



AES – Advanced Encryption Standard

Simone Forcella

Simone Lai

Daniele Ninfo

Francesco Forconi



AES – Cenni storici

Simone Forcella

Origini

- **1969** - La decisione di creare uno standard per la protezione dei dati nasce negli Stati Uniti per soddisfare una legge detta "Brooks Act";
- Relativa al miglioramento della sicurezza nell' utilizzo dei calcolatori del governo statunitense;
- **1972** - NBS (National Bureau of Standard) decide l'adozione di un metodo crittografico

Origini

- 1977 – viene implementato il DES
- DES è un' evoluzione della NSA di un algoritmo dell' IBM “Lucifer”
- 1997 – il DES viene violato per la prima volta (ci vogliono 39 giorni)
- 1999 – la “DES cracker machine” viola definitivamente lo standard in 22 ore!!

Concorso?

- Il governo U.S.A. si rende conto che ormai il **DES** stava giungendo alla fine della sua vita utile(anche nella versione 3DES)
- Il **NIST** (**National Institute of Standard and Technology, ex NBS**) l'agenzia del dipartimento di commercio, che ha il compito di approvare gli standard federali per il governo degli Stati Uniti opta per un concorso

Concorso?

- Se il NIST avesse annunciato un nuovo standard, chiunque sapesse qualcosa di crittografia avrebbe pensato automaticamente che l' **NSA** (National Security Agency, agenzia per la sicurezza nazionale, parte del governo USA che si occupa di forzare i cifrari) aveva una back door per questo standard, e avrebbe potuto leggere tutti i messaggi.

Concorso!!

- Per questo motivo NIST adottò un approccio diverso da quello della normale burocrazia governativa e sponsorizzò un concorso crittografico. Nel gennaio **1997** i ricercatori di tutto il mondo furono invitati a inviare delle proposte per un nuovo standard che avrebbe preso il nome di **AES** (**advanced encryption standard**).

Concorso

Nel 1997 inizia il concorso con i seguenti criteri:

Criteri di scelta:

- Il cifrario deve essere a chiave simmetrica;
- Il cifrario deve essere a blocchi;
- La lunghezza della chiave deve essere di 128 o 256bit
- I blocchi del testo in chiaro possono essere di 64, 128, 256 bit
- Deve poter essere implementabile su Smart Card
- Deve essere scritto in C o Java (scelti perché più diffusi)
- Deve poter garantire una distribuzione dell' algoritmo a livello mondiale

Concorso

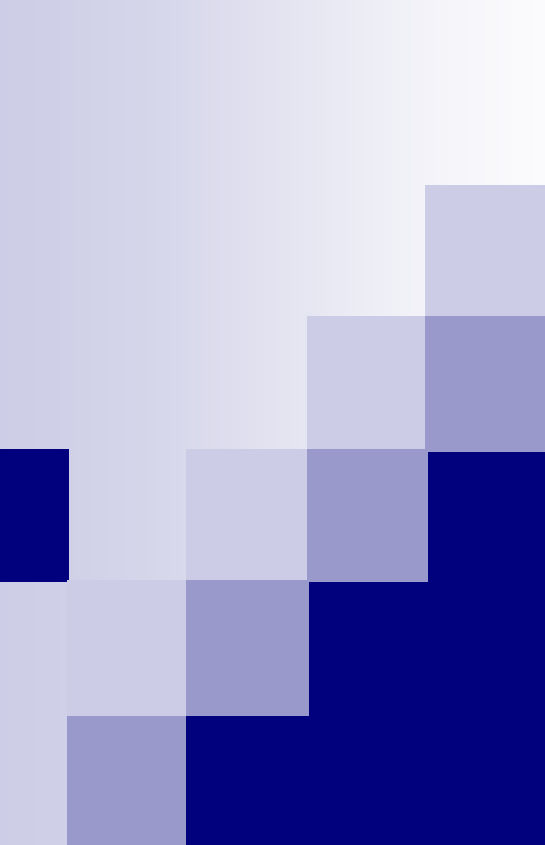
- Criteri d' analisi:
- **Sicurezza:** l'algoritmo doveva essere robusto
- **Costo:** in termini di efficienza e velocità computazionale
- **Semplicità:** ne determina le caratteristiche di costruzione (hardware) e di implementazione (software)
- La selezione avviene in Conferenze d'ufficializzazione chiamate: ROUND 1,2,3 ;

Concorso

- **1998** - Round 1 :
 - ◆ Vengono selezionati solo 15 algoritmi
- **1999** – Round2 :
 - ◆ Rimangono solo in 5
 1. MARS dell' IBM
 2. RC6 della RCS
 3. **Rijndael** (di Joan Daemen e Vincent Rijmen)
 4. Serpent (Anderson, Biham, Knudsen)
 5. TwoFish(Schneider,Kelsey,Whiting,Wagner,Hall,Ferguson)

Rijndael diventa AES

- 2000 – ROUND 3:
 - ◆ Si conclude con la vittoria di Rijndael
(si legge “Raindoll”)
- “Rijndael” diventa A.E.S.
- Documentato ufficialmente in F.I.P.S. PUB. 197
(Federal Information Processing Standard Publication)



AES – Basi matematiche

Simone Lai

I bytes in Rijndael

- Molte operazioni in Rijndael sono definite lavorando sui singoli bytes.
- I valori dei singoli bit di ciascun byte sono interpretati come coefficienti dei polinomi del campo finito $GF(2^8)$.

I campi di Galois (definizioni)

- Si dice campo un anello *commutativo unitario* $(A, +, \cdot)$ (in cui l'operazione \cdot è commutativa e esiste un elemento neutro per \cdot (ad esempio 1)) tale che:

per ogni $a \in A, a \neq 0$

esiste $a' \in A$

t.c. $a \cdot a' = a' \cdot a = 1$

I campi di Galois (definizioni)

- Un polinomio $f(x) \in K[x]$ (anello di polinomi con coefficienti nel campo K), $\partial(f) = n$, si dice *irriducibile* in $K[x]$ se non si può scrivere come prodotto di due polinomi $h(x), k(x)$ a coefficienti in $K[x]$, con $0 < \partial h < n$ e $0 < \partial k < n$.

□ Es. x^2+1 è riducibile in $\mathbb{R}[x]$?

I campi di Galois (definizioni)

- Sia Z_p il campo finito degli interi mod p (p primo, altrimenti Z_p non sarebbe un campo);
- Sia $Z_p[x]$ l'anello dei polinomi a coefficienti in Z_p , e sia $m(x) \in Z_p[x]$, $m(x)$ *irriducibile* in $Z_p[x]$, $\partial m = n$.
- Chiamiamo Campo di Galois rispetto al polinomio $m(x)$, il campo costituito dalle $q = p^n$ classi di congruenza mod $m(x)$:

$$GF(q) = GF(p^n) = Z_p[x]/m(x)$$

I campi di Galois (esempio)

- Costruire $GF(2^2)$, rispetto al polinomio irriducibile in $\mathbb{Z}_2[x]$:

$$m(x) = x^2 + x + 1$$

- Quindi bisogna determinare le $2^2 = 4$ classi di congruenza mod $x^2 + x + 1$.
- I resti della divisione di ogni polinomio $f(x) \in \mathbb{Z}_2[x]$ sono t.c. $r(x) = 0$ opp. $0 \leq \partial r(x) < 2$.

I campi di Galois nell'AES

- Nel Rijndael quindi ogni byte (sequenza di 8 bit) viene interpretato *come un polinomio* in $GF(2^8)$ in questo modo:

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

- Esempio: 57(hex) \Rightarrow ?($GF(2^8)$)

$$57(\text{hex}) \Rightarrow 01010111(\text{bin}) \Rightarrow x^6 + x^4 + x^2 + x + 1 \quad (GF(2^8))$$

Somma tra bytes

- La somma di due elementi nel campo finito è ottenuta sommando i coefficienti delle corrispondenti potenze nei due polinomi.
- Riportando il calcolo in Rijndael, la somma di due bytes si ottiene effettuando uno XOR (\oplus) bit a bit dei due bytes.

Somma tra bytes (esempio)

- Notazione polinomiale:

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

- Notazione binaria:

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

- Notazione esadecimale:

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

Moltiplicazione tra bytes

- La moltiplicazione nel campo finito è ottenuta moltiplicando i due polinomi modulo un polinomio irriducibile di ottavo grado, che nell'AES è:
$$m(x) = x^8 + x^4 + x^3 + x + 1$$
 (`{01}{1b}` in esadecimale)
- Purtroppo non c'è un'operazione semplice a livello byte per effettuare questo tipo di calcolo. Ma possiamo notare che questo tipo di moltiplicazione è associativo e che l'elemento neutro è `{01}`.

Moltiplicazione tra bytes

- Inoltre per qualunque polinomio $b(x)$, diverso da zero e di grado inferiore a 8, si può ricavare l'inverso $b^{-1}(x)$ tramite l'algoritmo esteso di Euclide partendo da:

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

- Quindi, dopo aver ottenuto $a(x)$ e $c(x)$, essendo:

$$a(x) \cdot b(x) \bmod m(x) = 1$$

- posso ricavare:

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

Riepilogo operatori in $GF(2^8)$

- Addizione in $GF(2^8) \Rightarrow \text{XOR}$

- Moltiplicazione in $GF(2^8) \Rightarrow \cdot$

- Proprietà:

- Identità (elemento neutro $\{01\}$)

- Associatività:

$$a(x) \cdot (b(x) \oplus c(x)) = a(x) \cdot b(x) \oplus a(x) \cdot c(x)$$

- Inversa:

per ogni $b(x) \neq 0$, $\partial b < 8$, esiste $b^{-1}(x)$

Moltiplicazione per x

- Moltiplicando un polinomio binario generico per il polinomio x ($\{00000010\}$ o $\{02\}$) risulta:

$$p(x) = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

- *Ora bisogna ridurre* questo risultato modulo $m(x)$:
 - Se $b_7 = 0$, il risultato è già ridotto;
 - Se $b_7 = 1$, la riduzione si può effettuare sottraendo (o sommando) a $m(x)$ $p(x)$! (effettuando uno XOR)
- Quindi la moltiplicazione per x può essere implementata con un left shift e se serve uno XOR con $\{1b\}$. (`xtime()`)

Implementazione della moltiplicazione

- Definendo la funzione $xtime()$, possiamo realizzare un'adeguata implementazione della moltiplicazione tra polinomi in $GF(2^8)$:
- Esempio: calcolare $\{57\} \cdot \{13\}$
 - $\{57\} \cdot \{02\} = \{01010111\} \cdot \{00000010\} = xtime(\{57\}) = \{ae\}$
 - $\{57\} \cdot \{04\} = \{01010111\} \cdot \{00000100\} = xtime(\{ae\}) = \{47\}$
 - $\{57\} \cdot \{08\} = \{01010111\} \cdot \{00001000\} = xtime(\{47\}) = \{8e\}$
 - $\{57\} \cdot \{10\} = \{01010111\} \cdot \{00010000\} = xtime(\{8e\}) = \{07\}$
- Quindi: $\{57\} \cdot \{13\} = \{57\} \cdot (\{01\} \oplus \{02\} \oplus \{10\})$
$$= \{57\} \oplus \{ae\} \oplus \{07\}$$
$$= \{fe\}$$

Polinomi con coefficienti in $GF(2^8)$

- Possono essere definiti polinomi di 4 termini con coefficienti in $GF(2^8)$:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

- *In questo modo ogni coefficiente a_x equivale ad un byte e viene gestito come spiegato nelle slides precedenti.*

$[a_3, a_2, a_1, a_0]$ \Leftarrow insieme dei coefficienti (word)

Polinomi con coefficienti in $GF(2^8)$ - Addizione

- Dati i due polinomi $a(x)$ e $b(x)$:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

- $a(x) + b(x)$ si effettua sommando i coefficienti delle x con lo stesso grado, ma la somma deve essere calcolata in $GF(2^8)$, quindi effettuiamo lo XOR dei coefficienti:

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

Polinomi con coefficienti in $GF(2^8)$ – Moltiplicazione - Step 1

- $a(x) \cdot b(x)$ si effettua calcolando inizialmente il prodotto algebrico tra $a(x)$ e $b(x)$, senza ridurre le x di grado superiore al terzo:

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

$$c_0 = a_0 \cdot b_0$$

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

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

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

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

$$c_6 = a_3 \cdot b_3$$

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

- *Problema: il risultato non è rappresentabile da una parola di 4 bytes!*

Polinomi con coefficienti in $GF(2^8)$ – Moltiplicazione - Step 2

- Il secondo passo consiste nel ridurre $c(x)$ modulo un polinomio di grado 4. Nell'AES, questo polinomio è $x^4 + 1$. Quindi:

$$x^i \text{ mod } (x^4 + 1) = x^{i \text{ mod } 4}$$

- Il 'prodotto modulare' $d(x) = a(x) \otimes b(x)$ è dato da:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

$$\text{con } d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3) = c_0 \oplus c_4$$

$$d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3) = c_1 \oplus c_5$$

$$d_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3) = c_2 \oplus c_6$$

$$d_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3) = c_3$$

Polinomi con coefficienti in $GF(2^8)$ – Moltiplicazione - Note

- Quando $a(x)$ è un polinomio fissato, i coefficienti d_x possono essere scritti in forma matriciale:

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

- Siccome $x^4 + 1$ non è irriducibile in $GF(2^8)$, la moltiplicazione per un polinomio di quattro termini fissati non è necessariamente invertibile; per questo l'AES specifica un polinomio di quattro termini che ha un'inversa:

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



AES – Algoritmo di cifratura

Daniele Ninfo

Requisiti dell'algoritmo

- Input diviso in blocchi da 128 bit, riuniti dopo la crittazione
- Chiave da 128, 192 o 256 bit
- Input e output sotto forma di sequenze di bit (array), ma considerati logicamente come matrici

Unità di calcolo

L'unità di elaborazione dell'algoritmo è il singolo byte. Ogni elemento delle matrici che rappresentano input e output è un byte.

- Matrice in input: 4×4 (128 bit = 16 byte)
- Chiave: 4×4 , 4×6 oppure 4×8

Notazioni

- Word – singola colonna di un blocco
- Nb – numero di word del blocco in input
(nella versione ufficiale di AES Nb=4)
- Nk – numero di word della chiave
(a seconda della versione, Nk=4,6,8)
- Nr – numero di round in cui avverrà la codifica

Round

Diversi round di cifratura, il cui numero dipende dalla lunghezza della chiave

	Key Length <i>(N_k words)</i>	Block Size <i>(N_b words)</i>	Number of Rounds <i>(N_r)</i>
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Funzioni principali

- KeyExpansion

- Basandosi sulla prima, genera altre Nr chiavi

- Cipher

- Codifica il testo sulla base delle chiavi fornite

Cipher riceve all'inizio tutte le chiavi da KeyExpansion, ad ogni round il messaggio sarà crittato con una chiave diversa

State

Per l'elaborazione verrà utilizzata una matrice d'appoggio chiamata State, su cui verrà copiato l'input e estratto l'output

input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

state

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

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}


$$S_{r,c} \leftarrow in_{r+4c}$$

$0 \leq r < 3 \quad 0 \leq c < Nb-1$


$$out_{r+4c} \leftarrow S_{r,c}$$

$0 \leq r < 3 \quad 0 \leq c < Nb-1$

Cipher (1)

È la funzione che codifica effettivamente il messaggio. La cifratura avverrà in N_r round, a seconda della lunghezza della chiave: $N_r - 1$ round sono identici, mentre un round è diverso dagli altri. Ogni round utilizzerà una chiave diversa.

Cipher (2)

Gli Nr-1 round consistono nell'applicazione in cascata di quattro funzioni all'input

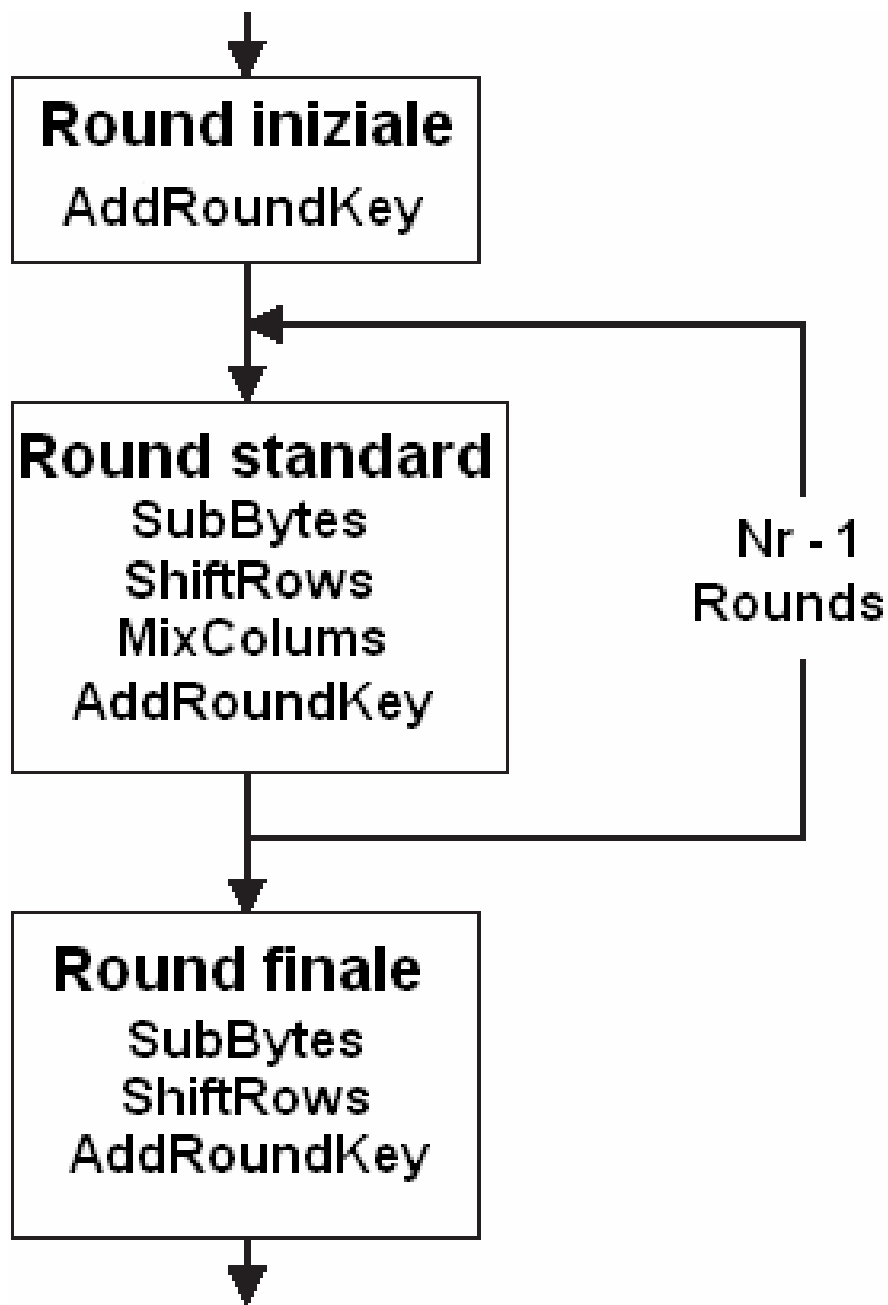
- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

Cipher (3)

Il restante round è diverso dagli altri perché sostituisce a MixColumns un'altra occorrenza di AddRoundKey, per questo necessita di due chiavi, e l'ordine di applicazione delle funzioni è diverso. Inoltre, questo round inizia prima degli altri e termina dopo gli altri

Cipher (4)

- I round iniziale e finale formano un unico round
- Per implementare gli $Nr-1$ round sarà necessario realizzare un ciclo



```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0, Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end
```

SubBytes

I byte vengono codificati attraverso una funzione di sostituzione, che presenta queste caratteristiche:

- Invertibilità
- Non linearità

SubBytes – S-Box (1)

La sostituzione avviene secondo una tabella che riporta, per ognuno dei 256 valori che un byte può esprimere, il valore (unico e diverso da tutti gli altri) che il byte deve assumere. Questa tabella è chiamata Substitution-Box, S-Box, ed è esprimibile come una matrice 16×16 (256 valori)

SubBytes – S-Box (2)

Costruzione: partendo dalla matrice 16×16 contenente tutti i possibili valori di un byte ordinati, si applicano 2 trasformazioni in cascata ad ogni elemento

- Si sostituisce il suo inverso in $GF(2^8)$ (il byte 00000000 rimane tale)

SubBytes – S-Box (3)

- Si applica questa trasformazione affine ad ogni singolo bit b_i del byte

$$b_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i$$

Con c_i i -esimo bit del byte 01100011, valore fisso

In pratica, ogni bit viene trasformato in base ai 4 bit precedenti nel byte e ad un bit esterno. Il modulo 8 consente di trasformare anche i primi bit.

$$\text{Es: } b_2 = b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2$$

SubBytes – S-Box (4)

	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	7d
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	cd
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	1e
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	7f
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	8e
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	c4
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a1
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	7c
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	df
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	7e
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	0d
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	d5
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	1a

ShiftRows (1)

Le righe della matrice vengono “shiftate” verso sinistra di un numero di posizioni determinato dalla formula

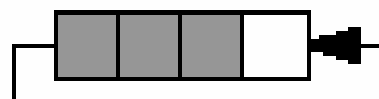
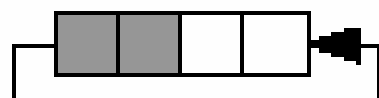
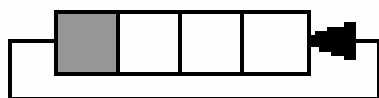
$$S'_{r,c} = S_{r,(c-\text{shift}(r,Nb))\bmod Nb} \quad 0 < r < 4, 0 \leq c < Nb$$

con $\text{shift}(1,4)=1$, $\text{shift}(2,4)=2$, $\text{shift}(3,4)=3$

In pratica, la prima riga rimane uguale, la seconda si sposta di una, la terza di due, la quarta di tre posizioni

ShiftRows (2)

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



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

MixColumns (1)

- La funzione opera sulle colonne, ogni byte della colonna è trattato come un polinomio in $GF(2^8)$, ed è un coefficiente di colonna
- Ogni colonna viene moltiplicata modulo x^4+1 per un polinomio fissato $a(x)$

$a(x)=03x^3+01x^2+01x+02$ (coeff. esadecimale)

ogni coefficiente di $a(x)$ è un polinomio in $GF(2^8)$ e sarà moltiplicato per ogni byte

MixColumns (2)

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb$$

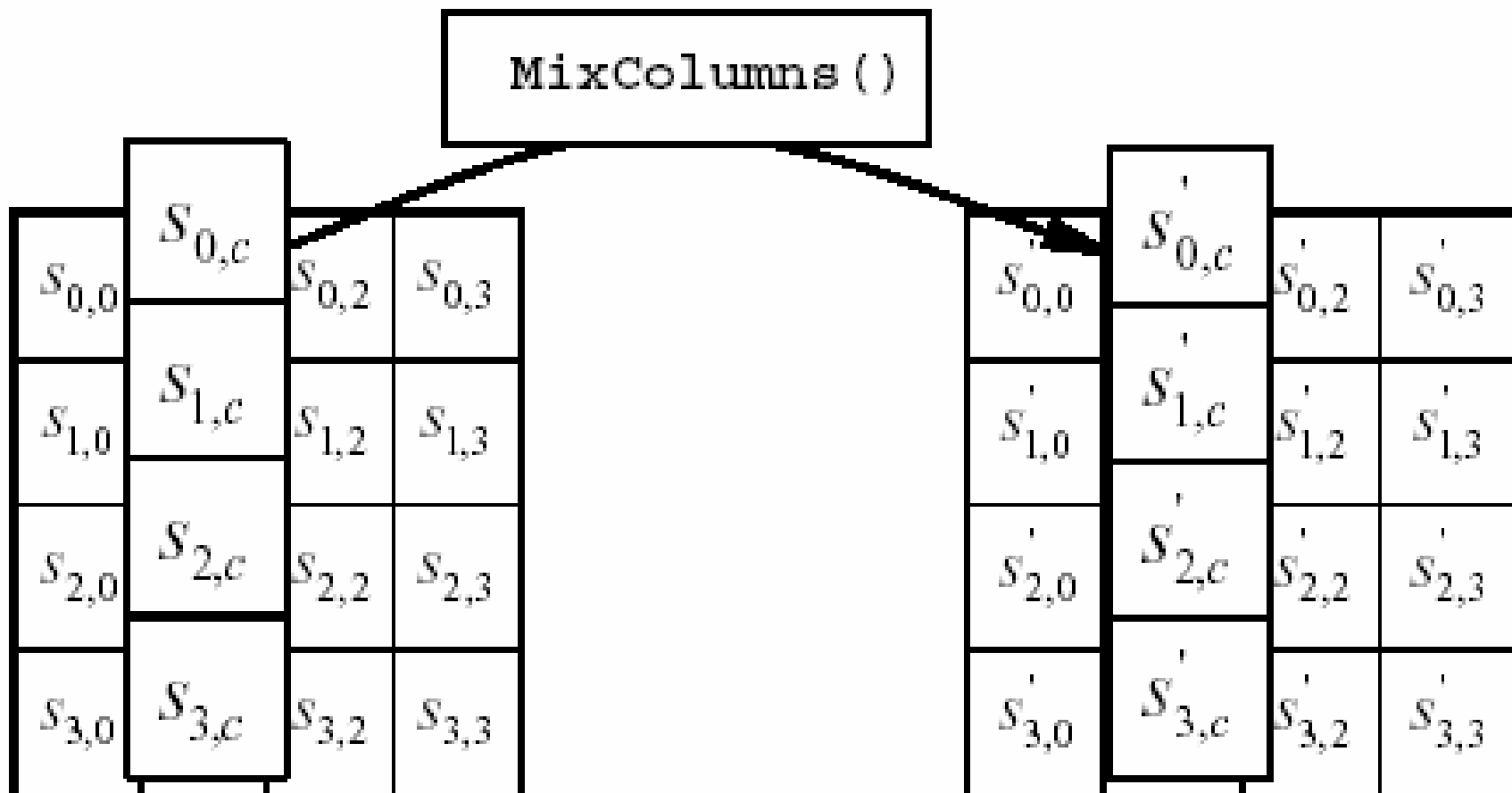
$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

MixColumns (3)

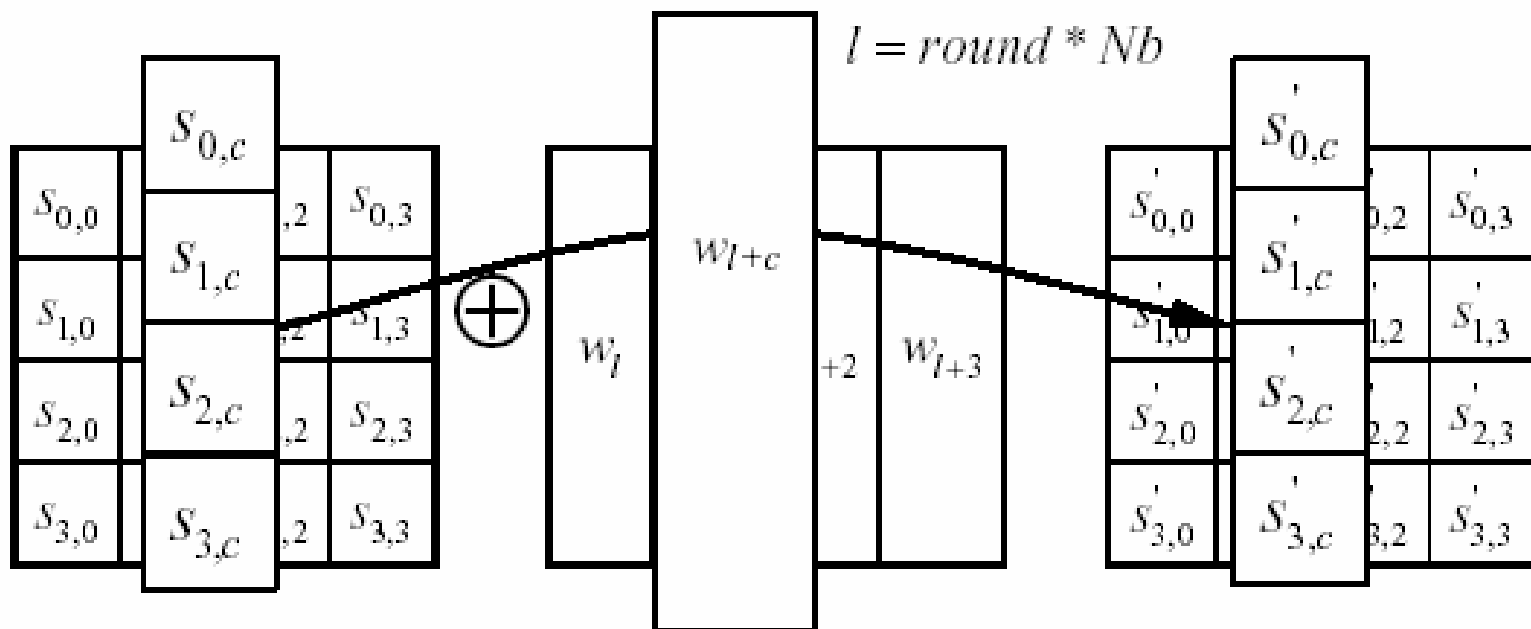


AddRoundKey (1)

- È l'unica funzione in cui interviene la chiave
- Quando viene invocata, riceve come parametri, oltre allo state, quattro word prese dall'array w (key schedule)
- La chiave corrente (round key) sarà aggiunta attraverso operazioni di xor su ogni bit di ogni byte

AddRoundKey (2)

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{\text{round} * \text{Nb} + c}] \text{ for } 0 \leq c < \text{Nb}$$



Riassumendo ...

- Cipher codifica blocchi da 128 bit
- Chiave a 128, 192 o 256 bit (3 versioni)
- Cifratura in diversi round in base alla lunghezza della chiave, applicando 4 funzioni: SubBytes, ShiftRows, MixColumns, AddRoundKey
- In ogni round la chiave è diversa



AES – Algoritmo di decifratura

Simone Forcella

Requisiti dell' algoritmo

- Anche l' algoritmo AES inverso, come il diretto, opera tra matrici (box).
- Partendo da un array quadrato d'ingresso, definisce una matrice di stato che verrà modificata nelle N_r iterazioni, per poi formare una matrice finale di output nella quale si troveranno i byte contenenti il testo in chiaro.
- Input/output diviso in blocchi da 128 bit
- Chiave da 128, 192 o 256 bit

Input – State – Output

input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

state

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

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}



$$S_{r,c} \leftarrow in_{r+4c}$$

$0 \leq r < 3$ $0 \leq c < Nb-1$

$$out_{r+4c} \leftarrow S_{r,c}$$

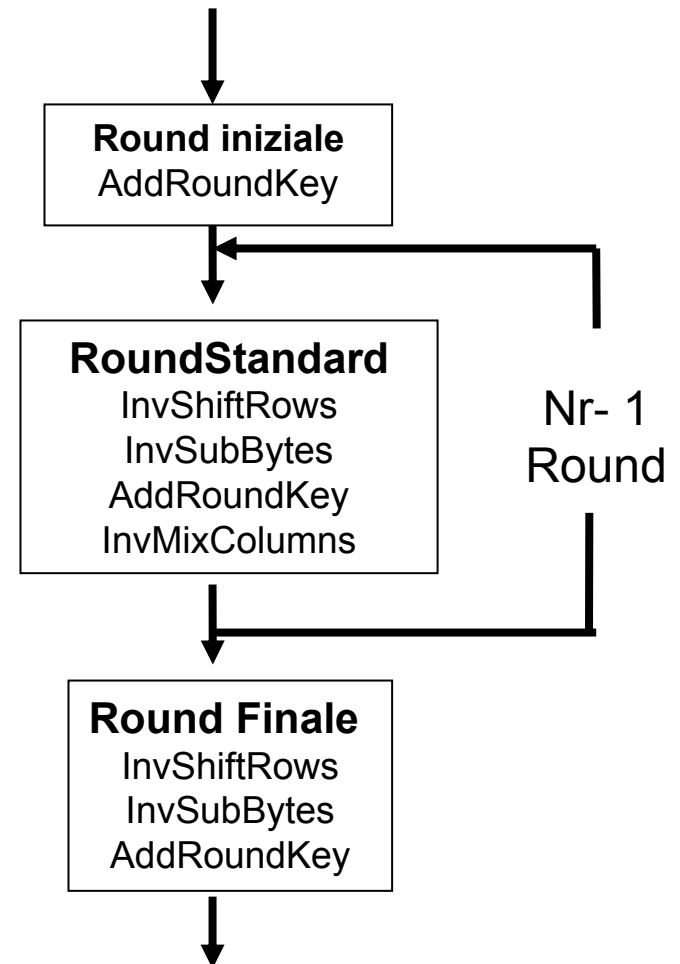
$0 \leq r < 3$ $0 \leq c < Nb-1$

Funzioni di InverseCipher

- L'algoritmo è composto da 4 funzioni:
 1. `InvShiftRows()`
 2. `InvSubBytes()`
 3. `InvMixColumns()`
 4. `AddRoundKey()`

InverseCipher

- I Round iniziale e finale come nel cipher sono 1 unico Round
- Il Round Standard viene ripetuto per $Nr - 1$ volte, anche in Inverse Cipher è necessario un ciclo



Pseudo-codice

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])

Begin

byte state[4,Nb]

state = in

AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

for round = Nr-1 step -1 downto 1

 InvShiftRows(state)

 InvSubBytes(state)

 AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

 InvMixColumns(state)

end for

InvShiftRows(state)

InvSubBytes(state)

AddRoundKey(state, w[0, Nb-1])

out = state

end

Input – State...

byte state[4,Nb] 

Definiamo un array quadrato (di tipo Byte) di 4 righe ed Nb colonne.

state = in 

L'input è la matrice di stato determinata dal testo in chiaro.

AddRoundKey

- La funzione AddRoundKey(), pone in XOR ogni colonna della matrice **STATE** con una WORD (colonna) della matrice rappresentante la **chiave** di round schedulata.
- AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb 1])
- $[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{\text{round} * \text{Nb} + c}]$

per $0 \leq c < \text{Nb}$

AddRoundKey

Matrice di STATO

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

Chiave

w_λ	$w_{\lambda+1}$	$w_{\lambda+2}$	$w_{\lambda+3}$

Matrice Risultato

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$

\oplus

$$\lambda = \text{round} * \text{Nb}$$

Ciclo

- AES cicla per il numero dei round indicato (Nr):

```
for round = Nr-1 step -1 downto 1
```

```
  InvShiftRows(state)
```

```
  InvSubBytes(state)
```

```
  AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
```

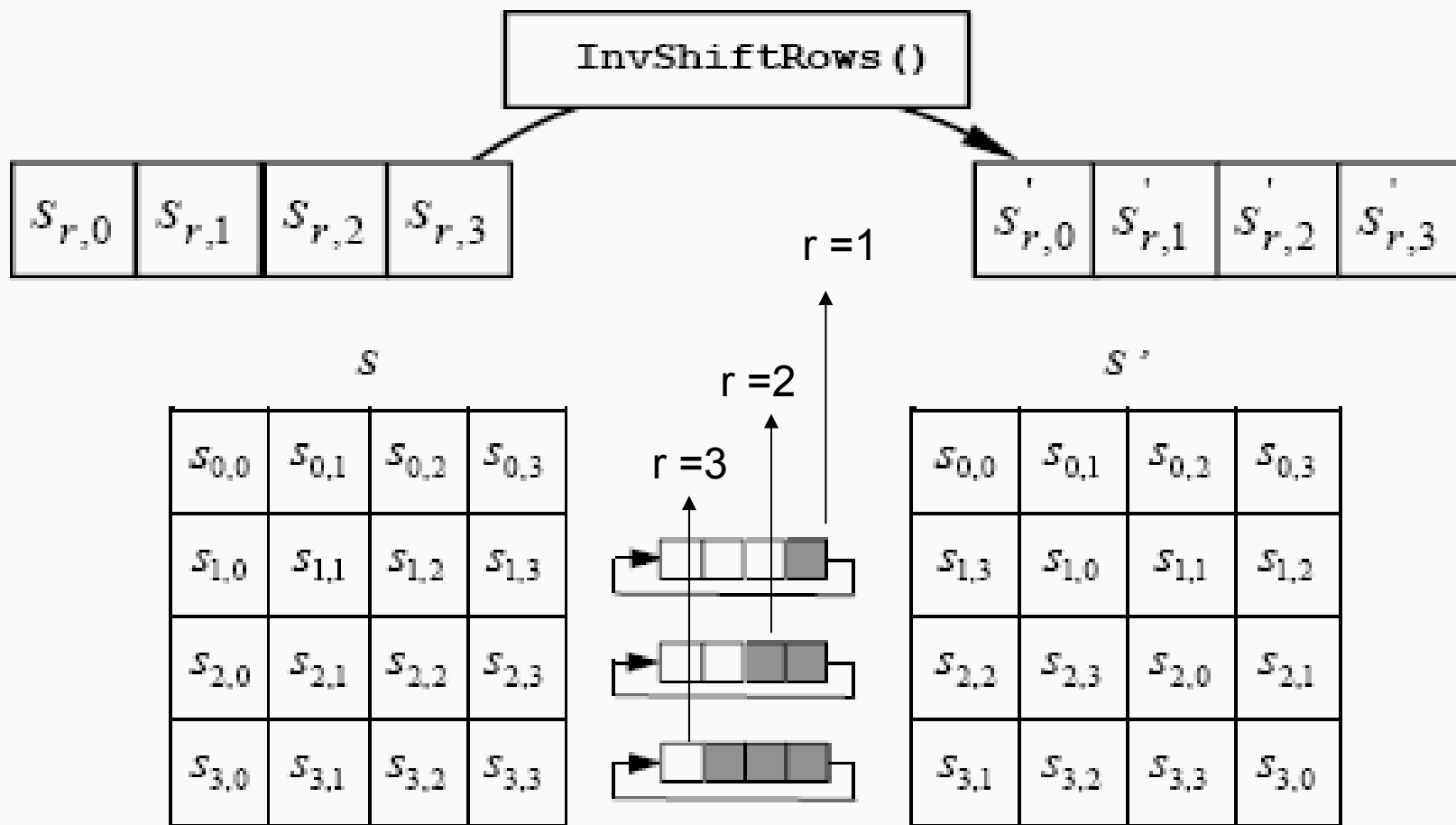
```
  InvMixColumns(state)
```

```
end for
```

InvShiftRows()

- I byte delle ultime 3 righe vengono spostati ciclicamente di un determinato offset (r)
- $S'_{r, (c + \text{shift}(r, Nb)) \bmod Nb} = S_{r, c}$ con $0 \leq c < Nb$ e $0 < r < 4$
- In **pratica** InvShiftRows() **inverte** il senso di shift della funzione ShiftRows() utilizzata nell'algoritmo cifrante

InvShiftRows()



InvSubBytes()

- Nella matrice di stato i byte sono rappresentati tramite valori in esadecimale
es. $s = '5a'$
- Come in SubBytes, anche InvSubBytes utilizza una tabella 16x16 di sostituzione, chiamata

Inverse_S Box

Inverse S-box

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

InvSubBytes()

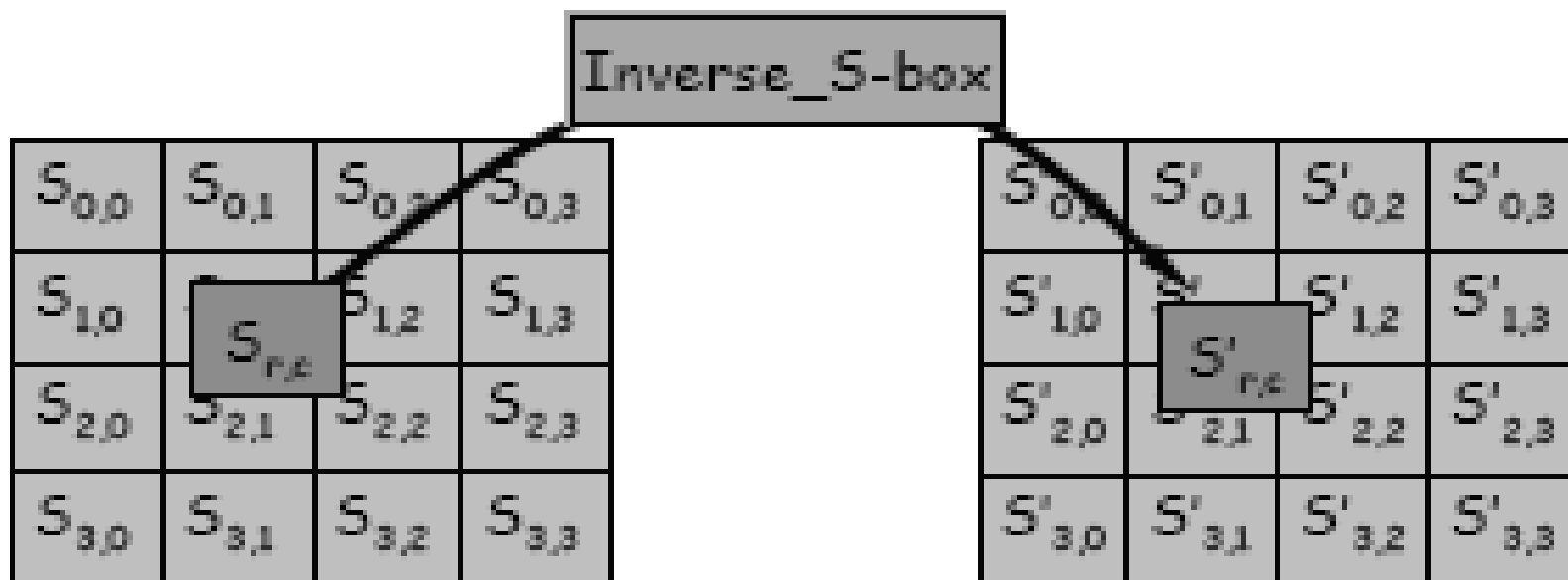
- Ogni elemento della S box è individuato da due coordinate X, Y
- X, Y sono le cifre del byte esadecimale nella 'STATE'
- Considerato il byte '5a', la X della S box ha valore '5', mentre la Y 'a'....

InvSubBytes()

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

InvSubBytes()

- Il “vecchio” byte ‘5a’ è ora identificato, nella stessa posizione dal valore, sempre esadecimale ‘46’



InvMixColumns()

- Come la funzione MixColumns() anche InvMixColumns opera sulle colonne
- Ricordo che ogni elemento della colonna è trattato come un polinomio in $GF(2^8)$
- Quindi il vettore colonna è un polinomio di 4° grado che ha per coefficienti i polinomi in $GF(2^8)$ rappresentanti il singolo byte

InvMixColumns()

- La funzione utilizza l'operazione di **ProdottoModulare**
 $s'(x) = a^{-1}(x) \otimes s(x)$
- Ovvero moltiplica il **vettore colonna** della matrice di stato per un **polinomio fissato** e considera in modulo x^4+1



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

InvMixColumns()

- Considerando le proprietà del prodotto modulare esposte in precedenza si può scrivere questo come prodotto matriciale:

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

for $0 \leq c < Nb$.

InvMixColumns()

$$a^{-1}(x) \otimes s(x)$$

Vettore colonna

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

InvMixColumns()

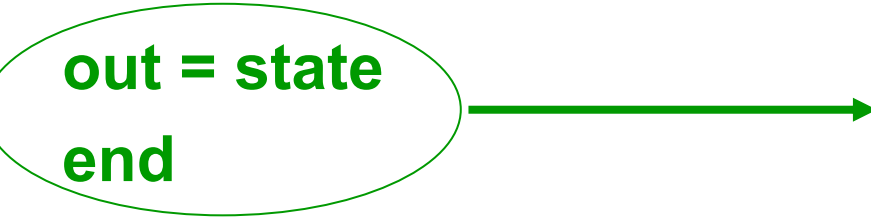
Vettore risultato

$S'_{0,0}$	$S'_{0,1}$	$S'_{0,2}$	$S'_{0,3}$
$S'_{1,0}$	$S'_{1,1}$	$S'_{1,2}$	$S'_{1,3}$
$S'_{2,0}$	$S'_{2,1}$	$S'_{2,2}$	$S'_{2,3}$
$S'_{3,0}$	$S'_{3,1}$	$S'_{3,2}$	$S'_{3,3}$

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

...State – Out

**out = state
end**



L' output finale è la matrice di stato determinata dalla fine dei round dell' algoritmo AES

Cifrario Inverso Equivalente

- AES inverso ha la possibilità e la proprietà di scambiare tra loro alcune funzioni:
 1. InvShiftRows() può essere scambiata con InvSubBytes() senza dover modificare altro nel codice, e senza peggiorarne le qualità

Cifrario Inverso Equivalente

2. Anche InvMixColumns() e AddRoundKey() possono essere invertite tra loro, ma questa volta è necessario modificare il codice .
 - E' necessario modificare il codice perché AddRoundKey() riceve come parametro la chiave schedulata differente in ogni round del ciclo.
 - Viene così implementato un nuovo array di word `dw[]`, contenente la nuova chiave di decifrazione schedulata

Codice del Cifrario Inverso Equivalente

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

  for round = Nr-1 step -1 downto 1
    InvSubBytes(state)
    InvShiftRows(state)
    InvMixColumns(state)
    AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
  end for

  InvSubBytes(state)
  InvShiftRows(state)
  AddRoundKey(state, dw[0, Nb-1])

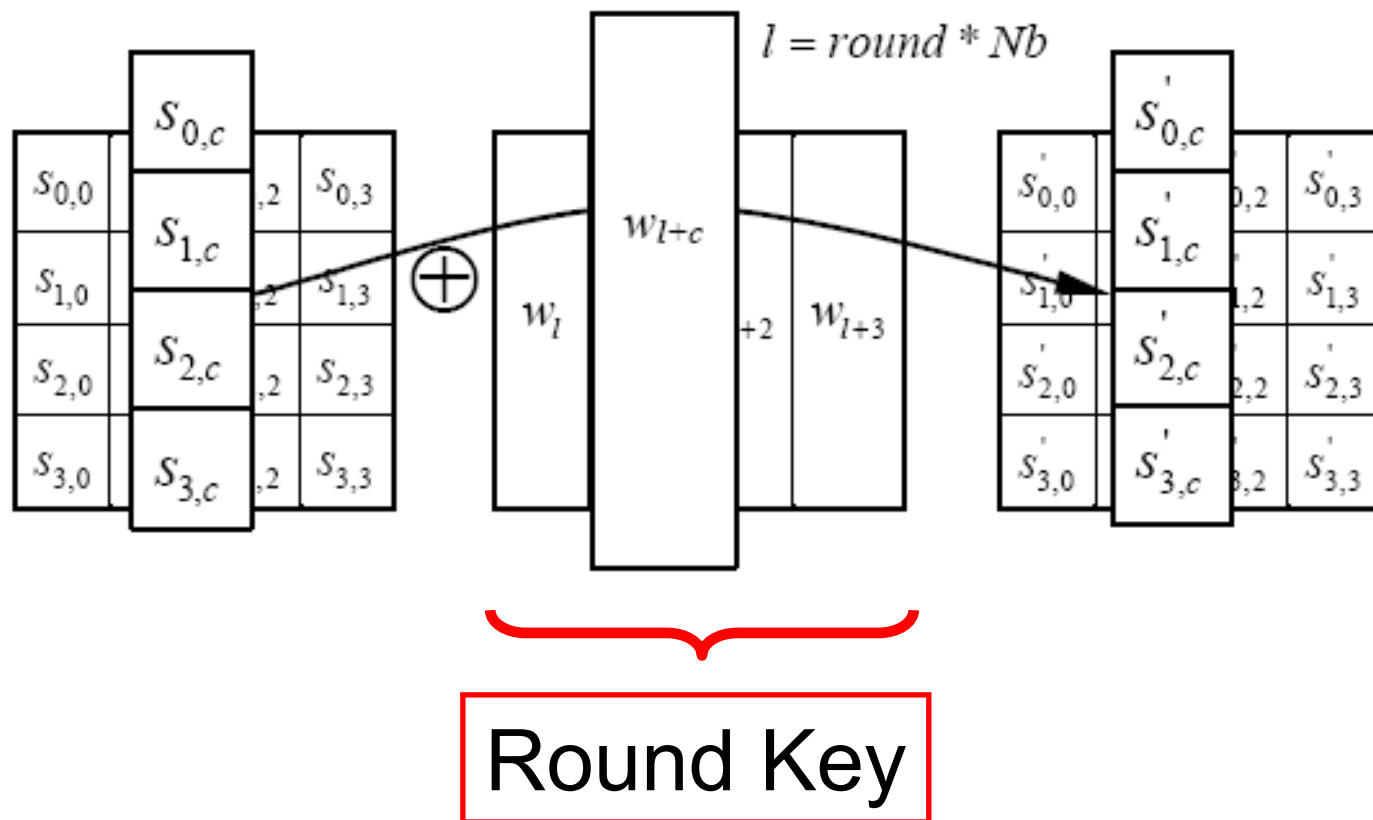
  out = state
end
```




AES – Key Schedule

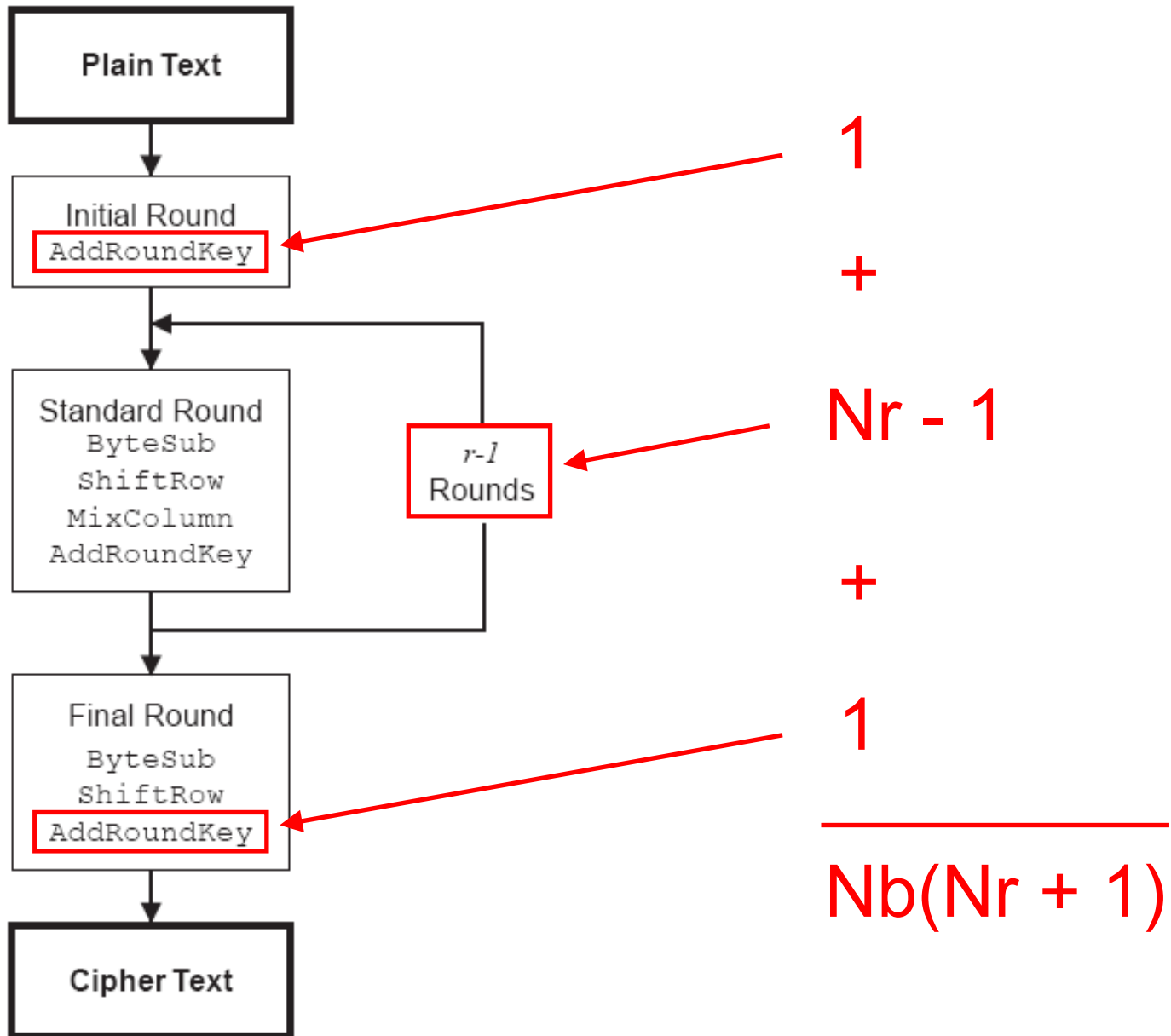
Francesco Forconi

AddRoundKey



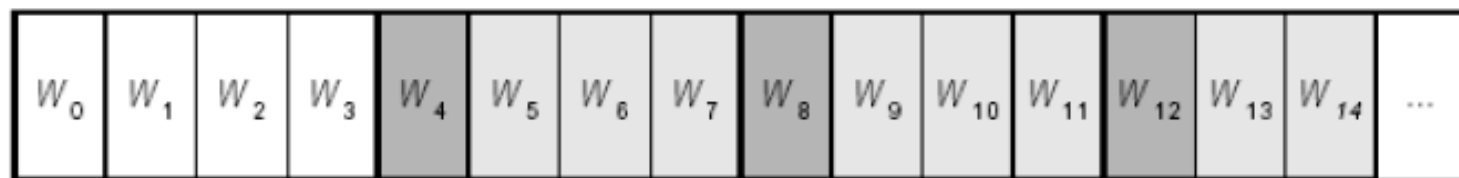
Round Keys

- Ottenute dalla chiave di cifratura tramite la “key expansion”
- La key expansion produce l’expanded key che contiene tutte le round key che verranno utilizzate dall’algoritmo
- L’expanded key è un array lineare (W) di $N_b(N_r+1)$ parole di 4 byte



Selezione della round key

- La i -esima round key è data dalle parole che vanno da $W[Nb*i]$ a $W[Nb*(i+1)]$
- In caso di $Nb = 6$ e $Nk = 4$



```

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp

    i = 0

    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while

    i = Nk

    while (i < Nb * (Nr+1))
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end

```

$$W[i] = W[i - Nk] \oplus \text{temp}$$

■ Se $i \bmod Nk = 0$

□ $\text{temp} = \text{SubWord}(\text{RotWord}(W[i - 1])) \oplus$
 $\text{Rcon}[i / Nk]$

■ Se $i \bmod Nk \neq 0$

□ $\text{temp} = W[i - 1]$

■ Se $Nk = 8$ e $i \bmod Nk = 4$

□ $\text{temp} = \text{SubWord}(W[i - 1])$

Key Expansion

- SubWord: restituisce una parola ottenuta applicando la S_box a tutti i 4 byte della parola di input
- RotWord: data la parola $[a_0, a_1, a_2, a_3]$ restituisce la parola $[a_1, a_2, a_3, a_0]$
- Rcon[i]: $[x^{(i-1)}, \{00\}, \{00\}, \{00\}]$ dove x è un elemento di $GF(2^8)$, $\{02\}$ in esadecimale
- Rcon[1] = $\{01\}$

Esempio

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
W[0] W[1] W[2] W[3]

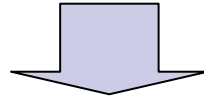
$i = 4 \longrightarrow$ $W[4] = W[0] \oplus \text{temp}$
 $\text{temp} = \text{SW}(\text{RW}(W[3])) \oplus \text{Rcon}[4/4]$

$\text{RotWord}(W[3]) \longrightarrow [cf\ 4f\ 3c\ 09]$

$\text{SubWord}([cf\ 4f\ 3c\ 09]) \longrightarrow [8a\ 84\ eb\ 01]$

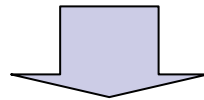
$\text{Rcon}[1] \longrightarrow [01\ 00\ 00\ 00]$

$$[8a\ 84\ eb\ 01] \oplus [01\ 00\ 00\ 00]$$



$$\begin{array}{cccc} 10001010 & 10000100 & 11101011 & 00000001 \\ 00000001 & 00000000 & 00000000 & 00000000 \\ \hline 10001011 & 10000100 & 11101011 & 00000001 \end{array}$$

$$[2b\ 7e\ 15\ 16] \oplus [8b\ 84\ eb\ 01]$$



$$\begin{array}{cccc} 00101011 & 01111110 & 00010101 & 00010110 \\ 10001011 & 10000100 & 11101011 & 00000001 \\ \hline 10100000 & 11111010 & 11111110 & 00010111 \end{array}$$

$W[4] = [a0\ fa\ fe\ 17]$

Key expansion: criteri progettuali

- Per resistere ad attacchi di vario tipo:
 - In cui si conosce una parte della chiave
 - In cui la chiave è conosciuta o può essere scelta
 - Attacchi related-key
- Gioca un ruolo fondamentale nell'eliminazione della simmetria introdotta dai round



AES – Sicurezza e attacchi

Francesco Forconi

Situazione attuale

- Standard utilizzato dal governo USA per documenti SECRET (128 bit) e TOP SECRET (192 o 256 bit)
- Ricerca esaustiva impraticabile:
 $3,4 * 10^{38}$ possibilità con chiave a 128 bit
- Nessun attacco andato a buon fine
- Non sono note chiavi deboli o semi deboli

Attacchi parziali

- Attacchi con successo a implementazioni di AES con meno round:
 - 2000 – attacco Square: 7 round su chiave a 128 e 9 round su chiave 256
 - 2003 – attacco di crittanalisi differenziale basata sulle chiavi: 8 round su chiave a 192 bit

Attacco Square

- Definito per l'algoritmo Square
- Attacco di tipo chosen plaintext
- Sfrutta la struttura orientata al byte di Rijndael
- Segue l'evoluzione della posizione di un byte attivo per eliminare possibili valori della chiave
- Necessita di 2^{32} plain text ed è comunque parziale

Attacco XSL

- Attacco di tipo algebrico
- Rappresenta Rijndael come un sistema di equazioni algebriche
- Complessità 2^{100}
- Attualmente non implementabile