4.10 Problemi al calcolatore

1. (Per chi si trova a proprio agio con la programmazione)

   (a) Scrivere un programma che esegua un round dell'algoritmo di tipo DES semplificato presentato nel Paragrafo 4.2.

   (b) Generare una stringa di bit di input e una chiave casuale. Calcolare il corrispondente testo cifrato quando si eseguono un round, due round, tre round e quattro round della struttura Feistel usando l'implementazione data in (a). Verificare che la procedura di decifrazione funziona in ogni caso.

   (c) Sia $E_K(M)$ la cifratura a quattro round mediante la chiave K. Provando tutte le 2^9 chiavi, mostrare che non esistono chiavi deboli per questo algoritmo di tipo DES semplificato. Si tenga presente che una chiave K è debole se cifrando un testo in chiaro due volte si torna al testo in chiaro, ossia se $E_K(E_K(M)) = M$ per ogni possibile M. (*Nota:* per ogni chiave K, bisogna trovare un M tale che $E_K(E_K(M)) \neq M$.)

   (d) Sia $E'_K(M)$ l'algoritmo di cifratura che si ottiene modificando l'algoritmo di cifratura $E_K(M)$ scambiando tra di loro la metà di sinistra e la metà di destra dopo i quattro round Feistel. Esistono chiavi deboli per questo algoritmo?

2. Usando l'implementazione di $E_K(M)$ data nel Problema al calcolatore 1(b), implementare la modalità operativa CBC per questo algoritmo di tipo DES semplificato.

   (a) Generare un messaggio di testo in chiaro formato da 48 bit e mostrare come si cifra e decifra usando CBC.

   (b) Se si hanno due testi in chiaro che differiscono nel 14-esimo bit, dire cosa accade ai corrispondenti testi cifrati.

Capitolo 5

Algoritmo AES: Rijndael

Nel 1997 il National Institute of Standards and Technology (NIST) richiese pubblicamente proposte di algoritmi tra i quali scegliere il sostituto di DES. Il nuovo algoritmo avrebbe dovuto consentire l'uso di chiavi di 128, 192 e 256 bit, avrebbe dovuto operare su blocchi di 128 bit di input e avrebbe dovuto lavorare su vari hardware di diverso tipo, dai processori a 8 bit che possono essere usati nelle *smart card* alle architetture a 32 bit comunemente usate nei personal computer. Inoltre avrebbe dovuto essere veloce e crittograficamente robusto. Nel 1998 fu chiesto alla comunità crittografica di esprimersi su 15 algoritmi candidati. Furono scelti cinque finalisti: MARS (dell'IBM), RC6 (degli RSA Laboratories), Rijndael (di Joan Daemen e Vincent Rijmen), Serpent (di Ross Anderson, Eli Biham e Lars Knudsen) e Twofish (di Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall e Niels Ferguson). Alla fine, Rijndael fu scelto per essere l'algoritmo AES (*Advanced Encryption Standard*, standard avanzato di cifratura). Gli altri quattro algoritmi sono anch'essi molto robusti ed è probabile che verranno usati in molti crittosistemi futuri.
Come per i cifrari a blocchi, Rijndael può essere usato in molte modalità, come per esempio ECB, CBC, CFB, OFB e CTR (si veda il Paragrafo 4.5).
Prima di passare all'algoritmo, daremo una risposta a una domanda fondamentale: come si pronuncia Rijndael? Citiamo la pagina Web dei due autori.

> Se siete olandesi, fiamminghi, indonesiani, surinamesi o sudafricani, si pronuncia come pensate debba essere pronunciato. Altrimenti, lo potete pronunciare come "Reign Dahl", "Rain Doll", "Rhine Dahl". Non siamo pignoli. Almeno, finché lo pronunciate diversamente da "Region Deal."[1]

[1] Il lettore italiano lo può pronunciare "Reindòl". In caso di dubbio, la pagina web di Vincent Rijmen contiene un file audio con la pronuncia corretta. (*N.d.Rev.*)

5.1 Algoritmo fondamentale

Rijndael è progettato per usare chiavi di 128, 192 e 256 bit. Per semplicità, ci limiteremo al caso di 128 bit. Prima verrà data una breve panoramica dell'algoritmo e poi verranno descritte in maggiore dettaglio le varie componenti.
L'algoritmo è formato da 10 round (12 round se la chiave è di 192 bit, 14 round se la chiave è di 256 bit). Ogni round ha una chiave di round, derivata dalla chiave originale. C'è anche una chiave relativa allo 0-esimo round, che coincide con la chiave originale. Un round inizia con un input di 128 bit e termina producendo un output di 128 bit. I round sono formati dai seguenti quattro passi fondamentali, chiamati **layer**.

1. **Trasformazione SubBytes (SB)**: questo passo non lineare serve per resistere agli attacchi di crittanalisi differenziale e lineare.
2. **Trasformazione ShiftRows (SR)**: questo passo di rimescolamento lineare produce diffusione dei bit su round multipli.
3. **Trasformazione MixColumns (MC)**: questo passo ha uno scopo simile a ShiftRows.
4. **AddRoundKey (ARK)**: la chiave di round è sommata XOR con il risultato del layer precedente.

Pertanto un round è dato da

$$\rightarrow \boxed{\text{SubBytes}} \rightarrow \boxed{\text{ShiftRows}} \rightarrow \boxed{\text{MixColumns}} \rightarrow \boxed{\text{AddRoundKey}} \rightarrow .$$

Mettendo tutto insieme, si ha la seguente (si veda anche la Figura 5.1).

Cifratura Rijndael

1. ARK, che usa la chiave del round 0.
2. Nove round di SB, SR, MC, ARK, che usano le chiavi dei round da 1 a 9.
3. Un round finale: SB, SR, ARK, che usa la chiave del decimo round.

Il round finale usa i passi SubBytes, ShiftRows, AddRoundKey e tralascia MixColumns (questa omissione verrà spiegata nel paragrafo sulla decifrazione).
L'output di 128 bit è il blocco di testo cifrato.

5.2 Layer AES

Descriviamo ora i vari passi più dettagliatamente. I 128 bit di input sono raggruppati in 16 byte di 8 bit:

$$a_{0,0},\ a_{1,0},\ a_{2,0},\ a_{3,0},\ a_{0,1},\ a_{1,1},\ \ldots,\ a_{3,3}$$

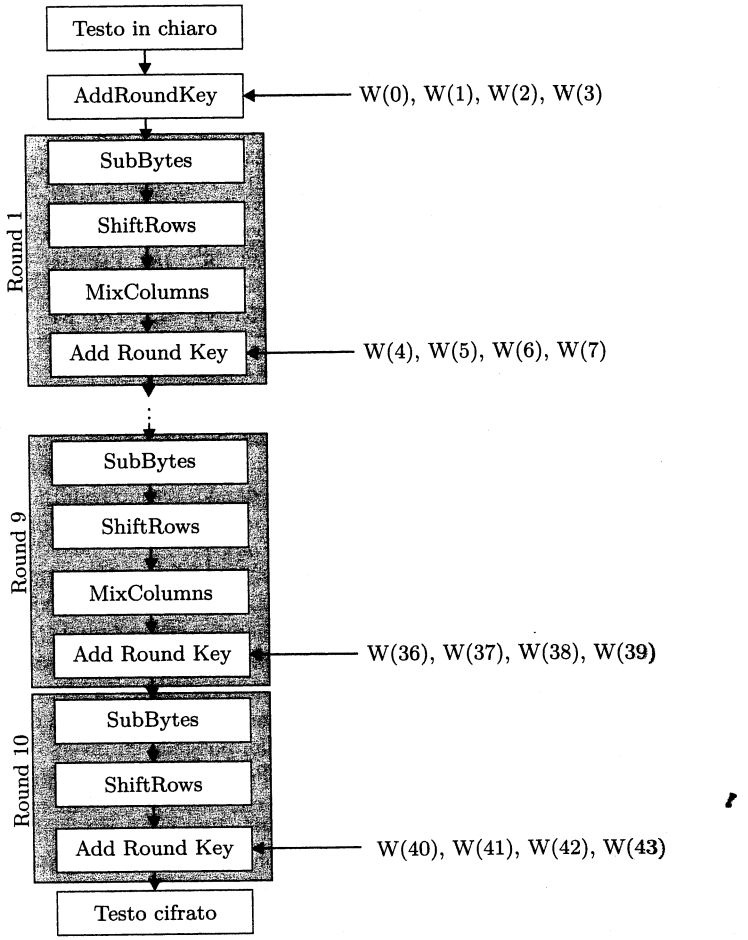

Figura 5.1 L'algoritmo AES-Rijndael.

S-Box															
99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

Tabella 5.1 L'S-Box per Rijndael.

che vengono disposti in una matrice 4×4 nel modo seguente

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}.$$

Nel seguito a volte avremo bisogno di lavorare con il campo finito $GF(2^8)$ (si veda il Paragrafo 3.11). Per il momento, basta ricordare che gli elementi di $GF(2^8)$ sono byte (formati da 8 bit) che possono essere sommati mediante la somma XOR e che possono essere opportunamente moltiplicati (in modo che il prodotto di due byte sia ancora un byte). Ogni byte b, diverso dal byte nullo, possiede un inverso moltiplicativo, ossia esiste un byte b' tale che $b \cdot b' = 00000001$. Poiché si possono eseguire operazioni aritmetiche sui byte, si può lavorare con matrici che hanno per elementi dei byte.
Si tenga presente che la costruzione di $GF(2^8)$ dipende dalla scelta di un polinomio irriducibile di grado 8 e che per Rijndael il polinomio scelto è $X^8 + X^4 + X^3 + X + 1$. Questo è anche il polinomio usato negli esempi del Paragrafo 3.11. Altre scelte per questo polinomio dovrebbero presumibilmente portare ad algoritmi altrettanto buoni.

5.2.1 Trasformazione SubBytes

In questo passo ognuno dei byte nella matrice viene trasformato in un altro byte mediante la Tabella 5.1, chiamata S-box.
Scritto il byte come una stringa di 8 bit *abcdefgh*, si considera l'elemento che si trova sulla riga *abcd* e sulla colonna *efgh* (le righe e le colonne sono numerate da 0 a 15). Questo elemento, una volta convertito in binario, è l'output. Per esempio, se il byte di

input è 10001011, si considera la riga 8 (la nona riga) e la colonna 11 (la dodicesima colonna). L'elemento è 61, ossia 111101 in binario; questo è l'output dell'S-box.
L'output di SubBytes è ancora una matrice di byte 4×4:

$$\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}.$$

5.2.2 Trasformazione ShiftRows

Le quattro righe della matrice precedente sono fatte scorrere ciclicamente a sinistra rispettivamente di 0, 1, 2 e 3:

$$\begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{pmatrix}.$$

5.2.3 Trasformazione MixColumns

Considerando ogni byte come un elemento di $GF(2^8)$, come nel Paragrafo 3.11, si ha che l'output del passo ShiftRows è una matrice $(c_{i,j})$ di tipo 4×4, con elementi in $GF(2^8)$. Si moltiplica questa matrice per una matrice, sempre a coefficienti in $GF(2^8)$, in modo da ottenere l'output $(d_{i,j})$:

$$\begin{pmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{pmatrix} \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix}$$

$$= \begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix}.$$

5.2.4 AddRoundKey

La chiave di round, derivata dalla chiave in un modo che descriveremo più avanti, è formata da 128 bit, disposti in una matrice $(k_{i,j})$ di tipo 4×4, formata da byte. Questa matrice è sommata XOR con l'output del passo MixColumns:

$$\begin{pmatrix} d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\ d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \end{pmatrix} \oplus \begin{pmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{pmatrix} = \begin{pmatrix} e_{0,0} & e_{0,1} & e_{0,2} & e_{0,3} \\ e_{1,0} & e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,0} & e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,0} & e_{3,1} & e_{3,2} & e_{3,3} \end{pmatrix}$$

Questo è l'output finale del round.

5.2.5 Chiavi di round

La chiave originale è formata da 128 bit, disposti in una matrice di byte 4×4. Questa matrice è ampliata aggiungendo 40 nuove colonne. Esse sono generate ricorsivamente a partire dalle prime quattro colonne $W(0)$, $W(1)$, $W(2)$, $W(3)$. Se le colonne fino a $W(i-1)$ sono già state definite, allora, se i non è un multiplo di 4, si pone

$$W(i) = W(i-4) \oplus W(i-1),$$

e se i è un multiplo di 4, si pone

$$W(i) = W(i-4) \oplus T(W(i-1)),$$

dove $T(W(i-1))$ è la trasformata di $W(i-1)$ ottenuta nel modo seguente. Gli elementi a, b, c, d della colonna $W(i-1)$ vengono fatti scorrere ciclicamente in modo da avere b, c, d, a. Poi ognuno di questi byte viene sostituito con il corrispondente elemento nell'S-box del passo SubBytes, ottenendo così 4 byte e, f, g, h. Infine, si calcola la costante di round

$$r(i) = 00000010^{(i-4)/4}$$

in $GF(2^8)$ (si tenga presente che siamo nel caso in cui i è un multiplo di 4). Allora $T(W(i-1))$ è il vettore colonna

$$(e \oplus r(i), f, g, h).$$

La **chiave di round** per l'i-esimo round è formata dalle colonne

$$W(4i),\ W(4i+1),\ W(4i+2),\ W(4i+3).$$

5.2.6 La costruzione dell'S-Box

Anche se è implementato come una tabella associativa, l'S-box possiede una semplice descrizione matematica. Si parte con un byte $x_7x_6x_5x_4x_3x_2x_1x_0$, dove ogni x_i è un bit binario. Si calcola il suo inverso in $GF(2^8)$, come nel Paragrafo 3.11. Se il byte è 00000000, l'inverso non esiste e quindi si considera il byte 00000000 stesso. Il byte $y_7y_6y_5y_4y_3y_2y_1y_0$ così ottenuto rappresenta un vettore colonna di lunghezza 8, con in cima il bit y_0. Si moltiplica per una matrice e poi si somma con il vettore colonna (1,1,0,0,0,1,1,0):

$$\begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{pmatrix} + \begin{pmatrix} 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0 \end{pmatrix} = \begin{pmatrix} z_0\\ z_1\\ z_2\\ z_3\\ z_4\\ z_5\\ z_6\\ z_7 \end{pmatrix}.$$

Il byte $z_7z_6z_5z_4z_3z_2z_1z_0$, corrispondente al vettore $(z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7)$ ottenuto, è l'elemento nell'S-box.

Per esempio, se si parte con il byte 11001011, allora si considera il suo inverso 00000100 in $GF(2^8)$ (come si è calcolato nel Paragrafo 3.11) e poi si calcola

$$\begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1 \end{pmatrix} \begin{pmatrix} 0\\ 0\\ 1\\ 0\\ 0\\ 0\\ 0\\ 0 \end{pmatrix} + \begin{pmatrix} 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0 \end{pmatrix} = \begin{pmatrix} 1\\ 1\\ 1\\ 1\\ 1\\ 0\\ 0\\ 0 \end{pmatrix}.$$

In questo modo si ottiene il byte 00011111. I primi 4 bit del byte di partenza (1100) rappresentano 12 in binario mentre gli ultimi 4 bit (1011) rappresentano 11 in binario. Si somma 1 a ognuno di questi numeri (poiché la prima riga e la prima colonna sono numerate con 0) e si guarda nella 13-esima riga e nella 12-esima colonna dell'S-box. L'elemento è 31, che in binario è 00011111.
Per ottenere la non linearità si è usata la funzione $x \mapsto x^{-1}$. Tuttavia, per evitare gli attacchi che potrebbero essere consentiti dalla sua semplicità, essa viene combinata con la moltiplicazione per una matrice e poi con la somma di un vettore, nel modo descritto qui sopra. La matrice è stata scelta principalmente per la sua forma semplice (si noti il modo in cui le righe sono fatte scorrere l'una rispetto all'altra). Il vettore è stato scelto in modo che nessun input coincida con il corrispondente output dell'S-box o con il complementare del corrispondente output dell'S-box (dove la complementazione consiste nel cambiare ogni 1 in 0 e ogni 0 in 1).

5.3 Decifrazione

Ognuno dei passi SubBytes, ShiftRows, MixColumns e AddRoundKey è invertibile.

1. L'inverso di SubBytes, chiamato **InvSubBytes**, è un'altra tabella di consultazione.

2. L'inverso di ShiftRows, chiamato **InvShiftRows**, si ottiene facendo scorrere le righe a destra invece che a sinistra.

3. L'inverso di MixColumns esiste perchè la matrice 4×4 usata in MixColumns è invertibile. La trasformazione **InvMixColumns** è data dalla moltiplicazione per la matrice

$$\begin{pmatrix} 00001110 & 00001011 & 00001101 & 00001001\\ 00001001 & 00001110 & 00001011 & 00001101\\ 00001101 & 00001001 & 00001110 & 00001011\\ 00001011 & 00001101 & 00001001 & 00001110 \end{pmatrix}.$$

4. AddRoundKey coincide con il suo inverso.

La cifratura Rijndael è formata dai passi

ARK
SB, SR, MC, ARK
...
SB, SR, MC, ARK
SB, SR, ARK.

Si ricordi che MC non compare nell'ultimo round.
Per decifrare, si devono eseguire, in ordine inverso, le operazioni inverse di ciascuno di questi passi. Si ha così la seguente versione preliminare della decifrazione:

ARK, ISR, ISB
ARK, IMC, ISR, ISB
...
ARK, IMC, ISR, ISB
ARK.

Questa procedura verrà riformulata in modo da rendere la decifrazione più simile alla cifratura.

- Si osservi che applicare SB e poi SR è la stessa cosa che applicare prima SR e poi SB. Ciò è dovuto al fatto che SB agisce su un byte alla volta e SR permuta i byte. Analogamente, anche l'ordine di applicazione dei passi inversi ISR e ISB può essere invertito.
- Sarebbe utile poter invertire l'ordine di applicazione anche per ARK e IMC. Anche se questo non è possibile, si può però procedere nel modo seguente. Si applica MC e poi ARK a una matrice $(c_{i,j})$:

$$(c_{i,j}) \quad \rightarrow \quad (m_{i,j})(c_{i,j}) \quad \rightarrow \quad (e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j}),$$

dove $(m_{i,j})$ è la matrice 4×4 che compare in MixColumns e $(k_{i,j})$ è la matrice delle chiavi di round. La trasformazione inversa si ottiene risolvendo l'equazione $(e_{i,j}) = (m_{i,j})(c_{i,j}) \oplus (k_{i,j})$ rispetto a $(c_{i,j})$ in termini di $(e_{i,j})$, ossia $(c_{i,j}) = (m_{i,j})^{-1}(e_{i,j}) \oplus (m_{i,j})^{-1}(k_{i,j})$. Pertanto si ha il processo

$$(e_{i,j}) \quad \rightarrow \quad (m_{i,j})^{-1}(e_{i,j}) \quad \rightarrow \quad (m_{i,j})^{-1}(e_{i,j}) \oplus (k'_{i,j}),$$

dove $(k'_{i,j}) = (m_{i,j})^{-1}(k_{i,j})$. La prima freccia è semplicemente InvMixColumns applicata a $(e_{i,j})$. Se definiamo **InvAddRoundKey** come la somma XOR con $(k'_{i,j})$, allora si ha che l'inverso di "MC e poi ARK" è "IMC e poi IARK". Quindi, nella precedente sequenza di decifrazione, i passi "ARK e poi IMC" possono essere sostituiti con i passi "IMC e poi IARK". Di conseguenza la decifrazione è data da

ARK, ISB, ISR
IMC, IARK, ISB, ISR
...
IMC, IARK, ISB, ISR
ARK.

In conclusione, si ha la seguente versione finale.

Decifrazione Rijndael

1. ARK, che usa la decima chiave di round
2. Nove round di ISB, ISR, IMC, IARK, che usano le chiavi di round da 9 a 1
3. Un round finale: ISB, ISR, ARK, che usa la chiave dello 0-esimo round

In questo modo, la decifrazione ha essenzialmente la medesima struttura della cifratura, dove però SubBytes, ShiftRows e MixColumns sono sostituiti dai loro inversi e AddRoundKey è sostituito con InvAddRoundKey, tranne che nel passo iniziale e finale. Naturalmente, le chiavi di round sono usate nell'ordine inverso, cosicché il primo passo ARK usa la chiave del decimo round e l'ultimo passo ARK usa la chiave dello 0-esimo round.
Quanto precede mostra perché il passo MixColumns non compare nell'ultimo round. Se comparisse, allora la cifratura partirebbe con ARK, SB, SR, MC, ARK, ... e finirebbe con ARK, SB, SR, MC, ARK. Quindi l'inizio della decifrazione sarebbe (dopo il riordino) IMC, IARK, ISB, ISR, ... Questo significa che la decifrazione avrebbe all'inizio un passo IMC inutile con l'unico effetto di rallentare l'algoritmo.
Equivalentemente, la cifratura è formata da un passo ARK iniziale, seguito da una successione di mezzi round alternanti

(SB, SR), (MC, ARK), (SB, SR), ..., (MC, ARK), (SB, SR),

seguita da un passo ARK finale. La decifrazione è un ARK iniziale, seguito da una susseccione di mezzi round alternanti

(ISB, ISR), (IMC, IARK), (ISB, ISR), ..., (IMC, IARK), (ISB, ISR),

seguita da un ARK finale. Da qui si vede che un MC finale non si inserirebbe in modo naturale in alcuno dei mezzi round. Di conseguenza, è naturale lasciarlo fuori.
Sui processori a 8 bit, la decifrazione non è così veloce come la cifratura. Ciò è dovuto al fatto che gli elementi della matrice 4×4 di InvMixColumns sono più complessi di quelli della matrice di MixColumns e questo è sufficiente a rendere la decifrazione più lunga della cifratura di circa il 30%. Tuttavia, in molte applicazioni, la decifrazione non è necessaria, come per esempio quando si usa una modalità CFB (si veda il Paragrafo 4.5). Quindi questo problema non è considerato un grosso svantaggio.
Il fatto che la cifratura e la decifrazione non sono processi identici porta a pensare che non ci siano chiavi deboli, contrariamente a quanto accade con DES (si veda l'Esercizio 5 del Capitolo 4) e con molti altri algoritmi.

5.4 Considerazioni di progetto

L'algoritmo Rijndael non è un sistema Feistel (si vedano i Paragrafi 4.1 e 4.2). In un sistema Feistel, metà dei bit sono spostati ma non modificati durante ogni round. In Rijndael, tutti i bit sono trattati in modo uniforme, con l'effetto di diffondere i bit di input più rapidamente. Si può dimostrare che sono sufficienti due round per avere una diffusione totale (ognuno dei 128 bit di output dipende da ognuno dei 128 bit di input).

L'S-box è stato costruito in un modo algebrico esplicito e semplice per evitare ogni sospetto di *trapdoor* nell'algoritmo. Il desiderio era quello di evitare i misteri che avevano circondato gli S-box di DES. L'S-box di Rijndael è altamente non lineare, basandosi sulla funzione $x \mapsto x^{-1}$ in $GF(2^8)$. Resiste in modo eccellente alla crittanalisi differenziale e lineare e ai metodi più recenti chiamati attacchi a interpolazione.

Il passo ShiftRows è stato aggiunto per resistere a due attacchi sviluppati di recente, ossia i differenziali troncati e l'attacco Square (Square era un predecessore di Rijndael).

Il passo MixColumns produce diffusione tra i byte. Un cambiamento in un byte di input in questo passo porta sempre a un cambiamento in tutti e quattro i byte di output. Se si modificano due byte di input, si modificano almeno tre byte di output.

L'algoritmo di generazione delle chiavi di round coinvolge un rimescolamento non lineare dei bit della chiave, in quanto utilizza l'S-box. Il rimescolamento è progettato in modo da resistere agli attacchi in cui si parte dalla conoscenza di una parte della chiave per cercare di dedurre i restanti bit. Inoltre, punta ad assicurare che due chiavi distinte non abbiano mai troppe chiavi di round in comune. Le costanti di round sono usate per eliminare simmetrie nel processo di cifratura, rendendo ogni round differente.

Il numero dei round è 10 perché esistono attacchi migliori della forza bruta solo fino a sei round. Nessun attacco noto batte la forza bruta per sette o più round. Si è pensato che quattro round in più fossero sufficienti per avere un largo margine di sicurezza. Naturalmente, se fosse necessario, il numero di round potrebbe essere facilmente aumentato.

5.5 Esercizi

1. La chiave per il round 0 in AES è formata da 128 bit, tutti uguali a 0.

   (a) Mostrare che la chiave per il primo round è $W(4)$, $W(5)$, $W(6)$, $W(7)$, dove

   $$W(4) = W(5) = W(6) = W(7) = \begin{pmatrix} 01100100 \\ 01100011 \\ 01100011 \\ 01100011 \end{pmatrix}.$$

   (b) Mostrare che $W(8) = W(10) \neq W(9) = W(11)$ (*suggerimento:* lo si può fare senza calcolare $W(8)$ esplicitamente).

2. La chiave per il round 0 in AES è formata da 128 bit, tutti uguali a 1.

   (a) Mostrare che la chiave per il primo round è $W(4)$, $W(5)$, $W(6)$, $W(7)$, dove

   $$W(4) = W(6) = \begin{pmatrix} 11101000 \\ 11101001 \\ 11101001 \\ 11101001 \end{pmatrix}, \quad W(5) = W(7) = \begin{pmatrix} 00010111 \\ 00010110 \\ 00010110 \\ 00010110 \end{pmatrix}.$$

   Si noti che $W(5) = \overline{W(4)}$ = complemento di $W(5)$ (il complemento può essere ottenuto sommando XOR con una stringa tutta di 1).

   (b) Mostrare che $W(10) = \overline{W(8)}$ e che $W(11) = \overline{W(9)}$ (*suggerimenti:* $W(5) \oplus W(6)$ è una stringa tutta di 1; inoltre, potrebbe essere utile considerare la relazione $\overline{A} \oplus B = \overline{A \oplus B}$).

3. Sia $f(x)$ una funzione che trasforma stringhe binarie (di fissata lunghezza N) in stringhe binarie. Per gli scopi di questo problema, diremo che $f(x)$ ha la *proprietà delle differenze uguali* se comunque scelte quattro stringhe binarie x_1, x_2, x_3, x_4 di lunghezza N tali che $x_1 \oplus x_2 = x_3 \oplus x_4$, allora

   $$f(x_1) \oplus f(x_2) = f(x_3) \oplus f(x_4).$$

   (a) Mostrare che se $\alpha, \beta \in GF(2^8)$ e $f(x) = \alpha x + \beta$ per ogni $x \in GF(2^8)$, allora $f(x)$ ha la proprietà delle differenze uguali.

   (b) Mostrare che la trasformazione ShiftRows, la trasformazione MixColumns e la AddRoundKey hanno la proprietà delle differenze uguali.

4. **(a)** Si eliminino tutti i passi SubBytes dall'algoritmo AES. Mostrare che la cifratura AES che ne risulta possiede la proprietà delle differenze uguali definita nell'Esercizio 3.

   (b) Nella situazione della parte (a), in cui si sono eliminati tutti i passi SubBytes, siano x_1 e x_2 due blocchi di testo in chiaro da 128 bit e siano $E(x_1)$ e $E(x_2)$ le loro cifrature in questo schema AES modificato. Mostrare che $E(x_1) \oplus E(x_2)$ è uguale alla cifratura di $x_1 \oplus x_2$ fatta solo mediante le trasformazioni ShiftRows e MixColumns (cioé, senza usare la AddRoundKey e la trasformazione SubBytes). In particolare, $E(x_1) \oplus E(x_2)$ non dipende dalla chiave.

   (c) Nella situazione della parte (a), Eva conosce x_1 ed $E(x_1)$ per qualche stringa x di 128 bit. Dire come può decifrare ogni messaggio $E(x_2)$ (la soluzione dovrebbe essere molto più veloce dell'uso della forza bruta o del fare una lista di tutte le cifrature) (*osservazione:* questo mostra che la trasformazione SubBytes è necessaria per prevenire la proprietà delle differenze uguali. Si veda anche l'Esercizio 5).

5. Siano $x_1 = 00000000$, $x_2 = 00000001$, $x_3 = 00000010$, $x_4 = 00000011$. Sia $SB(x)$ la trasformazione SubBytes di x. Mostrare che

   $$SB(x_1) \oplus SB(x_2) = 00011111 \neq 00001100 = SB(x_3) \oplus SB(x_4).$$

Concludere che la trasformazione SubBytes non è una funzione affine (cioé una funzione della forma $\alpha x + \beta$) da $GF(2^8)$ a $GF(2^8)$ (*suggerimento:* si veda l'Esercizio 3(a)).

Capitolo 6

Algoritmo RSA

6.1 Algoritmo RSA

Alice vuole inviare un messaggio a Bob. Il problema è che non hanno mai avuto alcun contatto in precedenza e che non vogliono aspettare che un corriere consegni una chiave. In questo modo, tutte le informazioni che Alice invia a Bob possono essere carpite da un osservatore malintenzionato come Eva. In realtà, è ancora possibile inviare un messaggio in modo che possa essere letto da Bob, ma non da Eva, anche se una cosa del genere sarebbe impossibile con tutti i metodi discussi in precedenza. Alice dovrebbe inviare una chiave, che Eva potrebbe intercettare e usare per decifrare ogni ulteriore messaggio. Uno schema di questo tipo, chiamato **crittosistema a chiave pubblica**, fu inizialmente proposto da Diffie e Hellman nel loro classico articolo [Diffie-Hellman]. Tuttavia, non avevano ancora un'implementazione pratica (anche se presentarono una procedura alternativa di scambio della chiave che funzionava su canali pubblici; si veda il Paragrafo 7.4). Negli anni che seguirono, vennero proposti vari metodi. Quello che ha avuto maggior successo, ossia l'algoritmo RSA, si basa sulla difficoltà di fattorizzare gli interi in fattori primi e fu proposto da Rivest, Shamir e Adleman nel 1977.
Si è a lungo affermato che le agenzie crittografiche governative avessero scoperto l'algoritmo RSA parecchi anni prima, anche se, per motivi di sicurezza, non ne avevano lasciato traccia alcuna. Infine, nel 1997 il CESG, un'agenzia crittografica inglese, rilasciò dei documenti che mostravano che nel 1970 James Ellis aveva scoperto la crittografia a chiave pubblica e che nel 1973 Clifford Cocks aveva scritto un documento interno in cui veniva descritta una versione dell'algoritmo RSA in cui l'esponente di cifratura e (si veda la discussione che segue) era uguale al modulo n.
Ecco come funziona l'algoritmo RSA. Bob sceglie due primi p e q grandi e distinti e li moltiplica per formare il numero

$$n = pq.$$