

Advanced data Security

Part I:

Cryptography and Network Security

Chapter3:

Advanced Encryption Standard

Prepared by:

Dr. Hamssa Hasrouny

Lecture slides prepared for “Cryptography and Network Security”, 6/e, by William Stallings, Chapter 5 – “Advanced Encryption Standard”.

Finite Field Arithmetic

- 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 $GF(2^8)$
 - which means that all operations (addition, subtraction, multiplication, and division) are carried out on bytes.

In AES, all operations are performed on 8-bit bytes. In particular, the arithmetic operations of addition, multiplication, and division are performed over the finite field $GF(2^8)$. Section 4.7 discusses such operations in some detail. For the reader who has not studied Chapter 6, and as a quick review for those who have, this section summarizes the important concepts.

In essence, 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 $\{0, 1, \dots, p-1\}$, where p is a prime number and in which arithmetic is carried out modulo p .

AES overall structure: Encryption Process

- Plaintext block size of 128 bits, or 16 bytes in single 128-bit-block or a 4 x 4 square matrix of bytes (FIPS PUB 197).
- The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits).
- Each block is copied in the **State** array (modified at each stage of encryption and decryption). Final stage, **State** is copied to an output matrix.
- Key : a square matrix expanded into an array of key schedule words.
- Cipher : N rounds (10 rounds for a 16-byte key, 12 rounds for a 23-byte key, 14 rounds for a 32-byte key).
- The first N-1 rounds: 4 distinct transformation functions: SubBytes, ShiftRows, MixColumns and AddRoundKey.
- Last round: three transformations preceded by an initial single transformation Round 0 (AddRoundKey).
- Each transformation takes one or more 4x4 matrices as input and produces a 4x4 matrix as output.
- The key expansion function generates $N + 1$ round keys (4 x 4 matrix). Each round key serves an input to the AddRoundKey transformation.

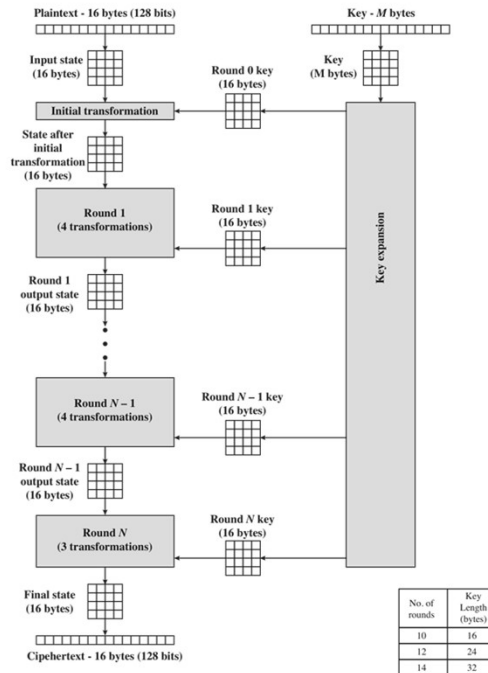
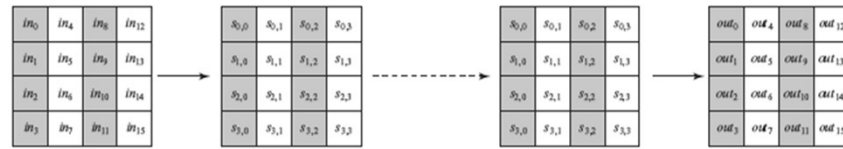


Figure 5.1 AES Encryption Process

<https://www.youtube.com/watch?v=O4xNJsitN6E>

Figure 5.1 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

AES Data Structures



(a) Input, state array, and output



(b) Key and expanded key

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a 4×4 square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. These operations are depicted in Figure 5.2a. Similarly, the key is depicted as a square matrix of bytes.

The key is depicted as a square matrix of byte then expanded into an array of key schedule words. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a 4×4 square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. These operations are depicted in Figure 5.2a. Similarly, the key is depicted as a square matrix of bytes.

This key is then expanded into an array of key schedule words. Figure 5.2b shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

Table 5.1
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

The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key (Table 5.1). The first $N - 1$ rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is a initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the ciphertext. Also, 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.

The cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key (Table 5.1). The first $N - 1$ rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is a initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the ciphertext.

Also, 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.

AES Encryption and Decryption

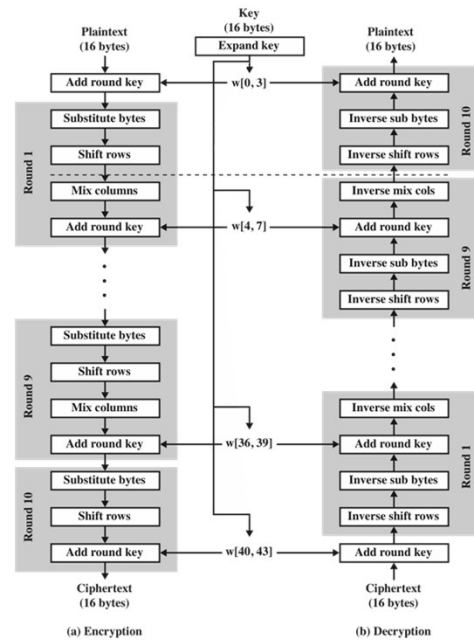


Figure 5.3 AES Encryption and Decryption

Figure 5.3 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function. As was done in Chapter 3, we show encryption proceeding down the page and decryption proceeding up the page.

Detailed Structure

- Processes the entire data block as a single matrix during each round using substitutions and permutation (not a Feistel structure)
- The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$

Four different stages are used (one of permutation and 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

- The cipher begins and ends with an AddRoundKey stage
- Can view the cipher as **alternating** operations of **XOR encryption** (AddRoundKey: a form of Vernam cipher) of a block, followed by scrambling of the block (the **other three stages**), followed by **XOR encryption**, and so on
- Each stage is easily **reversible**: For the **Substitute Byte**, **ShiftRows**, and **MixColumns** stages, an **inverse function** is used in the **decryption algorithm**. For the AddRoundKey stage, the **inverse** is achieved by **XORing the same round key to the block**, using the result that $A \oplus B \oplus B = A$.
- The decryption algorithm makes use of the **expanded key in reverse order**, however the decryption algorithm is not identical to the encryption algorithm
- State is the same for both encryption and decryption
- Final round of both encryption and decryption consists of only three stages

Before delving into details, we can make several comments about the overall AES structure.

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that, in the **classic Feistel structure**, **half of the data block** is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation.

2. The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[i]$. **Four distinct words** (128 bits) serve as a **round key for each round**;

these are indicated in Figure 5.3.

3. Four different stages are used, **one of permutation and three 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(28)

- AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.

5. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is **reversible without knowledge of the key** and so would **add no security**.

6. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. **The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key.** We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

7. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block.

8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, **the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.**

9. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 5.3 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), 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**.

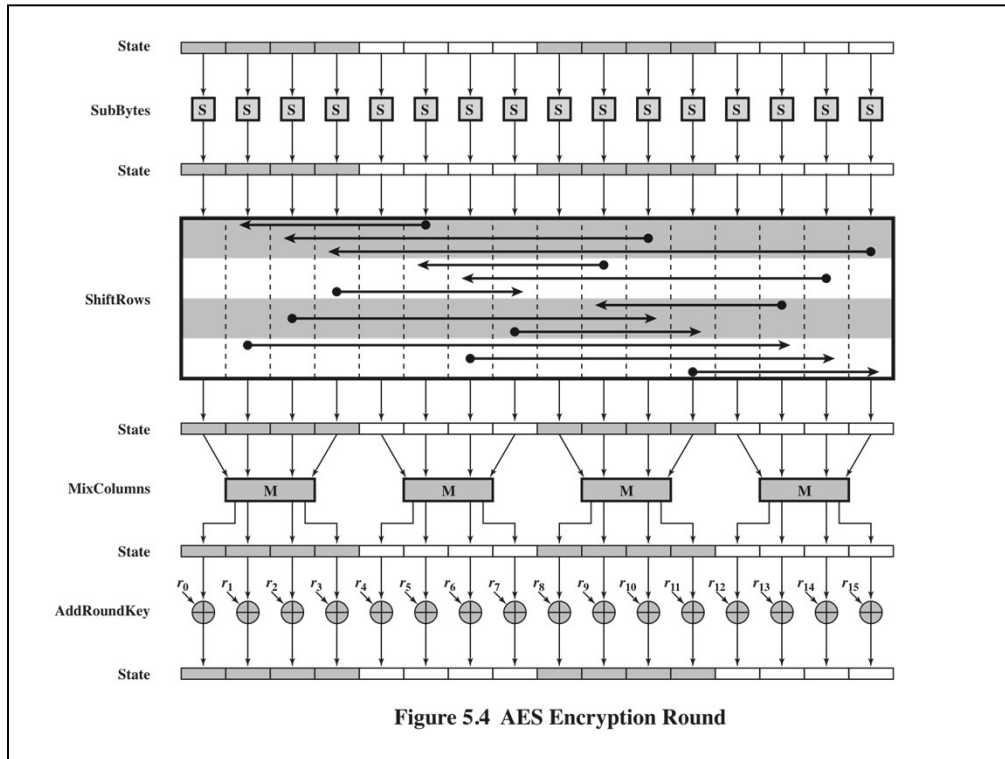
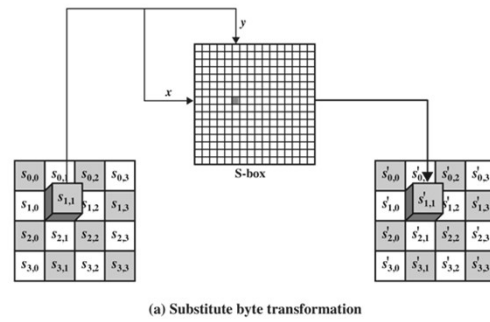


Figure 5.4 depicts the structure of a full encryption round.

AES Byte Level Operations



Substitute Bytes Transformation

The forward substitute byte transformation, called SubBytes, is a simple table lookup

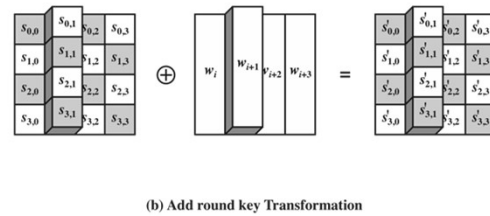


Figure 5.5 AES Byte-Level Operations

The forward substitute byte transformation , called SubBytes, is a simple table lookup (Figure 5.5a).

Table 5.2

AES defines a 16 * 16 matrix of byte values, called an S-box (Table 5.2a), that contains a permutation of all possible **256 8-bit values**.

Each individual byte of State is mapped into a new byte in the following way: The **leftmost 4 bits of the byte are used as a row value** and the **rightmost 4 bits are used as a column value**. These row and column values serve as indexes into the S-box to select a **unique 8-bit output value**.

Ex: the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the 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

(a) S-box

AES defines a 16 * 16 matrix of byte values, called an S-box (Table 5.2a), that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.

These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

Table 5.2

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

(b) Inverse S-box

The inverse substitute byte transformation , called InvSubBytes, makes use of the inverse S-box shown in Table 5.2b.

The S-box is constructed in the following fashion

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, ..., {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row y , column x is $\{yx\}$.

2. Map each byte in the S-box to its multiplicative inverse in the finite field $GF(2^8)$; the value {00} is mapped to itself.

3. Consider that each byte in the S-box consists of 8 bits labeled $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$. Apply the following transformation to each bit of each byte in the S-box:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \quad (6.1)$$

where c_i is the i th bit of byte c with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$. The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as in the figure.

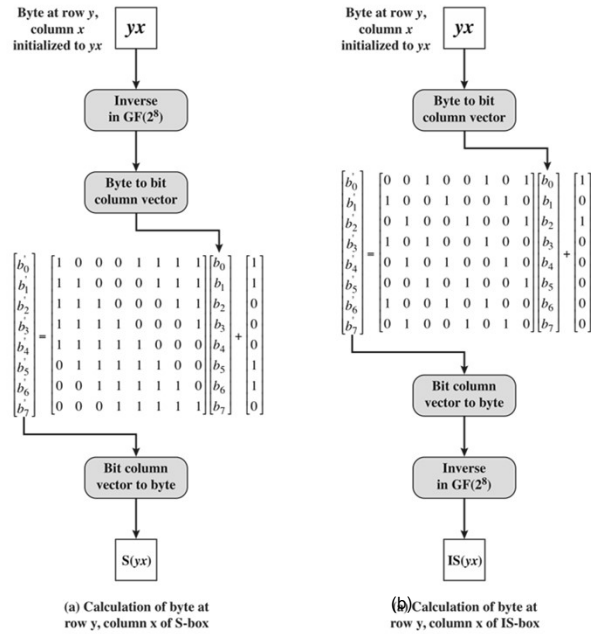


Figure 5.6 Construction of S-Box and IS-Box

Construction of S-Box and IS-Box

Inverse substitute byte transformation

- The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in Table 5.2b.
 - Note, for example, that the input {2A} produces the output {95}, and the input {95} to the S-box produces {2A}.
- The inverse S-box is constructed (Figure 5.6b) by applying the inverse of the transformation in Equation (6.1) followed by taking the multiplicative inverse in $GF(2^8)$.
- The inverse transformation is

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

where byte $d = \{05\}$, or 00000101.

The S-box is designed to be resistant to known **cryptanalytic attacks**. Specifically, the Rijndael developers sought a design that has a low correlation Between input bits and output bits and the **property that the output is not a linear** mathematical function of the input [DAEM01]. The nonlinearity is due to the use of the **multiplicative inverse**.

S-Box Rationale

- The S-box is designed to be **resistant to known cryptanalytic attacks**.
- The Rijndael developers sought a design that has a **low correlation between input bits and output bits** and the property that the output is not a linear mathematical function of the input.
- The **nonlinearity is due to the use of the multiplicative inverse**.

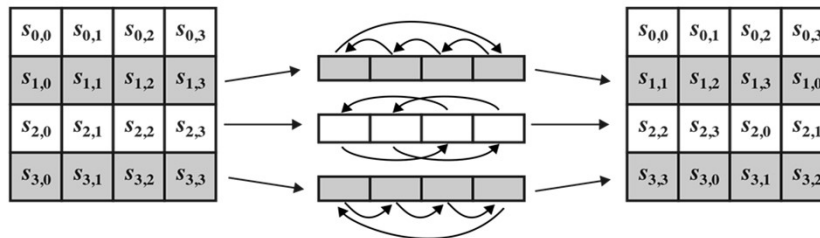
The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation Between input bits and output bits and the property that the output is not a linear mathematical function of the input [DAEM01]. The nonlinearity is due to the use of the multiplicative inverse.

Shift Row Transformation

The forward shift row transformation, called ShiftRows:

- The first row of State is not altered.
- For the second row, a 1-byte circular left shift is performed.
- For the third row, a 2-byte circular left shift is performed.
- For the fourth row, a 3-byte circular left shift is performed.

The following is an example of ShiftRows.



(a) Shift row transformation

Figure 5.7 AES Row and Column Operations

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

The forward shift row transformation, called ShiftRows, is depicted in Figure 5.7a. The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows.

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

Shift Row Rationale

- More substantial than it may first appear
- The State, as well as the cipher input and output, is treated as an array of four **4-byte columns**
- On encryption, the first **4 bytes of the plaintext are copied to the first column of State**, and so on
- 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
- Transformation ensures that the 4 bytes of one column are spread out to four different columns

The shift row transformation is more substantial than it may first appear. This is because the State, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, **the first 4 bytes of the plaintext are copied to the first column of State**, and so on. Furthermore, as will be seen, **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. Figure 5.4 illustrates the effect.

MixColumn Transformation

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. The transformation can be defined by the following matrix multiplication on State (Figure 5.7b)

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in $GF(2^8)$.

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

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

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

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

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

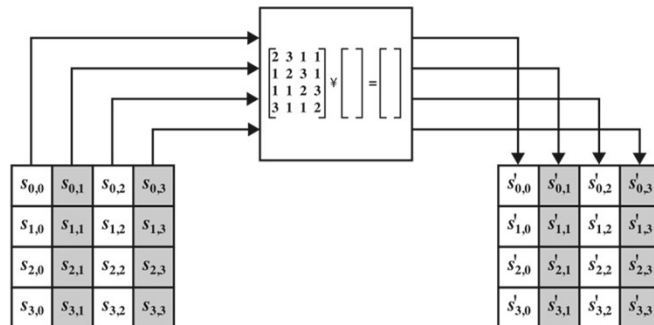


Figure 5.7 AES Row and Column Operations
(b) Mix column transformation

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. The transformation can be defined by the following matrix multiplication on State (Figure 5.7b)

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in $GF(2^8)$.

Inverse mix column transformation

- The inverse mix column transformation, called `InvMixColumns`, is defined by the **following** matrix multiplication:

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

The coefficients of the matrix in Equation (5.3) 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. See [DAEM99] for a discussion.

In addition, the choice of coefficients in `MixColumns`, **which are all {01}, {02}, or {03}, was influenced by implementation considerations**. As was discussed, multiplication

by these **coefficients involves at most a shift and an XOR**. The coefficients in `InvMixColumns` are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes (Figures 6.5 and 6.6; described in Chapter 6), only encryption is used.
2. As with any block cipher, **AES can be used to construct a message authentication code** (Chapter 12), and for this, only encryption is used.

Mix Columns Rationale

- 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**
- The mix column transformation combined with the shift row transformation ensures that after a few rounds all output bits depend on all input bits

The coefficients of the matrix in Equation (5.3) 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. See [DAEM99] for a discussion.

In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations. As was discussed, multiplication

by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes (Figures 6.5 and 6.6; described in Chapter 6), only encryption is used.
2. As with any block cipher, AES can be used to construct a message authentication code (Chapter 12), and for this, only encryption is used.

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
- The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

Rationale:

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

In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As shown in Figure 5.5b, the operation is viewed as a columnwise 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 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.**

Inputs for Single AES Round

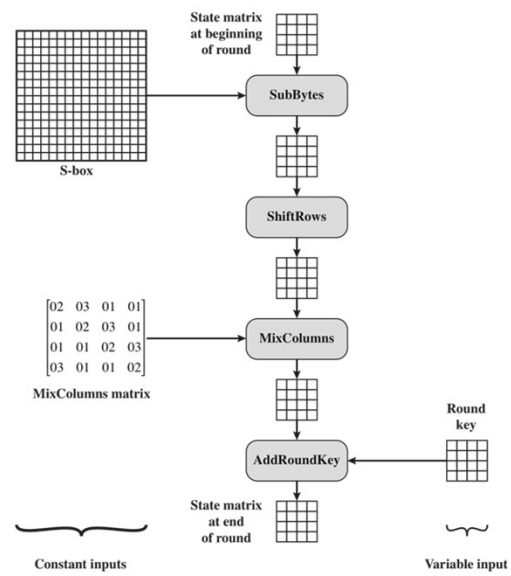


Figure 5.8 Inputs for Single AES Round

Figure 5.8 is another view of a single round of AES, emphasizing the mechanisms and inputs of each transformation.

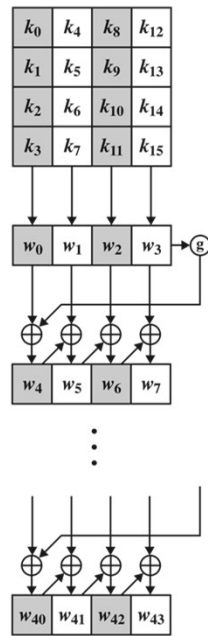
AES Key Expansion

- Takes as input a four-word (16 byte) key and produces a linear array of 44 words (176) bytes
 - This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher
- Key is copied into the first 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, $w[i - 4]$
 - 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

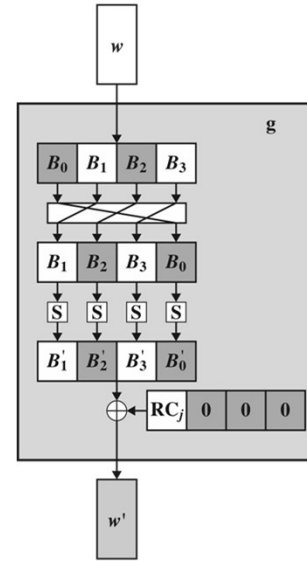
The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The pseudocode on the next page describes the expansion.

The key is copied into the first 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, $w[i - 4]$. 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.

AES Key Expansion



(a) Overall algorithm



(b) Function g

Figure 5.9 AES Key Expansion

Figure 5.9 illustrates the generation of the expanded key, using the symbol g to represent that complex function.

AES Key Expansion

Values of rc_i in hexadecimal										
i	1	2	3	4	5	6	7	8	9	A
rc_i	01	02	04	08	10	20	40	80	1B	36

- The function g consists of the following subfunctions:

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, $Rcon[j]$.

```

KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++) w[i] = (key[4*i], key[4*i+1],
                                     key[4*i+2],
                                     key[4*i+3]);

    for (i = 4; i < 44; i++)
    {
        temp = w[i - 1];
        if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                                $\oplus$  Rcon[i/4];
        w[i] = w[i-4]  $\oplus$  temp
    }
}

```

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The pseudocode on the next page describes the expansion.

The key is copied into the first 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, $w[i - 4]$. 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.

Key Expansion Rationale

- The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks
- Inclusion of a round-dependent round constant eliminates the symmetry between the ways in which round keys are generated in different rounds

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
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
- Simplicity of description

The Rijndael developers designed the expansion key algorithm 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]

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
- An invertible transformation [i.e., knowledge of any N_k consecutive words of the expanded key enables regeneration of the entire expanded key ($N_k = \text{key size in words}$)].
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, 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.

The authors do not quantify the first point on the preceding list, but the idea is that if you know less than Nk consecutive words of either the cipher key or one of the round keys, then it is difficult to reconstruct the remaining unknown bits. The fewer bits one knows, the more difficult it is to do the reconstruction or to determine other bits in the key expansion.

AES Example Key Expansion

This table shows the expansion of the 16-byte key into 10 round keys.

This process is performed word by word, with each four-byte word occupying one column of the word round-key matrix.

The left-hand column shows the four round-key words generated for each round.

The right-hand column shows the steps used to generate the auxiliary word used in key expansion.

We begin with the key itself serving as the round key for round 0.

Key Words	Auxiliary Function
w0 = 0f 15 71 c9	RotWord(w3)= 7f 67 98 af = x1
w1 = 47 d9 e8 59	SubWord(x1)= d2 85 46 79 = y1
w2 = 0c b7 ad d6	Rcon(1)= 01 00 00 00
w3 = af 7f 67 98	y1 ⊕ Rcon(1)= d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0	RotWord(w7)= 81 15 a7 38 = x2
w5 = w4 ⊕ w1 = 9b 49 df e9	SubWord(x4)= 0c 59 5c 07 = y2
w6 = w5 ⊕ w2 = 97 fe 72 3f	Rcon(2)= 02 00 00 00
w7 = w6 ⊕ w3 = 38 81 15 a7	y2 ⊕ Rcon(2)= 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7	RotWord(w11)= ff d3 c6 e6 = x3
w9 = w8 ⊕ w5 = 49 80 b4 5e	SubWord(x2)= 16 66 b4 8e = y3
w10 = w9 ⊕ w6 = de 7e c6 61	Rcon(3)= 04 00 00 00
w11 = w10 ⊕ w7 = e6 ff d3 c6	y3 ⊕ Rcon(3)= 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39	RotWord(w15)= ae 7e c0 b1 = x4
w13 = w12 ⊕ w9 = 89 2f 6b 67	SubWord(x3)= e4 f3 ba c8 = y4
w14 = w13 ⊕ w10 = 57 51 ad 06	Rcon(4)= 08 00 00 00
w15 = w14 ⊕ w11 = b1 ae 7e c0	y4 ⊕ Rcon(4)= ec f3 ba c8 = z4
w16 = w12 ⊕ z4 = 2c 5c 65 f1	RotWord(w19)= 8c dd 50 43 = x5
w17 = w16 ⊕ w13 = a5 73 0e 96	SubWord(x4)= 64 c1 53 1a = y5
w18 = w17 ⊕ w14 = f2 22 a3 90	Rcon(5)= 10 00 00 00
w19 = w18 ⊕ w15 = 43 8c dd 50	y5 ⊕ Rcon(5)= 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb	RotWord(w23)= 40 46 bd 4c = x6
w21 = w20 ⊕ w17 = fd ee 38 7d	SubWord(x5)= 09 5a 7a 29 = y6
w22 = w21 ⊕ w18 = 0f cc 9b ed	Rcon(6)= 20 00 00 00
w23 = w22 ⊕ w19 = 4c 40 46 bd	y6 ⊕ Rcon(6)= 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2	RotWord(w27)= a5 a9 ef cf = x7
w25 = w24 ⊕ w21 = 8c 29 74 bf	SubWord(x6)= 06 d3 df 8a = y7
w26 = w25 ⊕ w22 = 83 e5 ef 52	Rcon(7)= 40 00 00 00
w27 = w26 ⊕ w23 = cf a5 a9 ef	y7 ⊕ Rcon(7)= 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48	RotWord(w31)= 7d a1 4a f7 = x8
w29 = w28 ⊕ w25 = bb 3d e7 f7	SubWord(x7)= ff 32 d6 68 = y8
w30 = w29 ⊕ w26 = 38 d8 08 a5	Rcon(8)= 80 00 00 00
w31 = w30 ⊕ w27 = f7 7d a1 4a	y8 ⊕ Rcon(8)= 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20	RotWord(w35)= be 0b 38 3c = x9
w33 = w32 ⊕ w29 = f3 1b a2 d7	SubWord(x8)= ae 2b 07 eb = y9
w34 = w33 ⊕ w30 = cb c3 aa 72	Rcon(9)= 1b 00 00 00
w35 = w34 ⊕ w31 = 3c be 0b 38	y9 ⊕ Rcon(9)= b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb	RotWord(w39)= 6b 41 56 f9 = x10
w37 = w36 ⊕ w33 = 0e 16 e0 1c	SubWord(x9)= 7f 83 b1 99 = y10
w38 = w37 ⊕ w34 = c5 d5 4a 6e	Rcon(10)= 36 00 00 00
w39 = w38 ⊕ w35 = f9 6b 41 56	y10 ⊕ Rcon(10)= 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52	
w41 = w40 ⊕ w37 = ba 98 13 4e	
w42 = w41 ⊕ w38 = 7f 4d 59 20	
w43 = w42 ⊕ w39 = 86 26 18 76	

Table 5.3

D2	b5	46	79
01	00	00	00

11010010	10110101	01000110	01111001
00000001	00000000	00000000	00000000
11010011	10110101	01000110	01111001
D3	B5	46	79

$RCON[i] = [x^{i-1}, 00, 00, 00]$ $RCON[i] = [x^{i-1}, 00, 00, 00]$. The values x^{i-1} are to be computed in the same representation of $GF(2^8)$

Table 5.3 shows the expansion of the 16-byte key into 10 round keys. As previously explained, this process is performed word by word, with each four-byte word occupying one column of the word round-key matrix. The left-hand column shows the four round-key words generated for each round. The right-hand column shows the steps used to generate the auxiliary word used in key expansion. We begin, ofcourse, with the key itself serving as the round key for round 0.

0=0000
1=0001
2=0010
3=0011
4=0100
5=0101
6=0110
7=0111
8=1000
9=1001
A=10=1010

AES Ex Key expansion			Key Words	Auxiliary Function
Decimal(10)	Binary(2)	Hexadecimal(16)		
0	0000	0	w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad d6 w3 = af 7f 67 98	RotWord(w3)= 7f 67 98 af = x1 SubWord(x1)= d2 85 46 79 = y1 Rcon(1)= 01 00 00 00 y1 ⊕ Rcon(1)= d3 85 46 79 = z1
1	0001	1	w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord(w7)= 81 15 a7 38 = x2 SubWord(x4)= 0c 59 5c 07 = y2 Rcon(2)= 02 00 00 00 y2 ⊕ Rcon(2)= 0e 59 5c 07 = z2
2	0010	2	w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord(w11)= ff d3 c6 e6 = x3 SubWord(x2)= 16 66 b4 8e = y3 Rcon(3)= 04 00 00 00 y3 ⊕ Rcon(3)= 12 66 b4 8e = z3
3	0011	3	w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0	RotWord(w15)= ae 7e c0 b1 = x4 SubWord(x3)= e4 f3 ba c8 = y4 Rcon(4)= 08 00 00 00 y4 ⊕ Rcon(4)= ec f3 ba c8 = z4
4	0100	4	w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord(w19)= 8c dd 50 43 = x5 SubWord(x4)= 64 c1 53 1a = y5 Rcon(5)= 10 00 00 00 y5 ⊕ Rcon(5)= 74 c1 53 1a = z5
5	0101	5	w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord(w23)= 40 46 bd 4c = x6 SubWord(x5)= 09 5a 7a 29 = y6 Rcon(6)= 20 00 00 00 y6 ⊕ Rcon(6)= 29 5a 7a 29 = z6
6	0110	6	w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord(w27)= a5 a9 ef cf = x7 SubWord(x6)= 06 d3 df 8a = y7 Rcon(7)= 40 00 00 00 y7 ⊕ Rcon(7)= 46 d3 df 8a = z7
7	0111	7	w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord(w31)= 7d a1 4a f7 = x8 SubWord(x7)= ff 32 d6 68 = y8 Rcon(8)= 80 00 00 00 y8 ⊕ Rcon(8)= 7f 32 d6 68 = z8
8	1000	8	w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w31 = 3c be 0b 38	RotWord(w35)= be 0b 38 3c = x9 SubWord(x8)= ae 2b 07 eb = y9 Rcon(9)= 1b 00 00 00 y9 ⊕ Rcon(9)= b5 2b 07 eb = z9
9	1001	9	w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord(w39)= 6b 41 56 f9 = x10 SubWord(x9)= 7f 83 b1 99 = y10 Rcon(10)= 36 00 00 00 y10 ⊕ Rcon(10)= 49 83 b1 99 = z10
10	1010	A	w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	
11	1011	B		
12	1100	C		
13	1101	D		
14	1110	E		
15	1111	F		

Table 5.3

$RCON[i] = [x^{i-1}, 00, 00, 00]$ $RCON[i] = [x^{i-1}, 00, 00, 00]$. The values x^{i-1} are to be computed in the same representation of $GF(2^8)$

Table 5.3 shows the expansion of the 16-byte key into 10 round keys. As previously explained, this process is performed word by word, with each four-byte word occupying one column of the word round-key matrix. The left-hand column shows the four round-key words generated for each round. The right-hand column shows the steps used to generate the auxiliary word used in key expansion. We begin, ofcourse, with the key itself serving as the round key for round 0.

0=0000

1=0001

2=0010

3=0011

4=0100

5=0101

6=0110

7=0111

$$8=1000$$

$$9=1001$$

$$A=10=1010$$

Table 5.4 AES

Example

Plaintext:	0123456789abcdefedcba9876543210
Key:	0f1571c947d9e8590cb7add6af7f6798
Ciphertext:	ff0b844a0853bf7c6934ab4364148fb9

This table shows the progression of State through the AES encryption process.

The first column shows the value of State at the start of a round.

For the first row, State is just the matrix arrangement of the plaintext.

The second, third, and fourth columns show the value of State for that round after the SubBytes, ShiftRows, and MixColumns transformations, respectively.

The fifth column shows the round key. You can verify that these round keys equate with those shown in Table 5.3.

The first column shows the value of State resulting from the bitwise XOR of State after

the preceding MixColumns with the round key for the preceding round.

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

Next, Table 5.4 shows the progression of State through the AES encryption process. The first column shows the value of State at the start of a round. For the first row, State is just the matrix arrangement of the plaintext. The second, third, and fourth columns show the value of State for that round after the SubBytes, ShiftRows, and MixColumns transformations, respectively. The fifth column shows the round key. You can verify that these round keys equate with those shown in Table 5.3. The first column shows the value of State resulting from the bitwise XOR of State after the preceding MixColumns with the round key for the preceding round.

Finite Field-2⁸- Arithmetic Operations

- Over 8 bits.
- +, -, XOR are the same.
 - ✓ $(1+1=1-1=1\text{xor}1=0)$
- Multiplication:
 - Polynomial
 - **Ex:** $10101110 = x^7 + x^5 + x^3 + x^2 + x^1$
 - $10101111 = x^7 + x^5 + x^3 + x^2 + x^1 + 1$
 - Multiply
 - Consider: $x^8 = x^4 + x^3 + x + 1$
 - Ex:** $03 * C5 =$
 - $(00000011) * (11000101) = (x+1) * (x^7 + x^6 + x^2 + 1) = x^8 + x^7 + x^3 + x + x^7 + x^6 + x^2 + 1 =$
 - $x^4 + x^3 + x + 1 + x^7 + x^3 + x + x^7 + x^6 + x^2 + 1 = x^6 + x^4 + x^2 = 0101\ 0100$

Table 5.5

Avalanche Effect in AES: Change in Plaintext

Round		Number of Bits that Differ
	↓ 0123456789abcdef fedcba9876543210 0023456789abcdef fedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d689017507d485e62	59
4	f867aee8b437a5210c24c1974cffeabc 43efdb697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcdbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36ald891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

If a small change in the key or plaintext were to produce a corresponding small change in the ciphertext, this might be used to effectively reduce the size of the plaintext (or key) space to be searched. What is desired is the avalanche effect, in which a small change in plaintext or key produces a large change in the ciphertext.

Using the example from Table 5.4, Table 5.5 shows the result when the eighth bit of the plaintext is changed. The second column of the table shows the value of the State matrix at the end of each round for the two plaintexts. Note that after just one round, 20 bits of the State vector differ. After two rounds, close to half the bits differ. This magnitude of difference propagates through the remaining rounds. A bit difference in approximately half the positions in the most desirable outcome. Clearly, if almost all the bits are changed, this would be logically equivalent to almost none of the bits being changed. Put another way, if we select two plaintexts at random, we would expect the two plaintexts to differ in about half of the bit positions and the two ciphertexts to also differ in about half the positions.

Table 5.6

Avalanche
Effect
in AES:
Change
in Key

Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0123456789abcdeffedcba9876543210	0
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c5a9ad090ec7ff3fc1e8e8ca4cd02a9c	22
2	5c7bb49a6b72349b05a2317ff46d1294 90905fa9563356d15f3760f3b8259985	58
3	7115262448dc747e5cdac7227da9bd9c 18aeb7aa794b3b66629448d575c7cebf	67
4	f867aee8b437a5210c24c1974cffeabc f81015f993c978a876ae017cb49e7eec	63
5	721eb200ba06206dcbd4bce704fa654e 5955c91b4e769f3cb4a94768e98d5267	81
6	0ad9d85689f9f77bc1c5f71185e5fb14 dc60a24d137662181e45b8d3726b2920	70
7	db18a8ffa16d30d5f88b08d777ba4eaa fe8343b8f88bef66cab7e977d005a03c	74
8	f91b4fbfe934c9bf8f2f85812b084989 da7dad581d1725c5b72fa0f9d9d1366a	67
9	cca104a13e678500ff59025f3bafaa34 0ccb4c66bbfd912f4b511d72996345e0	59
10	ff0b844a0853bf7c6934ab4364148fb9 fc8923ee501a7d207ab670686839996b	53

Table 5.6 shows the change in State matrix values when the same plaintext is used and the two keys differ in the eighth bit. That is, for the second case, the key is 0e1571c947d9e8590cb7add6af7f6798. Again, one round produces a significant change, and the magnitude of change after all subsequent rounds is roughly half the bits. Thus, based on this example, AES exhibits a very strong avalanche effect.

Note that this avalanche effect is stronger than that for DES (Table 3.2), which **requires three rounds to reach a point at which approximately half the bits are changed, both for a bit change in the plaintext and a bit change in the key.**

Equivalent Inverse Cipher

- AES decryption cipher is not identical to the encryption cipher

- The sequence of transformations differs although the form of the key schedules is the same
- Has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption

Two separate changes are needed to bring the decryption structure in line with the encryption structure

The first two stages of the decryption round need to be interchanged

The second two stages of the decryption round need to be interchanged

As was mentioned, the AES decryption cipher is not identical to the encryption cipher (Figure 5.3). That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

Two separate changes are needed to bring the decryption structure in line with the encryption structure. As illustrated in Figure 5.3, an encryption round has the structure **SubBytes, ShiftRows, MixColumns, AddRoundKey**. The standard decryption round has the 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**.

Interchanging AddRoundKey and InvMixColumns

The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State

If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on State one column at a time

These two operations are linear with respect to the column input

The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State . If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on State one column at a time. These two operations are linear with respect to the column input.

Interchanging InvShiftRows and InvSubBytes

- InvShiftRows *affects the sequence* of bytes in State but *does not alter byte contents* and *does not depend on byte contents* to perform its transformation
- InvSubBytes *affects the contents* of bytes in State but *does not alter byte sequence* and *does not depend on byte sequence* to perform its transformation

Thus, these two operations commute and can be interchanged

InvShiftRows affects the sequence of bytes in State but does not alter byte contents and does not depend on byte contents to perform its transformation. InvSubBytes affects the contents of bytes in State but does not alter byte sequence and does not depend on byte sequence to perform its transformation. Thus, these two operations commute and can be interchanged.

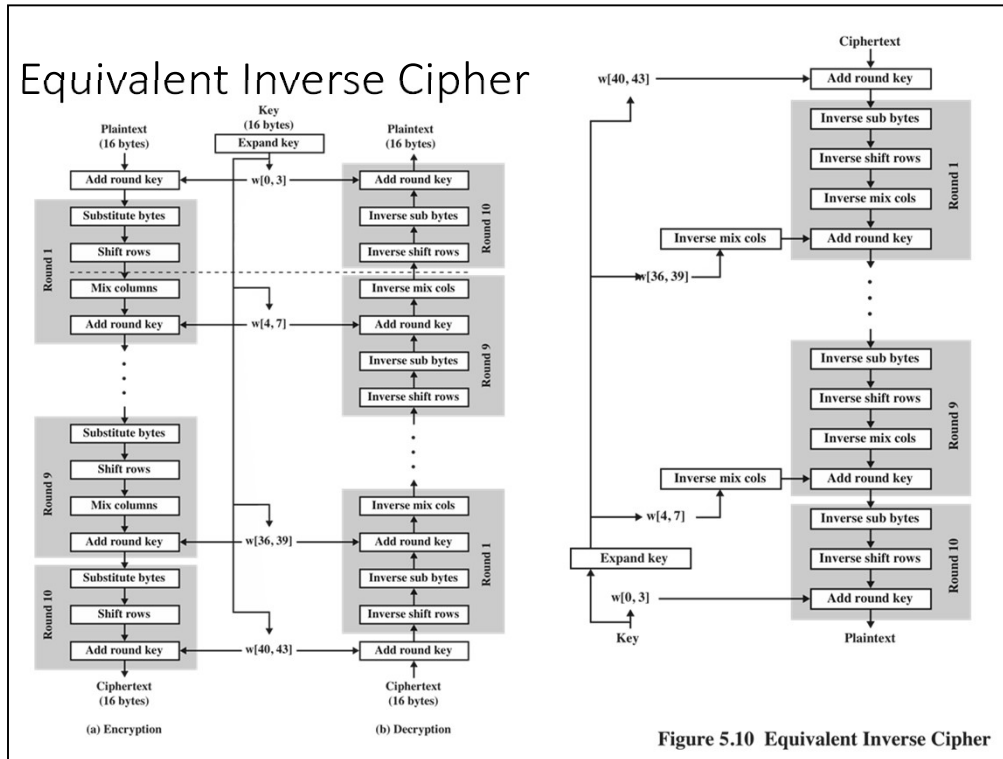


Figure 5.10 illustrates the equivalent decryption algorithm.

Implementation Aspects

- AES can be implemented very efficiently on an 8-bit processor
- AddRoundKey is a bitwise 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

The Rijndael proposal [DAEM99] provides some suggestions for efficient implementation on 8-bit processors, typical for current smart cards, and on 32-bit processors, typical for PCs.

AES can be implemented very efficiently on an 8-bit processor. AddRoundKey is a bitwise XOR operation. ShiftRows is a simple byteshifting operation. SubBytes operates at the byte level and only requires a table of 256 bytes.

The transformation MixColumns requires matrix multiplication in the field $GF(2^8)$, which means that all operations are carried out on bytes. MixColumns only requires multiplication by {02} and {03}, which, as we have seen, involved simple shifts, conditional XORs, and XORs. This can be implemented in a more efficient way that eliminates the shifts and conditional XORs.

Implementation Aspects

- Can efficiently implement on a 32-bit processor
 - 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

The implementation described in the preceding subsection uses only 8-bit operations. For a 32-bit processor, a more efficient implementation can be achieved if operations are defined on 32-bit words. To show this, we first define the four transformations of a round in algebraic form.

The developers of Rijndael believe that this compact, efficient implementation was probably one of the most important factors in the selection of Rijndael for AES.

Summary

- Finite field arithmetic
- AES structure
 - General structure
 - Detailed structure
- AES key expansion
 - Key expansion algorithm
 - Rationale
- AES transformation functions
 - Substitute bytes
 - ShiftRows
 - MixColumns
 - AddRoundKey
- AES implementation
 - Equivalent inverse cipher
 - Implementation aspects

Tools:

- <https://www.devglan.com/online-tools/aes-encryption-decryption>
- <https://aesencryption.net>
- <http://aes.online-domain-tools.com>

Course:

- https://www.youtube.com/watch?v=NHuibtoL_qk (1h:33min:18sec)
- <https://www.coursera.org/lecture/symmetric-crypto/advanced-encryption-standard-aes-Z7apS> (7min:13sec)

Chapter 5 summary.