4 Advanced Encryption Standard

4.1 Cifrari Iterati

Analizziamo ora una particolare classe di cifrari a blocchi: i **cifrari iterati**. Per costruire un cifrario iterato è necessario specificare una **funzione round** e una **programmazione delle chiavi**. La cifratura di un testo in chiaro avviene attraverso un numero finito N_r di round (o iterazioni). Fissata una chiave arbitraria K, rappresentata da una stringa binaria di lunghezza fissata, attraverso un algoritmo pubblico viene generata una sequenza $(K^1, \ldots, K^{N_r})$ di sottochiavi dette **chiavi round**.

La **funzione round** g ha due input: lo **stato** che denotiamo con w^{r-1} e la **chiave round**. Lo stato successivo è definito come $w^r = g(w^{r-1}, K^r)$.

Lo stato iniziale w^0 è il testo in chiaro x, il testo cifrato y corrispondente ad x è lo stato dopo che tutte le N_r iterazioni sono state eseguite. La cifratura avviene come segue:

$$\begin{aligned} w^0 &\leftarrow x \\ w^1 &\leftarrow g(w^0, K^1) \\ w^2 &\leftarrow g(w^1, K^2) \\ &\vdots \\ w^{N_r-1} &\leftarrow g(w^{N_r-2}, K^{N_r-1}) \\ w^{N_r} &\leftarrow g(w^{N_r-1}, K^{N_r}) \\ y &\leftarrow w^{N_r}. \end{aligned}$$

Affinchè la decifratura sia possibile, la funzione deve essere iniettiva rispetto alla prima variabile una volta che la seconda è fissata. Ciò equivale a dire che esiste una funzione g^{-1} con la proprietà $g^{-1}(g(w, y), y) = w$ per tutti i w e y.

La decifratura avviene come segue:

$$\begin{aligned} w^{N_r} &\leftarrow y \\ w^{N_r-1} &\leftarrow g^{-1}(w^{N_r}, K^{N_r}) \\ &\vdots \\ w^1 &\leftarrow g^{-1}(w^2, K^2) \\ w^0 &\leftarrow g^{-1}(w^1, K^1) \\ x &\leftarrow w^0. \end{aligned}$$

4.2 Substitution-Permutation Network

Nei cifrari appena presentati, le funzioni di cifratura e_K sono legate unicamente alle chiavi, perciò queste ultime sono variabili.
In seguito, analizzeremo invece crittosistemi iterati, tali che alcuni dei singoli cifrari presenti al loro interno risultano essere **fissati**, e pertanto non dipendenti più dalle chiavi. Il primo ad essere proposto è il **Substitution-Permutation Network (SPN)**.
Poichè si tratta di un crittosistema moderno, e quindi utilizzato principalmente da calcolatori (che lavorano in binario), l'alfabeto è $\mathbb{Z}_2$.
Analizziamo ora, in maniera pù dettagliata, la struttura del crittosistema SPN.
L'insieme dei messaggi in chiaro così come quelli cifrati è l'insieme delle stringhe binarie di lunghezza ℓm, dove ℓ ed m sono interi positivi fissati.
L'insieme delle chiavi è

$$\mathcal{K} = \{(K^1, \dots, K^{N_r+1})\} : K^i \in \{0,1\}^{\ell m}, 1 \leq i \leq N_r + 1\}.$$

Una qualsiasi chiave $(K^1, \dots, K^{N_r+1})$ è derivata da una chiave iniziale K, secondo un opportuno algoritmo pubblico.
La funzione di cifratura si basa su due permutazioni $\pi_S \in Sym(\{0,1\}^\ell)$ detta **S-box** (scatola di sostituzione) e $\pi_P \in Sym(\ell m)$.

Sia $x = (x_1, \dots, x_{\ell m})$ un messaggio in chiaro, allora x può essere riguardata come la concatenazione di m sottostringhe binarie denotate con $x_{<i>}$. Quindi

$$x = x_{<1>} || \dots || x_{<m>},$$

dove $x_{<i>} = (x_{(i-1)\ell+1}, \dots, x_{i\ell})$ per ogni $\leq i \leq m$.
La cifratura di x avviene secondo il seguente algoritmo:

Algoritmo 4.1. **(SPN** $(x, \pi_S, \pi_P, (K^1, ..., K^{N_r+1}))$**)**

$w^0 \leftarrow x$
for $r \leftarrow 1$ **to** $N_r - 1$
do
- $u^r \leftarrow w^{r-1} \oplus K^r$
- **for** $i \leftarrow 1$ **to** m
  - **do** $v^r_{(i)} \leftarrow \pi_S(u^r_{(i)})$
- $w^r \leftarrow (v^r_{\pi_{P(1)}}, ..., v^r_{\pi_{P(\ell m)}})$

$u^{N_r} \leftarrow w^{N_r-1} \oplus K^{N_r}$
for $i \leftarrow 1$ **to** m
do $v^{N_r}_{(i)} \leftarrow \pi_S(u^{N_r}_{(i)})$
$y \leftarrow v^{N_r} \oplus K^{N_r+1}$
output (y)

Come si evince dall'**Algoritmo 4.1**, l'SPN consiste di N_r iterazioni. In ciascuna di queste, tranne l'ultima, sulle m sottostringhe viene applicata prima una permutazione π_S e successivamente queste vengono permutate attraverso π_P.
Nella prima iterazione si calcola $u^1 = x \oplus K^1$ (questo procedimento è detto **sbiancamento**). Successivamente viene applicata π_S alle m sottostringhe di $u^1 = u^1_{<1>} \| \dots \| u^1_{<m>}$ ottenendo così $v^1 = v^1_{<1>} \| \dots \| v^1_{<m>}$ ($v^1_{<i>} = \pi_S(u^1_{<i>})$).
La prima iterazione termina con l'applicazione di π_P sulle ℓm posizioni di v^1 generando così $w^1 = v^1_{\pi_P(1)} \cdots v^1_{\pi_P(m)}$.
La seconda iterazione è analoga alla prima con $u^2 = w^1 \oplus K^2$ al posto di $u^1 = x \oplus K^1$, dove w^1 è la stringa generata nella prima iterazione.
Dopo $N_r - 1$ iterazioni viene così generato $u^{N_r} = w^{N_r - 1} \oplus K^{N_r}$. Applicando π_S a u^{N_r}, si ha $v^{N_r} = \pi_S(u^{N_r}_{<1>}) \| \dots \| \pi_S(u^{N_r}_{<m>})$. Infine l'ultima operazione dell'algoritmo è $y = v^{N_r} \oplus K^{N_r+1}$ (**sbiancamento**).

L'SPN ha diverse caratteristiche interessanti, il design è semplice ed estremamente efficiente sia nell'hardware che nel software. Per quanto riguarda il software, π_S viene implementata generalmente come una tabella dati (risulta più veloce consultare la tabella piuttosto che calcolare ogni volta il risultato di π_S applicato ad una stringa). Poichè $\pi_S \in Sym(\{0,1\}^\ell)$, la memoria da immagazzinare è di circa $\ell \cdot 2^\ell$ bit perchè bisogna immagazzinare 2^ℓ stringhe binarie ognuna di lunghezza ℓ. L'implementazione hardware, invece, necessita di avere π_S relativamente piccolo.
Di seguito viene riportata una versione esemplificata del crittosistema SPN in cui $\ell = m = N_r = 4$. Quindi, $\mathcal{P} = \mathcal{C} = \{0,1\}^{16}$. Le permutazioni $\pi_P \in Sym(16)$ e $\pi_S \in Sym(\{0,1\}^4)$ sono definite come segue:

$$\pi_P = (2\ 5)(3\ 9)(4\ 13)(7\ 10)(8\ 14)(12\ 15).$$

x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	0
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1
1	0	1	0	0	1	1	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	1	0	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	1	1	1

Tabella 4: S-Box.

Per completare l'esempio, è necessario definire un algoritmo che generi la sequenza delle sottochiavi a partire da una chiave iniziale K. Per fare ciò, un possibile algoritmo è quello che da come output la sequenza $(K^1, K^2, K^3, K^4, K^5)$ (si ricordi che $N_r = 4$), dove

$$K^i = (k_{4i-3}, k_{4i-2}, \ldots, k_{4i+12}).$$

Sia $K = 11010010000101111011101000111100$. Quindi le 5 sottochiavi sono

$$\begin{aligned} K^1 &= 1101\,0010\,0001\,0111 \\ K^2 &= 0010\,0001\,0111\,1011 \\ K^3 &= 0001\,0111\,1011\,1010 \\ K^4 &= 0111\,1011\,1010\,0011 \\ K^5 &= 1011\,1010\,0011\,1100 \end{aligned}$$

Supponiamo che il testo in chiaro da cifrare sia:

$$\mathbf{x{=}1001111000010001}.$$

Allora la cifratura di x procede come segue:

$$\begin{aligned} w^0 &= (1001)(1110)\ (0001)(0001) && \oplus \\ K^1 &= (1101)(0010)\ (0001)(0111) && = \\ u^1 &= (0100)(1100)\ (0000)(0110) && \swarrow \pi_S \\ v^1 &= (0010)(0101)\ (1110)(1011) && \swarrow \pi_P \\ w^1 &= (0011)(0110)\ (1011)(0101) && \oplus \\ K^2 &= (0010)(0001)\ (0111)(1011) && = \\ u^2 &= (0001)(0111)\ (1100)(1110) && \swarrow \pi_S \\ v^2 &= (0100)(1000)\ (0101)(0000) && \swarrow \pi_P \\ w^2 &= (0100)(1010)\ (0000)(0010) && \oplus \\ K^3 &= (0001)(0111)\ (1011)(1010) && = \\ u^3 &= (0101)(1101)\ (1011)(1000) && \swarrow \pi_S \\ v^3 &= (1111)(1001)\ (1100)(0011) && \swarrow \pi_P \\ w^3 &= (1110)(1010)\ (1001)(1101) && \oplus \\ K^4 &= (0111)(1011)\ (1010)(0011) && = \\ u^4 &= (1001)(0001)\ (0011)(1110) && \swarrow \pi_S \\ v^4 &= (1010)(0100)\ (0001)(0000) && \oplus \\ K^5 &= (1011)(1010)\ (0011)(1100) && = \\ y &= (0001)(1010)\ (0010)(1100) \end{aligned}$$

Pertanto **x=1001111000010001** viene cifrato con **y=0001101000101100**.

Dall'analisi della struttura del SPN si evince che sebbene le operazioni di sostituzione e di permutazione siano indipendenti dalla chiave, mentre l'iterazione non lo è.

Nell'esempio che abbiamo utilizzato, ad ogni iterazione la memoria richiesta era di 2^6 bit ($\ell = 4$). Si noti che per $\ell = 16$ la memoria occupata crescerebbe fino a 2^{20} diminuendo così l'efficienza del crittosistema. Lo scopo dell'esempio utilizzato è puramente illustrativo poichè la chiave iniziale consiste di 32 bit e quindi il sistema può essere violato attraverso la forza bruta.

Pertanto, nella realtà, per ottenere cifrari SPN sicuri, bisogna aumentare la lunghezza della chiave, la grandezza degli **S-box** e inoltre un maggior numero di iterazioni.

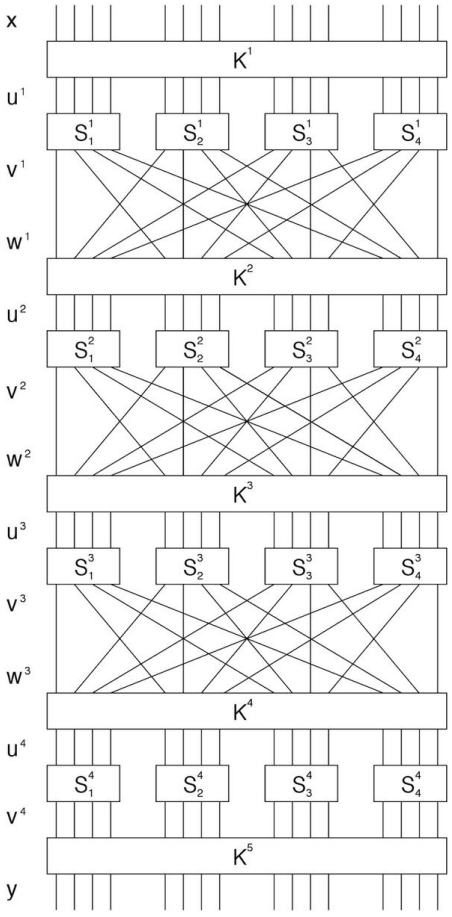

Figura 4.1: Struttura di un cifrario SPN

4.3 Advanced Encryption Standard

Nel 1997 il National Insitute of Standards and Technology (NIST) istituì una competizione per la creazione di un **crittosistema simmetrico** per proteggere le informazioni federali sensibili.

I cifrari proposti dai candidati dovevano assicurare requisiti minimi che erano:

- **Lunghezza del blocco** pari a 128 bit.
- **Lunghezza della chiave iniziale** di 128, 192 o 256 bit.

I criteri di valutazione erano i seguenti:

- **Sicurezza**: È il criterio principale di valutazione. Esso comprende resistenza dell'algoritmo alla crittanalisi, solidità delle basi matematiche, casualità dell'output dell'algoritmo e maggiore sicurezza rispetto ad altri candidati.
- **Costo**: Altro importante criterio, riguarda l'efficienza computazionale (velocità) su varie piattaforme, requisiti di licenza e di memoria.
- **Caratteristiche algoritmiche e di implementazione**: Flessibilità, semplicità algoritmica e adattabilità software e hardware.

La competizione fu vinta nel 2001 da due ricercatori belgi Vincent Rijmen e Joan Daemen con il crittosistema da essi inventato detto **Rijndael** (dalle loro iniziali) che successivamente venne denominato **Advanced Encryption Standard (AES)**.

Figura 4.2: Vincent Rijmen (1970) e Joan Daemen (1965)

4.3.1 Conversione Binario-Esadecimale-Elemento di $GF(2^8)$

Le unità di messaggio in chiaro e cifrato del cifrario AES sono costituite da 16 byte. Per una migliore comprensione delle operazioni utilizzate dal suddetto cifrario è utile convertire i byte sia in coppie di cifre esadecimali sia come elementi del campo finito $GF(2^8)$. La conversione da binario a esadecimale è fornita della seguente tabella.

BINARIO	ESA	BINARIO	ESA
(0000)	0	(1000)	8
(0001)	1	(1001)	9
(0010)	2	(1010)	A
(0011)	3	(1011)	B
(0100)	4	(1100)	C
(0101)	5	(1101)	D
(0110)	6	(1110)	E
(0111)	7	(1111)	F

Tabella 5: Conversione da binario a esadecimale.

Attraverso la **Tabella 5** è facile vedere che, ad esempio, la conversione esadecimale del byte 1001, 1110 è $9E$.

Prima di fornire la conversione dei byte in elementi di $GF(2^8)$, diamo una breve descrizione di quest'ultimo.

Si consideri il polinomio $p(x) = x^8 + x^4 + x^3 + x + 1$ a coefficienti in $GF(2)$. Poichè $p(x)$ è irriducibile su $GF(2)$, l'ideale $\langle p(x) \rangle$ è massimale in $GF(2)[x]$ e quindi $\frac{GF(2)[x]}{\langle p(x) \rangle}$ è una copia isomorfa del campo $GF(2^8)$.

Pertanto, gli elementi di $GF(2^8)$ sono descrivibili nella forma $\langle p(x) \rangle + a(x)$ con $a(x) \in GF(2)[x]$. Utilizzando la divisione tra polinomi è facile vedere che ogni laterale $\langle p(x) \rangle + a(x)$ ha un unico rappresentante di grado compreso tra 0 e 7, corrispondente al resto della divisione del polinomio $a(x)$ con $p(x)$.

Quindi,

$$GF(2^8) = \{b_7 x^7 + b_6 x^6 + \cdots + b_1 x + b_0,\ b_7, \ldots, b_0 \in GF(2)\}$$

in cui la somma tra due rappresentanti $b(x)$ e $b'(x)$ coincide con quella usuale in $GF(2)[x]$. Il prodotto $b(x)b'(x)$ è uguale a $r(x)$, dove $r(x)$ è il polinomio resto della divisione di $b(x)b'(x)$ per $p(x)$. In particolare, il rappresentante dell'inverso del polinomio $b(x)$ si calcola attraverso il ben noto algoritmo euclideo.

La conversione dei byte in elementi di $GF(2^8)$ avviene come segue.
Sia $b = b_7b_6b_5b_4b_3b_2b_1b_0$ il generico byte, ad esso viene associato il polinomio

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

e quindi l'elemento del campo $GF(2^8)$ individuato da quest'ultimo.
Pertanto, per esempio, alla stringa binaria 10011110 viene fatto corrispondere l'elemento del campo individuato dal polinomio $b(x) = x^7 + x^4 + x^3 + x^2 + x$.

Per il calcolo dell'inverso di $b(x)$ in $GF(2^8)$ si applica l'algoritmo euclideo:

1. $p(x) = b(x)x + (x^5 + x^2 + x + 1)$.
2. $b(x) = (x^5 + x^2 + x + 1)x^2 + x$.
3. $(x^5 + x^2 + x + 1) = x(x^4 + x + 1) + 1$.

Quindi,

$$1 = p(x)(x^6 + x^3 + x^2 + 1) + b(x)(x^7 + x^3 + 1).$$

Pertanto l'inverso di $b(x)$ *mod* $p(x)$ è $x^7 + x^3 + 1$, che corrisponde alla stringa binaria 10001001.

Per velocizzare la procedura di calcolo dell'inverso in $GF(2^8)$, il calcolatore si avvale della seguente tabella.

		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
X	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Tabella 6: Tabella degli inversi in $GF(2^8)$.

Infatti l'inverso di $b = 10011110 = 9E$ è dato dall'elemento nella riga 9 e colonna E della suddetta tabella che è 89 = 10001001 (che corrisponde a = $x^7 + x^3 + 1$).

Si noti che la procedura è coerente con il calcolo in $GF(2^8)$ effettuato prima. In realtà, il calcolatore utilizza l'equivalente binario della **Tabella 6**, bypassando la conversione in esadecimale e velocizzando ulteriormente la procedura per il calcolo degli inversi in $GF(2^8)$.

4.4 Struttura

Il cifrario AES è un particolare cifrario iterato in cui i messaggi in chiaro, così come quelli cifrati, sono stringhe binarie di lunghezza 128, ovvero blocchi binari di 4×4 byte.

Le chiavi hanno tre possibili lunghezze 128, 192 o 256 bit. Il numero N_r di iterazioni è 10, 12 o 14 a seconda che le chiavi abbiano lunghezza 128, 192 o 256 bit, rispettivamente. Le sottochiavi $(K_0, K_1, \ldots, K_{N_r})$ vengono calcolate attraverso un algoritmo pubblico che ha come input una chiave iniziale segreta K. La lunghezza di quest'ultima quindi determina sia il numero N_r di iterazioni che la lunghezza delle sottochiavi utilizzate in ognuna di esse. L'output di ogni iterazione è detto **stato**. Ogni iterazione, eccetto l'ultima, consiste di quattro operazioni:

1. **SubBytes**: Sostituzione attraverso l'S-box, ovvero si applica un opportuno elemento di $Sym\left(\{0,1\}^8\right)$.
2. **ShiftRows**: Permutazione sulle righe dello stato.
3. **MixColumns**: Trasformazione lineare sui blocchi di 4 byte.
4. **Addroundkey**: Operazione di XOR tra una sottochiave e lo stato.

Nell'ultima iterazione, l'operazione **MixColumn** non viene effettuata.

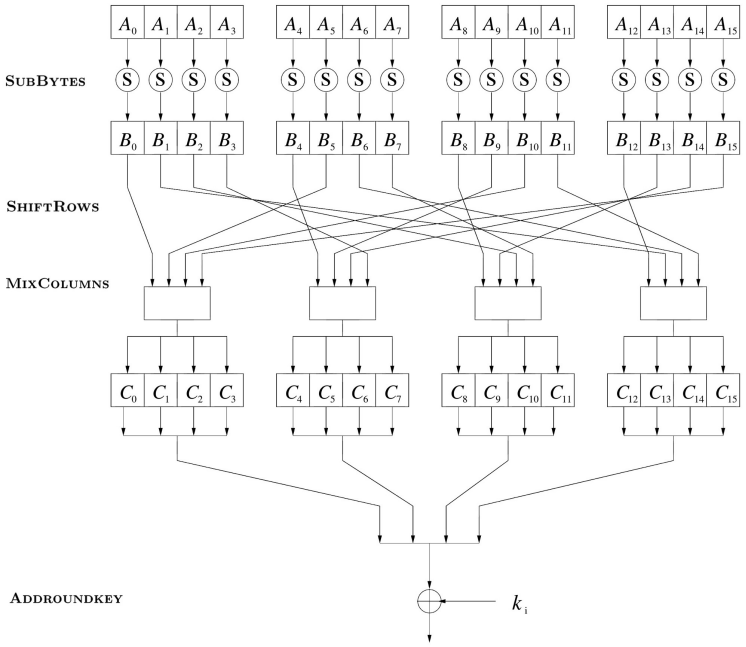

Figura 4.3: i-esima funzione round dell'AES, per $i = 1, \ldots, N_r - 1$

Esistono forti analogie tra la struttura del SPN e quella dell'AES: entrambi i crittosistemi presentano una combinazione di sottochiavi, una sostituzione ed una permutazione. Inoltre, contengono l'operazione di **sbiancamento**. La principale differenza tra questi due cifrari risiede nella presenza, per l'AES, della trasformazione lineare **MixColumn**.

Le variabili dell'AES, ovvero gli stati, sono rappresentate da una matrice 4×4 di byte:

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

Lo stato iniziale w^0, che corrisponde al testo in chiaro $x = (x_0, x_1, \ldots, x_{15})$, è

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

Analizziamo i singoli livelli dell'AES[1].

4.4.1 SubBytes

L'operazione **SubBytes** è una particolare sostituzione che opera in modo indipendente su tutti i byte dello stato applicando una **S-box** su ognuno di essi. In particolare, si tratta di una permutazione di elementi di $\{0,1\}^8$ che opera come segue: fissata una generica stringa binaria non nulla $\alpha_7\alpha_6\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1\alpha_0$ si determina l'inversa $a_7a_6a_5a_4a_3a_2a_1a_0$ in $GF(2^8)$, secondo la conversione vista precedentemente, e successivamente si calcola la stringa binaria $b_7b_6b_5b_4b_3b_2b_1b_0$ attraverso una trasformazione affine definita da

$$b_i = (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \quad 0 \leq i \leq 7$$

dove $c_7c_6c_5c_4c_3c_2c_1c_0 = 01100011$.

[1] In seguito, per rappresentare il contenuto di un byte spesso, per semplicità, si utilizzerà la notazione esadecimale.

La rappresentazione matriciale della trasformazione affine è la seguente

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \ mod\ 2 \qquad (*)$$

Più precisamente, l'algoritmo **SubBytes** opera come segue:

Algoritmo 4.2. **(SubBytes** $(a_7a_6a_5a_4a_3a_2a_1a_0)$**)**

external FieldInv, BinaryToField, FieldToBinary
$z \leftarrow$ BinaryToField$(a_7a_6a_5a_4a_3a_2a_1a_0)$
if $z \neq 0$
    $(a_7a_6a_5a_4a_3a_2a_1a_0) \leftarrow$ FieldToBinary(z)
then $z \leftarrow$ FieldInv(z)
    $(c_7c_6c_5c_4c_3c_2c_1c_0) \leftarrow (01100011)$
for $i \leftarrow 0$ **to** 7
do $b_i \leftarrow (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i)\ mod\ 2$
return $(b_7b_6b_5b_4b_3b_2b_1b_0)$

L'algoritmo esterno **FieldInv** è quello che permette di calcolare l'inverso dell'elemento in $GF(2^8)$ corrispondente alla stringa $a_7a_6a_5a_4a_3a_2a_1a_0$.

Un modo più immediato per determinare la stringa

$$b_7b_6b_5b_4b_3b_2b_1b_0$$

a partire dalla stringa

$$\alpha_7\alpha_6\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1\alpha_0$$

è l'utilizzo della **Tabella 7** che racchiude le operazioni di calcolo dell'inverso in $GF(2^8)$ e della trasformazione affine $(*)$.

		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	BE	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

Tabella 7: AES S-box.

Supponiamo di voler cifrare il testo in chiaro x (convertito nel sistema esadecimale):

B95EF2E0E302E88C1D61A0B8AC82B65C.

Prima di applicare la **SubBytes** nella prima iterazione, al testo in chiaro viene inizialmente eseguita una somma XOR con la chiave K data in input, che supponiamo essere

274FE51AB701794D3ADF95B90F6C8BEE,

generando così la stringa

9E1117FA540391C127BE3501A3EE3DB2.

Terminata questa prima operazione, si esegue ora la sostituzione.

La **Tabella 7** viene applicata per ogni coppia XY di cifre esadecimali. L'immagine tramite l'S-box di XY è l'elemento della tabella posizionato nella riga X e colonna Y. Pertanto, nel caso della prima coppia di cifre di x partendo da sinistra, cioè di 9E, la tabella restituisce l'elemento 0B.

Procedendo allo stesso modo con le altre coppie del testo in chiaro, si genera così la stringa:

0B82F02D207B8178CCAE967C0A282737.

Vediamo in dettaglio come opera l'S-box.

Disponendo il testo in chiaro lungo le colonne di una matrice 4×4, il blocco corrispondente al messaggio **x** è

$9E$	54	27	$A3$
11	03	BE	EE
17	91	35	$3D$
FA	$C1$	01	$B2$

(4.1)

L'equivalente in binario è

(1001)(1110)	(0101)(0100)	(0010)(0111)	(1010)(0011)
(0001)(0001)	(0000)(0011)	(1011)(1110)	(1110)(1110)
(0001)(0111)	(1001)(0001)	(0011)(0101)	(0011)(1101)
(1111)(1010)	(1100)(0001)	(0000)(0001)	(1011)(0010)

Utilizzando la **Tabella 7**, SubBytes(x) è

$0B$	20	CC	$0A$
82	$7B$	AE	28
$F0$	81	96	27
$2D$	78	$7C$	37

.

Convertiamo ora ogni elemento della **Matrice** (4.1) in un elemento del campo $GF(2^8)$:

BinaryToField

$x^7+x^4+x^3+x^2+x$	$x^6+x^4+x^2$	x^5+x^2+x+1	x^7+x^5+x+1
x^4+1	$x+1$	$x^7+x^5+x^4+x^3+x^2+x$	$x^7+x^6+x^5+x^3+x^2+x$
x^4+x^2+x+1	x^7+x^4+1	$x^5+x^4+x^2+1$	$x^5+x^4+x^3+x^2+1$
$x^7+x^6+x^5+x^4+x^3+x$	x^7+x^6+1	1	$x^7+x^5+x^4+x$

L'inverso di ogni elemento della matrice precedente in $GF(2^8)$ è stato tabulato in

FieldInv

x^7+x^3+1	$x^6+x^3+x^2$	$x^7+x^6+x^3+1$	x^7+x^6+x+1
$x^7+x^5+x^4+x^2$	$x^7+x^6+x^5+x^4+x^2+x$	x^7+x^2+x	$x^4+x^3+x^2+x$
$x^6+x^4+x^3+x^2+x+1$	x^6+x^4+x	$x^5+x^4+x^3+1$	$x^7+x^5+x^4+x^3+x+1$
$x^6+x^5+x^4+x^3+x^2+1$	x^5+x^3	1	$x^4+x^3+x^2+x+1$

Il corrispondente in binario è

FieldToBinary

(1000)(1001)	(0100)(1100)	(1100)(1001)	(1100)(0011)
(1011)(0100)	(1111)(0110)	(1000)(0110)	(0001)(1110)
(0101)(1111)	(0110)(1010)	(0011)(1001)	(1011)(1011)
(0111)(1101)	(0010)(1000)	(0000)(0001)	(0001)(1111)

Applichiamo ora la trasformazione affine $(*)$ al primo elemento $9E$ di (4.1). Si determina così la stringa

$$(b_7b_6b_5b_4b_3b_2b_1b_0) = 00001011,$$

che in esadecimale corrisponde a 0B.

Applicando lo stesso algoritmo su tutti gli altri elementi di (4.1), la matrice in uscita del **SubBytes** è

00001011	00100000	11001100	00001010
10000010	01111011	10101110	00101000
11110000	10000001	10010110	00100111
00101101	01111000	01111100	00110111

ovvero, in esadecimale

$0B$	20	CC	$0A$
82	$7B$	AE	28
$F0$	81	96	27
$2D$	78	$7C$	37

(4.2)

4.4.2 ShiftRows

ShiftRows agisce sulle righe della matrice stato, pertanto l'i-esima riga, $i = 0, 1, 2, 3$, viene shiftata verso sinistra i volte.

Quindi la **ShiftRows** di

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

(4.3)

è la matrice

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

(4.4)

In particolare, applicando l'operazione **ShiftRows** a (4.2), si ottiene

$0B$	20	CC	**0A**
$7B$	AE	**28**	82
96	**27**	$F0$	81
37	$2D$	78	$7C$

(4.5)

4.4.3 MixColumn

Per meglio comprendere l'operazione di **MixColumn**, è conveniente rappresentare le colonne della matrice in (4.5) come elementi dell'anello quoziente $\frac{GF(2^8)[y]}{\langle y^4+1\rangle}$.

Poichè ogni byte individua un unico elemento di $GF(2^8)$, allora ad ogni colonna della **Matrice** (4.4) può essere associato un polinomio di grado al più 3 a coefficienti in $GF(2^8)$. Siffatto polinomio individua un unico elemento dell'anello quoziente $\frac{GF(2^8)[y]}{\langle y^4+1\rangle}$ nella maniera usuale.

Quindi, per esempio, alla prima colonna nella **Matrice** (4.4) verrà associato l'elemento di $\frac{GF(2^8)[y]}{\langle y^4+1\rangle}$ individuato da

$$s_0(y) = \overline{s_{0,0}}\, y^3 + \overline{s_{1,1}}\, y^2 + \overline{s_{2,2}}\, y + \overline{s_{3,3}},$$

dove $\overline{s_{i,i}}$ è il corrispondente in $GF(2^8)$ di $s_{i,i}$.

I polinomi corrispondenti alle colonne della matrice in (4.5) (o anche (4.4)) sono moltiplicati per il polinomio

$$d(y) = \overline{03}y^3 + \overline{01}y^2 + \overline{01}y + \overline{02}.$$

Pertanto, l'output dell'operazione **MixColumn** è

$$s'(y) = d(y)s(y) \qquad mod\ (y^4 + 1).$$

Come nell'operazione **SubBytes**, la moltiplicazione può essere espressa in forma matriciale come segue

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} \overline{02} & \overline{03} & \overline{01} & \overline{01} \\ \overline{01} & \overline{02} & \overline{03} & \overline{01} \\ \overline{01} & \overline{01} & \overline{02} & \overline{03} \\ \overline{03} & \overline{01} & \overline{01} & \overline{02} \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}. \qquad (**)$$

Quindi, il risultato di **MixColumn** applicato a (4.4) è dato da

$$\begin{array}{|c|c|c|c|} \hline \overline{02} & \overline{03} & \overline{01} & \overline{01} \\ \hline \overline{01} & \overline{02} & \overline{03} & \overline{01} \\ \hline \overline{01} & \overline{01} & \overline{02} & \overline{03} \\ \hline \overline{03} & \overline{01} & \overline{01} & \overline{02} \\ \hline \end{array} \cdot \begin{array}{|c|c|c|c|} \hline \overline{s_{0,0}} & \overline{s_{0,1}} & \overline{s_{0,2}} & \overline{s_{0,3}} \\ \hline \overline{s_{1,1}} & \overline{s_{1,2}} & \overline{s_{1,3}} & \overline{s_{1,0}} \\ \hline \overline{s_{2,2}} & \overline{s_{2,3}} & \overline{s_{2,0}} & \overline{s_{2,1}} \\ \hline \overline{s_{3,3}} & \overline{s_{3,0}} & \overline{s_{3,1}} & \overline{s_{3,2}} \\ \hline \end{array} \qquad (4.6)$$

Lo pseudo-codice di **MixColumn** è il seguente:

Algoritmo 4.3. (**MixColumn**(c))
external FieldMult, BinaryToField, FieldToBinary
for $i \leftarrow$ **to** 3
do $t_i \leftarrow$ BinaryToField($s_{i,c}$)
$u_0 \leftarrow \text{FieldMult}(x, t_0) \oplus \text{FieldMult}(x+1, t_1) \oplus t_2 \oplus t_3$
$u_1 \leftarrow \text{FieldMult}(x, t_1) \oplus \text{FieldMult}(x+1, t_2) \oplus t_3 \oplus t_0$
$u_2 \leftarrow \text{FieldMult}(x, t_2) \oplus \text{FieldMult}(x+1, t_3) \oplus t_0 \oplus t_1$
$u_3 \leftarrow \text{FieldMult}(x, t_3) \oplus \text{FieldMult}(x+1, t_0) \oplus t_1 \oplus t_2$
for $i \leftarrow 0$ **to** 3
do $s_{i,c} \leftarrow \text{FieldToBinary}(u_i)$

Osservando gli input, l'algoritmo **FieldMult** moltiplica una costante con una variabile. Infatti, il primo argomento è il polinomio x o $x+1$ che corrisponde all'elemento $\overline{02}$ e $\overline{03}$ del campo $GF(2^8)$, rispettivamente.

In binario, la moltiplicazione di t_i per x è equivalente ad uno shift verso sinistra di t_i, mentre la moltiplicazione per $x+1$ corrisponde ad uno shift verso sinistra di t_i a cui viene successivamente sommato t_i. Inoltre, l'operazione di XORING corrisponde all'usuale somma modulo 2 componente per componente.

Applicare l'operazione **MixColumn** al risultato di **ShiftRows**

$\overline{0B}$	$\overline{20}$	$\overline{CC}$	$\overline{0A}$
$\overline{7B}$	$\overline{AE}$	$\overline{28}$	$\overline{82}$
$\overline{96}$	$\overline{27}$	$\overline{F0}$	$\overline{81}$
$\overline{37}$	$\overline{2D}$	$\overline{78}$	$\overline{7C}$

(4.7)

corrisponde a trasformare ogni colonna di (4.7) mediante (∗∗) ottenendo in questo modo

$\overline{3A}$	$\overline{A3}$	$\overline{73}$	$\overline{74}$
$\overline{6B}$	$\overline{23}$	$\overline{EF}$	$\overline{F1}$
$\overline{1E}$	$\overline{B7}$	$\overline{97}$	$\overline{15}$
$\overline{9E}$	$\overline{B3}$	$\overline{67}$	$\overline{E5}$

(4.8)

L'equivalente in binario di (4.8) è

00111010	10100011	01110011	01110100
01101011	00100011	11101111	11110001
00011110	10110111	10010111	00010101
10011110	10110011	01100111	11100101

(4.9)

.

Siccome le operazioni sono eseguite in $GF(2^8)$, spieghiamo in dettaglio come da (4.7) si ottiene (4.8). Lo faremo solo per la prima colonna siccome per le altre si procede in modo analogo. Vale che

t_0	$\overline{0B}$	x^3+x+1
t_1	$\overline{7B}$	$x^6+x^5+x^4+x^3+x+1$
t_2	$\overline{96}$	$x^7+x^4+x^2+x$
t_3	$\overline{37}$	$x^5+x^4+x^2+x+1$

Tenendo presente la riduzione modulo $p(x) = x^8+x^4+x^3+x+1$ si ha

$$\begin{array}{l}
\overline{u_{0,0}} = x(x^3+x+1)+(x+1)(x^6+x^5+x^4+x^3+x+1)+(x^7+x^4+x^2+x)+(x^5+x^4+x^2+x+1) \\
\overline{u_{1,0}} = x(x^6+x^5+x^4+x^3+x+1)+(x+1)(x^7+x^4+x^2+x)+(x^5+x^4+x^2+x+1)+(x^3+x+1) \\
\overline{u_{2,0}} = x(x^7+x^4+x^2+x)+(x+1)(x^5+x^4+x^2+x+1)+(x^3+x+1)+(x^6+x^5+x^4+x^3+x+1) \\
\overline{u_{3,0}} = x(x^5+x^4+x^2+x+1)+(x+1)(x^3+x+1)+(x^6+x^5+x^4+x^3+x+1)+(x^7+x^4+x^2+x)
\end{array}$$

Pertanto,

$\overline{u_{0,0}}$	$x^5+x^4+x^3+x$	$\overline{3A}$	00111010
$\overline{u_{1,0}}$	$x^6+x^5+x^3+x+1$	$\overline{6B}$	01101011
$\overline{u_{2,0}}$	$x^4+x^3+x^2+x$	$\overline{1E}$	00011110
$\overline{u_{3,0}}$	$x^7+x^4+x^3+x^2+x$	$\overline{9E}$	10011110

corrisponde alla prima colonna di (4.8) e di (4.9).

4.4.4 AddRoundKey

Quest'ultima operazione consiste in un'operazione XOR tra lo stato corrente calcolato w^i e la chiave K^i. Tale operazione bit-a-bit corrisponde all'addizione in $GF(2)$.

4.4.5 KeyExpansion

Prima di iniziare il processo di cifratura, il crittosistema prevede la creazione di una lista di N_r+1 sottochiavi $(K^0, K^1, \ldots, K^{N_r})$ a partire da una chiave iniziale segreta K data in input attraverso l'algoritmo spiegato qui di seguito (una per ogni iterazione più quella necessaria per lo sbiancamento nel primo **AddRoundKey**). La concatenazione delle sottochiavi è chiamata **chiave espansa**. Ogni sottochiave è una quadrupla di **word** (ciascuna costituita da 4 byte):

$$(w[4i], w[4i+1], w[4i+2], w[4i+3]).$$

Al suo interno vengono eseguite due operazioni:

1. **RotWord:** opera uno shift a sinistra di un byte di una word.
2. **SubWord:** applica l'S-box definito per l'AES su ognuno dei 4 byte di ogni word data in input.

L'input di **KeyExpansion** è una chiave K di lunghezza fissata che si presenta nella forma di una stringa $K_0, \ldots, K_j$ byte con $j = 15$ o 23 o 31 a seconda che K sia rispettivamente di 128 o 192 o 256 bit. L'algoritmo poi restituisce una matrice di elementi $w[i]$. Le due operazioni lavorano su ogni singolo elemento $w[i]$, pertanto richiedono come input una word. Se $B_0B_1B_2B_3$ è una word, con B_i un byte, allora

- **RotWord**$(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0)$
- **SubWord**$(B_0, B_1, B_2, B_3) = (B'_0, B'_1, B'_2, B'_3)$, dove B'_i si ottiene da B_i applicando l'inverso nel campo $GF(2^8)$ di B_i e poi la trasformazione affine definita in $(*)$.

Lo pseudocodice dell'algoritmo **KeyExpansion** è descritto di seguito:

Algoritmo 4.4. **(KeyExpansion**(K)**)**

external RotWord, SubWord
$RC[1] \leftarrow 01000000$
$RC[2] \leftarrow 02000000$
$RC[3] \leftarrow 04000000$
$RC[4] \leftarrow 08000000$
$RC[5] \leftarrow 10000000$
$RC[6] \leftarrow 20000000$
$RC[7] \leftarrow 40000000$
$RC[8] \leftarrow 80000000$
$RC[9] \leftarrow 1B000000$
$RC[10] \leftarrow 36000000$
for $i \leftarrow 0$ **to** 3
do $w[i] \leftarrow (chiave[4i], chiave[4i+1], chiave[4i+2], chiave[4i+3])$
for $i \leftarrow 4$ **to** 43
do
$temp \leftarrow w[i-1]$
if $i \equiv \mathbf{0} \ mod \ 4$
then $temp \leftarrow \text{SubWord}(\text{RotWord}(temp)) \oplus RC[i/4]$
else $w[i] \leftarrow w[i-4] \oplus temp$
return $(w[0], \ldots, w[43])$

L'algoritmo inizializza dieci word costanti denominate $RC[i]$ per $i = 1, \ldots, 10$. Successivamente, suddivide la chiave data in input in 4 word. Quindi,

$$K = (w[0], w[1], w[2], w[3]).$$

Le word $w[i]$ successive o vengono determinate da

$$w[i-4] \oplus \textbf{SubWord}(\textbf{RotWord}(w[i-1])) \oplus RC[i/4],$$

o vengono determinate attraverso $w[i-4] \oplus w[i-1]$, a seconda che i sia un multiplo di 4 o meno, rispettivamente.

La chiave espansa i-esima è

$$K^i = (w[4i], w[4i+1], w[4i+2], w[4i+3]),$$

dove $i = 0, \ldots, N_r$. In particolare, $K^0 = K$. Sia

$$g(w) = \text{SubWord}(\text{RotWord}(w)) \oplus RC[i/4],$$

per i multiplo di 4, allora la struttura dell'algoritmo **KeyExpansion** è descritta nella seguente figura.

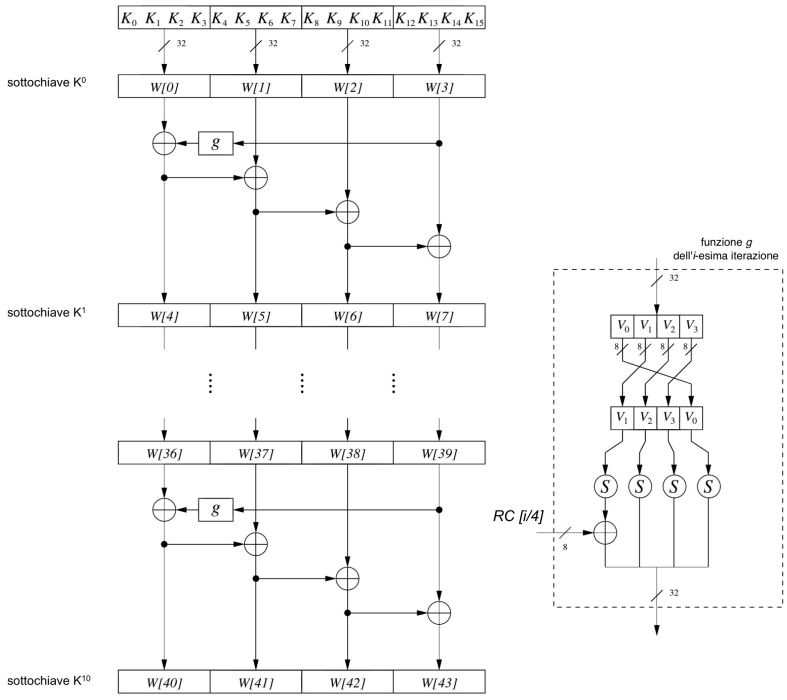

Figura 4.4: Struttura dell'operazione KeyExpansion

Forniamo un esempio concreto di come la **KeyExpansion** opera su una chiave K di 128 bit. Per semplicità useremo la notazione esadecimale ricordando che ogni byte corrisponde ad una coppia di cifre esadecimali.

Sia $K = (K_0, K_1, \ldots, K_{15})$ dove

K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
27	4F	E5	1A	B7	01	79	4D	3A	DF	95	B9	0F	6C	8B	EE

L'algoritmo genera 11 sottochiavi immagazzinate nella matrice che rappresenta la chiave espansa. Analizziamo nel dettaglio solo la creazione della seconda sottochiave K^1, il procedimento è analogo per le altre sottochiavi.

La sottochiave $K^0 = K$ è

$$(w[0], w[1], w[2], w[3]) = (274FE51A,\ B701794D,\ 3ADF95B9,\ 0F6C8BEE).$$

Dalla definizione di **RotWord** e **SubWord** segue che

$$\text{RotWord}(w[3]) = 6C8BEE0F$$
$$\text{SubWord}(\text{RotWord}(w[3])) = 503D2876.$$

Quindi

$$w[4] = (503D2876 \oplus 01000000) \oplus w[0] = 7672CD6C$$
$$w[5] = w[1] \oplus w[4] = C173B421$$
$$w[6] = w[2] \oplus w[5] = FBAC2198$$
$$w[7] = w[3] \oplus w[6] = F4C0AA76,$$

e pertanto K^1 è

$$(w[4], w[5], w[6], w[7]) = (7672CD6C,C173B421,FBAC2198,F4C0AA76).$$

Quindi,

$K^0 =$

w[0]	w[1]	w[2]	w[3]
27	B7	3A	0F
4F	01	DF	6C
E5	79	95	8B
1A	4D	B9	EE

$K^1 =$

w[4]	w[5]	w[6]	w[7]
76	C1	FB	F4
72	73	AC	C0
CD	B4	21	AA
6C	21	98	76

Procedendo in modo analogo per $K^2, \ldots, K^{10}$ si ottiene la tabella:

K^2 =

w[8]	w[9]	w[10]	w[11]
CE	0F	F4	00
DE	AD	01	C1
F5	41	60	CA
D3	F2	6A	1C

K^3 =

w[12]	w[13]	w[14]	w[15]
B2	BD	49	49
AA	07	06	C7
69	28	48	82
B0	42	28	34

K^4 =

w[16]	w[17]	w[18]	w[19]
7C	C1	88	C1
B9	BE	B8	7F
71	59	11	93
8B	C9	E1	D5

K^5 =

w[20]	w[21]	w[22]	w[23]
BE	7F	F7	36
65	DB	63	1C
72	2B	3A	A9
F3	3A	DB	0E

K^6 =

w[24]	w[25]	w[26]	w[27]
02	7D	8A	BC
B6	6D	0E	12
D9	F2	C8	61
F6	CC	17	19

K^7 =

w[28]	w[29]	w[30]	w[31]
8B	F6	7C	C0
59	34	3A	28
0D	FF	37	56
93	5F	48	51

K^8 =

w[32]	w[33]	w[34]	w[35]
3F	C9	B5	75
E8	DC	E6	CE
DC	23	14	42
29	76	3E	6F

K^9 =

w[36]	w[37]	w[38]	w[39]
AF	66	D3	A6
C4	18	FE	30
74	57	43	01
B4	C2	FC	93

K^{10} =

w[40]	w[41]	w[42]	w[43]
9D	FB	28	8E
B8	A0	5E	6E
A8	FF	BC	BD
90	52	AE	3D

Ciò completa la descrizione del cifrario AES.

4.5 Esempio di cifratura

Presentiamo ora un esempio di cifratura integrale in cui la lunghezza della chiave K è pari a 128 bit ed il numero delle iterazioni è 10.
I dati in input sono:

Blocco iniziale = B9 5E F2 E0 E3 02 E8 8C 1D 61 A0 B8 AC 82 B6 5C

Chiave K = 27 4F E5 1A B7 01 79 4D 3A DF 95 B9 0F 6C 8B EE

Iter.	Stato	SubBytes	ShiftRows	MixColumns		AddRoundKey
0	B9 E3 1D AC 5E 02 61 82 F2 E8 A0 B6 E0 8C B8 5C				⊕	27 B7 3A 0F 4F 01 DF 6C E5 79 95 8B 1A 4D B9 EE
1	9E 54 27 A3 11 03 BE EE 17 91 35 3D FA C1 01 B2	0B 20 CC 0A 82 7B AE 28 F0 81 96 27 2D 78 7C 37	0B 20 CC 0A 7B AE 28 82 96 27 F0 81 37 2D 78 7C	3A A3 73 74 6B 23 EF F1 1E B7 97 15 9E B3 67 E5	⊕	76 C1 FB F4 72 73 AC C0 CD B4 21 AA 6C 21 98 76

Pertanto, applicando l'intero Crittosistema **AES** al messaggio in chiaro

B95EF2E0E302E88C1D61A0B8AC82B65C,

si determina il seguente messaggio cifrato

6E717F1C64C34C4F95135C4C74CEE503.