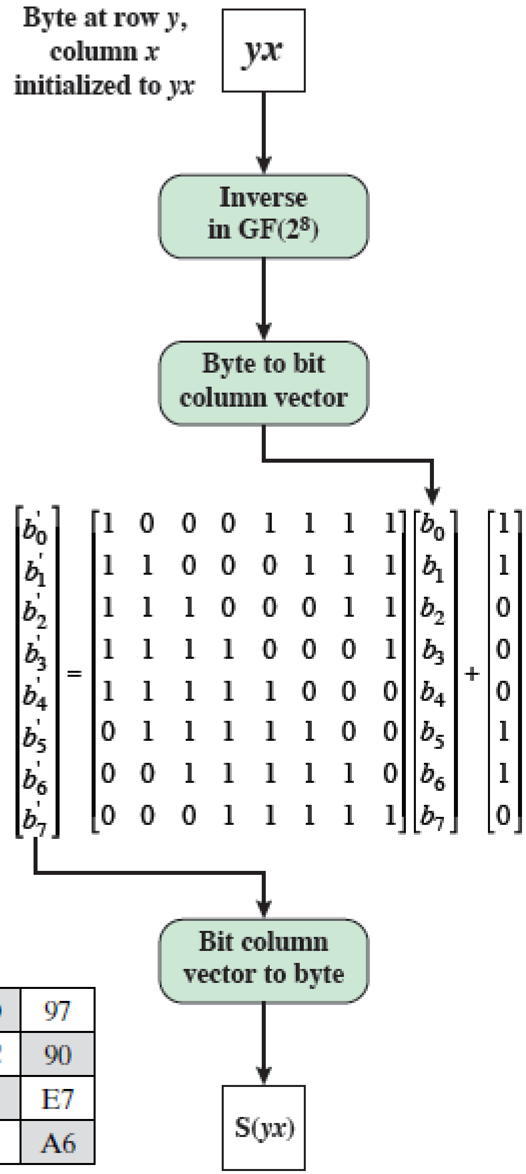
- The inverse substitute byte transformation , called InvSubBytes, makes use of the inverse S-box

		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

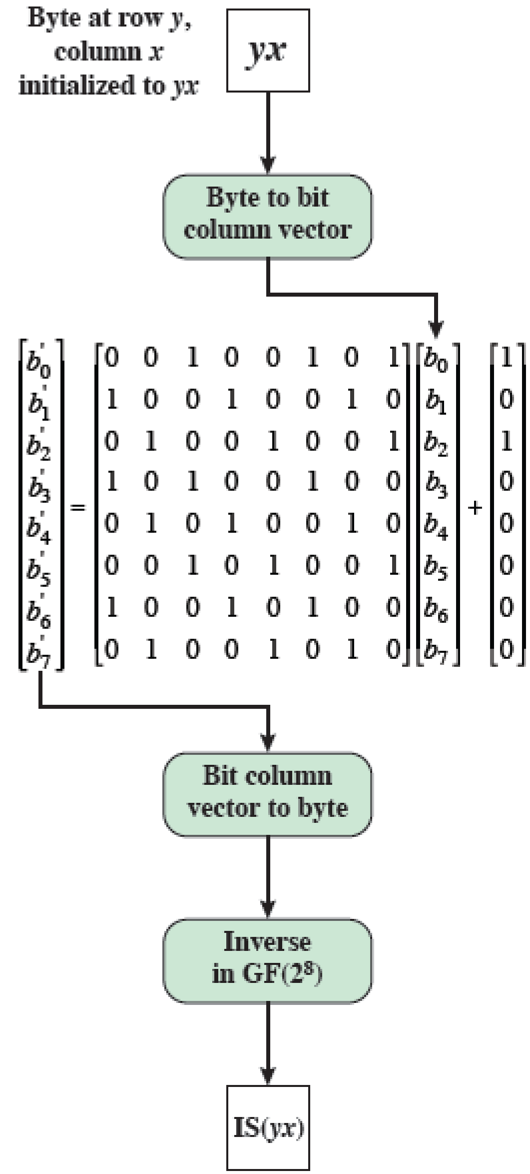
S-box

- The S-box is designed to be resistant to known cryptanalytic attacks
  - Low correlation between input bits and output bits
  - It is invertible but does not self-inverse
  - The nonlinearity is due to the use of the multiplicative inverse
  - Ex. S-box({95}) = {2A} IS-box({95}) = {AD}

Construction of S-Box and IS-Box



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



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

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

The Construction of S-box

- 1. Initialize the S-box with the byte values in ascending sequence row by row.
- 2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8).
- 3. Transform each bit of each byte in the S-box:

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

Bitwise XOR

  – c_i is the ith bit of byte c with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0)$ =(01100011).
  – Matrix form of this transformation:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Example

- The input value {95}
  - The multiplicative inverse in GF(2^8) is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

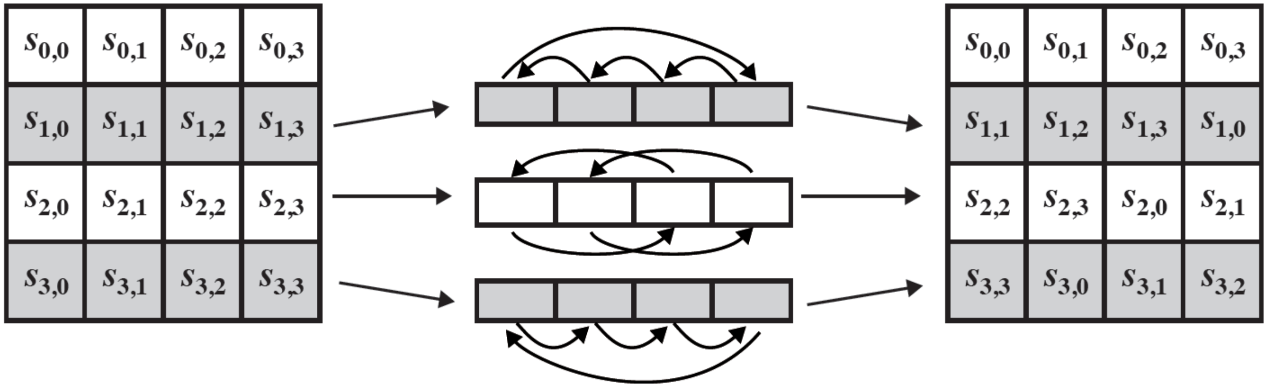
  - The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking Table 5.2a.

Rationale

- The S-box is designed to be resistant to known cryptanalytic attacks
  - a design that has a low correlation between input bits and output bits
  - the output is not a linear function of the input
    - The nonlinearity is due to the use of the multiplicative inverse. In addition, the constant in Equation (5.1) was chosen so that the S-box has no fixed points [S-box(a) = a] and no “opposite fixed points” [S-box(a) = ā], where ā is the bitwise complement of a
- The S-box must be invertible, that is, IS-box[S-box(a)] = a. However, the S-box does not self-inverse in the sense that it is not true that S-box(a) = IS-box(a).
  - Ex. S-box({95}) = {2A}, but IS-box({95}) = {AD}.

2. Shift Row Transformation

- Forward: ShiftRow
- Inverse: InvShiftRow



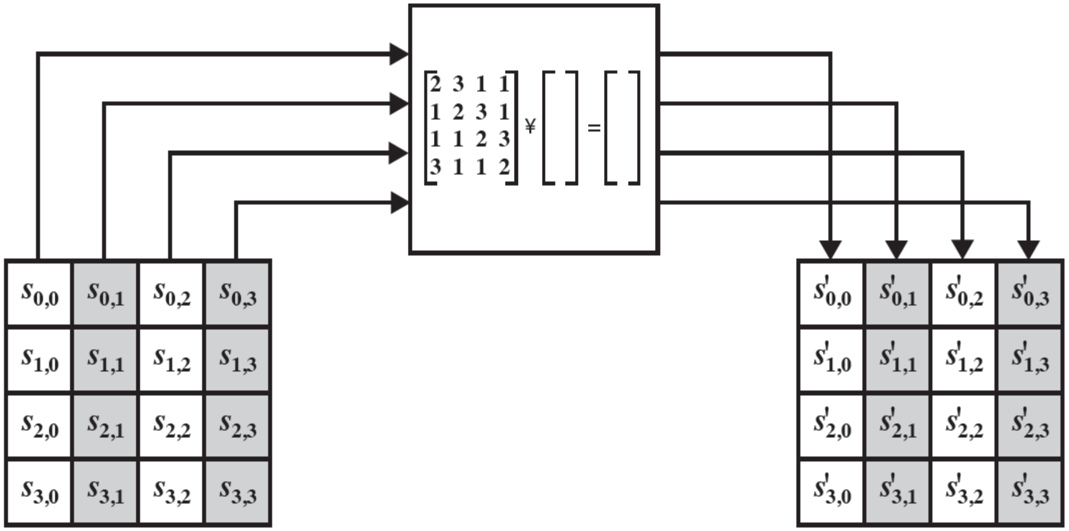
- The 1st row of State is not altered.
  For the 2nd/3rd/4th row, a 1/2/3 byte circular left shift is performed
- On encryption, the first 4 bytes of the plaintext are copied to the 1st column of State, and so on
- The round key is applied to State column by column
- Thus, a row shift equal to a linear distance of a multiple of 4 bytes
- Transformation ensures that the 4 bytes of one column are spread out to four different columns

Rationale

- On encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on.
- Furthermore, the round key is applied to **State** column by column.
- Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes.
- Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns.

3. MixColumn Transformation

- Forward: MixColumns
- Inverse: InvMixColumns



- 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
- Coefficients of a matrix based on a linear code with maximal distance between code words ensures a good mixing among the bytes of each column

Cont'd

- Matrix multiplication:

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

  - Single column transformation

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

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

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

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

Cont'd

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

$\rightarrow$

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

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

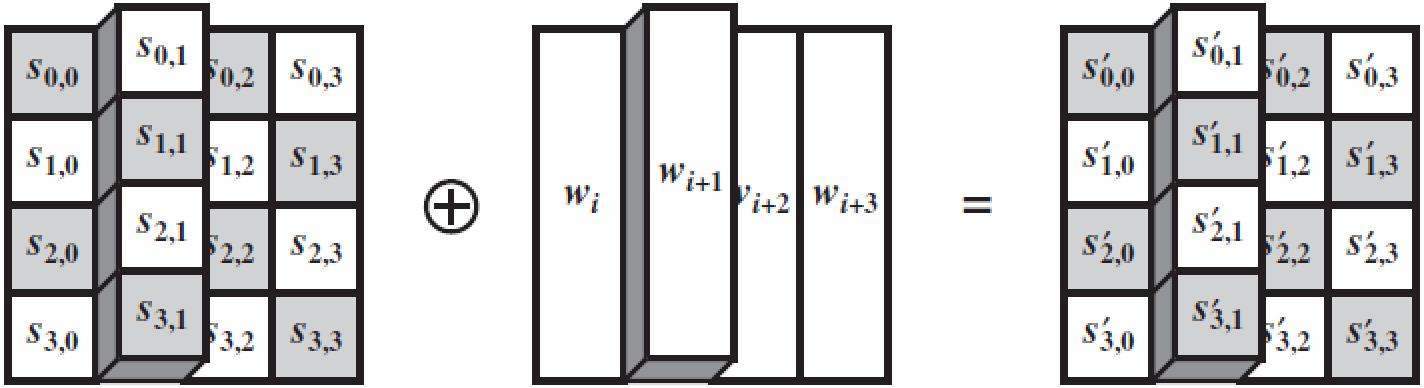
$$\begin{aligned}
\{02\} \cdot \{87\} &= 0001\ 0101 \\
\{03\} \cdot \{6E\} &= 1011\ 0010 \\
\{46\} &= 0100\ 0110 \\
\{A6\} &= \underline{1010\ 0110} \\
& \ 0100\ 0111 = \{47\}
\end{aligned}$$

Rationale

- The coefficients of the matrix are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column.
  - The mix column transformation combined with the shift row transformation ensures that after a few rounds all output bits depend on all input bits.
  - In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations.
    - Multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement.

4. AddRoundKey Transformation

- The 128 bits of State are bitwise XORed with the 128 bits of the round key
- Operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key



- Can also be viewed as a byte-level operation

Rationale

- The add round key transformation is as simple as possible and affects every bit of **State**.
- The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

Numerical Example

- SubByte

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

- 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

- 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

- AddRoundKey

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

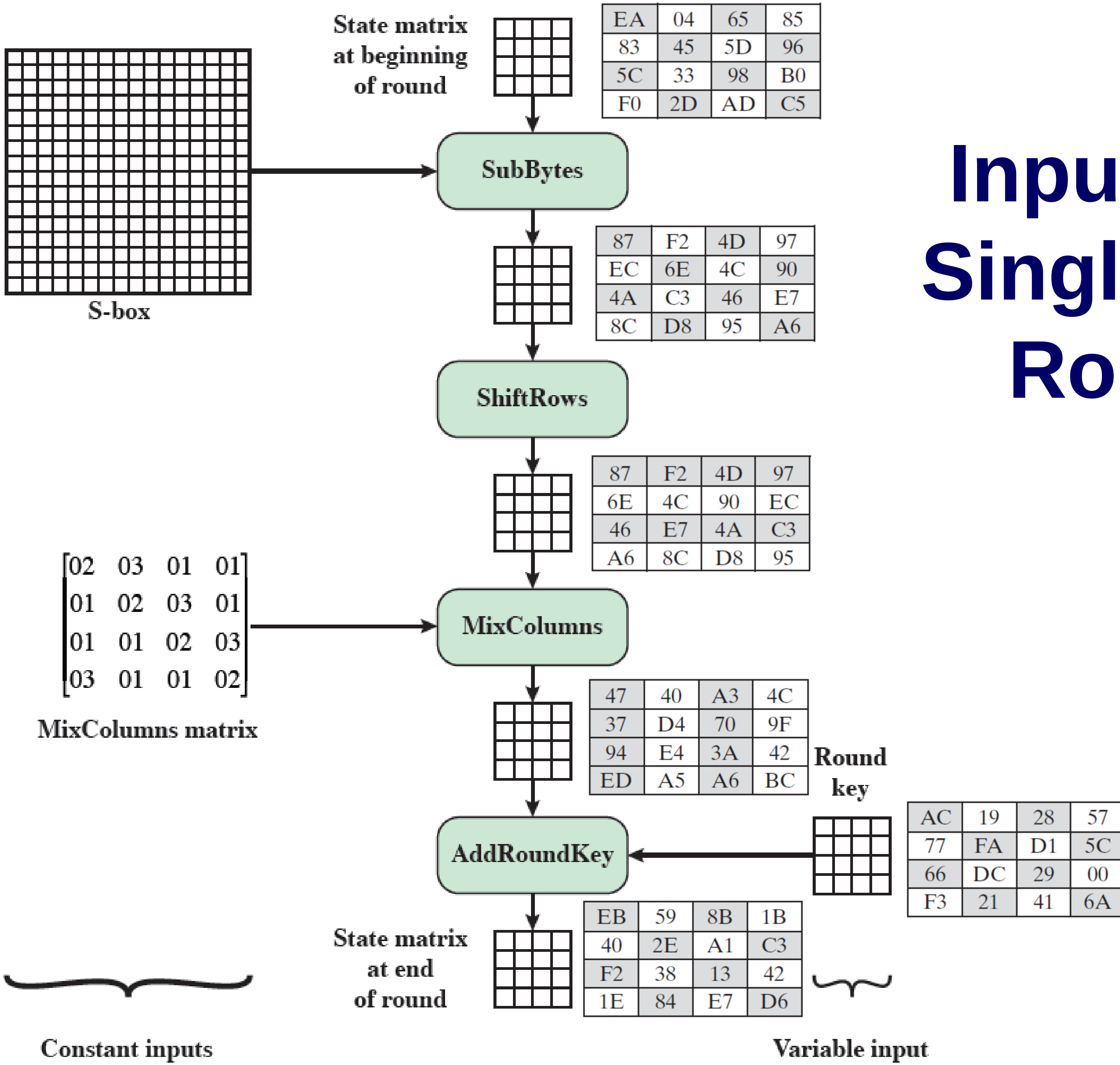
$\oplus$

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

Inputs for Single AES Round



State matrix at beginning of round:

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

After SubBytes:

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

After ShiftRows:

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

MixColumns matrix:

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

After MixColumns:

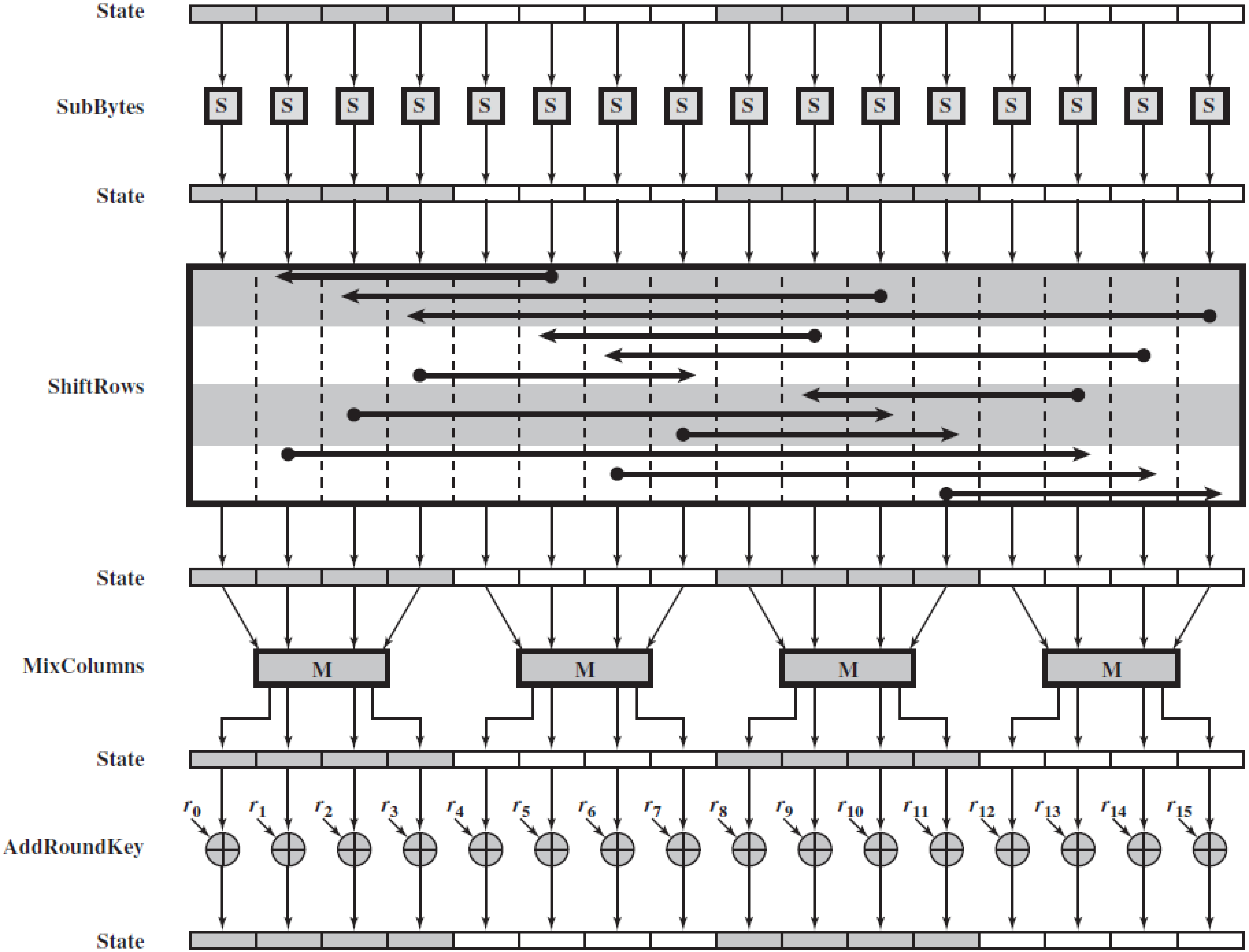
47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Round key:

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

State matrix at end of round:

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

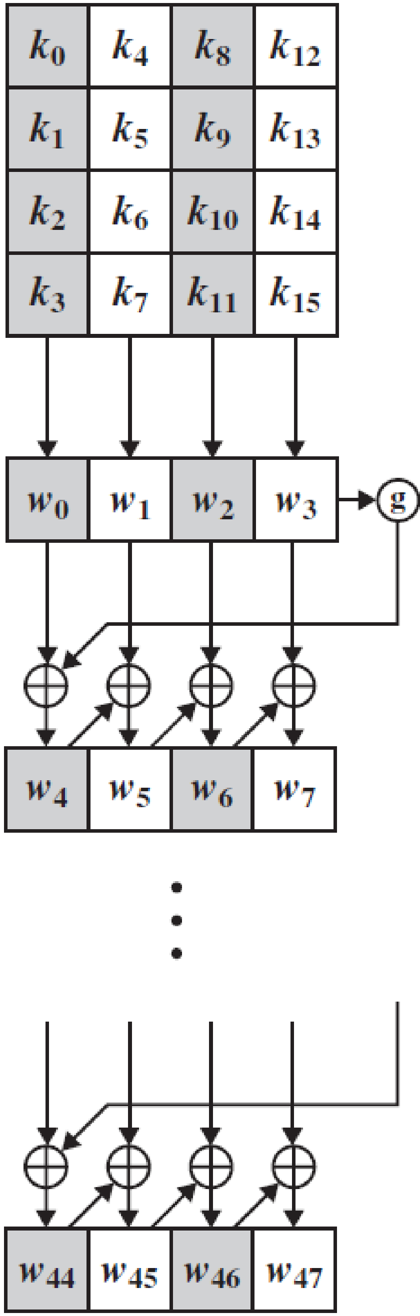

State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
AddRoundKey
State

AES Key Expansion

- The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks.
- Takes as input a 4-word (16 byte) key and produces a linear array of 44 words (176) bytes
  – This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher

Cont'd

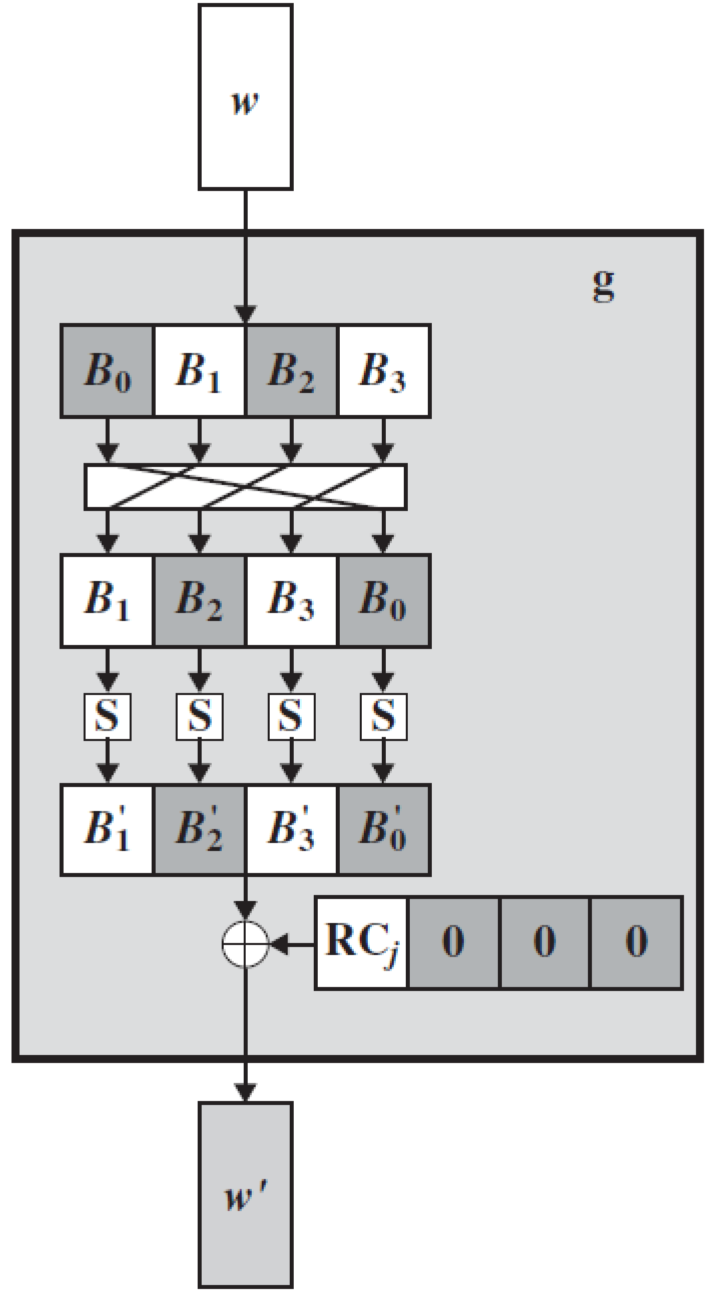
- Key is copied into the 1st four words of the expanded key
  – The remainder of the expanded key is filled in four words at a time
- Each added word w[i] depends on the immediately preceding word, w[i – 1], and the word four positions back, w[i – 4]
- Ex. w[4] = w[3] + w[0]
  – In three out of four cases a simple XOR is used
  – For a word whose position in the w array is a multiple of 4, a more complex function is used (i.e. Function g)



(a) Overall algorithm

Function g

- RotWord: 1-byte left shift on a word
- SubWord: substitute each byte by S-box
- XOR the RotWord and SubWord with a round constant Rcon[j]



(b) Function g

Round Constant

- The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.
- Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 · RC[j-1] and with multiplication defined over the field GF(2^8).
- The values of RC[j] in hexadecimal are

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

Example

- Suppose the round key for round 8 is:

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

    - Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

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

Rationale

- The expansion key algorithm is designed to be resistant to known cryptanalytic attacks.
- The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. The specific criteria that were used are [DAEM99] (next slide)

Criteria

The specific criteria that were used are:

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits
- An invertible transformation
- Speed on a wide range of processors
- Usage of round constants to eliminate symmetries
- Diffusion of cipher key differences into the round keys (i.e. each key bit affects many round key bits)
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
- Simplicity of description

An AES example

Plaintext:	0123456789abcdeffedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	ff0b844a0853bf7c6934ab4364148fb9

Table 5.4 AES Example

Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key
01 89 fe 76 23 ab dc 54 45 cd ba 32 67 ef 98 10				0f 47 0c af 15 d9 b7 7f 71 e8 ad 67 c9 59 d6 98
0e ce f2 d9 36 72 6b 2b 34 25 17 55 ae b6 4e 88	ab 8b 89 35 05 40 7f f1 18 3f f0 fc e4 4e 2f c4	ab 8b 89 35 40 7f f1 05 f0 fc 18 3f c4 e4 4e 2f	b9 94 57 75 e4 8e 16 51 47 20 9a 3f c5 d6 f5 3b	dc 9b 97 38 90 49 fe 81 37 df 72 15 b0 e9 3f a7
65 0f c0 4d 74 c7 e8 d0 70 ff e8 2a 75 3f ca 9c	4d 76 ba e3 92 c6 9b 70 51 16 9b e5 9d 75 74 de	4d 76 ba e3 c6 9b 70 92 9b e5 51 16 de 9d 75 74	8e 22 db 12 b2 f2 dc 92 df 80 f7 c1 2d c5 1e 52	d2 49 de e6 c9 80 7e ff 6b b4 c6 d3 b7 5e 61 c6
5c 6b 05 f4 7b 72 a2 6d b4 34 31 12 9a 9b 7f 94	4a 7f 6b bf 21 40 3a 3c 8d 18 c7 c9 b8 14 d2 22	4a 7f 6b bf 40 3a 3c 21 c7 c9 8d 18 22 b8 14 d2	b1 c1 0b cc ba f3 8b 07 f9 1f 6a c3 1d 19 24 5c	c0 89 57 b1 af 2f 51 ae df 6b ad 7e 39 67 06 c0
71 48 5c 7d 15 dc da a9 26 74 c7 bd 24 7e 22 9c	a3 52 4a ff 59 86 57 d3 f7 92 c6 7a 36 f3 93 de	a3 52 4a ff 86 57 d3 59 c6 7a f7 92 de 36 f3 93	d4 11 fe 0f 3b 44 06 73 cb ab 62 37 19 b7 07 ec	2c a5 f2 43 5c 73 22 8c 65 0e a3 dd f1 96 90 50
f8 b4 0c 4c 67 37 24 ff ae a5 c1 ea e8 21 97 bc	41 8d fe 29 85 9a 36 16 e4 06 78 87 9b fd 88 65	41 8d fe 29 9a 36 16 85 78 87 e4 06 65 9b fd 88	2a 47 c4 48 83 e8 18 ba 84 18 27 23 eb 10 0a f3	58 fd 0f 4c 9d ee cc 40 36 38 9b 46 eb 7d ed bd
72 ba cb 04 1e 06 d4 fa b2 20 bc 65 00 6d e7 4e	40 f4 1f f2 72 6f 48 2d 37 b7 65 4d 63 3c 94 2f	40 f4 1f f2 6f 48 2d 72 65 4d 37 b7 2f 63 3c 94	7b 05 42 4a 1e d0 20 40 94 83 18 52 94 c4 43 fb	71 8c 83 cf c7 29 e5 a5 4c 74 ef a9 c2 bf 52 ef
0a 89 c1 85 d9 f9 c5 e5 d8 f7 f7 fb 56 7b 11 14	67 a7 78 97 35 99 a6 d9 61 68 68 0f b1 21 82 fa	67 a7 78 97 99 a6 d9 35 68 0f 61 68 fa b1 21 82	ec 1a c0 80 0c 50 53 c7 3b d7 00 ef b7 22 72 e0	37 bb 38 f7 14 3d d8 7d 93 e7 08 a1 48 f7 a5 4a
db a1 f8 77 18 6d 8b ba a8 30 08 4e ff d5 d7 aa	b9 32 41 f5 ad 3c 3d f4 c2 04 30 2f 16 03 0e ac	b9 32 41 f5 3c 3d f4 ad 30 2f c2 04 ac 16 03 0e	b1 1a 44 17 3d 2f ec b6 0a 6b 2f 42 9f 68 f3 b1	48 f3 cb 3c 26 1b c3 be 45 a2 aa 0b 20 d7 72 38
f9 e9 8f 2b 1b 34 2f 08 4f c9 85 49 bf bf 81 89	99 1e 73 f1 af 18 15 30 84 dd 97 3b 08 08 0c a7	99 1e 73 f1 18 15 30 af 97 3b 84 dd a7 08 08 0c	31 30 3a c2 ac 71 8c c4 46 65 48 eb 6a 1c 31 62	fd 0e c5 f9 0d 16 d5 6b 42 e0 4a 41 cb 1c 6e 56
cc 3e ff 3b a1 67 59 af 04 85 02 aa a1 00 5f 34	4b b2 16 e2 32 85 cb 79 f2 97 77 ac 32 63 cf 18	4b b2 16 e2 85 cb 79 32 77 ac f2 97 18 32 63 cf	4b 86 8a 36 b1 cb 27 5a fb f2 f2 af cc 5a 5b cf	b4 ba 7f 86 8e 98 4d 26 f3 13 59 18 52 4e 20 76
ff 08 69 64 0b 53 34 14 84 bf ab 8f 4a 7c 43 b9				

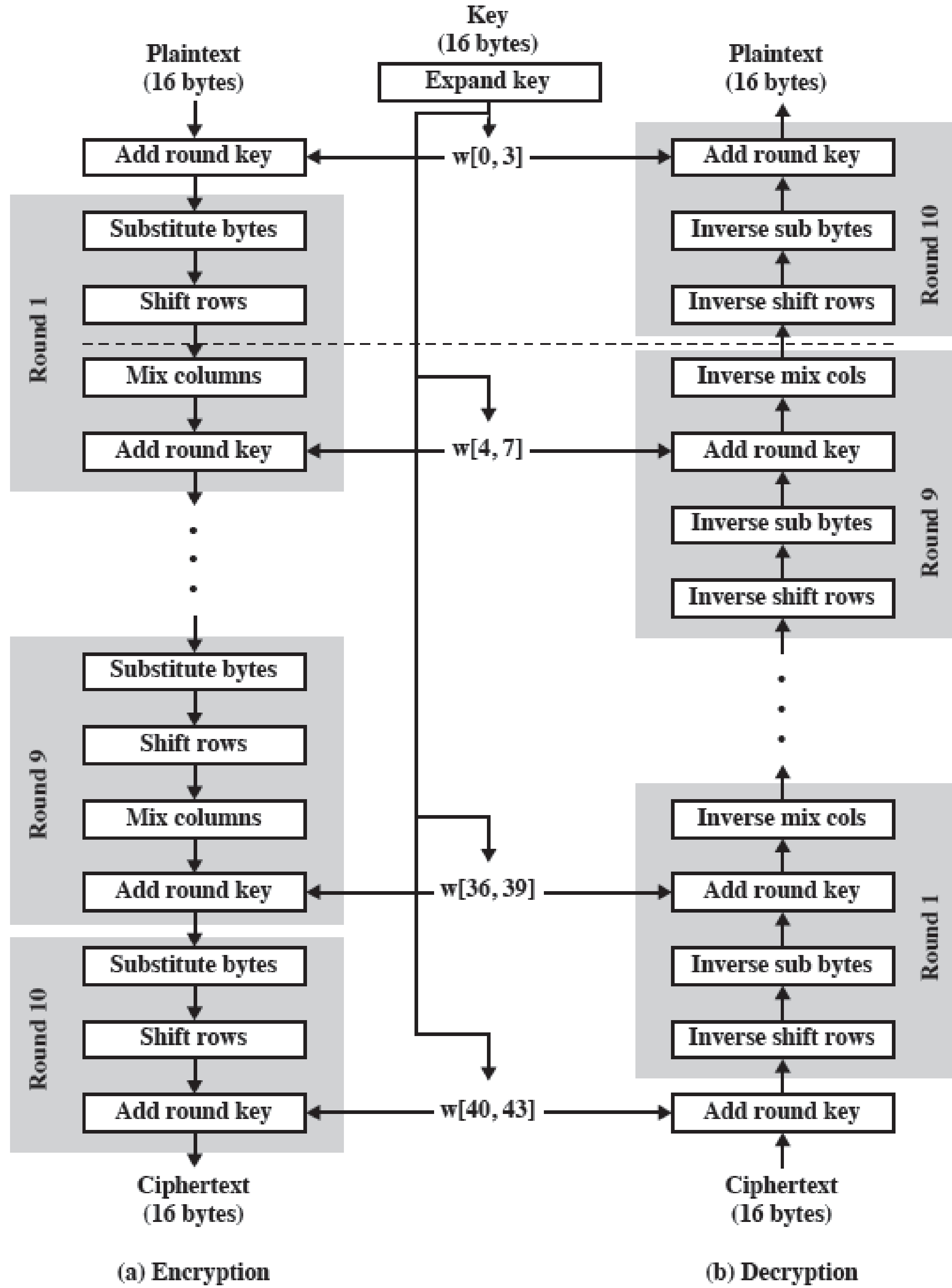
AES Implementation: Equivalent Inverse Cipher

- AES decryption cipher is not identical to the encryption cipher
- 1. The sequence of transformations differs, the form of the key schedules is the same
  - – Disadvantage: two separate software or firmware modules are needed
- Or using 2. Equivalent Inverse Cipher: has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). A change in key schedule is needed.

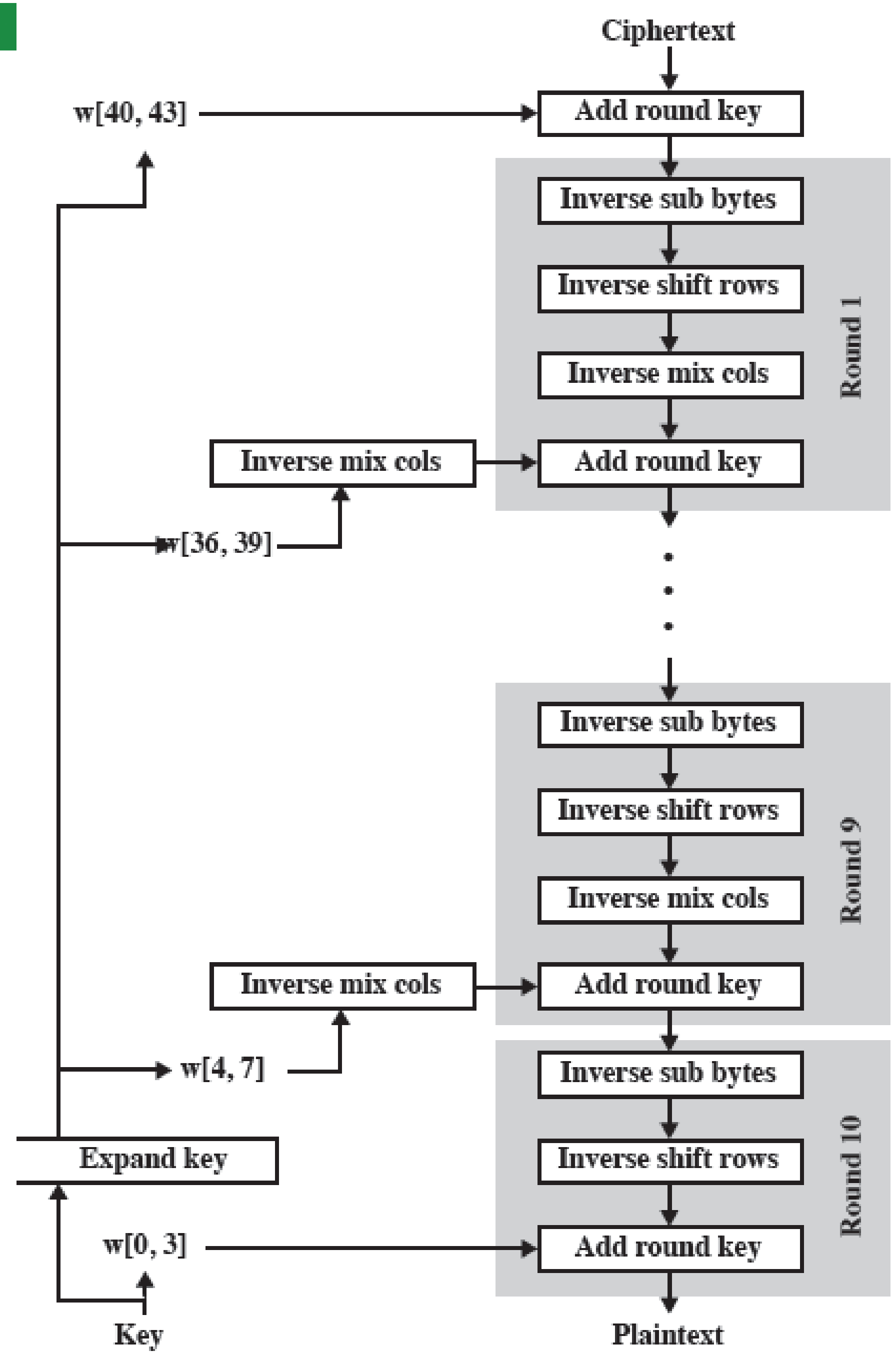
Interchanges

- The encryption round structure:
  SubBytes, ShiftRows, MixColumns, AddRoundKey
- The Decryption round structure:
  InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns
  - Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged

AES Encryption and Decryption



Equivalent Inverse Cipher



Implementation Aspects

- On an 8-bit processor: typical for current smart cards
  - AddRoundKey is a bytewise XOR operation
  - ShiftRows is a simple byte-shifting operation
  - SubBytes operates at the byte level and only requires a table of 256 bytes
  - MixColumns requires matrix multiplication in the field $GF(2^8)$, which means that all operations are carried out on bytes

Cont'd

- On 32-bit processors: typical for PCs.
  - Redefine steps to use 32-bit words
  - Can precompute 4 tables of 256-words
  - Then each column in each round can be computed using 4 table lookups + 4 XORs
  - At a cost of 4Kb to store tables
- Designers believe this very efficient implementation was a key factor in its selection as the AES cipher