Sicurezza su Reti	**Anno Acc. 1998-99**
PROF. ALFREDO DE SANTIS	

DATA ENCRYPTION STANDARD

a cura di Silvia Adinolfi e Anna Maria Bartilomo *31 Maggio 1999*

Indice

1 Un po' di storia

All'inizio degli anni settanta la crittografia ad uso privato veniva utilizzata senza che esistessero delle pubblicazioni specifiche in materia. Si sapeva, ad esempio, che i sistemi crittografici ad uso militare utilizzavano speciali apparecchiature delle quali si conosceva ben poco.

I prodotti in commercio venivano utilizzati da piccole aziende, e a causa delle conoscenze cosí frammentarie i sistemi venduti offrivano scarse garanzie in fatto di sicurezza ed affidabilità. Nel 1972 negli Stati Uniti il *National Bureau of Standard* (NBS), una parte dell'*US Department of Commerce*, iniziò un programma per la stesura di uno standard per la protezione dei dati.

In effetti questo atto fu la risposta ad una legge del 1969 nota come *Brooks Act*, che sottolineava l'esigenza di nuovi standard in materia di sicurezza dati al fine di una migliore utilizzazione dei calcolatori del governo.

L'*Institute for Computer Science and Tecnology* (ICST), una delle maggiori unità operative dell'NBS, produsse delle linee guida e degli standard. Le linee guida (che includevano aree come la sicurezza fisica) erano adeguate a contesti in cui non si richiedeva l'interoperabilità tra computer. Gli standard si occupavano di aree come la crittografia, l'autenticazione, il controllo degli accessi, l'immagazzinamento protetto dei dati e la trasmissione, e potevano effettivamente riguardare ambiti di interoperabilità.

Gli standard erano concepiti in differenti "sfumature"[25]:

Basic Standard: anche detto "standard di buona pratica", specificava funzioni generiche che richiedevano di raggiungere un insieme di risultati comuni.

Interoperability Standard: definiva i formati e le funzioni che permettevano ai dati di poter essere utilizzati su diversi calcolatori. A questo livello non sono prese in considerazione le implementazioni (hardware, software o firmware) né tantomeno le strutture dati.

Interface Standard: specificava non soltanto le funzioni e il formato dei dati attraverso l'interfaccia, ma includeva specificazioni fisiche, elettriche e logiche sufficienti a rimpiazzare un' implementazione (dispositivo o programma) dall'altro lato dell' interfaccia con una seconda implementazione.

Implementation Standard: specificava non soltanto i formati, le funzioni e le interfacce, ma anche le strutture e i metodi di implementazione. A questo livello potrebbe essere necessario garantire che alcune secondarie caratteristiche come la velocità, la regolarità di funzionamento e la sicurezza fisica, debbano andare incontro a particolari necessità.

Furono poi prese in considerazione le diverse "sfumature" di standard per definire l'ambiente in cui verrà sviluppato il sistema che poi diventerà il DES.

La *Implementation Standard* fu rigettata perché avrebbe scoraggiato i rivenditori all'utilizzo di nuove tecnologie.

Sebbene in alcune applicazioni era conveniente avere le interfacce definite, anche la *Interface Standard* fu rigettata perché troppo restrittiva (avrebbe lasciato fuori troppe interessanti applicazioni).

Il *Basic Standard* non riguardava l'interoperabilità e fu subito esclusa.

Fu scelto l'*Interoperability Standard* per specificare il DES, richiedendo specificazioni complete di funzioni di base e suoi formati, mantenendo comunque l'indipendenza dall'implementazione fisica.

L'NBS iniziò lo sviluppo del DES quando iscrisse nel *Federal Register* (15 Maggio 1973) una sollecitazione a concentrarsi sugli algoritmi di crittografia per la protezione dei dati sui computer. L'eco che ebbe questo invito dimostrò che c'era molto interesse nello sviluppo di uno standard, ma fu denunciato il fatto che in materia di crittografia le tecnologie disponibili erano poche.

La pubblicazione del 15 Maggio 1973 suggeriva dei requisiti che lo standard doveva possedere:

- un alto grado di sicurezza;
- specifica completa e semplicità di comprensione;
- segretezza basata sulla chiave e non sull'algoritmo;
- disponibilità per tutti gli utenti;
- adattabilità per diverse applicazioni;
- implementabilità economica su dispositivi differenti;
- efficienza;
- possibilità di essere certificato;
- esportabilità.

Già alla fine degli anni '60 l'IBM iniziò un progetto di ricerca nell' ambito della crittografia applicata ai computers. Il risultato di questo lavoro fu il sistema **LUCIFER** [26] (che utilizzava una chiave a 128 bit). L'attività di ricerca continuò e i risultati raggiunti apparvero in una serie di pubblicazioni e documenti ufficiali. Uno degli algoritmi prodotti stava per diventare il DES.

L'IBM sottomise il suo algoritmo all'NBS in seguito ad una seconda sollecitazione iscritta nel *Federal Register* il 27 Agosto 1974. L'NBS a sua volta chiese che fosse la *National Security Agency* (NSA) a testare l'algoritmo riguardo ad un insieme informale di requisiti. Contemporaneamente richiese che l'IBM accordasse diritti non esclusivi e senza copyright per costruire e vendere apparecchiature che implementassero l'algoritmo.

A distanza di due anni circa dalla prima sollecitazione, l'NBS iscrisse due note nel *Federal Register* il 17 Marzo 1975.

Nella prima venne pubblicata per intero la proposta **Encription Algorithm for Computer Data Protection**. L'NBS dichiarò che questo soddisfaceva i primari requisiti per un algoritmo DES. Fece anche notare che alcuni documenti nazionali e stranieri contenevano rivendicazioni sulla paternità dell' algoritmo. Il candidato al DES rischiava di cadere sotto il controllo dello straniero.

La seconda nota conteneva una dichiarazione dell'IBM che accordava le richieste avanzate dall'NSA circa l'esclusività e il copyright, purché il *Department of Commerce* definisse l'algoritmo proposto come DES.

L'1 Agosto 1975, l'NBS iscrisse nel *Federal Register* la quarta nota riguardo lo standard. Le agenzie federali e tutto il pubblico interessato richiedevano commenti circa lo standard proposto, ma nonostante le pressioni l'NBS continuò (e continua tutt'oggi) a valutare l'algoritmo, le richieste di sicurezza nel settore pubblico e privato e le alternative allo standard.

Nell'ottobre dello stesso anno Hellman presentò le sue critiche asserendo che, a dispetto delle 2^{56} chiavi necessarie per un attacco brutale dell' algoritmo, la metà di esse avrebbe permesso in media la violazione di DES [12] [17].

Infatti, la sicurezza dipendeva fortemente dalla chiave e quindi variarne la lunghezza era di vitale importanza per assicurare un grado minimo di affidabilità.

Una seconda critica all'algoritmo venne dall'uso delle S-box nel processo di cifratura e decifratura, perché si credeva che i valori delle S-box fossero stati scelti in modo da nascondere qualche tranello. Per questo motivo l'NBS (oggi NIST) organizzò due workshop allo scopo di analizzare eventuali tranelli e punti oscuri dell'algoritmo, e per valutare la possibilità di incrementare la lunghezza della chiave. Ai workshop vennero invitati implementatori, progettisti, acquirenti e matematici affiché si potesse dar vita ad uno standard che soddisfacesse tutte le richieste da parte degli addetti ai lavori.

Il risultato fu la approvazione di DES come standard il cui algoritmo non presentava alcun tipo di tranello. L'NBS evidenziò, tuttavia, la necessità di rivedere il DES periodicamente.

L'NBS continuò a valutare l'algoritmo DES che venne publicato definitivamente il 15 gennaio 1977 [21].

Ma in che termini DES venne adottato come standard?

L'ANSI lo adottò come uno standard (X3.92) nel settore privato, chiamandolo **Data Encryption Algorithm** e pubblicando i modi standard con cui esso poteva operare [1].
Altri due gruppi dell'ANSI svilupparono degli standard basati su DES per il settore bancario e delle transazioni finanziarie.
Anche la *General Services of Administration* (GSA) formulava i suoi standard per le telecomunicazioni con l'ausilio di DES, prima che il *Computer Security Act* del 1987 trasferisse l'onere al NIST [16].

Nonostante il Dipartimento del Tesoro non adottò il DES quale standard per usi crittografici, il gruppo ISO per gli standard commerciali e finanziari lo adottò per le autenticazioni [10].

Nel 1992 il NIST sollecitò ulteriori commenti sul DES proponendo come alternative [23]:

- la riaffermazione dello standard DES;
- il ritiro del DES con la sua sostituzione;
- modifiche del DES (che continuava comunque ad essere lo standard).

Il risultato della discussione fu la certificazione del DES per ulteriori cinque anni. La successiva sessione dei lavori fu fissata per il 1998.
Prima della scadenza dei 5 anni sono state indette da una società americana l' RSA Data Security, Inc. delle vere e proprie gare, dette challenges, per rompere il DES. Tale società offriva 10.000 dollari a chi per primo rompeva la sua challenge a patto che venisse rotta entro il 25% del miglior tempo precedente. Questa challenge è stata rotta nel 1997, nel 1998 ed anche nel 1999:

- nel Luglio del 1997 furono impiegati 39 giorni e fu testato il 24% di tutte le possibili chiavi;
- nel Luglio del 1998 una società Americana l'EFF spendendo 250.000 dollari riuscí a costruire una macchina per rompere il DES impiegando solo 56 ore;

- nel Gennaio 1999 sono stati impiegati 22 ore e 15 minuti testando 245 miliardi di chiavi per secondo.

Queste challenges sono un modo spettacolare per svalutare il DES dal punto di vista della sicurezza. Intanto il NIST aveva lanciato il bando per l'identificazione del successore del DES, che si chiamerà AES (*Advanced Encryption Standard*) e si pensa sarà utilizzato nel 2000.
Il sito internet dell' AES è: http://www.nist.gov/aes/ . Tra le richieste all'AES vi sono quelle di resistere alla crittoanalisi differenziale e, di avere una chiave minima di 128 bit, in modo che resista agli attacchi di forza bruta per un lunghissimo periodo di tempo, anche tenendo in conto un incremento nella potenza di calcolo pari a quello mantenuto negli ultimi venti anni. Ulteriori notizie sulla storia del DES sono reperibili in [25].

2 Descrizione del DES

Il sistema DES cifra una stringa x di 64 bit in una stringa y di ancora 64 bit utilizzando una chiave K di 56 bit. Cioè

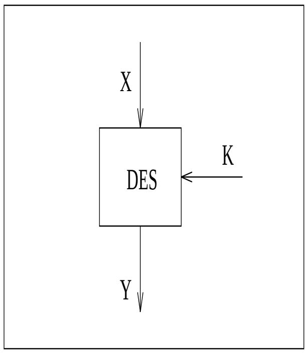


Figura 1: Funzionamento DES.

Formalmente possiamo scrivere :

$$DES : \{0,1\}^{64} \times \{0,1\}^{56} \to \{0,1\}^{64}.$$

La chiave K in realtà è memorizzata sfruttando 64 bit, dove l'ottavo, il 16-esimo,$\cdots$, il 64-esimo, sono i bit di parità[1] per i sette bit precedenti. Possiamo immaginare i 64 bit della chiave K divisi in 8 byte, dove per ognuno di questi l'ottavo bit definisce la parità del byte (Figura 2). Tuttavia concretamente ci sono solo 56 bit su cui puó variare la chiave.

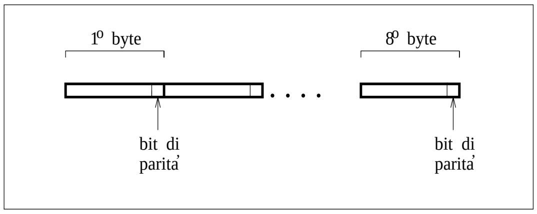


Figura 2: Struttura della chiave.

[1] Se il numero di 1 è pari, allora il bit di parità vale 1.

La struttura del DES è la seguente:

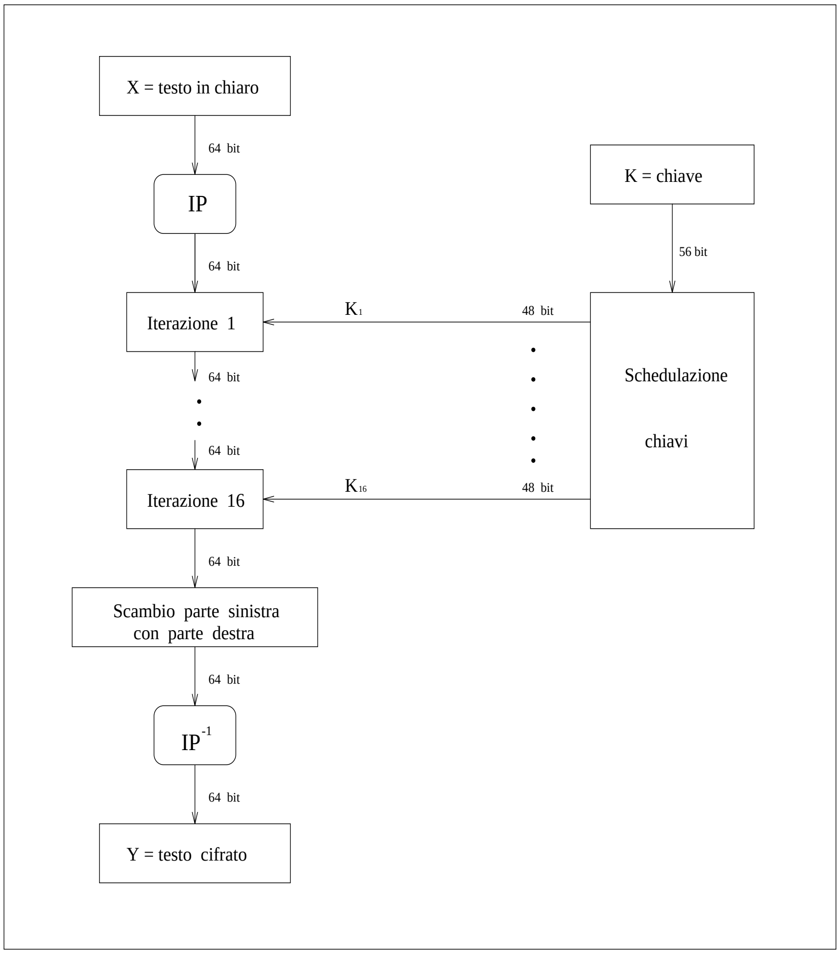


Figura 3: Struttura del DES.

Dove IP è una permutazione di 64 bit detta *permutazione iniziale*; le **sedici iterazioni** che trasformano la stringa in input sono uguali nel senso che la funzione che le realizza è sempre la stessa (vedremo piú avanti che l'unica cosa che cambia ad ogni iterazione è la chiave che viene utilizzata); **scambio parte sinistra con parte destra** effettua uno scambio sulla stringa prodotta come output della 16-esima iterazione, in particolare scambia i 32 bit piú a sinistra con i 32 bit piú a destra); IP^{-1} è la *permutazione inversa* della permutazione iniziale; la **schedulazione delle chiavi**, sfruttando la sola chiave K, produce le chiavi $K_1, K_2, \cdots, K_{16}$ di 48 bit ciascuna, per le iterazioni da 1 a 16.

Piú precisamente l'algoritmo di cifratura procede nel seguente modo:

Passo 1 Sia x un testo in chiaro di 64 bit: l'algoritmo costruisce una stringa binaria x_0 permutando la stringa iniziale mediante la permutazione iniziale fissata IP:

$$x_0 = IP(x) = L_0R_0$$

dove L_0 e R_0 sono due stringhe binarie che comprendono rispettivamente i primi e gli ultimi 32 bit della stringa originaria x_0.

Passo 2 Effettua 16 *iterazioni* (detti anche *round*).

Per $1 \leq i \leq 16$ l'input della i-esima iterazione è una stringa di 64 bit divisa in due parti, L_{i-1} e R_{i-1}, ognuna di 32 bit (chiamate rispettivamente *parte sinistra* e *parte destra* della stringa).

I valori di output L_i e R_i sono prodotti in accordo alla seguente regola:

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

L'operatore $\oplus$ denota la funzione xor bit a bit delle due stringhe di bit, ed f è una funzione che, data la chiave K_i e la sottostringa R_{i-1}, produce una stringa lunga 32 bit (vedi Figura 5). Descriveremo f in dettaglio successivamente. $K_1, K_2, \cdots, K_{16}$ sono le chiavi prodotte dal processo di schedulazione (anche questo sarà illustrato piú avanti).

All'i-esima iterazione verrà usata la sola chiave K_i.

Passo 3 L'output dell'algoritmo è ottenuto applicando la permutazione IP^{-1} a $R_{16}L_{16}$.

Si noti l'ordine inverso delle ultime due stringhe di bit R_{16} e L_{16} dovuto a *scambio di 32 bit.*

La permutazione IP definita dallo standard DES è fissa ed è specificata dalla Tabella 1.

IP							
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

Tabella 1: Permutazione Iniziale.

Vediamo questa tabella come si legge. Il primo valore della tabella, 58, specifica che il primo bit della stringa permutata $IP(x)$ è il 58-esimo bit della stringa da permutare x; il secondo elemento della tabella, 50, ne specifica il secondo e cosí via. Un esempio di applicazione della permutazione IP ad una stringa x è illustrato in Figura 2.

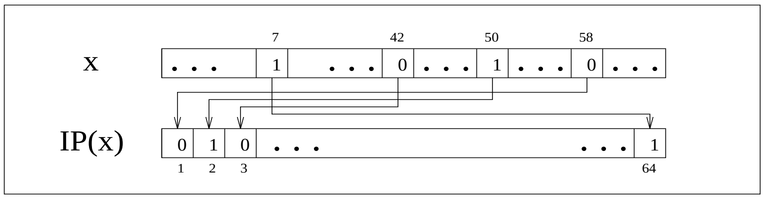


Figura 4: Esempio di applicazione della permutazione IP.

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

Tabella 2: Permutazione inversa di IP.

La permutazione inversa IP^{-1} è mostrata in Tabella 2.

Questa tabella si legge nello stesso modo della precedente.

In effetti non è ancora chiaro perché è stata scelta proprio questa particolare permutazione e non un'altra, tuttavia la sicurezza di questo schema non dipende dalla scelta della permutazione, ma è solo una scelta dello standard.

Potremmo infatti fare a meno della permutazione iniziale e finale senza perdere in sicurezza, anche se in questo modo verremmo meno a quello che è lo standard del DES per le comunicazioni.

Riportiamo in Figura 5 le operazioni eseguite all'iterazione i.

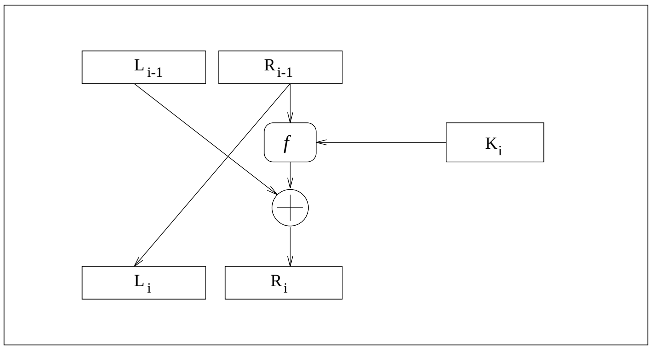


Figura 5: L'iterazione i della cifratura DES.

2.1 La funzione f

La funzione f è forse l'aspetto piú intricato dell'algoritmo DES a causa della presenza delle S-box. Illustriamo questa funzione:

$$f : \{0,1\}^{32} \times \{0,1\}^{48} \rightarrow \{0,1\}^{32}.$$

La funzione f prende in input una stringa A di 32 bit, una stringa J di 48 bit, e fornisce in output una stringa di 32 bit.

La computazione realizzata da f è rappresentata dalla Figura 6.

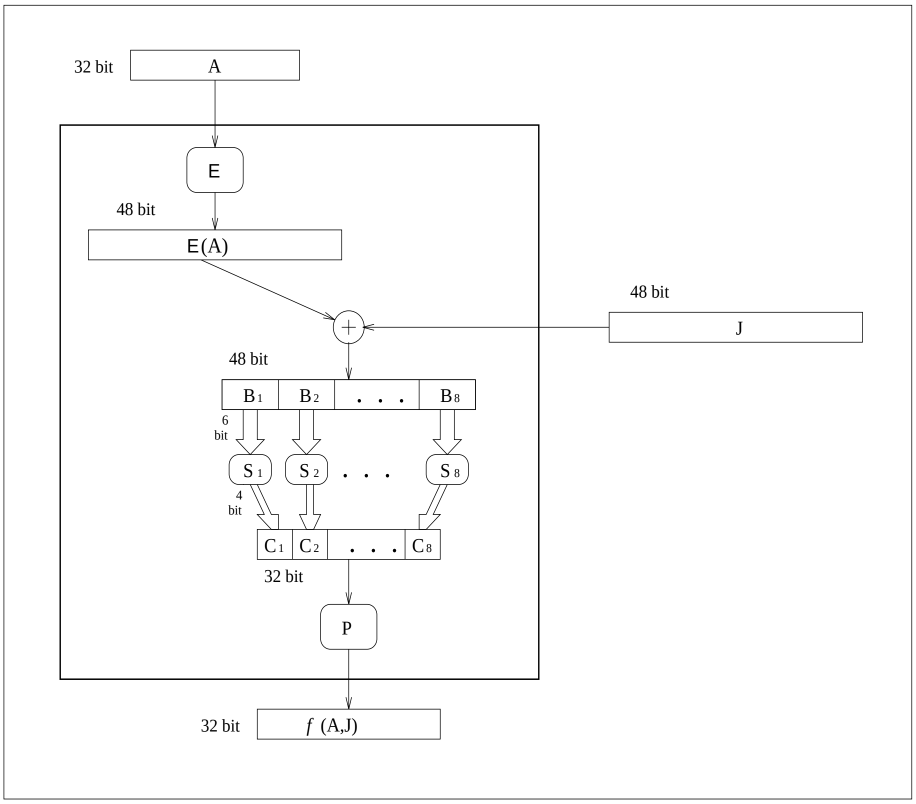


Figura 6: Schema della funzione f.

Descriviamo in dettaglio la figura:

1. La stringa A a 32 bit è espansa dalla funzione E in una stringa 48 bit. Possiamo immaginare che E prima duplica 16 bit dei 32 di A, poi permuta la stringa ottenuta.

2. Viene computato lo xor bit a bit tra J e $E(A)$, che è una stringa di 48 bit. La sequenza viene divisa in 8 sottosequenze di 6 bit ciascuna, dette sottosequenze $B_1, B_2, \cdots, B_8$.

3. L'S-box S_i (per $1 \leq i \leq 8$) prende in input B_i (a 6 bit) e da in output una sequenza a 4 bit, che chiamiamo C_i.

4. La stringa $C_1 C_2 \cdots C_8$ (a 32 bit) viene permutata tramite la permutazione P, e questo è il valore $f(A, J)$.

E bit-selection table					
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

Tabella 3: Espansione E.

La funzione di espansione E viene realizzata in accordo con la Tabella 3.

Anche questa tabella si legge come le altre. Si noti come le prime due colonne di E ricopino nella stringa finale gli stessi bit ricopiati dalle ultime due colonne.

Ecco invece la tabella che specifica la permutazione P:

P			
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

Tabella 4: Permutazione P.

2.2 Le S-box di DES

Fino ad ora sono state utilizzate funzioni di tipo lineare (cioè gli xor, le permutazioni e le espansioni), e se un sistema fosse realizzato utilizzando solo funzioni di questo tipo sarebbe molto facile romperlo. Ricordiamo cosa accadeva col cifrario di Hill, che utilizzava una funzione lineare, e poteva essere rotto abbastanza facilmente utilizzando un attacco di tipo *known plaintext*.

L'unico punto in cui il sistema DES non utilizza funzioni di tipo lineare è nelle S-box, che costituiscono quindi il fulcro dell'algoritmo. Riportiamo qui di seguito le tabelle che specificano le S-box:

S_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_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_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_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_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_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_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_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

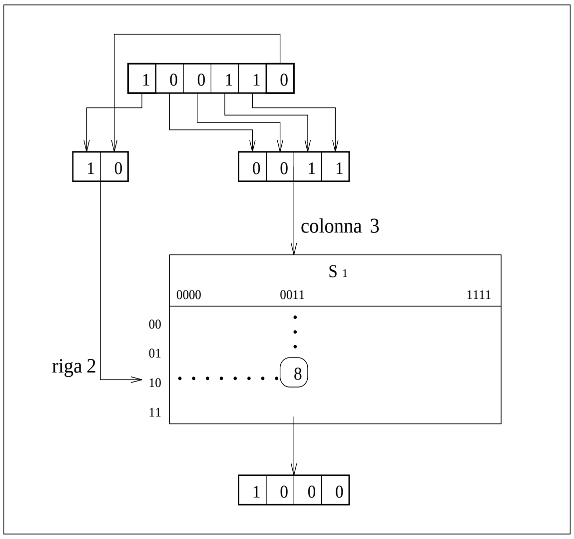


Figura 7: Esempio di applicazione di S_1.

Vediamo come si leggono. Dato B_i (l'input a 6 bit dell'S-box S_i), il primo e l'ultimo bit di B_i vengono interpretati come indice di riga, mentre i bit centrali come indice di colonna. La prima riga di S_i è la riga 00, la seconda riga è la 01, la terza riga è la 10 e la quarta riga è la 11. Per le colonne una indicizzazione analoga: la prima colonna è la colonna 0000, la seconda colonna è la colonna 0001,$\cdots$, l'ultima colonna è la colonna 1111. Si consulta la tabella S_i alle coordinate prescritte e il valore in uscita dall'S-box è l'equivalente binario del decimale trovato. In Figura 7 vi è un esempio di applicazione di S_1 sull'input $B_1 = 100110$.

Come abbiamo già detto in precedenza le S-box appaiono a molti come qualcosa di misterioso: alcuni asseriscono addirittura che esse nascondano in sé delle trapdoor. Tali sospetti nascono dal fatto che i criteri di progettazione delle S-box non sono al momento completamente noti.

Ad ogni modo nel '76 l'NSA rese note alcune proprietà di cui le S-box godevano:

1. Ogni riga deve essere una permutazione degli interi 0,..,15.
2. Nessuna S-box è una funzione affine o lineare dei suoi input.
3. Cambiando un solo bit di input ad una S-box devono variare almeno due bit nell'output.
4. Per ogni S-box e per ogni input x a 6 bit, $S(x)$ e $S(x \oplus 001100)$ differiscono in almeno due bit.
5. Per ogni S-box, per ogni input x e per ogni $d, g \in \{0, 1\}, S(x) \neq S(x \oplus 11dg00)$.
6. Per ogni S-box, se fissiamo un bit di input e osserviamo il valore di un fissato bit di output, il numero degli input per i quali il bit di output vale 0 è circa uguale al numero degli input per i quali tale bit vale 1.

Le ultime due proprietà sono conseguenza dei criteri di progettazione. In tutti i casi non è noto se altre proprietà furono usate nel progetto delle S-box.

2.3 Schedulazione delle chiavi

Le chiavi usate durante i vari round sono estratte dalla chiave K che noi stabiliamo. Ricordiamo che K è una chiave di 64 bit di cui 8 utilizzati per il controllo di parità. Tali bit sono quelli in posizione $8, 16, 24, \cdots, 64$ e sono tali che ogni byte della chiave contenga un numero dispari di bit 1. Ciò significa che solo un errore puó essere rivelato in ogni byte.

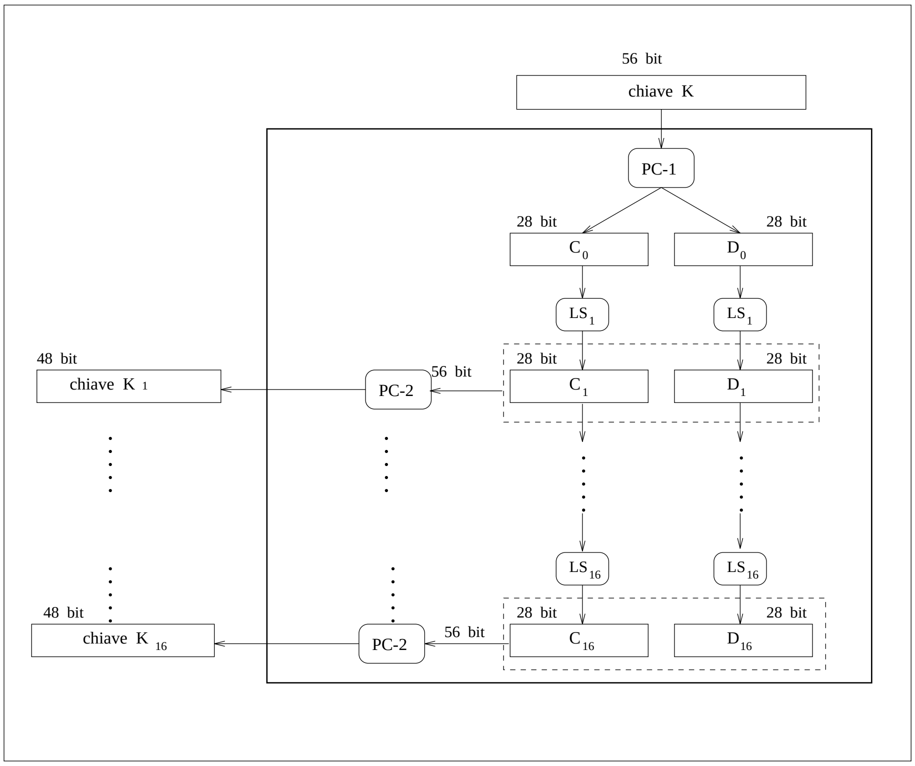


Figura 8: Schedulazione delle chiavi.

Vediamo ora piú precisamente come vengono ricavate le chiavi usate durante le iterazioni (Figura 8).

1. Data una chiave K di 64 bit, non consideriamo gli 8 bit di parità e permutiamo i rimanenti 56 bit tramite la permutazione PC-1 riportata in Tabella 5. Sia $PC\text{-}1(K) = C_0D_0$ dove C_0 e D_0 sono i primi e gli ultimi 28 bit di $PC\text{-}1(K)$.

2. Per $1 \leq i \leq 16$ computiamo
$$C_i = LS_i(C_{i-1})$$
$$D_i = LS_i(D_{i-1})$$
$$K_i = PC\text{-}2(C_iD_i)$$
dove LS_i è uno shift ciclico a sinistra di una o due posizioni a seconda del valore di i (lo shift riguarda una sola posizione alle iterazioni 1, 2, 9 e 16, e due posizioni a tutte le altre iterazioni). PC-2 rappresentata in Tabella 6 è una compressione di una stringa di input a 56 bit, in una stringa di output a 48 bit.

Osserviamo che i bit che vengono soppressi dalla compressione sono quelli in posizione 9, 18, 22, 25, 35, 38, 43 e 54 della stringa input. Tutto il procedimento di questo algoritmo è noto (in quanto pubblicato già da una ventina di anni) e chiunque volendo puó seguirlo per effettuare cifrature; l'unica cosa che viene nascosta è il valore della chiave iniziale K.

PC-1						
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

Tabella 5: Permutazione PC-1.

PC-2					
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

Tabella 6: Compressione PC-2.

2.4 La decifratura del DES

Vogliamo trovare l'algoritmo di decifratura DES con chiave K, che indicheremo con DES_K^{-1} per cui valga la seguente proprietà:

$$DES_K^{-1}(DES_K(x)) = x = DES_K(DES_K^{-1}(x))$$

per ogni messaggio in chiaro x e per ogni chiave K. Sebbene DES possa apparire complicato come metodo per cifrare un messaggio in chiaro, nasconde in sé una caratteristica molto importante. DES_K^{-1} è lo stesso algoritmo DES_K eccezion fatta per la schedulazione delle chiavi, che è invertita (le chiavi vengono prodotte inordine inverso). Vediamo perché.

Consideriamo il messaggio cifrato $R_{16}L_{16}$ a meno della permutazione finale IP^{-1}. Riguardando lo schema della 16-esima iterazione di cifratura (Figura 9) si ricavano le seguenti relazioni:

$$L_{16} = R_{15}$$

e

$$R_{16} = L_{15} \oplus f(R_{15}, K_{16}).$$

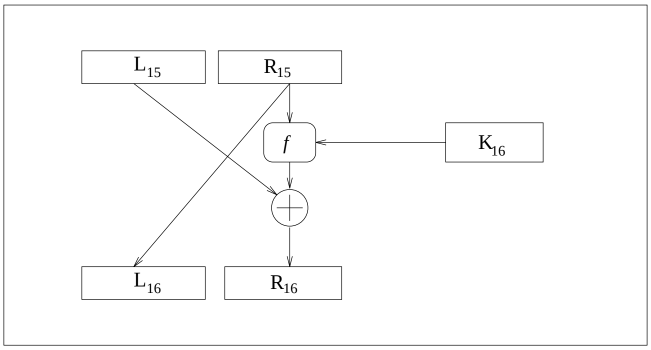


Figura 9: L'iterazione 16 della cifratura DES.

Da qui riscrivendo le due equazioni possiamo ricavare il blocco precedente[2]:

$$R_{15} = L_{16}$$

e

$$L_{15} = R_{16} \oplus f(R_{15}, K_{16}) = R_{16} \oplus f(L_{16}, K_{16}).$$

In conclusione: da R_{16} e L_{16} abbiamo ricavato R_{15} ed L_{15} come si vede in Figura 10.

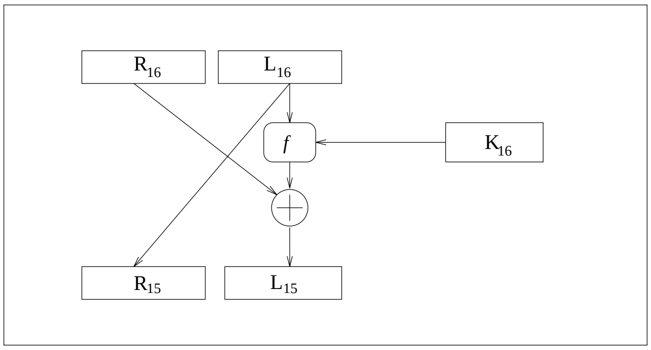


Figura 10: Prima iterazione della decifratura DES.

Quindi, invertendo il ruolo di R ed L e utilizzando le chiavi in maniera inversa, da K_{16} a K_1, si puó ritornare al messaggio in chiaro passando attraverso le seguenti coppie:

$$(R_{16}, L_{16}) \rightarrow (R_{15}, L_{15}) \rightarrow (R_{14}, L_{14}) \rightarrow \cdots \rightarrow (R_0, L_0).$$

Questo è il motivo per cui alla fine dell'algoritmo si effettua lo scambio tra le parti L e R.

Le iterazioni della codifica e della decodifica sono le stesse, ma quello che cambia passando dalla cifratura alla decifratura è l'ordine con cui vengono schedulate le chiavi. Alla prima iterazione servirà la chiave K_{16}, alla seconda la chiave K_{15} e cosí via. Dal momento che per la fase di decifratura, le chiavi devono essere schedulate in ordine inverso alla cifratura, dobbiamo effettuare delle variazioni al dispositivo di schedulazione definito in precedenza.

In particolare dobbiamo fare in modo che pur prendendo in input la chiave K, tale dispositivo produca le chiavi nell'ordine desiderato. Il passo principale per ottenere ciò è quello

[2]Si ricorda che se $a = b \oplus c$ allora $b = a \oplus c$.

di invertire lo shift che veniva applicato alle metà C_iD_i effettuando uno shift ciclico a destra (anziché a sinistra) di due posizioni eccetto all'iterazione 16 (la prima per la decifratura), all'iterazione 9 (l'ottava per la decifratura), all'iterazione 2 e 1 (quindicesima e sedicesima) in cui lo shift è di una posizione.

Ci sono state molte implementazioni sia hardware che software e molti sono stati gli sforzi per velocizzare al massimo le operazioni di cifratura e decifratura [2] [14] [15]. Per il momento la piú veloce implementazione hardware è stata quella prodotta dalla Digital Equipment Corporation: il chip, che supporta le modalità ECB e CBC (vedi *Modalità operative del DES*) è costruito con 50K transistor e puó cifrare o decifrare alla velocità di un Gbit/secondo ossia 15.6 milioni di blocchi di 64 bit al secondo [13].

Un'implementazione software del DES su IBM 3090 mainframe puó effettuare 32.000 cifrature al secondo. Riportiamo di seguito i tempi di alcuni microcomputer in particolare Motorola e Intel (il software è stato scritto in linguaggio C e assembler) [24].

Processore	Frequenza del clock (Mhz)	Parola macchina (in bits)	Numero di blocchi DES (per second)
8088	4.7	8	370
68000	7.6	16	900
80286	6.0	16	1.100
68020	16.0	32	3.500
68030	16.0	32	3.900
80286	25.0	16	5.000
68030	50.0	32	10.00
68040	25.0	32	16.000
68040	40.0	32	23.000
80486	66.0	32	43.000
Sun ELC			26.000
HyperSparc			32.000
RS6000-350			53.000
Sparc 10-52			84.000
DEC Alpha 4000-610			154.000
HP 9000-887	125		196.000

Tabella 7: Prestazioni del DES su vari processori.

3 Critiche al DES

Ci sono state molte critiche al DES, una di queste riguarda i controlli di esportazione.

Il governo degli Stati Uniti mediante l'ITAR (*International Traffic in Arms Regulations*) restringe attualmente l'esportazione di tecnologie crittografiche con chiave di lunghezza superiore a 56 bit (considerate di interesse strategico a fini militari) di fatto causando, secondo alcuni, la difficoltà nel garantire la sicurezza delle comunicazioni.

Alcune aziende statunitensi vedevano sempre più di cattivo occhio questa limitazione, sostenendo che faceva perdere loro di competitività nei confronti di concorrenti europei o asiatici non sottoposti alle regolamentazioni ITAR.

Il nascere poi di sistemi di crittografia a chiave pubblica, nei quali la chiave deve strutturalmente essere più lunga rispetto a quella usata nei sistemi tradizionali, ha contribuito ad aumentare la tensione e le discussioni anche in sede governativa.

Nel dicembre 1996, il Governo Americano offrì formalmente agli esportatori la possibilità di incorporare il DES, ma niente di più, nei loro prodotti. La decisione era che alcune compagnie avrebbero dovuto firmare un accordo col governo che le obbligava ad installare un recupero chiavi nei loro prodotti entro due anni.

La tecnologia per il recupero delle chiavi permetteva al governo americano di decifrare messaggi a volontà, fornendogli una copia della chiave usata in ogni messaggio, in modo che l'utente che usa il prodotto non si accorga di questo nè lo può controllare. Questo faceva sì che il governo avrebbe continuato a vincere ed il riserbo del pubblico a perdere. Inoltre ciò significava che si poteva esportare un prodotto crittografico sotto condizione. Questo divieto di esportare prodotti crittografici se non a condizione, sembrava incoraggiare le richieste di voler liberalizzare le esportazioni crittografiche.

Secono alcuni il governo quindi ingannava il pubblico per quanto riguardava la forza di encryption del DES, incoraggiandolo ad usare il DES, prevenendo l'adozione di standard più sicuri del DES, che avrebbe avuto più difficoltà ad esportare. Infatti esso offriva l'esportabilità del DES per mezzo di un chip commerciale che costava davvero poco per il governo, ma era considerato prezioso.

Altre critiche al DES riguardano la sua sicurezza ed in particolare i seguenti punti:

Le S-box. L'algoritmo di per sé è molto semplice e tutte le funzioni sono funzioni lineari (gli xor e le permutazioni) tranne le S-box. Esse sembrano delle semplici matrici costruite a caso; tuttavia esse rispondono a dei precisi criteri che non sono completamente noti, ed è per questo che a molti appaiono come qualcosa di misterioso. In effetti si teme che l'NSA abbia nascosto in esse delle trapdoor in modo tale da permettere (proprio a quelli dell'NSA) di decifrare facilmente messaggi intercettati.

Chiaramente è difficile (se non impossibile) dimostrare che le S-box contengano delle trapdoor, come pure d'altronde dimostrare che esse non ne contengano.

Tuttavia c'è gente che spera di trovare qualche proprietà nascosta nelle S-box . Ad esempio la S4 ha importanti proprietà: gli ultimi 3 output possono essere derivati nello stesso modo come i primi tre complementando alcuni bit di input.

Due input diversi alle S-box, ma scelti in maniera opportuna, possono produrre lo stesso output.

È possibile inoltre ottenere lo stesso output di un solo ciclo di DES cambiando i bit solo in tre consecutive S-box [9].

Gli shift. Altre critiche riguardano gli shift: perché uno shift di una sola posizione solo alla prima, seconda, nona e sedicesima iterazione, mentre alle altre iterazioni lo shift interessa due posizioni?

Le iterazioni. Perché solo 16 iterazioni e non 24 o 32? *Alan Konheim*, dimostrò che con DES ad 8 iterazioni, il testo cifrato era una funzione random di ogni bit del testo in chiaro e di ogni bit della chiave [19]. Perché allora non fermare DES dopo 8 iterazioni?

E. Biham e A. Shamir, "inventori" della crittoanalisi differenziale hanno dimostrato, nel 90, che con meno di 16 iterazioni si puó rompere il DES con un attacco known-plaintext piú efficiente di un attacco "brute force" che esamina 2^{56} chiavi. Con 16 iterazioni i due attacchi sono equivalenti [3].

Don Coppersmith, ricercatore dell'IBM, quando fu chiamato a rispondere sulla questione affermò che l'IBM conosceva la crittoanalisi differenziale già nel '74 e che quindi progettarono il sistema proprio per evitare attacchi di quel tipo. La diffusione di questa nuova analisi non avvenne per ovvi motivi. In tutti i casi Adi Shamir rispose affermando che fino al '90 non si conoscevano attacchi migliori della crittoanalisi differenziale, e Coppersmith decise di non protrarre ulteriormente la questione decidendo di rimanere in silenzio.

La lunghezza della chiave. La maggiore critica al DES è che la dimensione dello spazio delle chiavi, 2^{56}, è troppo piccolo per essere realmente sicuro. Ci si chiede quindi: perché DES usa una chiave a 56 bit (in effetti 64 poiché bisogna anche aggiungere gli 8 bit di parità) mentre LUCIFER ne usava 128?

Un attacco brute force su una chiave a 128 bit non è nemmeno immaginabile mentre sono noti attacchi brute force con chiave a 64 bit.Notiamo che anche quando il metodo di scelta è la forza bruta, le chiavi raramente sono veramente casuali, e possono essere cercate nell'ordine più probabile.

Uno di questi potrebbe essere l'attacco known-plaintext, in cui dato un blocco di 64 bit di testo in chiaro x, ed il corrispondente testo cifrato y, si vanno a testare tutte le possibili chiavi fino a che se ne trova una per cui $e_k(x) = y$. Si noti che potrebbe esistere piú di una chiave che verifica tale proprietà.

Relativamente a questo esistono limiti pratici alle dimensioni della chiave, che può essere scoperta con una ricerca di forza bruta, ma dal momento che NSA ha deliberatamente limitato la dimensione della chiave del DES a 56 bit, già nel lontano 1970 il DES era rompibile con un algoritmo di forza bruta.

La maggior parte dei cifrari ha una struttura interna molto diversa dal DES, e può essere possibile eliminare il grande numero di possibili chiavi per trarre vantaggio dalla struttura del cifrario.

Una valutazione del 1996, relativa alle dimensioni di chiavi necessarie per la sicurezza, stabiliva che le chiavi dovevano avere un minimo di 75 bit per proteggere contro gli attacchi di forza bruta ed un minimo di 90 bit per proteggere per ulteriori venti anni.

Tutto ciò a dimostrazione del fatto che lo spazio delle chiavi è relativamente piccolo e che comunque esaminando lo spazio delle chiavi in maniera esaustiva servono in media 2^{55} tentativi prima di indovinare la chiave (cioè la metà di quelle possibili).

4 Caratteristiche delle chiavi DES

4.1 Chiavi speciali

Tra le 2^{56} chiavi possibili (gli 8 bit di parità non vengono considerati nella definizione dello spazio delle chiavi, perciò la chiave risulta di 56 bit), ve ne sono di particolari per le quali le sottochiavi prodotte dalla fase di schedulazione sono tutte uguali.
Queste chiavi sono dette **weak key** [7] [6].

Formalmente possiamo dire che :

Definizione 1 *Una weak key DES è una key k tale che* $E_k(E_k(x)) = x$ *per tutti gli* x*, il che equivale ad un'involuzione.*

Quindi se recifriamo un testo cifrato con una weak key otteniamo il testo in chiaro. Questo succede perchè, visto che le sottochiavi prodotte dal processo di schedulazione sono le stesse sia nel caso della cifratura che della decifratura, queste due operazioni coincidono.

Le chiavi weak key sono quelle composte da tutti 0, tutti 1, oppure da una metà da tutti 0 e l'altra da tutti 1 ($K = 0^{56}, K = 1^{56}, K = 0^{28}1^{28}$ $K = 1^{28}0^{28}$). Tali chiavi diminuiscono la sicurezza del sistema in quanto basta trovare una sola sottochiave K_i per scoprire tutte le altre.

In Tabella 8 riportiamo le quattro weak key in rappresentazione esadecimale (la lunghezza di tali chiavi è di 64 bit, tuttavia ricordiamo che ogni sette bit vi è un bit di parità e che la permutazione iniziale modifica leggermente l'ordine delle chiavi).

VALORE DELLA WEAK KEY	C_0	D_0
0101 0101 0101 0101	$\{0\}^{28}$	$\{0\}^{28}$
FEFE FEFE FEFE FEFE	$\{1\}^{28}$	$\{1\}^{28}$
1F1F 1F1F 0E0E 0E0E	$\{0\}^{28}$	$\{1\}^{28}$
E0E0 E0E0 F1F1 F1F1	$\{1\}^{28}$	$\{0\}^{28}$

Tabella 8: Weak Key.

Definizione 2 *Una coppia di semi-weak key DES è una coppia* (K_1, K_2) *tale che* $E_{K_1}(E_{K_2}(x)) = x$ *per ogni* x.

Il numero delle **semi-weak key** è di sei coppie ed il loro effetto è che la cifratura con una chiave della coppia seguita dalla cifratura con l'altra chiave della coppia fornisce il testo in chiaro. Queste chiavi hanno la particolarità che invece di generare 16 sottochiavi diverse ne generano solo 2, ognuna utilizzata 8 volte nell'algoritmo.

Quindi il DES ha 4 weak key e 6 coppie di semi-weak key. Le quattro weak key sono listate nella Tabella 8 con i corrispondenti 28 bit di C_0 e D_0 dove $\{0\}^j$ rappresenta j ripetizioni di 0 bit e rispettivamente $\{1\}^j$ rappresenta j ripetizioni di 1 bit. Cosi C_0 e D_0 sono tutti vettori di tutti zero o di tutti uno e le rotazioni non hanno effetti.

C_0	D_0	Primi elementi	Secondi elementi	C_0	D_0
$\{01\}^{14}$	$\{01\}^{14}$	01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01	$\{10\}^{14}$	$\{10\}^{14}$
$\{01\}^{14}$	$\{10\}^{14}$	1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E	$\{10\}^{14}$	$\{01\}^{14}$
$\{01\}^{14}$	$\{0\}^{28}$	01E0 01E0 01F1 01F1	E001 E001 F101 F101	$\{10\}^{14}$	$\{0\}^{28}$
$\{01\}^{14}$	$\{1\}^{28}$	1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E	$\{10\}^{14}$	$\{1\}^{28}$
$\{0\}^{28}$	$\{01\}^{14}$	010F 011F 010E 010E	1F01 1F01 0E01 0E01	$\{0\}^{28}$	$\{10\}^{14}$
$\{1\}^{28}$	$\{01\}^{14}$	E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1	$\{1\}^{28}$	$\{10\}^{14}$

Tabella 9: Le sei coppie Weak Key.

Le sei coppie di DES semi-weak key sono listate nella Tabella 9. Tenendo presente la Definizione 2 e guardando la tabella possiamo dire che queste chiavi godono di una particolare proprietà:

i bit delle sottochiavi K_1 fino ad arrivare a K_{16} della prima coppia sono uguali ai bit di K_{16} fino ad arrivare a K_1 della seconda coppia. Cioè se osserviamo la sottochiave C_0 del primo elemento della coppia notiamo che è uguale al complemento di C_0 del secondo elemento della coppia. Analogamente vale per D_0.

Oltre alle weak key e alle semi-weak key, ci sono 48 chiavi particolari che producono questa volta solo quatto sottochiavi, ognuna usata solo quattro volte nell'algoritmo. Comunque fino ad ora nessuno è riuscito a trarre vantaggio dall'esistenza di queste ultime 48 chiavi [18] e non è quindi ritenuto necessario evitare di usarle. Anche se l'esistenza di queste 64 chiavi (weak key, semi-weak key e le rimanenti 48) minasse la sicurezza del DES, controllare se una chiave è tra queste è molto semplice.

4.2 Proprietà di complementazione

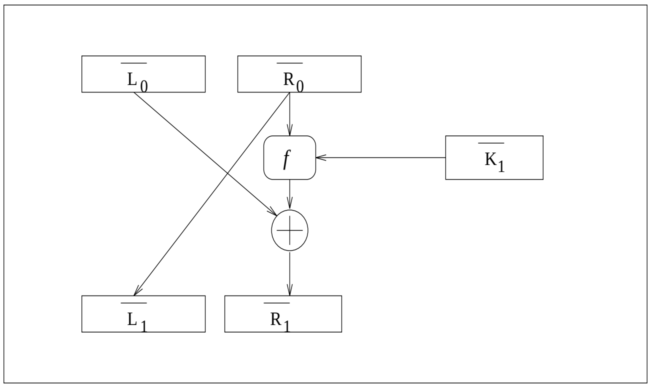
Se al posto della chiave K, nel DES, si considera la chiave $\overline{K}$, allora si può osservare il seguente fenomeno: dati un qualsiasi messaggio M e una chiave K, se

$$C = DES_K(M)$$

allora

$$\overline{C} = DES_{\overline{K}}(\overline{M}).$$

Diamo ora una prova che vale questa implicazione. Considerando la Figura 3 (struttura del DES) avremo che se il testo in chiaro M è complementato, allora l'input alla prima iterazione è dato da $\overline{L_0}$ e $\overline{R_0}$. Vediamo ora cosa accade nella prima iterazione:



come si evince dalla figura, avremo che l'output é dato da $\overline{L_1}$ e $\overline{R_1}$, in accordo con le formule:

$$\overline{L_1} = \overline{R_0}$$

$$\overline{R_1} = f(\overline{R_0}, \overline{K_1}) \oplus \overline{L_0}$$

Questo accade perchè la parte sinistra dell'output corrisponde alla parte destra dell'input che è complementata rispetto al caso in cui facciamo $DES_K(M)$, mentre per quanto riguarda la parte destra, notando che

$$f(\overline{R_0}, \overline{K_1}) = f(R_0, K_1)$$

avremo che essa é complementata, rispetto al caso in cui facciamo $DES_K(M)$, perché effettuiamo lo XOR con l'input $\overline{L_0}$ invece che con L_0.

Infatti : $f(\overline{R_0}, \overline{K_1}) \oplus \overline{L_0} = f(R_0, K_1) \oplus \overline{L_0} = \overline{f(R_0, K_1) \oplus L_0} = \overline{R_1}$

in quanto $f(R_0, K_1) \oplus L_0 = R_1$

Notiamo inoltre che, nella prima iterazione utilizziamo il valore $\overline{K_1}$, perchè la schedulazione delle chiavi, partendo da una chiave $\overline{K}$, produce 16 sottochiavi $\overline{K_i}$. Vediamo ora perchè vale l'uguaglianza

$$f(\overline{R_0}, \overline{K_1}) = f(R_0, K_1)$$

Se ci rifacciamo alla Figura 6 (schema della funzione f), avremo che ad un certo punto, viene calcolato $E(\overline{R_0}) \oplus \overline{K_1}$, che risulta uguale a $E(R_0) \oplus K_1$, per cui il valore della f è lo stesso in entrambi i casi. Quanto detto per la prima iterazione vale anche per le altre, per cui dopo la sedicesima iterazione, scambiando parte destra e sinistra, ed applicando la permutazione inversa, quello che otteniamo è $\overline{C}$, a differenza del caso in cui facciamo $DES_K(M)$ dove l'output dove l'output è C.

La proprietà di complementazione non è di aiuto ad un crittoanalista se per la ricerca esaustiva della chiave sceglie un attacco known-plaintext, mentre se sceglie di effettuare un attacco chosen plaintext esaminerà la metà delle chiavi cioè 2^{55} e questo consegue proprio dalla proprietà di complementazione. Ci si può chiedere se questa sia una debolezza o meno dal momento che molti messaggi (la maggior parte) non hanno blocchi che sono l'uno il complemento dell'altro.

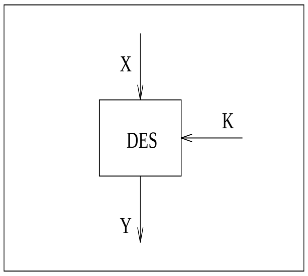
5 Possibili attacchi al DES

5.1 Attacchi non specifici per il DES

Non esistono pubblicazioni riguardo ad attacchi al DES portati con successo, ma esistono attacchi che funzionano per opportune varianti del DES (vedi *crittoanalisi differenziale*) e che danno un'idea della sicurezza del sistema. D'altra parte non è stata nemmeno dimostrata la non attacabilità del DES.

Un primo semplice attacco è quello di forza bruta: vediamo fin dove possiamo giungere con questa strategia. Questo tipo di attacco è universale, nel senso che va bene per qualsiasi sistema.

Osserviamo innanzitutto che, a dispetto delle 2^{56} chiavi possibili, in media bastano 2^{55} tentativi. Se volessimo analizzare il DES con modalità operative come l'OFB, l'ECB o altre (vedi sezione *Modalità Operative*), l'analisi risulterebbe enormemente complessa. Scegliamo perciò il caso più semplice.



Il DES codifica 64 bit con chiave a 56 bit, e produce un output di 64 bit. Portiamo un attacco di tipo *known plaintext*. Un attacco esaustivo richiederebbe di provare tutte e 2^{56} chiavi con:

- Complessità in tempo dell'ordine di 2^{56} (2^{55} in media).

- Complessità in spazio costante.

Per far fronte alla dimensione del tempo sono state proposte macchine estremamente veloci. Queste macchine dovrebbero avere tanti processori paralleli ognuno dei quali testa una chiave. Ci si è posti il problema di determinare:

- Quanto grande deve essere tale macchina;
- Quanto tempo è necessario per trovare una chiave;
- Quanto costa una macchina del genere.

Con macchine di questo tipo il tempo è enormemente ridotto, mentre paghiamo il guadagno in termini di spazio oltre che di soldi.

Cambiamo adesso strategia di attacco. Portiamo un attacco di tipo *Chosen plaintext*. Dopo un certo numero di tentativi si vorrà essere capace di trovare la chiave.

Il metodo più immediato è quello di provare (in monoprocessore) tutte le chiavi. Le risorse richieste sono:

- Tempo 2^{56}.
- Spazio costante.

Questa via è però impraticabile. Per far fronte ad un valore di tempo così grande possiamo immaginare un algoritmo che sacrifica dello spazio per guadagnare in tempo.

L'algoritmo lavorerebbe nel seguente modo. Per un dato testo in chiaro x precomputa $y_K = DES_K(x)$ per ogni K tra le 2^{56} chiavi possibili. Manteniamo una tabella di coppie (y_K, K) ordinate per testo cifrato.

Dopo aver ottenuto y, il testo cifrato di x con chiave sconosciuta, cerchiamo y nella tabella e teniamo memoria della chiave K associata. Questa è la chiave cercata.

Se la ricerca è fatta in modo binario il tempo è logaritmico; se la tabella è mantenuta come una tabella hash il tempo di ricerca è addirittura costante.

In definitiva, pagato il tempo di pre-computazione (dell'ordine di 2^{56}, già troppo!), la ricerca della chiave prende un tempo costante mentre il mantenimento della tabella è all'ordine di 2^{56}.

Anche questo algoritmo non è implementabile perchè spazi di queste dimensioni sono troppo costosi oltre che tecnicamente difficili da costruire.

Se questi algoritmi gravano troppo o solo sul tempo o solo sullo spazio diventano non implementabili. Cerchiamo perciò di fare un trade off tra i due algoritmi proposti.

L'attacco che adesso analizzeremo richiede un tempo ridotto rispetto alla ricerca esaustiva e uno spazio ridotto rispetto al mantenimento della tabella delle coppie (y_K, K).

Il nuovo algoritmo richiede una funzione di riduzione $F : \{0,1\}^{64} \to \{0,1\}^{56}$ che riduce una stringa binaria di 64 bit ad una a 56 bit (F potrebbe semplicemente eliminare gli ultimi 8 bit).

Sia x un fissato testo in chiaro di 64 bit e $X(1,0)$ una stringa arbitraria di 56 bit. Interpretiamo $X(1,0)$ come una chiave per codificare x:

$$y = DES_{X(1,0)}(x).$$

A questo punto y è una stringa a 64 bit, e con l'ausilio di F otteniamo $X(1,1)$ a 56 bit:

$$X(1,1) = F(y) = F(DES_{X(1,0)}(x)).$$

Potremo ancora interpretare $X(1,1)$ come una chiave per codificare x:

$$X(1,2) = F(DES_{X(1,1)}(x)).$$

Ripetiamo questo procedimento per t volte ottenendo:

$$X(1,0) \rightarrow X(1,1) \rightarrow X(1,2) \rightarrow \cdots \rightarrow X(1,t).$$

Abbiamo ottenuto una sequenza di stringhe di 56 bit, logicamente correlate, che possono essere interpretate sia come messaggi codificati (e poi ridotti) e sia come chiavi di cifratura.

È chiaro che se invece di scegliere la sola stringa $X(1,0)$, scegliamo m stringhe diverse $X(1,0), X(2,0), \cdots X(m,0)$, tutte di 56 bit, possiamo applicare il procedimento illustrato per $X(1,0)$ a tutte queste stringhe ottenendo la Tabella 10. Ogni elemento di questa tabella (eccezion fatta per la prima colonna o colonna 0) è computato secondo la seguente regola:

$$X(i,j) = F(DES_{X(i,j-1)}(x))$$

per $1 \leq i \leq m$ e $1 \leq j \leq t$.

0		1		2		....		t
$X(1,0)$	$\rightarrow$	$X(1,1)$	$\rightarrow$	$X(1,2)$	$\rightarrow$	....	$\rightarrow$	$X(1,t)$
$X(2,0)$	$\rightarrow$	$X(2,1)$	$\rightarrow$	$X(2,2)$	$\rightarrow$	....	$\rightarrow$	$X(2,t)$
$X(m,0)$	$\rightarrow$	$X(m,1)$	$\rightarrow$	$X(m,2)$	$\rightarrow$	....	$\rightarrow$	$X(m,t)$

Tabella 10: Valori $X(i,j)$.

A questo punto non è chiaro né come utilizzare la Tabella 10, né quali sono i valori di m e t ottimali. Rispondiamo ora alla prima domanda.

Dal testo in chiaro x otteniamo il messaggio y cifrato con la chiave sconosciuta K. Stiamo cercando di scoprire K. Cercheremo perciò y nell'ultima colonna; ma y è a 64 bit mentre gli elementi della tabella sono a 56 bit. Per questo motivo non verrà cercato y ma $F(y)$, la riduzione di y.

Supponiamo che $F(y)$ è presente nell'ultima colonna, e sia $X(i,t) = F(y)$. $X(i,t)$ è la riduzione di $DES_{X(i,t-1)}(x)$, per cui $X(i,t-1)$ è candidato a essere la chiave cercata. Il dubbio che $X(i,t-1)$ sia la chiave giusta è legittimo, e dovuto al fatto che $X(i,t) = F(y)$ è la riduzione di y, e non y stesso. Si potrebbe cioè avere che $F(y) = X(i,t)$ e $F(DES_{X(i,t-1)}(x)) = X(i,t)$ ma $y \neq DES_{X(i,t-1)}(x)$.

Non ci resta perciò che cifrare x secondo $X(i,t-1)$ e verificare se la cifratura è proprio y. In caso affermativo ci fermiamo concludendo con successo. In caso negativo andiamo avanti.

Sia nel caso in cui $F(y)$ non è presente nell'ultima colonna, sia quando $F(y)$ c'è ma la presunta chiave si rivela sbagliata, si procede come segue.

Adesso la colonna $t-1$ prenderà il ruolo della colonna t, e la colonna $t-2$ (quella che precede $t-1$) prenderà il ruolo di $t-1$. Questa modalità puó essere ripetuta fino ad esaurire tutta la tabella (cioè fino a raggiungere la colonna 1).

Anche se ci convinciamo che questo algoritmo funziona, non è chiaro dove è la sua convenienza rispetto a quelli già illustrati. Anzi, sembrerebbe addirittura il peggiore tra quelli esposti. Ebbene, l'intero procedimento appena illustrato puó essere implementato mantenendo soltanto la colonna 0 e la colonna t. Vediamo come.

È evidente che conviene mantenere ordinata la colonna t (meglio se utilizziamo una tabella hash). Adesso si puó cercare $F(y)$ efficientemente nell'ultima colonna. Se tale valore è presente e $F(y) = X(i,t)$ la presunta chiave è $X(i,t-1)$ che non ho piú. Ma dal valore $X(i,0)$ potremo ricalcolare il valore desiderato in $t-1$ applicazioni ripetute del DES.

Se $X(i,t-1)$ non è la chiave giusta procederemo cercando nella colonna $t-1$, che però non manteniamo in memoria. Il ricalcolo di tutta la colonna $t-1$ è troppo costoso, per cui converrà seguire un'altra strada.

Se $F(y)$ è nella colonna $t-1$, vuol dire che $F(DES_{F(y)}(x))$ è nella colonna t. Potremo perciò utilizzare ancora la colonna t cercando in essa non piú $F(y)$, ma la riduzione della cifratura di x secondo la chiave $F(y)$. Se la ricerca termina con successo (diciamo che $X(j,t-1)$ è l'elemento cercato) il candidato ad essere la chiave è $X(j,t-2)$, che non abbiamo piú, ma che possiamo ricalcolare da $X(j,0)$ con l'applicazione ripetuta di DES.

Se anche sulla colonna $t-1$ il processo di ricerca della chiave fallisce, tentiamo con la colonna $t-2$ che nemmeno abbiamo. Ma se $F(y)$ è in questa colonna vuol dire che

$$F(DES_z(x)) \in colonna\ t$$

dove

$$z = F(DES_{F(y)}(x)).$$

Questo perché se $F(y)$ si trova nella colonna $t-2$ allora $z = F(DES_{F(y)}(x))$ si trova nella colonna $t-1$ e $F(DES_z(x))$ si trova nella colonna t.

Quindi, ancora una volta puó essere utilizzata la sola colonna t. E questo per tutte le fasi della computazione.

In sintesi l'algoritmo (senza precomputazione) è il seguente:

1. calcola $y_1 = F(y)$
2. **for** $j = 1$ **to** t **do**
3.   **if** $y_j = X(i,t)$ per qualche i **then**
4.   **for** $h = 1$ **to** $t-j$ **do**
5. $X(i,h) = F(DES_{X(i,h-1)}(x))$
6.   **if** $y = DES_{X(i,t-j)}(x)$ **then STOP** (La chiave è: $K = X(i,t-j)$)
7.   calcola $y_{j+1} = F(DES_{y_j}(x))$

In definitiva, pagato il prezzo iniziale di precomputazione per il calcolo di tutti i valori $X(i,s)$, basterà mantenere le sole colonne 0 e t guadagnando perciò in spazio, ma pagando in termini di tempo per la produzione delle presunte chiavi e degli elementi da cercare nell'ultima colonna.

Purtroppo puó capitare che, pur trovando diversi $F(y)$ nella Tabella 10, la chiave associata non è quella giusta, e questo perché per ogni u nel codominio di F ci sono 2^8 valori y nel dominio tale che $F(y) = u$.

Possiamo a questo punto costruire più tabelle del tipo Tabella 10, ripetendo per ognuna di esse tutto l'algoritmo visto in precedenza. Tuttavia, anche utilizzando più tabelle la ricerca puó fallire.

La filosofia che sta alla base di questo algoritmo è la seguente: la tabella completa puó essere molto grande, per cui converrà memorizzare non tutti gli elementi ma solo alcuni. Daltronde questo guadagno in termini di spazio verrà pagato in termini di tempo.

Ci restano ancora da definire i valori di m, t, e il numero di tabelle, tutti valori vitali per l'efficacia dell'algoritmo. Senza entrare nei dettagli diciamo che l'analisi probabilistica mostra che se $m \cdot t^2 \approx N = 2^{56}$ allora la probabilità che la chiave si trovi nella tabella è circa[3] $0,8 \cdot m \cdot t/N$. Si consiglia di scegliere i seguenti valori:

— $m \approx N^{1/3}$;

— $t \approx N^{1/3}$;

— $\#tabelle \approx N^{1/3}$, ognuna realizzata con funzione F diversa.

Procediamo con l'analisi delle complessità con questi valori.

Lo spazio complessivo è:

$$(\#tabelle) \cdot (\#colonne\ per\ tabella) \cdot (\#elementi\ per\ colonna) \cdot$$

$$\cdot (\#bit\ per\ elemento) = N^{1/3} \cdot 2 \cdot N^{1/3} \cdot 56 = 112 \cdot N^{2/3}.$$

Il tempo di precomputazione è:

$$(\#tabelle) \cdot (\#colonne\ per\ tabella) \cdot (\#elementi\ per\ colonna) =$$

$$= N^{1/3} \cdot N^{1/3} \cdot N^{1/3} = N.$$

Per l'analisi del tempo effettivo di calcolo (al netto della precomputazione) procediamo nel seguente modo: il caso pessimo consiste nello scandire tutte le tabelle senza trovare la chiave. Supponiamo che il Passo 3. dell'algoritmo fallisce sempre (test sempre negativo). La ricerca puó essere efficientemente implementata con le tabelle hash, per cui il tempo complessivo di ricerca è:

$$(\#tabelle) \cdot (\#colonne) = N^{2/3}.$$

Un'analisi piú dettagliata (che non facciamo) puó dimostrare che anche se vengono eseguiti i passi da 4. a 6. (il test al passo 3. non fallisce sempre, ma fallisce sempre il test al passo 6.) il tempo di computazione cresce solo di un fattore costante.

[3] Il valore 0.8 dipende dal fatto che i valori della tavola possono non essere tutti diversi.

5.2 Cracking DES

Il cracking DES consiste nell'andare a cercare la chiave segreta che ha permesso di cifrare l'informazione trasmessa usando il DES.

Questo può essere fatto molto semplicemente provando tutte le possibili chiavi fino a che non se ne trova una, tedioso processo che viene chiamato ricerca di forza bruta. Osserviamo che se fosse facile rompere il DES la sicurezza delle trasmissioni cifrate tramite DES sarebbe minata, quindi molti rappresentanti del Governo U.S.A. facevano affermazioni circa quanto tempo ci si impiega per rompere il DES, essi affermavano che il Cracking DES è molto costoso e consuma molto tempo.

Sempre a riguardo del DES Cracking qualcuno ha creduto che se avesse più risorse a disposizione, riuscirebbe a testare più chiavi allo stesso tempo. Questa idea è semplicemente irrealizzabile in quanto la decifrazione di forza bruta prende comunque molto tempo.

Osserviamo che un computer ordinario mal si adatta per essere usato come un DES cracker. In primo luogo, il DES è tale che esso è molto lento per quanto riguarda il software, ma veloce in hardware. In secondo luogo i computer correnti fanno veramente poco in parallelo; non si sa precisamente quali istruzioni saranno eseguite e si devono permettere tutte le combinazioni.

Il modo giusto di rompere il DES è con hardware con scopi speciali. Un chip appositamente disegnato anche se con clock lento può facilmente produrre l'output anche con un computer General-Purpouse più veloce. Inoltre si possono prendere molti di questi chip su una singola piastra, piuttosto che uno o due di un tipico computer con una piastra madre.

5.3 RSA Challenges

La discussione sulla possibile debolezza del DES dovuta alla dimensione troppo ridotta della chiave sono continuate praticamente senza sosta nella comunità crittologica, ma senza approdare a qualcosa di concreto.

In effetti il DES, però, ha passato già ben tre revisioni periodiche, una ogni cinque anni, che ne hanno riconfermato la validità come standard federale.

Agli inizi del 1997 l'RSA, decise di verificare se i 56 bit del vecchio DES erano ancora sufficienti contro l'attacco a forza bruta.

Il 28 gennaio 1997, la RSA lanciò una pubblica sfida, estesa a chiunque, avente come oggetto la decifrazione di alcuni brevi testi cifrati col DES e con l'RC5 secondo chiavi di varie lunghezze.

I messaggi da decifrare consistevano in una prima parte nota, in modo che al decifratore era possibile accorgersi di aver trovato la giusta chiave di decifrazione, ed in una seconda parte segreta che costituiva la prova dell'avvenuta decifrazione.

I messaggi da decifrare furono pubblicati alle 9:00 a.m. del 28 gennaio 1997 sul Web della RSA, e subito partì la gara. In palio, vi erano i seguenti premi in denaro:

- $ 1000 per la forzatura del cifrario RC5 con chiave di 40 bit;
- $ 5000 per la forzatura del cifrario RC5 con chiave a 48 bit;
- $ 10000 per la forzatura del cifrario RC5 con chiave maggiore di 48 bit;
- $ 10000 per la forzatura del DES con chiave fissa di 56 bit;

Alle 12:30 a.m. dello stesso 28 gennaio, dopo solo tre ore e mezza dall'inizio della sfida, l'RC5 a 40 bit venne forzato da uno studente con una workstation.

Che 40 bit fossero davvero troppo pochi lo sospettavano tutti, ma che ci volesse davvero così poco per forzarli lo immaginavano in pochi.

Il passo successivo, la forzatura dell'RC5 a 48 bit, arrivò poco dopo: per la precisione il 10 febbraio alle 10:00 a.m. a sole 313 ore di distanza dalla pubblicazione dei messaggi cifrati da parte dell'RSA. Un intero "campus" universitario si era mobilitato per questa prova.

5.3.1 Attacco distribuito

Lo spazio delle chiavi del DES è formato da 2^{56} elementi, che in notazione decimale equivale a circa 72 milioni di miliardi di combinazioni distinte, per cui un ipotetico computer a 500 MHz che potesse sondare una chiave ad ogni ciclo di clock ci impiegherebbe 144 milioni di secondi ad esaminarle tutte; supponendo che sia sufficiente esaminare solo la metà delle chiavi ci vorrebbe comunque ben 834 giorni pari a 2 anni e 3 mesi di lavoro continuativo. Da questo possiamo concludere che 56 bit sono tanti, per cui non basta un supercomputer e nemmeno i calcolatori di una intera università.

Così il 13 marzo 1997 Rocke Verser, un programmatore di Loveland nel Colorado, fece partire una gigantesca iniziativa, chiamata **DESCHALL**, con la quale mirava a demolire il DES utilizzando la potenza di calcolo, altrimenti inutilizzata, di migliaia di computer sparsi in tutto il mondo.

Il progetto prevedeva la partecipazione cooperativa, volontaria e disinteressata, di migliaia di partecipanti, singoli utenti od organizzazioni. Costoro coordinarono i propri sforzi servendosi della rete Internet, essi si suddivisero lo spazio delle chiavi esaminandone ciascuno un pezzetto a seconda delle proprie disponibilità di hardware e di tempo macchina.

Verser scrisse un apposito client di ricerca che venne distribuito gratuitamente a tutti i partecipanti. Di giorno in giorno aumentarono i partecipanti che si dedicavano a questa impresa.

La sera del 17 giugno un programmatore Michael K. Sanders con un computer con le seguenti caratteristiche:

- Cpu Pentium 90 MHz; Ram 16 megabytes; Speed (key/second) 250;

riuscì a vincere la gara. Bastò cercare solo il 24,6% dello spazio delle chiavi. La ricerca coinvolse complessivamente circa 70000 computer e procedette ad un ritmo costantemente crescente: la velocità media negli ultimi giorni era di circa sette miliardi di chiavi esaminate al secondo, ossia lo 0,7% dell'intero spazio delle chiavi al giorno; se si fosse mantenuto questo ritmo già dall'inizio allora l'intera ricerca sarebbe durata solo 32 giorni, anzichè 39 giorni. Il messaggio segreto era: *Strong cryptography makes the world a safer place* (*una crittografia robusta rende il mondo più sicuro.*)

I diecimila dollari del premio furono divisi in due quote: il 60% a Verser ideatore del progetto **DESCHALL** ed il 40% a Michael Sanders che effettivamente trovò la chiave giusta.

Ora questo costituisce un attacco di tipo distribuito che portò i responsabili della FBI e della NSA ad affermare che il DES era praticamente indecifrabile, dato che per riuscire a decifrarlo occorreva lo sforzo congiunto di migliaia di computer per settimane di tempo macchina.

5.3.2 Attacco tramite chip

Per decenni si pensava di costruire un computer specializzato in grado di forzare il DES mediante ricerca esaustiva delle chiavi.

Nel 77 *Diffie* ed *Hellman*, due importanti crittologi americani, inventori tra l'altro dei sistemi di crittografia a chiave pubblica, cominciarono a chiedersi pubblicamente se 56 bit di chiave non fossero davvero pochi; e giunsero alla conclusione che il Governo di una potenza industriale avrebbe potuto tranquillamente investire una ventina di milioni di dollari necessari, a realizzare una macchina superparallela per forzare il DES.

Essi suggerirono che si sarebbe potuto costruire un chip a tecnologia VLSI che avrebbe potuto testare 10^6 chiavi al secondo [11]. Una macchina che utilizzava 10^6 chip di questo tipo avrebbe rotto il sistema in un giorno. Il costo di tale macchina era di $20.000.000.

Al loro primo studio ne seguirono altri di persone diverse, che comunque concordavano che la realizzazione di una macchina del genere fosse realmente possibile, anzi, qualcuno sosteneva che essa fosse stata realmente costruita e venisse utilizzata correntemente dalla NSA e forse da qualche altra organizzazione.

Fra questi ci fu *Michael Wiener*, che alla Rump Session di CRYPTO '93 propose il progetto di una macchina composta da vari chip collegati tra loro (pipelined) che operassero in parallelo. Ogni chip costava $10.50, per cui con una macchina costruita con 5760 chip (e per un costo di 100.000 dollari) si sarebbe rotto il sistema all'incirca in un giorno e mezzo. Con 10 di tali macchine e quindi con 1.000.000 di dollari si sarebbe rotto il DES in tre ore e mezzo [28].

Altre macchine (con costi simili) sono state progettate e costruite, ma nonostante ciò il problema non è stato risolto in quanto bisognava "cablare" tale macchina sul canale di comunicazione, avendo, quindi, tutta una serie di problemi di natura diversa.

Una di queste macchine fu progettata dall'EFF, che per mostrare che il DES non era più sicuro condusse una ricerca che mostrava che il DES poteva essere rotto con un costo basso.

L'EFF con la collaborazione di alcuni valenti crittologi (sotto la guida di Paul Kocher della Cryptografy Research) e di un'azienda produttrice di semiconduttori (la *Advanced Wireless Technologies*), progettò e realizzò un chip ASIC in tecnologia FPGA ("field programmable gate array", che è un chip che può essere riprogrammato dopo che è stato impiegato in una varietà di circuiti diversi) che mirava alla forzatura del DES mediante ricerca esaustiva delle chiavi. In particolare la Advanced Wireless Technologies oltre alla specifica del chip sviluppò anche il layout della board e l'assemblaggio del sistema.

Il chip, denominato Deep Crack, conteneva al suo interno 24 unità di ricerca.

Una unità di ricerca è un piccolo pezzo di hardware che prende una chiave e due blocchi di 64 bit di testo cifrato, decifrando un blocco di testo cifrato con la chiave, e controllando se il blocco risultante di testo in chiaro è significativo.

Se non lo è essa aggiunge 1 alla chiave e ripete, effettuando la ricerca attraverso lo spazio delle chiavi.

Se la prima decryption produce un risultato significativo, la stessa chiave è usata per decifrare il secondo blocco di testo cifrato. Se entrambi sono significativi, l'unità di ricerca si arresta, e, informa il software che è stata trovata una chiave significativa. Se la seconda decryption del blocco non è significativa, l'unità di ricerca aggiunge 1 alla chiave e prosegue ricercando nello spazio delle chiavi.

Come si evince, ogni unità di ricerca decifra 8 byte per volta e compie analisi statistiche sul risultato per capire se esso può costituire una soluzione oppure no.

Nel momento in cui un'unità di ricerca si ferma dopo aver trovato un risultato significativo il software sull'host computer dovrebbe esaminare il risultato e determinare se è la reale decifrazione.

Ogni risultato di decifrazione contiene 8 byte. L'unità di ricerca analizza ogni byte del risultato. Un tale byte può avere uno qualsiasi dei 256 valori.

L'unità di ricerca si basa su una tabella che definisce quali byte dei 256 valori sono significativi e quali sono non significativi. Per esempio, se il testo in chiaro è numerico, il software imposta la tabella cosicchè i 10 digit (0-9) sono significativi e tutti gli altri potenziali valori sono non significativi.

Le unità di ricerca presenti sul chip potevano controllare il blocco decifrato al fine di determinare se esso conteneva solo caratteri ASCII, o di verificare se byte specifici combaciavano con valori noti ed attesi.

Sessantaquattro chip Deep Crack erano assemblati su un'unica board che costituiva il mattone principale di una macchina DESCRACK parallela e modulare.

La seguente Figura mostra le performance del DES Cracker nel caso in cui si prendono 12 board e 2 chassis. Essa cercherà 92 miliardi di chiavi per secondo, coprendo la metà dello spazio delle chiavi in circa 4,5 giorni.

Device	How Many In Next Device	key/Second	Days/avg search
Search Unit	24	2.500.000	166.800
Chip	64	60.000.000	6.950
Board	12	3.840.000.000	109
Chassis	2	46.080.000.000	9,05
EFF DES Cracker		92.160.000.000	4,524

Tabella 11: Sommario delle performance del DES Cracker.

Per migliorare queste performance la EFF costruì un prototipo che conteneva una trentina di board ospitate in sei chassis recuperati da vecchi computer Sun, ma il sistema era facilmente estensibile, bastava aggiungere altre schede per aumentare linearmente la potenza di ricerca. In effetti l'EFF ha cominciato la sua ricerca sul DES CRACKING nel 1997.

Il progetto originario era di vedere se un Des Cracker potesse essere costruito al di fuori di una macchina che conteneva un gran numero di FPGA.

Un gran numero di macchine costruite al di fuori di quelle che utilizzavano i chip FPGA, esistevano sul mercato che potevano essere usate per simulare molti disegni di chip, prima che il chip venisse costruito. Una collezione di migliaia di chip FPGA relativamente inefficienti poteva essere messa insieme per simulare un chip efficiente, sebbene a $\frac{1}{10}$ o a $\frac{1}{100}$ della velocità alla quale un chip commerciale viaggerebbe.

La macchina risultante avrebbe richiesto meno di una settimana, in media, per determinare la chiave da un singolo campione di 8 byte servendosi di un attacco know-plaintext e know-ciphertext, inoltre avrebbe determinato la chiave da un campione di 16 byte di testo cifrato in al più lo stesso tempo, se le caratteristiche statistiche del testo in chiaro fossero conosciute o ricavabili.

Per esempio, se il testo in chiaro fosse un messaggio della posta elettronica, allora essa potrebbe trovare tutte le chiavi che producono un testo in chiaro che non contiene nè lettere nè numeri e punteggiatura.

Non vi era nulla di rivoluzionario nella Des Cracker, infatti usava idee all'ordine del giorno su come rompere il DES. Essa era stata costruita dalla EFF per determinare quanto era conveniente o costoso andare a costruire una macchina per decifrare i messaggi del DES.

Macchine molto simili erano state già costruite in precedenza con costi più elevati e maggiore lentezza.

Il sistema della EFF fu messo alla prova il 15 luglio 1998 quando la RSA diede il via ad una nuova sfida, ottenendo come risultato che in sole 56 ore il DES Cracker ricavò la chiave di cifratura, battendo ogni record.

Il costo reale del DES Cracker era stato di poco superiore ai $200000, di cui circa $130000 di spese per i chip e circa $80000 per il progetto e il suo sviluppo.

5.3.3 Ultima sfida RSA

L'ultima sfida "DES Challenge III", è stata pubblicata alle ore 9.00 a.m. del 18 gennaio 1999. In essa vi era il seguente testo cifrato: *92 2C 68 C4 7A EA DF F2* ricavato da un messaggio sconosciuto di testo in chiaro, che aveva un messaggio fisso e conosciuto come header.

La chiave segreta era generata in modo random e poi distribuita fra il software che generava la disputa così che nessuno, neanche gli amministratori della disputa, sapevano qual'era. La strutturazione del premio per questa challenge è stata la seguente:

- $ 10000 con l'impiego di massimo 24 ore;
- $ 5000 con l'impiego di massimo 48 ore;
- $ 1000 con un massimo di 56 ore;
- niente se si superano le 56 ore.

I vincoli di tempo hanno suggerito ai due vincitori delle dispute precedenti di collaborare, infatti per battere il record precedente di 56 ore, Distributed.Net, una coalizione mondiale di entusiasti di computer, ha lavorato con il "Deep Crack" dell' EFF, un supercomputer disegnato in modo speciale, ed una rete mondiale di quasi 100.000 PC su Internet riuscendo a vincere la sfida dell'RSA, DES Challenge III, riguardante la sicurezza dei dati, in un tempo record di **22 ore e 15 minuti**.

I PC coinvolti in questa sfida vanno da modesti PC del 1990 fino alle più potenti macchine multiprocessore.

Partendo come base dalla RSA Data Security Conference & EXPO, una grande conferenza sulla sicurezza dei dati e sulla crittografia, che è stata tenuta a S.Jose, California, il "Deep Crack" dell'EFF e i computer di Distributed.Net hanno testato 245 miliardi di chiavi per secondo finchè la chiave non è stata trovata.

Il messaggio in chiaro ricavato è: *"See you in Rome (second AES conference, March 22-23, 1999)"* (*"Vediamoci a Roma (seconda conferenza di AES, 22-23 Marzo 1999)"*)

Come parte della disputa, l'RSA ha dato un premio di 10.000 dollari ai vincitori. La meta della sfida DES Challenge III era non solo recuperare la chiave segreta usata nel messaggio di testo in chiaro cifrato col DES, ma far sì da essere più veloci dei vincitori precedenti nella serie.

5.3.4 Conclusioni

La lunghezza della chiave di cifratura DES, considerata più che sicura nel 1977, è ormai troppo insufficiente per resistere ad un attacco di forza bruta, come l'iniziativa DESCHALL ha dimostrato in maniera evidente.

Per avere una reale sicurezza dei sistemi crittografici non bastano più chiavi corte, occorrono chiavi lunghe almeno il doppio in modo da portare la lunghezza fuori dalla portata di supercomputer attuali e futuri.

L'ultima challenge mostra che stiamo giungendo rapidamente al momento in cui chiunque, con un PC standard, può potenzialmente costituire una vera minaccia per i sistemi che si basano su una sicurezza così vulnerabile.

Tuttavia quanto è cifrato con il DES non è compromesso del tutto, ma chi lo usa è invitato a riconsiderare i livelli di sicurezza richiesti e passare a sistemi ben più sicuri quali il TRIPLO DES proposto da Tuchmann e già standardizzato dall'ANSI e dall'ISO che con i suoi 112 bit offre una protezione più che adeguata.

Per ulteriori chiarimenti oppure approfondimenti consultare i seguenti siti http://www.eff.org/, http://www.rsa.com/, http://www.distributed.net/, http://www.replay.com/craking-des/ .

5.4 Crittoanalisi differenziale

Il miglior attacco al DES conosciuto è senza dubbio quello definito dal metodo della *Crittoanalisi differenziale*, introdotto da E. Biham e A. Shamir [4]. Esso, come vedremo, definisce un attacco di tipo Chosen Plaintext, che si presta con successo alla rottura del DES nell'ipotesi che il numero di iterazioni sia ridotto (il DES a 8 iterazioni puó essere rotto in un paio di minuti su un personal computer).

Tuttavia quando il numero di iterazioni sale a 16 o piú, attacchi di questo tipo hanno complessità uguale a quella che si avrebbe con un algoritmo di ricerca nello spazio delle chiavi.

Questa potrebbe essere una motivazione al fatto che siano state scelte proprio 16 iterazioni nel DES standard e non meno. Si dice infatti che i due ideatori della crittoanalisi differenziale — di cui uno è proprio lo Shamir noto per l'RSA — quando pubblicarono questo tipo di attacco si consultarono con uno degli ideatori del DES. Quest'ultimo dichiarò poi che il sistema fu progettato in modo da resistere a questo tipo di attacco.

Nella nostra analisi ci limiteremo quindi ad illustrare un attacco con sole tre iterazioni. In esso supporremo per semplicità che le permutazioni, iniziale IP e finale IP^{-1}, non vengano considerate in quanto esse non hanno effetto sulla crittoanalisi.

Si considererà inizialmente un attacco di tipo *Known Plaintext*, cioè si considereranno note delle coppie, testo in chiaro - testo cifrato, con chiave segreta K uguale per tutte.

Una versione piú semplice si ha quando è possibile scegliere quelli che sono i testi da cifrare in modo da velocizzare l'attacco. In mancanza di tale possibilità, bisogna aspettare che vengano scelti un maggior numero di testi affinché vengano scoperte certe proprietà (con un peggioramento di un fattore di circa 2^8 per la velocità dell'attacco).

Ciò su cui si basa la crittoanalisi differenziale è il confronto tra lo xor bit a bit di due testi in chiaro e lo xor bit a bit dei corrispondenti testi cifrati, o meglio delle differenze (da cui deriva anche il nome di questo tipo di attacco), al fine di scoprire la chiave utilizzata.

Nel testo che segue ci riferiremo a stringhe binarie; in particolare quelle marcate con un apice indicheranno lo xor bit a bit di altre due stringhe, ad esempio $x' = x \oplus x^*$.

Per illustrare la crittoanalisi differenziale ci serviremo delle seguenti definizioni:

Definizione 3 *Sia S_j una particolare S-box $(1 \leq j \leq 8)$, e consideriamo una coppia (ordinata) di stringhe binarie di lunghezza sei (B_j, B_j^*). Diremo:*

input xor *il valore $B_j \oplus B_j^*$ (a sei bit)*

output xor *il valore $S_j(B_j) \oplus S_j(B_j^*)$ (a quattro bit).*

Definizione 4 *Per ogni $B_j' \in Z_2^6$ definiamo l'insieme $\Delta(B_j')$ come l'insieme delle coppie ordinate (B_j, B_j^*) tali che*

$$B_j \oplus B_j^* = B_j'$$

cioè

$$\Delta(B_j') = \{(B_j, B_j^*) : B_j \oplus B_j^* = B_j', \text{con } B_j, B_j^* \in Z_2^6\}.$$

Tale insieme ha cardinalità 2^6, cioè uguale alla cardinalità di Z_2^6. 18 Infatti, considerando fissata la prima componente B_j, l'altra B_j^* (dove $B_j^* = B_j \oplus B_j'$) è definita in maniera univoca (dato che entrambi gli operandi sono a loro volta fissati). Quindi poiché le possibili stringhe

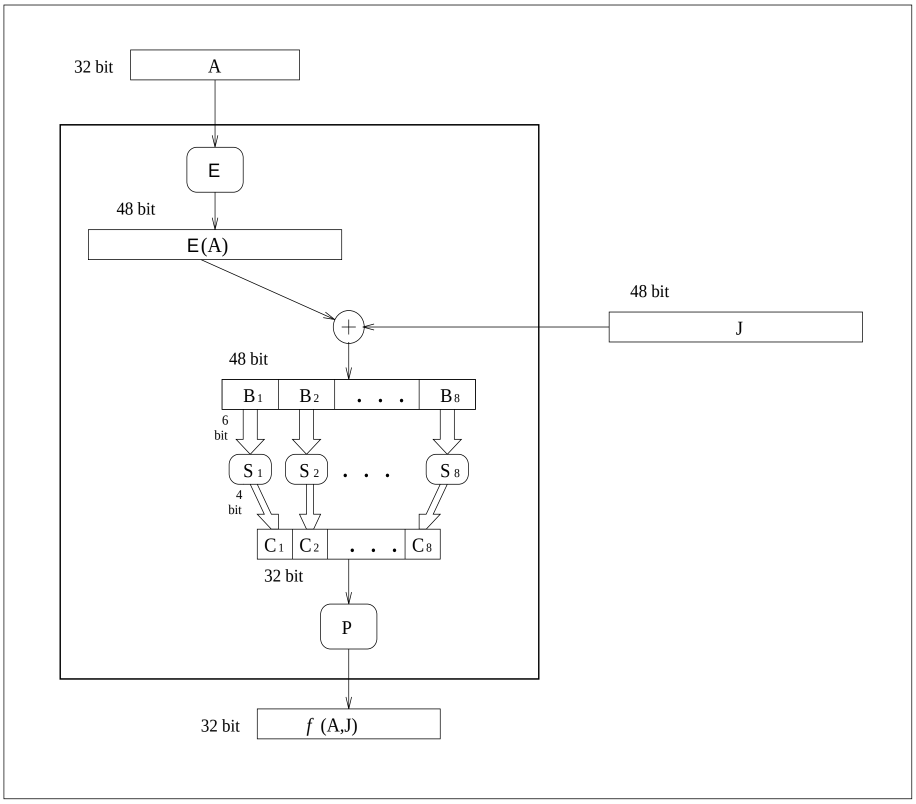


Figura 11: Schema della funzione f.

B_j a 6 bit sono 2^6, risulta che $|\Delta(B_j^{'})| = 2^6$. Consideriamo quindi due coppie (x_1, y_1) , (x_2, y_2) con x_1, x_2 testi in chiaro e y_1, y_2 i corrispondenti testi cifrati, cioè:

$$y_1 = DES_K(x_1) \quad , \quad y_2 = DES_K(x_2)$$

consideriamo per la nostra analisi le stringhe

$$x^{'} = x_1 \oplus x_1^*$$

e

$$y^{'} = y_1 \oplus y_1^*.$$

Testiamo quindi le differenze (xor) tra le coppie di testi considerando una singola iterazione dell'algoritmo DES (Figura 5).

Riportiamo quindi lo schema della funzione f in Figura 11.

L'attacco differenziale si concentra sulle S-box, che rappresentano l'unico punto non lineare dell'algoritmo.

Supponiamo di considerare una particolare S-box S_j, con $1 \leq j \leq 8$. Dati quindi due possibili input ad essa, definiti da B_j e B_j^* (di 6 bit ciascuno), avremo un input xor definito da: $B_j^{'} = B_j \oplus B_j^*$; considerando invece gli output all'S-box S_j, cioè $S_j(B_j)$ e $S_j(B_j^*)$ avremo un output xor definito da: $S_j(B_j) \oplus S_j(B_j^*)$.

A questo punto per ogni coppia in $\Delta(B_j^{'})$ possiamo calcolare l'output xor di S_j, $S_j(B_j) \oplus S_j(B_j^*)$, determinando cosí una distribuzione del numero di occorrenze per l'output xor[4], cioè possiamo determinare quante coppie in $\Delta(B_j^{'})$ hanno output xor uguale a y con $y \in Z_2^4$.

[4]la non uniformità di queste distribuzioni (una per ogni $\Delta(B_j^{'})$) costituirà la base dell'attacco.

Ad esempio consideriamo la prima S-box S_1 e supponiamo $B_1' = 110100$. Avremo che

$$\Delta(B_1') = \Delta(110100) = \{(000000, 110100), (000001, 110101) \cdots (111111, 001011)\}.$$

Per ogni coppia in $\Delta(110100)$ calcoliamo quindi l'output xor relativamente a S_1. Considerando per esempio la coppia $(000000, 110100)$, abbiamo:

$$S_1(000000) = E_{16} = 1110$$

e

$$S_1(110100) = 9_{16} = 1001$$

allora l'output xor per la coppia (000000,110100) è data da:

$$S_1(000000) \oplus S_1(110100) = 1110 \oplus 1001 = 0111.$$

Se tali operazioni vengono eseguite per tutte le 64 coppie in $\Delta(110100)$ si ottiene la seguente distribuzione del numero di occorrenze per l'output xor:

0000	0001	0010	0011	0100	0101	0110	0111
0	8	16	6	2	0	0	12

1000	1001	1010	1011	1100	1101	1110	1111
6	0	0	0	0	8	0	6

Tabella 12: Distribuzione del numero di occorrenze per l'output xor S_1.

Come si puó notare, per alcuni valori di output xor (0000, 0101, etc.) non vi sono coppie corrispondenti in $\Delta(110100)$. La cosa importante mostrata dalla Tabella 12 è che in realtà non tutte le possibili stringhe di 4 bit sono possibili come valori di output xor. Nell'esempio considerato solo la metà dei valori sono possibili (anche se questo rappresenta un caso estremo), tuttavia in generale solo il 75-80 % dei possibili output xor si possono effettivamente presentare.

Cerchiamo ora di sfruttare la non uniforme distribuzione degli output xor, ed in particolare il fatto che alcuni di tali valori sono possibili ed altri no. L'idea è quella di collezionare abbastanza statistiche in modo da conoscere poi i valori possibili degli output xor relativi ad una particolare stringa B_1', e cosí per ogni altra stringa.

Diamo ora alcune definizioni per meglio descrivere queste distribuzioni:

Definizione 5 *Al variare di j, $1 \leq j \leq 8$, e per stringhe binarie B_j' (di lunghezza sei) e C_j' (di lunghezza quattro) definiamo*

$$IN_j(B_j', C_j') = \{B_j \in Z_2^6 : S_j(B_j) \oplus S_j(B_j \oplus B_j') = C_j'\} \tag{1}$$

e

$$N_j(B_j', C_j') = |IN_j(B_j', C_j')|.$$

$IN_j(B_j', C_j')$ rappresenta, relativamente alla S-box j, l'insieme delle stringhe binarie di lunghezza sei tali che l'input xor di tale stringa con un'altra è B_j' e l'output xor da C_j'.

Fissati i valori B_j' e C_j', la definizione data sarà utile per cercare le coppie di stringhe (B_j, B_j^*) che godono della proprietà che il loro input xor è B_j', ed il loro output xor è C_j', senza dover analizzare necessariamente tutte le coppie possibili. Infatti dalla definizione, basterà effettuare la ricerca sul solo valore B_j, in quanto fissato un B_j tale che $S_j(B_j) \oplus S_j(B_j \oplus B_j') = C_j'$, la seconda componente della coppia che si vuole trovare è univocamente determinata da $B_j^* = B_j \oplus B_j'$.

Ad esempio, con i valori:

$$IN_1(110100, 0100) = \{010011, 100111\}$$

e

$$N_1(110100, 0100) = 2$$

si ottengono i valori di $IN_1(110100, x)$ riportati in Tabella 13 (con $x \in Z_2^4$ e $B_j = 110100$).

Ci chiediamo a questo punto quante diverse tabelle di questo tipo si possono avere. Consideriamo l'insieme definito da $IN(z, x)$, abbiamo già visto che per un fissato valore di z, al variare di x, otteniamo una particolare tabella. Sapendo che il valore z puó essere scelto in 2^6 modi diversi ($z \in Z_2^6$), si possono costruire 2^6 tabelle di questo tipo per ogni S-box S_j.

Ripetendo quindi il discorso fatto per tutte le 8 S-box si ottengono $64 \cdot 8 = 512$ tabelle totali (distribuzioni), che possono essere calcolate agevolmente anche con un PC.

Cerchiamo ora di trovare quelle che sono le possibili stringhe B che possono presentarsi come input alle S-box.

Ricordiamo che l'input alle S-box, per una qualunque iterazione i, è definito da $B = E \oplus J$, dove $E = \mathsf{E}(R_{i-1})$ rappresenta l'espansione della stringa R_{i-1} e $J = k$ (vedi Figura 6), quindi l'input xor a tutte le S-box puó essere calcolato come segue:

$$B \oplus B^* = (E \oplus J) \oplus (E^* \oplus J) = E \oplus E^*.$$

Si vede che l'input xor non dipende dai valori della chiave mentre l'output xor sí, come si puó notare dalla sua definizione:

$$S(E \oplus J) \oplus S(E^* \oplus J).$$

Scriviamo ora B, E e J come concatenazione di stringhe binarie di lunghezza sei cioè:

$$B = B_1 B_2 \cdots B_8$$
$$E = E_1 E_2 \cdots E_8$$
$$J = J_1 J_2 \cdots J_8$$

cosí come

$$B^* = B_1^* B_2^* \cdots B_8^*$$
$$E^* = E_1^* E_2^* \cdots E_8^*$$
$$J^* = J_1^* J_2^* \cdots J_8^*.$$

output xor	possibili input
0000	
0001	000011, 001111, 011110, 011111 101010, 101011, 110100, 111011
0010	000100, 000101, 001110, 010001 010010, 010100, 011010, 011011 100000, 100101, 010110, 101110 101111, 110000, 110001, 111010
0011	000001, 000010, 010101, 100001 110101, 110110
0100	010011, 100111
0101	
0110	
0111	000000, 001000, 001101, 010111 011000, 011101, 100011, 101001 101100, 110100, 111000, 111100
1000	001001, 001100, 011001, 101101 111000, 111101
1001	
1010	
1011	
1100	
1101	000110, 010000, 010110, 011100 100010, 100100, 101000, 110010
1110	
1111	000111, 001010, 010110, 011100 111110, 111111

Tabella 13: Valori di $IN_1(110100, x)$.

Supponiamo ad un certo punto dell'algoritmo di conoscere: E_j e E_j^* per qualche j (j è l'indice della S-box su cui stiamo lavorando), e supponiamo che sia noto anche il valore dell'output xor per S_j, cioè:

$$C_j' = S_j(B_j) \oplus S_j(B_j^*). \tag{2}$$

Si possono fare ora delle considerazioni circa la chiave. Innanzitutto si puó calcolare:

$$E_j \oplus E_j^* = B_j \oplus B_j^* = B_j'$$

da cui

$$B_j^* = B_j \oplus B_j'. \tag{3}$$

Quindi dalla definizione (1) e dalle relazioni definite dalla (2) e (3), si puó concludere che

$$B_j \in IN_j(B_j', C_j')$$

dove in realtà B_j è ottenuto come lo xor bit a bit dei valori E_j e J_j (vedi Figura 6), e quindi abbiamo:

$$E_j \oplus J_j \in IN_j(B_j', C_j'). \tag{4}$$

L'idea dell'algoritmo che andremo a definire è quella di sfruttare proprio quest'ultima relazione per arrivare a scoprire i bit della chiave $J = K_i$. In particolare:

- per ogni possibile coppia (B_j', C_j') si determina l'insieme corrispondente $IN_j(B_j', C_j')$, che definisce anche dei possibili valori di chiavi J_j come si puó notare dalla (4);
- poiché la chiave reale deve appartenere necessariamente ad ognuno degli insiemi cosí trovati, facendo l'intersezione di tali insiemi si otterrà l'unica chiave J_j (costituente il j-esimo blocco, di 6 bit, della chiave J);
- ripetendo quindi lo stesso ragionamento per tutte le 8 S-box si ottiene la chiave

$$J = J_1 \cdots J_8.$$

Introduciamo a questo punto la seguente:

Definizione 6 *Supponiamo che E_j e E_j^* siano due stringhe di 6 bit e C_j' una stringa di 4 bit. Definiamo:*

$$test_j(E_j, E_j^*, C_j') = \{B_j \oplus E_j : B_j \in IN_j(E_j', C_j')\}$$

dove $E_j' = E_j \oplus E_j^*$.

Questo insieme altro non è che un insieme di possibili chiavi J_j (dato che $B_j \oplus E_j = J_j$) per le quali, supponendo di partire dai valori E_j, E_j^*, si ottiene un output xor che vale C_j', cioè:

$$C_j' = S_j(E_j \oplus J_j) \oplus S_j(E_j^* \oplus J_j) = S_j(B_j) \oplus S_j(B_j^*).$$

Supponendo quindi che: E_j e E_j^* siano due input all'S-box S_j; l'output xor per S_j è C_j'; e conoscendo l'insieme $IN_j(E_j', C_j')$ da cui prendere i valori B_j, utili al calcolo delle chiavi J_j $(J_j = B_j \oplus E_j)$; allora si ha che:

$$J_j \in test_j(E_j, E_j^*, C_j')$$

cioè

$$\{J_j : S_j(E_j \oplus J_j) \oplus S_j(E_j^* \oplus J_j) = C_j'\} = test_j(E_j, E_j^*, C_j').$$

Quest'insieme definisce i possibili valori che J_j puó assumere relativamente alla tripla (E_j, E_j^*, C_j'). Al variare della tripla avremo un altro insieme di possibili valori per J_j.

Tuttavia poiché la chiave J usata è sempre la stessa, allora il j-esimo blocco J_j dovrà appartenere a tutti gli insiemi corrispondenti alle varie triple, e quindi alla loro intersezione.

È da notare che in ognuno degli insiemi sopra definiti solo un blocco è realmente utilizzato, e quindi dall' intersezione degli insiemi si otterrà sempre un solo valore.

Ad esempio siano:

$$E_1 = 000001, \; E_1^* = 110101, \; C' = 1101$$

allora $N_1(110100, 1101) = 8$ e i valori di B_j sono quelli che si possono leggere in Tabella 13, cioè $IN_1(110100, 1101) = \{000110, 010000 \cdots\}$.

Quindi risulta che:

$$\begin{aligned} test_1(000001, 110101, 1101) &= \{000111 = 000110 \oplus 000001, \\ &\quad 010001 = 010000 \oplus 000001, \\ &\quad \cdots\}. \end{aligned}$$

Con diverse triple possiamo determinare i valori di J_j. Poiché i possibili valori di J_j sono 64 utilizziamo allora un array di 64 contatori, e ogni volta che calcoliamo i valori nell'insieme $test_j$ incrementiamo i contatori corrispondenti a tali valori nell'array definito. Dopo t tentativi (t scelte di triple) il contatore che assume valore t (cioè che appartiene a tutti gli insiemi) ci indicherà il valore di J_j.

Se, ad esempio, il secondo contatore dell'array vale t allora il valore 000001 è il vero valore di J_j corrispondente all'intersezione dei vari insiemi $test_j$. Poiché J_j è solo un blocco della sottochiave, con 8 array riusciamo a calcolare l'intera sottochiave.

5.5 Un attacco al DES a 3 iterazioni

Vediamo ora come applicare concretamente il discorso fatto fino ad ora nell'ipotesi di usare il DES a 3 iterazioni.

Supponendo che L_0R_0 e $L_0^*R_0^*$ siano due testi in chiaro, e $L_0^{'}R_0^{'}$ il loro or esclusivo; applicando ad entrambe l'algoritmo DES con chiave K, definito dallo schema in Figura 12, otteniamo le stringhe cifrate L_3R_3 e $L_3^*R_3^*$.

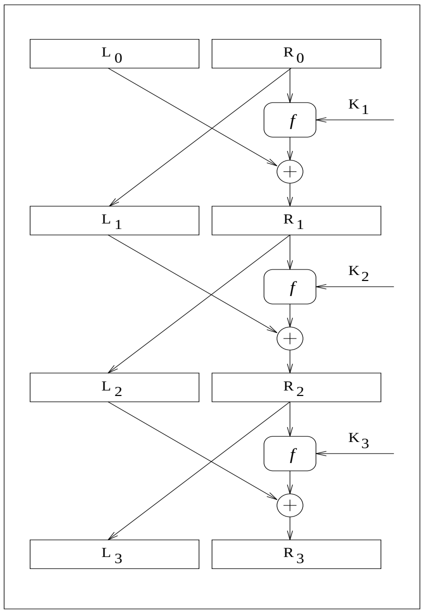


Figura 12: Schema del DES a tre iterazioni.

Sfruttando quindi le operazioni definite dal DES a tre iterazioni, posso esprimere R_3 e R_3^* come segue:

$$\begin{aligned} R_3 &= L_2 \oplus f(R_2, K_3) \\ &= R_1 \oplus f(R_2, K_3) \\ &= L_0 \oplus f(R_0, K_1) \oplus f(R_2, K_3) \end{aligned}$$

e analogamente
$$R_3^* = L_0^* \oplus f(R_0^*, K_1) \oplus f(R_2^*, K_3).$$

Da queste otteniamo l'espressione per $R_3^{'}$

$$R_3^{'} = R_3 \oplus R_3^* = L_0^{'} \oplus f(R_0, K_1) \oplus f(R_0^*, K_1) \oplus f(R_2, K_3) \oplus f(R_2^*, K_3).$$

Quindi scegliamo i testi in chiaro tali che $R_0 = R_0^*$ (ad esempio $R_0 = R_0^* = 0^{32}$). Ed ecco perché questo è definito come un chosen plaintext attack. Allora:

$$f(R_0, K_1) = f(R_0^*, K_1)$$

e quindi
$$R_3^{'} = L_0^{'} \oplus f(R_2, K_3) \oplus f(R_2^*, K_3)$$

o anche
$$R_3^{'} \oplus L_0^{'} = f(R_2, K_3) \oplus f(R_2^*, K_3).$$

Ricordiamo ora che
$$f(R_2, K_3) = P(C)$$

e
$$f(R_2^*, K_3) = P(C^*)$$

dove C e C^* rappresentano l'output delle 8 S-box alla terza iterazione (vedi Figura 6), e P è la permutazione definita nella funzione f, che è conosciuta pubblicamente. Quindi

$$\begin{aligned} R_3^{'} \oplus L_0^{'} &= P(C) \oplus P(C^*) \\ &= P(C \oplus C^*) \end{aligned}$$

da cui
$$P^{-1}(R_3^{'} \oplus L_0^{'}) = C \oplus C^* = C^{'}.$$

Ricordiamo ora che per calcolare il valore della stringa E, all'ultima iterazione del DES, si applicava alla stringa A (R_2 in questo caso) la funzione di espansione E, ottenendo $E = \mathsf{E}(A)$. Si effettuava poi lo xor bit a bit tra quest'ultimo valore e la stringa J (in questo caso K_3) ottenendo quindi l'input delle 8 S-box. Sapendo inoltre che nell'ultima iterazione il valore di L_3 è definito essere uguale al valore R_2, cioè $R_2 = L_3$ come pure $R_2^* = L_3^*$, e poiché i valori di L_3 e L_3^* sono noti, conosciamo di conseguenza anche i valori di R_2 e R_2^* da cui, per quanto detto sopra, anche i valori di E per la terza iterazione. Infatti: $E = \mathsf{E}(R_2) = \mathsf{E}(L_3)$, $E = \mathsf{E}(R_2^*) = \mathsf{E}(L_3^*)$ (Si ricordi che le permutazioni sono operatori lineari e sono noti).

A questo punto conosciamo E, E^* e $C^{'}$ per la terza iterazione e quindi siamo nelle ipotesi precedenti e possiamo quindi procedere come visto, costruendo gli insiemi $test_1 \cdots test_8$ e calcolando i valori $J_1 \cdots J_8$ della chiave K_3.

Dalla conoscenza di questa chiave K_3 possiamo poi ricavare 48 bit della chiave K. Basta infatti osservare l'algoritmo di schedulazione alla terza iterazione.

I rimanenti 8 bit della chiave vengono trovati mediante una ricerca esaustiva delle $2^8 = 256$ possibilità di completare la chiave K.

L'algoritmo che calcola gli insiemi $test_1, test_2, \cdots, test_8$ con input

$$L_0R_0, \quad L_0^*R_0^*, \quad L_3R_3 \quad \text{e} \quad L_3^*R_3^*$$

dove $R_0 = R_0^*$ è il seguente [27]:

1. calcola $C' = P^{-1}(R'_3 \oplus L'_0)$
2. calcola $E = \mathsf{E}(L_3)$ e $E^* = \mathsf{E}(L^*_3)$
3. **for** $j = 1$ **to** 8 **do**
4. calcola $test_j(E_j, E^*_j, C'_j)$

Questo algoritmo puó essere ripetuto per n volte (dove n è scelto opportunamente), al variare dell'input, ottenendo una sequenza di insiemi $(test_1^{(1)}, test_2^{(1)}, \cdots, test_8^{(1)})$, $\cdots\cdots$, $(test_1^{(n)}, test_2^{(n)}, \cdots, test_8^{(n)})$. L'intersezione $test_1^{(1)} \cap\ test_1^{(2)} \cap \cdots \cap\ test_1^{(n)}$ fornisce J_1.

Similmente possiamo ottenere $J_2, \cdots, J_8$. La sottochiave J è uguale a $J_1 J_2 \cdots J_8$.

Nella Tabella 14 riportiamo il numero di attacchi necessari per rompere il sistema DES: si noti come per un attacco chosen plaintext il numero di tentativi da effettuare sia sempre minore rispetto a quelli per un attacco known plaintext [11].

num. round	chosen plaintext	known plaintext
8	2^{14}	2^{38}
9	2^{24}	2^{44}
10	2^{24}	2^{43}
11	2^{31}	2^{47}
12	2^{31}	2^{47}
13	2^{39}	2^{52}
14	2^{39}	2^{51}
15	2^{47}	2^{56}
16	2^{47}	2^{55}

Tabella 14: Tentativi necessari per rompere il DES.

6 Modalità operative del DES

Al fine di usare il DES in una varietà di applicazioni crittografiche sono state sviluppate per esso quattro modalità operative [22]:

- Electronic Codebook Chaining (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB).

6.1 Electronic Codebook Chaining

Questo metodo è il piú semplice tra quelli proposti. In esso un messaggio in chiaro $x_1x_2\cdots x_n$ (dove ogni x_i rappresenta un blocco di 64 bit) viene cifrato applicando ripetutamente ad ogni blocco x_i l'algoritmo DES, sempre con la stessa chiave K, producendo un testo cifrato $y_1y_2\cdots y_n$.

Quindi i valori dei blocchi y_i sono dati da

$$y_i = DES_K(x_i).$$

Lo schema è illustrato in Figura 13.

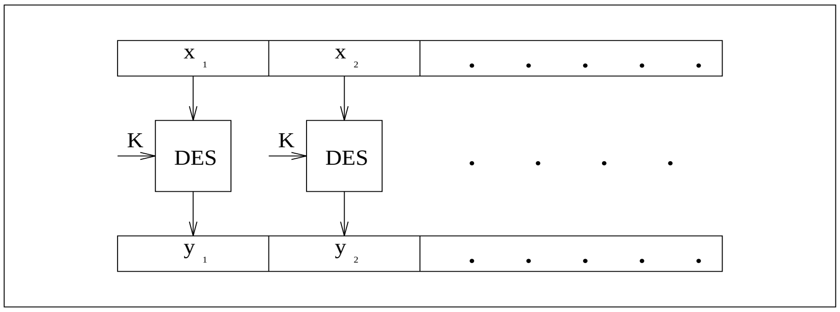


Figura 13: Schema dell'Electronic Codebook Chaining.

L'unico problema con questo metodo si ha quando il messaggio è costituito da un numero di bit che non è un multiplo di 64, cioè quando l'ultimo blocco non contiene esattamente 64 bit, ma di meno. La soluzione per questo caso non è definita dallo standard, ma bisogna convenire su come risolverlo.

L'idea potrebbe essere quella di inserire del testo opportuno alla fine del blocco, in modo tale che, in fase di decifratura, possa essere semplice distinguere il testo aggiunto da quello originale. Una possibile soluzione è quella di aggiungere al messaggio originale la sequenza $100\cdots 0$, in modo da completare l'ultimo blocco a 64 bit. Naturalmente, questa operazione deve essere fatta anche quando l'ultimo blocco è già di 64 bit, in modo da stabilire un criterio generale.

Questo significa che, in fase di decifratura si sa con certezza che la stringa finale del messaggio decifrato, cioè la stringa $100\cdots 0$, è stata aggiunta in fase di cifratura, e quindi non viene considerata come facente parte del messaggio originario.

L'operazione di decifratura considera un blocco alla volta del messaggio, cosí come accadeva per la cifratura, applicando però in questo caso l'algoritmo di decifratura DES.

Il *vantaggio* di questo metodo è che è abbastanza veloce.

Lo *svantaggio* è dato dal fatto che, durante la trasmissione del messaggio, potrebbe aversi sia la perdita di un blocco che l'attacco da parte di un nemico. In particolare, quest'ultimo potrebbe sostituire un blocco con un altro, senza la possibilità che colui che riceve il messaggio possa accorgersene. Per esempio egli potrebbe cambiare in un messaggio il blocco "\$1900" con "\$9100", oppure il blocco "do" con "do not" senza essere scoperto. Questo è dovuto alla mancanza di dipendenza tra i vari blocchi. Il primo metodo che evita questi inconvenienti è il Cipher Block Chaining.

6.2 Cipher Block Chaining

Dato un messaggio in chiaro $x_1 x_2 \cdots x_n$ ed una chiave nascosta K, in questa modalità un generico y_i è dato da:

$$y_i = DES_K(y_{i-1} \oplus x_i) \quad 1 \leq i \leq n \tag{5}$$

con $y_0 = IV$ (vettore di inizializzazione a 64 bit).

Il funzionamento di questa modalità è definito dallo schema di Figura 14.

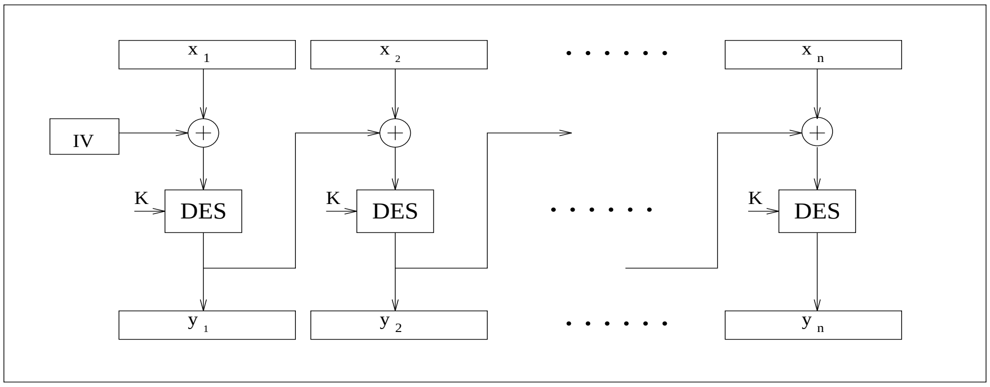


Figura 14: Schema di cifratura col Cipher Block Chaining.

Il procedimento di cifratura considera inizialmente il vettore IV, di 64 bit, quindi effettua lo xor bit a bit tra questi ed i primi 64 bit del messaggio in chiaro, infine applica alla stringa cosí ottenuta l'algoritmo di cifratura DES con chiave K, ottenendo il blocco cifrato y_1. Quest'ultimo viene poi utilizzato per fare lo xor bit a bit col blocco successivo del messaggio in chiaro x_2, quindi si applica alla stringa cosí ottenuta l'algoritmo DES sempre con la stessa chiave K, ottenendo il blocco cifrato y_2 e cosí via. Con questo metodo, si definisce in qualche modo un legame tra y_i ed il blocco precedente y_{i-1}.

Nota: Il vettore di inizializzazione IV potrebbe essere scelto a caso e tenuto nascosto per una maggiore sicurezza, tuttavia in molte implementazioni esso viene addirittura reso pubblico.

Per decifrare si utilizza lo schema di Figura 15.

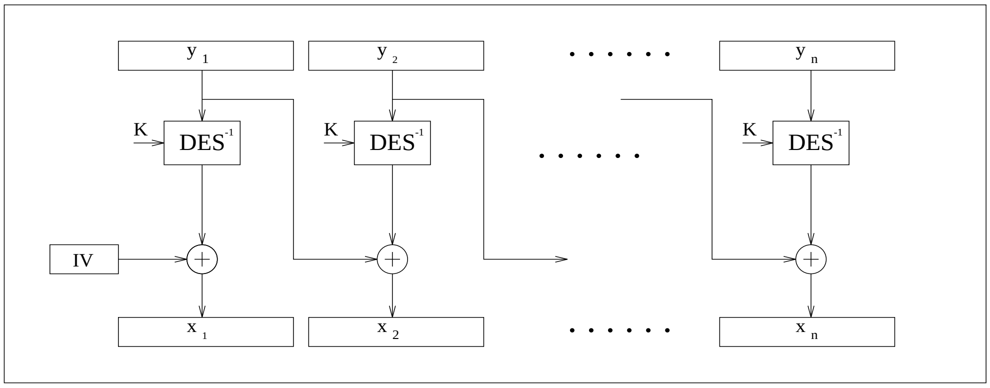


Figura 15: Schema di decifratura col Cipher Block Chaining.

Dal momento che y_1, come è noto, è il primo blocco in output del processo di cifratura, ottenuto come output all'algoritmo DES (Figura 14), per ottenere x_1 possiamo ragionare all'inverso del caso della cifratura, cioè si applica y_1 come input del DES^{-1} (algoritmo di decifratura) con la stessa chiave K, ottenendo quindi la stringa che precedentemente definiva lo xor bit a bit tra x_1 ed IV, che costituiva l'input del DES. Ora poiché si vuole calcolare x_1 si effettua l'operazione inversa, cioè lo xor bit a bit del valore in output del DES^{-1} con IV e ottenendo x_1. In generale, dalla (5) si ricava:

$$x_i = y_{i-1} \oplus DES_K^{-1}(y_i) \text{ per } 1 \leq i \leq n.$$

Il *vantaggio* di questo schema è che se si cambia un solo bit del messaggio originario cambia di conseguenza anche la parte rimanente del messaggio. Questa caratteristica elimina la possibilità di un attacco tramite sostituzione di un blocco. Infatti, in una tale evenienza il messaggio sarebbe irrimediabilmente compromesso.

Questo legame (dipendenza) esistente tra i vari blocchi del messaggio è alla base del funzionamento del Message Autentication Code (MAC) che verrà analizzato nella sezione *Applicazioni del DES*. Lo *svantaggio* di tale metodo è che in esso vengono eseguite piú operazioni rispetto all'ECC, e quindi rispetto ad esso è piú lento.

6.3 Cipher Feedback

In questa modalità viene fornito un vettore di inizializzazione IV, che è una stringa di 64 bit. Sia $x_1x_2\cdots x_n$ il messaggio in chiaro. Allora il generico y_i è definito da:

$$y_i = x_i \oplus DES_K(y_{i-1})$$

con $1 \leq i \leq n$. Il funzionamento di questa modalità è descritto nello schema di Figura 16.

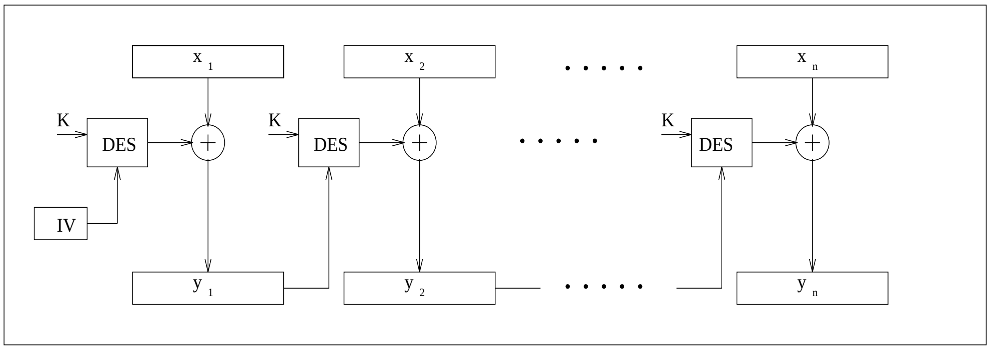


Figura 16: Schema di cifratura Cipher Feedback.

Per questa modalità ne esiste una versione piú generale detta Cipher Feedback con j bit (Figura 17).

Per essa si considera inizialmente una stringa di 64 bit (costituita dal vettore IV), che si puó immaginare suddivisa in due parti, di $64-j$ bit a sinistra e j bit a destra. Quindi si cifra tale stringa usando l'algoritmo di cifratura DES con chiave K.

Della stringa cifrata cosí ottenuta vengono considerati solo i j bit piú a sinistra (gli altri $64-j$ bit non vengono considerati). Quindi si effettua lo xor bit a bit tra questi j bit ed i prossimi j bit del messaggio in chiaro, ottenendo j bit del messaggio cifrato. Tali j bit cosí definiti rientrano nel circuito come definito nello schema, in quanto al passo successivo

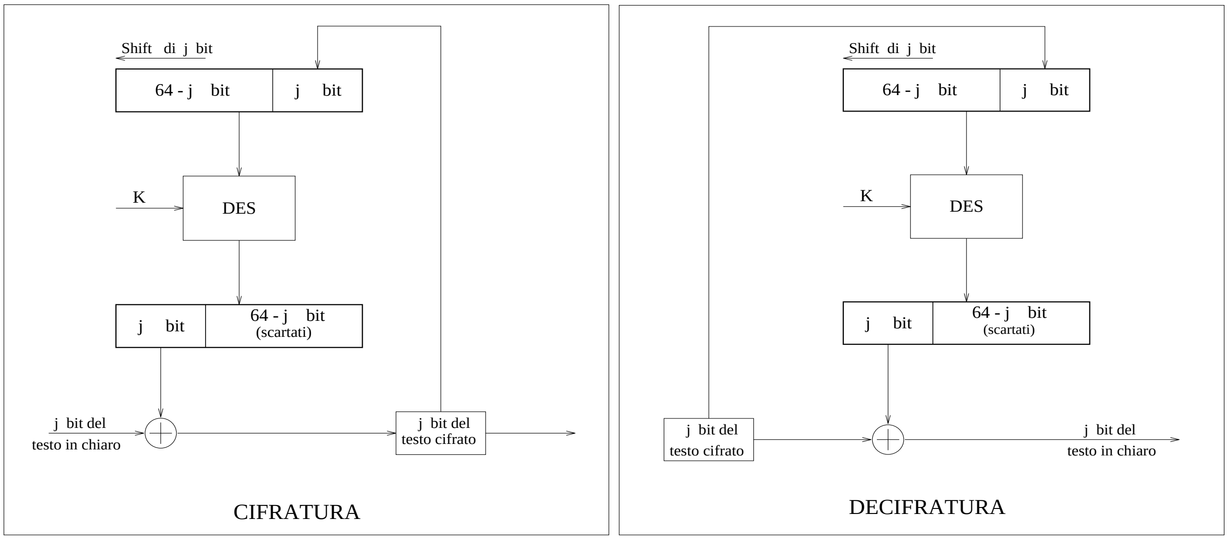


Figura 17: Schemi di cifratura e decifratura Cipher Feedback con j bit.

il vettore originale di 64 bit viene shiftato di j posizioni a sinistra, e gli ultimi j bit di tale vettore diventano i j bit del messaggio cifrato definiti. Ad ogni passo, da j bit del testo in chiaro otteniamo j bit del testo cifrato.

La decifratura avviene in maniera analoga alla cifratura, e il suo schema è quello di Figura 17.

L'unica variante significativa in tale schema è data dal fatto che i j bit del messaggio cifrato, vengono utilizzati per effettuare lo xor bit a bit con i j bit piú a sinistra dell'output del DES, oltre a costituire in seguito i nuovi j bit piú a destra del vettore iniziale (su cui viene applicato l'algoritmo DES), definendo cosí j bit del messaggio in chiaro.

Il *vantaggio* di questo schema è dato dal fatto che il valore di j puó essere scelto a piacimento. Per esempio alcune applicazioni scelgono $j = 8$, che significa cifrare 8 bit (1 byte) alla volta del messaggio in chiaro.

La cosa vantaggiosa in questo caso è che il valore della cifratura di questo byte puó essere prodotto subito, diversamente da quanto accadeva nei casi precedenti in cui bisognava aspettare di leggere prima tutti i 64 bit del messaggio in chiaro per poter produrre poi la cifratura del messaggio. Questo metodo si presta particolarmente alla cifratura *on-line*; infatti con $j = 8$ puó essere trasmesso un carattere alla volta.

Lo *svantaggio* di questa modalità è dovuta al fatto che per valori di j piccoli esso diventa sicuramente piú oneroso di quelli visti in precedenza.

6.4 Output Feedback

Anche in questa modalità si utilizza un vettore di inizializzazione IV. Essa presenta come primo passo la costruzione di una sequenza:

$$z_0 z_1 z_2 z_3 \cdots$$

dove $z_0 = IV$ e $z_i = DES_K(z_{i-1})$ con chiave K segreta. Come si puó notare tale stringa è completamente indipendente dal messaggio in chiaro, ed inoltre essa è utilizzata come un *one-time-pad*. Sia $x_1 x_2 \cdots x_n$ il messaggio in chiaro; il messaggio cifrato è definito da $y_1 y_2 \cdots y_n$ dove:

$$y_i = x_i \oplus z_i. \qquad (6)$$

Per la fase di decifratura si calcola la sequenza $z_0 z_1 z_2 \cdots$ allo stesso modo del caso della cifratura. Quindi i blocchi del messaggio in chiaro sono definiti da:

$$x_i = y_i \oplus z_i.$$

Anche per tale metodo c'è la variante a j bit, in particolare per la cifratura abbiamo lo schema in Figura 18.

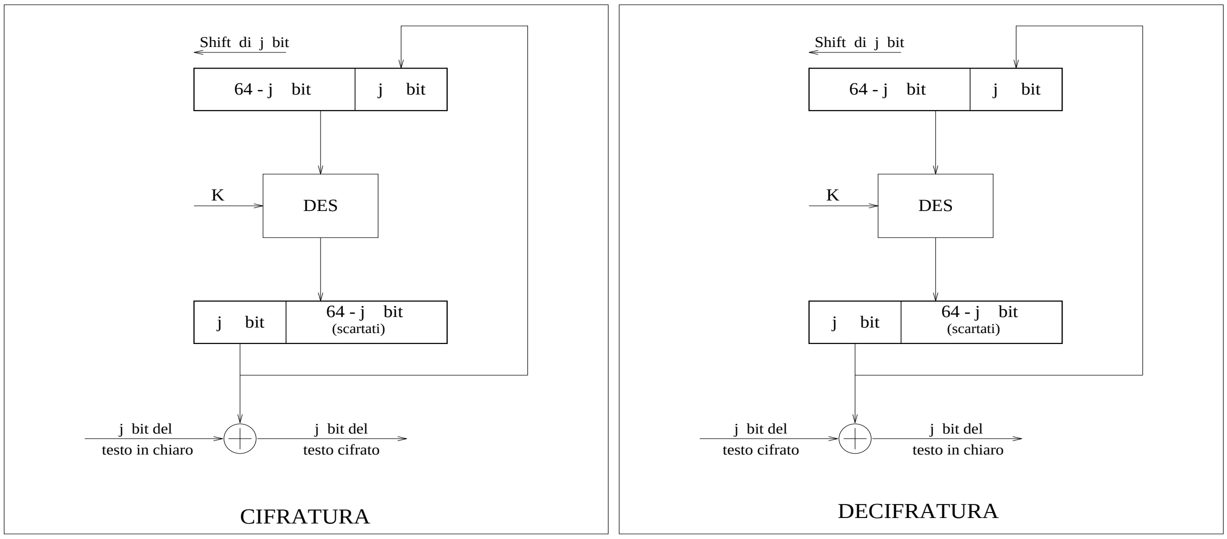


Figura 18: Schemi di cifratura e decifratura Output Feedback con j bit.

Questo schema è analogo a quello definito per il CFB, l'unica diversità stà nel fatto che il feed-back (cioè il riutilizzo di una certa parte dell'informazione) non riguarda j bit del testo cifrato, ma i j bit piú a sinistra della stringa in output prodotta dal DES.

Si puó notare che la parte superiore dello schema di cifratura è completamente indipendente dai bit del messaggio in chiaro, ed inoltre per $j = 64$ otteniamo lo schema generale per la cifratura, descritto precedentemente nella (6). Infatti considerando inizialmente il valore $z_0 = IV$, ed applicando ad esso l'algoritmo DES_K, ottengo i 64 bit che costituiscono z_1, quindi considerando tale stringa come input al DES ottengo la stringa z_2, cosí proseguendo definisco la sequenza di chiavi $z_0 z_1 z_2 \cdots$.

Queste chiavi verranno poi utilizzate per effettuare lo xor bit a bit con le stringhe x_i del messaggio in chiaro. La decifratura ha come schema quello in Figura 18.

Come si puó notare, la prima parte di esso è uguale a quella che viene utilizzata per effettuare la cifratura (ciò è valido dal momento che la sequenza del messaggio in chiaro generata da questa prima parte è indipendente dal messaggio in chiaro, cosí come dal messaggio cifrato).

Quindi, ad ogni passo si considerano i j bit piú a sinistra delle stringhe fornite in output dall'algoritmo DES, e si effettua lo xor bit a bit con j bit del messaggio cifrato, ottenendo cosí j bit del messaggio in chiaro.

Tale metodo ha il *vantaggio* che la sequenza $z_0 z_1 z_2 \cdots$ puó essere computata indipendentemente dal messaggio in chiaro, e quindi puó essere precalcolata (sfruttando per esempio dei tempi in cui la macchina è inattiva), in modo tale che quando viene immesso il messaggio da cifrare, la cifratura puó essere eseguita immediatamente: $y_i = x_i \oplus z_i$ (lo stesso vale per la decifratura). Quindi in realtà le operazioni di cifratura/decifratura vere è proprie consisteranno semplicemente di operazioni di xor che come sappiamo sono estremamente veloci da realizzare.

Lo *svantaggio* rispetto al metodo precedente è dovuto al fatto che il cambiamento di un bit nel messaggio originario provoca il cambiamento di un solo bit nel testo cifrato e viceversa, e tutti gli altri bit che precedono e seguono questi rimangono inalterati.

In definitiva possiamo dire che per tale modalità non c'è dipendenza tra i vari blocchi del messaggio cifrato, e questo porta ai problemi già visti in precedenza.

7 Applicazioni del DES

L'algoritmo DES puó essere usato come standard sia per la cifratura che per l'autenticazione dei dati:

Cifratura Dati: è semplice vedere come il DES puó essere usato per cifrare un testo in chiaro di 64 bit, tuttavia la lunghezza dei testi è raramente limitata a 64 bit. Per testi piú lunghi possono essere utilizzate le modalità di operazioni viste nella sezione precedente.

Ognuna di tali modalità ha vantaggi e svantaggi. Per esempio l'ECB è eccellente per la cifratura delle chiavi; CFB è usato tipicamente per cifrare caratteri individuali; OFB è tipicamente usato per cifrare comunicazioni via satellite (in cui è necessario ridurre al minimo la possibilità di errori alla stazione ricevente); ed infine entrambi CBC e CFB possono essere usati per l'autenticazione di dati.
Tuttavia per essi non esistono delle analisi formali che dimostrino qual'è il piú sicuro, a meno che non si faccia uso di ipotesi molto restrittive.

Autenticazione Dati: Originariamente il DES fu introdotto per permettere la cifratura e la decifratura di dati usati sul computer. Tuttavia però la sua applicazione si è estesa anche alla autenticazione dei dati. Tale necessità è nata dalla difficoltà di riconoscere se in un messaggio c'erano state delle modifiche o meno, durante la fase di trasmissione. Piú in particolare, nel caso in cui i dati erano cifrati con una delle due modalità CFB o CBC, allora queste possono essere usate per produrre un **Codice di Autenticazione del Messaggio** (MAC). Il MAC viene posto alla fine del messaggio in chiaro, ed è usato per convincere chi riceve il messaggio che esso non è stato alterato da nessuno. Cosí il MAC garantisce l'integrità (o autenticità) del messaggio (ma non la sua segretezza).
La modalità di operazione CBC viene usata per produrre un MAC nel seguente modo: supponiamo che Alice voglia trasmettere a Bob la sequenza $x_1 x_2 \cdots x_n$; costruiamo i blocchi del testo cifrato $y_1 y_2 \cdots y_n$ con chiave K (si noti che inizialmente il vettore IV contiene tutti zero). Definiamo y_n come il MAC del messaggio, quindi Alice trasmette la sequenza in chiaro $x_1 x_2 \cdots x_n$ seguita dal MAC y_n.
Quando Bob riceve il messaggio, egli puó ricostruire semplicemente il messaggio cifrato $y_1 y_2 \cdots y_n$ usando la chiave segreta K, e verificare se y_n ha lo stesso valore del MAC ricevuto come parte del messaggio.
Si noti che un nemico che vuole attaccare il sistema (crittoanalista) non puó produrre un MAC consistente, poiché non conosce la chiave di cifratura. Quindi se tale crittoanalista

intercetta il messaggio $x_1x_2\cdots x_ny_n$ e cambia uno o piú bit di tale messaggio, è altamente improbabile che egli riesca a cambiare opportunamente anche il MAC del messaggio, evitando che Bob si accorga delle modifiche.

Cifratura ed Autenticazione dei Dati: è spesso desiderabile combinare autenticità e segretezza. Questo potrebbe essere fatto nel modo seguente: Alice utilizza una prima chiave K_1 per produrre un MAC per il testo in chiaro definito da $x_1x_2\cdots x_n$, cosí come è stato visto al punto 2. Si supponga di indicare tale MAC con x_{n+1}.
Quindi Alice cifra la sequenza $x_1\cdots x_nx_{n+1}$ usando una seconda chiave K_2, e producendo il messaggio cifrato $y_1y_2\cdots y_{n+1}$. Quando Bob riceve tale messaggio, prima lo decifra usando la chiave K_2, e poi controlla che x_{n+1} è effettivamente il MAC per $x_1x_2\cdots x_n$ usando la chiave K_1.

7.1 Settori di applicazione del DES

Il DES trova applicazione nei seguenti settori:

Immagazzinamento Dati e Sistemi di Posta (ANSI X9.19): il DES viene utilizzato per la cifratura di password per l'accesso a sistemi computerizzati. L'algoritmo confronta la cifratura della stringa data in input come password con quella conservata in memoria, ed attiva o nega l'accesso al sistema a seconda che il match fra le due stringhe abbia o meno esito positivo. Un algoritmo FORTRAN implementa questa funzione utilizzando come chiave la password e come testo in chiaro l'identificazione dell'utente.
Il DES puó anche essere utilizzato per la protezione di file in memoria. Una pubblicazione speciale dell'NBS 500-54 descrive un sistema di notarizzazione di chiavi che puó essere integrato in un sistema computerizzato, per proteggere file da modifiche e divulgazioni non rintracciabili e per provvedere una segnatura digitale usando il DES. Questa tecnica di notarizzazione è usata nello standard ANSI $X9.17$. Il sistema di notarizzazione che incorpora il DES puó anche essere usato congiunto ad un sistema di posta per provvedere della posta sicura. Un trattatore crittografico che contiene le informazioni necassarie a decifrare ed autenticare un file posta è automaticamente aggiunto al file che è trasmesso al ricevente. Quest'ultimo in seguito potrà decifrare ed autenticare il file facilmente sfruttando quest'informazione.

Trasferimento Elettronico di Fondi (EFT) (ANSI X9.9): probabilmente l'uso piú significativo del DES è quello riguardante la protezione dei messaggi di trasferimento, relativi a vendite al dettaglio ed all'ingrosso. Le piccole e grandi comunità finanziarie hanno sviluppato degli standard per l'autenticazione di messaggi EFT, in particolare per la cifratura (ANSI X9.23) e per la gestione delle chiavi (ANSI X9.17 e ANSI X9.24). I dati protetti tramite DES sono relativi a trasferimenti di fondi che variano da $50 a multi-milioni di dollari. In particolare il governo degli Stati Uniti è responsabile di trasferimenti di bilioni di dollari giornalieri. Per far si che questi trasferimenti siano sicuri, il Dipartimento del Tesoro Americano ha dato inizio alla politica precedentemente citata sull'autenticazione dei messaggi EFT.
Anche la Banca federale coopera con il Tesoro per la buona riuscita di queste operazioni. Uno dei sistemi che si sta considerando consiste di strumenti manuali contenenti chiavi DES usati nella cifratura di messaggi finanziari. Questi strumenti forniscono la chiave per l'autenticazione delle firme di documenti cartacei.

Scambi di Dati Economici: molte aziende stanno automatizzando il processo delle transazioni commerciali per ridurre i costi ed aumentare l'efficienza aziendale. Le transazioni commerciali saranno realizzate tramite strumenti elettronici invece che tramite strumenti tradizionali basati su supporti cartacei, e l'**ANSI Accredited Standard Committee X12** stà sviluppando i formati che saranno usati per questo tipo di comunicazioni che definiranno lo standard chiamato Electronic Business data interchange (EBDI).
L'EBDI incorporerà parecchi standard basati sul DES [ ]. L'ANSI X9.9 provvederà le protezioni circa modifiche e sostituzioni non autorizzate; lo standard ANSI X9.23 preverrà scoperte non autorizzate; mentre per avere una generazione, distribuzione e memorizzazione sicura di chiavi DES verranno usate le tecniche specificate nello standard ANSI X9.17.
Attualmente la General Motors e sette banche associate fanno uso di questi standard.

A Appendice

NSA: la *National Security Agency* (NSA) è responsabile per il coordinamento centralizzato, direzione ed esecuzione di funzioni tecniche altamente specializzate, in supporto ad attività governative statunitensi per proteggere le comunicazioni.
L'NSA fu fondata da direttive presidenziali nel 1952 come un'agenzia del *Department of Defense* sotto la direzione, l'autorità e il controllo del Segretario della difesa. Per molti anni la sua esistenza fu tenuta nascosta.

L'NSA è organizzata per compiere due missioni nazionali:

1. La sicurezza dei sistemi d'informazione (missione INFOSEC) fornisce prodotti e servizi, per proteggere i sistemi di sicurezza nazionali, classificati e non, da intercettazioni e accessi non autorizzati.
2. La missione SIGINT consite nell'intercettare e decodificare comunicazioni straniere inerenti la sicurezza nazionale.

Un risultato della missione SIGINT è stato raggiunto nella seconda guerra mondiale, quando cioè gli statunitensi ruppero il codice militare giapponese e conobbero il piano di invasione del Midway Island. Oggi SIGINT continua a giocare un ruolo importante al mantenimento dello stato di superpotenza degli Stati Uniti.
Si dice che l'NSA è uno dei più grandi datori di lavoro per matematici negli Stati Uniti e probabilmente nel mondo. I matematici all'NSA contribuiscono direttamente alle due missioni dell'agenzia: partecipano alla progettazione di sistemi di cifratura che proteggeranno l'integrità dei sistemi di informazione statunitensi, mentre altri studiano la debolezza dei codici avversari.
La sua forza-lavoro rappresenta una inusuale combinazione di specialisti che include: analisti, ingegnieri, fisici, matematici, linguisti, informatici, ricercatori, addetti alle pubbliche relazioni, ufficiali di sicurezza, esperti in data flow, manager e assistenti amministrativi.

Il sito internet dell'NBS è il seguente: *http://www.nsa.gov:8080/* .

NIST: è il *National Institute of Standard and Technology* (NIST), una divisione dell'*U.S. Department of Commerce*. Esso fu fondato nel 1901, col nome di *National Bureau of Standard* (NBS), dal Congresso "per l'assistenza industriale nello sviluppo di tecnologie necessarie a migliorare la qualità dei prodotti per assicurarne l'affidabilità e per facilitare la rapida commercializzazione di prodotti basati su nuove scoperte scientifiche". Nel 1988 l'originario NBS ha cambiato il suo nome in NIST. Per quasi 100 anni i laboratori del NIST/NBS hanno lavorato con l'industria per lo sviluppo di standard. Gli standard ufficiali sono pubblicati come pubblicazioni FIPS (Federal Information Processing Standard).
Con l'approvazione, nel 1987, del Computer Security Act, il NIST ricevette un mandato per definire degli standard per assicurare la sicurezza di informazioni nei sistemi di computer governativi.
Il compito primario del NIST è di promuovere lo sviluppo economico statunitense, lavorando affianco all'industria, per sviluppare ed applicare tecnologie, fornire valutazioni e standard. Per realizzare questo compito il NIST sviluppa i seguenti programmi.

Advanced Technology Program (ATP): fornisce premi ad industrie per la ricerca e lo sviluppo in aree tecnologiche ad alto rischio.

Manufacturing Extension Partnership (MEP): offre assistenza tecnologica ed economica a piccole e medie compagnie statunitensi.

Laboratory Research and Services: sviluppa e propone tecniche di valutazione, metodi di test, standard e altri tipi di infrastrutture tecnologiche e servizi che provvedono ad un linguaggio comune necassario all'industria in tutti i livelli del commercio: ricerca, sviluppo, produzione e marketing.

Baldrige National Quality Program: con le industrie gestisce i premi *Malcolm National Quality* che possono essere conferiti annualmente a compagnie di piccole dimensioni, industrie e servizi, e forniscono delle guide per migliorare la qualità di prodotti.

Il NIST ha un budget di circa 810 milioni di dollari (stime per il '96); ha uno staff costituito da circa 3.300 dipendenti tra scienziati, ingegnieri, tecnici e personale di servizio, più 1.250 ricercatori in visita ogni anno.
Il sito internet del NIST è : *http://www.nist.gov/* .

Riferimenti

[1] ANSI X3.92, *American National Standard for Data Encryption Algorithm (DEA)*, American National Standards Institute, 1981.

[2] S. K. Baneryce, *High Speed Implementation of DES*, Computer and Security, v.1, 1982, pp. 261-267.

[3] E. Biham e A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, J. Cryptology, v.4, 1991, pp. 3-72.

[4] E. Biham e A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

[5] E. F. Brickell, J. H. Moore e M. R. Purtill, *Structure in the S-BOXES of DES, Lecture Notes in Computer Science, Advances in Cryptology - Proceedings of CRYPTO '86*, Springer-Verlag, 1987, v. 263, pp. 3-8.

[6] P. Christoffersson et al., *Crypto Users' Handbook*, IFIP 1988, Elsevier Science Publishers.

[7] D. W. Davies, *Some Regular Properties of the DES, Advances in Cryptology - Proceedings of CRYPTO '82*, Plenum Press 1983, pp. 89-96.

[8] M. Davio e Y. Desmedt, *Analitical Characteristics of the Data Encryption Standard, Advances in Cryptology - Proceedings of CRYPTO '83*, Plenum Press, 1984, pp. 171-202.

[9] M. Davio e Y. Desmedt, *Dependance of Output on Input in DES: Small Avalanche Characteristics, Advances in Cryptology - Proceedings of CRYPTO '84*, Berlin: Springer - Verlag, 1985, pp. 359-376.

[10] Department of the Treasury, *Electronic Funds and Securities Transfer Policy-Message Authentication and Enhanced Security*, No. 106-09, Department of the Treasury, Oct 2, 1986.

[11] W. Diffie, *Cryptography Techology: Fifteen Years Forecast*, BNR Inc., Jan 1981.

[12] W. Diffie e M. Hellman, *Exhaustive Cryptanalysis of the NBS Data Encryption Standard*, Computer, v. 10, No. 6, Jun 1977, pp. 74-84.

[13] H. Eberle, *A High Speed DES Implementation for Network Applications, Advances in Cryptology - Proceedings of CRYPTO '92* , Berlin: Springer - Verlag, 1993, pp. 527-545.

[14] R. C. Fairfield, A. Matusevich e J. Plany, *An LSI Digital Encryption Processor (DEP), Advances in Cryptology - Proceedings of CRYPTO '84*, Berlin: Springer - Verlag, 1985, pp. 115-143.

[15] J. Goubertand e Y. Desmedt, *Efficient Hardware Implementation of the DES, Advances in Cryptology - Proceedings of CRYPTO '84*, Berlin: Springer - Verlag, 1985, pp. 147-173.

[16] GSA Federal Standards 1026, *Telecommunications: General Security Requirements for Equipment Using the Data Encryption Standard*, General Services Administration, Apr 1982.

[17] M. E. Hellman, R. C. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig e P. Schweitzer, *Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard*, Technical Report SEL 76-042, Information System Lab, Department of Electrical Engineering, Stanford University, 1976.

[18] G. B. Kolata, *Flaws Found in Popular Code*, Science, 219, January 28, 1983, pp. 369-370.

[19] A. G. Konheim, *Cryptography: a Primer*, New York, J. Wiley & Sons, 1981.

[20] R. Morris, N. J . Soane e A. D. Wyner, *Assessment of NBS Proposed Data Encryption Standard*, Cryptologia, v.1, No.3, Jul 1977, pp. 281-291.

[21] NBS FIPS PUB 46, *Data Encryption Standard*, National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[22] NBS FIPS PUB 46, *Des Modes of operation*, National Bureau of Standards, 1977.

[23] *Proposed Reaffirmation of Federal Information Processing Standards (FIPS) 46-1, Data Encryption Standard DES*, Federal Register, v.57, N.177, 11 Sep, 1992.

[24] B. Schneier, *Applied Criptography: Protocols, Algorithms and Source Code in C*, John Wiley & Sons, Inc., 1994.

[25] M. E. Smid e D. K. Branstad, *The Data Encryption Standard: past and future*, Contemporary Cryptology, The Science of Information Integrity, pp. 43-64, IEEE Press, 1992.

[26] J. L. Smith, *The Design of Lucifer, A Cryptographic Device for Data Communications*, IBM Research Report RC3326, 1971.

[27] D. R. Stinson, *Criptography Theory and Practice*, 1995, by CRC Press, Inc..

[28] M. J. Wiener, *Efficient DES key search*, unpublished manuscript presented at CRYPTO '93 Rump Session.