Azzolini Riccardo 2021-03-22

Advanced Encryption Standard

1 Storia

Alla fine degli anni '90, le chiavi di 56 bit del DES erano ormai troppo piccole per prevenire gli attacchi a forza bruta. La soluzione più immediata, come già detto, era il TDES, che consiste nell'applicare il DES tre volte di seguito a ciascun blocco, con tre chiavi diverse, estendendo così la lunghezza della chiave a 168 bit. Tuttavia, l'algoritmo DES era stato progettato per le implementazioni hardware, dunque non si presta a implementazioni software efficienti, e la necessità di applicarlo 3 volte a ciascun blocco avrebbe esacerbato il problema dell'inefficienza, soprattutto considerando che i blocchi su cui DES opera sono relativamente piccoli (64 bit), quindi per grandi quantità di dati TDES richiede molte esecuzioni del DES.

Per risolvere questi problemi, nel 1997 il NIST indisse una gara pubblica finalizzata alla selezione di un nuovo standard di cifratura simmetrica, chiamato **Advanced Encryption Standard** (**AES**), con il quale sostituire il DES. I principali criteri della gara erano i seguenti:

- gli algoritmi partecipanti dovevano operare su blocchi di 128 bit ed essere utilizzabili con chiavi di 128, 192 e 256 bit (questa flessibilità era richiesta in previsione dell'aumento della capacità di calcolo nel tempo);
- l'analisi della sicurezza degli algoritmi partecipanti era aperta a tutta la comunità di crittografi;
- le scelte progettuali degli algoritmi dovevano essere note;
- gli algoritmi dovevano prestarsi a implementazioni sia software che hardware efficienti, con la possibilità di eseguire in parallelo varie parti delle operazioni di cifratura/decifratura, e dovevano essere implementabili anche su dispositivi con poca memoria.

Nel 1998, il NIST selezionò inizialmente 15 algoritmi, poi nel 1999 la selezione fu ridotta a 5 finalisti, e infine nel 2000 vinse l'algoritmo **Rijndael**, sviluppato dai crittografi belgi Vincent Rijmen e Joan Daemen,[1] che nel 2001 venne standardizzato dal NIST.

Le motivazioni per cui fu selezionato Rijndael sono le seguenti:

[1] Il nome di quest'algoritmo, "Rijndael", è basato su una combinazione dei nomi dei progettisti, "Rijmen" e "Daemen".

- è molto resistente agli attacchi crittoanalitici noti (come del resto lo erano tutti gli altri algoritmi finalisti della gara AES);
- può essere implementato con un'ottima efficienza sia nell'hardware che nel software, su varie piattaforme, e comprende varie operazioni che possono essere eseguite in parallelo;
- ha un algoritmo di key scheduling molto veloce;
- è adatto ad ambienti con spazio di memoria ridotto;
- l'unità di elaborazione principale è il byte, quindi è possibile implementarlo facilmente anche su processori a 8 bit;
- è molto flessibile, perché consente di specificare non solo la dimensione della chiave (come richiesto dalla gara AES), ma anche il numero di round e la dimensione del blocco.

2 Caratteristiche

L'algoritmo Rijndael è un cifrario del prodotto: esso prevede 10, 12 o 14 fasi uguali (solo l'ultima, come si vedrà a breve, presenta alcune differenze), eseguite con sottochiavi diverse generate a partire dalla singola chiave data in input. Esso *non* è però basato su una rete di Feistel: ogni fase elabora l'intero blocco in parallelo, effettuando su di esso tre sostituzioni e una permutazione, senza suddividerlo in due metà.

Il messaggio in chiaro è un blocco di n bit, dove n può valere 128, 192 o 256 in Rijndael, ma lo standard AES fissa $n = 128$. Tale blocco viene rappresentato come una matrice di $\frac{n}{8}$ byte, chiamata **stato**, che ha 4 righe e $N_b = \frac{n}{8 \cdot 4} = \frac{n}{32}$ colonne, cioè 4, 6 o 8 colonne, a seconda della dimensione del blocco; in particolare, nel caso di AES, che fissa $n = 128$, si hanno sempre $N_b = 4$ colonne, ovvero una matrice di 4×4 byte. Lo stato è riempito colonna per colonna: i primi 4 byte del blocco di input vengono messi nella prima colonna, dall'alto verso il basso, poi i 4 byte successivi vengono messi nella seconda colonna, sempre dall'alto verso il basso, e così via. Allora, i byte contenuti, ad esempio, in una matrice di stato a 4 colonne

$$S = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

corrispondono ai byte di un blocco di 128 bit in questo modo:

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

Sia la versione originale di Rijndael che lo standard AES ammettono chiavi di 128, 192 e 256 bit. Analogamente al blocco di testo in chiaro, la chiave è rappresentata come una matrice di byte avente 4 righe e $N_k = \frac{k}{8 \cdot 4} = \frac{k}{32}$ colonne, dove k è il numero di bit della chiave. Ad esempio, una chiave di $k = 192$ bit viene rappresentata da una matrice di 4×6 byte:

$$\begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \end{bmatrix}$$

Come lo stato, anche la matrice della chiave viene riempita per colonne.

La dimensione della chiave determina il numero di round da eseguire:

k	Round
128	10
192	12
256	14

Tipicamente, per indicare l'uso di AES si adotta una nomenclatura che specifica anche la dimensione della chiave: AES-128, AES-192 o AES-256.

3 Algoritmo di cifratura

La cifratura con l'algoritmo Rijndael avviene tramite le seguenti operazioni:

1. Per prima cosa, si esegue un'operazione chiamata **AddRoundKey**, che consiste nello XOR bit a bit dello stato (che in questo caso corrisponde al blocco in input) con una sottochiave.
2. In seguito, vengono eseguite **9, 11 o 13 fasi uguali** (una in meno rispetto al numero totale di fasi da eseguire). Ciascuna di queste prevede 4 operazioni:
   a) **SubstituteBytes**: ciascun byte dello stato viene sostituito in base a una S-box;
   b) **ShiftRows**: si effettua uno scorrimento delle righe dello stato (ovvero una semplice permutazione);
   c) **MixColumns**: si applica una trasformazione sulle colonne, che sostanzialmente è una sostituzione basata su principi matematici;
   d) L'ultima operazione è ancora una **AddRoundKey**, analoga a quella effettuata all'inizio, ma fatta con una nuova sottochiave.

3. Infine, si esegue la **fase finale** (cioè la decima, dodicesima o quattordicesima fase), che è leggermente diversa dalle precedenti, perché comprende le operazioni

   a) SubstituteBytes

   b) ShiftRows

   c) AddRoundKey

   ma non MixColumns.

Tutte queste operazioni sono invertibili, il che consente di effettuare la decifratura.

Un'osservazione importante è che ogni operazione SubstituteBytes avviene subito dopo una AddRoundKey (sia essa quella iniziale, o quella alla fine del round precedente), dunque in sostanza la sostituzione non dipende solo dai valori dello stato, ma anche dai valori di una sottochiave.

3.1 SubstituteBytes

L'operazione SubstituteBytes è una semplice sostituzione di ciascun byte dello stato, effettuata utilizzando una singola S-box di 16×16 byte, che contiene una permutazione di tutti i 256 possibili valori di 8 bit (qui rappresentati in esadecimale):

	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

Dato ciascun byte b dello stato, i primi 4 bit di b (letti come numero decimale o esadecimale, a seconda delle etichette della tabella) determinano il numero di una riga della S-box e i restanti 4 bit determinano il numero di una colonna. L'elemento presente nella S-box all'intersezione di questa riga e di questa colonna è il byte che viene inserito nello stato come sostituto. Ad esempio, $b = 00101010$ viene sostituito con il valore situato nella riga $(0010)_{10} = 2$ e nella colonna $(1010)_{10} = 10$ (A in esadecimale), che è $(E5)_2 = 11100101$.

Per la decifratura si usa una S-box inversa, che associa a ciascun byte sostituito il valore del byte originale: ad esempio, siccome $b = 00101010$ viene sostituito con $b' = 11100101$, la S-box inversa contiene il byte $b = 00101010$ nella posizione determinata da $b' = 11100101$, che è l'intersezione della riga $(1110)_{10} = 14$ (E in esadecimale) e della colonna $(0101)_{10} = 5$.

3.1.1 Costruzione della S-box

Come detto in precedenza, tutte le scelte progettuali di AES/Rijndael sono note, compreso il modo in cui è stata costruita questa S-box: essa è una tabella che contiene i risultati precomputati di una trasformazione non lineare definita sui singoli byte interpretati come valori del **campo finito** (**di Galois**) $\mathrm{GF}(2^8)$.

In generale, ogni campo finito ha *ordine* p^n, cioè p^n elementi, dove p è un numero primo e $n \geq 1$. Se $n > 1$, gli elementi di un campo finito di ordine p^n possono essere visti come polinomi di grado massimo $n-1$ aventi coefficienti interi compresi tra 0 e $p-1$, e allora le operazioni del campo sono definite tramite l'aritmetica polinomiale. In particolare, gli elementi di $\mathrm{GF}(2^8)$ sono polinomi di grado 7 i cui coefficienti possono assumere solo i valori 0 e 1:

$$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0 \quad \text{dove } a_i \in \{0, 1\}$$

Gli 8 coefficienti binari $(a_7, a_6, \ldots, a_0)$ che caratterizzano un polinomio di questo tipo possono essere rappresentati come gli 8 bit di un byte, ovvero ciascuno dei $2^8 = 256$ elementi di $\mathrm{GF}(2^8)$ corrisponde a uno dei possibili valori di un byte, dunque le operazioni definite su $\mathrm{GF}(2^8)$ sono utilizzabili come operazioni sui byte.

Per costruire la S-box, si parte da una tabella che non effettua alcuna sostituzione, cioè sostituisce ciascun byte con se stesso:

	0	1	2	$\cdots$	F
0	00	01	02	$\cdots$	0F
1	10	11	12	$\cdots$	1F
2	20	21	22	$\cdots$	2F
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
F	F0	F1	F2	$\cdots$	FF

Poi, si applica a ciascun byte nella tabella una trasformazione non lineare definita sui valori di $\mathrm{GF}(2^8)$, che consiste nel calcolare l'inverso moltiplicativo del polinomio corrispondente alla configurazione di bit del byte considerato. Infine, i valori così ottenuti vengono ulteriormente modificati tramite una trasformazione affine.

Si noti che eseguire concretamente tutti questi calcoli è stato necessario solo durante la progettazione dell'algoritmo, per costruire "da zero" la S-box. Invece, un'implementazione dell'algoritmo può semplicemente usare la S-box precomputata (riportata direttamente nello standard AES), per cui l'operazione di sostituzione è molto semplice ed efficiente.

3.2 ShiftRows

L'operazione ShiftRows è uno scorrimento circolare verso sinistra delle righe della matrice di stato:

- la prima riga non cambia;
- la seconda riga scorre di 1 byte a sinistra;
- la terza riga scorre di 2 byte a sinistra;
- la quarta riga scorre di 3 byte a sinistra.

Questa permutazione assicura che i 4 byte di ciascuna colonna (ovvero 4 byte consecutivi nel blocco elaborato, dato che lo stato è riempito per colonne) vengano distribuiti in 4 colonne diverse (in tutte le colonne, se in particolare il blocco è di $n = 128$ bit, come nel caso di AES). Come si vedrà a breve, ciò è importante per l'operazione successiva, MixColumns.

Ad esempio, se si considera un blocco di $n = 192$ bit, cioè se lo stato ha $N_b = 6$ colonne, allora la permutazione eseguita è:

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} & s_{0,4} & s_{0,5} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} & s_{2,4} & s_{2,5} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} & s_{3,4} & s_{3,5} \end{bmatrix} \longrightarrow \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} & s_{0,4} & s_{0,5} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,4} & s_{1,5} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,4} & s_{2,5} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,4} & s_{3,5} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

Per la decifratura, l'operazione ShiftRows viene invertita semplicemente eseguendo gli scorrimenti circolari verso destra invece che verso sinistra.

3.3 MixColumns

Nell'operazione MixColumns ogni byte di ciascuna colonna viene sostituito con un valore che dipende da tutti i byte presenti nella stessa colonna. Questo è il motivo dell'importanza della precedente operazione ShiftRows: essa fa in modo che la sostituzione effettuata da MixColumns non dipenda sempre dagli stessi 4 byte in ciascuna colonna, ma invece consideri a ogni round una configurazione di byte diversa. Si può infatti dimostrare che, dopo un certo numero di round, la combinazione di ShiftRows e MixColumns fa sì che

tutti i bit dell'output dipendano da tutti i bit dell'input, realizzando dunque un buon effetto di diffusione.

La sostituzione effettuata da MixColumns avviene tramite un prodotto matriciale $C \cdot S = S'$: la matrice di stato S viene moltiplicata con una matrice di byte costante C,

$$C = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

interpretando i valori delle due matrici come elementi di $\mathrm{GF}(2^8)$. Così, per la definizione del prodotto matriciale, ciascun elemento dello stato risultante S' è la somma dei prodotti tra gli elementi di una riga di C e gli elementi di una colonna di S (dove le operazioni di somma e prodotto sono quelle definite su $\mathrm{GF}(2^8)$). Ad esempio, nel caso di AES, in cui lo stato ha $N_b = 4$ colonne, il prodotto matriciale eseguito è

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

ovvero i singoli elementi $s'_{i,j}$ sono calcolati come

$$\begin{aligned} s'_{0,0} &= 02 \cdot s_{0,0} + 03 \cdot s_{1,0} + 01 \cdot s_{2,0} + 01 \cdot s_{3,0} \\ s'_{0,1} &= 02 \cdot s_{0,1} + 03 \cdot s_{1,1} + 01 \cdot s_{2,1} + 01 \cdot s_{3,1} \\ &\vdots \\ s'_{3,3} &= 03 \cdot s_{0,3} + 01 \cdot s_{1,3} + 01 \cdot s_{2,3} + 01 \cdot s_{3,3} \end{aligned}$$

(dove $+$ e $\cdot$ rappresentano, rispettivamente, la somma e il prodotto su $\mathrm{GF}(2^8)$). Come si può osservare da questi calcoli, ciascun elemento $s'_{i,j}$ della j-esima colonna di S' dipende appunto da *tutti* gli elementi della j-esima colonna di S $(s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j})$.

La trasformazione MixColumns è invertibile, e la sua inversa è ancora esprimibile come un prodotto matriciale.

3.4 AddRoundKey

L'operazione AddRoundKey consiste nello XOR bit a bit tra lo stato e una sottochiave, chiamata **RoundKey**. La RoundKey è lunga n bit, cioè quanto il blocco di testo in chiaro e lo stato (il che è necessario per poterla mettere in XOR bit a bit con lo stato), è diversa per ogni round dell'algoritmo (o meglio, per ogni occorrenza dell'operazione

AddRoundKey, poiché essa viene fatta una volta in più, prima del primo round) ed è calcolata a partire dalla **CipherKey** — la chiave di cifratura data in input all'algoritmo — secondo la procedura di key scheduling.

In simboli, dati lo stato S e una sottochiave K (anch'essa vista come una matrice di dimensione uguale a quella dello stato), il calcolo eseguito è $S \oplus K = S'$. Ad esempio, nel caso di blocchi di $n = 128$ bit (AES), si ha

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

ovvero $s'_{i,j} = s_{i,j} \oplus k_{i,j}$ (per ogni $i = 0, \ldots, 3$ e $j = 0, \ldots, 3$).

Come per ogni operazione di XOR bit a bit, per invertire AddRoundKey nella decifratura è sufficiente fare di nuovo lo XOR con la stessa sottochiave:

$$S' \oplus K = (S \oplus K) \oplus K = S$$

3.5 Operazioni eseguibili in parallelo

Tutte le operazioni necessarie per la cifratura si prestano all'esecuzione di più calcoli in parallelo:

- SubstituteBytes sostituisce ciascun byte dello stato in modo indipendente dagli altri, quindi può essere effettuata in parallelo su tutti i byte;
- ShiftRows può avvenire in parallelo su ciascuna delle righe dello stato, che permuta indipendentemente dalle altre;
- MixColumns può essere fatta in parallelo sulle colonne dello stato, poiché i calcoli per ciascuna colonna non coinvolgono i valori di altre colonne;
- AddRoundKey opera bit a bit, quindi è fattibile in parallelo su tutti i bit o i byte dello stato.

4 Decifratura

Come appena visto, tutte le trasformazioni effettuate durante la cifratura sono facilmente invertibili: AddRoundKey è l'inversa di se stessa, SubstituteBytes e MixColumns si invertono effettuando sostanzialmente gli stessi passi con tabelle di riferimento diverse, e ShiftRows si inverte facendo gli scorrimento a destra invece che a sinistra.

Perciò, la decifratura consiste nell'eseguire, una a una, le inverse di tutte le trasformazioni applicate nella cifratura, in ordine dall'ultima alla prima. In particolare, per AddRoundKey bisogna utilizzare le sottochiavi in ordine inverso.

Al fine di rendere la struttura dell'algoritmo di decifratura simile a quella dell'algoritmo di cifratura, le trasformazioni inverse sono organizzate in questo modo:

1. Una AddRoundKey iniziale, con l'ultima sottochiave.
2. 9, 11 o 13 fasi uguali, ciascuna costituita da:
   a) l'inversa di ShiftRows;
   b) l'inversa di SubstituteBytes;
   c) AddRoundKey, usando di fase in fase le sottochiavi dalla penultima alla seconda;
   d) l'inversa di MixColumns.
3. Un'ultima fase diversa, costituita da:
   a) l'inversa di ShiftRows;
   b) l'inversa di SubstituteBytes;
   c) AddRoundKey con la prima sottochiave.

5 Procedura di key scheduling

La procedura di key scheduling è composta da due parti:

1. l'espansione della CipherKey (la chiave data in input) a una chiave più lunga, chiamata **ExpandedKey**, che contiene una dopo l'altra tutte le sottochiavi necessarie, ed è rappresentata come un vettore w di word, ciascuna composta da 4 byte;
2. l'estrazione delle RoundKey (sottochiavi) dalla ExpandedKey.

5.1 Espansione della chiave

Se N_r è il numero di round, allora è necessario generare $N_r + 1$ sottochiavi: una per l'AddRoundKey effettuato in ciascuna fase, più una per l'AddRoundKey iniziale. Sapendo che ciascuna sottochiave deve essere lunga n bit, ovvero quanto lo stato, si deduce che la lunghezza della ExpandedKey è $n(N_r + 1)$. Ad esempio, nel caso di AES-128 (AES con chiave di 128 bit, che implica 10 round), l'algoritmo di espansione riceve in input una CipherKey di 128 bit (16 byte, 4 word) e, siccome $n = 128$ e $N_r = 10$, deve restituire una

ExpandedKey di $128 \cdot 11 = 1408$ bit (176 byte, 44 word). Per questo caso, l'algoritmo di espansione è descritto dal seguente pseudocodice:[2]

```
KeyExpansion(byte key[16], word w[44]) {
    for (i = 0; i < 4; i++)
        w[i] = (key[4*i], key[4*i + 1], key[4*i + 2], key[k*i + 3]);
    for (i = 4; i < 44; i++) {
        word temp = w[i - 1];
        if (i mod 4 == 0)
            temp = SubWord(RotWord(temp)) XOR Rcon[i/4];
        w[i] = w[i - 4] XOR temp;
    }
}
```

Innanzitutto, i 16 byte della CipherKey vengono direttamente inseriti come prime 4 word ($w[0, 3]$) del vettore w, cioè della della ExpandedKey. Poi, si calcolano iterativamente le restanti 40 word, ricavando ciascuna di esse dai valori di alcune delle word precedenti. In particolare, la word $w[i]$ viene calcolata in un modo che varia a seconda che l'indice i sia o meno un multiplo di 4:

- Se i *non* è multiplo di 4 (`i mod 4 != 0`), allora $w[i]$ è data dallo XOR bit a bit tra la word precedente e la word situata 4 posizioni prima:

$$w[i] = w[i-4] \oplus w[i-1]$$

- Se invece i è un multiplo di 4 (`i mod 4 == 0`), allora $w[i]$ è data ancora da uno XOR che coinvolge $w[i-4]$ e $w[i-1]$, ma prima di calcolare tale XOR la word $w[i-1]$ viene trasformata in questo modo:
  1. Si esegue uno scorrimento circolare a sinistra di un byte (`RotWord`).
  2. Si sostituisce ciascun byte della word risultante usando la stessa S-box impiegata per la cifratura (`SubWord`).
  3. Si fa lo XOR della word risultante con una **costante di fase**, contenuta nel vettore `Rcon` (all'indice $\frac{i}{4}$, perché si usa una costante ogni volta che i è un multiplo di 4, ovvero ogni 4 incrementi di i). Ciascuna di queste costanti è una word nella quale solo il primo byte ha un valore diverso da 0, dunque lo XOR trasforma solo il primo byte del risultato restituito da `SubWord`.

Effettuare calcoli diversi per ogni indice multiplo di 4 serve ad aggiungere del "rumore" per evitare ripetizioni nella chiave espansa. Questo rumore viene poi propagato anche alle word a indici non multipli di 4: ad esempio, il rumore introdotto nel calcolo di $w[4]$

[2] Per le altre combinazioni di dimensioni del blocco e della chiave, l'algoritmo di espansione è simile.

influenza il calcolo di $w[5] = w[1] \oplus w[4]$, e $w[5]$ a sua volta influenza il calcolo di il calcolo di $w[6] = w[2] \oplus w[5]$, e così via.

Come anticipato, si può osservare che la procedura di espansione della chiave è molto veloce, perché comprende solo trasformazioni semplici (XOR, scorrimenti circolari e sostituzioni con una S-box), e per la maggior parte delle word da calcolare esegue solo un singolo XOR.

5.2 Estrazione delle sottochiavi

Una volta ottenuta la ExpandedKey di $n(N_r + 1)$ bit, bisogna estrarre da essa le $N_r + 1$ sottochiavi, ciascuna di n bit, ovvero $N_b = \frac{n}{32}$ word, che servono per effettuare le altrettante operazioni AddRoundKey.

Se si indicano con gli indici $i = 0, 1, \ldots, N_r$ le $N_r + 1$ RoundKey, allora l'i-esima RoundKey è formata dall'i-esima sequenza di N_b word della ExpandedKey w, cioè dalle word $w[i, i+N_b-1]$. Ad esempio, se lo stato è di $n = 192$ bit, cioè $N_b = 6$ word, e si effettuano $N_r = 10$ round, allora la ExpandedKey è costituita da $6 \cdot 11 = 66$ word: le word $w[0, 5]$ costituiscono la prima RoundKey ($i = 0$), $w[6, 11]$ costituiscono la seconda RoundKey ($i = 1$), e così via fino all'undicesima RoundKey ($i = 10$), che è formata dalle word $w[60, 65]$.

Ricordando che le prime word della ExpandedKey sono esattamente le 4, 6 o 8 word della CipherKey (a seconda della dimensione di quest'ultima), si può fare un'osservazione interessante: quando lo stato è di dimensione minore o uguale alla CipherKey (il che avviene sempre in AES, che fissa il blocco a 128 bit, ma non necessariamente avviene nella versione generale di Rijndael, che supporta tutte le combinazioni di dimensioni del blocco e della chiave), l'operazione AddRoundKey iniziale viene eseguita usando esclusivamente word provenienti appunto dalla CipherKey, senza coinvolgere alcuna word calcolata dalla procedura KeyExpansion. Ciò rende tale operazione particolarmente efficace dal punto di vista della sicurezza.