Chapter 8 The Advanced Encryption Standard: Rijndael

In 1997, the National Institute of Standards and Technology put out a call for candidates to replace DES. Among the requirements were that the new algorithm should allow key sizes of 128, 192, and 256 bits, it should operate on blocks of 128 input bits, and it should work on a variety of different hardware, for example, eight-bit processors that could be used in smart cards and the 32-bit architecture commonly used in personal computers. Speed and cryptographic strength were also important considerations. In 1998, the cryptographic community was asked to comment on 15 candidate algorithms. Five finalists were chosen: MARS (from IBM), RC6 (from RSA Laboratories), Rijndael (from Joan Daemen and Vincent Rijmen), Serpent (from Ross Anderson, Eli Biham, and Lars Knudsen), and Twofish (from Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson). Eventually, Rijndael was chosen as the Advanced Encryption Standard. The other four algorithms are also very strong, and it is likely that they will used in many future cryptosystems.

As with other block ciphers, Rijndael can be used in several modes, for example, ECB, CBC, CFB, OFB, and CTR (see Section 6.3).

Before proceeding to the algorithm, we answer a very basic question: How do you pronounce Rijndael? We quote from their Web page:

If you're Dutch, Flemish, Indonesian, Surinamer or South-African, it's pronounced like you think it should be. Otherwise, you could pronounce it like "Reign Dahl,"

“Rain Doll,” “Rhine Dahl”. We’re not picky. As long as you make it sound different from “Region Deal.”

8.1 The Basic Algorithm

Rijndael is designed for use with keys of lengths 128, 192, and 256 bits. For simplicity, we'll restrict to 128 bits. First, we give a brief outline of the algorithm, then describe the various components in more detail.

The algorithm consists of 10 rounds (when the key has 192 bits, 12 rounds are used, and when the key has 256 bits, 14 rounds are used). Each round has a round key, derived from the original key. There is also a 0th round key, which is the original key. A round starts with an input of 128 bits and produces an output of 128 bits.

There are four basic steps, called **layers**, that are used to form the rounds:

1. The SubBytes Transformation (SB): This nonlinear layer is for resistance to differential and linear cryptanalysis attacks.
2. The ShiftRows Transformation (SR): This linear mixing step causes diffusion of the bits over multiple rounds.
3. The MixColumns Transformation (MC): This layer has a purpose similar to ShiftRows.
4. AddRoundKey (ARK): The round key is *XOR*ed with the result of the above layer.

A round is then

$$\rightarrow \boxed{SubBytes} \rightarrow \boxed{ShiftRows} \rightarrow \boxed{MixColumns} \rightarrow \boxed{AddRoundKey} \rightarrow .$$

Putting everything together, we obtain the following (see also Figure 8.1):

Figure 8.1The AES-Rijndael Algorithm

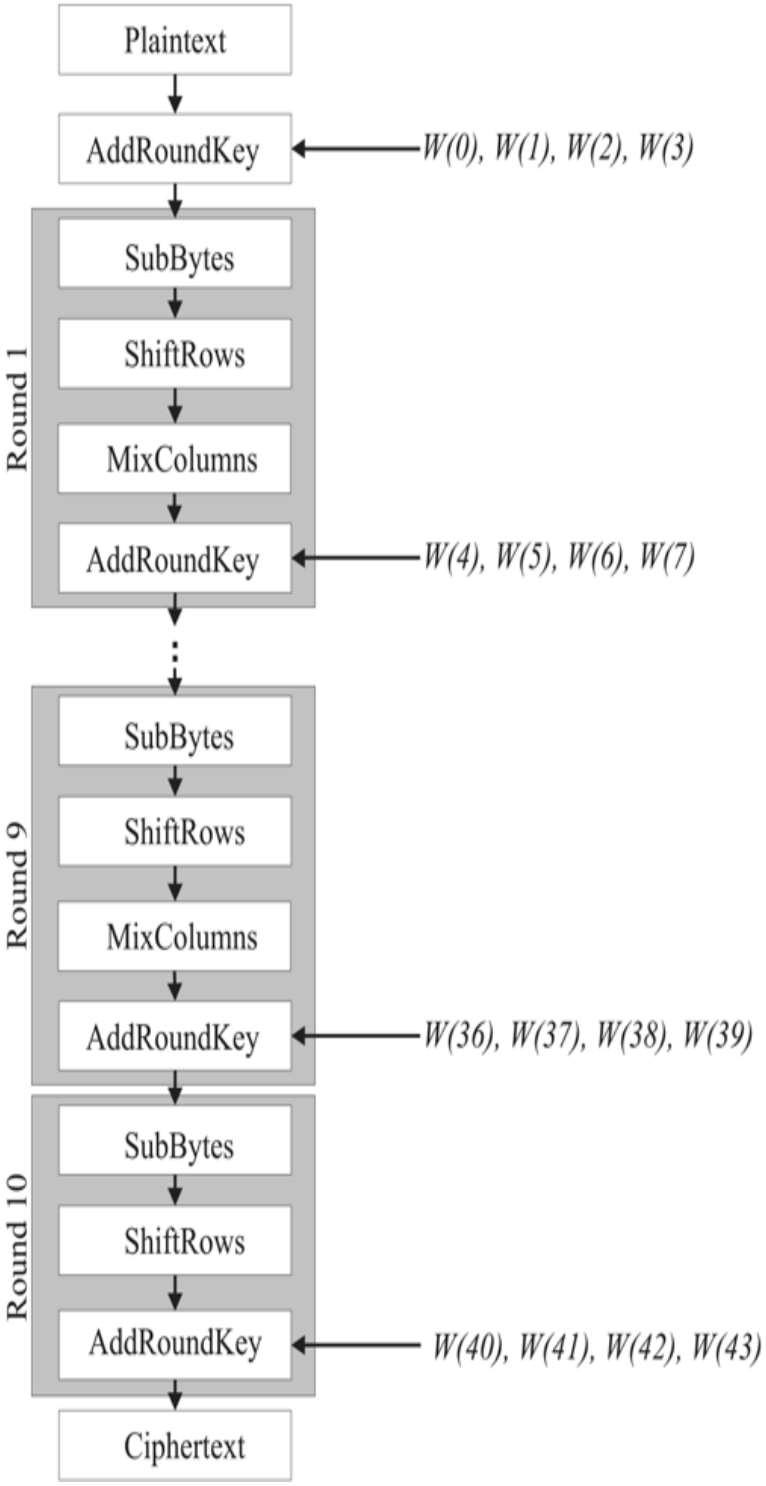
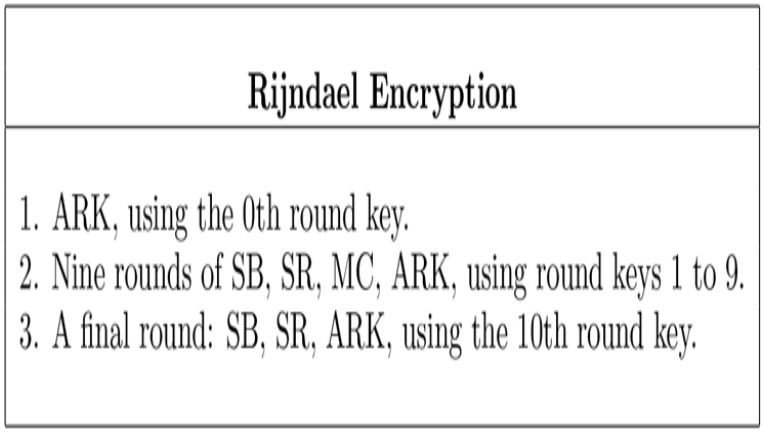


Figure 8.1 Full Alternative Text



8.1-1 Full Alternative Text

The final round uses the SubBytes, ShiftRows, and AddRoundKey steps but omits MixColumns (this omission will be explained in the decryption section).

The 128-bit output is the ciphertext block.

8.2 The Layers

We now describe the steps in more detail. The 128 input bits are grouped into 16 bytes of eight bits each, call them

$$a_{0,0},\ a_{1,0},\ a_{2,0},\ a_{3,0},\ a_{0,1},\ a_{1,1},\ \ldots,\ a_{3,3}.$$

These are arranged into a 4×4 matrix

$$\begin{matrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{matrix}.$$

In the following, we'll occasionally need to work with the finite field $GF(2^8)$. This is covered in Section 3.11. However, for the present purposes, we only need the following facts. The elements of $GF(2^8)$ are bytes, which consist of eight bits. They can be added by *XOR*. They can also be multiplied in a certain way (i.e., the product of two bytes is again a byte), but this process is more complicated. Each byte b except the zero byte has a multiplicative inverse; that is, there is a byte b' such that $b \cdot b' = 00000001$. Since we can do arithmetic operations on bytes, we can work with matrices whose entries are bytes.

As a technical point, we note that the model of $GF(2^8)$ depends on a choice of irreducible polynomial of degree 8. The choice for Rijndael is $X^8 + X^4 + X^3 + X + 1$. This is also the polynomial used in the examples in Section 3.11. Other choices for this polynomial would presumably give equally good algorithms.

8.2.1 The SubBytes Transformation

In this step, each of the bytes in the matrix is changed to another byte by Table 8.1, called the S-box.

Table 8.1 S-Box for Rijndael

S-Box															
99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

Table 8.1 Full Alternative Text

Write a byte as eight bits: *abcdefgh*. Look for the entry in the *abcd* row and *efgh* column (the rows and columns are numbered from 0 to 15). This entry, when converted to binary, is the output. For example, if the input byte is 10001011, we look in row 8 (the ninth row) and column 11 (the twelfth column). The entry is 61, which is 111101 in binary. This is the output of the S-box.

The output of SubBytes is again a 4×4 matrix of bytes, let's call it

$$\begin{matrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{matrix}.$$

8.2.2 The ShiftRows Transformation

The four rows of the matrix are shifted cyclically to the left by offsets of 0, 1, 2, and 3, to obtain

$$\begin{matrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{matrix} = \begin{matrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{matrix}.$$

8.2.3 The MixColumns Transformation

Regard a byte as an element of $GF(2^8)$, as in Section 3.11. Then the output of the ShiftRows step is a 4×4 matrix $(c_{i,j})$ with entries in $GF(2^8)$. Multiply this by a matrix, again with entries in $GF(2^8)$, to produce the output $(d_{i,j})$, as follows:

$$\begin{matrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{matrix} \begin{matrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{matrix}$$

$$= \begin{matrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{matrix}.$$

8.2.4 The RoundKey Addition

The round key, derived from the key in a way we'll describe later, consists of 128 bits, which are arranged in a 4×4 matrix $(k_{i,\,j})$ consisting of bytes. This is *XOR*ed with the output of the MixColumns step:

$$\begin{matrix} d_{0,\,0} & d_{0,\,1} & d_{0,\,2} & d_{0,\,3} \\ d_{1,\,0} & d_{1,\,1} & d_{1,\,2} & d_{1,\,3} \\ d_{2,\,0} & d_{2,\,1} & d_{2,\,2} & d_{2,\,3} \\ d_{3,\,0} & d_{3,\,1} & d_{3,\,2} & d_{3,\,3} \end{matrix} \oplus \begin{matrix} k_{0,\,0} & k_{0,\,1} & k_{0,\,2} & k_{0,\,3} \\ k_{1,\,0} & k_{1,\,1} & k_{1,\,2} & k_{1,\,3} \\ k_{2,\,0} & k_{2,\,1} & k_{2,\,2} & k_{2,\,3} \\ k_{3,\,0} & k_{3,\,1} & k_{3,\,2} & k_{3,\,3} \end{matrix}$$

$$= \begin{matrix} e_{0,\,0} & e_{0,\,1} & e_{0,\,2} & e_{0,\,3} \\ e_{1,\,0} & e_{1,\,1} & e_{1,\,2} & e_{1,\,3} \\ e_{2,\,0} & e_{2,\,1} & e_{2,\,2} & e_{2,\,3} \\ e_{3,\,0} & e_{3,\,1} & e_{3,\,2} & e_{3,\,3} \end{matrix}.$$

This is the final output of the round.

8.2.5 The Key Schedule

The original key consists of 128 bits, which are arranged into a 4×4 matrix of bytes. This matrix is expanded by adjoining 40 more columns, as follows. Label the first four columns $W(0)$, $W(1)$, $W(2)$, $W(3)$. The new columns are generated recursively. Suppose columns up through $W(i-1)$ have been defined. If i is not a multiple of 4, then

$$W(i) = W(i-4) \oplus W(i-1).$$

If i is a multiple of 4, then

$$W(i) = W(i-4) \oplus T(W(i-1)),$$

where $T(W(i-1))$ is the transformation of $W(i-1)$ obtained as follows. Let the elements of the column $W(i-1)$ be a, b, c, d. Shift these cyclically to obtain b, c, d, a. Now replace each of these bytes with the corresponding element in the S-box from the SubBytes step, to get 4 bytes e, f, g, h. Finally, compute the round constant

$$r(i) = 00000010^{(i-4)/4}$$

in $GF(2^8)$ (recall that we are in the case where i is a multiple of 4). Then $T(W(i-1))$ is the column vector

$$(e \oplus r(i),\ f,\ g,\ h).$$

In this way, columns $W(4),\ \ldots,\ W(43)$ are generated from the initial four columns.

The **round key** for the ith round consists of the columns

$$W(4i), W(4i+1), W(4i+2), W(4i+3).$$

8.2.6 The Construction of the S-Box

Although the S-box is implemented as a lookup table, it has a simple mathematical description. Start with a byte $x_7x_6x_5x_4x_3x_2x_1x_0$, where each x_i is a binary bit. Compute its inverse in $GF(2^8)$, as in Section 3.11. If the byte is 00000000, there is no inverse, so we use 00000000 in place of its inverse. The resulting byte $y_7y_6y_5y_4y_3y_2y_1y_0$ represents an eight-dimensional column vector, with the rightmost bit y_0 in the top position. Multiply by a matrix and add the column vector $(1,\ 1,\ 0,\ 0,\ 0,\ 1,\ 1,\ 0)$ to obtain a vector $(z_0,\ z_1,\ z_2,\ z_3,\ z_4,\ z_5,\ z_6,\ z_7)$ as follows:

$$\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array} \begin{array}{c} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{array} + \begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{array} = \begin{array}{c} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{array}.$$

The byte $z_7z_6z_5z_4z_3z_2z_1z_0$ is the entry in the S-box.

For example, start with the byte 11001011. Its inverse in $GF(2^8)$ is 00000100, as we calculated in Section

3.11. We now calculate

$$\begin{matrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{matrix} \begin{matrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} + \begin{matrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{matrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{matrix}.$$

This yields the byte 00011111. The first four bits 1100 represent 12 in binary and the last four bits 1011 represent 11 in binary. Add 1 to each of these numbers (since the first row and column are numbered 0) and look in the 13th row and 12th column of the S-box. The entry is 31, which in binary is 00011111.

Some of the considerations in the design of the S-box were the following. The map $x \mapsto x^{-1}$ was used to achieve nonlinearity. However, the simplicity of this map could possibly allow certain attacks, so it was combined with multiplication by the matrix and adding the vector, as described previously. The matrix was chosen mostly because of its simple form (note how the rows are shifts of each other). The vector was chosen so that no input ever equals its S-box output or the complement of its S-box output (complementation means changing each 1 to 0 and each 0 to 1).

8.3 Decryption

Each of the steps SubBytes, ShiftRows, MixColumns, and AddRoundKey is invertible:

1. The inverse of SubBytes is another lookup table, called **InvSubBytes**.
2. The inverse of ShiftRows is obtained by shifting the rows to the right instead of to the left, yielding **InvShiftRows**.
3. The inverse of MixColumns exists because the 4×4 matrix used in MixColumns is invertible. The transformation **InvMixColumns** is given by multiplication by the matrix

$$\begin{matrix} 00001110 & 00001011 & 00001101 & 00001001 \\ 00001001 & 00001110 & 00001011 & 00001101 \\ 00001101 & 00001001 & 00001110 & 00001011 \\ 00001011 & 00001101 & 00001001 & 00001110 \end{matrix}.$$

4. AddRoundKey is its own inverse.

The Rijndael encryption consists of the steps

ARK

SB, SR, MC, ARK

...

SB, SR, MC, ARK

SB, SR, ARK.

Recall that MC is missing in the last round.

To decrypt, we need to run through the inverses of these steps in the reverse order. This yields the following preliminary version of decryption:

ARK, ISR, ISB

ARK, IMC, ISR, ISB

...

ARK, IMC, ISR, ISB

ARK.

However, we want to rewrite this decryption in order to make it look more like encryption.

Observe that applying SB then SR is the same as first applying SR then SB. This happens because SB acts one byte at a time and SR permutes the bytes. Correspondingly, the order of ISR and ISB can be reversed.

We also want to reverse the order of ARK and IMC, but this is not possible. Instead, we proceed as follows. Applying MC and then ARK to a matrix $(c_{i,j})$ is given as

$$(c_{i,j}) \rightarrow (m_{i,j})(c_{i,j}) \rightarrow (e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j}),$$

where $(m_{i,j})$ is a the 4×4 matrix in MixColumns and $(k_{i,j})$ is the round key matrix. The inverse is obtained by solving $(e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j})$ for $(c_{i,j})$ in terms of $(e_{i,j})$, namely,
$(c_{i,j}) = (m_{i,j})^{-1}(e_{i,j}) \oplus (m_{i,j})^{-1}(k_{i,j})$. Therefore, the process is

$$(e_{i,j}) \rightarrow (m_{i,j})^{-1}(e_{i,j}) \rightarrow (m_{i,j})^{-1}(e_{i,j}) \oplus (k'_{i,j}),$$

where $(k'_{i,j}) = (m_{i,j})^{-1}(k_{i,j})$. The first arrow is simply InvMixColumns applied to $(e_{i,j})$. If we let **InvAddRoundKey** be *XOR*ing with $(k'_{i,j})$, then we have that the inverse of “MC then ARK” is “ IMC then IARK.” Therefore, we can replace the steps “ARK then IMC” with the steps “IMC then IARK” in the preceding decryption sequence.

We now see that decryption is given by

ARK, ISB, ISR

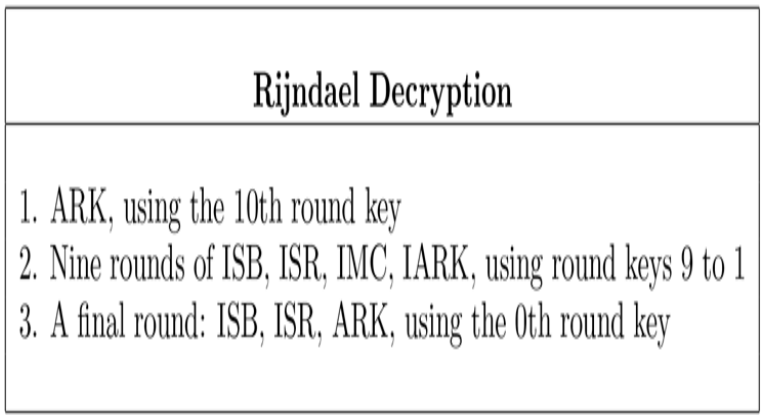
IMC, IARK, ISB, ISR

...

IMC, IARK, ISB, ISR

ARK.

Regroup the lines to obtain the final version:



8.3-3 Full Alternative Text

Therefore, the decryption is given by essentially the same structure as encryption, but SubBytes, ShiftRows, and MixColumns are replaced by their inverses, and AddRoundKey is replaced by InvAddRoundKey, except in the initial and final steps. Of course, the round keys are used in the reverse order, so the first ARK uses the 10th round key, and the last ARK uses the 0th round key.

The preceding shows why the MixColumns is omitted in the last round. Suppose it had been left in. Then the encryption would start ARK, SB, SR, MC, ARK, ..., and it would end with ARK, SB, SR, MC, ARK. Therefore, the beginning of the decryption would be (after the reorderings) IMC, IARK, ISB, ISR, ... This means the decryption would have an unnecessary IMC at the beginning, and this would have the effect of slowing down the algorithm.

Another way to look at encryption is that there is an initial ARK, then a sequence of alternating half rounds

$$(\text{SB}, \text{SR}), (\text{MC}, \text{ARK}), (\text{SB}, \text{SR}), \ldots, (\text{MC}, \text{ARK}), (\text{SB}, \text{SR}),$$

followed by a final ARK. The decryption is ARK, followed by a sequence of alternating half rounds

$$(\text{ISB}, \text{ISR}), (\text{IMC}, \text{IARK}), (\text{ISB}, \text{ISR}), \ldots, (\text{IMC}, \text{IARK}), (\text{ISB}, \text{ISR}),$$

followed by a final ARK. From this point of view, we see that a final MC would not fit naturally into any of the half rounds, and it is natural to leave it out.

On eight-bit processors, decryption is not quite as fast as encryption. This is because the entries in the 4×4 matrix for InvMixColumns are more complex than those for MixColumns, and this is enough to make decryption take around 30% longer than encryption for these processors. However, in many applications, decryption is not needed, for example, when CFB mode (see Section 6.3) is used. Therefore, this is not considered to be a significant drawback.

The fact that encryption and decryption are not identical processes leads to the expectation that there are no weak keys, in contrast to DES (see Exercise 5 in Chapter 7) and several other algorithms.

8.4 Design Considerations

The Rijndael algorithm is not a Feistel system (see Sections 7.1 and 7.2). In a Feistel system, half the bits are moved but not changed during each round. In Rijndael, all bits are treated uniformly. This has the effect of diffusing the input bits faster. It can be shown that two rounds are sufficient to obtain full diffusion, namely, each of the 128 output bits depends on each of the 128 input bits.

The S-box was constructed in an explicit and simple algebraic way so as to avoid any suspicions of trapdoors built into the algorithm. The desire was to avoid the mysteries about the S-boxes that haunted DES. The Rijndael S-box is highly nonlinear, since it is based on the mapping $x \mapsto x^{-1}$ in $GF(2^8)$. It is excellent at resisting differential and linear cryptanalysis, as well as more recently studied methods called interpolation attacks.

The ShiftRows step was added to resist two recently developed attacks, namely truncated differentials and the Square attack (Square was a predecessor of Rijndael).

The MixColumns causes diffusion among the bytes. A change in one input byte in this step always results in all four output bytes changing. If two input bytes are changed, at least three output bytes are changed.

The Key Schedule involves nonlinear mixing of the key bits, since it uses the S-box. The mixing is designed to resist attacks where the cryptanalyst knows part of the key and tries to deduce the remaining bits. Also, it aims to ensure that two distinct keys do not have a large number of round keys in common. The round constants

are used to eliminate symmetries in the encryption process by making each round different.

The number of rounds was chosen to be 10 because there are attacks that are better than brute force up to six rounds. No known attack beats brute force for seven or more rounds. It was felt that four extra rounds provide a large enough margin of safety. Of course, the number of rounds could easily be increased if needed.

8.5 Exercises

1. Suppose the key for round 0 in AES consists of 128 bits, each of which is 0.

    1. Show that the key for the first round is $W(4), W(5), W(6), W(7)$, where

    $$W(4) = W(5) = W(6) = W(7) = \begin{matrix} 01100010 \\ 01100011 \\ 01100011 \\ 01100011 \end{matrix}.$$

    2. Show that $W(8) = W(10) \neq W(9) = W(11)$ (Hint: This can be done without computing $W(8)$ explicitly).

2. Suppose the key for round 0 in AES consists of 128 bits, each of which is 1.

    1. Show that the key for the first round is $W(4), W(5), W(6), W(7)$, where

    $$W(5) = W(7) = \begin{matrix} 00010111 \\ 00010110 \\ 00010110 \\ 00010110 \end{matrix},$$

    $$W(4) = W(6) = \begin{matrix} 11101000 \\ 11101001 \\ 11101001 \\ 11101001 \end{matrix}.$$

    Note that $W(5) = W(4)$ = the complement of $W(5)$ (the complement can be obtained by XORing with a string of all 1s).

    2. Show that $W(10) = W(8)$ and that $W(11) = W(9)$ (Hints: $W(5) \oplus W(6)$ is a string of all 1s. Also, the relation $A \oplus B = A \oplus B$ might be useful.)

3. Let $f(x)$ be a function from binary strings (of a fixed length N) to binary strings. For the purposes of this problem, let's say that $f(x)$ has the *equal difference property* if the following is satisfied: Whenever x_1, x_2, x_3, x_4 are binary strings of length N that satisfy $x_1 \oplus x_2 = x_3 \oplus x_4$, then

$$f(x_1) \oplus f(x_2) = f(x_3) \oplus f(x_4).$$

1. Show that if α, $\beta \in GF(2^8)$ and $f(x) = \alpha x + \beta$ for all $x \in GF(2^8)$, then $f(x)$ has the equal difference property.

2. Show that the ShiftRows Transformation, the MixColumns Transformation, and the RoundKey Addition have the equal difference property.

4.

1. Suppose we remove all SubBytes Transformation steps from the AES algorithm. Show that the resulting AES encryption would then have the equal difference property defined in Exercise 3.

2. Suppose we are in the situation of part (a), with all SubBytes Transformation steps removed. Let x_1 and x_2 be two 128-bit plaintext blocks and let $E(x_1)$ and $E(x_2)$ be their encryptions under this modified AES scheme. Show that $E(x_1) \oplus E(x_2)$ equals the result of encrypting $x_1 \oplus x_2$ using only the ShiftRows and MixColumns Transformations (that is, both the RoundKey Addition and the SubBytes Transformation are missing). In particular, $E(x_1) \oplus E(x_2)$ is independent of the key.

3. Suppose we are in the situation of part (a), and Eve knows x_1 and $E(x_1)$ for some 128-bit string x. Describe how she can decrypt any message $E(x_2)$ (your solution should be much faster than using brute force or making a list of all encryptions). (Remark: This shows that the SubBytes transformation is needed to prevent the equal difference property. See also Exercise 5.)

5. Let $x_1 = 00000000$, $x_2 = 00000001$, $x_3 = 00000010$, $x_4 = 00000011$. Let $SB(x)$ denote the SubBytes Transformation of x. Show that

   $$SB(x_1) \oplus SB(x_2) = 00011111 \neq 00001100 = SB(x_3) \oplus SB(x_4).$$

   Conclude that the SubBytes Transformation is not an affine map (that is, a map of the form $\alpha x + \beta$) from $GF(2^8)$ to $GF(2^8)$. (Hint: See Exercise 3(a).)

6. Your friend builds a very powerful computer that uses brute force to find a 56-bit DES key in 1 hour, so you make an even better machine that can try 2^{56} AES keys in 1 second. How long will this machine take to try all 2^{128} AES keys?