AES

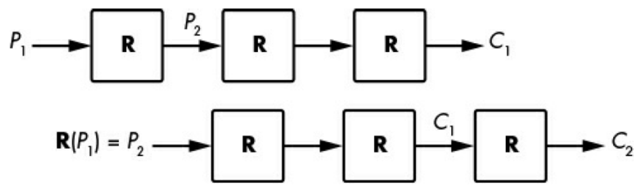
25 March 2021

Introduction

AES (aka Rijndael from its invetor's name) is a block ciphers with blocks of 128 (27) bits. It is important that blocks are not too large in order to minimize both the length of ciphertext and the memory footprint. As for the memory footprint, in order to process a 128-bit block, you need at least 128 bits of memory. This is small enough to fit in the registers of most CPUs or to be implemented using dedicated hardware circuits. Blocks of 64, 128, or even 512 bits are short enough to allow for efficient implementations in most cases. When ciphertexts' length or memory footprint is critical, you may have to use 64-bit blocks, because these will produce shorter ciphertexts and consume less memory. If it is possible , 128-bit or larger blocks are better. If you need to encrypt a 16-bit message when blocks are 128 bits, you'll first need to convert the message into a 128-bit block by means of padding. Blocks shouldn't be too small otherwise, they may be susceptible to codebook attacks. The codebook attack works like this with 16-bit blocks: 1. Get the 65536 (216) ciphertexts corresponding to each 16-bit plaintext block. 2. Build a lookup table—the codebook—mapping each ciphertext block to its corresponding plaintext block. 3. To decrypt an unknown ciphertext block, look up its corresponding plaintext block in the table.

Rounds

Computing a block cipher boils down to computing a sequence of rounds. In a block cipher, a round is a basic transformation that is simple to specify and to implement, and which is iterated several times. Each round should also have an inverse in order to make it possible for a recipient to compute back to plaintext. The round functions—R1, R2, and so on—are usually identical algorithms, but they are parameterized by a value called the round key. In a block cipher, no round should be identical to another round in order to avoid a slide attack. When rounds are identical, the relation between the two plaintexts, P2 = R(P1), implies the relation C2 = R(C1). Knowing the input and output of a single round often helps recover the key.



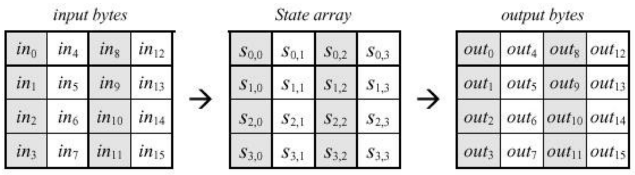
1 Confusion and DIffusion

Confusion means that the input (plaintext and encryption key) undergoes complex transformations so that it is hard to recover the key having ciphertext (at a high level, confusion is about depth whereas diffusion is about breadth). Diffusion means that these transformations depend equally on all bits of the input (ability of distribute the statistical correlation among the entire alphabet used by the algorithm, making harder a statistical attack which means that some letters are more used than other and the same for the combination of letter) . In the design of a block cipher, confusion and diffusion take the form of substitution and permutation operations, which are combined within substitution–permutation networks (SPNs). Substitution often appears in the form of S-boxes, or substitution boxes, which are small lookup tables that transform chunks of 4 or 8 bits (S-boxes should be as nonlinear as possible). The permutation can be as simple as changing the order of the bits, which is easy to implement but does not mix up the bits very much. Some ciphers use basic linear algebra and matrix multiplications to mix up the bits: they perform a series of multiplication operations with fixed values (the matrix's coefficients) and then add the results.

2 AES

AES processes blocks of 128 bits using a secret key of 128, 192, or 256 bits (the number of transformation rounds that convert the plaintext into the ciphertext depends on the length of the key). The 128-bit key being the most common because it makes encryption slightly faster and because the difference between 128- and 256-bit security is meaningless for most applications. AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

The totality of the operations are made on a bidimensional array, called state, that consist of 4 rows of byte, each of Nb byte, where Nb is the block size divided for 32. In the State array, denominated with the symbol "S", each byte is indicated by 2 indexes: the row index varies in the range 0<=r<4, while the column index avrues in the range 0<=c<Nb (with block size equal 128 bits, Nb takes value of 4).



2.1 Galois Field

All the bytes of the algorithm are interpreted as elements of Galois finite field. A byte b is composed by the bit b7 b6 b5 b4 b3 b2 b1 b0 and is considered as a polynomial with coefficient in 0,1:

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

e.g the byte with binary value 01010111 corresponds to $x6 + x4 + x2 + x + 1$. The elements of a finite field can be added and multiplied.

Addition example:

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

In binary notation: "01010111"+"10000011" = "11010100". Modulo 2 addition corresponds to a XOR.

In the polynomial representation, multiplication in GF(2^8) (with GF(2^8) we are relating to the finite field composed by 256 elements), corresponds to the multiplication of the polynomials modulo of an irreducible binary polynomial of grade 8. A polynomial is irreducible if it has no other dividers different from 1 and itself. For Rijndael (AES) it is :

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

e.g:

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) =$$
$$x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1 =$$
$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 modulo(x^8 + x^4 + x^3 + x + 1) =$$
$$x^7 + x^6 + 1$$

The set of the 256 possible values obtainable with a byte, joined with the XOR operation used as addition and with multiplication give birth to the finite field GF(2^8).

2.2 Polynomials with coefficient in GF(2^8)

Polynomials of 4 terms can be represented as coefficients that belong to a finite field:

$$a(x) = a_3\,x^3 + a_2\,x^2 + a_1\,x + a_0 \rightarrow word : [a_0, a_1, a_2, a_3]$$

This polynomials work in a different way because coefficients themselves are elements of a finite field and differently from before this time they are byte while in the previous definition they were bits. In this way an array of 4 bytes is related to a polynomial of grade less than 4 with coefficient in the GF(2^8).

2.2.1 Addition

$$a(x) + b(x) = (a_3 \oplus b_3) * x^3 + (a_2 \oplus b_2) * x^2 + (a_1 \oplus b_1) * x + (a_0 \oplus b_0)$$

2.2.2 Multiplication

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

Their product $c(x) = a(x)b(x)$ is obtained as:

$$c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

Clearly, c(x) will be represented as a 4 byte array by means of a modulo 4 polynomial. In Rijndael is used $m(x) = x^4 + 1$, so that the modular product of a(x) and b(x) is $d(x) = a(x)b(x)$:

$$d(x) = d3x3 + d2x2 + d1x + d0\ with$$

$$d0 = a_0 * b_0 \oplus a_3 * b_1 \oplus a_2 * b_2 \oplus a_1 * b_3$$
$$d1 = a_1 * b_0 \oplus a_0 * b_1 \oplus a_3 * b_2 \oplus a_2 * b_3$$
$$d2 = a_2 * b_0 \oplus a_1 * b_1 \oplus a_0 * b_2 \oplus a_3 * b_3$$
$$d3 = a_3 * b_0 \oplus a_2 * b_1 \oplus a_1 * b_2 \oplus a_0 * b_3$$

Can also be written as matricial multiplication where the matrix is a circular matrix:

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

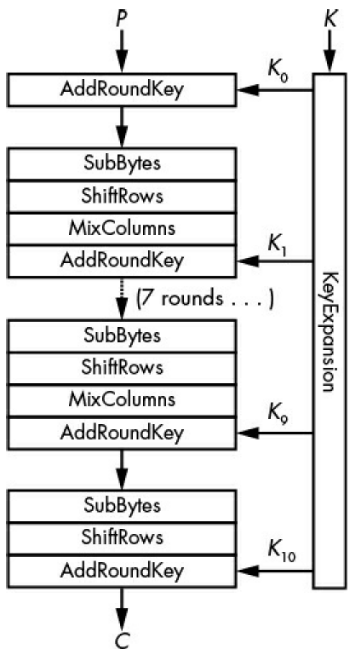
Since $x^4 + 1$ is not an irreducible polynomail in GF(28), the multiplication with a fixed polynomial of 4 terms is not necessarily invertible. For this reason Rijndael specifies a fixed polynomial that has an inverse:

$$a(x) = 03 * x^3 + 01 * x^2 + 01 * x + 02 \quad a^{-1}(x) = 0b * x^3 + 0d * x^2 + 09 * x + 0e$$

That will be used in the encription and decription in the MixColumns phase.

2.3 Encryption

At the beginning of the encryption, the memorised input is copied in the State array. After an initial XOR of the Round Key, the State is transformed through a Round Transformation of 10 rounds for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys. Moreover, the last round differs from the previous Nr-1. When the last round has been performed, the State is saved in the output array. The transformation used are SubBytes (), ShiftRows (), MixColumns () e AddRoundKey ().



2.3.1 SubBytes ()

Replaces each byte (s_0, s_1, . . . , s_15) with another byte according to an S-box. In this example, the S-box is a lookup table of 256 elements. The S-box used is derived from the multiplicative inverse over GF(28), known to have good non-linearity properties:

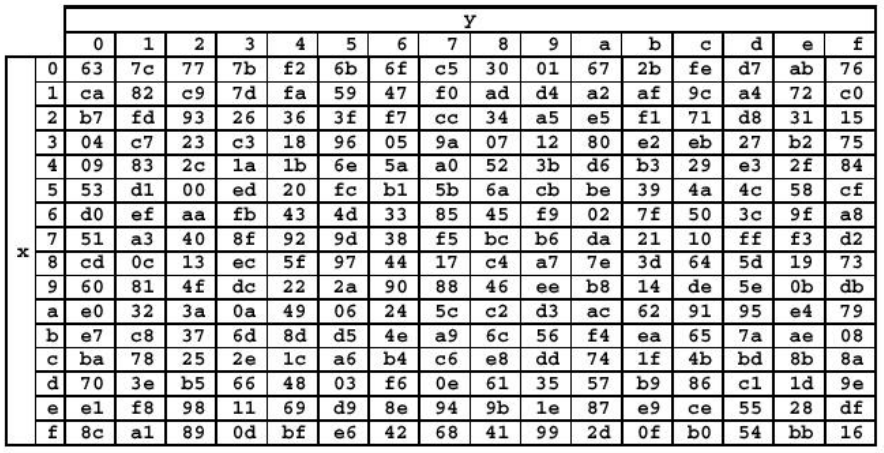
1. At first, is taken the multiplicative inverse in GF(2^8)
2. Then, is applied an GF(2^8) affine transformation defined by:

$$b_i^{'} = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

in this way the S-Box element resulted from the affine transformation can be expressed as: To simplify the S-box used can be expressed in an hexadecimal

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

form: For example, if $s_{1,1} = \{5\ 3\}$, then the value of the substitution will be



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

determined by the intersection of the row of index '5' with the column of index '3'. The result of this substitution will be therefore $s'_{1,1} = e\,d$.

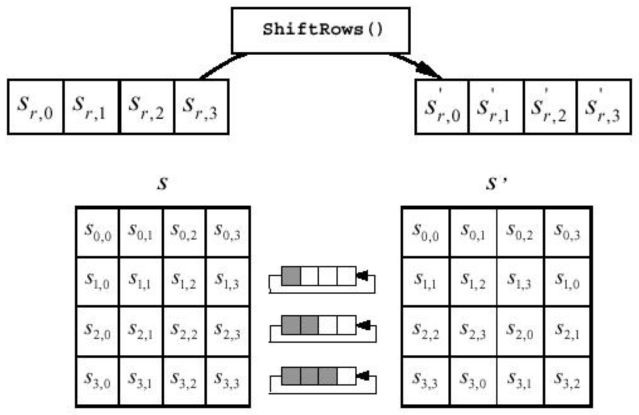
2.3.2 ShiftRows ()

The first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively.

2.3.3 MixColumns ()

This transformation works on the column of the State array. The columns are treated as polynomials of 4 terms with coefficient in the GF(2^8) finite order and are modulo $x^4 + 1$ multiplied with a fixed polynomial a(x):

$$a(x) = 03x3 + 01x2 + 01x + 02$$


ShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

As explained in the section Polynomials with coefficient in GF(2^8), this multiplication is performed with a rounded matrix such as: Giving with a row by

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

column multiplication

$$s'_{0,c} = (\{0\ 2\} * s_{0,c}) \oplus (\{0\ 3\} * s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{0\ 2\} * s_{1,c}) \oplus (\{0\ 3\} * s_{2,c}) \oplus s_{3,c}$$

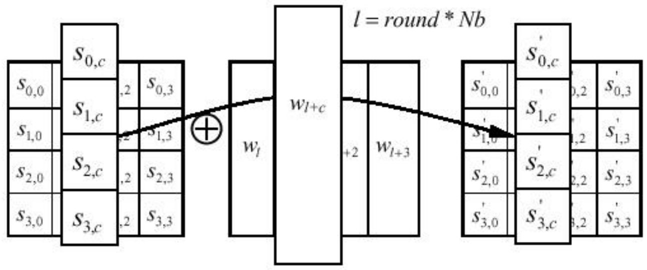
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{0\ 2\} * s_{2,c}) \oplus (\{0\ 3\} * s_{3,c})$$

$$s'_{3,c} = (\{0\ b\} * s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{0\ e\} * s_{3,c})$$

2.3.4 AddRoundKey ()

The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [RoundKey]$$



i	1	2	3	4	5	6	7	8	9	A
rc_i	01	02	04	08	10	20	40	80	1B	36

2.3.5 Key schedule

How the key round is obtained: The first keys are obtained directly from the encryption key, while the ones of the next rounds from a transformation followed by a xor with the round constant (Rcon). Rcon is a constant that takes different values depending on the round need. Can be summarised as this table:

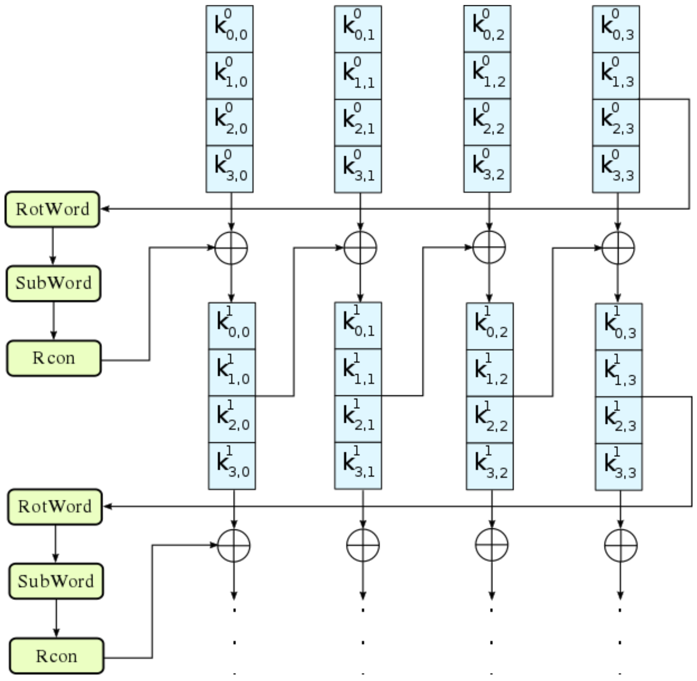
where the bits of rc_i are treated as the coefficients of an element of the finite field $GF(2)[x]/(X^8+x^4+x^3+x+1)$ so that $rc_{10} = 3616 = 001101102$ represents the polynomial $x^5+x^4+x^2+x$ As it is possible to see from the previous picture, the transformation consist of a shift of the word and of a function that applies an S-box. . This type of action is called “key whitening” and consist in the iteration of a group of operations that allow varying the key in order to increase the security of a block cipher.

2.4 Attacks

The KeyExpansion algorithm is useful to provide the security against attacks like:

- Related-Key attack(some mathematical relationship connecting the keys is known to the attacker. For example, the attacker might know that the last 80 bits of the keys are always the same, even though they don’t know, at first, what the bits are)
- Attacks in which part of the key encryption is known by the cryptoanalyst
- Attacks in which the key encryption is known and it is used as compression function (hash function)

KeyExpasion, specifically its the round structure, has also an important role in the elimination of symmetries like Round Trasformation symmetry (treats all the byte of the state in the same way) and simmetry among rounds (the Round



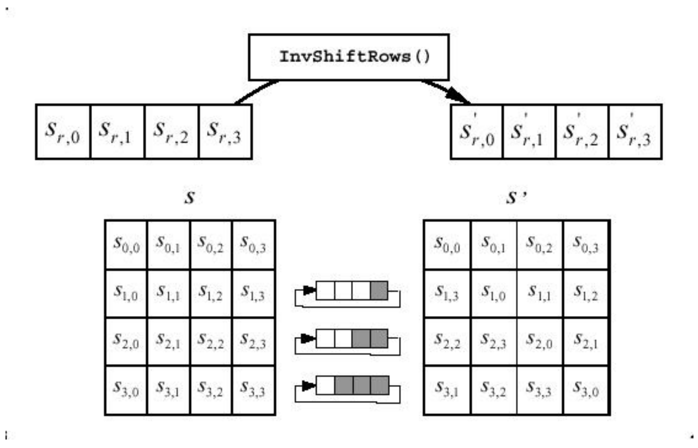
Trasformation is the same for all the rounds). Each operation contributes to AES's security in a specific way:

- Without KeyExpansion, all rounds would use the same key, K, and AES would be vulnerable to slide attacks.
- Without AddRoundKey, encryption would not depend on the key; hence, anyone could decrypt any ciphertext without the key.
- SubBytes brings nonlinear operations, which add cryptographic strength.
- Without ShiftRows, changes in a given column would never affect the other columns, meaning you could break AES by building four232-element codebooks for each column.
- Without MixColumns, changes in a byte would not affect any other bytes of the state. A chosen-plaintext attacker could then decrypt any ciphertext after storing 16 lookup tables of 256 bytes each that hold the encrypted values of each possible value of a byte.

2.5 Decryption

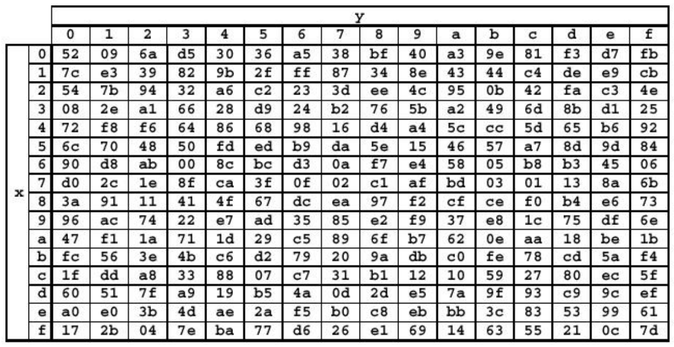
The Rijndael structure is such that the sequence of transformation in the decryption is the same of the encryption, with the transformation substituted by their inverse and a modification of the scheduling of the key.

2.5.1 InvShiftRows ()


InvShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
S
S'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,3}$ $s_{1,0}$ $s_{1,1}$ $s_{1,2}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,1}$ $s_{3,2}$ $s_{3,3}$ $s_{3,0}$

2.5.2 InvSubBytes ()

The inverse of the S-box, used in the transformation, can be simplified in the hexadecimal notation: Using the same example of the section SubBytes() :



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
x	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

$$s'_{1,1} = \{e\ d\} \longrightarrow s_{1,1} = \{5\ 3\}$$

2.5.3 InvMixColumns()

Similarly to the encryption the columns represent the polynomials with coefficient in the GF(2^8) finite field that are modulo $x^4 + 1$ multiplied with a fixed polynomial $a^{-1}(x)$:

$$a^{-1}(x) = \{0\ b\}x^3 + \{0\ d\}x^2 + \{0\ 9\}x + \{0\ e\}$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

As seen in Polynomials with coefficient in GF(2^8):

$$s'_{0,c} = (\{0\ e\} * s_{0,c}) \oplus (\{0\ b\} * s_{1,c}) \oplus (\{0\ d\} * s_{2,c}) \oplus (\{0\ 9\} * s_{3,c})$$

$$s'_{1,c} = (\{0\ 9\} * s_{0,c}) \oplus (\{0\ e\} * s_{1,c}) \oplus (\{0\ b\} * s_{2,c}) \oplus (\{0\ d\} * s_{3,c})$$

$$s'_{2,c} = (\{0\ d\} * s_{0,c}) \oplus (\{0\ 9\} * s_{1,c}) \oplus (\{0\ e\} * s_{2,c}) \oplus (\{0\ b\} * s_{3,c})$$

$$s'_{3,c} = (\{0\ b\} * s_{0,c}) \oplus (\{0\ d\} * s_{1,c}) \oplus (\{0\ 9\} * s_{2,c}) \oplus (\{0\ e\} * s_{3,c})$$

2.5.4 InvAddRounfKey()

This step only performs a XOR, therefore it is the same of the AddRoundKey().