

The SubBytes Operation

Each byte of State is substituted (independently). Can be implemented via table lookup (memory permitting), but is described algebraically. Let ϕ be the function

$$\phi : (a_7 a_6 \dots a_0) \mapsto \sum_{i=0}^7 a_i x^i, a_i \in \mathbb{F}_2 = \{0, 1\} .$$

Then:

$$\text{SubBytes}(a) = \phi^{-1} \left[(x^4 + x^3 + x^2 + x + 1) \phi(a)^{-1} + (x^6 + x^5 + x + 1) \bmod (x^8 + 1) \right] .$$

This operation can be performed using the following steps:

1. $z = \phi(a)$ (field representation of the byte a)
2. $z = z^{-1}$ (take the inverse in \mathbb{F}_{2^8})
3. $b = \phi^{-1}(z)$ (map the field element z to the byte b)

4. Output the byte b' using the following affine transformation:

$$\begin{array}{c}
 \left| \begin{array}{c} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{array} \right| = \begin{array}{c} \left| \begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array} \right| \begin{array}{c} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{array} \oplus \left| \begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{array} \right|
 \end{array}$$

Note that $b' = (b'_7 b'_6 \dots b'_0)$ where

$$b'_i = b_i \oplus b_{i+4 \bmod 8} \oplus b_{i+5 \bmod 8} \oplus b_{i+6 \bmod 8}$$

$$\oplus b_{i+7 \bmod 8} \oplus c_i$$

and $c = (11000110)$.

The inverse of SubBytes (called InvSubBytes, p. 22) is defined by

$$\text{InvSubBytes}(a) = \phi^{-1} \left[((x^6 + x^3 + x)\phi(a) + (x^2 + 1) \bmod (x^8 + 1))^{-1} \right] .$$

The ShiftRows Operation

Shifts the rows of State by 0, 1, 2, or 3 cells to the left:

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

←

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

The inverse operation `InvShiftRows` applies right shifts instead of left shifts.

The MixColumns Operation

Each column of State is a 4-byte vector which can be interpreted as a four-term polynomial with coefficients in $GF(2^8)$ as described above. For example:

$$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \mapsto s_{3,0}x^3 + s_{2,0}x^2 + s_{1,0}x + s_{0,0} = col_0(x) .$$

Let $a(x) = 3x^3 + x^2 + x + 2$ be fixed. Then MixColumns multiplies $col_i(x)$ by $a(x)$ as described above (multiplication of two 4-byte vectors), resulting in a new 4-byte column.

MixColumns can also be described as a linear transformation applied to each column of State, i.e., multiplying each 4-element column vector by the 4×4 matrix.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 1 \end{pmatrix}$$

The InvMixColumns Operation

The inverse (called InvMixColumns) multiplies each column of State by the inverse of $a(x) \pmod{x^4 + 1}$ which is $a^{-1}(x) = Bx^3 + Dx^2 + 9x + E$ in hex notation. It can be described as multiplication by the matrix (in hex):

$$\begin{pmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{pmatrix}$$

The AddRoundKey Operation

In AddRoundKey, each column of State is X-ORed with one word of the round key:

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

←

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	\oplus	$w_{0,i+0}$	$w_{0,i+1}$	$w_{0,i+2}$	$w_{0,i+3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$		$w_{1,i+0}$	$w_{1,i+1}$	$w_{1,i+2}$	$w_{1,i+3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		$w_{2,i+0}$	$w_{2,i+1}$	$w_{2,i+2}$	$w_{2,i+3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		$w_{3,i+0}$	$w_{3,i+1}$	$w_{3,i+2}$	$w_{3,i+3}$

The KeyExpansion Operation

Consider 128-bit Rijndael.

- There are 10 rounds.
- Plus one prelim. appl. of AddRoundKey
- We need 11 round keys, each consisting of four 4-byte words, from the 128-bit key.
- KeyExpansion produces an expanded key consisting of the required 44 words. In the following, the key $K = (k_0, k_1, k_2, k_3)$, where the k_i are 4-byte words, and the expanded key is denoted by the word-vector $(w_0, w_1, w_2, \dots, w_{44})$.

1. for $i \in \{0, 1, 2, 3\}$, $w_i = k_i$

2. for $i \in \{4, \dots, 44\}$:

$$w_i = w_{i-4} \oplus$$

$$\begin{cases} \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \text{Rcon}_{i/4} & \text{if } 4|i \\ w_{i-1} & \text{otherwise} \end{cases}$$

The components of KeyExpansion are:

- RotWord is a one-byte circular left shift on a word.
- SubWord performs a byte substitution (using the S-box SubBytes on each byte of its input word).
- Rcon is a table of round constants ($Rcon_j$ is used in round j). Each is a word with the three rightmost bytes equal to 0.