Note on naming

1. Introduction

After the selection of Rijndael as the AES, it was decided to change the names of some of its component functions in order to improve the readability of the standard. However, we see that many recent publications on Rijndael and the AES still use the old names, mainly because the original submission documents using the old names, are still available on the Internet. In this note we repeat quickly the new names for the component functions. Additionally, we remind the reader on the difference between AES and Rijndael and present an overview of the most important references for Rijndael and the AES.

2. References

[1] Joan Daemen and Vincent Rijmen, AES submission document on Rijndael, June 1998.

[2] Joan Daemen and Vincent Rijmen, AES submission document on Rijndael, Version 2, September 1999. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf

[3] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[4] Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.)

3. Naming

The names of the component functions of Rijndael have been modified between the publication of [2] and that of [3]. Table 1 lists the two versions of names. **We recommend using the new names**.

Old naming	New naming
`ByteSub`	`SubBytes`
`ShiftRow`	`ShiftRows`
`MixColumn`	`MixColumns`
`AddRoundKey`	`AddRoundKey`

Table 1: Old and new names of the Rijndael component functions

4. Range of key and block lengths in Rijndael and AES

Rijndael and AES differ only in the range of supported values for the block length and cipher key length.

For Rijndael, the block length and the key length can be independently specified to any multiple of 32 bits, with a minimum of 128 bits, and a maximum of 256 bits. The support for block and key lengths 160 and 224 bits was introduced in reference [2].

AES fixes the block length to 128 bits, and supports key lengths of 128, 192 or 256 bits only.

5. Referencing

Reference [3] is the US Federal Information Processing Standard defining AES and hence the **definitive reference on AES**.

Reference [4] is the definitive reference on Rijndael. It is a book we have written after the selection of Rijndael as AES and was published in February 2002. It describes all aspects of Rijndael and is only available on paper.

Reference [1] is the original Rijndael documentation submitted to AES and dates from June 11, 1998. Reference [2] is an improved version dating from September 3, 1999 that supersedes reference [1]. Both were made available electronically in PDF formats on several sites. Both references should be used only when referring to the actual historical documents. Technical or scientific references should be restricted to [3] and [4].

We propose to use the following BibTex entries:

```
@Book{Daemen:2002:DRA,
  author =       "Joan Daemen and Vincent Rijmen",
  title =        "The design of {Rijndael}: {AES} --- the {Advanced
                 Encryption Standard}",
  publisher =    "Spring{\-}er-Ver{\-}lag",
  pages =        "238",
  year =         "2002",
  ISBN =         "3-540-42580-2"
}

@misc{AES-FIPS,
   title = "Specification for the Advanced Encryption Standard (AES)",
   howpublished = "Federal Information Processing Standards Publication 197",
   year = "2001",
   url = " http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf"
}
```

AES Proposal: Rijndael

Joan Daemen, Vincent Rijmen

Joan Daemen
Proton World Int.l
Zweefvliegtuigstraat 10
B-1130 Brussel, Belgium
daemen.j@protonworld.com

Vincent Rijmen
Katholieke Universiteit Leuven, ESAT-COSIC
K. Mercierlaan 94
B-3001 Heverlee, Belgium
vincent.rijmen@esat.kuleuven.ac.be

Table of Contents

Table of Figures

List of Tables

1. Introduction

In this document we describe the cipher Rijndael. First we present the mathematical basis necessary for understanding the specifications followed by the design rationale and the description itself. Subsequently, the implementation aspects of the cipher and its inverse are treated. This is followed by the motivations of all design choices and the treatment of the resistance against known types of attacks. We give our security claims and goals, the advantages and limitations of the cipher, ways how it can be extended and how it can be used for functionality other than block encryption/decryption. We conclude with the acknowledgements, the references and the list of annexes.

Patent Statement: Rijndael or any of its implementations is not and will not be subject to patents.

1.1 Document history

This is the second version of the Rijndael documentation. The main difference with the first version is the correction of a number of errors and inconsistencies, the addition of a motivation for the number of rounds, the addition of some figures in the section on differential and linear cryptanalysis, the inclusion of Brian Gladman's performance figures and the specification of Rijndael extensions supporting block and key lengths of 160 and 224 bits.

2. Mathematical preliminaries

Several operations in Rijndael are defined at byte level, with bytes representing elements in the finite field GF(2^8). Other operations are defined in terms of 4-byte *words*. In this section we introduce the basic mathematical concepts needed in the following of the document.

2.1 The field GF(2^8)

The elements of a finite field [LiNi86] can be represented in several different ways. For any prime power there is a single finite field, hence all representations of GF(2^8) are isomorphic. Despite this equivalence, the representation has an impact on the implementation complexity. We have chosen for the classical polynomial representation.

A byte b, consisting of bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$, is considered as a polynomial with coefficient in {0,1}:

$$b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

Example: the byte with hexadecimal value '57' (binary `01010111`) corresponds with polynomial

$$x^6 + x^4 + x^2 + x + 1 .$$

2.1.1 Addition

In the polynomial representation, the sum of two elements is the polynomial with coefficients that are given by the sum modulo 2 (i.e., 1 + 1 = 0) of the coefficients of the two terms.

Example: ‘`57`’ + ‘`83`’ = ‘`D4`’, or with the polynomial notation:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 .$$

In binary notation we have: “`01010111`” + “`10000011`” = “`11010100`”. Clearly, the addition corresponds with the simple bitwise EXOR (denoted by ⊕) at the byte level.

All necessary conditions are fulfilled to have an Abelian group: internal, associative, neutral element (‘`00`’), inverse element (every element is its own additive inverse) and commutative. As every element is its own additive inverse, subtraction and addition are the same.

2.1.2 Multiplication

In the polynomial representation, multiplication in GF(2^8) corresponds with multiplication of polynomials modulo an irreducible binary polynomial of degree 8. A polynomial is irreducible if it has no divisors other than 1 and itself. For Rijndael, this polynomial is called $m(x)$ and given by

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

or ‘`11B`’ in hexadecimal representation.

Example: ‘`57`’ • ‘`83`’ = ‘`C1`’, or:

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

$$\begin{aligned}x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \text{ modulo } x^8 + x^4 + x^3 + x + 1 \\ = \quad x^7 + x^6 + 1\end{aligned}$$

Clearly, the result will be a binary polynomial of degree below 8. Unlike for addition, there is no simple operation at byte level.

The multiplication defined above is associative and there is a neutral element (‘`01`’). For any binary polynomial $b(x)$ of degree below 8, the extended algorithm of Euclid can be used to compute polynomials $a(x)$, $c(x)$ such that

$$b(x)a(x) + m(x)c(x) = 1 .$$

Hence, $a(x) \bullet b(x) \bmod m(x) = 1$ or

$$b^{-1}(x) = a(x) \bmod m(x)$$

Moreover, it holds that $a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$.

It follows that the set of 256 possible byte values, with the EXOR as addition and the multiplication defined as above has the structure of the finite field GF(2^8).

2.1.3 Multiplication by *x*

If we multiply $b(x)$ by the polynomial x, we have:

$$b_7\, x^8 + b_6\, x^7 + b_5\, x^6 + b_4\, x^5 + b_3\, x^4 + b_2\, x^3 + b_1\, x^2 + b_0\, x$$

$x \bullet b(x)$ is obtained by reducing the above result modulo $m(x)$. If $b_7 = 0$, this reduction is the identity operation, If $b_7 = 1$, $m(x)$ must be subtracted (i.e., EXORed). It follows that multiplication by x (hexadecimal '02') can be implemented at byte level as a left shift and a subsequent conditional bitwise EXOR with '1B'. This operation is denoted by `b = xtime(a)`. In dedicated hardware, `xtime` takes only 4 EXORs. Multiplication by higher powers of x can be implemented by repeated application of `xtime`. By adding intermediate results, multiplication by any constant can be implemented.

Example: '57' • '13' = 'FE'

'57' • '02' = `xtime(57)` = 'AE'

'57' • '04' = `xtime(AE)` = '47'

'57' • '08' = `xtime(47)` = '8E'

'57' • '10' = `xtime(8E)` = '07'

'57' • '13' = '57' • ('01' ⊕ '02' ⊕ '10') = '57' ⊕ 'AE' ⊕ '07' = 'FE'

2.2 Polynomials with coefficients in GF(2^8)

Polynomials can be defined with coefficients in GF(2^8). In this way, a 4-byte vector corresponds with a polynomial of degree below 4.

Polynomials can be added by simply adding the corresponding coefficients. As the addition in GF(2^8) is the bitwise EXOR, the addition of two vectors is a simple bitwise EXOR.

Multiplication is more complicated. Assume we have two polynomials over GF(2^8):

$$a(x) = a_3\, x^3 + a_2\, x^2 + a_1\, x + a_0 \text{ and } b(x) = b_3\, x^3 + b_2\, x^2 + b_1\, x + b_0.$$

Their product $c(x) = a(x)b(x)$ is given by

$$c(x) = c_6\, x^6 + c_5\, x^5 + c_4\, x^4 + c_3\, x^3 + c_2\, x^2 + c_1\, x + c_0 \qquad \text{with}$$

$$c_0 = a_0 \bullet b_0 \qquad c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \qquad c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \qquad c_6 = a_3 \bullet b_3$$

$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

Clearly, $c(x)$ can no longer be represented by a 4-byte vector. By reducing $c(x)$ modulo a polynomial of degree 4, the result can be reduced to a polynomial of degree below 4. In Rijndael, this is done with the polynomial $M(x) = x^4 + 1$. As

$$x^j \bmod x^4 + 1 = x^{j \bmod 4},$$

the modular product of $a(x)$ and $b(x)$, denoted by $d(x) = a(x) \otimes b(x)$ is given by

$$d(x) = d_3\, x^3 + d_2\, x^2 + d_1\, x + d_0 \qquad \text{with}$$

$$d_0 = a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$d_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$d_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3$$

$$d_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

The operation consisting of multiplication by a fixed polynomial $a(x)$ can be written as matrix multiplication where the matrix is a circulant matrix. We have

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Note: $x^4 + 1$ is not an irreducible polynomial over GF(2^8), hence multiplication by a fixed polynomial is not necessarily invertible. In the Rijndael cipher we have chosen a fixed polynomial that does have an inverse.

2.2.1 Multiplication by *x*

If we multiply $b(x)$ by the polynomial x, we have:

$$b_3\, x^4 + b_2\, x^3 + b_1\, x^2 + b_0\, x$$

$x \otimes b(x)$ is obtained by reducing the above result modulo $1 + x^4$. This gives

$$b_2\, x^3 + b_1\, x^2 + b_0\, x + b_3$$

The multiplication by x is equivalent to multiplication by a matrix as above with all a_i ='00' except a_1 ='01'. Let $c(x) = x \otimes b(x)$. We have:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Hence, multiplication by *x*, or powers of *x*, corresponds to a cyclic shift of the bytes inside the vector.

3. Design rationale

The three criteria taken into account in the design of Rijndael are the following:

- Resistance against all known attacks;
- Speed and code compactness on a wide range of platforms;
- Design simplicity.

In most ciphers, the round transformation has the Feistel Structure. In this structure typically part of the bits of the intermediate State are simply transposed unchanged to another position. The round transformation of Rijndael does *not* have the Feistel structure. Instead, the round transformation is composed of three distinct invertible uniform transformations, called *layers*. By "uniform", we mean that every bit of the State is treated in a similar way.

The specific choices for the different layers are for a large part based on the application of the Wide Trail Strategy [Da95] (see Annex), a design method to provide resistance against linear and differential cryptanalysis (see Section 8.2). In the Wide Trail Strategy, every layer has its own function:

The linear mixing layer:	guarantees high diffusion over multiple rounds.
The non-linear layer:	parallel application of S-boxes that have optimum worst-case nonlinearity properties.
The key addition layer:	A simple EXOR of the Round Key to the intermediate State.

Before the first round, a key addition layer is applied. The motivation for this initial key addition is the following. Any layer after the last key addition in the cipher (or before the first in the context of known-plaintext attacks) can be simply peeled off without knowledge of the key and therefore does not contribute to the security of the cipher. (e.g., the initial and final permutation in the DES). Initial or terminal key addition is applied in several designs, e.g., IDEA, SAFER and Blowfish.

In order to make the cipher and its inverse more similar in structure, the linear mixing layer of the last round is different from the mixing layer in the other rounds. It can be shown that this does not improve or reduce the security of the cipher in any way. This is similar to the absence of the swap operation in the last round of the DES.

4. Specification

Rijndael is an iterated block cipher with a variable block length and a variable key length. The block length and the key length can be independently specified to 128, 192 or 256 bits.

Note: this section is intended to explain the cipher structure and not as an implementation guideline. For implementation aspects, we refer to Section 5.

4.1 The State, the Cipher Key and the number of rounds

The different transformations operate on the intermediate result, called the *State*:

Definition: the intermediate cipher result is called the *State.*

The State can be pictured as a rectangular array of bytes. This array has four rows, the number of columns is denoted by **Nb** and is equal to the block length divided by 32.

The Cipher Key is similarly pictured as a rectangular array with four rows. The number of columns of the Cipher Key is denoted by **`Nk`** and is equal to the key length divided by 32.

These representations are illustrated in Figure 1.

In some instances, these blocks are also considered as one-dimensional arrays of 4-byte vectors, where each vector consists of the corresponding column in the rectangular array representation. These arrays hence have lengths of 4, 6 or 8 respectively and indices in the ranges 0..3, 0..5 or 0..7. 4-byte vectors will sometimes be referred to as words.

Where it is necessary to specify the four individual bytes within a 4-byte vector or word the notation (a, b, c, d) will be used where a, b, c and d are the bytes at positions 0, 1, 2 and 3 respectively within the column, vector or word being considered.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Figure 1: Example of State (with Nb = 6) and Cipher Key (with Nk = 4) layout.

The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4***`Nb`**–1. These blocks hence have lengths of 16, 24 or 32 bytes and array indices in the ranges 0..15, 0..23 or 0..31. The Cipher Key is considered to be a one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4***`Nk`**–1. These blocks hence have lengths of 16, 24 or 32 bytes and array indices in the ranges 0..15, 0..23 or 0..31.

The cipher input bytes (the "plaintext" if the mode of use is ECB encryption) are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{4,1}$ ... , and the bytes of the Cipher Key are mapped onto the array in the order $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$, $k_{4,1}$ ... At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order.

Hence if the one-dimensional index of a byte within a block is n and the two dimensional index is (i, j), we have:

$$i = n \bmod 4; \qquad j = \lfloor n / 4 \rfloor; \qquad n = i + 4 * j$$

Moreover, the index i is also the byte number within a 4-byte vector or word and j is the index for the vector or word within the enclosing block.

The number of rounds is denoted by **`Nr`** and depends on the values **`Nb`** and **`Nk`**. It is given in Table 1.

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

Table 1: Number of rounds (Nr) as a function of the block and key length.

4.2 The round transformation

The round transformation is composed of four different transformations. In pseudo C notation we have:

```
Round(State,RoundKey)
{
ByteSub(State);
ShiftRow(State);
MixColumn(State);
AddRoundKey(State,RoundKey);
}
```

The final round of the cipher is slightly different. It is defined by:

```
FinalRound(State,RoundKey)
{
ByteSub(State) ;
ShiftRow(State) ;
AddRoundKey(State,RoundKey);
}
```

In this notation, the "functions" (`Round`, `ByteSub`, `ShiftRow`, …) operate on arrays to which pointers (`State`, `RoundKey`) are provided.

It can be seen that the final round is equal to the round with the MixColumn step removed.

The component transformations are specified in the following subsections.

4.2.1 The ByteSub transformation

The ByteSub Transformation is a non-linear byte substitution, operating on each of the State bytes independently. The substitution table (or S-box) is invertible and is constructed by the composition of two transformations:

1. First, taking the multiplicative inverse in GF(2^8), with the representation defined in Section 2.1. '`00`' is mapped onto itself.
2. Then, applying an affine (over GF(2)) transformation defined by:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The application of the described S-box to all bytes of the State is denoted by:

```
ByteSub(State) .
```

Figure 2 illustrates the effect of the ByteSub transformation on the State.

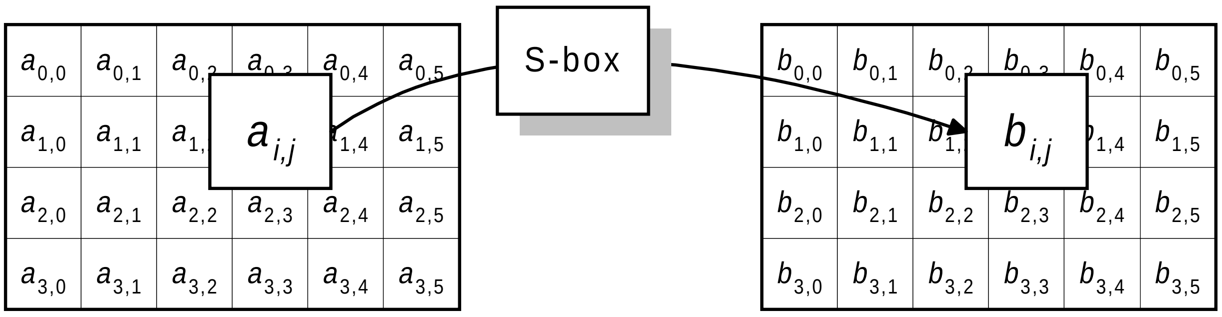
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$a_{i,j}$
S-box
$b_{i,j}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$ $b_{0,4}$ $b_{0,5}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$ $b_{1,5}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$ $b_{2,5}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$ $b_{3,5}$

Figure 2: ByteSub acts on the individual bytes of the State.

The inverse of ByteSub is the byte substitution where the inverse table is applied. This is obtained by the inverse of the affine mapping followed by taking the multiplicative inverse in GF(2^8).

4.2.2 The ShiftRow transformation

In ShiftRow, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted, Row 1 is shifted over `C1` bytes, row 2 over `C2` bytes and row 3 over `C3` bytes.

The shift offsets `C1`, `C2` and `C3` depend on the block length **`Nb`**. The different values are specified in Table 2.

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Table 2: Shift offsets for different block lengths.

The operation of shifting the rows of the State over the specified offsets is denoted by:

`ShiftRow(State)` .

Figure 3 illustrates the effect of the ShiftRow transformation on the State.

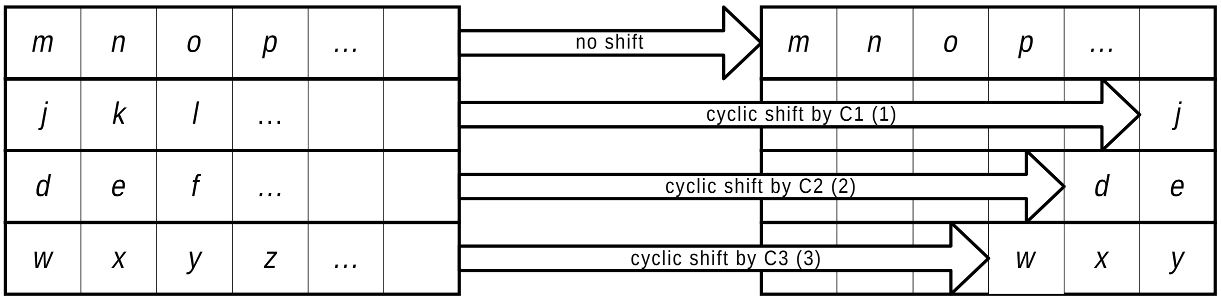

Figure 3: ShiftRow operates on the rows of the State.

The inverse of ShiftRow is a cyclic shift of the 3 bottom rows over **Nb**-C1, **Nb**-C2 and **Nb**-C3 bytes respectively so that the byte at position *j* in row *i* moves to position (*j* + **Nb**-Ci) mod **Nb**.

4.2.3 The MixColumn transformation

In MixColumn, the columns of the State are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by

$$c(x) = \text{'03'}\, x^3 + \text{'01'}\, x^2 + \text{'01'}\, x + \text{'02'} .$$

This polynomial is coprime to $x^4 + 1$ and therefore invertible. As described in Section 2.2, this can be written as a matrix multiplication. Let b(x) = c(x) ⊗ a(x),

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The application of this operation on all columns of the State is denoted by

`MixColumn(State)`.

Figure 4 illustrates the effect of the MixColumn transformation on the State.

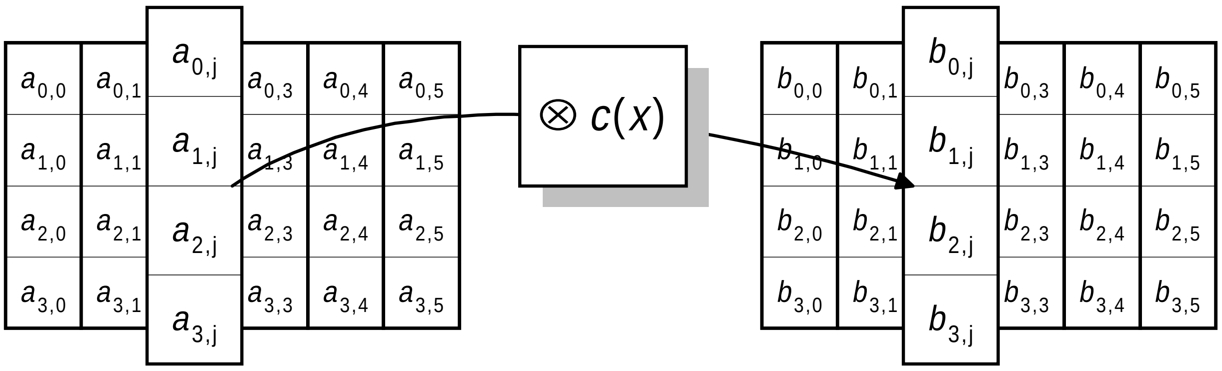
$a_{0,0}$ $a_{0,1}$ $a_{0,j}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$
$a_{1,0}$ $a_{1,1}$ $a_{1,j}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$
$a_{2,0}$ $a_{2,1}$ $a_{2,j}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$
$a_{3,0}$ $a_{3,1}$ $a_{3,j}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,j}$ $b_{0,3}$ $b_{0,4}$ $b_{0,5}$
$b_{1,0}$ $b_{1,1}$ $b_{1,j}$ $b_{1,3}$ $b_{1,4}$ $b_{1,5}$
$b_{2,0}$ $b_{2,1}$ $b_{2,j}$ $b_{2,3}$ $b_{2,4}$ $b_{2,5}$
$b_{3,0}$ $b_{3,1}$ $b_{3,j}$ $b_{3,3}$ $b_{3,4}$ $b_{3,5}$

Figure 4: MixColumn operates on the columns of the State.

The inverse of MixColumn is similar to MixColumn. Every column is transformed by multiplying it with a specific multiplication polynomial $d(x)$, defined by

$$(\text{‘03’}\,x^3 + \text{‘01’}\,x^2 + \text{‘01’}\,x + \text{‘02’}) \otimes d(x) = \text{‘01’}\,.$$

It is given by:

$$d(x) = \text{‘0B’}\,x^3 + \text{‘0D’}\,x^2 + \text{‘09’}\,x + \text{‘0E’}\,.$$

4.2.4 The Round Key addition

In this operation, a Round Key is applied to the State by a simple bitwise EXOR. The Round Key is derived from the Cipher Key by means of the key schedule. The Round Key length is equal to the block length **`Nb`**.

The transformation that consists of EXORing a Round Key to the State is denoted by:

`AddRoundKey(State,RoundKey).`

This transformation is illustrated in Figure 5.

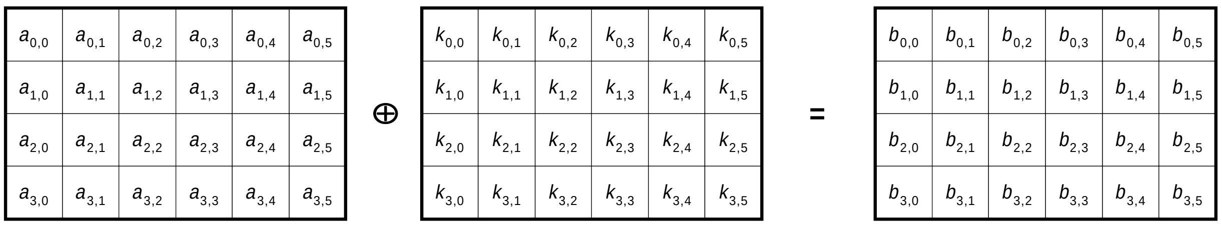
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$
$\oplus$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$ $k_{0,4}$ $k_{0,5}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$ $k_{1,4}$ $k_{1,5}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$ $k_{2,4}$ $k_{2,5}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$ $k_{3,4}$ $k_{3,5}$
=
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$ $b_{0,4}$ $b_{0,5}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$ $b_{1,5}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$ $b_{2,5}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$ $b_{3,5}$

Figure 5: In the key addition the Round Key is bitwise EXORed to the State.

AddRoundKey is its own inverse.

4.3 Key schedule

The Round Keys are derived from the Cipher Key by means of the key schedule. This consists of two components: the Key Expansion and the Round Key Selection. The basic principle is the following:

- The total number of Round Key bits is equal to the block length multiplied by the number of rounds plus 1. (e.g., for a block length of 128 bits and 10 rounds, 1408 Round Key bits are needed).
- The Cipher Key is expanded into an *Expanded Key*.
- Round Keys are taken from this Expanded Key in the following way: the first Round Key consists of the first **Nb** words, the second one of the following **Nb** words, and so on.

4.3.1 Key expansion

The Expanded Key is a linear array of 4-byte words and is denoted by `W[Nb*(Nr+1)]`. The first **Nk** words contain the Cipher Key. All other words are defined recursively in terms of words with smaller indices. The key expansion function depends on the value of **Nk**: there is a version for **Nk** equal to or below 6, and a version for **Nk** above 6.

For **Nk** $\leq$ 6, we have:

```
KeyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
{
     for(i = 0; i < Nk; i++)
          W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);

     for(i = Nk; i < Nb * (Nr + 1); i++)
     {
          temp = W[i - 1];
          if (i % Nk == 0)
               temp = SubByte(RotByte(temp)) ^ Rcon[i / Nk];
          W[i] = W[i - Nk] ^ temp;
     }
}
```

In this description, `SubByte(W)` is a function that returns a 4-byte word in which each byte is the result of applying the Rijndael S-box to the byte at the corresponding position in the input word. The function `RotByte(W)` returns a word in which the bytes are a cyclic permutation of those in its input such that the input word (a,b,c,d) produces the output word (b,c,d,a).

It can be seen that the first **Nk** words are filled with the Cipher Key. Every following word `W[i]` is equal to the EXOR of the previous word `W[i-1]` and the word **Nk** positions earlier `W[i-Nk]`. For words in positions that are a multiple of **Nk**, a transformation is applied to `W[i-1]` prior to the EXOR and a round constant is EXORed. This transformation consists of a cyclic shift of the bytes in a word (`RotByte`), followed by the application of a table lookup to all four bytes of the word (`SubByte`).