attack six rounds. In this respect, adding four rounds actually doubles the number of rounds through which a propagation trail has to be found.

For Rijndael versions with a longer key, the number of rounds was raised by one for every additional 32 bits in the cipher key. This was done for the following reasons:

1. One of the main objectives is the absence of shortcut attacks, i.e. attacks that are more efficient than an exhaustive key search. Since the workload of an exhaustive key search grows with the key length, shortcut attacks can afford to be less efficient for longer keys.
2. (Partially) known-key and related-key attacks exploit the knowledge of cipher key bits or the ability to apply different cipher keys. If the cipher key grows, the range of possibilities available to the cryptanalyst increases.

The publications on the security of Rijndael with longer keys have shown that this strategy leads to an adequate security margin [31, 36, 62]. For Rijndael versions with a higher block length, the number of rounds is raised by one for every additional 32 bits in the block length, for the following reasons:

1. For a block length above 128 bits, it takes three rounds to realize that full diffusion, i.e. the diffusion power of the round transformation, relative to the block length, diminishes with the block length.
2. The larger block length causes the range of possible patterns that can be applied at the input/output of a sequence of rounds to increase. This additional flexibility may allow the extension of attacks by one or more rounds.

We have found that extensions of attacks by a single round are even hard to realize for the maximum block length of 256 bits. Therefore, this is a conservative margin.

Table 3.2 lists the value of N_r as a function of N_b and N_k. For the AES, N_b is fixed to the value 4; $N_r = 10$ for 128-bit keys ($N_k = 4$), $N_r = 12$ for 192-bit keys ($N_k = 6$) and $N_r = 14$ for 256-bit keys ($N_k = 8$).

Table 3.2. Number of rounds (N_r) as a function of N_b (N_b = block length/32) and N_k (key length/32).

	N_b				
N_k	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

3.6 Key Schedule

The key schedule consists of two components: the key expansion and round key selection. The key expansion specifies how ExpandedKey is der from the cipher key. The total number of bits in ExpandedKey is equa the block length multiplied by the number of rounds plus 1, since the ci requires one round key for the initial key addition, and one for each of rounds. Please note that the ExpandedKey is always derived from the ci key; it should never be specified directly.

3.6.1 Design Criteria

The key expansion has been chosen according to the following criteria:

1. **Efficiency.**
   a) **Working memory.** It should be possible to execute the key sche using a small amount of working memory.
   b) **Performance.** It should have a high performance on a wide ra of processors.
2. **Symmetry elimination.** It should use round constants to elimi symmetries.
3. **Diffusion.** It should have an efficient diffusion of cipher key differe into the expanded key,
4. **Non-linearity.** It should exhibit enough non-linearity to prohibit full determination of differences in the expanded key from cipher differences only.

For a more thorough treatment of the criteria underlying the design of key schedule, we refer to Sect. 5.8.

3.6.2 Selection

In order to be efficient on 8-bit processors, a lightweight, byte-oriented pansion scheme has been adopted. The application of the non-linear ensures the non-linearity of the scheme, without adding much in the wa temporary storage requirements on an 8-bit processor.

During the key expansion the cipher key is expanded into an expar key array, consisting of 4 rows and $N_b(N_r + 1)$ columns. This array is denoted by $W[4][N_b(N_r+1)]$. The round key of the ith round, ExpandedKe is given by the columns $N_b \cdot i$ to $N_b \cdot (i+1) - 1$ of W:

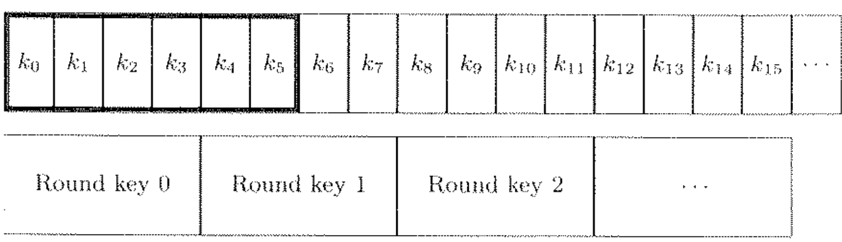
$$\texttt{ExpandedKey}[i] = \mathsf{W}[\cdot][\mathrm{N_b}\cdot i] \parallel \mathsf{W}[\cdot][\mathrm{N_b}\cdot i+1] \parallel \cdots \parallel \mathsf{W}[\cdot][\mathrm{N_b}\cdot(i+1)-1],$$
$$0 \le i \le \mathrm{N_r}. \tag{3.16}$$

The key expansion function depends on the value of $\mathrm{N_k}$: there is a version for $\mathrm{N_k}$ equal to or below 6, shown in List. 3.3, and a version for $\mathrm{N_k}$ above 6, shown in List. 3.4. In both versions of the key expansion, the first $\mathrm{N_k}$ columns of W are filled with the cipher key. The following columns are defined recursively in terms of previously defined columns. The recursion uses the bytes of the previous column, the bytes of the column $\mathrm{N_k}$ positions earlier, and round constants $\mathrm{RC}[j]$.

The recursion function depends on the position of the column. If i is not a multiple of $\mathrm{N_k}$, column i is the bitwise XOR of columns $i-\mathrm{N_k}$ and column $i-1$. Otherwise, column i is the bitwise XOR of column $i-\mathrm{N_k}$ and a non-linear function of column $i-1$. For cipher key length values $\mathrm{N_k} > 6$, this is also the case if $i \bmod \mathrm{N_k} = 4$. The non-linear function is realized by means of the application of $\mathrm{S_{RD}}$ to the four bytes of the column, an additional cyclic rotation of the bytes within the column and the addition of a round constant (for elimination of symmetry). The round constants are independent of $\mathrm{N_k}$, and defined by a recursion rule in $\mathrm{GF}(2^8)$:

$$\mathrm{RC}[1] = x^0 \text{ (i.e. 01)} \tag{3.17}$$
$$\mathrm{RC}[2] = x \text{ (i.e. 02)} \tag{3.18}$$
$$\mathrm{RC}[j] = x \cdot \mathrm{RC}[j-1] = x^{j-1}, \quad j > 2. \tag{3.19}$$

The key expansion process and the round key selection are illustrated in Fig. 3.10.



$$k_{6n} = k_{6n-6} \oplus f(k_{6n-1})$$
$$k_i = k_{i-6} \oplus k_{i-1}, \quad i \neq 6n$$

Fig. 3.10. Key expansion and round key selection for $\mathrm{N_b} = 4$ and $\mathrm{N_k} = 6$.

```
KeyExpansion(byte K[4][Nk], byte W[4][Nb(Nr + 1)])
                                                /* for Nk ≤ 6 */
{
for(j = 0; j < Nk; j++)
    for(i = 0; i < 4; i++) W[i][j] = K[i][j];
for(j = Nk; j < Nb(Nr + 1); j++)
    {
    if (j mod Nk == 0)
        {
        W[0][j] = W[0][j - Nk] ⊕ S[W[1][j - 1]] ⊕ RC[j/Nk];
        for(i = 1; i < 4; i++)
            W[i][j] = W[i][j - Nk] ⊕ S[W[i + 1 mod 4][j - 1]];
        }
    else
        {
        for(i = 0; i < 4; i++)
            W[i][j] = W[i][j - Nk] ⊕ W[i][j - 1];
        }
    }
}
```

List. 3.3. The key expansion for $\mathrm{N_k} \le 6$.

3.7 Decryption

The algorithm for decryption can be found in a straightforward way by using the inverses of the steps `InvSubBytes`, `InvShiftRows`, `InvMixColumns` and `AddRoundKey`, and reversing their order. We call the resulting algorithm the *straightforward decryption algorithm*. In this algorithm, not only so the steps themselves differ from those used in encryption, but also the sequence in which the steps occur is different. For implementation reasons, it is often convenient that the only non-linear step (`SubBytes`) is the first step of the round transformation (see Chap. 4). This aspect has been anticipated in the design. The structure of Rijndael is such that it is possible to define an *equivalent algorithm for decryption* in which the sequence of steps is equal to that for encryption, with the steps replaced by their inverses and a change in the key schedule. We illustrate this in Sect. 3.7.1–3.7.3 for a reduced version of Rijndael, that consists of only one round followed by the final round. Note that this identity in *structure* differs from the identity of *components* and *structure* (cf. Sect. 5.3.5) that is found in most ciphers with the Feistel structure, but also in IDEA [56].

3.7.1 Decryption for a Two-Round Rijndael Variant

The straightforward decryption algorithm with a two-round Rijndael variant consists of the inverse of `FinalRound`, followed by the inverse of `Round`,

```
KeyExpansion(byte K[4][Nk], byte W[4][Nb(Nr + 1)])
                                                  /* for Nk > 6 */
{
for(j = 0; j < Nk; j++)
     for(i = 0; i < 4; ++) W[i][j] = K[i][j];
for(j = Nk; j < Nb(Nr + 1); j++)
     {
     if (j mod Nk == 0)
          {
          W[0][j] = W[0][j - Nk] ⊕ S[W[1][j - 1]] ⊕ RC[j/Nk];
          for(i = 1; i < 4; i++)
               W[i][j] = W[i][j - Nk] ⊕ S[W[i + 1 mod 4][j - 1]];
          }
     else if (j mod Nk == 4)
          {
          for(i = 0; i < 4; i++)
               W[i][j] = W[i][j - Nk] ⊕ S[W[i][j - 1]];
          }
     else
          {
          for(i = 0; i < 4; i++)
               W[i][j] = W[i][j - Nk] ⊕ W[i][j - 1];
          }
     }
}
```

ist. 3.4. The key expansion for $N_k > 6$.

ıllowed by a key addition. The inverse transformation of `Round` is denoted ıvRound. The inverse of `FinalRound` is denoted `InvFinalRound`. Both trans- ırmations are described in List. 3.5. Listing 3.6 gives the straightforward ecryption algorithm for the two-round Rijndael variant.

.7.2 Algebraic Properties

ı order to derive the equivalent decryption algorithm, we use two properties the steps:

1. The order of `InvShiftRows` and `InvSubBytes` is indifferent.
2. The order of `AddRoundKey` and `InvMixColumns` can be inverted if the round key is adapted accordingly.

he first property can be explained as follows. `InvShiftRows` simply trans- ɔses the bytes and has no effect on the byte values. `InvSubBytes` operates ı individual bytes, independent of their position. Therefore, the two steps ɔmmute.


```
InvRound(State,ExpandedKey[i])
{
AddRoundKey(State,ExpandedKey[i]);
InvMixColumns(State);
InvShiftRows(State);
InvSubBytes(State);
}

InvFinalRound(State,ExpandedKey[Nr])
{
AddRoundKey(State,ExpandedKey[Nr]);
InvShiftRows(State);
InvSubBytes(State);
}
```

List. 3.5. Round transformations of the straightforward decryption algorithm.

```
AddRoundKey(State,ExpandedKey[2]);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State,ExpandedKey[1]);
InvMixColumns(State);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State,ExpandedKey[0]);
```

List. 3.6. Straightforward decryption algorithm for a two-round variant.