Tecniche di Progettazione Automatica	**Anno Acc. 1995-96**
PROF. ALFREDO DE SANTIS	

Cifrari a blocchi

a cura di Maria Coppola e Ornella Panza *11 Luglio 1996*

Indice

1 Introduzione

Circa 20 anni fa è stato sviluppato quello che ancora oggi è uno degli algoritmi di cifratura a blocchi più usati nel mondo, il DES, o Data Encryption Standard. Dal momento in cui è stato adottato come standard, il DES viene rivisto approssimativamente ogni 5 anni; la revisione più recente, effettuata nel gennaio del 1994, lo ha confermato fino al 1998.

Dopo il 1998, però, il governo americano non userà più il DES come standard, poiché recentemente tale algoritmo è stato oggetto di molte critiche che hanno messo in discussione la sua forza crittografica.

Negli Stati Uniti, inoltre, il materiale crittografico è regolato dalle stesse leggi che regolano gli armamenti. In particolare, l'esportazione delle implementazioni di algoritmi crittografici è un reato grave quanto l'esportazione di armi nucleari. Il governo americano permette di esportare codice crittografico solo nel caso in cui la chiave utilizzata abbia una lunghezza inferiore a 40 bit. In tal caso, infatti, il cifrario risulta piuttosto debole.

Dunque, mentre tutti possono conoscere l'algoritmo DES, nessuna implementazione realizzata negli Stati Uniti può essere resa nota altrove. Questo implica che un pacchetto software prodotto negli Stati Uniti non può contenere il DES se deve essere utilizzato in un altro Paese.

Per rimpiazzare il DES recentemente sono stati progettati molti algoritmi di cifratura a blocchi alternativi. I più conosciuti sono:

- **Triplo DES:** variante del DES che aumenta la dimensione dello spazio di chiavi da 2^{56} a 2^{112}, facendo uso di due chiavi e di una tripla cifratura mediante DES.
- **IDEA:** cifrario a blocchi che utilizza una chiave di 128 bit per cifrare blocchi di 64 bit.
- **RC5:** algoritmo che utilizza una chiave di lunghezza variabile, stabilita al momento della cifratura.

Gli algoritmi di cifratura IDEA e RC5 non violano le leggi sull'esportazione di materiale crittografico dagli Stati Uniti: il primo, infatti, è stato sviluppato a Zurigo; il secondo, invece, utilizzando una chiave a lunghezza variabile, può essere esportato in una modalità opportuna, per poi essere eventualmente modificato.

Il pacchetto software PGP (Pretty Good Privacy), prodotto negli Stati Uniti, realizza la crittografia a chiave pubblica mediante l'algoritmo di cifratura IDEA.

2 Triplo DES

Una delle debolezze del DES relative alla sicurezza è la lunghezza della chiave che, essendo pari a 56 bit (64 bit se consideriamo gli 8 bit di parità) fornisce uno spazio delle chiavi piuttosto piccolo: con un attacco che esamina lo spazio delle chiavi in maniera esaustiva, sono necessari in media 2^{55} tentativi per indovinare la chiave.

2.1 Doppio DES

Da questa osservazione è nata l'idea di progettare un cifrario a blocchi alternativo al DES: si è pensato, infatti, di rendere il DES stesso più resistente agli attacchi di forza bruta, cifrando due volte il messaggio in chiaro con due chiavi diverse (una per ogni passo di cifratura).

In tal modo la chiave diventa lunga $56 \cdot 2 = 112$ bit con un notevole incremento dello spazio delle chiavi.

Il cifrario a blocchi che si ottiene apportando tali modifiche al DES viene detto *Doppio DES*.

- **Cifratura.** Dato un blocco X di 64 bit del testo in chiaro e 2 chiavi K_1 e K_2 a 56 bit, cifriamo X con la chiave K_1 ed otteniamo un blocco A di 64 bit; cifriamo quindi A con la chiave K_2 ed otteniamo il corrispondente testo cifrato Y:

$$Y = DES_{K_2}(DES_{K_1}(X)).$$

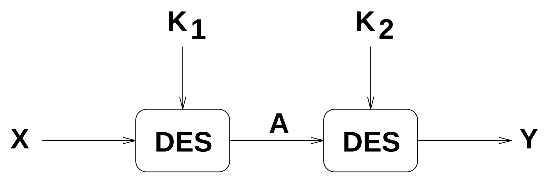

Figura 1: Doppia cifratura.

- **Decifratura.**

  La decifratura è analoga alla cifratura, ma richiede che le chiavi siano applicate nell'ordine inverso. In particolare, dato un blocco Y di 64 bit del testo cifrato e 2 chiavi K_1 e K_2 a 56 bit, decifriamo Y con la chiave K_2 ed otteniamo un blocco A di 64 bit; decifriamo, quindi, A con la chiave K_1 ed otteniamo il corrispondente testo in chiaro X:

$$X = DES^{-1}_{K_1}(DES^{-1}_{K_2}(Y)).$$

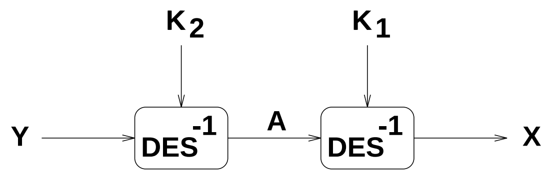

Figura 2: Doppia decifratura.

2.2 Non chiusura del DES rispetto all'operazione di cifratura/ decifratura

Soltanto nel 1992 si è avuta la certezza che "l'allungamento" della chiave da 56 bit a 112 bit rende effettivamente il Doppio DES più resistente del DES agli attacchi di forza bruta. È stato dimostrato, infatti, che DES non è un gruppo e che quindi non è chiuso rispetto all'operazione di cifratura/decifratura [1]. Se fosse stato vero il contrario, se cioè l'utilizzo di due chiavi fosse stato equivalente all'utilizzo di una sola chiave, allora la cifratura doppia, ed in generale la cifratura con un numero qualsiasi di passi (e di chiavi), sarebbe stata inutile; in tal caso, infatti, la ricerca esaustiva della chiave di cifratura del Doppio DES avrebbe richiesto uno sforzo computazionale pari a quello necessario per rompere il DES singolo.

Vale, dunque, la seguente

Proposizione 2.1 *Per ogni coppia di chiavi (K_1, K_2) non esiste alcuna chiave K_3 tale che*

$$DES_{K_2}(DES_{K_1}(X)) = DES_{K_3}(X)$$

per ogni testo in chiaro X.

Dimostrazione: Dimostriamo in modo informale tale proposizione.

Consideriamo la cifratura col DES come una funzione che associa blocchi di 64 bit a blocchi di 64 bit:

$$DES : \{0,1\}^{64} \rightarrow \{0,1\}^{64}.$$

Più precisamente a ciascuno dei 2^{64} possibili blocchi in input, la cifratura col DES associa uno dei 2^{64} possibili blocchi in modo univoco. Questo è necessario affinché la decifratura sia possibile; se, infatti, a due distinti blocchi input fosse associato, utilizzando una specifica chiave, lo stesso blocco output, allora in fase di decifratura sarebbe impossibile risalire al testo in chiaro originario.

Dunque, partendo dai 2^{64} possibili blocchi del testo in chiaro, la cifratura col DES produce 2^{64} blocchi di testo cifrato, i quali non sono altro che una permutazione dei blocchi di partenza.

Essendo i blocchi considerati pari a 2^{64}, vi sono $2^{64}!$ funzioni distinte che generano una permutazione di tali blocchi. Per avere un'idea dell'ordine di grandezza di questo numero, osserviamo che

$$2^{64}! > 10^{347380000000000000000} > 10^{10^{20}}.$$

D'altra parte il DES definisce una corrispondenza tra blocchi, diversa per ogni chiave; perciò, se utilizziamo una singola chiave di 56 bit, il numero totale di associazioni distinte è pari a 2^{56}. Osserviamo che

$$\begin{aligned} 2^{56} &= 10^{(\log_{10} 2)\cdot 56} \\ &\approx 10^{56/3.32} \\ &\approx 10^{16.86} \ll 10^{10^{20}}. \end{aligned}$$

Pertanto è ragionevole pensare che, se applichiamo DES utilizzando due chiavi differenti, l'associazione univoca dei blocchi di 64 bit prodotta sarà una delle $2^{64}!$ associazioni possibili e non una delle 2^{56} che si possono avere con una singola applicazione del DES. $\square$

2.3 Attacco Meet in the Middle

Sebbene resista molto bene agli attacchi di forza bruta, il Doppio DES non viene utilizzato in pratica, in quanto non incrementa la sicurezza rispetto alla singola cifratura. Esiste, infatti, un algoritmo che permette la rottura di tale schema con uno sforzo computazionale pari a quello necessario per rompere il DES singolo [2].

Tale algoritmo, detto *Meet in the Middle*, non dipende dalla particolare struttura del DES, e pertanto può essere usato per attaccare un qualunque cifrario a blocchi. Esso si basa sul fatto che, se

$$Y = DES_{K_2}(DES_{K_1}(X))$$

allora
$$A = DES_{K_1}(X) = DES_{K_2}^{-1}(Y).$$
Supponiamo di conoscere una particolare coppia (X, Y), dove X è il testo in chiaro e Y il testo cifrato, e di voler determinare la chiave (K_1, K_2) utilizzata nella cifratura di X.

L'attacco Meet in the Middle viene effettuato nel modo seguente. Per prima cosa cifriamo X utilizzando tutte le 2^{56} possibili chiavi K_1, memorizziamo i testi cifrati in una tabella ed eventualmente ordiniamo la tabella in base a questi risultati:

K_1	A
$K_{1,1}$	$DES_{K_{1,1}}(X)$
$K_{1,2}$	$DES_{K_{1,2}}(X)$
$K_{1,3}$	$DES_{K_{1,3}}(X)$
$K_{1,4}$	$DES_{K_{1,4}}(X)$
.	.
.	.
.	.
.	.
$K_{1,2^{56}}$	$DES_{K_{1,2^{56}}}(X)$

Successivamente decifriamo Y usando tutte le 2^{56} possibili chiavi K_2 e memorizziamo i risultati in un'altra tabella:

K_2	A
$K_{2,1}$	$DES_{K_{2,1}}^{-1}(Y)$
$K_{2,2}$	$DES_{K_{2,2}}^{-1}(Y)$
$K_{2,3}$	$DES_{K_{2,3}}^{-1}(Y)$
$K_{2,4}$	$DES_{K_{2,4}}^{-1}(Y)$
.	.
.	.
.	.
.	.
$K_{2,2^{56}}$	$DES_{K_{2,2^{56}}}^{-1}(Y)$

(I valori contenuti in questa seconda tabella potrebbero anche non essere memorizzati, ma utilizzati man mano che vengono calcolati.)

A questo punto per conoscere la coppia di chiavi (K_1, K_2) che fa passare dal testo in chiaro X al testo cifrato Y, basta confrontare i valori $DES_{K_{1,i}}(X)$ e $DES_{K_{2,j}}^{-1}(Y)$ elencati nelle rispettive tabelle. Se
$$DES_{K_{1,i}}(X) = DES_{K_{2,j}}^{-1}(Y) \qquad \text{per qualche } i \text{ e } j$$
allora le due chiavi corrispondenti potrebbero formare la coppia cercata. Non è detto, però, che questa sia effettivamente la chiave corretta perché, in realtà, possono esistere diverse chiavi che dallo stesso testo in chiaro X generano lo stesso testo cifrato Y. In particolare, per ogni testo in chiaro X, il Doppio DES può produrre 2^{64} possibili valori di testo cifrato. D'altra parte il Doppio DES usa una chiave a 112 bit, per cui le chiavi possibili sono 2^{112}. Segue che,

se scegliamo a caso una coppia (X,Y), il numero di coppie di chiavi che trasformano X in Y è in media

$$\frac{2^{112}}{2^{64}} = 2^{48}.$$

Pertanto l'attacco considerato produce circa 2^{48} collisioni sulla coppia (X,Y), e questo non ci consente di risalire alla coppia di chiavi realmente utilizzata, cioè non rende ancora possibile la rottura del cifrario. Allora scegliamo a caso un'altra coppia (X',Y') di testo in chiaro-testo cifrato (indipendente dalla prima) tale che Y' è prodotto da X' utilizzando la stessa coppia di chiavi (K_1,K_2). Costruiamo, poi, le relative tabelle cifrando X' per tutti i possibili valori di K_1 e decifrando Y' per tutti i possibili valori di K_2:

K_1	A
$K_{1,1}$	$DES_{K_{1,1}}(X')$
$K_{1,2}$	$DES_{K_{1,2}}(X')$
$K_{1,3}$	$DES_{K_{1,3}}(X')$
$K_{1,4}$	$DES_{K_{1,4}}(X')$
.	.
.	.
.	.
.	.
$K_{1,2^{56}}$	$DES_{K_{1,2^{56}}}(X')$

K_2	A
$K_{2,1}$	$DES^{-1}_{K_{2,1}}(Y')$
$K_{2,2}$	$DES^{-1}_{K_{2,2}}(Y')$
$K_{2,3}$	$DES^{-1}_{K_{2,3}}(Y')$
$K_{2,4}$	$DES^{-1}_{K_{2,4}}(Y')$
.	.
.	.
.	.
.	.
$K_{2,2^{56}}$	$DES^{-1}_{K_{2,2^{56}}}(Y')$

Se in corrispondenza di una stessa coppia di chiavi $(K_{1,i},K_{2,j})$ si ha

$$\begin{cases} DES_{K_{1,i}}(X) & = & DES^{-1}_{K_{2,j}}(Y) \\ \\ DES_{K_{1,i}}(X') & = & DES^{-1}_{K_{2,j}}(Y') \end{cases}$$

allora con molta probabilità la coppia di chiavi $(K_{1,i},K_{2,j})$ è quella effettivamente utilizzata nella cifratura. Osserviamo, infatti, che il numero di collisioni è, in questo caso, estremamente basso, ovvero

$$\frac{2^{112}}{2^{128}} = 2^{-16}$$

pertanto la probabilità di determinare la chiave corretta è $1 - 2^{-16} = 0.99998474$.

Questa probabilità può essere aumentata ulteriormente se, invece di una coppia di valori, ne consideriamo una tripla, una quadrupla, e così via.

L'algoritmo Meet in the Middle può essere riassunto nei passi seguenti:

Input: (X, Y) e (X', Y')

1. calcola $DES_{K_1}(X)$ e $DES_{K_1}(X')$ per ogni $K_1 \in \{0,1\}^{56}$
2. calcola $DES^{-1}_{K_2}(Y)$ e $DES^{-1}_{K_2}(Y')$ per ogni $K_2 \in \{0,1\}^{56}$
3. ricerca K_1 e K_2 tali che $DES_{K_1}(X) = DES^{-1}_{K_2}(Y)$ e $DES_{K_1}(X') = DES^{-1}_{K_2}(Y')$
4. return $K = (K_1, K_2)$.

Calcoliamo il tempo richiesto da questa procedura:

- i passi 1 e 2 richiedono uno sforzo computazionale pari a 2^{56};
- la ricerca richiede un tempo che dipende dalla struttura utilizzata per memorizzare i dati. Se, ad esempio, ordiniamo tali dati rispetto alla cifratura DES, allora la ricerca richiede $\log_2 2^{56} = 56$ tentativi. Se, invece, utilizziamo una tabella hash (ammesso che riusciamo a costruire una tabella hash così grande), allora il tempo è, in media, una costante.

Osserviamo che la precomputazione delle 4 tabelle, eventualmente effettuata nei tempi morti della macchina, richiede un tempo elevato, ma rende la ricerca estremamente veloce.

In generale, il numero di test effettuati nella ricerca è abbastanza piccolo, pertanto il costo dell'intera procedura è dell'ordine di 2^{56}.

In definitiva il Doppio DES è un cifrario a blocchi che utilizza una chiave di 112 bit e che può essere rotto con uno sforzo dell'ordine di 2^{56}, quindi non è migliore del DES singolo. Per questo motivo il Doppio DES non viene utilizzato in pratica.

2.4 Triplo DES con due chiavi

Un ovvio contrattacco all'algoritmo Meet in the Middle è l'uso di tre passi di cifratura con tre chiavi diverse. Il cifrario che otteniamo prende il nome di *Triplo DES*.

Dato un blocco X di 64 bit del testo in chiaro e tre chiavi K_1, K_2 e K_3 a 56 bit, la prima idea è quella di cifrare X utilizzando tre volte il DES nel modo seguente:

$$Y = DES_{K_3}(DES_{K_2}(DES_{K_1}(X))).$$

La chiave $K = (K_1, K_2, K_3)$ ha, in questo caso, una lunghezza pari a $3 \cdot 56 = 168$ bit.

Analizziamo l'attacco Meet in the Middle su un cifrario di questo tipo.

Da una parte, dato X, calcoliamo $DES_{K_1}(X)$ per ogni possibile chiave K_1; dall'altra, dato Y, calcoliamo $DES^{-1}_{K_2}(DES^{-1}_{K_3}(Y))$ per tutte le possibili chiavi K_2 e K_3. Il costo è di 2^{56} per la prima operazione e di 2^{112} per la seconda. Alternativamente avremmo potuto calcolare $DES_{K_2}(DES_{K_1}(X))$ da una parte e $DES^{-1}_{K_3}(Y)$ dall'altra con uguale costo.

L'attacco prosegue, poi, con la ricerca di K_1, K_2 e K_3 tali che

$$DES_{K_1}(X) = DES^{-1}_{K_2}(DES^{-1}_{K_3}(Y))$$

e con la verifica della validità di questa uguaglianza in corrispondenza di altre coppie di testo in chiaro-testo cifrato, in modo da individuare la chiave corretta con una certa sicurezza.

Osserviamo che lo sforzo computazionale dell'intera procedura è pari a 2^{112}, e non $2^{3 \cdot 56}$; perciò, anche se utilizziamo 3 chiavi, è come se ne utilizzassimo solo due: infatti, un cifrario che utilizza due chiavi di 56 bit, può essere rotto con un attacco di forza bruta in un tempo pari a 2^{112}. Inoltre tre chiavi sono piuttosto ingombranti e poco maneggevoli.

Allora eliminiamo la chiave K_3 e al suo posto riutilizziamo la chiave K_1; la cifratura di X avviene nel modo seguente:

$$Y = DES_{K_1}(DES_{K_2}(DES_{K_1}(X))).$$

Il Triplo DES può essere ulteriormente modificato sostituendo il secondo passo di cifratura con un passo di decifratura [3]. Pertanto, dato il testo in chiaro X, si ha:

$$Y = DES_{K_1}(DES^{-1}_{K_2}(DES_{K_1}(X))).$$

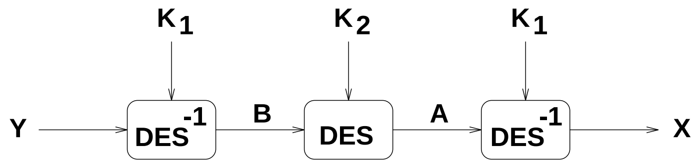

Figura 3: Tripla cifratura.

Questo permette di utilizzare il Triplo DES anche per decifrare dati cifrati col DES singolo; infatti, dato X, vale la seguente uguaglianza:

$$Y = DES_K(DES^{-1}_K(DES_K(X)))$$

(al secondo membro la cifratura e la decifratura più esterne si annullano).

Analizziamo, infine, la decifratura del Triplo DES.

Dato il testo cifrato Y e la coppia di chiavi (K_1, K_2), per ottenere il testo in chiaro X dobbiamo effettuare le operazioni inverse alla cifratura, cioè dobbiamo decifrare Y con la chiave K_1, quindi cifrare il valore ottenuto con la chiave K_2 ed infine decifrare il risultato utilizzando di nuovo la chiave K_1:

$$X = DES^{-1}_{K_1}(DES_{K_2}(DES^{-1}_{K_1}(Y))).$$

Questa ultima versione del Triplo DES è spesso utilizzata in pratica in alternativa al DES ed il suo grado di sicurezza è considerato piuttosto alto; lo svantaggio è che risulta meno efficiente del DES singolo di un fattore 3.

Attualmente non sono noti attacchi crittoanalitici pratici al Triplo DES. Coppersmith [4] ha osservato che un attacco di forza bruta per la ricerca della chiave è dell'ordine di $2^{112} \approx 5 \cdot 10^{35}$ e ha stimato che il costo della crittoanalisi differenziale cresce in modo esponenziale rispetto al DES singolo, ed è maggiore di 10^{52}.

Altri possibili attacchi al Triplo DES sono descritti in [5] e [6]. Tali attacchi, sebbene non siano realizzabili in pratica, potrebbero costituire le basi per attacchi futuri.

Il Triplo DES è stato adottato negli standard per il trattamento delle chiavi ANSI X9.17 ed ISO 8732, e nel PEM (Privacy Enhanced Mail).

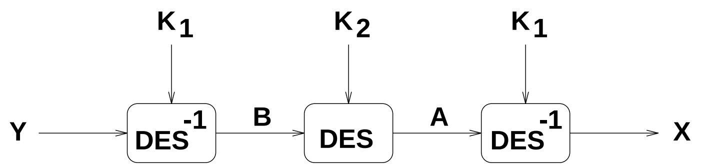

Figura 4: Tripla decifratura.

3 IDEA

Xuejia Lai e James Massey dello Swiss Federal Institute of Technology hanno progettato un efficiente algoritmo di cifratura a blocchi, chiamato IDEA (International Data Encryption Algorithm). La versione originale fu pubblicata nel 1990 [7]. Una seconda versione dell'algoritmo, progettata per resistere meglio agli attacchi di crittoanalisi differenziale, fu presentata nel 1991 [8] e descritta in dettaglio nel 1992 [9].

IDEA è un algoritmo di pubblico dominio, ma solo per uso privato. Infatti, essendo munito di brevetto internazionale, non può essere utilizzato liberamente nell'ambito di applicazioni di tipo commerciale.

3.1 Principi di progettazione

IDEA è un cifrario a blocchi che usa una chiave di 128 bit per cifrare blocchi di 64 bit, a differenza del DES che usa una chiave di 56 bit.

Gli obiettivi fondamentali di questo schema di cifratura sono:

- forza crittografica;
- facilità d'implementazione.

Analizziamo le caratteristiche di questo algoritmo che garantiscono una elevata forza crittografica.

- **Lunghezza del blocco**: il blocco da cifrare deve essere abbastanza lungo da impedire analisi statistiche, cioè deve evitare che il crittoanalista tragga vantaggio dal fatto che alcuni blocchi si presentino con una frequenza maggiore di altri. D'altro canto la complessità d'implementazione di un'efficiente funzione di cifratura cresce esponenzialmente con la misura del blocco [10]. Un compromesso accettabile è l'uso, in IDEA, di un blocco di 64 bit.

- **Lunghezza della chiave**: la chiave deve essere abbastanza lunga da prevenire ricerche esaustive. IDEA, con una chiave di 128 bit, garantisce un'ampia sicurezza.

- **Confusione**: il testo cifrato deve dipendere dal testo in chiaro e dalla chiave in modo non lineare. L'obiettivo è complicare la dipendenza delle statistiche del testo cifrato da quelle del testo in chiaro. Per ottenere ciò, IDEA usa tre operazioni di gruppo "incompatibili" su coppie di sottoblocchi di 16 bit, a differenza del DES che si basa principalmente sull'operazione $\oplus$ e su piccole S-boxes non lineari.

- **Diffusione**: per nascondere la struttura statistica del testo in chiaro, ogni bit del testo in chiaro e della chiave deve influenzare tutti i bit del testo cifrato. Si è visto che in IDEA ciascun bit output della prima iterazione dipende da ogni bit del testo in chiaro e da ogni bit delle sottochiavi [7]. La struttura di IDEA, utilizzando 8 iterazioni, fornisce un buon grado di diffusione. Inoltre, si può dimostrare che questa struttura usa il minor numero di operazioni necessario per ottenere una diffusione completa [7].

I seguenti requisiti, invece, facilitano l'implementazione sia hardware che software [7].

- **Uso di sottoblocchi**: IDEA assume che l'architettura della macchina sia a 16 bit e che le operazioni siano eseguite su sottoblocchi di dimensioni pari alla lunghezza della parola macchina, cioè 16 bit. Questo incrementa notevolmente la velocità di implementazione.
- **Somiglianza tra cifratura e decifratura**: IDEA, come il DES, realizza sia la cifratura che la decifratura usando lo stesso modulo, ma un diverso ordine delle sottochiavi.
- **Struttura regolare**: IDEA è costituito da due moduli ripetuti più volte.
- **Uso di operazioni semplici**: IDEA usa operazioni semplici da implementare, quali addizione, shift e così via.

Ciascuna delle tre operazioni utilizzate da IDEA è eseguita su due input a 16 bit e produce un singolo output a 16 bit. Tali operazioni sono:

1. OR-esclusivo bit a bit, denotato con $\oplus$;
2. addizione di interi modulo 2^{16}, denotata con $\boxplus$ ed eseguita senza considerare i riporti;
3. moltiplicazione di interi modulo $2^{16}+1$, denotata con $\odot$. L'input e l'output sono trattati come interi a 16 bit unsigned, con la variante che il blocco costituito da tutti zeri rappresenta 2^{16}. Per esempio

   $$0000000000000000 \odot 1000000000000000 = 1000000000000001$$

   poiché
   $$2^{16} \cdot 2^{15} \bmod (2^{16}+1) = 2^{15}+1.$$

Al contrario di quanto si potrebbe immaginare, anche questa ultima operazione può essere realizzata velocemente. Vale, infatti, il seguente

Lemma 3.1 *Siano a, b due interi non negativi a n bit. Allora*

$$ab \bmod (2^n+1) = \begin{cases} (ab \bmod 2^n) - (ab \operatorname{div} 2^n) & \textit{se } (ab \bmod 2^n) \geq (ab \operatorname{div} 2^n) \\ (ab \bmod 2^n) - (ab \operatorname{div} 2^n) + 2^n + 1 & \textit{altrimenti} \end{cases}$$

Dimostrazione: Siano $a, b \neq 0$ in $\mathbf{Z}_{2^n+1}$. Allora esistono un unico q e un unico r tali che

$$ab = q(2^n+1) + r \tag{1}$$

con

$$0 \leq r < 2^n + 1 \tag{2}$$

$$0 \leq q < 2^n \tag{3}$$

Sommando membro a membro la (2) e la (3) otteniamo

$$q + r < 2^{n+1}. \tag{4}$$

Inoltre, dalla (1) segue

$$ab = q2^n + (q + r). \tag{5}$$

Poiché valgono la (4) e la (5), si ha

$$ab \operatorname{div} 2^{\mathrm{n}} = \begin{cases} q & se\ q + r < 2^n \\ q + 1 & se\ q + r \geq 2^n \end{cases}$$

$$ab \bmod 2^n = \begin{cases} q + r & se\ q + r < 2^n \\ q + r - 2^n & se\ q + r \geq 2^n. \end{cases}$$

Allora, sottraendo la prima uguaglianza dalla seconda nei due casi considerati, si ottiene

$$r = \begin{cases} (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) & se\ q + r < 2^n \\ (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) + 2^{\mathrm{n}} + 1 & se\ q + r \geq 2^{\mathrm{n}}. \end{cases} \tag{6}$$

Ma

$$q + r < 2^n \iff (ab \bmod 2^n) \geq (ab \operatorname{div} 2^{\mathrm{n}}).$$

Infatti

$$\begin{aligned} q + r < 2^n &\implies (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) = (\mathrm{q} + \mathrm{r}) - \mathrm{q} = \mathrm{r} \geq 0 \\ &\implies (ab \bmod 2^n) \geq (ab \operatorname{div} 2^{\mathrm{n}}). \end{aligned}$$

D'altra parte, se fosse vero che $(ab \bmod 2^n) \geq (ab \operatorname{div} 2^{\mathrm{n}})$ e $q + r \geq 2^n$, avremmo

$$\begin{aligned} (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) \geq 0 &\implies (q + r - 2^n) - (q + 1) \geq 0 \\ &\implies r - 2^n - 1 \geq 0 \\ &\implies r \geq 2^n + 1. \end{aligned}$$

Ma questo è impossibile per come abbiamo definito r. Pertanto

$$(ab \bmod 2^n) \geq (ab \operatorname{div} 2^{\mathrm{n}}) \Longrightarrow \mathrm{q} + \mathrm{r} < 2^{\mathrm{n}}.$$

Osserviamo, infine, che $r = ab \bmod (2^n + 1)$. Quindi possiamo anche scrivere la (6) come

$$ab \bmod (2^n + 1) = \begin{cases} (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) & \textit{se } (ab \bmod 2^n) \geq (ab \operatorname{div} 2^{\mathrm{n}}) \\ (ab \bmod 2^n) - (ab \operatorname{div} 2^{\mathrm{n}}) + 2^{\mathrm{n}} + 1 & \textit{altrimenti.} \end{cases}$$

Si noti che $ab \bmod 2^n$ corrisponde agli n bit meno significativi di ab, mentre $ab \operatorname{div} 2^{\mathrm{n}}$ si ottiene shiftando ab a destra di n bit in modo non ciclico. ▯

Le tre operazioni elencate sono dotate di inverse altrettanto facili da calcolare. In particolare, se Z è un blocco di 16 bit,

- l'inversa dell'OR-esclusivo è l'OR-esclusivo stesso;
- l'inverso additivo, indicato con $-Z$, è tale che $-Z \boxplus Z = 0$;
- l'inverso moltiplicativo, indicato con Z^{-1}, è tale che $Z \odot Z^{-1} = 1$.

La Tabella 1 mostra i valori che le tre operazioni producono quando sono applicate a numeri di 2 bit (anziché di 16 bit). In ciascuna colonna sono riportati i valori decimali e i corrispondenti valori binari.

X		Y		$X \boxplus Y$		$X \odot Y$		$X \oplus Y$	
0	00	0	00	0	00	1	01	0	00
0	00	1	01	1	01	0	00	1	01
0	00	2	10	2	10	3	11	2	10
0	00	3	11	3	11	2	10	3	11
1	01	0	00	1	01	0	00	1	01
1	01	1	01	2	10	1	01	0	00
1	01	2	10	3	11	2	10	3	11
1	01	3	11	0	00	3	11	2	10
2	10	0	00	2	10	3	11	2	10
2	10	1	01	3	11	2	10	3	11
2	10	2	10	0	00	0	00	0	00
2	10	3	11	1	01	1	01	1	01
3	11	0	00	3	11	2	10	3	11
3	11	1	01	0	00	3	11	2	10
3	11	2	10	1	01	1	01	1	01
3	11	3	11	2	10	0	00	0	00

Tabella 1: Funzioni usate in IDEA.

Nel processo di cifratura di IDEA, le tre diverse operazioni di gruppo sono organizzate in modo tale che l'output di una operazione di un tipo non è mai usata come input ad un'operazione dello stesso tipo. Le 3 operazioni sono incompatibili, nel senso che:

1. Nessuna coppia di operazioni soddisfa la proprietà distributiva. Ad esempio:

$$a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c).$$

Se $a = b = 1_{10} = 0000000000000001_2$, il lato sinistro di questa disuguaglianza è uguale a $2_{10} = 0000000000000010_2$, mentre il lato destro è uguale a $4_{10} = 0000000000000100_2$.

2. Nessuna coppia di operazioni soddisfa la proprietà associativa. Ad esempio:

$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c.$$

Se $a = b = 1_{10} = 0000000000000001_2$, il lato sinistro di questa disuguaglianza è uguale a $1_{10} = 0000000000000001_2$, mentre il lato destro è uguale a $3_{10} = 0000000000000011_2$.

Questo rende lo schema di cifratura particolarmente resistente agli attacchi, poiché ne deriva l'impossibilità di modificare l'ordine della sequenza di operazioni, o di fare delle semplificazioni.

La combinazione di queste tre operazioni fornisce una trasformazione complessa dell'input che rende la crittoanalisi molto più difficile di quanto non lo sia per il DES, che utilizza la sola funzione $\oplus$.

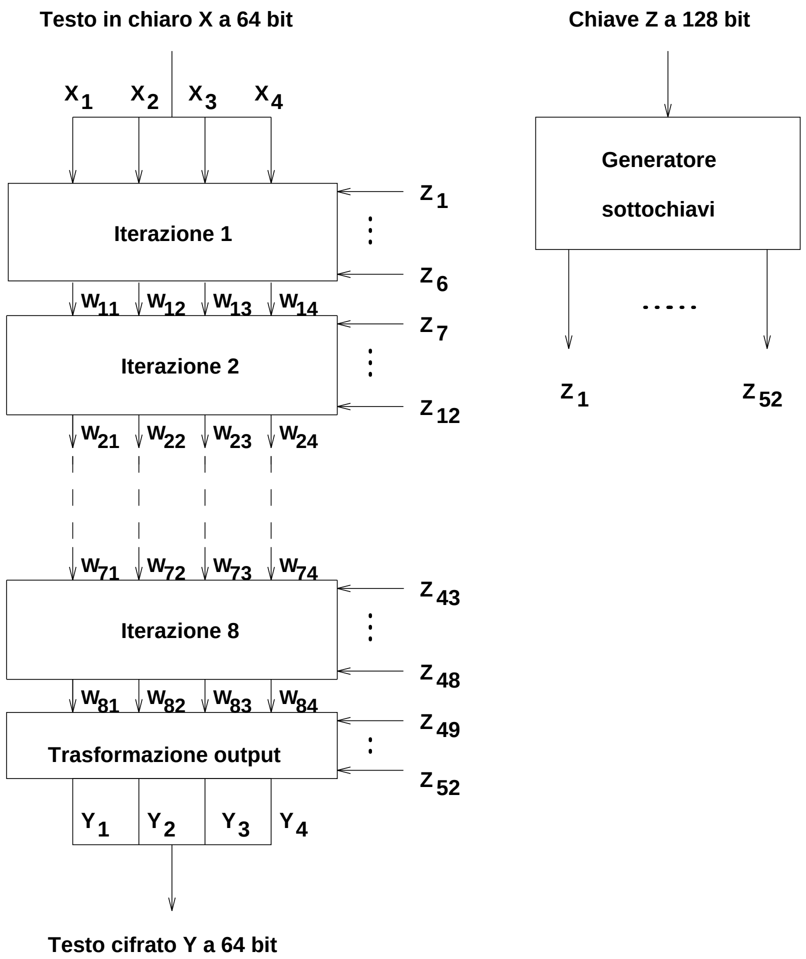

Figura 5: Struttura di IDEA.

3.2 Cifratura

Nella cifratura il blocco di 64 bit del testo in chiaro è partizionato in 4 sottoblocchi a 16 bit X_1, X_2, X_3, X_4. I 4 sottoblocchi del testo in chiaro sono poi trasformati in 4 sottoblocchi di testo cifrato Y_1, Y_2, Y_3, Y_4 sotto il controllo di 52 sottochiavi di 16 bit che sono determinate dalla chiave segreta di 128 bit.

Lo schema di cifratura, illustrato in Figura 5, è costituito da 8 iterazioni uguali seguite da una trasformazione finale. Ciascuna iterazione prende in input 4 sottoblocchi e 6 sottochiavi. La trasformazione finale, invece, prende in input 4 sottoblocchi e 4 sottochiavi.

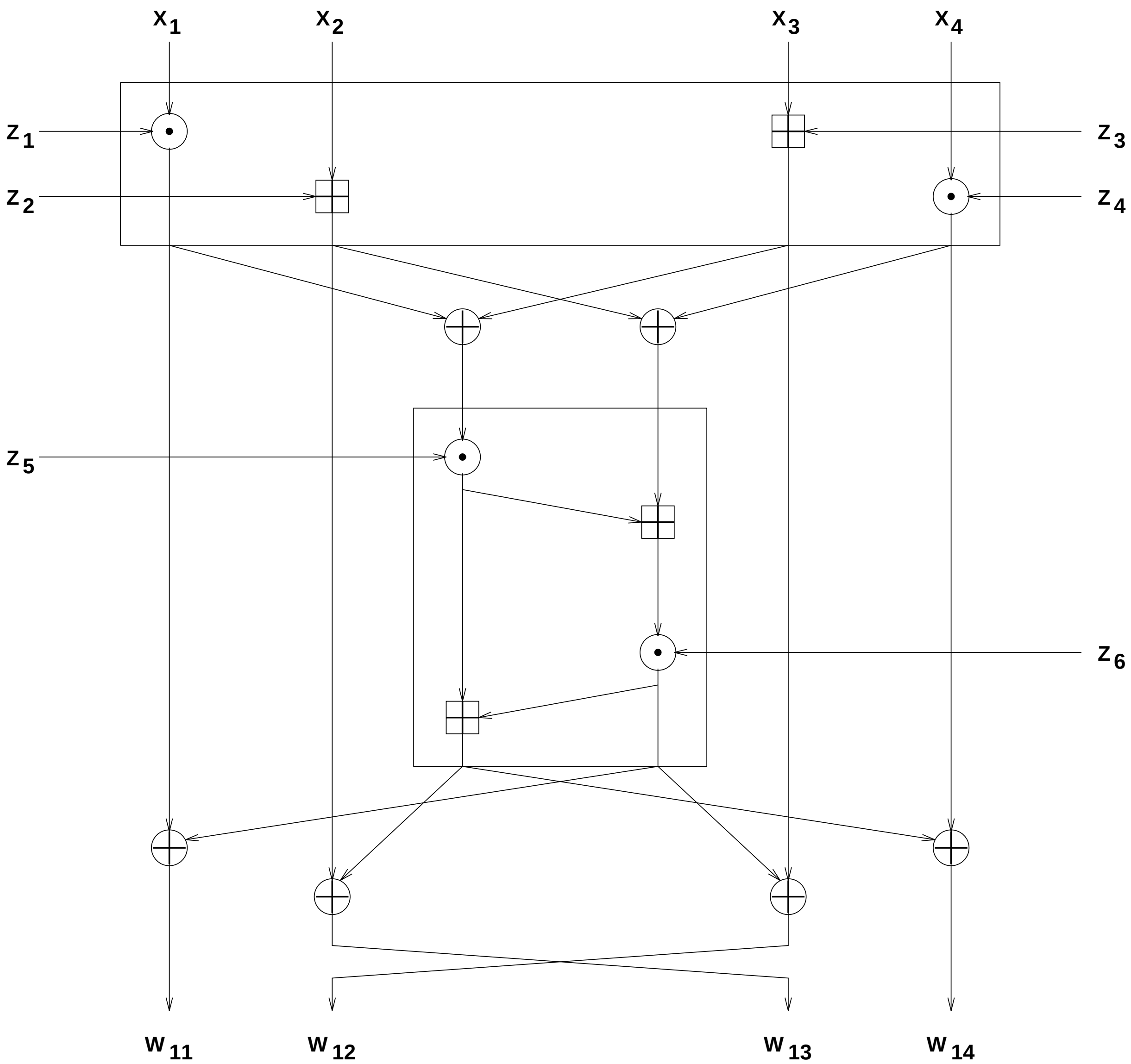

Figura 6: Prima iterazione di IDEA.

La prima iterazione, illustrata in Figura 6, inizia con una trasformazione che combina i quattro sottoblocchi input con le prime quattro sottochiavi, mediante le operazioni di addizione e moltiplicazione. Questa trasformazione è messa in risalto dal rettangolo superiore in Figura 6. I risultati sono poi combinati, mediante l'operazione $\oplus$, per produrre i due blocchi in input ad una particolare struttura, chiamata MA (Moltiplicazione/Addizione) che è illustrata in Figura

7. La struttura MA, inoltre, prende in input due sottochiavi e produce due blocchi di 16 bit. Infine i quattro blocchi risultanti dalla prima trasformazione e i due blocchi prodotti da MA vengono combinati tramite $\oplus$. I risultati così ottenuti, dopo aver scambiato fra loro il secondo e il terzo blocco, rappresentano l'output dell'iterazione. Quest'ultimo passo incrementa il mixing dei bit processati e rende l'algoritmo più resistente alla crittoanalisi differenziale.

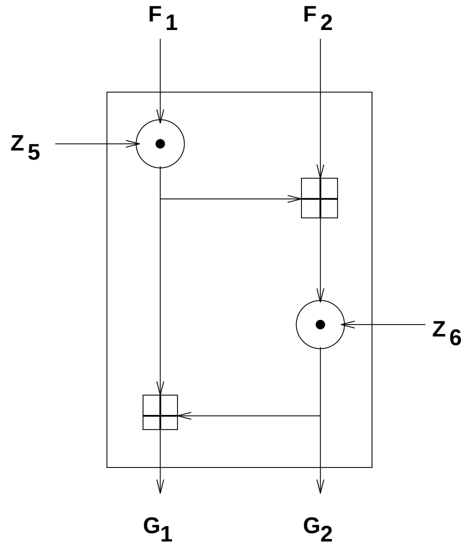

Figura 7: Struttura MA.

Il nono passo dell'algoritmo, denominato "trasformazione output", viene illustrato in maniera dettagliata nella Figura 8. Notiamo che esso ha la stessa struttura del rettangolo superiore delle iterazioni precedenti. La sola differenza è che il secondo ed il terzo input vengono scambiati prima di essere applicati all'unità operazionale. Questo accorgimento annulla l'interscambio effettuato alla fine dell'ottava iterazione ed è necessario per mantenere la stessa struttura sia nella cifratura che nella decifratura, come vedremo in seguito.

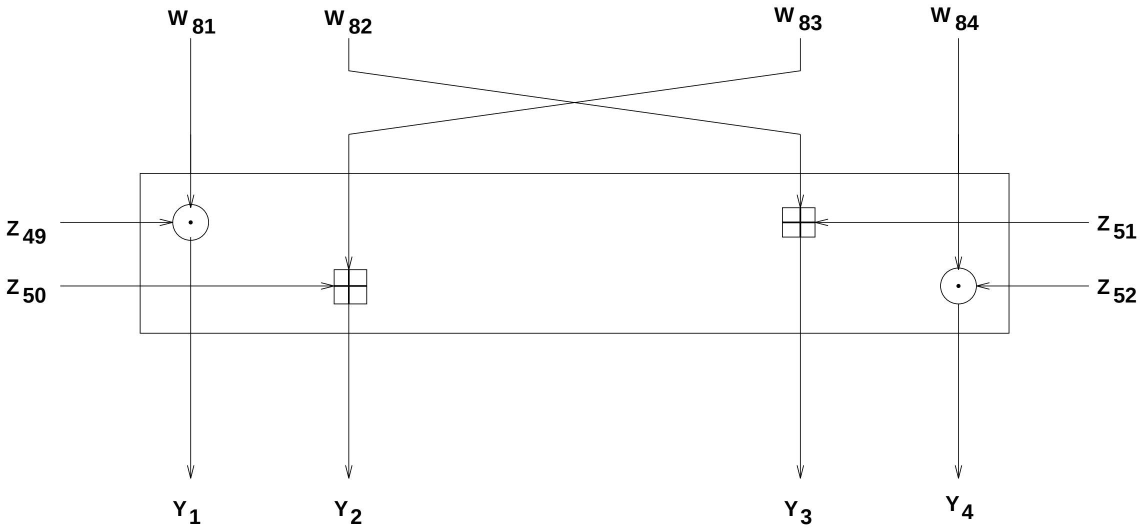

Figura 8: Trasformazione output.

L'algoritmo ha il seguente schema di generazione delle sottochiavi: le prime otto sottochiavi

$Z_1 ... Z_8$ sono prese direttamente dalla chiave, con Z_1 uguale ai 16 bit più significativi, Z_2 corrispondente ai successivi 16 bit e così via. Poi alla chiave è applicato uno shift circolare a sinistra di 25 posizioni e sono estratte le successive otto sottochiavi. La procedura è ripetuta sino alla generazione delle 52 sottochiavi.

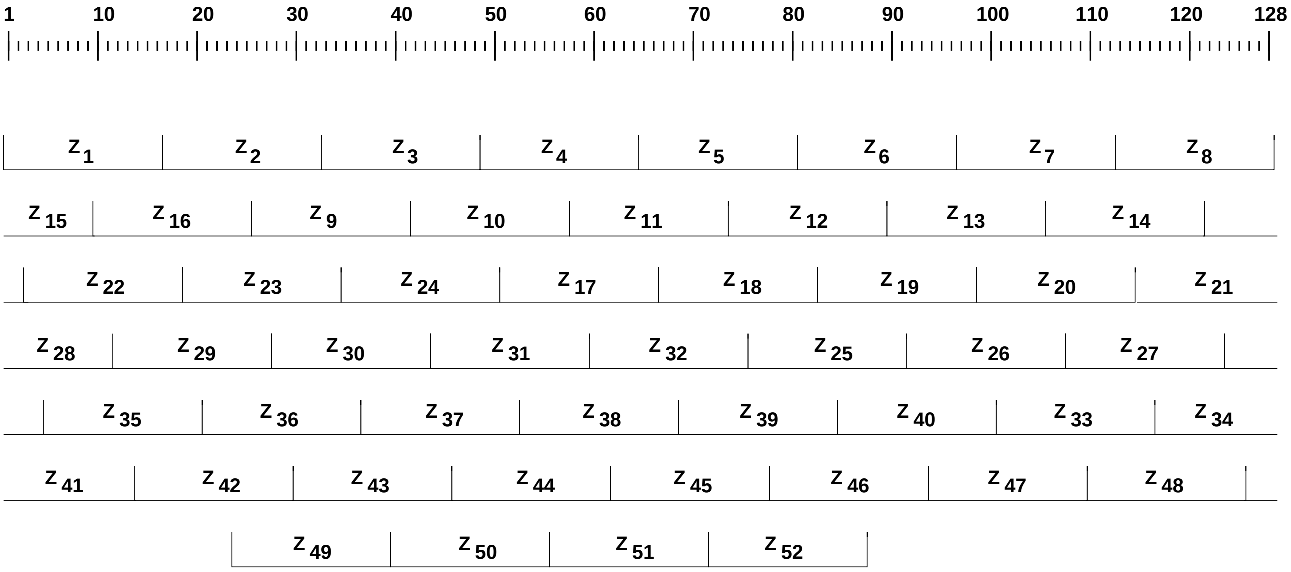

Figura 9: Sottochiavi IDEA.

La Figura 9 evidenzia la corrispondenza fra i bit della chiave e le sottochiavi. Si deduce che la schedulazione delle chiavi è eseguita in maniera molto veloce.

Questa tecnica assegna i bit della chiave alle sottochiavi in modo che la prima sottochiave usata in ciascuna iterazione è composta da un differente insieme di bit della chiave di partenza, come illustrato di seguito:

$$\begin{array}{ll} Z_1 = \mathrm{Z}[1...16] & Z_{25} = \mathrm{Z}[76...91] \\ Z_7 = \mathrm{Z}[97...112] & Z_{31} = \mathrm{Z}[44...59] \\ Z_{13} = \mathrm{Z}[90...105] & Z_{37} = \mathrm{Z}[37...53] \\ Z_{19} = \mathrm{Z}[83...98] & Z_{43} = \mathrm{Z}[30...45]. \end{array}$$

Ciò preclude la possibilità di avere sottochiavi uguali.

Inoltre i 96 bit delle sottochiavi usate in una singola iterazione sono, ad eccezione delle iterazioni 1 e 8, non contigui, cosicché non c'è una semplice relazione di shift tra le sottochiavi di una iterazione e quelle di un'altra. Infatti, solo 6 sottochiavi sono usate in ciascuna iterazione, mentre le sottochiavi estratte ad ogni rotazione della chiave sono 8.

3.3 Decifratura

Il processo di decifratura è essenzialmente uguale al processo di cifratura. In pratica la decifratura si ottiene usando il testo cifrato come input della struttura IDEA mostrata in Figura 5, ma con una diversa schedulazione delle sottochiavi.

Le sottochiavi di decifratura $U_1 ... U_{52}$ dipendono dalle sottochiavi di cifratura nel modo seguente:

1. Le prime quattro sottochiavi dell'iterazione di decifratura i derivano dalle prime quattro sottochiavi dell'iterazione di cifratura $10 - i$. Il passo di trasformazione è considerato come l'iterazione 9. In particolare la prima e la quarta sottochiave sono uguali all'inverso moltiplicativo modulo $2^{16}+1$ delle corrispondenti prima e quarta sottochiave di cifratura. Poiché $2^{16} + 1$ è un numero primo, ogni intero non nullo $Z_j \leq 2^{16}$ ha un unico inverso moltiplicativo modulo $2^{16} + 1$.

   Le iterazioni 1 e 9 hanno seconda e terza sottochiave uguali all'inverso additivo modulo 2^{16} delle corrispondenti seconda e terza sottochiave di di cifratura.

2. Nelle prime otto iterazioni le ultime due sottochiavi dell'iterazione di decifratura i sono esattamente uguali alle ultime due sottochiavi dell'iterazione di cifratura $9 - i$.

Passo	Designazione	Equivalente a
Iterazione 1	$Z_1\ Z_2\ Z_3\ Z_4\ Z_5\ Z_6$	$Z[1..96]$
Iterazione 2	$Z_7\ Z_8\ Z_9\ Z_{10}\ Z_{11}\ Z_{12}$	$Z[97..128; 26..89]$
Iterazione 3	$Z_{13}\ Z_{14}\ Z_{15}\ Z_{16}\ Z_{18}\ Z_{19}$	$Z[90..128; 1..25; 51..82]$
Iterazione 4	$Z_{19}\ Z_{20}\ Z_{21}\ Z_{22}\ Z_{23}\ Z_{24}$	$Z[83..128; 1..50]$
Iterazione 5	$Z_{25}\ Z_{26}\ Z_{27}\ Z_{28}\ Z_{29}\ Z_{30}$	$Z[76..128; 1..43]$
Iterazione 6	$Z_{31}\ Z_{32}\ Z_{33}\ Z_{34}\ Z_{35}\ Z_{36}$	$Z[44..75; 101..128; 1..36]$
Iterazione 7	$Z_{37}\ Z_{38}\ Z_{39}\ Z_{40}\ Z_{41}\ Z_{42}$	$Z[37..100; 126..128; 1..29]$
Iterazione 8	$Z_{43}\ Z_{44}\ Z_{45}\ Z_{46}\ Z_{47}\ Z_{48}$	$Z[30..125]$
Trasformazione	$Z_{49}\ Z_{50}\ Z_{51}\ Z_{52}$	$Z[23..86]$

Tabella 2: Sottochiavi di cifratura.

Passo	Designazione	Equivalente a
Iterazione 1	$U_1\ U_2\ U_3\ U_4\ U_5\ U_6$	$Z_{49}^{-1}\ -Z_{50}\ -Z_{51}\ Z_{52}^{-1}\ Z_{47}\ Z_{48}$
Iterazione 2	$U_7\ U_8\ U_9\ U_{10}\ U_{11}\ U_{12}$	$Z_{43}^{-1}\ -Z_{45}\ -Z_{44}\ Z_{46}^{-1}\ Z_{41}\ Z_{42}$
Iterazione 3	$U_{13}\ U_{14}\ U_{15}\ U_{16}\ U_{18}\ U_{19}$	$Z_{37}^{-1}\ -Z_{39}\ -Z_{38}\ Z_{40}^{-1}\ Z_{35}\ Z_{36}$
Iterazione 4	$U_{19}\ U_{20}\ U_{21}\ U_{22}\ U_{23}\ U_{24}$	$Z_{31}^{-1}\ -Z_{33}\ -Z_{32}\ Z_{34}^{-1}\ Z_{29}\ Z_{30}$
Iterazione 5	$U_{25}\ U_{26}\ U_{27}\ U_{28}\ U_{29}\ U_{30}$	$Z_{25}^{-1}\ -Z_{27}\ -Z26\ Z_{28}^{-1}\ Z_{23}\ Z_{24}$
Iterazione 6	$U_{31}\ U_{32}\ U_{33}\ U_{34}\ U_{35}\ U_{36}$	$Z_{19}^{-1}\ -Z_{21}\ -Z_{20}\ Z_{22}^{-1}\ Z_{17}\ Z_{18}$
Iterazione 7	$U_{37}\ U_{38}\ U_{39}\ U_{40}\ U_{41}\ U_{42}$	$Z_{13}^{-1}\ -Z_{15}\ -Z_{14}\ Z_{16}^{-1}\ Z_{11}\ Z_{12}$
Iterazione 8	$U_{43}\ U_{44}\ U_{45}\ U_{46}\ U_{47}\ U_{48}$	$Z_7^{-1}\ -Z_9\ -Z_8\ Z_{10}^{-1}\ Z_5\ Z_6$
Trasformazione	$U_{49}\ U_{50}\ U_{51}\ U_{52}$	$Z_1^{-1}\ -Z_2\ -Z_3\ Z_4^{-1}$

Tabella 3: Sottochiavi di decifratura.

Le Tabelle 2 e 3 riassumono queste relazioni.

Per verificare che lo stesso algoritmo con le opportune chiavi di decifratura produce il giusto risultato, consideriamo la Figura 10, la quale mostra il processo di cifratura a sinistra e il processo di decifratura a destra.

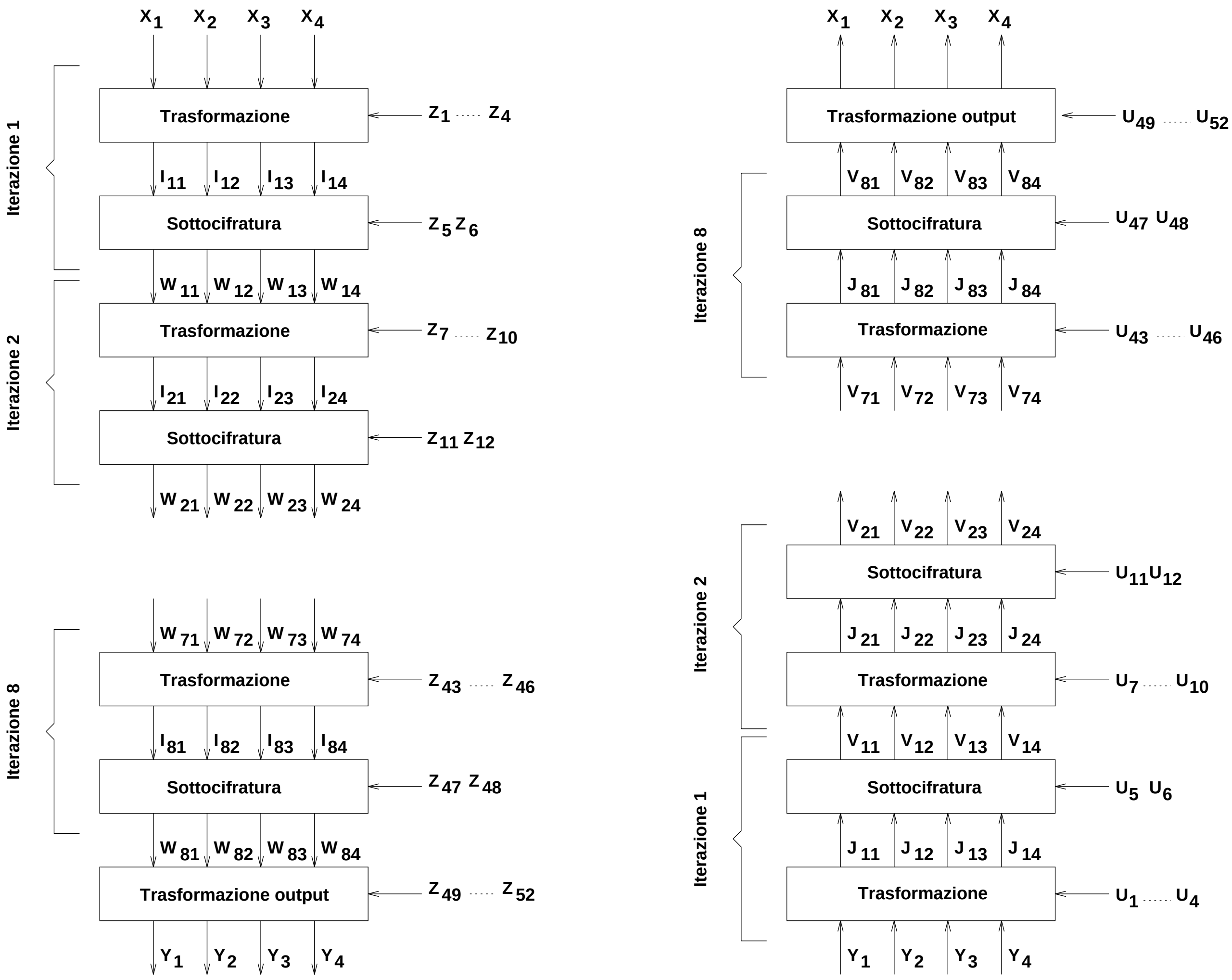

Figura 10: Cifratura e decifratura IDEA.

Ciascuna delle otto iterazioni è ulteriormente suddivisa in “trasformazione” e “sottocifratura”.

Nella cifratura otteniamo le seguenti relazioni riguardanti la trasformazione output:

$$\begin{aligned} Y_1 &= W_{81} \odot Z_{49} \\ Y_2 &= W_{83} \boxplus Z_{50} \\ Y_3 &= W_{82} \boxplus Z_{51} \\ Y_4 &= W_{84} \odot Z_{52} \end{aligned}$$

Nella decifratura il primo sottopasso della prima iterazione prende in input $Y_1 Y_2 Y_3 Y_4$ e calcola:

$$\begin{aligned} J_{11} &= Y_1 \odot U_1 \\ J_{12} &= Y_2 \boxplus U_2 \\ J_{13} &= Y_3 \boxplus U_3 \\ J_{14} &= Y_4 \odot U_4 \end{aligned}$$

Osserviamo che, sostituendo opportunamente i valori delle chiavi come indicato nella Tabella 3, otteniamo:

$$
\begin{array}{lcllllllllll}
J_{11} & = & Y_1 & \odot & Z_{49}^{-1} & = & W_{81} & \odot & Z_{49} & \odot & Z_{49}^{-1} & = & W_{81} \\
J_{12} & = & Y_2 & \boxplus & -Z_{50} & = & W_{83} & \boxplus & Z_{50} & \boxplus & -Z_{50} & = & W_{83} \\
J_{13} & = & Y_3 & \boxplus & -Z_{51} & = & W_{82} & \boxplus & Z_{51} & \boxplus & -Z_{51} & = & W_{82} \\
J_{14} & = & Y_4 & \odot & Z_{52}^{-1} & = & W_{84} & \odot & Z_{52} & \odot & Z_{52}^{-1} & = & W_{84}
\end{array}
$$

Quindi l'output del primo sottopasso nel processo di decifratura è uguale all'input dell'ultimo passo del processo di cifratura, eccetto un interscambio tra secondo e terzo blocco.

Resta da provare ora l'esattezza delle ultime due sottochiavi di decifratura in ciascuna iterazione.

Consideriamo le seguenti relazioni che si possono dedurre dalla Figura 6:

$$
\begin{array}{lclcl}
W_{81} & = & I_{81} & \oplus & MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{82} & = & I_{83} & \oplus & MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{83} & = & I_{82} & \oplus & MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
W_{84} & = & I_{84} & \oplus & MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})
\end{array}
$$

dove $MA_R(X,Y)$ è l'output destro del blocco MA e $MA_L(X,Y)$ è l'output sinistro.

Consideriamo l'ottavo blocco sottocifratura nella cifratura IDEA. Esso utilizza le chiavi Z_{47} e Z_{48}. Nel completare la prima iterazione di decifratura, usiamo questo blocco con tali chiavi dando in input J_{11}, J_{12}, J_{13}, J_{14}. È possibile verificare che avremo in output proprio I_{81}, I_{82}, I_{83}, I_{84}, cioè gli input della ottava iterazione nella fase di cifratura. Infatti

$$
\begin{array}{lcl}
V_{11} & = & J_{11} \oplus MA_R(J_{11} \oplus J_{13}, J_{12} \oplus J_{14}) \\
& = & W_{81} \oplus MA_R(W_{81} \oplus W_{82}, W_{83} \oplus W_{84}) \\
& = & I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus \\
& & MA_R[I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus I_{83} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}), \\
& & I_{82} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus I_{84} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})] \\
& = & I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
& = & I_{81}.
\end{array}
$$

Analogamente si ha:

$$
\begin{array}{lcl}
V_{12} & = & I_{83} \\
V_{13} & = & I_{82} \\
V_{14} & = & I_{84}.
\end{array}
$$

Usando la stessa tecnica ad ogni iterazione si ottiene

$$
\begin{array}{lcl}
V_{81} & = & I_{11} \\
V_{82} & = & I_{13} \\
V_{83} & = & I_{12} \\
V_{84} & = & I_{14}.
\end{array}
$$

Infine, poiché la trasformazione output del processo di decifratura è uguale al primo passo del processo di cifratura, a meno di un interscambio tra secondo e terzo blocco, l'output dell'intero processo di decifratura è uguale all'input del processo di cifratura.

4 RC5

RC5 è un cifrario a blocchi semplice e veloce, progettato da Ronald Rivest, del MIT Laboratory of Computer Science [11]. Tale algoritmo di cifratura, adatto per l'implementazione hardware e per quella software, prevede che la lunghezza della parola, il numero di iterazioni e la lunghezza della chiave privata siano parametri variabili.

In particolare RC5 ha le seguenti caratteristiche:

1. è un cifrario **simmetrico**, cioè sia nella cifratura che nella decifratura viene utilizzata la stessa chiave privata;

2. è **idoneo ad implementazioni sia hardware che software**: RC5 usa, infatti, solo operazioni elementari, eseguite dai comuni microprocessori;

3. è **veloce**, e quindi *word-oriented*: le operazioni sono eseguite su blocchi di dimensioni corrispondenti alla parola macchina;

4. è **adattabile a processori con differenti lunghezze di parola**: se, ad esempio, abbiamo a disposizione un processore a 64 bit, RC5 è in grado di sfruttare l'intera lunghezza della parola;

5. ha una **struttura iterativa** con un numero variabile di iterazioni;

6. ha una **chiave crittografica a lunghezza variabile**: ciò permette all'utente di scegliere il livello di sicurezza desiderato per la propria applicazione e risolve i problemi relativi alla esportazione di algoritmi crittografici che usano chiavi lunghe più di 40 bit;

7. è **facile da implementare**, cosicché la sua forza crittografica può essere rapidamente determinata;

8. ha una **richiesta di memoria limitata**, pertanto può essere facilmente implementato su smart card o altri dispositivi con memoria piccola;

9. fornisce una **elevata sicurezza** quando vengono scelti i valori dei parametri in modo opportuno;

10. utilizza **rotazioni data-dependent**, in cui le parole intermedie sono ruotate ciclicamente di un certo numero di posizioni determinato in base ai bit di ordine basso di un altro risultato intermedio.

Va detto, comunque, che l'algoritmo di cifratura RC5 è stato proposto solo recentemente e la sua forza crittografica non è stata ancora completamente determinata.

4.1 Parametrizzazione

Una delle caratteristiche fondamentali di RC5 è l'uso di tre parametri, **w**, **r** e **b**, al variare dei quali si ottiene una famiglia di algoritmi di cifratura. Naturalmente due utenti che si scambiano messaggi utilizzando RC5 per la cifratura devono accordarsi su **w**, **r** e **b**.

Esaminiamo più in dettaglio tali parametri.

- **w**: rappresenta la lunghezza della parola, in bit. Ogni parola contiene, dunque, $\frac{w}{8}$ byte.

  RC5 è ben definito per ogni **w**> 0, sebbene siano preferibili i valori 16, 32 e 64.

  Osserviamo che RC5 cifra blocchi di 2 parole, per cui il testo in chiaro e il testo cifrato sono suddivisi in blocchi lunghi $2w$ bit. Se, ad esempio, **w**=32, i blocchi di testo in chiaro e testo cifrato sono di 64 bit.

- **r**: rappresenta il numero di iterazioni dell'algoritmo e varia da 0 a 255. Il livello di sicurezza del cifrario dipende notevolmente da questo parametro: più grande scegliamo **r**, maggiore è la sicurezza.

- **b**: rappresenta la lunghezza della chiave privata K. I valori permessi per **b** sono 0,1,...,255.

Per comodità, fissati **w**, **r** e **b**, indichiamo il corrispondente algoritmo RC5 con RC5-**w**/**r**/**b**. Ad esempio l'algoritmo RC5-32/16/10 usa parole di 32 bit, effettua 16 iterazioni dell'algoritmo ed ha una chiave di 80 bit (10 byte).

Non necessariamente i parametri devono essere tutti specificati. All'occorrenza può essere omesso l'ultimo parametro, oppure possono essere omessi gli ultimi due. Con RC5-32, ad esempio, viene indicato l'algoritmo RC5 che usa parole di 32 bit, senza alcun riferimento al numero di iterazioni e alla lunghezza della chiave.

4.2 Notazioni e operazioni primitive

RC5 usa solo le seguenti tre operazioni elementari (e le loro inverse):

1. Addizione di parole modulo 2^w, eseguita senza considerare il riporto. Tale operazione è denotata con "+". L'operazione inversa, la sottrazione, è denotata con "$-$".

2. OR-esclusivo bit a bit, denotato con $\oplus$.

3. Rotazione a sinistra di parole. La rotazione ciclica della parola x di y bit a sinistra è denotata con $x \lll y$. y è considerato modulo **w**, cosicché, quando **w** è una potenza di due, sono necessari solo $\log w$ bit di ordine basso di y per determinare il numero di posizioni da ruotare. L'operazione inversa, la rotazione a destra, è denotata con $\ggg$.

Queste operazioni sono supportate in modo efficiente da molti processori.

La principale caratteristica di RC5 è l'uso di rotazioni *data-dependent*, cioè di rotazioni non fisse, ma dipendenti dal testo in chiaro e dalla chiave. Osserviamo che sui moderni microprocessori una rotazione del tipo $x \lll y$ impiega un tempo costante, indipendente dal valore y. Osserviamo, inoltre, che le rotazioni sono gli unici operatori non lineari in RC5.

4.3 Algoritmo

Il cifrario RC5 contiene le tre procedure seguenti:

- algoritmo di espansione della chiave;

- algoritmo di cifratura;

- algoritmo di decifratura.

Ricordiamo che il testo in chiaro in input a RC5 consiste di due parole a **w** bit, contenute nei registri A e B. RC5, inoltre, usa una "tabella della chiave espansa" $S[0 \ldots t-1]$ contenente $t = 2(r+1)$ parole a **w** bit, ottenute a partire dalla chiave privata K dell'utente.

A seconda dell'architettura della macchina, i dati espressi in byte possono essere memorizzati nei registri in uno dei seguenti modi:

1. **Architettura little-endian**: il primo byte occupa le posizioni dei bit di ordine basso del registro, il secondo byte occupa le posizioni immediatamente successive e così via, fino ad occupare le posizioni dei bit di ordine più alto. Questa architettura è tipica della famiglia INTEL.
2. **Architettura big-endian**: il primo byte occupa le posizioni dei bit di ordine alto del registro, il secondo byte occupa le posizioni immediatamente precedenti e così via, fino ad occupare le posizioni dei bit di ordine più basso. Questa architettura è tipica delle macchine SPARC.

RC5 utilizza l'architettura *little-endian*.

4.3.1 Espansione della chiave

L'algoritmo di espansione della chiave, adoperando la chiave segreta K, riempie la tabella delle chiavi S con $t = 2(r+1)$ parole binarie. Usa, inoltre, due "costanti magiche" P_w e Q_w, che sono così definite per un **w** arbitrario:

$$\begin{aligned} P_w &= \mathrm{Odd}((e-2)2^w) \\ Q_w &= \mathrm{Odd}((\phi-1)2^w) \end{aligned}$$

dove

$$\begin{aligned} e &= 2.718281828459... \text{ (base del logaritmo naturale)} \\ \phi &= 1.618033988749... \text{ (funzione aurea)} \end{aligned}$$

e $Odd(x)$ è l'intero dispari più vicino a x.
Per **w**=16, 32, 64 queste costanti hanno i seguenti valori in binario ed esadecimale:

$P_{16} = 1011011111100001 = \mathrm{b7e1}$
$Q_{16} = 1001111000110111 = \mathrm{9e37}$

$P_{32} = 10110111111000010101000101100011 = \mathrm{b7e15163}$
$Q_{32} = 10011110001101110111100110111001 = \mathrm{9e3779b9}$

$P_{64} = 1011011111100001010100010110001010001010111011010010101001101011$
$\quad = \mathrm{b7e151628aed2a6b}$
$Q_{64} = 1001111000110111011110011011100101111111010010100111110000010101$

$= 9e3779b97f4a7c15$

L'algoritmo di espansione della chiave si divide in 2 passi:

Conversione della chiave segreta da byte in parole. La chiave segreta $K[0...b-1]$ di **b** byte viene copiata in un array $L[0...c-1]$ di c parole, dove $c = \lceil \frac{8 \cdot b}{w} \rceil$ è il numero di parole nella chiave.

Il numero di byte per parola è $u = \frac{w}{8}$. Quindi ciascuna parola dell'array L è riempita con u byte di K, utilizzando la tecnica propria dell'architettura *little-endian*.

Il seguente pseudocodice produce lo stesso effetto, assumendo che tutti i byte siano unsigned e che l'array L sia inizialmente vuoto:

for $i = b-1$ **downto** 0 **do**
  $L[i/u] = (L[i/u] \lll 8) + K[i];$

Creazione della tabella della chiave espansa. L'array S è inizializzato usando una progressione aritmetica modulo 2^w determinata dalle costanti magiche P_w e Q_w:

$S[0] = P_w;$
for $i = 1$ **to** $t-1$ **do**
  $S[i] = S[i-1] + Q_w;$

Quindi l'algoritmo effettua la vera e propria espansione della chiave elaborando gli array S e L nel modo seguente:

$i = j = 0;$
$A = B = 0;$
do $3 * \max(t, c)$ **times:**
  $A = S[i] = (S[i] + A + B) \lll 3;$
  $B = L[j] = (L[j] + A + B) \lll (A + B);$
  $i = (i + 1) \bmod(t);$
  $j = (j + 1) \bmod(c);$

La funzione di espansione della chiave può essere considerata *one-way*: non è facile determinare K da S.

4.3.2 Cifratura

Ricordiamo che il blocco in input è contenuto in due registri di **w** bit A e B. L'algoritmo, in pseudo-codice, è il seguente:

$A = A + S[0];$
$B = B + S[1];$
for $i = 1$ **to** r **do**
  $A = ((A \oplus B) \lll B) + S[2 * i];$
  $B = ((B \oplus A) \lll A) + S[2 * i + 1];$

L'output è contenuto nei registri A e B.

Osserviamo che RC5, ad ogni iterazione, aggiorna entrambi i registri A e B. Durante ciascuna iterazione del DES, invece, solo metà parola viene realmente trasformata, mentre l'altra metà rimane inalterata. Pertanto una iterazione di RC5 è "equivalente" a due iterazioni del DES.

4.3.3 Decifratura

La routine di decifratura può essere facilmente derivata dalla routine di cifratura: basta eseguire le operazioni inverse nell'ordine inverso.

for $i = r$ **downto** 1 **do**
  $B = ((B - S[2 * i + 1]) \ggg A) \oplus A;$
  $A = ((A - S[2 * i]) \ggg B) \oplus B;$
$B = B - S[1];$
$A = A - S[0];$

4.4 Considerazioni sull'algoritmo

L'algoritmo RC5 è estremamente semplice. La sua forza crittografica dipende pesantemente dall'uso delle rotazioni *data-dependent*. Poiché esse dipendono dai dati in input, non permettono di collezionare statistiche ed evitano attacchi di crittoanalisi differenziale e di crittoanalisi lineare.

Ricordiamo, inoltre, che una parte della tabella S è inizialmente sommata al testo in chiaro, e ogni iterazione termina aggiungendo un elemento di S a risultati intermedi già calcolati. Ciò assicura che ciascuna iterazione agisca, in termini di rotazioni usate, in maniera differente.

Anche l'operazione $\oplus$, eseguita avanti e indietro tra A e B, è rilevante. Infatti il cambio di un singolo bit in un blocco input produce la variazione di bit multipli nelle iterazioni successive.

4.5 Considerazioni sui parametri

Non è detto che l'algoritmo RC5 sia sicuro per tutti i possibili valori dei parametri: per **r**=0, ad esempio, non viene fatta alcuna cifratura, e per **r**=1 il cifrario può essere rotto facilmente; analogamente **b**=0 non dà nessuna sicurezza.

D'altra parte assegnare ai parametri il massimo valore consentito appesantirebbe la maggior parte delle applicazioni.

Supponiamo di voler rimpiazzare il DES con un algoritmo RC5 "equivalente". Una scelta ragionevole potrebbe essere RC5-32/16/7: in questo caso la lunghezza del blocco, il numero di iterazioni e la lunghezza della chiave sono esattamente gli stessi.

A differenza del DES, che non offre nessuna parametrizzazione e quindi nessuna flessibilità, RC5 può essere migliorato, se necessario. Ad esempio, un miglioramento dell'algoritmo per il rimpiazzamento del DES può essere ottenuto utilizzando una chiave a 80 bit, e cioè RC5-32/16/10.

La scelta di **r** influenza sia la velocità che la sicurezza della cifratura. Per alcune applicazioni l'alta velocità potrebbe essere la richiesta più importante: si potrebbe richiedere, cioè, la massima sicurezza ottenibile entro un tempo stabilito. La scelta di un valore piccolo di **r** (ad esempio **r**=6), può fornire una certa sicurezza, anche se modesta, rispettando i vincoli di velocità prefissati.

In altre applicazioni, come la gestione delle chiavi, la sicurezza rappresenta, invece, l'interesse primario, mentre la velocità è relativamente importante. La scelta di **r**=32 potrebbe essere appropriata per tali applicazioni.

Riportiamo di seguito i risultati di due esperimenti che hanno dimostrato come il numero di iterazioni influenzi le proprietà di RC5.

Il primo test ha esaminato l'uniformità di correlazione tra i bit dell'input e dell'output. Si è visto che quattro iterazioni sono sufficienti per ottenere tale correlazione in RC5-32.

Il secondo test ha esaminato la dipendenza tra rotazioni *data-dependent* e i bit del testo in chiaro. Si è ottenuto come risultato che 8 iterazioni in RC5-32 sono sufficienti perché ogni bit del messsaggio influenzi le rotazioni.

In pratica, il numero di iterazioni deve essere grande almeno quanto suggerito da questi semplici test. I valori proposti sono **r**=12 per RC5-32 e **r**=16 per RC5-64.

Anche la misura **w** della parola influenza velocità e sicurezza del cifrario: la scelta di un valore di **w** più grande della misura del registro della CPU può compromettere la velocità della cifratura. Se, invece, RC5 deve essere usato come base per una funzione hash, può essere conveniente scegliere **w**=64 (o anche di più), allo scopo di avere blocchi input/output a 128 bit (o più).

Un algoritmo di cifratura come RC5 che permette la scelta di parametri può essere considerato rischioso. A questo proposito osserviamo che

- i parametri sono fissati a priori e non possono essere modificati nel corso dell'esecuzione;
- le implementazioni reali richiedono parametri opportuni, lasciando così all'utente una minima possibilità di scelta.

Infine osserviamo che RC5 può essere usato in alcune applicazioni che non richiedono alcuna sicurezza crittografica. Per esempio, si potrebbe usare RC5-32/8/0 (senza alcuna chiave) applicata agli input 0,1,2,... per generare una sequenza di numeri pseudo-random.

4.6 Implementazione

L'algoritmo di cifratura RC5 è molto compatto, e può essere codificato efficientemente in linguaggio assembler su più processori. Essendo la tabella S abbastanza piccola e ad accesso sequenziale, la richiesta di memoria cache è ridotta.

Segue un'implementazione di RC5-32/12/16 in C.

```
/* Copyright (C) 1995 RSA Data Security, Inc.                            */
#include <stdio.h>
#include <time.h>
typedef unsigned long int WORD; /* Deve essere di 32 bit = 4 byte          */
#define w        32             /* misura della parola in bit              */
#define r        12             /* numero di iterazioni                    */
#define b        16             /* numero di byte nella chiave             */
#define c         4             /* numero di parole nella chiave           */
#define t        26             /* misura della tabella S = 2*(r+1) parole */
WORD S[t];                      /* tabella della chiave espansa            */
WORD P = 0xb7e15163, Q = 0x9e3779b9;  /* costanti magiche                  */

#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
```

```
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

void CIFRATURA(WORD *pt, WORD *ct) /* pt e ct sono l'input e l'output      */
{ WORD i, A=pt[0]+S[0], B=pt[1]+S[1];
  for (i=1; i<=r; i++)
    { A = ROTL(A^B,B)+S[2*i];
      B = ROTL(B^A,A)+S[2*i+1];
    }
  ct[0] = A; ct[1] = B;
}

void DECIFRATURA(WORD *ct, WORD *pt) /* pt e ct sono l'input e l'output    */
{ WORD i, B=ct[1], A=ct[0];
  for (i=r; i>0; i--)
    { B = ROTR(B-S[2*i+1],A)^A;
      A = ROTR(A-S[2*i],B)^B;
    }
  pt[1] = B-S[1]; pt[0] = A-S[0];
}

void ESPANSIONE(unsigned char *K) /* chiave segreta in input K[0...b-1]    */
{  WORD i, j, k, u=w/8, A, B, L[c];
   /* Inizializzazione di L e costruzione della tabella S                  */
   for (i=b-1,L[c-1]=0; i!=-1; i--) L[i/u] = (L[i/u]<<8)+K[i];
   for (S[0]=P,i=1; i<t; i++) S[i] = S[i-1]+Q;
   for (A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c)   /* 3*t > 3*c        */
     { A = S[i] = ROTL(S[i]+(A+B),3);
       B = L[j] = ROTL(L[j]+(A+B),(A+B));
     }
}

void main()
{ WORD i, j, pt1[2], pt2[2], ct[2] = {0,0};
  unsigned char key[b];
  time_t t0, t1;
  if (sizeof(WORD)!=4)
  printf("Errore RC5: WORD ha %d byte.\n",sizeof(WORD));
  printf("Esempi RC5-32/12/16:\n");
  for (i=1;i<6;i++)
    { /* Inizializzazione di pt1 e key in modo pseudocasuale               */
      pt1[0]=ct[0]; pt1[1]=ct[1];
      for (j=0;j<b;j++) key[j] = ct[0]%(255-j);
      /* Espansione, cifratura e decifratura                               */
      ESPANSIONE(key);
      CIFRATURA(pt1,ct);
      DECIFRATURA(ct,pt2);
      /* Stampa dei risultati e ricerca errori di decifratura              */
      printf("\n%d^ chiave = ",i);
      for (j=0; j<b; j++) printf("%.2X ",key[j]);
      printf("\nTesto in chiaro = %.8lX %.8lX --> Testo cifrato = %.8lX %.8lX \n",
```

```
        pt1[0], pt1[1], ct[0], ct[1]);
        if (pt1[0]!=pt2[0] || pt1[1]!=pt2[1]) printf("Errore di decifratura!\n");
    }
}
```

Questa implementazione cifra 100 Kbyte/sec su un laptop '486 a 50 Mhz (con un compilatore C++ Borland a 16 bit), e 2.4 Mbyte/sec su uno Sparc 5 (compilatore gcc). Questa velocità può essere certamente migliorata. In linguaggio assembler l'operazione di rotazione è accessibile direttamente: una routine in linguaggio assembler per il '486 esegue mezza iterazione con sole quattro istruzioni. Un'implementazione in linguaggio assembler cifra 1.2 Mbyte/sec su un SLC '486 a 50 Mhz. Un Pentium dovrebbe essere in grado di cifrare alcuni Mbyte/sec.

Riferimenti

[1] K. Campbell e M. Wiener, *Proof that DES is not a Group*, Proceedings di CRYPTO '92, Berlino: Springer-Verlag, 1992.

[2] W. Diffie e M. Hellman, *Exhaustive Cryptanalysis of the NBS Data Encryption Standard*, Computer, Giugno 1977.

[3] W. Tuchman, *Hellman Presents no Shortcut Solutions to DES*, IEEE Spectrum, Luglio 1979.

[4] D. Coppersmith, *The Data Encryption Standard (DES) and its Strenght Against Attacks*, Dicembre 1992.

[5] R. Merkle e M. Hellman, *On the Security of Multiple Encryption*, Communications of the ACM, Luglio 1981.

[6] P. Oorschot e M. Weiner, *A Known-Plaintext Attack on Two-Key Triple Encryption*, Proceedings di EUROCRYPT '90, Springer-Verlag, 1990.

[7] X. Lai e J. Massey, *A Proposal for a New Block Encryption Standard*, Proceedings di EUROCRYPT '90, Springer-Verlag, 1990.

[8] X. Lai e J. Massey, *Markov Ciphers and Differential Cryptanalysis*, Proceedings di EUROCRYPT '91, Springer-Verlag, 1991.

[9] X. Lai, *On the Design and Security of Block Ciphers*, Hartung-Gorre, Germania, 1992.

[10] I. Wegenerm, *The Complexity of Boolean Functions*, New York: Wiley, 1987

[11] R. L. Rivest, *The RC5 Encryption Algorithm*, Fast Software Encryption, B. Preneel Ed., vol. 1008 di Lecture Notes in Computer Science, Springer-Verlag, 1994.