Viene analizzato un ultimo cifrario a blocco che ha sostituito il DES.

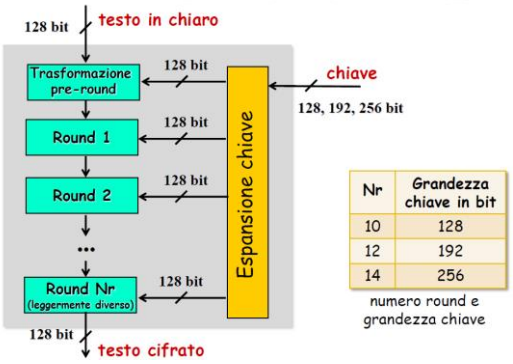
5.0 INTRODUZIONE

Le debolezze del DES sono state, la lunghezza della chiave di soli 56 bit, i blocchi di 64 bit, criteri costruttivi non chiari (trapdoor nelle S-Box), lento nell'implementazione software. L'obiettivo era dunque creare un nuovo cifrario a blocchi più ***sicuro*** (lunghezza chiave) ed ***efficiente*** (complessità computazionale dell'algoritmo) di DES triplo. Vennero stipulate un insieme di scrutini pubblici per garantiere l'autencità dei cifrari, in modo da non nascondere trappole, con l'obiettivo di garantire sicurezza, efficienza di implementazioni hardware e software, grandezza codice e memoria utilizzata. Era necessario mostrare un'analisi dell'algoritmo rispetto agli attacchi di crittoanalisi più conosciuti (es. known o chosen plaintext). I requisiti richiesti tecnici erano:

- Cifrario a blocchi.
- Lunghezza chiave 128, 192 o 256 bit (più la lunghezza è grande, più il tempo di cifratura e decifratura sarà lungo.)
- Lunghezza blocco 128 bit
- Permettere l'implementazione su smart-card (processori più piccoli, 8 bit).
- Royalty-free, ossia nessuna copertura da brevetti, senza dover pagare chi l'ha creato.

Nasce l'AES formulato da ***RIJNDAEL*** (Joan Daemon, Vincent Rijmen).

5.1 CIFRARI A BLOCCHI: ADVANCED ENCRYPTION STANDARD (AES)



Nr	Grandezza chiave in bit
10	128
12	192
14	256

numero round e grandezza chiave

L'AES si basa su un numero di round, ossia su varie iterazioni, fino ad arrivare a Nr round il cui numero dipende dalla lunghezza della chiave. Se la lunghezza è 128 bit allora ci saranno 10 iterazioni, 192 bit equivale a 12 iterazioni, 256 bit equivale a 14 iterazioni. L'ultima iterazione si differisce dalle altre per una semplice differenza. Anche all'inizio avviene una trasformazione pre-round. Ad ogni iterazione c'è una sottochiave di 128 bit e ognuna di queste sottochiavi viene calcolata in output da un'espansione della chiave.

Un'osservazione: quanto è più lenta la cifratura con una chiave a 192 bit, rispetto a 128? Naturalmente il 20%, perché da 10 passo a 12. Analogamente un'implementazione a 256 bit rispetto a quella a 128 bit, sarà più lenta del 40%.

Il seguente AES non è un cifrario di Feistel poiché si lavora sull'intero blocco in input, e la cifratura e la decifratura sono due processi differenti. Usa operazioni facilmente ed efficientemente implementabili su architetture ad 8 e 32 bit.

Una caratteristica importante dell'AES è la sua struttura matematica elegante ed efficiente. Il concetto su cui si basa l'AES è lo stato.

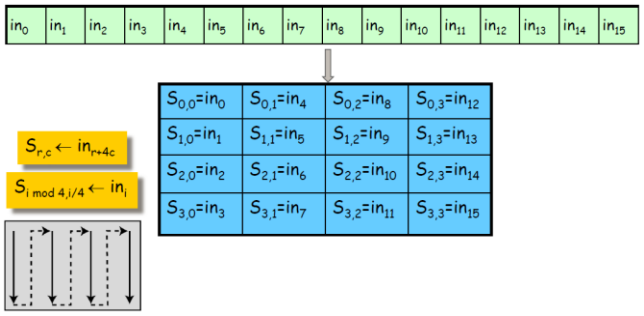
5.1.1 STATO

Tutte le trasformazioni che vengono fatte di volta in volta, avvengono in una *matrice di byte*, detta ***state***. È una matrice 4x4 in cui ogni colonna di questa matrice è una parola di 32 bit. Una riga e una colonna sono quindi 128 bit (32x4).

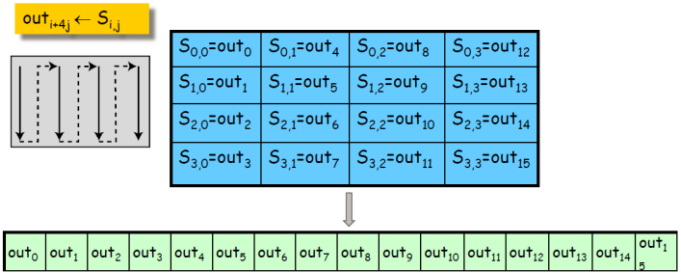
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

RICORDA: 32 bit = 4 byte; 128 bit = 16 byte.

Il primo step dunque da fare è quello di riempire la matrice di stato. Inizialmente si ha in input una stringa di 128 bit, che deve essere inserita nella matrice di stato. Il numero di bit sono 128 e si possono dividere in 16 byte (in_0, in_1, in_2, …, in_{15}), infatti 16x8=128. L'inserimento è molto intuitivo, basta osservare l'immagine seguente (in particolare la parte in basso a sinistra che descrive come avviene il posizionamento):

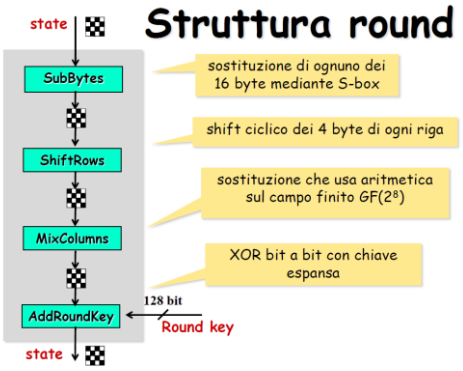


Una effettuate tutte le trasformazioni la matrice deve essere scritta per ottenere il testo cifrato. L'idea è la stessa della precedente e viene ora mostrata:



5.2 STRUTTURA ROUND

La struttura di ogni round è la seguente, c'è uno stato iniziale che viene sottoposto a 4 operazioni:



L'idea essenziale è quella di effettuare prima una sostituzione degli elementi della matrice, un'operazione sulle righe, una sulle colonne e poi aggiungo la sottochiave. All'inizio abbiamo detto che l'ultimo round era un po' diverso, infatti manca solo l'operazione di *MixColumns*. Nella trasformazione pre-round invece, manca solo lo step *AddRoundKeyMixColumns.*

Prima di analizzare le 4 operazioni di ogni round, è necessario dire che ***il byte è l'unità di base nella computazione dell'AES***, per far girare l'implementazione su smart-card. Sui byte, l'AES ha definito operazioni di addizione e moltiplicazione, con una struttura di GF(2^8), cioè campo finito con 256 elementi.

I valori dei byte sono rappresentati in notazione esadecimale: due cifre per ciascun byte → $\{11010100\}$ → d4

$\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ → $b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$

L'altro modo per interpretare il byte, che usa l'AES, è vederlo come coefficiente di un polinomio: $\{11010100\}$ → $x^7+x^6+x^4+x^2$

Osservando l'esempio, si ha questo byte {11010100} che è equivalente al polinomio scritto alla destra, i cui coefficienti sono i bit del byte. Ad esempio, è presente x^7, poiché il valore b_7 è uguale a 1, dunque è come se ci fosse 1* x_7. Analogamente per x_6. Giustamente x_5 non appare poiché è come se fosse 0 * x_5 che equivale a 0. Tutte le operazioni effettuate sul byte è come se le andassi a fare sul polinomio. Addizioni e moltiplicazioni sui byte, corrispondono ad addizioni e moltiplicazioni sui polinomi.

5.2.1 ADDIZIONE SU BYTE

Viene svolta l'addizione sui polinomi, quindi si assuma la seguente somma:

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

$$\{01010111\} \oplus \{10000011\} = \{11010100\}$$

$$\{57\} \oplus \{83\} = \{d4\}$$

I valori dei byte posti alla seconda riga, sono equivalenti ai polinomi scritti sulla prima riga. La somma dei polinomi avviene banalmente, si osserva però che la x non compare nella somma, questo perché avendo x ad entrambi i polinomi, la risultante dovrebbe essere 2x, ma in binario 1+1 è un'operazione di XOR che corrisponde a 0. Dunque verrebbe 0 * x= 0. Situazione analoga per 1+1.

L'addizione di due byte, naturalmente avviene in maniera immediata, senza fare l'operazione sui polinomi, mediante lo XOR bit a bit tra i due byte.

5.2.1 MOLTIPLICAZIONE SU BYTE

Quando si effettua la moltiplicazione tra 2 polinomi, la risultante ovviamente è un polinomio più grande, ad esempio $x^6 * x^7 = x^{13}$, naturalmente se volessi rappresentare questo termine, 1 byte non sarebbe sufficiente, in quanto, con 1 byte posso arrivare al massimo a x^7. L'idea per far rientrare la moltiplicazione risultante all'interno di un singolo byte, la divido per un polinomio di grado 8 e ne considero il resto. Questo mi assicura che il polinomio risultante, è di grado al massimo 7, e riesco quindi a rappresentarlo in un byte.

L'AES ha scelto un polinomio fissato per effettuare la divisione: $m(x)= x^8 + x^4 + x^3 + x + 1$ che viene definito polinomio ***irriducibile*** che verrà descritto dopo.

Analizziamo l'esempio precedente, questa volta con la moltiplicazione:

$\{01010111\} \bullet \{10000011\} = \{11000001\}$

$\{57\} \bullet \{83\} = \{c1\}$

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

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

dividiamo per m(x) e teniamo il resto

Come prima, il primo byte corrisponde al primo polinomio e il secondo byte al secondo polinomio. Si effettua banalmente la moltiplicazione tra i due polinomi e la risultante è un polinomio in cui nessun esponente è ripetuto. Il seguente polinomio non entra in un byte ovviamente, quindi si effettua un'operazione di modulo tra il polinomio risultante e il polinomio fissato dall'AES. Il resto di questo modulo è il mio risultato (in questo caso $x^7 + x^6 + 1$, che corrisponde a 1100001). La divisione è descritta qui sotto:

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ diviso $x^8 + x^4 + x^3 + x + 1$

$$\begin{array}{r|l} & x^5 + x^3 \\ \hline x^8 + x^4 + x^3 + x + 1 & x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \\ & x^{13} + x^9 + x^8 + x^6 + x^5 \\ \hline & x^{11} + x^4 + x^3 + 1 \\ & x^{11} + x^7 + x^6 + x^4 + x^3 \\ \hline & x^7 + x^6 + 1 \end{array}$$

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

Il modo più veloce per implementare la moltiplicazione in questo caso è quello di costruire una tabella in cui vengono precalcolati tutti i valori perché sono 256 x 256. Analogamente per calcolare gli inversi moltiplicativi che sono 256, si possono calcolare in un array di 256 valori. Viene precalcolato e quando serve un valore si vede qual è il valore.

Torniamo al discorso del polinomio che l'AES ha scelto. Avevamo detto che viene definito irriducibile e questo significa che gli unici divisori di questo polinomio sono 1 e se stesso, un comportamento simile ad un numero primo. Inoltre permette che ogni elemento ha un inverso di tipo moltiplicativo. Se il polinomio non fosse stato irriducibile, allora esistevano elementi che non avrebbero avuto un inverso moltiplicativo. A seconda del grado del polinomio esistono un numero fissato di polinomi irriducibili.

n	1	2	3	4	5	6	7	8	9	10	11	12
#poly	2	1	2	3	6	9	18	30	56	99	186	335

L'addizione e la moltiplicazione soddisfano le seguenti proprietà:

- Addizione
  - gruppo abeliano
    - Associativa: $(a(x)+b(x))+c(x) = a(x)+(b(x)+c(x))$
    - Commutativa: $a(x)+b(x) = b(x)+a(x)$
    - Identità {00}: $a(x)+0 = 0+a(x) = a(x)$
    - Esiste inverso -a(x) per ogni a(x): $-a(x)+a(x) = 0$
- Moltiplicazione
  - gruppo abeliano
    - Associativa: $(a(x)\bullet b(x))\bullet c(x) = a(x)\bullet(b(x)\bullet c(x))$
    - Commutativa: $a(x)\bullet b(x) = b(x)\bullet a(x)$
    - Identità {01}: $a(x)\bullet 1 = 1\bullet a(x) = a(x)$
    - Esiste inverso $a^{-1}(x)$ per ogni $a(x)\neq 0$: $a^{-1}(x)\bullet a(x) = 1$
- $a(x) \bullet (b(x) + c(x)) = a(x) \bullet b(x) + a(x) \bullet c(x)$
- Struttura del campo finito GF(2^8)

Queste operazioni viste, sono sui singoli byte, ma a noi per l'argomento proposto, le operazioni interessano sulle word che sono a 32 bit che corrispondono a 4 byte. Le operazioni quindi sono le stesse descritte sopra, ad esempio:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

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

Moltiplicazione $c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$

$c_0 = a_0 \bullet b_0$

$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1$

$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$

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

$c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$

$c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$

$c_6 = a_3 \bullet b_3$

L'addizione, come prima, è uno XOR bit a bit dei 32 bit, banalmente. Per la moltiplicazione, come descritto in precedenza, il prodotto ci dà un risultato che non riesco a rappresentare in una word perché il grado massimo del polinomio deve essere 3, ma può essere anche 4,5,6. Come prima, si fa il modulo del prodotto e un polinomio non irriducibile questa volta che è x^4+1, e si prende il resto che è sicuramente rappresentabile in una word. In questo caso non posso velocizzare il processo della moltiplicazione, perché si creerebbe una tabella $2^{32}x2^{32}= 2^{64}$ entrate che è un'enormità. La tabella quindi non si può fare, il processo di moltiplicazione risulta essere lungo, con molti calcoli. La soluzione a ciò è nel polinomio x^4+1, infatti:

$$x^4 \equiv -1 \equiv 1 \bmod (x^4+1)$$
$$x^5 \equiv -x \equiv x \bmod (x^4+1)$$
$$x^6 \equiv -x^2 \equiv x^2 \bmod (x^4+1)$$
$$\dots$$
$$x^i \equiv x^{i \bmod 4} \bmod (x^4+1) \qquad \text{in generale}$$

Questa proprietà facilita il calcolo della moltiplicazione mod x^4+1

Quindi, se dopo aver effettuato un prodotto, esce un x^6, mediante questa piccola tabella io posso subito scriverlo come x^2. x^5 posso scriverlo come x e x^4 lo scrivo come 1. Si ricorda che queste proprietà derivano esclusivamente dalla scelta di x^4+1. La moltiplicazione quindi diventa è efficiente e semplice ed è:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

Moltiplicazione mod x^4+1 $\quad d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$

$$\begin{cases} d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{cases}$$

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

Si ricorda che la moltiplicazione tra matrici si fa mediante prodotto riga per colonna.

Come detto in precedenza, $x^4 + 1$ non è irriducibile su GF(2^8), questo implica che non tutti i polinomi hanno un inverso mod x^4+1. Però AES usa un particolare polinomio cioè:

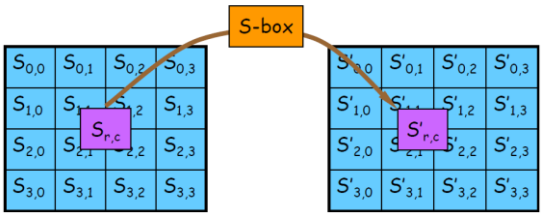
$a(x)= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ che ha un inverso:

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

Dopo aver descritto le operazioni, possiamo descrivere le fasi che formano un round:

5.2.2 SUBBYTES TRANSFORMATION

$S'_{r,c} \leftarrow \text{S-box}(S_{r,c}) \qquad 0 \le r < 3 \qquad 0 \le c < Nb\text{-}1$



La prima fase che prevede un round è la trasformazione, come detto in precedenza. Come si vede dalla foto, lo stato a sinistra è quello che entra nel round, e quello a destra è il trasformato. Nel dettaglio, ogni singolo byte della matrice, viene sostituito da un nuovo byte. Effettuo quindi 16 sostituzioni, una per ogni byte.

La sostituzione è data da questa matrice:

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Se ad esempio ho l'elemento 3C in esadecimale, trovo l'elemento corrispondente alla riga e alla colonna e osservo che la tabella mi restituisce eb, che quindi sostituirò. Un funzionamento simile, avveniva nell'S-box del DES. Questa tabella ci spiega il modo diretto per calcolare un output per la sostituzione, per non sollevare alcun alibi riguardo la segretezza, esiste un algoritmo che ci spiega la costruzione di questo s-box, che si basa su questi step:

- Inizializzare la S-box con i valori dei byte in ordine ascendente riga per riga (Prima riga: {00}, {01}… {0F}; Seconda riga: {10}… {1F},…)
- Sostituire ciascun byte con il suo inverso moltiplicativo in GF(2^8)
- Applicare una trasformazione affine in GF(2^8)

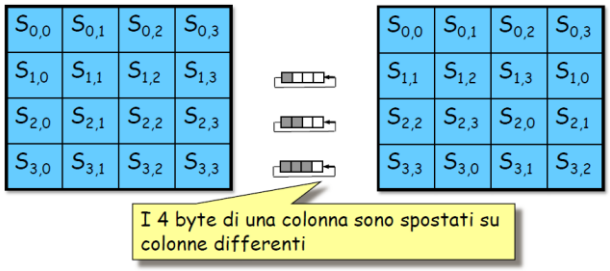
$$b_i' \leftarrow b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus \{01100011\}$$

i-esimo bit del byte {63}

L'S-box è stata progettata per resistere ad attacchi crittoanalitici noti. L'S-box può essere invertita, dunque nella decifratura, si inverte ogni singolo passaggio. La tabella inversa è:

x \ 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
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

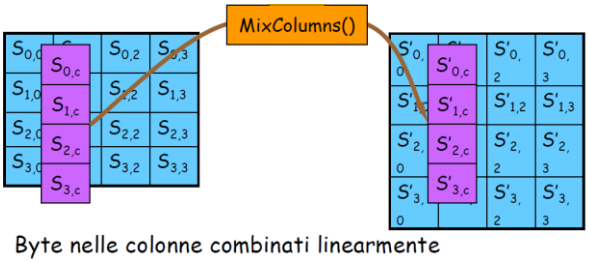
5.2.3 SHIFTROWS TRANSFORMATION



Rappresenta la seconda fase che avviene all'interno di un round. L'immagine rende molto chiara l'idea dello shift: la prima riga non subisce variazioni. Nella seconda riga, il primo byte rientra come ultimo byte dopo lo shift degli altri 3 byte. Nella terza riga si fa uno shift di 2 byte sempre a sinistra. Nella quarta riga si fa uno shift di 3 byte sempre a sinistra.

Anche questa operazione ha un'inversa, cioè passare dalla matrice di destra a quella di sinistra, mediante uno shift ciclico a destra, questa volta, con la stessa quantità di byte presi per ogni riga.

5.2.4 MIXCOLUMNS TRANSFORMATION



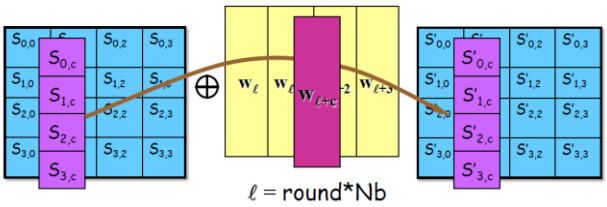
Nel terzo step di un round, l'operazione che viene effettuata è sulle colonne. Tenendo conto che ogni colonna sono 32 bit (perché ogni singolo elemento della matrice è 1 byte, dunque una colonna sono 4 byte, che corrispondono a 32 bit), considero la word della singola colonna, la moltiplico per il polinomio $a(x)= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ modulo x^4+1. Questo calcolo mi restituisce un nuovo polinomio, di 32 bit, che scrivo sulla colonna. Questa fase si traduce più semplicemente in somma prodotto tenendo in considerazione questa matrice:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} \leftarrow \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Per il processo inverso, ossia rientrare alla matrice di sinistra, moltiplico la matrice per l'inversa di a(x), cioè: $a^{-1}(x)= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$, modulo x^4+1, e la nuova matrice da considerare è:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} \leftarrow \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

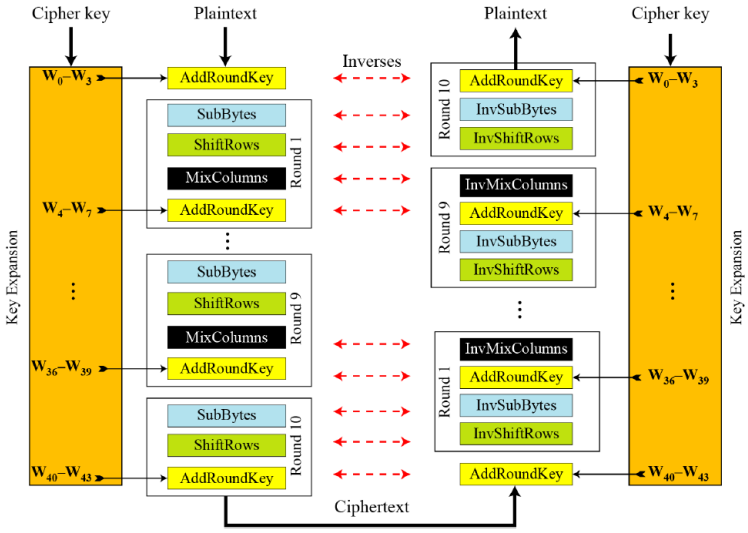
5.2.5 ADDROUNDKEY TRANSFORMATION



L'ultima fase di un round è l'aggiunta di una sottochiave, che consiste nel fare uno XOR bit a bit con i vari pezzi della sottochiave.

Quando vado a fare l'inversione per la sottochiave, essendo un'operazione di XOR, l'operazione con la sottochiave chiaramente è l'inversa di sé stesso perché è un'operazione XOR bit a bit.

Possiamo riassumere l'operazione di cifratura e decifratura con la seguente immagine:



Quindi, in alto a sinistra entra il plaintext, vengono eseguiti i vari passaggi fino ad ottenere il testo cifrato. Successivamente, nell'algoritmo di decifrazione, vengono eseguiti i vari passaggi uno alla volta come mostrato in figura, da notare che si invertono i round.

Anche l'AES ha un avalanche effect, questo significa che un piccolo cambiamento del testo in chiaro oppure della chiave produce un grande cambiamento nel testo cifrato.

Cipher Key:	24 75 A2 B3 34 75 56 88 31 E2 12 00 13 AA 54 87
Plaintext 1:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Plaintext 2:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
Ciphertext 1:	63 2C D4 5E 5D 56 ED B5 62 04 01 A0 AA 9C 2D 8D
Ciphertext 2:	26 F3 9B BC A1 9C 0F B7 C7 2E 7E 30 63 92 73 13