Advanced Encryption Standard (AES)

Killian O'Brien

6G6Z0024 Applied Cryptography 2023/24

Lecture Week 06 – Mon 06 November 2023

Introduction

- The **Advanced Encryption Standard** (AES), published by NIST in 2001.
- To replace DES due to security concerns over small key size and other considerations.
- AES designed to be more secure and fast.
- From 2008 on, chip manufacturers implement AES capabilities in low-level chip design.
- Now the most widely used cipher.

Polynomial arithmetic and the fields $GF(2^n)$

- Consider all polynomials

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i, \quad a_i \in \mathbb{Z}_2.$$

  of degree $n-1$ or less.
- Arithmetic follows the rules of $+$ and $\cdot$ for polynomials, with arithmetic of the coefficients a_i carried out in $\mathbb{Z}_2$, i.e. addition of coefficients is the same as XOR.
- If multiplication results in a polynomial of degree greater than $n-1$ then the product is reduced modulo a specified irreducible polynomial $m(x)$ of degree n, the modulos polynomial.

AES and $\mathrm{GF}(2^8)$.

- The Advanceed Encryption Standard (AES) uses such a field $\mathrm{GF}(2^8)$, consisting of polynomials of degree less than or equal to 7, with binary coefficients and polynomial operations carried out modulo the irreducible polynomial

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

- The figure on the right shows the calculation of an example product in $\mathrm{GF}(2^8)$.
- AES uses this since it is designed to operate on 8-bit bytes.
    - addition of bytes is just bit-wise XOR.
    - *multiplication* of bytes is defined as multiplication in the finite field $\mathrm{GF}(2^8)$, modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1.$

The Advanced Encryption Standard (AES) uses arithmetic in the finite field GF(2^8), with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Consider the two polynomials $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$. Then

$$\begin{aligned} f(x) + g(x) &= x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1 \\ &= x^7 + x^6 + x^4 + x^2 \end{aligned}$$

$$\begin{aligned} f(x) \times g(x) &= x^{13} + x^{11} + x^9 + x^8 + x^7 \\ &\quad + x^7 + x^5 + x^3 + x^2 + x \\ &\quad + x^6 + x^4 + x^2 + x + 1 \\ &= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \end{aligned}$$

$$\begin{array}{r|l} & x^5 + x^3 \\ \hline x^8 + x^4 + x^3 + x + 1 & x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\ & x^{13} + x^9 + x^8 + x^6 + x^5 \\ \cline{2-2} & x^{11} + x^4 + x^3 \\ & x^{11} + x^7 + x^6 + x^4 + x^3 \\ \cline{2-2} & x^7 + x^6 + 1 \end{array}$$

Therefore, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$.

- Operates on plaintext message blocks of 16 bytes = 128 bits.
- Various key lengths allowed, 16, 24 or 32 bytes. Ciphers referred to as AES-128, AES-192 or AES-256, depending on how many bits used in key.
- Throughout encryption (and decryption) the message block is maintained as a 4×4 array of bytes. This is referred to as the **state**.
- First four bytes form the first column, next four bytes the second column, and so on.
- An initial transformation of the state is followed by N rounds. Where N depends on the ley length used.
  - $N = 10$ for 128 bit key.
  - $N = 12$ for 192 bit key.
  - $N = 14$ for 256 bit key.
- The key passes through a *key explansion* transformation to provide $N + 1$ sub-keys to be used in the initial transformation and N rounds.
- Each sub-key consists for four 4-byte **words**, which form the columns of the **round key matrix**.

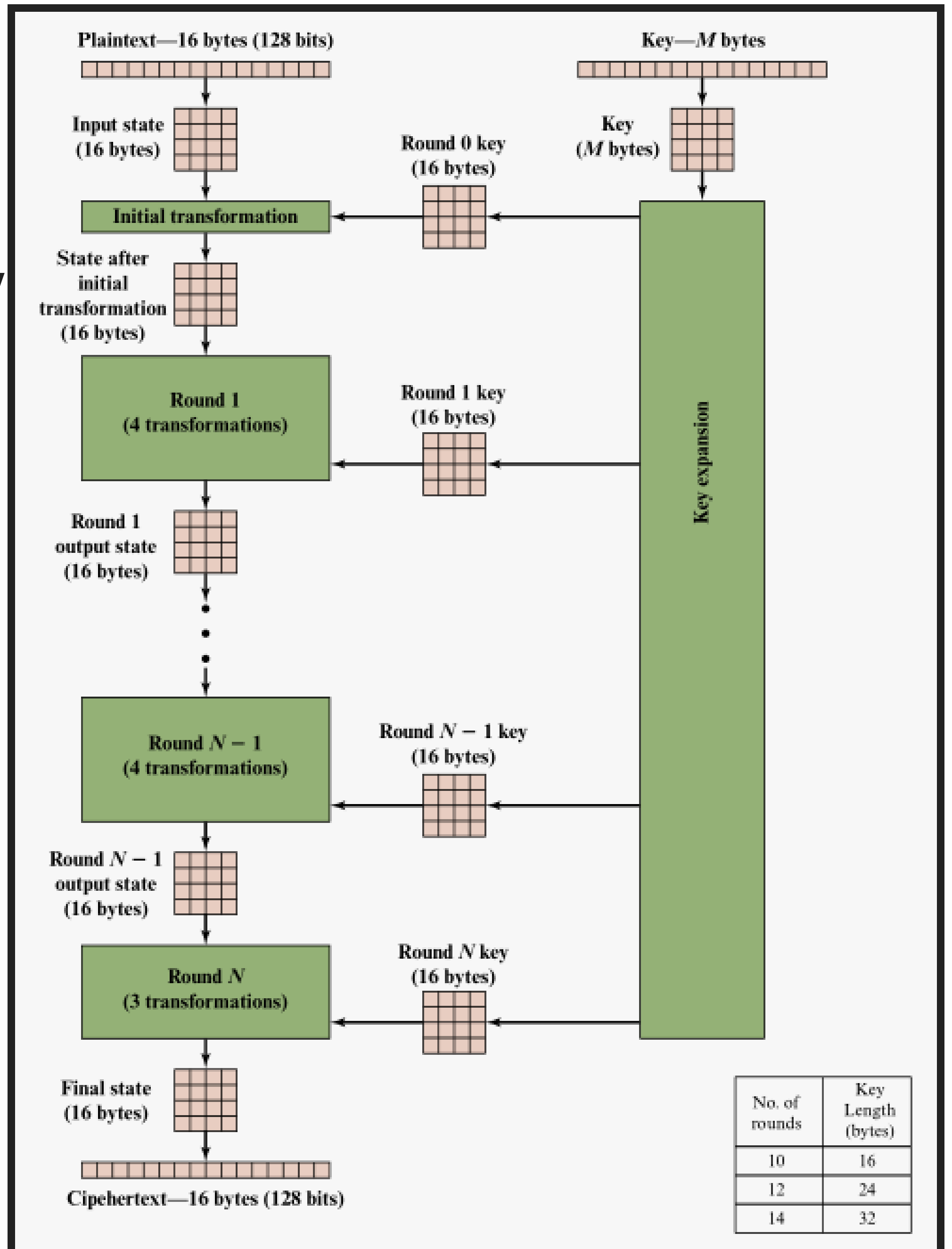


Figure 6.1 AES Encryption Process

AES detail

- This figure exposes the transformations within each round, for AES-128. The other schemes are similar.
- Note that is departs from the Feistel design. There is no notion of dividing the block into halves.
- Rounds 1 - 9 consists of four transformations
    - **Substitute bytes**: and S-box type permutation of the bytes of the state.
    - **Shift rows**: a simple permutation of the bytes within each row of the state.
    - **Mix columns**: a transformation that combines the bytes within each column of the state. This transformation utilies the $\mathrm{GF}(2^8)$ field.
    - **Add round key**: bit-wise XOR of the state with the appropriate round key matrix.
- The decryption algorithm reverses all the transformations. At each horizontal level, the intermediate states of the encryption and decryption algorithms are the same.

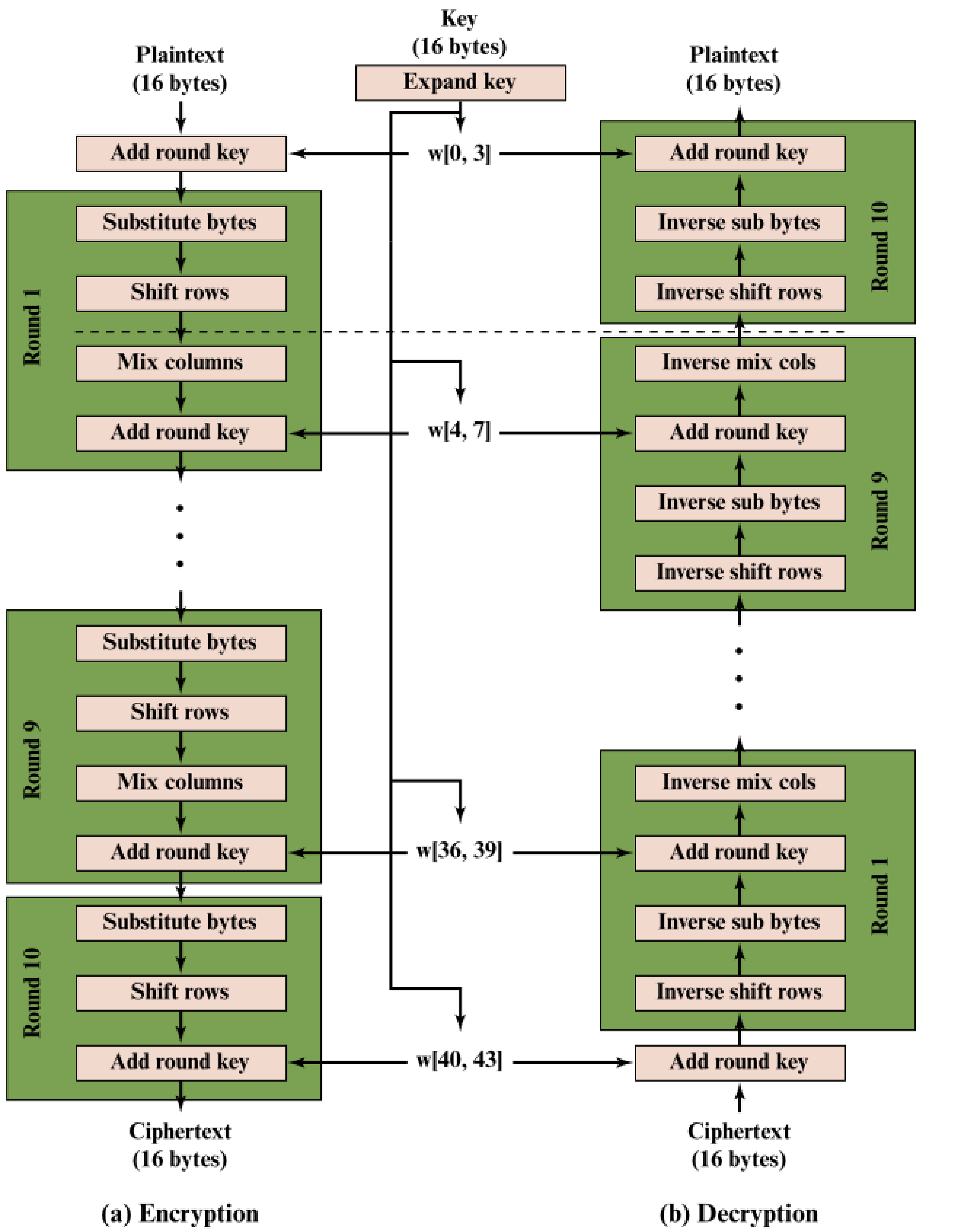


Figure 6.3 AES Encryption and Decryption

- This figure visualizes the four transformations within a typical round.
- Here the state matrix is shown laid out as a row of 16 bytes.

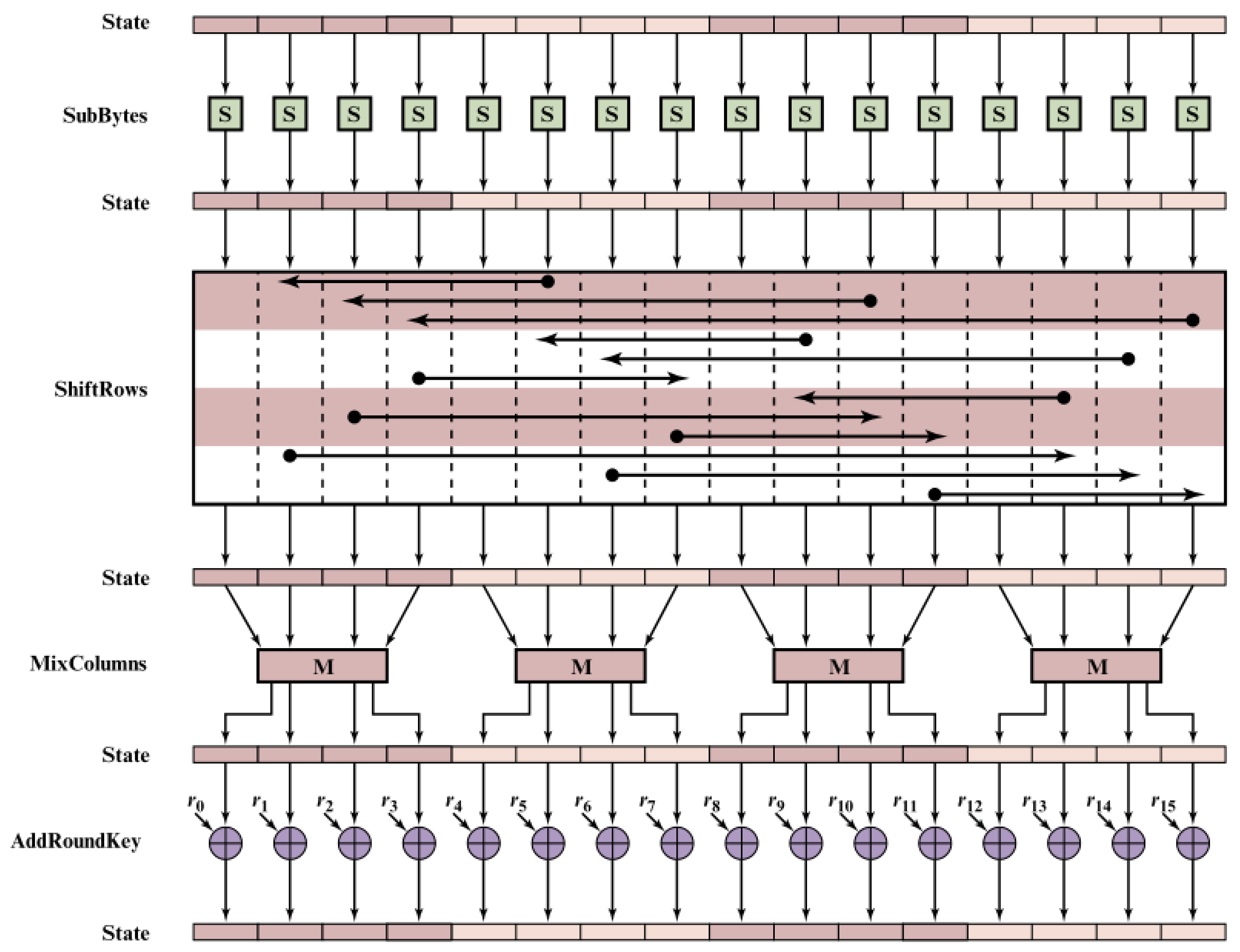
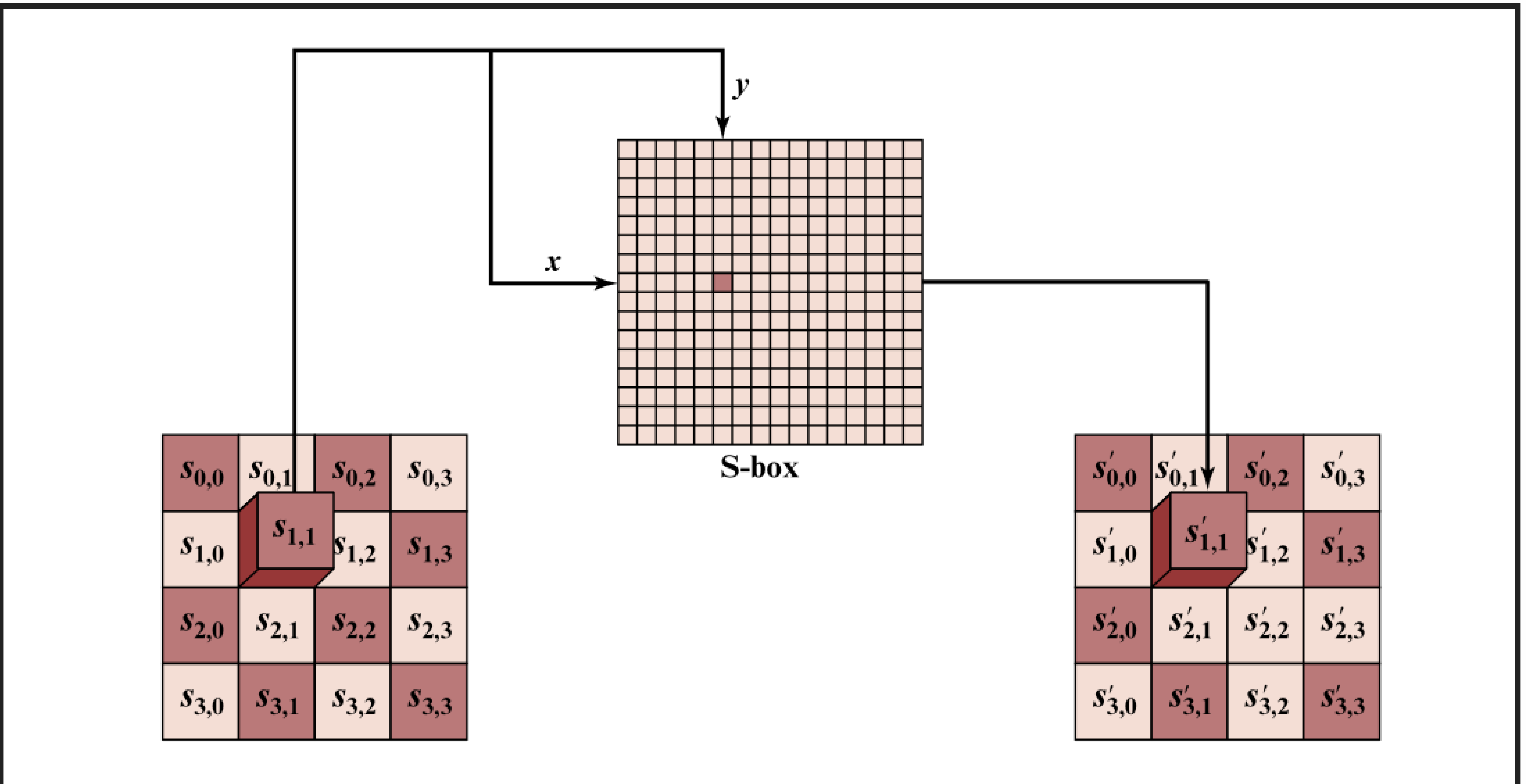


Figure 6.4 AES Encryption Round

- This figure shows the how the substitute bytes transformation is defined.
- For each entry of the incoming state matrix, i.e. for each byte
    - the first four bits denote the row index x of the S-box
    - the second four bits denote the column index y of the S-box
    - the S-box entry at that row and column is a byte that replaces the original byte of the state.
- After replacing each entry of the incoming state matrix, we get the outgoing state matrix.



(a) Substitute byte transformation

The S-box itself

- This figure shows the S-box.
- Remember
  - a four bit block is denoted by a hexadecimal digit `0,1,...9,a,b,c,d,e,f`.
  - a single byte (i.e. a 8-bit block) is denoted by a two-digit hexadecimal number.
- A corresponding inverse S-box table is used in the decryption algorithm.
- Lots of detail in Stallings on the contruction of this S-box table.
  - Designed like this to minimize any correlation between incoming and outgoing bits.

		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

(a) S-box

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

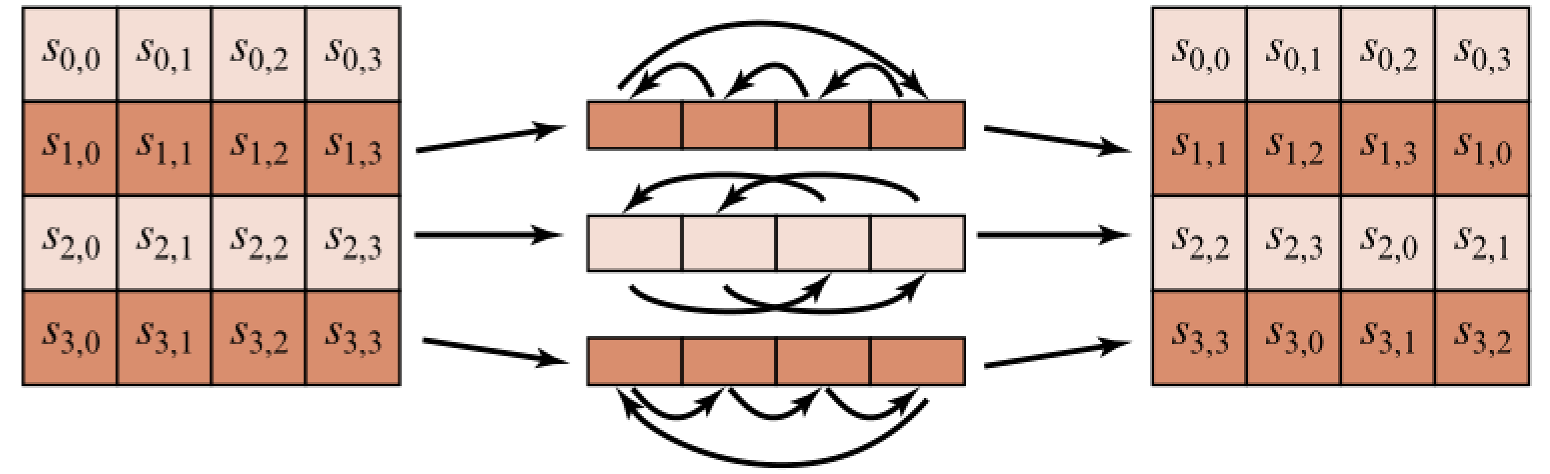
→

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

- An example substitute bytes transformation is shown here

The shift rows transformation

- In each of the second, third and fourth rows of state, permute the bytes in each row as shown.
- This has a significant effect on the original positions of the bits within the 128 bit message block.



(a) Shift row transformation

Mix columns transformation

- The equation shows how the incoming state matrix $s_{i,j}$ is multiplied by the matrix of constants to get the outgoing state matrix $s'_{i,j}$, shown on the right of the equation.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(6.3)}$$

- This results in the following transformations within the j^{th} column of the state

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

- The operations $\oplus$ and $\cdot$ shown here are the operations from $\mathrm{GF}(2^8)$, carried out on the entries of the state, i.e. on the bytes, i.e. on the 8-bit blocks which are interpreted as the coefficients of degree 7 polynomials.
- So $\oplus$ is bitwise XOR and $\cdot$ is the multiplication obtained from the multiplication of these polynomials, modulo the polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$.
- The design of this Mix Columns transformation ensures good mixing of the bytes within a column, and the use of the constants `01`, `02` and `03` results in efficient implementation of the encryption algorithm.

Add round key transformation

- Perhaps the most straightforward, this is just bitwise XOR amongst the bit entries of the state and round key matrices.
- Consider the examples shown here of

$$\text{incoming state} \oplus \text{round key} = \text{outgoing state}$$

$$\begin{array}{|c|c|c|c|}\hline 47 & 40 & A3 & 4C \\ \hline 37 & D4 & 70 & 9F \\ \hline 94 & E4 & 3A & 42 \\ \hline ED & A5 & A6 & BC \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline AC & 19 & 28 & 57 \\ \hline 77 & FA & D1 & 5C \\ \hline 66 & DC & 29 & 00 \\ \hline F3 & 21 & 41 & 6A \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline EB & 59 & 8B & 1B \\ \hline 40 & 2E & A1 & C3 \\ \hline F2 & 38 & 13 & 42 \\ \hline 1E & 84 & E7 & D6 \\ \hline \end{array}$$

A summary flowchart for AES encryption

- This figure summarizes a typical encryption round.
- We still need to describe the key expansion process for the derivation of the round keys from the original key.

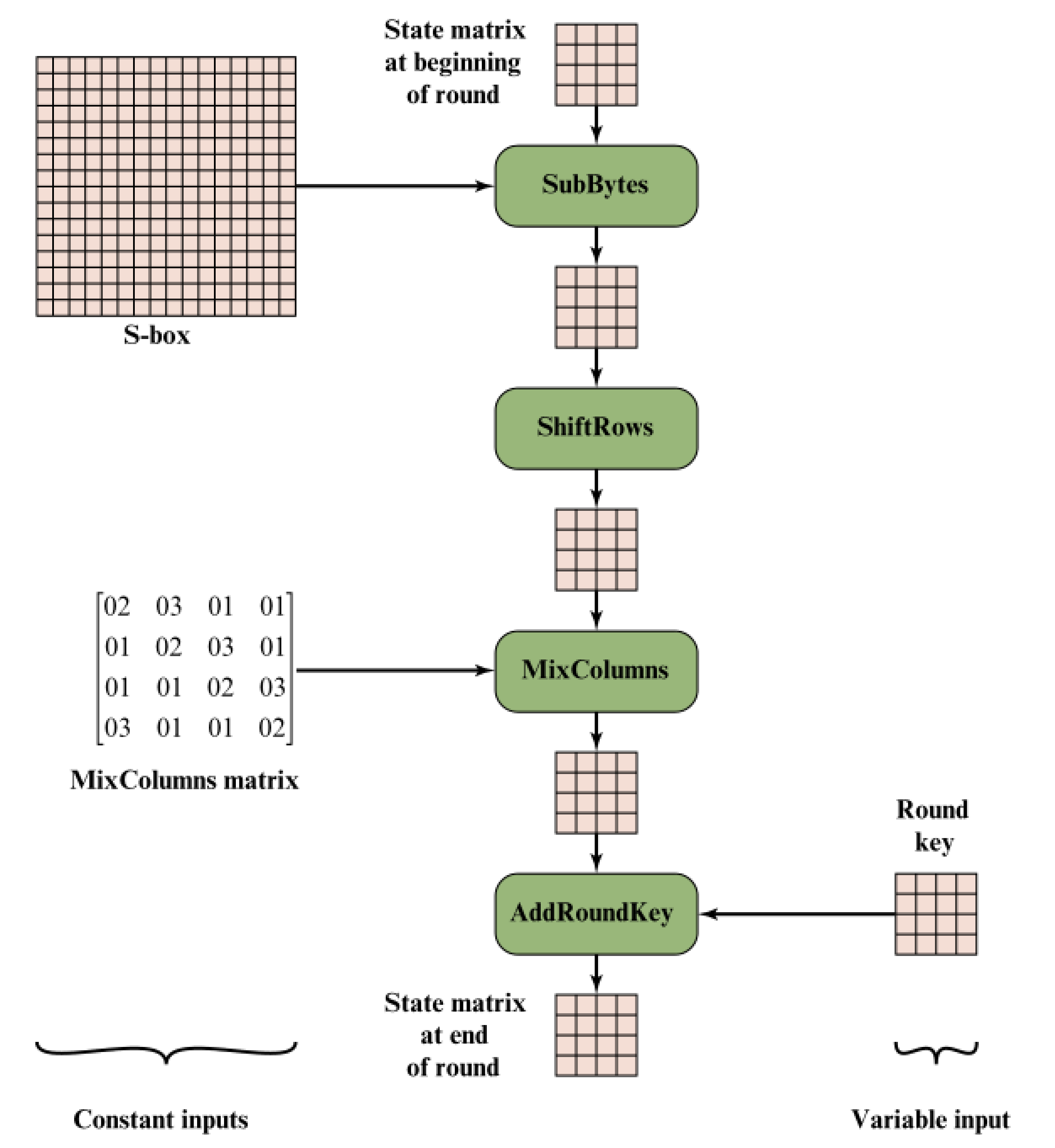
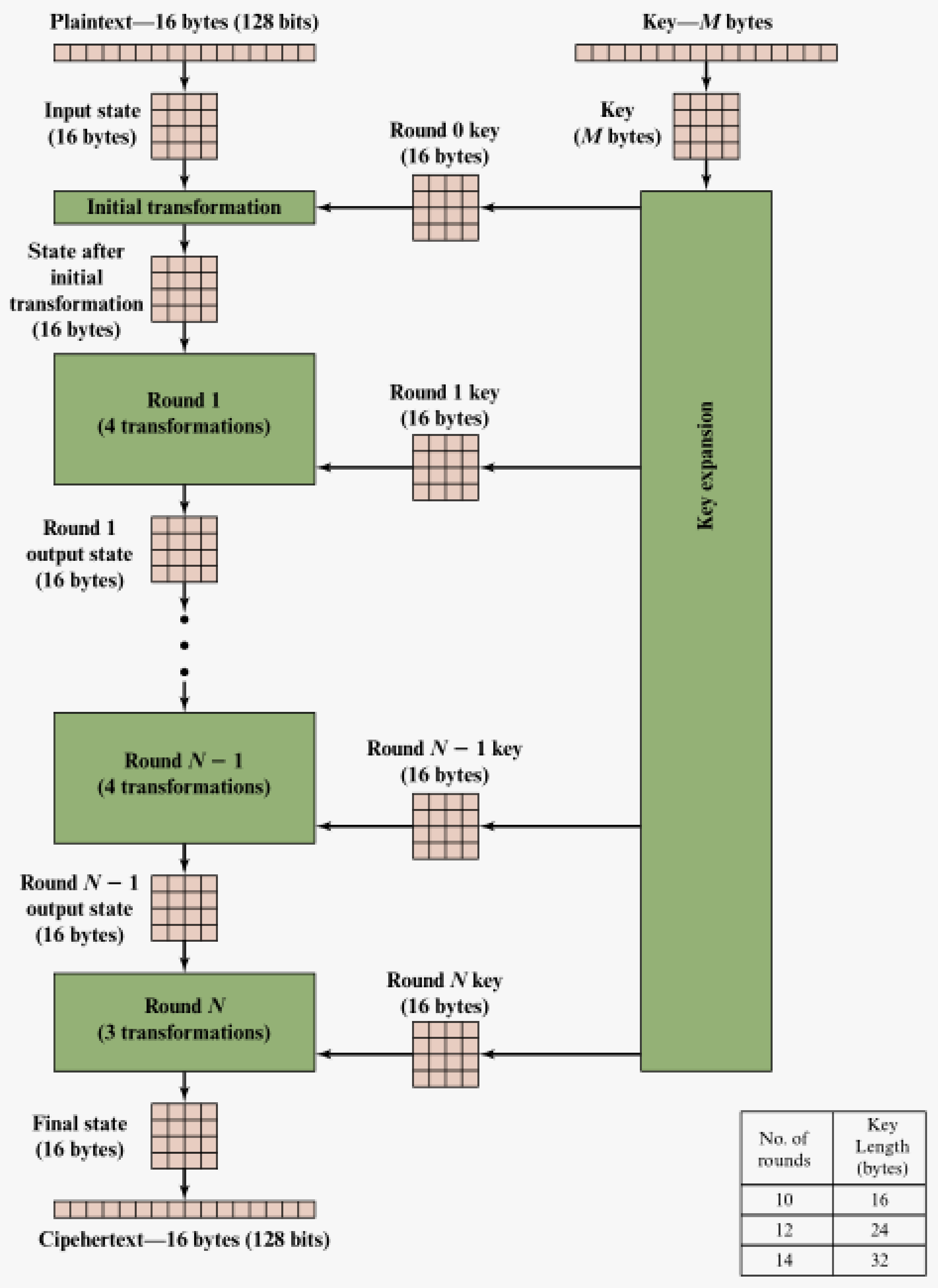


Figure 6.8 Inputs for Single AES Round

Key expansion in AES

- **Key expansion** is the process where the initial key is expanded to produce the $N+1$ round keys for the initial transformation and the N rounds of AES.
- Each round key consists of four 4-byte words, i.e. the four columns of the round key matrix.



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

Figure 6.1 AES Encryption Process

Key expansion in AES-128

- The i^{th} of the $N+1$ round keys consist of the four keywords

$$w_{4i+0}, w_{4i+1}, w_{4i+2}, w_{4i+3}.$$

- The 16 bytes of the initial key form the first four key words w_0, w_1, w_3, w_3 as shown.
- An iterative process creates the 40 subsequent key words.
  - for $j = 1, 2, 3$, and $i = 1, \ldots, 10$,

$$w_{4i+j} = w_{4i+j-1} \oplus w_{4(i-1)+j}.$$

  - for $j = 0$ and $i = 1 \ldots 10$,

$$w_{4i} = g(w_{4i-1}) \oplus w_{4(i-1)}.$$

  - the function g is the composition of
    - a circular left-shift of the word bytes
    - an S-box byte substitution using the same S-box table as in the AES encryption rounds
    - a bitwise XOR with the word formed by the bytes $\mathrm{RC_j}$, 00, 00, 00. The round constants $\mathrm{RC_j}$ are shown on table on next slide.

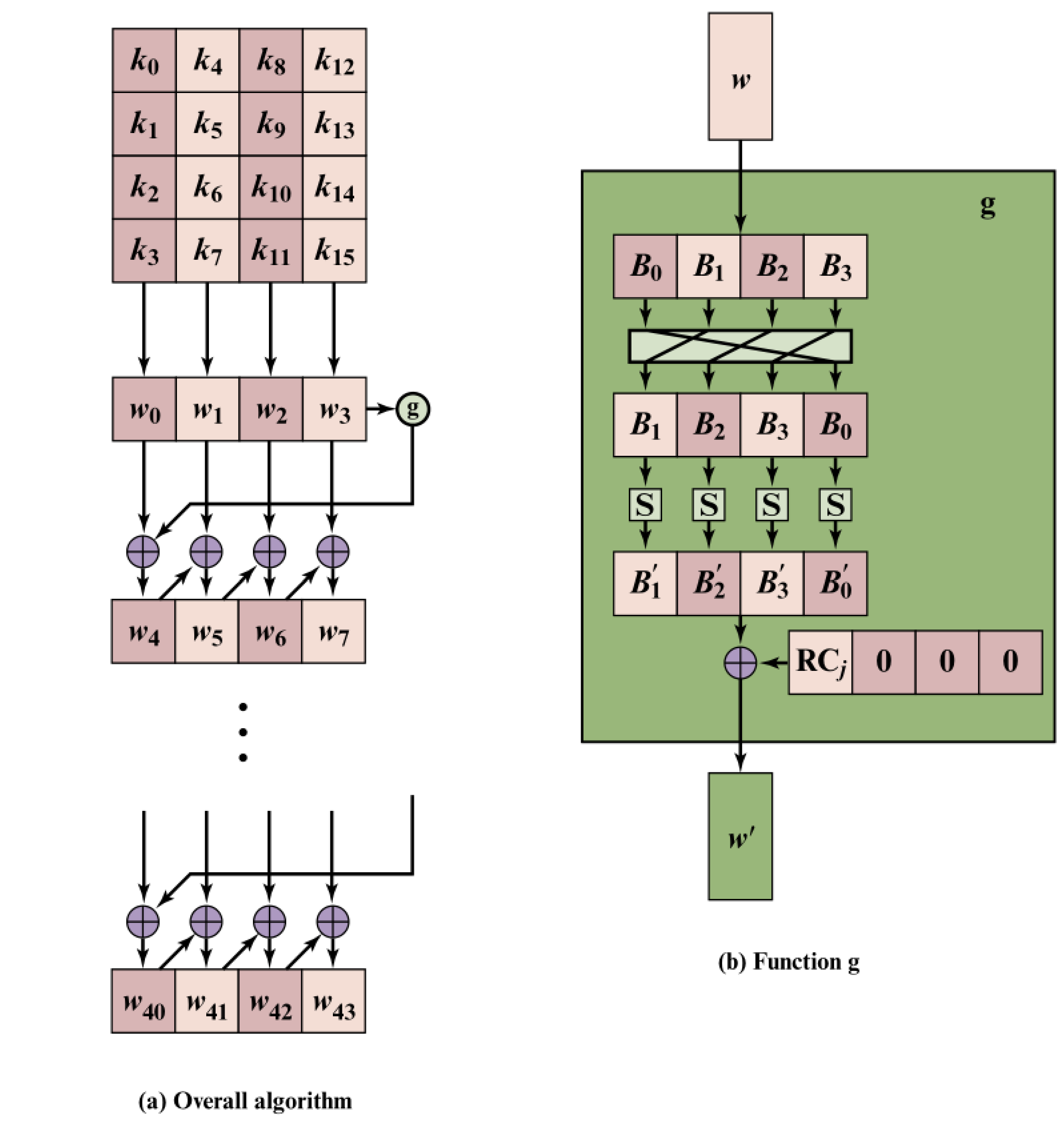


Figure 6.9 AES Key Expansion

Design considerations of the key expansion algorithm

- The round constants RC_j.

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

- These transformations were chosen to ensure these features, amongst others,
    - speedy implementations in software and chip hardware,
    - partial knowledge of the original key or intermediate round keys will not enable determination of many other bits of other key words,
    - use of different round constants eliminates any potential symmetries in the round key generations,
    - diffusion, i.e. each bit of the original key effects many round key bits.
- Further details, worked examples, and illustrations of bit diffusion from plaintext and key differences can be found in [Stallings, Chapter 6: AES]

// reveal.js plugins