Capitolo 4

Algoritmo DES

4.1 Introduzione

Nel 1973 il National Bureau of Standards (NBS), che in seguito diventerà il National Institute of Standards and Technology (NIST), [1] richiese pubblicamente un algoritmo crittografico che potesse diventare uno standard nazionale. Nel 1974 IBM propose un algoritmo chiamato LUCIFER. NBS lo inoltrò alla National Security Agency, che lo analizzò e che, dopo alcune modifiche, lo restituì in una versione che era sostanzialmente l'algoritmo Data Encryption Standard (DES). Nel 1975 NBS pubblicò l'algoritmo DES con una licenza d'uso gratuito e nel 1977 lo adottò come standard ufficiale per la cifratura dei dati.
DES è stato ampiamente usato nel commercio elettronico, per esempio in ambito bancario. Se due banche vogliono scambiare dati, prima usano un metodo a chiave pubblica come RSA per trasmettere una chiave per DES, poi usano DES per trasmettere i dati. Questo metodo ha il vantaggio di essere molto veloce e ragionevolmente sicuro.
Dal 1975 DES è stato circondato da varie controversie: alcuni ritenevano che la chiave fosse troppo corta, altri erano preoccupati per il coinvolgimento della NSA, che avrebbe potuto, per esempio, inserire una *trapdoor* (letteralmente "scappatoia"), ovvero un punto debole segreto che avrebbe permesso loro di violare il sistema. Si è anche sospettato che la NSA avesse modificato l'algoritmo per prevenire l'eventualità che IBM avesse inserito una trapdoor in LUCIFER. In ogni caso, il modo in cui era stato ideato l'algoritmo rimase un mistero per molti anni.
Nel 1990 Eli Biham e Adi Shamir mostrarono come il loro metodo di crittoanalisi differenziale potesse essere usato per attaccare DES. L'algoritmo DES è strutturato in 16 round. La crittoanalisi differenziale sarebbe stata più efficiente della ricerca

[1] NIST è un'agenzia governativa degli Stati Uniti d'America che ha come compito quello di sviluppare metodi di misura, standard e tecnologie volte a promuovere l'innovazione, la competizione industriale e la sicurezza del commercio. (*N.d.Rev.*)

esaustiva tra tutte le possibili chiavi se l'algoritmo avesse usato al più 15 round[2]. Questo fa supporre che gli ideatori di DES fossero consapevoli di questo tipo di attacco. Alcuni anni dopo, IBM rilasciò alcuni dettagli dei criteri del progetto, che mostrarono come effettivamente avessero costruito il sistema in modo da essere resistente alla crittoanalisi differenziale. Questo chiarì almeno alcuni dei misteri che circondavano l'algoritmo.
DES è durato parecchio tempo, anche se ormai è diventato obsoleto. Infatti attacchi di forza bruta (vedi il Paragrafo 4.6), benché costosi, ora lo possono violare. Pertanto nel 2000 NIST lo ha sostituito con un nuovo sistema. Nonostante tutto questo, vale la pena di studiare DES poiché rappresenta una classe molto diffusa di algoritmi ed è stato uno degli algoritmi crittografici maggiormente usati nella storia recente.
DES è un cifrario a blocchi, ossia spezza il testo in chiaro in blocchi di 64 bit e cifra ogni blocco separatamente. L'effettivo processo di cifratura è comunemente chiamato **sistema Feistel**, in onore di Horst Feistel, uno dei componenti della squadra che sviluppò LUCIFER in IBM. Nel prossimo paragrafo, verrà descritto un semplice algoritmo che possiede molte delle caratteristiche dei sistemi di questo tipo e che è sufficientemente compatto per poter essere usato come esempio. Nel Paragrafo 4.3 si mostrerà come attaccare questo semplice sistema mediante la crittoanalisi differenziale. Nel Paragrafo 4.4 si presenterà l'algoritmo DES e nel Paragrafo 4.5 verranno descritti alcuni modi in cui può essere implementato. Infine, nel Paragrafo 4.6, verranno descritti i recenti progressi nella crittoanalisi di DES.
Per una discussione esaustiva sui cifrari a blocchi, si veda [Schneier].

4.2 Algoritmo di tipo DES semplificato

Poiché l'algoritmo DES non presta a essere usato per gli esempi, in questo paragrafo verrà presentato un algoritmo che possiede molte delle sue caratteristiche, ma che è molto più compatto. Come DES, questo algoritmo è un cifrario a blocchi. Poiché i blocchi sono cifrati separatamente, in questa discussione si supporrà che l'intero messaggio sia formato da un solo blocco.
Il messaggio è formato da 12 bit ed è scritto nella forma L_0R_0, dove L_0 è formato dai primi 6 bit e R_0 dagli ultimi 6 bit. La chiave K è di 9 bit. L'i-esimo round dell'algoritmo trasforma l'input $L_{i-1}R_{i-1}$ nell'output L_iR_i usando una chiave K_i di 8 bit derivata da K.
La parte principale del processo di cifratura è una funzione $f(R_{i-1},K_i)$ che prende in input i 6 bit di R_{i-1} e gli 8 bit di K_i e produce un'output di 6 bit, come sarà descritto in seguito.
L'output dell'i-esimo round è definito come segue:

$$L_i = R_{i-1} \qquad \text{e} \qquad R_i = L_{i-1} \oplus f(R_{i-1},K_i),$$

dove $\oplus$ indica l'operazione XOR, ossia la somma bit a bit modulo 2.
Questa operazione, mostrata in Figura 4.1, viene ripetuta per n round e alla fine genera il testo cifrato L_nR_n.

[2]Biham e Shamir hanno pubblicato un attacco di crittoanalisi differenziale più efficiente della ricerca esaustiva anche contro DES a 16 round.

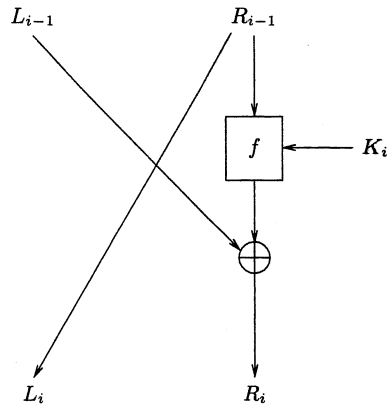

Figura 4.1 Un round di un sistema Feistel.

Per decifrare, si parte da L_nR_n e si scambiano[3] la parte di sinistra con quella di destra in modo da ottenere R_nL_n. A questo punto si usa la stessa procedura di prima, ma applicando le chiavi K_i in ordine inverso: $K_n,\ldots,K_1$. Il primo passo prende R_nL_n e genera l'output

$$[L_n] \quad [R_n \oplus f(L_n,K_n)].$$

Dalla procedura di cifratura si ha $L_n = R_{n-1}$ e $R_n = L_{n-1} \oplus f(R_{n-1},K_n)$. Quindi

$$\begin{aligned}[L_n] \quad [R_n \oplus f(L_n,K_n)] &= [R_{n-1}] \quad [L_{n-1} \oplus f(R_{n-1},K_n) \oplus f(L_n,K_n)] \\ &= [R_{n-1}] \quad [L_{n-1}].\end{aligned}$$

L'ultima uguaglianza segue dal fatto che $f(R_{n-1},K_n) \oplus f(L_n,K_n) = 0$, essendo $L_n = R_{n-1}$. Analogamente, il secondo passo della decifrazione trasforma $R_{n-1}L_{n-1}$ in $R_{n-2}L_{n-2}$. Continuando in questo modo, il processo di decifrazione porta a R_0L_0. Scambiando tra di loro la metà di sinistra e la metà di destra, si ottiene il testo in chiaro originale L_0R_0.
Il processo di decifrazione è essenzialmente lo stesso del processo di cifratura. Si tratta semplicemente di scambiare il blocco di sinistra con il blocco di destra e usare le chiavi K_i in ordine inverso. Sia chi invia il messaggio sia chi lo riceve usa la stessa chiave e usa la stessa macchina (anche se chi riceve deve invertire l'input di sinistra con quello di destra).
Per il momento non si è ancora detto nulla sulla funzione f. In realtà, nelle procedure viste in precedenza si può usare una funzione qualunque, anche se alcune funzioni f forniscono una sicurezza molto maggiore di altre. L'algoritmo DES usa una funzione f costruita a partire da poche componenti. La prima componente è una funzione di espansione che trasforma un input di 6 bit in un output di 8 bit. La funzione usata nel nostro DES semplificato è descritta nella Figura 4.2.

[3]Come si vedrà più avanti, questo scambio è già incluso nell'algoritmo di cifratura DES e non sarà effettuato quando si decifra.

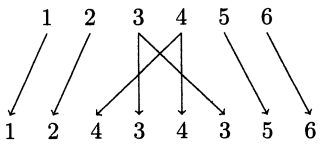

Figura 4.2 Funzione di espansione.

Il primo bit di input diventa il primo bit di output, il terzo bit di input diventa il quarto e il sesto bit di output e così via. Per esempio, 011001 è espanso in 01010101.
Le componenti principali sono funzioni di sostituzione chiamate S-box. Ne verranno usate due:

$$S_1 \quad \begin{bmatrix} 101 & 010 & 001 & 110 & 011 & 100 & 111 & 000 \\ 001 & 100 & 110 & 010 & 000 & 111 & 101 & 011 \end{bmatrix}$$

$$S_2 \quad \begin{bmatrix} 100 & 000 & 110 & 101 & 111 & 001 & 011 & 010 \\ 101 & 011 & 000 & 111 & 110 & 010 & 001 & 100 \end{bmatrix}.$$

L'input di un S-box è di 4 bit. Il primo bit indica quale riga deve essere usata: 0 per la prima riga, 1 per la seconda. Gli altri 3 bit rappresentano un numero binario che indica la colonna: 000 per la prima colonna, 001 per la seconda, ..., 111 per l'ultima colonna. L'output di un S-box è formato dai tre bit che appaiono nella posizione indicata. Così, per esempio, se S_1 riceve in input 1010, allora restituisce in output 110, ossia l'elemento che si trova sulla seconda riga e sulla terza colonna.
La chiave K è formata da 9 bit. La chiave K_i per l'i-esimo round di cifratura si ottiene usando 8 bit di K, a partire dall'i-esimo bit. Per esempio, se $K = 010011001$ si ha $K_4 = 01100101$ (poiché dopo 5 bit si raggiunge la fine di K, gli ultimi 2 bit si ottengono dall'inizio di K).
Consideriamo infine la funzione $f(R_{i-1},K_i)$. L'input R_{i-1} è formato da 6 bit, che vengono poi espansi a 8 bit dalla funzione di espansione. Il risultato è sommato XOR con K_i per generare altri 8 bit. I primi 4 bit sono dati in input a S_1 e gli ultimi 4 bit a S_2. Ciascun S-box restituisce un output di 3 bit. I due output sono concatenati, formando un numero di 6 bit, che dà $f(R_{i-1},K_i)$ (vedi Figura 4.3).
Se, per esempio, $R_{i-1} = 100110$ e $K_i = 01100101$, allora si ha

$$E(100110) \oplus K_i = 10101010 \oplus 01100101 = 11001111.$$

I primi 4 bit vengono elaborati mediante S_1 e gli ultimi 4 bit mediante S_2. L'elemento sulla seconda riga e sulla quinta colonna di S_1 è 000. L'elemento sulla seconda riga e sull'ultima colonna di S_2 è 100. Concatenando questi valori si ottiene $f(R_{i-1},K_i) = 000100$.
A questo punto è possibile descrivere ciò che accade in un round. Supponiamo che l'input sia

$$L_{i-1}R_{i-1} = 011100100110$$

e che $K_i = 01100101$, come prima. Questo significa che $R_{i-1} = 100110$, come nell'esempio appena discusso. Quindi $f(R_{i-1},K_i) = 000100$. Sommando XOR questo elemento

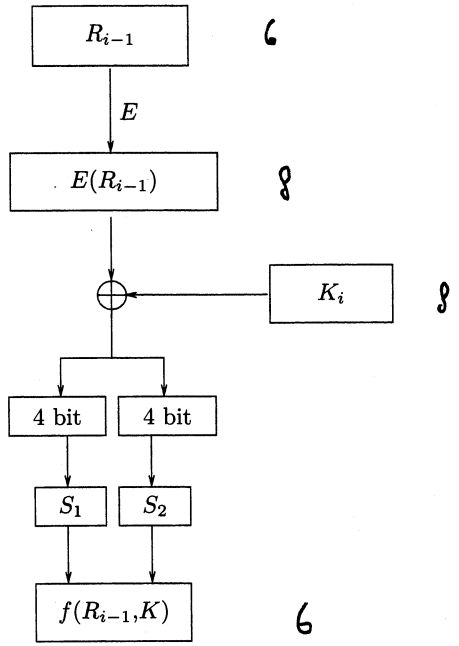

Figura 4.3 La funzione $f(R_{i-1},K_i)$.

con $L_{i-1} = 011100$ si ottiene $R_i = 011000$. Poiché $L_i = R_{i-1}$, si ha

$$L_iR_i = 100110011000.$$

L'output diventa l'input del round successivo.

4.3 Crittanalisi differenziale

Questo paragrafo è piuttosto tecnico e può essere saltato a una prima lettura.

La crittoanalisi differenziale fu introdotta da Biham e Shamir introno al 1990, anche se probabilmente era già nota agli ideatori di DES in IBM e alla NSA. L'idea è di calcolare le differenze nei testi cifrati corrispondenti a coppie opportunamente scelte di testi in chiaro e, da queste, dedurre informazioni sulla chiave. Si noti che la differenza tra due stringhe di bit può essere trovata facendone la somma XOR. Poiché la chiave è introdotta mediante la somma XOR con $E(R_{i-1})$, considerando la somma XOR di due input si può eliminare l'effetto della chiave a questo livello e quindi si può eliminare parte della casualità introdotta dalla chiave. Come vedremo, questo permette di dedurre informazioni sul possibile valore della chiave.

4.3.1 Crittannalisi differenziale a tre round

Anche se l'obiettivo finale è di descrivere un attacco al sistema DES semplificato a quattro round, conviene iniziare ad analizzare il caso in cui si usano solo tre round. Pertanto inizieremo con L_1R_1 invece che con L_0R_0.
La situazione è la seguente. Si è ottenuto l'accesso a un dispositivo di cifratura a tre round. Si conosce completamente il funzionamento interno dell'algoritmo di cifratura (gli S-box e così via), ma non si conosce la chiave, che si vuole trovare con un attacco di tipo testo in chiaro scelto. Si generano vari input L_1R_1 ottenendo i rispettivi output L_4R_4. Si ha

$$\begin{aligned} R_2 &= L_1 \oplus f(R_1,K_2) \\ L_3 &= R_2 = L_1 \oplus f(R_1,K_2) \\ R_4 &= L_3 \oplus f(R_3,K_4) = L_1 \oplus f(R_1,K_2) \oplus f(R_3,K_4). \end{aligned}$$

Si supponga di avere un altro messaggio $L_1^*R_1^*$ con $R_1 = R_1^*$. Per ogni i, poniamo $R_i' = R_i \oplus R_i^*$ e $L_i' = L_i \oplus L_i^*$. Allora $L_i'R_i'$ è la "differenza" (o somma, visto che si sta lavorando modulo 2) di L_iR_i e $L_i^*R_i^*$. Il precedente calcolo applicato ad $L_1^*R_1^*$ fornisce una formula per R_4^*. Poiché si è fatta l'ipotesi che $R_1 = R_1^*$, si ha $f(R_1,K_2) = f(R_1^*,K_2)$. Di conseguenza, $f(R_1,K_2) \oplus f(R_1^*,K_2) = 0$ e

$$R_4' = R_4 \oplus R_4^* = L_1' \oplus f(R_3,K_4) \oplus f(R_3^*,K_4).$$

Questo può essere riscritto come

$$R_4' \oplus L_1' = f(R_3,K_4) \oplus f(R_3^*,K_4).$$

Infine, poiché $R_3 = L_4$ e $R_3^* = L_4^*$, si ha

$$R_4' \oplus L_1' = f(L_4,K_4) \oplus f(L_4^*,K_4).$$

Se si conosce la somma XOR degli input, ossia $L_1'R_1'$, e se si conoscono gli output L_4R_4 e $L_4^*R_4^*$, allora si conosce tutto in quest'ultima equazione, tranne K_4.
Passiamo ora ad analizzare gli input degli S-box usati per calcolare $f(L_4,K_4)$ e $f(L_4^*,K_4)$. Se si inizia con L_4, prima si espande e poi si somma XOR con K_4 per ottenere $E(L_4) \oplus K_4$, che sono i bit inviati a S_1 e S_2. Analogamente, L_4^* produce $E(L_4^*) \oplus K_4$. La somma XOR di questi è

$$E(L_4) \oplus E(L_4^*) = E(L_4 \oplus L_4^*) = E(L_4').$$

(La prima uguaglianza segue facilmente dalla descrizione bit a bit della funzione di espansione). Quindi si conosce:

1. la somma XOR degli input inviati ai due S-box (ossia, i primi 4 e gli ultimi 4 bit di $E(L_4')$);
2. la somma XOR dei due output (ossia, i primi 3 e gli ultimi 3 bit di $R_4' \oplus L_1'$).

Restringiamo l'attenzione a S_1 (l'analisi per S_2 è del tutto simile). È abbastanza veloce scorrere tutte le coppie di possibili input di 4 bit (ce ne sono solo 16) e vedere quali danno una delle somme XOR desiderate. Queste coppie possono essere calcolate una volta per tutte e annotate in una tabella.
Per esempio, supponiamo di avere la somma XOR degli input uguale a 1011 e di cercare una somma XOR degli output uguale a 100. Si possono scorrere le coppie di input (1011, 0000), (1010, 0001), (1001, 0010), ..., ognuna delle quali ha XOR uguale a 1011, ed esaminare le somme XOR dei corrispondenti output. Troviamo che le coppie (1010, 0001) e (0001, 1010) producono entrambe in output una somma XOR uguale a 100. Per esempio, 1010 significa che si deve considerare la seconda riga e la terza colonna di S_1, dove si ha 110. Mentre 0001 significa che si deve considerare la prima riga e la seconda colonna, dove si ha 010. La somma XOR in output è quindi $110 \oplus 010 = 100$.
L_4 e L_4^* sono noti. Supponiamo, per esempio, che $L_4 = 101110$ e $L_4^* = 000010$. Poiché $E(L_4) = 10111110$ e $E(L_4^*) = 00000010$, gli input di S_1 sono $1011 \oplus K_4^L$ e $0000 \oplus K_4^L$, dove K_4^L denota i 4 bit di sinistra di K_4. Se sappiamo che la somma XOR in uscita da S_1 è 100, allora $(1011 \oplus K_4^L, 0000 \oplus K_4^L)$ deve essere una delle coppie nella lista appena calcolata, ossia (1010, 0001) e (0001, 1010). Questo significa che $K_4^L = 0001$ o 1010.
Se si ripete questa procedura qualche altra volta, si dovrebbe riuscire a eliminare una delle due scelte per K_4 e pertanto determinare 4 bit di K. Analogamente, usando S_2, si troveranno altri 4 bit di K e quindi 8 dei 9 bit di K. L'ultimo bit può essere trovato provando entrambe le possibilità e guardando quale produce la stessa cifratura della macchina che si sta attaccando.
Ecco una sintesi della procedura (che, per semplicità di notazione, verrà descritta nel caso in cui entrambi gli S-box siano usati simultaneamente, anche se negli esempi sono usati separatamente).

1. Si considera la lista di coppie con somma XOR degli input uguale a $E(L_4')$ e con somma XOR degli output uguale a $R_4' \oplus L_1'$.
2. La coppia $\big(E(L_4) \oplus K_4, E(L_4^*) \oplus K_4\big)$ è in questa lista.
3. Si deducono le possibilità per K_4.
4. Si ripete finché non rimane una sola possibilità per K_4.

Esempio. Iniziamo con

$$L_1R_1 = 000111011011.$$

La macchina compie la cifratura in tre round usando la chiave $K = 001001101$, che non conosciamo ancora. Si ottiene (si osservi che iniziando con L_1R_1, si parte con la chiave traslata $K_2 = 01001101$)

$$L_4R_4 = 000011100101.$$

Se si inizia con

$$L_1^*R_1^* = 101110011011$$

(si noti che $R_1 = R_1^*$), si ottiene

$$L_4^* R_4^* = 100100011000.$$

Si ha $E(L_4) = 00000011$ e $E(L_4^*) = 10101000$. Gli input di S_1 hanno somma XOR uguale a 1010 e gli input di S_2 hanno somma XOR uguale a 1011. Poiché gli S-box hanno somma XOR in output $R_4' \oplus L_1' = 111101 \oplus 101001 = 010100$, la somma XOR in output da S_1 è 010 e quella da S_2 è 100.

Per le coppie $(1001, 0011)$ e $(0011, 1001)$, S_1 produce una somma XOR in output uguale a 010. Poiché il primo membro di una di queste coppie deve dare i 4 bit a sinistra di $E(L_4) \oplus K_4 = 0000 \oplus K_4$, i primi 4 bit di K_4 sono un elemento dell'insieme $\{1001, 0011\}$. Per le coppie $(1100, 0111)$ e $(0111, 1100)$, S_2 produce una somma XOR in uscita uguale a 100. Poiché il primo membro di una di queste coppie deve dare i 4 bit di destra di $E(L_4) \oplus K_4 = 0011 \oplus K_4$, gli ultimi 4 bit di K_4 sono un elemento dell'insieme $\{1111, 0100\}$.

Ora si ripete tutto (con la stessa macchina e la stessa chiave K) con

$$L_1 R_1 = 010111011011 \quad \text{e} \quad L_1^* R_1^* = 101110011011.$$

Un'analoga analisi mostra che i primi 4 bit di K_4 sono in $\{0011, 1000\}$ e gli ultimi 4 bit sono in $\{0100, 1011\}$. Combinando questo risultato con le informazioni precedenti, si vede che i primi 4 bit di K_4 sono 0011 e gli ultimi 4 bit sono 0100. Quindi $K = 00{*}001101$ (si ricordi che K_4 inizia con il quarto bit di K).

Rimane da trovare il terzo bit di K. La chiave $K = 000001101$, cifra $L_1 R_1$ in 001011101010, che non è $L_4 R_4$, mentre $K = 001001101$ produce la corretta cifratura. Quindi, la chiave è $K = 001001101$. ■

4.3.2 Crittanalisi differenziale a quattro round

Si supponga ora di avere ottenuto l'accesso a un dispositivo a quattro round. Come prima, si conosce tutto sul funzionamento interno dell'algoritmo, tranne la chiave che si vuole determinare. L'analisi svolta per tre round continua a valere. Tuttavia, per poter estendere tale analisi a quattro round si dovranno usare tecniche probabilistiche.

Nell'S-box S_1 è presente una debolezza: se si guardano le 16 coppie in input che hanno somma XOR uguale a 0011, si scopre che, per 12 di esse, la somma XOR degli output è uguale a 011. Poiché ci si aspetterebbe che, in media, soltanto due coppie forniscano la stessa somma XOR in output, si è di fronte a un caso piuttosto eccezionale. Piccole deviazioni dalla media possono capitare, ma una deviazione grande come questa facilita la ricerca della chiave.

Anche in S_2 è presente un analogo punto debole, benché non così grave. Tra le 16 coppie in input con somma XOR uguale a 1100, ce ne sono 8 con somma XOR degli output uguale a 010.

Si supponga ora di iniziare con R_0 e R_0^* scelti casualmente, ma tali che $R_0' = R_0 \oplus R_0^* = 001100$. Poiché $E(001100) = 00111100$, la somma XOR degli input in S_1 è 0011 e la somma XOR degli input in S_2 è 1100. Con probabilità 12/16 la somma XOR degli output da S_1 sarà 011 e con probabilità 8/16 la somma XOR degli output da S_2 sarà 010. Se si assume che gli output dei due S-box sono indipendenti, la somma XOR degli output combinati sarà 011010 con probabilità $(12/16)(8/16) = 3/8$. Poiché la funzione di espansione manda i bit 3 e 4 a entrambi gli S-box S_1 e S_2, non si può assumere che le due funzioni abbiano output indipendenti, ma 3/8 dovrebbe essere ancora una stima ragionevole.

Si supponga ora di scegliere L_0 e L_0^* in modo che $L_0' = L_0 \oplus L_0^* = 011010$. Si ricordi che nell'algoritmo di cifratura l'output degli S-box è sommato XOR con L_0 per ottenere R_1. Supponiamo che la somma XOR degli output degli S-box sia 011010. Allora $R_1' = 011010 \oplus L_0' = 000000$. Poiché $R_1' = R_1 \oplus R_1^*$, segue che $R_1 = R_1^*$.

Mettendo insieme tutto quanto, si vede che se si parte con due messaggi scelti casualmente con XOR uguale a $L_0' R_0' = 011010001100$, allora si ha una probabilità di circa 3/8 che $L_1' R_1' = 001100000000$.

Per trovare la chiave si provano varie coppie di input scelte a caso con XOR uguale a 011010001100 e si raccolgono i corrispondenti output $L_4 R_4$ e $L_4^* R_4^*$. A questo punto si assuma che $L_1' R_1' = 001100000000$ e si usi la crittanalisi differenziale per tre round con $L_1' = 001100$ e con gli output che si conoscono per dedurre un insieme di possibili chiavi K_4. Quando $L_1' R_1' = 001100000000$, cosa che dovrebbe accadere circa 3/8 delle volte, questa lista di chiavi conterrà K_4, insieme a qualche altra chiave casuale. I rimanenti 5/8 delle volte, la lista dovrebbe contenere chiavi casuali. Poiché non sembrano esserci ragioni per cui una qualche chiave sbagliata debba apparire frequentemente, probabilmente la chiave corretta K_4 apparirà nelle liste di chiavi più spesso delle altre chiavi.

Vediamo un esempio. Supponiamo di attaccare un dispositivo a quattro round. Proviamo un centinaio di coppie di input casuali $L_0 R_0$ e $L_0^* R_0^* = L_0 R_0 \oplus 011010001100$. Le frequenze delle possibili chiavi che otteniamo appaiono nella Tabella 4.1 in cui, per semplicità, consideriamo separatamente i primi 4 bit e gli ultimi 4 bit di K_4. Il caso più frequente è $K_4 = 11000010$, quindi la chiave K è $10 * 110000$.

Per determinare il bit rimanente, si può procedere come nel caso precedente. Si ha che 000000000000 viene cifrato in 100011001011 usando $K = 101110000$, mentre viene cifrato in 001011011010 usando $K = 100110000$. Se la macchina che si sta attaccando cifra 000000000000 in 100011001011, si può concludere che la seconda chiave non può essere corretta e che la chiave corretta è probabilmente $K = 101110000$.

L'attacco precedente può essere esteso a più round estendendo questi metodi. Si osservi che, in questo modello semplificato, si sarebbe potuta ottenere la chiave almeno altrettanto velocemente scorrendo tutte le possibilità per la chiave. Tuttavia, in sistemi più elaborati come DES, le tecniche di crittanalisi differenziale sono molto più efficienti della ricerca esaustiva tra tutte le chiavi, almeno finché il numero di round non diventa troppo grande. In particolare, la ragione per cui DES usa 16 round è perché la crittanalisi differenziale è più efficiente della ricerca esaustiva fino a quando non si usano 16 round[4].

Esiste un altro attacco a DES, chiamato **crittanalisi lineare**, che è stato sviluppato da Mitsuru Matsui [Matsui] e che, probabilmente, gli ideatori di DES non conoscevano. L'ingrediente principale è un'approssimazione di DES mediante una funzione lineare dei bit in input. La crittanalisi lineare è, teoricamente, più veloce di una ricerca esaustiva della chiave e richiede circa 2^{43} coppie testo in chiaro-testo cifrato per trovare la chiave. Per i dettagli del metodo, si veda [Matsui].

[4] Anche qui vale la stessa osservazione fatta nella nota 2.

Tabella 4.1 Frequenza delle possibili chiavi

Primi 4 bit	Frequenza	Primi 4 bit	Frequenza
0000	12	1000	33
0001	7	1001	40
0010	8	1010	35
0011	15	1011	35
0100	4	1100	59
0101	3	1101	32
0110	4	1110	28
0111	6	1111	39
Ultimi 4 bit	**Frequenza**	**Ultimi 4 bit**	**Frequenza**
0000	14	1000	8
0001	6	1001	16
0010	42	1010	8
0011	10	1011	18
0100	27	1100	8
0101	10	1101	23
0110	8	1110	6
0111	11	1111	17

4.4 DES

Un blocco di testo cifrato è formato da 64 bit. La chiave ha 56 bit, ma è espressa come una stringa di 64 bit. I bit 8°, 16°, 24°, ... sono bit di parità, disposti in modo che ogni blocco di 8 bit abbia un numero dispari di 1. Questo consente l'individuazione di eventuali errori. L'output della cifratura è un testo cifrato di 64 bit.
L'algoritmo DES, rappresentato nella Figura 4.4, elabora un testo in chiaro m di 64 bit ed è formato da tre fasi:

1. I bit di m sono permutati mediante una permutazione iniziale fissata, in modo da ottenere $m_0 = IP(m)$. Sia $m_0 = L_0R_0$, dove L_0 è formato dai primi 32 bit di m_0 e R_0 dagli ultimi 32 bit.

2. Per $1 \leq i \leq 16$, si eseguono le seguenti operazioni:
$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned}$$
dove K_i è una stringa di 48 bit ottenuta dalla chiave K e f è una funzione che verrà descritta più avanti.

3. Si scambia la parte sinistra con la parte destra in modo da ottenere $R_{16}L_{16}$ e poi si applica l'inversa della permutazione iniziale in modo da ottenere il testo cifrato $c = IP^{-1}(R_{16}L_{16})$.

La decifrazione si ottiene usando esattamente la stessa procedura, tranne per il fatto che le chiavi $K_1, \ldots, K_{16}$ vengono usate nell'ordine inverso. Tutto questo funziona per lo stesso motivo per cui funziona per il sistema semplificato descritto nel Paragrafo 4.2. Si noti che lo scambio sinistra-destra nel passo 3 dell'algoritmo DES significa che non bisogna fare lo scambio sinistra-destra che era necessario per la decifrazione nel Paragrafo 4.2.
Ora descriveremo in dettaglio i singoli passi. La permutazione iniziale non sembra avere un significato crittografico e forse fu scelta semplicemente per rendere più efficiente il caricamento dell'algoritmo nei chip che erano disponibili negli anni settanta del secolo scorso. Essa è riportata nella Tabella Permutazione Iniziale. Questo significa che il 58-esimo bit di m diventa il primo bit di m_0, il 50-esimo bit di m diventa il secondo bit di m_0 e così via.

Permutazione Iniziale

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

La funzione $f(R,K_i)$, rappresentata nella Figura 4.5, è definita attraverso i seguenti passi.

1. Come prima cosa, R viene espanso in $E(R)$ mediante la seguente tabella.

Permutazione di Espansione

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Questo significa che il primo bit di $E(R)$ è il 32-esimo bit di R e così via. Si noti che $E(R)$ ha 48 bit.

2. Si calcola $E(R) \oplus K_i$, che ha 48 bit, e lo si scrive come $B_1B_2 \cdots B_8$, dove ogni B_j ha 6 bit.

3. Ci sono 8 S-box $S_1, \ldots, S_8$, dati a pagina 115. B_j è l'input per S_j. Sia $B_j = b_1b_2 \cdots b_6$. La riga dell'S-box è specificata da b_1b_6 mentre $b_2b_3b_4b_5$ determina la colonna. Per esempio, se $B_3 = 001001$, si considera la riga 01, che è la seconda riga (00 dà la prima riga) e la colonna 0100, che è la quinta colonna (0100 rappresenta 4 in binario; poiché la prima colonna è numerata 0, la quinta è etichettata con 4). L'elemento di S_3 in questo posto è 3, che è 3 in binario. Quindi, in questo caso, l'output di S_3 è 0011. In questo modo, si ottengono otto output $C_1, C_2, \ldots, C_8$ di 4 bit.

4. La stringa $C_1C_2 \cdots C_8$ viene permutata secondo la seguente tabella.

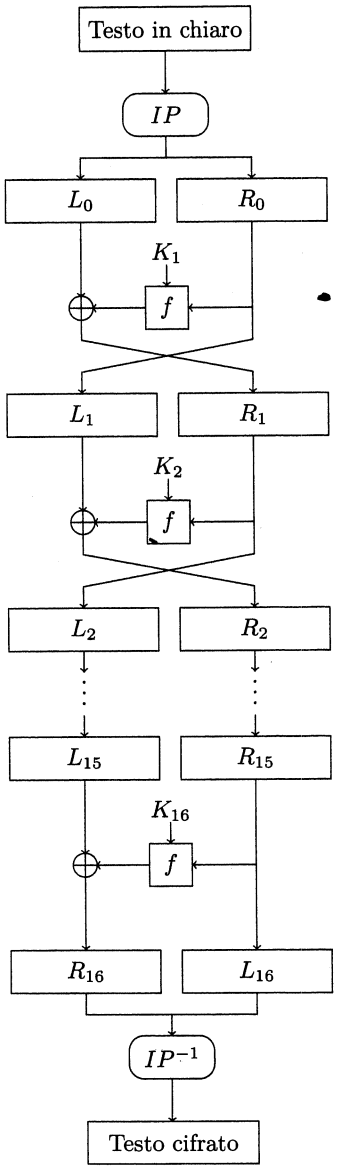

Figura 4.4 L'algoritmo DES.

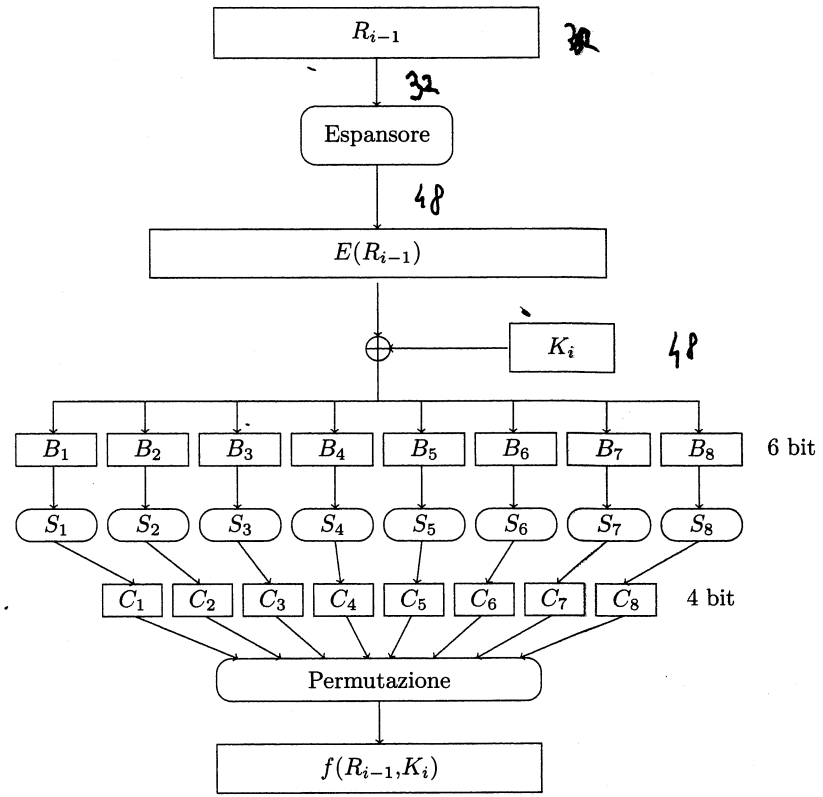

Figura 4.5 La funzione DES $f(R_{i-1},K_i)$.

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

La stringa di 32 bit che risulta è $f(R, K_j)$.

Infine, mostriamo come ottenere $K_1, \ldots, K_{16}$. Si tenga presente che si parte con una K di 64 bit.

1. I bit di parità vengono eliminati e i rimanenti bit vengono permutati mediante la seguente tabella.

Permutazione della chiave													
57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Il risultato viene scritto come C_0D_0, dove C_0 e D_0 hanno 28 bit.

2. Per $1 \leq i \leq 16$, siano $C_i = LS_i(C_{i-1})$ e $D_i = LS_i(D_{i-1})$. Qui LS_i significa far scorrere l'input uno o due bit verso sinistra, in base alla seguente tabella.

Numero di bit di scorrimento della chiave per round																
Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit di scorrimento	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

3. Si scelgono 48 bit dalla stringa C_iD_i di 56 bit, in base alla seguente tabella.

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

L'output è K_i.

Si ha che ogni bit della chiave viene usato in circa 14 dei 16 round.
In un buon sistema di cifratura, ogni bit del testo cifrato dovrebbe dipendere da tutti i bit del testo in chiaro. L'espansione $E(R)$ è pensata in modo che occorrano pochi round perché questo accada. Lo scopo della permutazione iniziale non è del tutto chiaro. Non ha scopi crittografici. Gli S-box sono il cuore dell'algoritmo e forniscono la sicurezza. I criteri progettuali di questi S-box sono stati una sorta di mistero fino a quando IBM publicò i seguenti criteri nei primi anni novanta del secolo scorso (per i dettagli, si veda [Coppersmith1]).

1. Ogni S-box ha 6 bit di input e 4 bit di output. Questo era il massimo che si potesse caricare su un chip nel 1974.

S-Box DES

S-box 1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box 2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-box 3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-box 4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-box 5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-box 6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-box 7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-box 8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

2. Gli output degli S-box non devono avvicinarsi a funzioni lineari degli input (la linearità renderebbe il sistema molto più facile da analizzare).
3. Ogni riga di un S-box contiene tutti i numeri da 0 a 15.
4. Se due input di un S-box differiscono di 1 bit, gli output devono differire di almeno 2 bit.
5. Se due input di un S-box differiscono nei loro primi 2 bit ma hanno gli stessi ultimi 2 bit, gli output devono essere diversi.
6. Ci sono 32 coppie di input che hanno una data somma XOR. Per ognuna di queste coppie, si calcola la somma XOR degli output. Di queste somme XOR di output, non più di otto devono essere uguali. Questo serve chiaramente ad impedire un attacco mediante crittanalisi differenziale.
7. L'ultimo criterio è simile a (6), ma coinvolge tre S-box.

Nei primi anni settanta del secolo scorso, un calcolatore impiegava vari mesi per trovare gli S-box appropriati. Ora, per completare tale ricerca basta un tempo molto breve.

4.4.1 DES non è un gruppo

Un modo per aumentare la grandezza della chiave di DES è dato dalla cifratura doppia. Si scelgono le chiavi K_1 e K_2 e si cifra un testo in chiaro P mediante $E_{K_2}(E_{K_1}(P))$. Questo aumenta la sicurezza?
Gli attacchi meet-in-the-middle (letteralmente: incontriamoci nel mezzo) ai crittosistemi sono discussi nel Paragrafo 4.7. Se un attaccante ha sufficiente memoria, la cifratura doppia fornisce un po' di protezione in più. Tuttavia, se un crittosistema è tale che la cifratura doppia è equivalente a una singola cifratura, allora una cifratura doppia non dà maggiore sicurezza.
Inoltre, se una cifratura doppia è equivalente a una singola cifratura, allora il crittosistema (a cifratura singola) è molto meno sicuro di quanto si potrebbe supporre inizialmente (si veda l'Esercizio 9 del Capitolo 8). Se questo fosse vero per DES, per esempio, una ricerca esaustiva tra tutte le 2^{56} chiavi potrebbe essere sostituita da una ricerca di lunghezza circa 2^{28}, che sarebbe molto più facile.
Per i cifrari affini (Paragrafo 2.2) e per RSA (Capitolo 6), la cifratura doppia con due chiavi K_1 e K_2 è equivalente alla cifratura con una terza chiave K_3. È così anche per DES? Ossia, esiste una chiave K_3 tale che $E_{K_3} = E_{K_2}E_{K_1}$? Questo problema viene spesso riformulato nella forma equivalente "DES è un gruppo?" (o, per chi non ha familiarità con la teoria dei gruppi, "DES è chiuso rispetto alla composizione?").
Fortunatamente, si ha che DES non è un gruppo. Qui daremo l'idea generale della dimostrazione. Per maggiori dettagli, si veda [Campbell-Wiener]. Sia E_0 la cifratura con la chiave formata tutta da 0 e sia E_1 la cifratura con la chiave formata tutta da 1. Queste chiavi sono deboli per scopi crittografici (si veda l'Esercizio 5). Inoltre, D. Coppersmith ha scoperto che applicando $E_1 \circ E_0$ ripetutamente a certi testi in chiaro si ottiene il testo in chiaro originale dopo circa 2^{32} iterazioni. Un ciclo di lunghezza n è una successione di cifrature (per lo stesso testo in chiaro P)

$$E_1E_0(P),\ E_1E_0(E_1E_0(P)),\ E_1E_0(E_1E_0(E_1E_0(P))),\ \ldots,\ (E_1E_0)^n(P) = P,$$

dove n è il più piccolo intero positivo tale che $(E_1E_0)^n(P) = P$.

Lemma. *Se m è il più piccolo intero positivo per cui $(E_1E_0)^m(P) = P$ per ogni P e se n è la lunghezza di un ciclo (in modo che $(E_1E_0)^n(P_0) = P_0$ per un particolare P_0), allora n divide m.*

Dimostrazione. Si divida m per n, con resto r. Questo significa che $m = nq + r$ per un opportuno intero q e $0 \le r < n$. Poiché $(E_1E_0)^n(P_0) = P_0$, cifrando q volte con $(E_1E_0)^n$ lascia P_0 inalterato. Quindi

$$P_0 = (E_1E_0)^m(P_0) = (E_1E_0)^r(E_1E_0)^{nq}(P_0) = (E_1E_0)^r(P_0).$$

Poiché n è il più piccolo intero positivo per cui $(E_1E_0)^n(P_0) = P_0$ e $0 \le r < n$, si deve avere $r = 0$. Questo significa che $m = nq$, ossia che n divide m. □

Si supponga ora che DES sia chiuso rispetto alla composizione. Allora $E_1E_0 = E_K$ per qualche chiave K. Inoltre, anche E_K^2, E_K^3, ... sono rappresentati da chiavi DES. Poiché ci sono solo 2^{56} chiavi possibili, si deve avere $E_K^j = E_K^i$ per due interi i e j con $0 \le i < j \le 2^{56}$ (altrimenti si avrebbero $2^{56} + 1$ chiavi di cifratura distinte). Si decifra i volte: $E_K^{j-i} = D_K^iE_K^j = D_K^iE_K^i$, che è la funzione identità. Poiché $0 < j - i \le 2^{56}$, per il più piccolo intero positivo m per cui E_K^m è la funzione identità, si ha anche $m \le 2^{56}$.
Coppersmith trovò le lunghezze dei cicli per 33 testi in chiaro P_0. Per il lemma, m è un multiplo delle lunghezze di questi cicli. Quindi m è maggiore o uguale del minimo comune multiplo delle lunghezze di questi cicli, che risulta essere circa 10^{277}. Ma se DES fosse chiuso rispetto alla composizione, si sarebbe mostrato che $m \le 2^{56}$. Quindi DES non è chiuso rispetto alla composizione.

4.5 Modalità operative

DES è un esempio di cifrario a blocchi. Un blocco di testo in chiaro, 64 bit nel caso di DES, viene cifrato in un blocco di testo cifrato. Tuttavia, esistono molte circostanze in cui è necessario cifrare messaggi che sono più lunghi o più corti della lunghezza dei blocchi del cifrario. Per esempio, si potrebbe avere un messaggio con un testo che è molte volte più lungo di 64 bit. Si potrebbe avere un testo in chiaro più corto dei blocchi in quelle situazioni in cui i dati vengono generati bit a bit, o carattere a carattere, e il testo cifrato deve essere fornito in output non appena il testo in chiaro viene ricevuto in input.
I cifrari a blocchi possono essere eseguiti in varie modalità operative differenti, permettendo agli utenti di scegliere le modalità appropriate per soddisfare le richieste delle applicazioni. Esistono cinque modalità operative comuni: la modalità ECB (*Electronic Codebook*), la modalità CBC (*Cipher Block Chaining*), la modalità CFB (*Cipher Feedback*), la modalità OFB (*Output Feedback*) e la modalità CTR (*Counter*).

4.5.1 Modalità ECB

Il modo più naturale per usare un cifrario a blocchi è di spezzare un lungo testo in chiaro in blocchi di testo in chiaro di lunghezza appropriata ed elaborare ogni blocco

separatamente con la funzione di cifratura E_K. Questa è la modalità operativa ECB. Il testo in chiaro P viene spezzato in pezzi più piccoli $P = [P_1, P_2, \ldots, P_L]$ e il testo cifrato è $C = [C_1, C_2, \ldots, C_L]$ dove

$$C_j = E_K(P_j)$$

è la cifratura di P_j mediante la chiave K.
C'è una vulnerabilità intrinseca nella modalità ECB, che diventa evidente quando si ha a che fare con testi in chiaro lunghi. Supponiamo che Eva, che è un avversario, abbia osservato le comunicazioni tra Alice e Bob per un periodo di tempo sufficientemente lungo. Se Eva è riuscita ad acquisire qualche frammento di testo in chiaro corrispondente ai frammenti di testo cifrato che ha osservato, può iniziare a costruire una lista di corrispondenze tra blocchi di testo in chiaro e blocchi di testo cifrato con la quale può decifrare le comunicazioni future tra Alice e Bob. Eva non ha mai bisogno di calcolare la chiave K. Le basta cercare il messaggio di testo cifrato nella propria lista e usare il corrispondente testo in chiaro (se disponibile) per decifrare il messaggio.
Questo può essere un problema grave poiché molti messaggi del mondo reale sono formati da frammenti ripetuti. Le e-mail sono un primo esempio. Una e-mail tra Alice e Bob potrebbe iniziare con la seguente intestazione:

```
Date: Tue, 29 Feb 2000 13:44:38 -0500 (EST)
```

Il testo cifrato inizia con la versione cifrata di "Date: Tu". Se Eva scopre che questo tratto di testo cifrato si presenta spesso il Martedì, potrebbe essere in grado di indovinare, senza conoscere nulla del testo in chiaro, che quei messaggi sono e-mail inviate di Martedì. Con pazienza e ingegnosità, Eva potrebbe riuscire a mettere insieme gran parte dell'intestazione del messaggio e riuscire a immaginarne il contesto. Con ancora maggiore pazienza e memoria di computer, potrebbe riuscire a mettere insieme frammenti importanti del messaggio.
Un altro problema nella modalità ECB è il fatto che Eva potrebbe cercare di modificare il messaggio cifrato che deve essere inviato a Bob. Potrebbe essere capace di estrarre porzioni rilevanti del messaggio e, usando la propria lista, potrebbe costruire un messaggio di testo cifrato falso da inserire nel flusso dei dati.

4.5.2 Modalità CBC

Un metodo per ridurre i problemi che si presentano nella modalità ECB è quello di usare la concatenazione. La concatenazione è un meccanismo di retroazione dove la cifratura di un blocco dipende dalla cifratura dei blocchi precedenti.
In particolare, nella cifratura si ha

$$C_j = E_K(P_j \oplus C_{j-1}),$$

mentre nella decifrazione si ha

$$P_j = D_K(C_j) \oplus C_{j-1},$$

dove C_0 è un valore iniziale scelto. Come al solito, E_K e D_K denotano le funzioni di cifratura e di decifrazione per il cifrario a blocchi.

Così, nella modalità CBC, il testo in chiaro viene sommato XOR con il precedente blocco di testo cifrato e il risultato viene poi cifrato. La modalità CBC è rappresentata nella Figura 4.6.

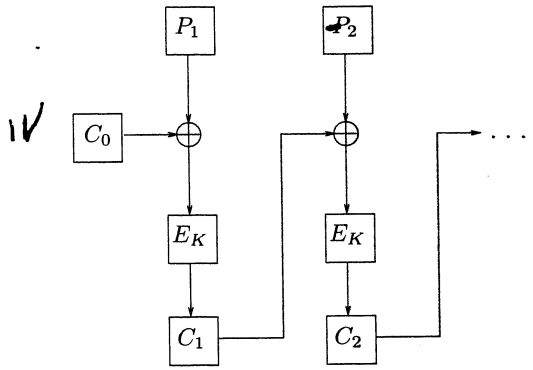

Figura 4.6 La modalità CBC.

Il valore iniziale C_0 è spesso chiamato vettore di inizializzazione o semplicemente IV. Se si usa un valore fissato per C_0, per esempio $C_0 = 0$, e si ha sempre lo stesso messaggio di testo in chiaro, il risultato sarà che i testi cifrati risultanti saranno gli stessi. Ciò non è auspicabile, poiché permette all'avversario di dedurre che è stato trasmesso lo stesso testo in chiaro. Questa può essere un'informazione molto importante e può essere usata dall'avversario per inferire il significato del testo in chiaro originale.
In pratica, questo problema si risolve scegliendo C_0 sempre a caso e inviando C_0 in chiaro insieme al primo testo cifrato C_1. In questo modo, anche se si invia ripetutamente lo stesso messaggio di testo in chiaro, un osservatore vedrà ogni volta un testo cifrato differente.

4.5.3 Modalità CFB

Uno dei problemi che si hanno con le modalità CBC ed ECB è che la cifratura (e di conseguenza la decifrazione) non può iniziare finché non è disponibile un blocco completo di 64 bit di testo in chiaro. La modalità CFB funziona in un modo molto simile al modo in cui si usa LFSR per cifrare il testo in chiaro. Invece di usare una ricorrenza lineare per generare bit casuali, la modalità CFB è una modalità operativa a flusso che produce bit pseudocasuali usando il cifrario a blocchi E_K. In generale, CFB funziona in una modalità a k bit, dove ogni applicazione produce k bit casuali da sommare XOR con il testo in chiaro. Per la nostra discussione, tuttavia, ci concentreremo sulla versione di CFB a 8 bit. Usare la modalità CFB a 8 bit permette di cifrare un frammento di messaggio di 8 bit (per esempio, un singolo carattere) senza dovere aspettare di avere un intero blocco di dati. Ciò è utile, per esempio, nelle comunicazioni interattive al computer.

Il testo in chiaro viene spezzato in blocchi di 8 bit: $P = [P_1, P_2, \ldots]$, dove ogni P_j è formato da 8 bit, invece che dei 64 bit usati in ECB e CBC. La cifratura procede nel modo seguente. Si sceglie un X_1 iniziale di 64 bit. Poi, per $j = 1, 2, 3, \ldots$, si eseguono le operazioni

$$\begin{aligned} O_j &= L_8(E_K(X_j)) \\ C_j &= P_j \oplus O_j \\ X_{j+1} &= R_{56}(X_j) \,\|\, C_j, \end{aligned}$$

dove $L_8(X)$ indica gli 8 bit più a sinistra di X, $R_{56}(X)$ indica i 56 bit più a destra di X e $X\|Y$ indica la stringa ottenuta scrivendo X seguito da Y. La modalità operativa CFB è rappresentata nella Figura 4.7.

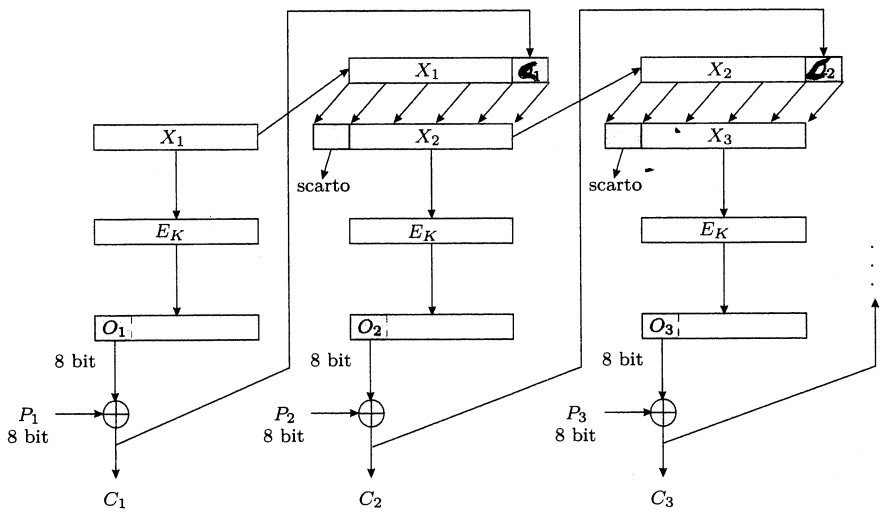

Figura 4.7 La modalità CFB.

La decifrazione avviene mediante i seguenti passi:

$$\begin{aligned} P_j &= C_j \oplus L_8(E_K(X_j)) \\ X_{j+1} &= R_{56}(X_j) \,\|\, C_j. \end{aligned}$$

La decifrazione non coinvolge la funzione di decifrazione D_K. Questo è uno dei vantaggi dell'impiego di un cifrario a blocchi in una modalità a flusso nel caso in cui la funzione di decifrazione per il cifrario a blocchi sia più lenta della funzione di cifratura.
Diamo un'occhiata all'interno di un round dell'algoritmo CFB. Come prima cosa, un registro di 64 bit viene inizializzato con X_1. Questi 64 bit vengono cifrati usando E_K. Gli 8 bit più a sinistra di $E_K(X_1)$ vengono estratti e sommati XOR con gli 8 bit di P_1 in modo da formare C_1. Poi C_1 viene inviato al destinatario. Prima di passare a P_2,

il registro X_1 di 64 bit viene aggiornato estraendo i 56 bit più a destra. Gli 8 bit di C_1 vengono concatenati a destra in modo da formare $X_2 = R_{56}(X_1)\|C_1$. Quindi P_2 viene cifrato mediante lo stesso processo, ma usando X_2 al posto di X_1. Dopo che P_2 è stato cifrato in C_2, il registro di 64 bit viene aggiornato in modo da formare

$$X_3 = R_{56}(X_2)\|C_2 = R_{48}(X_1)\|C_1\|C_2.$$

Entro la fine dell'ottavo round, l'X_1 iniziale è scomparso dal registro di 64 bit e $X_9 = C_1\|C_2\|\cdots\|C_8$. C_j continua a passare attraverso il registro, così per esempio $X_{20} = C_{12}\|C_{13}\|\cdots\|C_{19}$.
CFB cifra il testo in chiaro in un modo simile alla stringa monouso o agli LFSR. La chiave K e i numeri X_j vengono usati per produrre stringhe binarie che vengono sommate XOR con il testo in chiaro per produrre il testo cifrato. Questo tipo di cifratura è molto differente da quello di ECB e di CBC, dove il testo cifrato è l'output di DES.
Nelle applicazioni pratiche, CFB è utile perché può riprendere il funzionamento corretto anche in presenza di errori di trasmissione nella trasmissione del testo cifrato. Supponiamo che chi trasmette invii i blocchi di testo cifrato $C_1, C_2, \ldots, C_k, \ldots$ e che durante la trasmissione C_1 si modifichi in modo che il ricevente osserva $\widetilde{C}_1, C_2, \ldots$. La decifrazione prende $\widetilde{C}_1$ e produce una versione alterata di P_1 con errori di bit nei posti in cui $\widetilde{C}_1$ aveva errori di bit. Ora, dopo aver decifrato questo blocco di testo cifrato, il ricevente forma un X_2 non corretto, che indichiamo con $\widetilde{X}_2$. Se X_1 era $(*,*,*,*,*,*,*,*)$, allora $\widetilde{X}_2 = (*,*,*,*,*,*,*,*,\widetilde{C}_1)$. Quando il ricevente acquisisce un C_2 non alterato e decifra, viene prodotta una versione di P_2 completamente alterata. Quando forma X_3, il decifratore in realtà forma $\widetilde{X}_3 = (*,*,*,*,*,*,*,*,\widetilde{C}_1, C_2)$. Il decifratore ripete questo processo, ottenendo alla fine le versioni errate di $P_1, P_2, \ldots, P_9$. Quando il decifratore calcola X_9, il blocco alterato risulta essere il blocco più a sinistra di $\widetilde{X}_9$, ossia $\widetilde{X}_9 = (\widetilde{C}_1, C_2, \ldots, C_8)$. Al passo successivo, l'errore sarà fluito via dal registro X_{10}. Quindi X_{10} e i registri successivi non saranno alterati. Per una versione semplificata di queste idee, si veda l'Esercizio 9.

4.5.4 Modalità OFB

Uno degli inconvenienti delle modalità CBC e CFB risiede nel fatto che gli errori si propagano per un tempo corrispondente alla lunghezza dei blocchi del cifrario. La modalità OFB è un altro esempio di modalità operativa a flusso per un cifrario a blocchi, dove la cifratura avviene sommando XOR il messaggio con un flusso di bit pseudocasuali generato dal cifrario a blocchi. Un vantaggio della modalità OFB è che evita la propagazione degli errori.
Analogamente a CFB, OFB può lavorare su blocchi di varie lunghezze. Per la nostra discussione, ci concentreremo sulla versione di OFB a 8 bit, dove OFB è usato per cifrare blocchi di 8 bit di testo in chiaro in una modalità a flusso. Come in CFB, si spezza il testo in chiaro in blocchi di 8 bit, con $P = [P_1, P_2, \ldots]$. Si parte con un valore iniziale X_1, che ha una lunghezza uguale a quella dei blocchi del cifrario, per esempio 64 bit. X_1 viene spesso chiamato IV e dovrebbe essere scelto a caso. X_1 viene cifrato mediante la chiave K, ottenendo 64 bit di output. Gli 8 bit più a sinistra O_1 del testo

cifrato, dopo essere stati estratti, vengono sommati XOR con i primi 8 bit P_1 del testo in chiaro, ottenendo 8 bit C_1 di testo cifrato.
Fin qui, è come in CFB. OFB differisce da CFB per quello che avviene dopo. Nel passo successivo, CFB aggiorna il registro X_2 estraendo i 56 bit di destra di X_1 e concatenando C_1 a destra. Invece di usare il testo cifrato, OFB usa l'output della cifratura. Cioé, OFB aggiorna il registro X_2 estraendo i 56 bit di destra di X_1 e concatenando O_1 a desta.
In generale, viene eseguita la seguente procedura per $j = 1, 2, 3, \ldots$:

$$\begin{aligned} O_j &= L_8\left(E_K(X_j)\right) \\ X_{j+1} &= R_{56}(X_j) \parallel O_j \\ C_j &= P_j \oplus O_j. \end{aligned}$$

I passi per la modalità operativa OFB vengono rappresentati nella Figura 4.8.

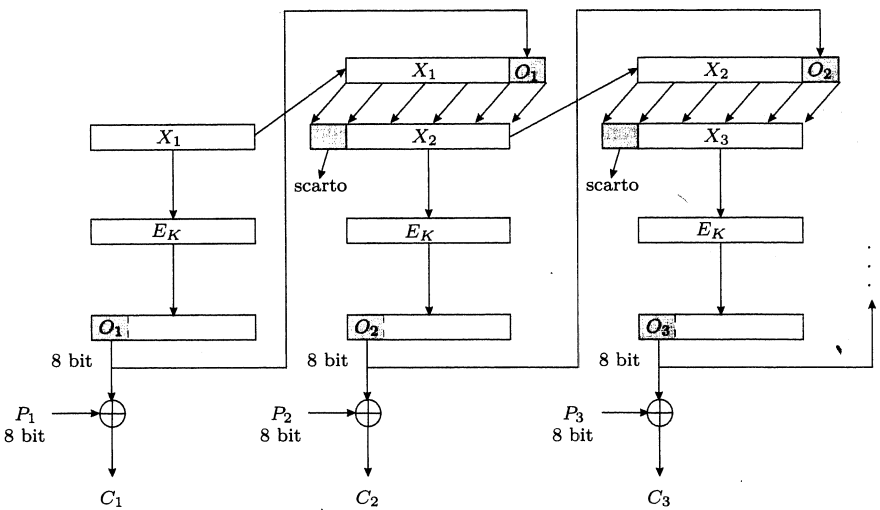

Figura 4.8 La modalità OFB.

Qui, il flusso di output O_j è la cifratura del registro contenente il precedente output del cifrario a blocchi. Questo output viene poi trattato come un flusso di chiave (*keystream*) e viene sommato XOR con i testi in chiaro P_j entranti per produrre un flusso di testo cifrato. La decifrazione è molto semplice. Si ottiene il testo in chiaro P_j sommando XOR il corrispondente testo cifrato C_j con il flusso di chiave di output O_j. Ancora, come in CFB, non si ha bisogno della funzione di decifrazione D_K.
Ma perché si dovrebbe costruire questa modalità anziché usare la modalità CFB? Ci sono alcuni vantaggi nella strategia OFB. Come prima cosa, la generazione del flusso di chiave O_j in output può essere eseguito completamente senza alcun testo in chiaro. Questo significa che il flusso di chiave può essere generato in anticipo. Questo può essere utile per le applicazioni in cui non c'è il tempo di eseguire le operazioni di cifratura quando arriva il messaggio di testo in chiaro.
Un altro vantaggio risiede nel suo modo di operare quando il testo cifrato C_j presenta errori di trasmissione. Quando si esegue la decifrazione, nel testo in chiaro risultano alterati solo i corrispondenti bit. Poiché i flussi in output successivi vengono costruiti usando la cifratura del registro, e non il testo cifrato alterato, il flusso in output rimarrà sempre pulito e gli errori nel testo cifrato non verranno propagati.
Per riassumere, CFB richiedeva che i bit errati uscissero dal registro, cosa che produceva un numero di bit di testo in chiaro errati pari alla lunghezza di un intero blocco. OFB, al contagio, si corregge immediatamente.
Tuttavia, vi è un problema con OFB, comune a tutti i cifrari a flusso che si ottengono sommando XOR numeri pseudocasuali al testo in chiaro. Se un avversario conosce un particolare testo in chiaro P_j e il corrispondente testo cifrato C_j, può condurre il seguente attacco. Prima calcola

$$O_j = C_j \oplus P_j$$

per ottenere il flusso di chiave. Poi può generare un qualsiasi testo in chiaro falso P'_j. Infine, produce il testo cifrato semplicemente sommando XOR con il flusso in output che ha calcolato:

$$C'_j = P'_j \oplus O_j.$$

Questo gli permette di modificare i messaggi.
La modalità OFB produce un flusso di bit e cifra in modo analogo al metodo LFSR (si veda il Paragrafo 2.11). La differenza risiede nel fatto che LFSR è più veloce, mentre OFB è più sicuro.

4.5.5 Modalità CTR

La modalità CTR, o modalità contatore, si basa sugli stessi principi della modalità OFB. Come OFB, CTR genera in output un flusso di chiave che viene sommato XOR con blocchi di testo in chiaro per produrre il testo cifrato. La differenza principale tra CTR e OFB risiede nel fatto che l'output del flusso O_j in CTR non è legato ai precedenti flussi in output.
CTR parte con il testo in chiaro suddiviso in blocchi di 8 bit, $P = [P_1, P_2, \ldots]$. Si parte con un valore iniziale X_1, che ha una lunghezza uguale a quella dei blocchi del cifrario, per esempio 64 bit. X_1 viene cifrato mediante la chiave K per produrre 64 bit di output. Gli 8 bit più a sinistra del testo cifrato vengono estratti e sommati XOR con P_1 per produrre il testo cifrato C_1 di 8 bit.
Ora, invece di aggiornare il registro X_2 inserendo l'output del cifrario a blocchi, semplicemente si prende $X_2 = X_1 + 1$. In questo modo, X_2 non dipende dal precedente output. CTR allora genera un nuovo flusso in output cifrando X_2. Analogamente, si procede usando $X_3 = X_2 + 1$ e così via. Il j-esimo testo cifrato viene prodotto sommando XOR gli 8 bit di sinistra dalla cifratura del j-esimo registro con il corrispondente testo in chiaro P_j.

In generale, la procedura per CTR è

$$\begin{aligned} X_j &= X_{j-1} + 1 \\ O_j &= L_8(E_K(X_j)) \\ C_j &= P_j \oplus O_j \end{aligned}$$

per $j = 2, 3, \ldots$ ed è rappresentata nella Figura 4.9.

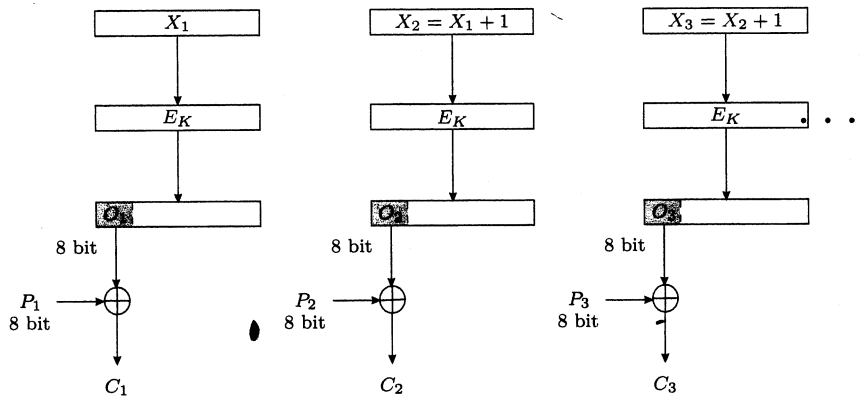

Figura 4.9 La modalità CTR.

Il lettore potrebbe chiedersi cosa accade a X_j se si continua a sommare 1. Non diventerà troppo grande? È improbabile che accada, ma se dovesse accadere, semplicemente si torna indietro e si riparte da 0.
Come OFB, i registri X_j possono essere calcolati in anticipo e la reale cifratura del testo in chiaro è semplice poichè coinvolge solo l'operazione XOR. Di conseguenza, le sue prestazioni sono identiche a quelle di OFB quando si introducono errori nel testo cifrato. Tuttavia, il vantaggio della modalità CTR rispetto alla modalità OFB risiede nel fatto che possono essere calcolati in parallelo molti blocchi di output O_j. Non bisogna calcolare O_j prima di calcolare O_{j+1}. Questo rende la modalità CTR ideale per la parallelizzazione.

4.6 Violazione di DES

DES è stato il sistema crittografico standard per gli ultimi 20 anni del Ventesimo secolo, anche se nell'ultima parte di questo periodo ha iniziato a mostrare i segni dell'età. In questo paragrafo verrà discussa la violazione di DES e verranno presentate alcune alternative possibili.
Dal 1975 in poi, vennero posti vari interrogativi relativi alla robustezza di DES. Molte persone appartenenti alla comunità accademica si lamentavano delle dimensioni limitate delle chiavi DES, affermando che una chiave di 56 bit era insufficiente per la

sicurezza. Infatti, pochi mesi dopo il rilascio di DES da parte di NBS, Whitfield Diffie e Martin Hellman pubblicarono un articolo intitolato "Exhaustive cryptanalysis of the NBS Data Encryption Standard" [Diffie-Hellman2] in cui stimavano che si potesse costruire una macchina da 20 milioni di dollari (del 1977) in grado di violare DES in circa un giorno. Lo scopo specifico di questa macchina era quello di eseguire un attacco a DES, cosa su cui torneremo più avanti.
Nel 1987 ci fu la seconda revisione quinquennale di DES. Questo volta NBS chiese suggerimenti per decidere se accettare lo standard per un altro periodo, se modificarlo o se cancellarlo completamente. Le discussioni riguardanti DES videro la NSA opporsi alla ricertificazione di DES. NBS sostenne che ormai DES iniziava a mostrare segni di debolezza, visti i livelli di potenza di calcolo raggiunti in quegli anni, e propose di sbarazzarsi completamente di DES sostituendolo con un insieme di algoritmi ideati dalla NSA, il cui funzionamento interno sarebbe stato noto solo alla NSA e che sarebbe stato ben protetto contro le tecniche di *reverse engineering*. Questa proposta fu respinta, in parte perché molte importanti industrie americane sarebbero rimaste prive di protezione durante l'allestimento degli algoritmi sostitutivi. Alla fine DES fu riapprovato come standard, anche se si era riconosciuto che DES stava mostrando segni di debolezza.
Quando, dopo cinque anni, venne il momento della nuova revisione quinquennale, NBS era diventato NIST. Nonostante le debolezze osservate nel 1987 e l'avanzamento della tecnologia in quei cinque anni, nel 1992 NIST ricertificò l'algoritmo DES.
Nel 1993 Michael Wiener, un ricercatore del Bell-Northern Research, propose e progettò un dispositivo che avrebbe attaccato DES in modo più efficace. L'idea era quella di usare la già ben sviluppata tecnologia di commutazione disponibile nell'industria telefonica.
Il 1996 vide la formulazione di tre approcci fondamentali agli attacchi dei cifrari simmetrici come DES. Il primo metodo consisteva nel distribuire il calcolo su un'ampia serie di macchine. Questo aveva il vantaggio di essere relativamente economico e di poter facilmente distribuire i costi su un gran numero di persone. Un altro approccio consisteva nel progettare apposite architetture per attaccare DES (come nell'idea di Michael Wiener). Benché questo approccio promettesse di essere molto più efficace, era anche l'approccio più costoso. L'approccio intermedio sfruttava le matrici logiche programmabili ed è quello che fino ad oggi ha ricevuto minore attenzione.
In tutti e tre i casi, l'approccio più comune per attaccare DES è stato quello di eseguire una ricerca esaustiva nello spazio delle chiavi. Per DES questo sembra essere ragionevole, poiché, come osservato in precedenza, tecniche crittanalitiche più complicate non sono riuscite a mostrare miglioramenti significativi rispetto alla ricerca esaustiva.
L'approccio del calcolo distribuito per violare DES divenne molto popolare, specialmente con la crescente popolarità di Internet. Nel 1997 la RSA Data Security Company lanciò una sfida, in cui bisognava trovare la chiave e violare un messaggio cifrato con DES. Chiunque avesse violato il messaggio avrebbe vinto un premio di 10.000$. Solo cinque mesi dopo l'annuncio della Sfida DES del 1997, Rocke Verser inviò la chiave DES vincente. Il fatto più importante qui è che si ha un esempio in cui l'approccio del calcolo distribuito ha permesso di attaccare DES con successo. Rocke Verser ha implementato un programma in cui migliaia di computer distribuiti in tutta Internet sono riusciti a violare il cifrario DES. Un gran numero di persone lasciarono girare il

programma di Verser sulle proprie macchine personali (e corporative) con l'accordo che Verser avrebbe spartito la vincita, 60% e 40%, con il possessore del computer che avrebbe effettivamente trovato la chiave. Alla fine la chiave fu trovata da Michael Sanders, dopo che era stato esplorato circa il 25% dello spazio delle chiavi DES. La frase della Sfida DES si decifrava in "Strong cryptography makes the world a safer place".

L'anno successivo, l'RSA Data Security bandì la Seconda Sfida DES. Questa volta la chiave corretta fu trovata da Distributed Computing Technologies e il messaggio si decifrava in "Many hands make light work". La chiave fu trovata in 39 giorni dopo una ricerca di circa l'85% delle possibili chiavi. Il fatto che il vincitore della seconda sfida abbia esplorato una maggiore quantità dello spazio delle chiavi in un tempo minore, mostra quanto drammatico possa essere l'effetto di un anno di avanzamento tecnologico sulla crittanalisi.

Nell'estate del 1998 l'Electronic Frontier Foundation (EFF) sviluppò un progetto chiamato DES Cracker, il cui proposito era quello di rivelare la vulnerabilità dell'algoritmo DES contro un attacco realizzato avendo a disposizione un'architettura specializzata. Il progetto DES Cracker si fondava su un semplice principio: il computer medio non è adatto al compito di violare DES. Questa è una affermazione ragionevole, poiché i computer ordinari, per loro natura, sono progettati per affrontare compiti generici, che vanno dall'eseguire un sistema operativo ai giochi elettronici. Il proposito del team EFF era quello di costruire un hardware specializzato che sfruttasse la natura parallelizzabile della ricerca esaustiva. Il team aveva un budget di 200.000$.

Ora descriveremo brevemente l'architettura prodotta dal team EFF. Per maggiori informazioni sull'EFF Cracker e su altre sue funzioni, si veda [Gilmore].

L'EFF DES Cracker era sostanzialmente costituito da tre parti principali: un personal computer, un software e un'ampia collezione di chip specializzati. Il computer era connesso alla matrice di chip e il software controllava il lavoro di ogni chip. Per la maggior parte del tempo, il software non interagiva molto con l'hardware; dava semplicemente ai chip le informazioni necessarie per iniziare l'elaborazione e aspettava che i chip fornissero le chiavi candidate. In questo senso, l'hardware eliminava efficientemente un gran numero di chiavi non valide e forniva solo chiavi che erano potenzialmente promettenti. Poi il software elaborava per conto proprio ognuna delle chiavi candidate promettenti, andando a controllare se una di queste fosse effettivamente la chiave reale.

Il DES Cracker prendeva un campione di testo cifrato di 128 bit (16 byte) e lo spezzava in due blocchi di testo di 64 bit (8 byte). Ogni chip nell'EFF DES Cracker era formato da 24 unità di ricerca. Un'unità di ricerca era un sottoinsieme di un chip con il compito di prendere una chiave e due blocchi di testo cifrato di 64 bit e di cercare di decifrare il primo blocco di 64 bit usando la chiave. Se il testo cifrato "decifrato" sembrava interessante, allora l'unità di ricerca decifrava il secondo blocco e andava a vedere se anche quel testo cifrato "decifrato" fosse interessante. Se entrambi i testi decifrati erano interessanti, allora l'unità di ricerca diceva al software che la chiave che aveva controllato era promettente. Se, dopo la decifrazione del primo blocco di testo cifrato di 64 bit, il testo decifrato non sembrava sufficientemente interessante, allora l'unità di ricerca incrementava la sua chiave di 1 per formare una nuova chiave. Poi provava questa nuova chiave, andando ancora a vedere se il risultato fosse interessante, e

procedeva in questo modo attraverso tutta la regione di spazio delle chiavi che gli era assegnata.

Come faceva il team EFF a dire che un testo decifrato era "interessante"? Come prima cosa supponeva che il testo in chiaro soddisfacesse alcuni requisiti basilari: per esempio che fosse scritto usando lettere, numeri e segni di punteggiatura. Poiché i dati che stava decifrando erano testi, sapeva che ogni byte corrispondeva a un carattere di 8 bit. Dei 256 possibili valori che potevano essere assunti da un carattere di 8 bit, solo 69 erano interessanti (le lettere maiuscole e minuscole dell'alfabeto, i numeri, lo spazio e pochi segni di punteggiatura). Perché un byte venisse considerato interessante, doveva contenere uno di questi 69 caratteri. Quindi aveva 69/256 possibilità di essere interessante. Approssimando questo rapporto a 1/4 e supponendo che i byte decifrati siano indipendenti, si ha che la possibilità che un blocco di 8 byte di testo decifrato sia interessante è $1/4^8 = 1/65536$. Così solo 1/65536 delle chiavi esaminate erano da considerarsi promettenti.

Questa riduzione non era ancora sufficiente. Il software avrebbe speso ancora troppo tempo nella ricerca di falsi candidati. Per poter ridurre ulteriormente il campo delle chiavi candidate promettenti, fu necessario usare il secondo blocco di 8 byte di testo. Questo blocco veniva decifrato per vedere se il risultato fosse interessante. Assumendo l'indipendenza tra i blocchi, si ha che solo $1/4^{16} = 1/65536^2$ delle chiavi potevano essere considerate promettenti. Questo riduceva significativamente la quantità spazio delle chiavi che il software doveva esaminare.

Ogni chip era formato da ventiquattro unità di ricerca e a ogni unità era assegnata la sua propria regione di spazio delle chiavi da esplorare. Un singolo chip a 40-MHz avrebbe impiegato circa 38 anni per esplorare l'intero spazio delle chiavi. Per ridurre ancora il tempo necessario per elaborare le chiavi, il team EFF usava 64 chip su ogni singola scheda, dodici schede su ogni chassis e infine due chassis erano connessi al personal computer che controllava le comunicazioni con il software.

Alla fine il DES Cracker risultava formato da circa 1500 chip e poteva violare DES mediamente in circa 4,5 giorni. Il DES Cracker non fu affatto un modello ottimale per violare DES. In particolare, ognuno dei chip utilizzati lavorava a 40 MHz, cioé lentamente per gli standard moderni. Certamente in futuro si sarebbero potuti produrre nuovi modelli basati su chip operanti a una frequenza di clock più elevata.

Questo sviluppo indica fortemente la necessità di sostituire DES. Ci sono due approcci principali per ottenere una maggiore sicurezza in futuro. Il primo coinvolge l'uso di DES più volte e porta al popolare metodo chiamato Triplo DES. Il secondo approccio è quello di trovare un nuovo sistema che utilizzi una chiave di dimensione maggiore e non di soli 56 bit.

Iniziamo col descrivere l'idea che sta alla base degli schemi di DES multiplo. L'idea è di cifrare più volte lo stesso testo in chiaro, usando lo stesso algoritmo con chiavi diverse.

Doppio DES cifra il testo in chiaro cifrandolo prima con una chiave e poi una seconda volta con una chiave differente. Poiché DES non forma un gruppo (si veda Paragrafo 4.4), si può ipotizzare che Doppio DES raddoppi lo spazio delle chiavi e quindi che lo spazio delle chiavi sia formato da 2^{112} chiavi. Questo, tuttavia, non è vero. Merkle e Hellman hanno mostrato che lo schema a cifratura doppia ha in realtà un livello di sicurezza di una chiave di 57 bit. La riduzione da 2^{112} a 2^{57} è dovuta all'efficacia dell'**attacco meet-in-the-middle** (che verrà descritto nel prossimo paragrafo).

Poiché Doppio DES presenta una debolezza, spesso si usa **Triplo DES**, che sembra avere un livello di sicurezza circa uguale a quello di un chiave di 112 bit. Triplo DES può essere implementato in almeno due modi. Nel primo si scelgono tre chiavi K_1, K_2, K_3 e si esegue $E_{K_1}(E_{K_2}(E_{K_3}(m)))$. Nel secondo si scelgono due chiavi K_1 e K_2 e si esegue $E_{K_1}(D_{K_2}(E_{K_1}(m)))$. Quando $K_1 = K_2$, questo si riduce al DES singolo. Questa compatibilità è la ragione per cui nel mezzo si usa D_{K_2} invece di E_{K_2}; l'uso di D invece di E non dà maggiore robustezza crittografica. Entrambe le versioni di Triplo DES sono resistenti agli attacchi meet-in-the-middle (si veda l'Esercizio 6). Tuttavia, ci sono altri attacchi alla versione a due chiavi ([Merkle-Hellman] e [van Oorschot-Wiener]) che indicano possibili debolezze, benché essi richiedano talmente tanta memoria da essere inattuabili.

Un altro rafforzamento di DES è stato proposto da Rivest. Si scelgono tre chiavi K_1, K_2, K_3 e si esegue $K_3 \oplus E_{K_2}(K_1 \oplus m)$. In altre parole, si modifica il testo in chiaro sommandolo *XOR* con K_1, poi si applica DES con K_2, poi si somma *XOR* il risultato con K_3. Questo metodo, noto come DESX, è risultato essere abbastanza sicuro. Si veda [Kilian-Rogaway].

Un altro approccio è quello di usare una nuova famiglia di algoritmi di cifratura. Nel 1998 NIST chiese di esprimere un giudizio su 15 algoritmi candidati a prendere il posto di DES come nuovo standard di cifratura, quello che poi sarebbe stato l'Advanced Encryption Standard (AES). Nel 2000 venne scelto Rijndael per essere l'AES. Esso verrà descritto nel prossimo capitolo.

4.7 Attacchi meet-in-the-middle

Alice e Bob stanno usando un metodo di cifratura. Le funzioni di cifratura sono chiamate E_k e le funzioni di decifrazione sono chiamate D_k, dove k è una chiave. Supponiamo che la conoscenza di k porti alla conoscenza anche di E_k e di D_k (così Alice e Bob possono usare uno dei sistemi classici a chiave non pubblica, come DES). Hanno una grande idea. Invece di cifrare una volta, usano due chiavi k_1 e k_2 e cifrano due volte. Partendo con un messaggio di testo in chiaro m, il testo cifrato è $c = E_{k_2}(E_{k_1}(m))$. Per decifrare, semplicemente si calcola $m = D_{k_1}(D_{k_2}(c))$. Eva avrà bisogno di scoprire entrambe le chiavi k_1 e k_2 se vuole decifrare i loro messaggi.

Ma questo fornisce una sicurezza maggiore? Per molti crittosistemi, applicare due cifrature è lo stesso che usare una cifratura per qualche altra chiave. Per esempio, la composizione di due funzioni affini è ancora una funzione affine (si veda l'Esercizio 2.5). Analogamente, usare due cifrature RSA (con lo stesso n) con esponenti e_1 ed e_2 corrisponde a eseguire un'unica cifratura con esponente e_1e_2. In questi casi, la cifratura doppia non offre alcun vantaggio. Tuttavia, ci sono sistemi, come DES (si veda il Paragrafo 4.4), dove la composizione di due cifrature non si riduce semplicemente a una cifratura con un'altra chiave. Potrebbe sembrare che per questi sistemi la cifratura doppia offra un livello di sicurezza molto più alto. Tuttavia, il seguente attacco mostra che in realtà le cose non stanno così, almeno se si ha un computer dotato di molta memoria.

Supponiamo che Eva abbia intercettato un messaggio m e un testo $c = E_{k_2}(E_{k_1}(m))$ doppiamente cifrato. Vuole trovare k_1 e k_2. Prima calcola e salva $E_k(m)$ per tutte le possibili chiavi k. Poi calcola $D_k(c)$ per tutte le possibili chiavi k. Infine, confronta le due liste. Questo attacco è simile a un attacco del compleanno (Paragrafo 8.4), tranne per il fatto che Eva sa che deve esserci almeno una corrispondenza, poiché la coppia corretta di chiavi deve essere una di esse. Se ci sono più corrispondenze, allora prende un'altra coppia testo in chiaro–testo cifrato e determina quale delle coppie che ha trovato cifra il testo in chiaro nel testo cifrato. Questo dovrebbe ridurre di molto la lista. Se c'è ancora più di una coppia rimanente, continua finché non resta solo una coppia (o decide che due o più coppie danno la stessa funzione di cifratura doppia). Eva ora ha la coppia k_1, k_2 desiderata.

Se Eva ha solo una coppia di testo in chiaro e testo cifrato, ha ancora ridotto l'insieme delle possibili coppie di chiavi a una lista più corta. Se in futuro intercettasse una trasmissione, potrebbe provare ognuna di queste possibilità e ottenere una lista molto corta di testi in chiaro dotati di senso.

Se ci sono N possibili chiavi, Eva deve calcolare e salvare N valori $E_k(m)$. Poi deve calcolare altri N numeri $D_k(c)$ e confrontarli con la lista salvata. Ma questi $2N$ calcoli (più i confronti) sono molto meno degli N^2 calcoli richiesti per una ricerca tra tutte la coppie di chiavi k_1, k_2.

Questa procedura meet-in-the-middle richiede poco tempo in più di una ricerca esaustiva tra tutte le chiavi per una singola cifratura. Inoltre richiede molta memoria per salvare la prima lista. Tuttavia, la conclusione è che in molte situazioni la cifratura doppia non aumenta significativamente il livello di sicurezza.

Analogamente, si può usare la cifratura tripla, usando terne di chiavi. Un attacco analogo riduce il livello di sicurezza al più a quello che ci si può aspettare ingenuamente da una cifratura doppia, ossia elevando al quadrato il numero possibile di chiavi.

4.8 Sicurezza delle password

Quando si accede a un computer e si inserisce la propria password, il computer controlla che la password appartenga a voi e poi permette l'accesso. Tuttavia, sarebbe piuttosto pericoloso conservare le password in un file sul computer. Chi ottenesse quel file sarebbe in grado di aprire l'account di chiunque. Una soluzione potrebbe essere quella di rendere il file accessibile solo all'amministratore del computer. Ma cosa accade se l'amministratore fa una copia del file poco prima di cambiare lavoro? La soluzione è di cifrare le password prima di salvarle.

Sia $f(x)$ una **funzione unidirezionale**. Questo significa che è facile calcolare $f(x)$, ma è molto difficile risolvere $y = f(x)$ rispetto a x. Una password x può allora essere salvata come $f(x)$, insieme al nome dell'utente. Quando l'utente accede al proprio account e inserisce la password x, il computer calcola $f(x)$ e controlla se coincide con il valore di $f(x)$ corrispondente a quell'utente. Un intruso che ottenesse il file delle password avrebbe solo il valore di $f(x)$ per ogni utente. Per accedere all'account, l'intruso dovrebbe conoscere x, che è difficile da calcolare, poiché $f(x)$ è una funzione unidirezionale.

In molti sistemi, le password cifrate sono conservate in un file pubblico. Quindi chiunque abbia accesso al sistema può ottenere questo file. Supponiamo che la funzione $f(x)$ sia nota. Allora tutte le parole di un vocabolario e molte loro varianti (ottenute, per esempio, scrivendole al contrario) possono essere date in pasto a $f(x)$. Se si confrontano i risultati con il file delle password spesso si ottengono le password di vari utenti.

Questo **attacco del vocabolario** può essere parzialmente prevenuto non rendendo il file delle password disponibile pubblicamente, anche se rimane il problema dell'amministratore del computer in procinto di cambiare lavoro (o che viene licenziato). È quindi necessario avere altri metodi che permettano migliorare la sicurezza delle informazioni. Ecco un altro problema interessante. Può sembrare auspicabile che $f(x)$ possa essere calcolata molto rapidamente. Tuttavia, una $f(x)$ un po' più lenta può rallentare un attacco del vocabolario. Ma rallentare troppo $f(x)$ può anche causare problemi. Se $f(x)$ è ideata per funzionare in un decimo di secondo su un calcolatore molto veloce, l'accesso a un calcolatore più lento potrebbe richiedere un tempo inaccettabile. Non sembra ci sia un modo del tutto soddisfacente di risolvere questo problema.

Un modo per impedire gli attacchi del vocabolario è di usare ciò che è chiamato ***salt***. Ogni password è riempita a caso con ulteriori 12 bit. Questi 12 bit vengono usati per modificare la funzione $f(x)$. Il risultato viene salvato nel file delle password, con il nome dell'utente e i valori del *salt* di 12 bit. Quando un utente inserisce una password x, il computer trova all'interno del file il valore del *salt* per questo utente e lo usa nel calcolo della $f(x)$ modificata, che viene confrontata con il valore conservato nel file. Quando si usa il *salt* e le parole nel vocabolario vengono date in pasto a $f(x)$, devono essere riempite con ognuno dei $2^{12} = 4096$ possibili valori del *salt*. Questo rallenta i calcoli in modo considerevole. Inoltre, supponiamo che un attaccante conservi i valori di $f(x)$ per tutte le parole nel dizionario. Questo potrebbe essere fatto in anticipo, di fatto attaccando contemporaneamente numerosi file di password. Con il *salt*, le richieste di memorizzazione aumentano drasticamente, poiché ogni parola deve essere salvata 4096 volte.

Lo scopo principale del *salt* è quello di bloccare gli attacchi che mirano a trovare la password di una persona a caso. In particolare, rende un po' più sicuro l'insieme delle password scelte in modo poco accurato e questo è auspicabile, visto che molte persone usano password deboli. Il *salt* non rallenta un attacco contro una singola password, tranne l'effetto di prevenire l'uso dei chip DES commerciali[5]. Se Eva vuole trovare la password di Bob e ha accesso al file delle password, trova il valore del *salt* usato per Bob e prova un attacco del vocabolario usando, per esempio, solo il valore del *salt* corrispondente a Bob. Se la password di Bob non è nel vocabolario, l'attacco fallisce ed Eva potrebbe dover ricorrere a una ricerca esaustiva tra tutte le possibili password. In molti schemi di password Unix, la funzione unidirezionale si basa su DES. I primi otto caratteri della password vengono convertiti in formato ASCII a 7 bit (si veda il Paragrafo 2.8). Questi 56 bit diventano una chiave DES. Se la password è più corta di otto simboli, viene completata con degli zeri, per ottenere i 56 bit. Quindi il "testo in chiaro" di tutti zeri viene cifrato usando DES per 25 volte con questa chiave. L'output è conservato nel file delle password. La funzione

$$\text{password} \quad \rightarrow \quad \text{output}$$

è ritenuta unidirezionale. Ossia, si conosce il "testo cifrato," che è l'output, e il "testo in chiaro," che è formato tutto di zeri. Trovare la chiave, che è la password, consiste in un attacco di testo in chiaro noto a DES, che generalmente è ritenuto difficile.

[5]Come spiegato più avanti, la cifratura delle password con *salt* usa un algoritmo DES leggermente modificato, rendendo inutilizzabili i chip DES standard. (*N.d.Rev.*)

Per aumentare la sicurezza, si aggiunge il *salt* nel modo seguente. Il *salt* viene generato come un numero casuale di 12 bit. Si tenga presente che in DES, la funzione di espansione E prende un input R di 32 bit (la parte destra dell'input per il round) e lo espande in $E(R)$ che è formato da 48 bit. Se il primo bit del *salt* è 1, il primo e il venticinquesimo bit di $E(R)$ vengono scambiati tra di loro. Se il secondo bit del *salt* è 1, il secondo e il ventiseiesimo bit di $E(R)$ vengono scambiati tra di loro, e così via fino al dodicesimo bit del *salt*. Se esso è 1, il dodicesimo e il trentaseiesimo bit di $E(R)$ vengono scambiati tra di loro. Quando un bit del *salt* è 0, esso non provoca alcuno scambio di bit. Se il *salt* è tutto zero, allora non si verificano scambi e si lavora con il classico DES. In questo modo, il salt permette di avere 4096 possibili varianti di DES. Uno dei vantaggi di usare il *salt* per modificare DES è che non si possono usare implementazioni di DES in hardware ad alta velocità per calcolare la funzione unidirezionale quando si esegue un attacco del vocabolario. Invece, si dovrebbe ideare un chip in grado di provare tutte le 4096 modificazioni di DES introdotte dal *salt*. In alternativa l'attacco potrebbe essere effettuato con un software, che è molto più lento. Il *salt* è considerato da molti solo come una misura temporanea. Con l'aumento dello spazio di memoria e della velocità dei computer, un fattore di 4096 invecchia rapidamente. Per questo motivo, sono stati studiati molti nuovi schemi di password da usare nelle implementazioni future.

4.9 Esercizi

1. Si consideri il seguente metodo di cifratura ispirato a DES. Si parte con un messaggio di $2n$ bit. Lo si divide in due blocchi di lunghezza n (una metà sinistra e una metà destra): M_0M_1. La chiave K è formata da k bit, per un intero k. C'è una funzione $f(K,M)$ che prende un input di k bit e di n bit, e produce un output di n bit. Un round di cifratura parte con una coppia M_jM_{j+1}. L'output è la coppia $M_{j+1}M_{j+2}$, dove

$$M_{j+2} = M_j \oplus f(K,M_{j+1}).$$

($\oplus$ significa somma XOR, ossia somma modulo 2 su ogni bit). Questo viene ripetuto per m round e alla fine il testo cifrato è M_mM_{m+1}.

(a) Se si ha una macchina che esegue la cifratura a m round appena descritta, come si può usare la stessa macchina per decifrare il testo cifrato M_mM_{m+1} (usando la stessa chiave K)? Dimostrare che questo metodo di decifrazione funziona.

(b) Si supponga che K sia di n bit, che $f(K,M) = K \oplus M$ e che il processo di cifratura sia formato da $m = 2$ round. Se si conosce solo un testo cifrato, si può dedurre il testo in chiaro e la chiave? Se si conosce un testo cifrato e il corrispondente testo in chiaro, si può dedurre la chiave? Giustificare le risposte.

(c) Si supponga che K sia di n bit, che $f(K,M) = K \oplus M$ e che il processo di cifratura sia formato da $m = 3$ round. Perché questo sistema non è sicuro?

2. Come descritto nel Paragrafo 4.8, un modo comune per salvare le password su un computer è di usare DES impiegando la password come chiave per cifrare un testo in chiaro fissato (usualmente $000\cdots0$). Il testo cifrato viene poi salvato in un file. Quando si esegue l'accesso, la procedura viene ripetuta e i testi cifrati vengono confrontati. Perché questo metodo è più sicuro del metodo apparentemente analogo in cui si usa la password come testo in chiaro e si usa una chiave fissa (per esempio $000\cdots0$)?

3. Mostrare che le procedure di decifrazione date per le modalità CBC e CFB eseguono effettivamente le decifrazioni desiderate.

4. Data una stringa $\mathcal{S}$ di bit, sia $\overline{\mathcal{S}}$ la stringa complementare ottenuta cambiando tutti gli 1 in 0 e tutti gli 0 in 1 (equivalentemente, $\overline{\mathcal{S}} = \mathcal{S} \oplus 11111\cdots$). Mostrare che se la chiave DES K cifra P in C, allora $\overline{K}$ cifra $\overline{P}$ in $\overline{C}$. (*Suggerimento:* questo non ha nulla a che fare con la struttura degli S-box. Per risolvere questo problema, basta lavorare sull'algoritmo di cifratura.)

5. **(a)** Sia $K = 111\cdots111$ la chiave DES formata tutta da 1. Mostrare che se $E_K(P) = C$, allora $E_K(C) = P$. Così cifrare due volte con questa chiave riporta al testo in chiaro.

(b) Trovare un'altra chiave con la stessa propertà di K in (a).

6. Triplo DES viene eseguito scegliendo due chiavi K_1, K_2 e calcolando $E_{K_1}(E_{K_2}(E_{K_2}(m)))$ (l'ordine delle chiavi è stato modificato rispetto alla versione ordinaria di Triplo DES a due chiavi). Mostrare come attaccare questa versione modificata con un attacco meet-in-the-middle.

7. Siano E^1 ed E^2 due metodi di cifratura e siano K_1 e K_2 le chiavi. Si consideri la cifratura doppia

$$E_{K_1,K_2}(m) = E^1_{K_1}(E^2_{K_2}(m)).$$

(a) Se si conosce una coppia testo in chiaro-testo cifrato, mostrare come eseguire un attacco meet-in-the-middle a questa cifratura doppia.

(b) Una cifratura affine data da $x \mapsto \alpha x + \beta \pmod{26}$ può essere considerata come una cifratura doppia, dove la prima cifratura consiste nel moltiplicare il testo in chiaro per α e la seconda consiste in uno shift di β. Si ha un testo in chiaro e un testo cifrato sufficientemente lunghi da rendere α e β unici. Mostrare che l'attacco meet-in-the-middle della parte (a) richiede al più 38 passi (senza contare i confronti tra le liste). Questo metodo è molto più veloce di una ricerca di forza bruta tra tutte le 312 chiavi.

8. Lo schema di Feistel viene modificato nel modo seguente. Si divide il testo in chiaro in tre blocchi uguali: L_0, M_0, R_0. La chiave per l'i-esimo round è K_i ed f è una qualche funzione che produce un output di lunghezza appropriata. L'i-iesimo round di cifratura è dato da

$$L_i = R_{i-1}, \qquad M_i = L_{i-1}, \qquad R_i = f(K_i, R_{i-1}) \oplus M_{i-1}.$$

Questo continua per n round. Si consideri l'algoritmo di decifrazione che parte con il testo cifrato A_n, B_n, C_n e usa l'algoritmo

$$A_{i-1} = B_i, \qquad B_{i-1} = f(K_i, A_i) \oplus C_i, \qquad C_{i-1} = A_i.$$

Questo continua per n round, fino ad A_0, B_0, C_0. Mostrare che $A_i = L_i$, $B_i = M_i$, $C_i = R_i$ per tutti gli i, ossia che l'algoritmo di decifrazione fornisce il testo in chiaro. (*Osservazione:* l'algoritmo di decifrazione è simile all'algoritmo di cifratura, ma non può essere implementato sulla stessa macchina così facilmente come nel caso del sistema Feistel.)

9. Si consideri la seguente versione semplificata della modalità CFB. Il testo in chiaro è spezzato in blocchi di 32 bit: $P = [P_1, P_2, \ldots]$, dove ogni P_j è formato da 32 bit, invece che dagli 8 bit usati in CFB. La cifratura procede come segue. Si sceglie un X_1 iniziale di 64 bit. Poi, per $j = 1, 2, 3, \ldots$, si eseguono le seguenti operazioni:

$$C_j = P_j \oplus L_{32}(E_K(X_j))$$
$$X_{j+1} = R_{32}(X_j) \parallel C_j,$$

dove $L_{32}(X)$ indica i 32 bit più a sinistra di X, $R_{32}(X)$ indica i 32 bit più a destra di X e $X \| Y$ indica la stringa ottenuta scrivendo X seguito da Y.

(a) Trovare l'algoritmo di decifrazione.

(b) Il testo cifrato è formato da blocchi $C_1, C_2, C_3, C_4, \ldots$ di 32 bit. Per un errore di trasmissione C_1 viene ricevuto come $\widetilde{C}_1 \neq C_1$, mentre C_2, C_3, C_4, ... vengono ricevuti correttamente. Questo testo cifrato alterato viene poi decifrato ottenendo i blocchi di testo in chiaro $\widetilde{P}_1, \widetilde{P}_2, \ldots$. Mostrare che $\widetilde{P}_1 \neq P_1$ e che $\widetilde{P}_i = P_i$ per ogni $i \geq 4$. Quindi l'errore intacca solo tre blocchi della decifrazione.

10. La modalità CBC ha la proprietà di ripristinare il funzionamento corretto dopo un errore nei blocchi di testo cifrato. Mostrare che se si verifica un errore nella trasmissione di un blocco C_j, ma tutti gli altri blocchi sono trasmessi correttamente, allora questo intacca solo due blocchi per la decifrazione. Quali sono questi due blocchi?

11. Sia $E_K(M)$ la cifratura DES di un messaggio M mediante la chiave K. Nell'Esercizio 4 si è mostrato che DES possiede la proprietà della complementazione, ossia che se $y = E_K(M)$ allora $\overline{y} = E_{\overline{K}}(\overline{M})$, dove $\overline{M}$ è il complementare (bit a bit) di M. In altre parole, i complementari della chiave e del testo in chiaro danno il complementare del testo cifrato DES. Spiegare come un avversario può usare questa proprietà in un attacco di forza bruta di testo in chiaro scelto per ridurre il numero di chiavi che dovrebbero essere provate da 2^{55} a 2^{54}. (*Suggerimento:* considerare un insieme di testi in chiaro scelti di (M_1, C_1) e $(\overline{M_1}, C_2)$).

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.*)