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>
<thead>
<tr>
<th>$N_k$</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_b$</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.6 Key Schedule

The key schedule consists of two components: the key expansion and
round key selection. The key expansion specifies how $\text{ExpandedKey}$ is
derived from the cipher key. The total number of bits in $\text{ExpandedKey}$ is
equal to 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 $\text{ExpandedKey}$ 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 r:
   of processors.

2. **Symmetry elimination.** It should use round constants to elim
   symmetries.

3. **Diffusion.** It should have an efficient diffusion of cipher key differ
   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 re
temporary storage requirements on an 8-bit processor.

During the key expansion the cipher key is expanded into an expan
array, consisting of 4 rows and $N_b(N_r+1)$ columns. This array is
denoted by $W[d][N_b(N_r+1)]$. The round key of the $i$th round, $\text{ExpandedKe}$
is given by the columns $N_b \cdot i$ to $N_b \cdot (i+1)$ of $W$: 
ExpandedKey[i] =
W[1][N_b \cdot i] \parallel W[2][N_b \cdot i + 1] \parallel \cdots \parallel W[i][N_b \cdot (i + 1) - 1],
0 \leq i \leq N_t. \quad (3.16)

The key expansion function depends on the value of \( N_k \): there is a version for \( N_k \) equal to or below 6, shown in List. 3.3, and a version for \( N_k \) above 6, shown in List. 3.4. In both versions of the key expansion, the first \( N_k \) 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 \( N_k \) positions earlier, and the round constants \( RC[j] \).

The recursion function depends on the position of the column. If \( i \) is not a multiple of \( N_k \), column \( i \) is the bitwise XOR of columns \( i - N_k \) and column \( i - 1 \). Otherwise, column \( i \) is the bitwise XOR of column \( i - N_k \) and a non-linear function of column \( i - 1 \). For cipher key length values \( N_k > 6 \), this is also the case if \( i \mod N_k = 4 \). The non-linear function is realized by means of the application of \( S_{R_{16}} \) 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 \( N_k \), and defined by a recursion rule in \( GF(2^8) \):

\[
\begin{align*}
RC[1] &= x^0 \quad (i.e. \ 01) \\
RC[2] &= x \quad (i.e. \ 02) \\
RC[j] &= x \cdot RC[j - 1] = x^{j-1}, \quad j > 2.
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>k_0</th>
<th>k_1</th>
<th>k_2</th>
<th>k_3</th>
<th>k_4</th>
<th>k_5</th>
<th>k_6</th>
<th>k_7</th>
<th>k_8</th>
<th>k_9</th>
<th>k_{10}</th>
<th>k_{11}</th>
<th>k_{12}</th>
<th>k_{13}</th>
<th>k_{14}</th>
<th>k_{15}</th>
<th>\ldots</th>
</tr>
</thead>
</table>

Round key 0 | Round key 1 | Round key 2 | \ldots

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

List. 3.3. The key expansion for \( N_k \leq 6 \).

### 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,
3. Specification of Rijndael

\[
\text{KeyExpansion}(\text{byte } K[|K|], \text{ byte } \text{byte}[|\text{byte}|], \text{ byte } \text{byte}[N_0(N_0 + 1)])
\]

\[
\text{for } (j = 0; j < N_k; j++) \\
\text{for } (i = 0; i < 4; i++) \; \text{byte}[i][j] = K[i][j]; \\
\text{for } (j = N_k; j < N_0(N_0 + 1); j++) \\
\{
\text{if } (j \mod N_k == 0) \\
\text{\quad \text{byte}[i][j] = \text{byte}[0][j - N_k] \oplus S[\text{byte}[i][j - 1]] \oplus RC[j/N_k];} \\
\text{\quad for } (i = 1; i < 4; i++) \\
\text{\quad \quad \text{byte}[i][j] = \text{byte}[i][j - N_k] \oplus S[\text{byte}[i+1 \mod 4][j - 1]];}
\}
\text{else if } (j \mod N_k == 4) \\
\text{\quad for } (i = 0; i < 4; i++) \\
\text{\quad \quad \text{byte}[i][j] = \text{byte}[i][j - N_k] \oplus S[\text{byte}[i][j - 1]];}
\}
\text{else} \\
\text{\quad for } (i = 0; i < 4; i++) \\
\text{\quad \quad \text{byte}[i][j] = \text{byte}[i][j - N_k] \oplus \text{byte}[i][j - 1];}
\}
\]

\textbf{List. 3.4.} The key expansion for \(N_k > 6\).

allowed by a key addition. The inverse transformation of \texttt{Round} is denoted \texttt{invRound}. The inverse of \texttt{FinalRound} is denoted \texttt{invFinalRound}. Both transformations are described in List. 3.5. Listing 3.6 gives the straightforward decryption algorithm for the two-round Rijndael variant.

\section{7.2 Algebraic Properties}

:\textit{In order to derive the equivalent decryption algorithm, we use two properties of the steps:}

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

The first property can be explained as follows. \texttt{InvShiftRows} simply transposes the bytes and has no effect on the byte values. \texttt{InvSubBytes} operates on individual bytes, independent of their position. Therefore, the two steps commute.

\textit{List. 3.5. Round transformations of the straightforward decryption algorithm.}

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

InvFinalRound(State, ExpandedKey[N_0])
  AddRoundKey(State, ExpandedKey[N_0]);
  InvShiftRows(State);
  InvSubBytes(State);
}

\textbf{List. 3.6. Straightforward decryption algorithm for a two-round variant.}

\begin{verbatim}
AddRoundKey(State, ExpandedKey[2]);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State, ExpandedKey[1]);
InvMixColumns(State);
InvShiftRows(State);
InvSubBytes(State);
AddRoundKey(State, ExpandedKey[0]);
\end{verbatim}