4. CIFRARI A BLOCCHI

I cifrari simmetrici, ossia crittosistemi a chiave privata/segreta, sono suddivisi in due grandi famiglie:

- ***Cifrari a blocchi***, dove i messaggi vengono divisi in blocchi e poi cifrati
- ***Stream cipher***, dove i messaggi vengono cifrati carattere per carattere.

I **cifrari a blocchi** ricevono in input un testo in chiaro a n bit e la chiave, e restituiscono un testo cifrato. Alcuni suoi esempi sono: Data Encryption Standard (DES), DES triplo, Blowfish, RC5, RC6, Advanced Encryption Standard (AES), …

Il messaggio viene suddiviso in blocchi di n bit: si hanno 2^n possibili input per il testo in chiaro, 2^n possibili output per il testo cifrato, e $2^n!$ possibili trasformazioni (reversibili). La cifratura deve essere reversibile, quindi deve essere costituita da una funzione biiettiva. Il mapping è memorizzabile in una *tabella*: quest'ultima è la chiave del cifrario.

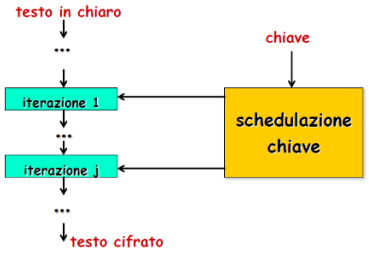
L'uso della tabella è poco pratico; per blocchi di n bit, la tabella ha dimensione $n \cdot 2^n$: per n = 64, la dimensione è $n \cdot 2^n = 2^{70} = 10^{21}$ bit.

Occorre un'approssimazione, cioè un metodo compatto per rappresentare la funzione di cifratura/decifratura.

Chiaro	Cifrato
0000	1110
0001	0100
0010	1101
0011	0001
0100	1111
0101	1011
0110	1011
0111	1000

Chiaro	Cifrato
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Tipicamente, un cifrario a blocchi ha la seguente struttura posta in basso: riceve in input il testo in chiaro, in base alla grandezza vengono effettuate j iterazioni, ed ognuna di esse riceve in input una sottochiave (opportunamente schedulata). In seguito alla j-esima iterazione, viene restituito in output il testo cifrato.

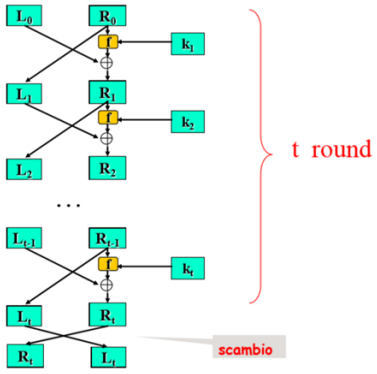


4.1 CIFRARI DI FEISTEL

Horst Feistel (1973) fu il primo a proporre un cifrario basato su una sequenza di permutazioni e sostituzioni, basandosi sui principi di Shannon (1949), di:

- ***diffusione***, ossia indipendenza della struttura statistica del testo in chiaro da quello cifrato;
- ***confusione***, relazione complicata fra testo cifrato e chiave segreta.

Tali principi rendono difficili gli attacchi statistici. In particolare, Feistel costruì un cifrario basandosi sulla struttura di cui a destra, dove L_0 ed R_0 sono rispettivamente parte sinistra e destra del messaggio, ***f*** è una funzione che riceve in input R_0 ed una sottochiave k_1 e restituisce un nuovo R_0, si procede come segue:



- l'input (L_0 ed R_0) viene suddiviso in parte sinistra (***left***) e destra (***right***), ognuna delle quali costituita da ***32 bit***;
- si utilizza la funzione ***f***, ed il nuovo R_0 va in input in uno ***XOR*** bit a bit con L_0;
- il risultato del bit a bit è R_1, che va in input al nuovo round con L_1, k_2 e la medesima funzione ***f***.

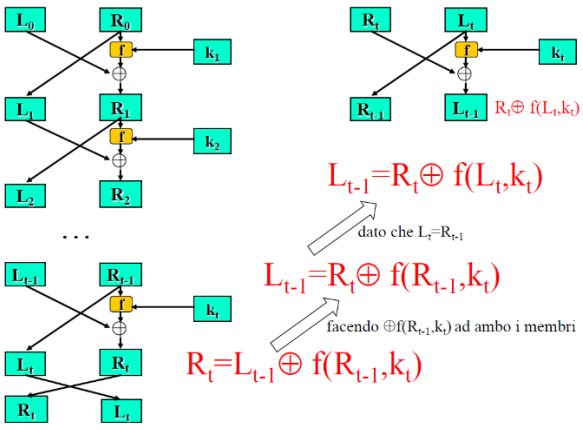
Si può espandere tale iterazione per ***t*** volte. Alla fine dell'ultimo round si effettua uno scambio: si inverte L_t con R_t.

In generale:

- blocchi grandi migliorano la sicurezza, ma riducono la velocità;
- chiavi grandi migliorano la sicurezza, ma riducono la velocità;
- tutti i round/iterazioni hanno la stessa struttura;
- è presente un *algoritmo di* ***scheduling della chiave***, in cui a partire dalla chiave iniziale vengono prodotte tante sottochiavi quanti sono i round.

Chiaramente, la decifratura coinvolge l'inverso dell'algoritmo precedentemente esposto. Tuttavia, sorge un dubbio: occorre invertire anche la funzione f? No, in quanto sapendo che $L_t = R_{t-1}$:

$$R_t = L_{t-1} \oplus f(R_{t-1}, k_t) \Rightarrow L_{t-1} = R_t \oplus f(R_{t-1}, k_t) \Rightarrow L_{t-1} = R_t \oplus f(L_t, k_t)$$

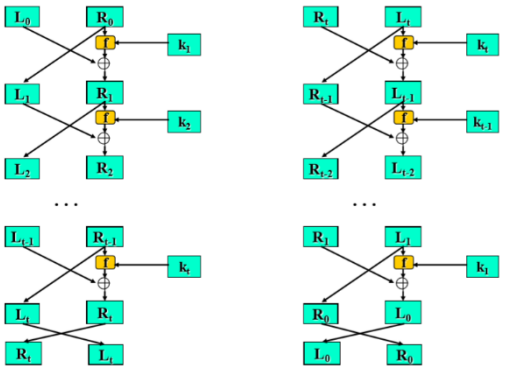


Il vantaggio dell'utilizzo dei cifrari di Feistel è proprio dovuto alla struttura, che resta invariata sia per la cifratura che per la decifratura: in queste due fasi, vengono invertite le sole sottochiavi.

Quindi, in sintesi, i cifrari di Feistel utilizzano l'algoritmo di cifratura/decifratura posto a destra:

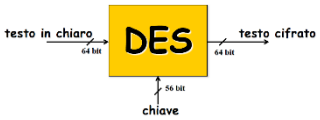
- per la ***cifratura***, basta implementare un solo round e lo stesso codice può essere usato per ogni round;
- per la ***decifratura***, usa lo stesso algoritmo per la cifratura ed usa soltanto le sottochiavi in ordine inverso.

Esempi di cifrari di Feistel sono DES e Blowfish.



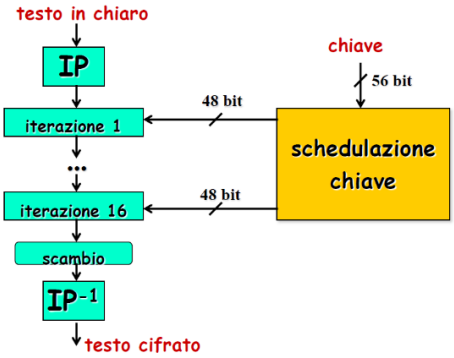
4.2 DATA ENCRYPTION STANDARD (DES)

Il **DES** (Data Encryption Standard) è un cifrario di Feistel che riceve in input 64 bit di testo in chiaro e, utilizzando una chiave di 56 bit, produce un testo cifrato di 64 bit.



Nello standard DES, la ***chiave*** è in realtà lunga 64 bit: 8 byte, di cui il primo e l'ultimo sono utilizzati per il calcolo del bit di parità (effettuando lo XOR dei primi 7 bit di ognuno).
Il DES, essendo un cifrario di Feistel, ne mantiene la ***struttura***, utilizzando aggiuntivamente:

- una **IP** (Initial Permutation) del testo in chiaro prima della prima iterazione, indipendente dalla chiave;
- una **IP^{-1}** (l'inverso della IP) dopo lo scambio;
- uno **scheduler** per la chiave, che riceve in input la chiave a 56 bit e restituisce ad ogni iterazione (esattamente 16) una sottochiave di 48 bit.



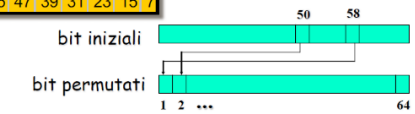
La ***permutazione iniziale*** è definita da una tabella: il primo valore è 58, significa che nella permutazione il primo bit è il 58° del testo in chiaro; il secondo bit è il 50° del testo in chiaro, il terzo sarà il 42°, …, ed il 64° bit è il settimo del testo in chiaro. Questa permutazione è fissa, e non è chiaro il motivo per cui è stata implementata: infatti, non cambia nulla dal punto di vista della sicurezza.

Nella ***permutazione inversa***, ad esempio, il 40° bit viene rimesso nella prima posizione del testo cifrato, l'8° bit va nella seconda posizione, e così via. In basso al centro è posta la tabella per IP^{-1}. Si comprenda che tali tabelle sono le medesime per ogni utilizzo dell'implementazione del DES.

La ***singola iterazione*** (***round***) ha la medesima struttura del cifrario di Feistel.

PERMUTAZIONE INIZIALE IP

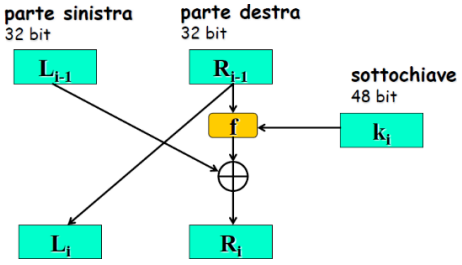
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7



PERMUTAZIONE INVERSA IP^{-1}

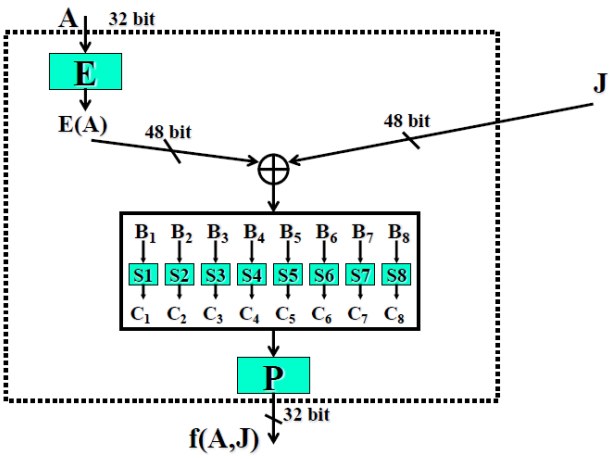
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

SINGOLA ITERAZIONE



La ***funzione f*** prende in input i 32 bit della ***parte destra del messaggio (A),*** la ***sottochiave*** di ***48 (J) bit*** e dà in ***output 32 bit***, funziona come segue:

- i 32 bit della parte dx vengono ***espansi*** in 48 bit da una ***funzione di espansione E***;
- i bit espansi ***E(A)*** vanno in input ad uno ***XOR*** con la ***sottochiave J***;
- l'output dello ***XOR*** va in input ad 8 "*scatolette*", dette ***s-box*** (***substitution box***), ognuna delle quali prende in input 6 bit, sostituendoli con altri 6;
- effettua una ***permutazione*** dei 48 bit, restituendone 32.



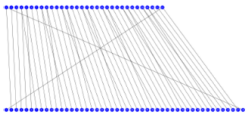
Anche la ***funzione di espansione E*** è uno standard, dove i 16 bit aggiuntivi sono duplicati: nel primo bit andrà il 32° bit, nel secondo ci andrà l'1, e così via. Così come la permutazione iniziale, anche l'espansione utilizza una tabella fissata che fa parte dello standard. La tabella per la funzione di espansione E, implementata dal DES, è posta a sinistra.

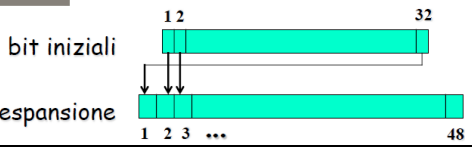
Anche la ***permutazione P*** della funzione f utilizza una tabella come le precedenti, posta a destra.

FUNZIONE DI ESPANSIONE (E)

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

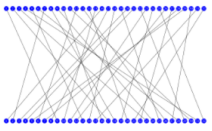
16 bit sono duplicati

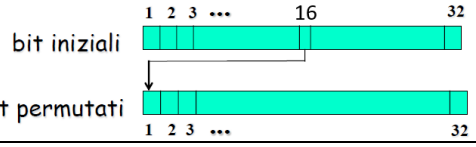




PERMUTAZIONE (P)

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

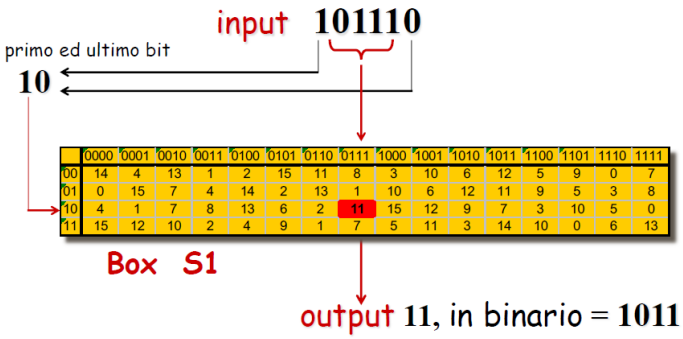




La singola ***S-box*** è descritta da una tabella di quattro righe (00, 01, 10, 11) e sedici colonne (0000, …, 1111), e per ogni cella della tabella c'è un valore diverso, fissato dallo standard. In questa S-box, dati 6 bit, occorre dare in output 4 bit: l'idea è che i 6 bit in input (es. 101110) si dividono in due parti:

- la prima parte, costituito da primo e ultimo bit (es. 10), e la seconda parte, costituita dai 4 bit centrali (es. 0111);
- si effettua il mapping prima parte – seconda parte nella tabella, e si produce l'output della cella corrispondente.

Ogni S-box ha una diversa tabella, definita dallo standard. Nell'esempio scritto tra parentesi, l'output corrispondente alla stringa 101110 sarebbe 11, cioè 0101.



In tutto l'algoritmo, tutti i singoli passi possono essere scritti come funzioni lineari dei valori precedenti. In crittografia, tutto ciò che è lineare è sempre stato rotto da un lavoro di crittoanalisi.

Il DES, invece, usa l'S-box come unico elemento non lineare, per cui il DES non può essere rotto con tecniche normali dell'algebra lineare. Se le S-box fossero state lineari, allora l'intero DES sarebbe stato lineare e l'output sarebbe stato ottenibile come una combinazione lineare dell'input, secondo una formula del tipo di cui a destra. Di conseguenza, se denotiamo con m_1, m_2, m_3 tre messaggi da crittare, allora:

$$DES_k(m) = B \begin{bmatrix} m \\ k_1 \\ k_2 \\ \dots \\ k_{16} \end{bmatrix} = C \pmod 2$$

$$DES_k(m_1) \oplus DES_k(m_2) \oplus DES_k(m_3) = DES_k(m_1 \oplus m_2 \oplus m_3)$$

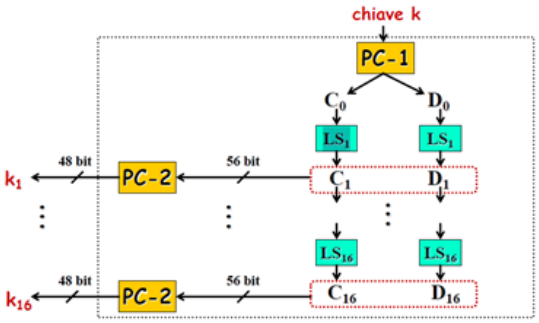
Proprietà delle S-box:

Il dipartimento del commercio americano pubblicò i criteri di funzionamento delle S-box, che sembrano più delle proprietà interessanti dal punto di vista matematico. Ogni riga è una permutazione degli interi 0, …, 15. Cambiando un solo bit di input ad una S-box variano almeno due bit nell'output. Per ogni S-box, il numero degli input per i quali il bit di output è 0 è circa uguale al numero degli input per i quali tale bit è 1.

Schedulazione delle chiavi:

La ***schedulazione delle chiavi*** avviene mediante uno ***scheduler*** che riceve in input la ***chiave k*** e, prima delle iterazioni effettua una prima permutazione-compressione (***PC-1***) della chiave, che passerà da 64 a 56 bit (vengono rimossi i ***bit di parità***, cioè il LSB di ogni byte, e permutati secondo una tabella), e la suddivide in C_0 e D_0 i quali, ad ogni iterazione:

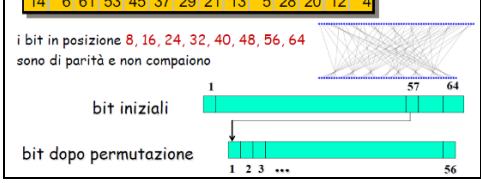
- verranno ***shiftati a sinistra*** secondo una tabella (in totale 28 posizioni in 16 iterazioni);
- verranno combinati nuovamente e dati in input ad un ***PC-2***, che rimuoverà altri 8 bit e effettuerà una nuova ***permutazione***.



PERMUTAZIONE 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

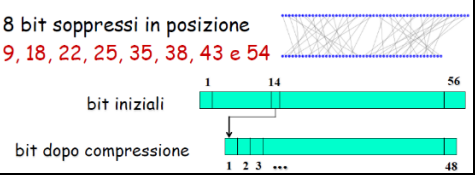
i bit in posizione 8, 16, 24, 32, 40, 48, 56, 64 sono di parità e non compaiono



COMPRESSIONE-PERMUTAZIONE 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	26	41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

8 bit soppressi in posizione 9, 18, 22, 25, 35, 38, 43 e 54



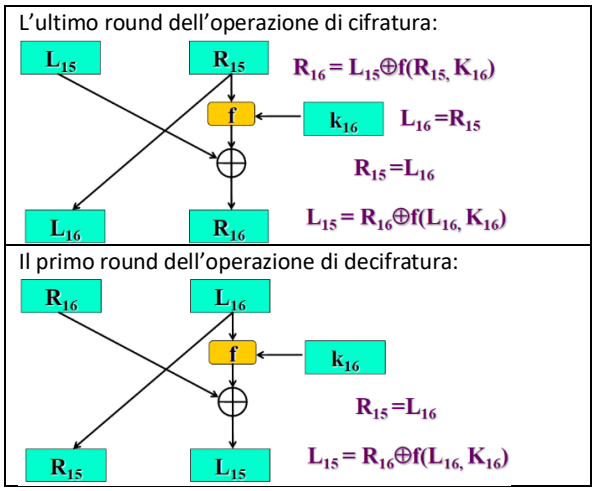
SCHEDULAZIONE SHIFT A SINISTRA

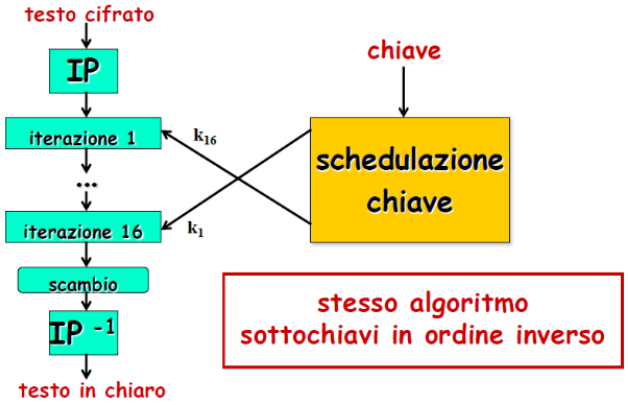
iterazione	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Totale shift nelle 16 iterazioni = 28 posizioni

Decifratura:

Essendo un cifrario di ***Feistel***, la decifratura avviene nel modo generico di tali cifrari. Relativamente alla sicurezza, permutazione iniziale e finale non influiscono affatto.





Avalanche Effect:

L'**avalanche effect** dice che un piccolo cambiamento del testo in chiaro oppure della chiave produce un grande cambiamento del testo cifrato: DES mostra un forte avalanche effect.

L'unica ***debolezza*** del DES è la lunghezza della chiave, troppo piccola per esser definita sicura.
La macchina parallela, fu chiamata ***Deep Crack***; essa funziona come segue:

- si dà in input ad un DES^{-1} la chiave ed un blocco;
- se il blocco è interessante, cioè se è costituito da caratteri stampabili, allora la invia ad un software centrale;
- altrimenti, si incrementa la chiave e la si invia nuovamente in input al DES^{-1}.

L'unità di ricerca di tale macchina era costituita da un clock di 40MHz, poteva decifrare in 16 cicli di clock: di conseguenza, poteva provare 40000000/16 = 2500000 chiavi al secondo; con 24 unità di ricerca, si potevano provare 24*2500000 chiavi al secondo, per cui si provavano tutte le chiavi in 13900 giorni (circa 38 anni).
Parallelizzando con 64 processori, si sarebbero provate 64*3840000000 chiavi al secondo (circa 218 giorni).
Inserendo 12 schede in vari chassis, si sarebbero provate 12*3840000000 chiavi al secondo, per un totale di circa 18 giorni. A causa di problemi di calore, per separare le schede i vari chassis sono stati posti in vari scatoloni.

Ci sono stati anche attacchi più sofisticati al DES, che dipendono dalla struttura del cifrario.

4.3 CRITTOANALISI DIFFERENZIALE E LINEARE

La ***crittoanalisi differenziale*** (Eli Biham e Adi Shamir, 1990) analizza come le differenze in input diventano differenze in output. La differenza, in genere, è lo XOR. Venne effettuata un'analisi dei differenziali per ogni S-box (ΔX, ΔY), dove $\Delta Y = S(X \oplus \Delta X) \oplus S(X)$. Questa crittoanalisi è efficace con un numero basso di iterazioni; essendo attacchi di tipo ***chosen ciphertext attack***:

- con 8 iterazioni, sono necessarie 2^{14} coppie (plaintext, ciphertext) di testi scelti;
- con 16 iterazioni, sono necessarie 2^{47} coppie (plaintext, ciphertext) di testi scelti.

Il DES è resistente alla crittoanalisi differenziale, in quanto fu previsto da Don Coppersmith durante la progettazione di tale cifrario. Tuttavia, con piccole modifiche il DES sarebbe stato vulnerabile anche a questi attacchi.
La ***crittoanalisi lineare*** (Mitsuru Matsui, 1993) è una evoluzione della precedente, che ricerca approssimazioni affini (anziché lo XOR). per il cifrario: recupera la chiave a partire da 2^{47} coppie (plaintext, ciphertext) di testi noti.

4.4 MODALITÀ OPERATIVE DEL DES

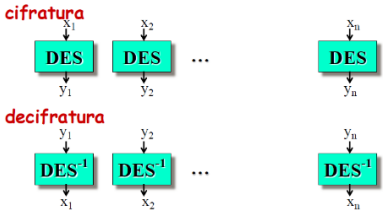
Il ***modulo del DES***, nella sua implementazione, consente di cifrare al più 64 bit. Le seguenti ***modalità operative*** consentono di cifrare più di 64 bit sfruttando il DES. Vedremo le seguenti:

- Electronic Codebook Chaining (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- Output FeedBack (OFB)
- CounTeR (CTR)

Queste modalità sono tutte standardizzate, alcune delle quali già nel 1981.

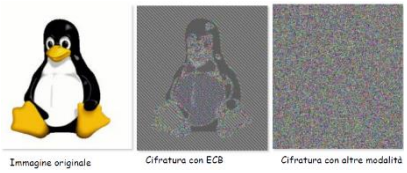
4.4.1 ELECTRONIC CODEBOOK CHAINING (ECB)

L'***Electronic Codebook Chaining*** (***ECB***) partiziona il messaggio in chiaro $x = x_1x_2...x_n$ in n blocchi di 64 bit. Una volta fatto ciò, si cifra ogni singolo blocco con la stessa chiave. La decifratura avviene esattamente invertendo il meccanismo di cifratura. Si può parallelizzare sia la cifratura che la decifratura. Se la lunghezza del messaggio non è un multiplo di 64 bit, si può estendere il messaggio con un ***padding*** con 100...000: è più sicuro utilizzare questo padding anziché estendere il messaggio con tutti 0, per meglio definire dove termina il messaggio e dove inizia il padding.



L'ECB è il metodo più veloce, ed eventuali errori non si propagano (essendo parallelizzabile). Tuttavia, non c'è dipendenza tra i blocchi ed è possibile attaccare il cifrario mediante sostituzione di tali blocchi, in modo tale da rendere il testo decifrato (spesso) indecifrabile. Non è sicuro, inoltre, per messaggi lunghi, in quanto allo stesso blocco in chiaro corrisponde sempre lo stesso blocco cifrato.

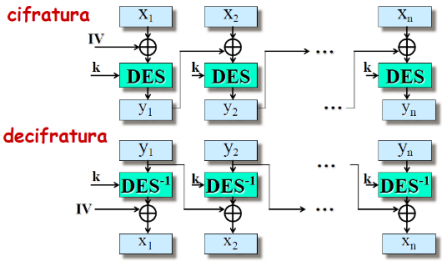
Ad esempio, cifrando i colori di una immagine bitmap con grandi aree di colori uniformi, succede che ogni area dello stesso colore viene cifrata utilizzando lo stesso colore differente, di conseguenza, la struttura dell'immagine potrebbe risultare ancora visibile.



4.4.2 CIPHER BLOCK CHAINING (CBC)

Il ***Cipher Block Chaining*** (***CBC***) partiziona il messaggio in chiaro $x = x_1x_2...x_n$ in n blocchi di 64 bit. Una volta fatto ciò, si cifra ogni singolo blocco con la stessa chiave. Si differenzia dall'ECB con lo ***XOR*** (che avviene prima di utilizzare il DES), dal secondo blocco in poi, tra 64 bit (blocchi $x_2, x_3, ..., x_n$) del testo in chiaro con i 64 bit del testo cifrato del blocco precedente (blocchi $y_1, y_2, ..., y_{n-1}$). Lo XOR del primo blocco avviene, invece, tra i 64 bit del blocco x_1 ed un ***vettore di inizializzazione IV*** (64 bit), solitamente pubblico.

La ***decifratura*** inverte la sola operazione di ***XOR***, inserendola dopo l'utilizzo del modulo DES invertito anziché prima. Inoltre, i blocchi successivi al primo ricevono subito il blocco cifrato precedente (prima che vada in input al DES): questo consente una parallelizzazione della decifratura.



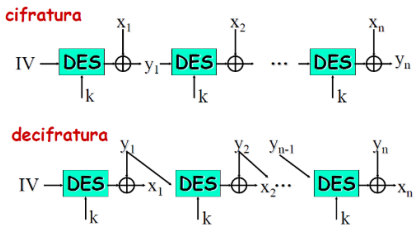
La ***cifratura***, invece, non è parallelizzabile. Decifrando con un IV errato, quindi, si ottiene un errore solo nel primo blocco. L'utilizzo dell'IV consente di ignorare completamente il primo blocco in caso di errori: questo perché esso è una propagazione dei 64 bit iniziali del messaggio in chiaro.
Il CBC è meno veloce dell'ECB, ed è possibile che gli errori vengano propagati. I vantaggi riguardano, invece, la dipendenza tra i blocchi, quindi gli attacchi di sostituzione sono ininfluenti.

4.4.3 CIPHER FEEDBACK (CFB)

Il ***Cipher FeedBack*** (***CFB***) partiziona il messaggio in chiaro $x = x_1x_2...x_n$ in n blocchi di 64 bit. Una volta fatto ciò, si cifra ogni singolo blocco con la stessa chiave, sequenzialmente. La differenza tra questa modalità e la precedente risiede nel fatto che il CFB effettua prima il DES e poi lo ***XOR***.

Nella ***decifratura*** non si usa il DES invertito, in quanto il risultato del DES che va in input allo XOR viene calcolato allo stesso modo di come è stato calcolato nella cifratura.

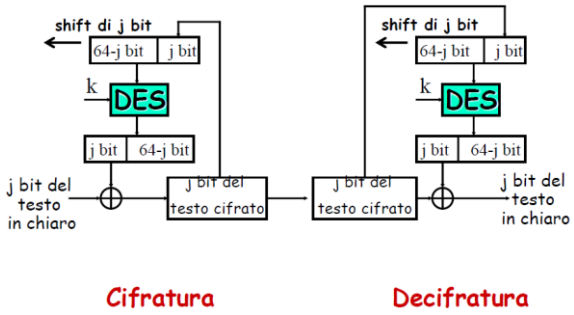
La ***cifratura*** non è parallelizzabile, mentre la decifratura lo è. Decifrando con un IV errato, avremo il solo primo blocco errato



4.4.4 J-BIT CFB

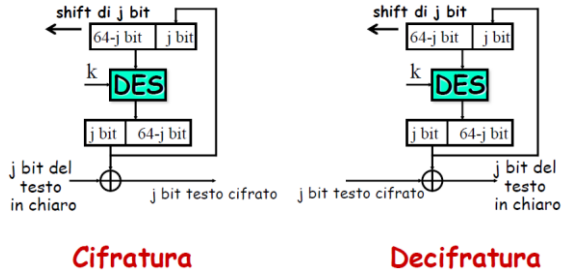
Il ***j-bit CFB*** cifra j bit per volta utilizzando il CFB. Si parte inizialmente con un ***IV*** in input al DES da cifrare con una ***chiave k.*** Ottenuto il testo cifrato, i primi j bit servono per la cifratura del testo in chiaro (ottenuta mediante ***XOR*** con j bit del testo in chiaro), mentre i 64-j bit vengono scartati. Il risultato dello XOR (i j bit del testo cifrato), oltre a darlo come output, viene riutilizzato nello schema di cifratura:
l'***IV*** viene shiftato di j bit a sinistra, quindi i primi j bit vengono persi e gli ultimi j bit vengono sostituiti dai j bit cifrati in precedenza.

La ***decifratura*** inverte semplicemente lo XOR. Il CFB è uguale al 64-bit CFB. "j" può essere scelto a piacimento, ma questa modalità è operativa è più lenta al decrescere di j. Un fattore positivo riguarda il poter utilizzare j bit cifrati senza aspettarne i 64 del blocco.



4.4.5 OUTPUT FEEDBACK (OFB)

L'***Output FeedBack*** (***OFB***, generalizzato nella versione *j-bit*) è simile alla modalità precedente: l'unica differenza risiede nel fatto che, anziché riutilizzare i j bit del testo cifrato, si riutilizzano i j bit dell'output del DES (prima che vadano in input allo XOR).
Anche in questo caso, nella ***decifratura*** viene invertito il solo ***XOR,*** il resto è esattamente uguale alla cifratura: questo è un aspetto molto importante.

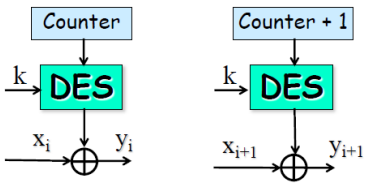


La modalità operativa dell'OFB è simile al cifrario di ***Vigenère*** o al cifrario perfetto. Un altro aspetto importante è il seguente: se l'output del processo di cifratura-shift-cifratura-… non dipende né dal testo in chiaro né dal testo cifrato, significa che i valori in input allo XOR possono essere precomputati, per cui questa modalità operativa può essere ottimizzata in termini di velocità. Tuttavia, non c'è dipendenza tra i blocchi: cambiando un bit del testo in chiaro, cambia un solo bit nel testo cifrato. L'OFB è uguale al 64-bit OFB.

4.4.6 COUNTER (CTR)

Il ***CounTeR*** (***CTR***) impiega un contatore di 64 bit, anche in questo caso si partiziona il messaggio in chiaro $x = x_1x_2…x_n$ in n blocchi di 64 bit.
Il ***contatore*** va in input al DES con la ***chiave k***, e l'output del DES va in input ad uno ***XOR*** con il blocco x_i del testo in chiaro. Il blocco seguente x_{i+1} viene cifrato allo stesso modo del precedente, in aggiunta incrementando il contatore di 1.

Per la ***decifratura***, viene invertito il solo XOR: deve essere, però, comunicato il valore iniziale del ***counter*** a chi riceve il messaggio. Bisogna evitare il riuso della stessa chiave e dei valori *Counter, Counter+1, Counter+2, …*, già utilizzati: si otterrebbero correlazioni tra i relativi blocchi di testo in chiaro.



I vantaggi sono i seguenti: efficienza hardware e software, preelaborazione (in quanto l'output del DES può essere computato offline), accesso casuale (è parallelizzabile, dato che il blocco del testo cifrato y_i non va in input alla cifratura del blocco successivo x_{i+1}), sicurezza dimostrabile (se il cifrario è sicuro) e semplicità.

Nelle varie metodologie che abbiamo visto fino ad ora, se il testo non ha una lunghezza opportuna, si esegue il padding del testo in chiaro per poterne effettuare la cifratura.

Alcune volte si vuole che la lunghezza del testo in chiaro sia uguale al testo cifrato ad esempio in un database, se si mettono entrate di una lunghezza, si vuole che i corrispettivi cifrati siano della stessa lunghezza. Tutto ciò trova soluzione nel **Ciphertext Stealing.**

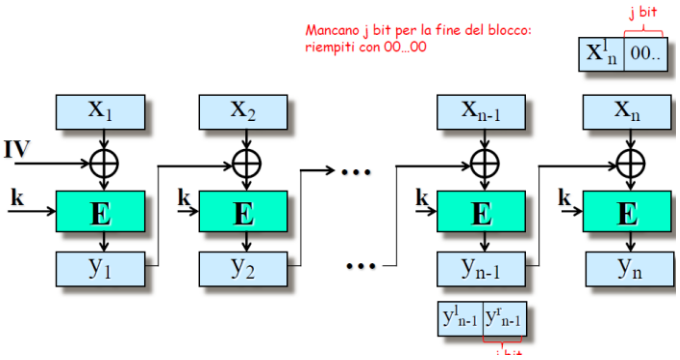
4.5 CIPHERTEXT STEALING

Metodo per evitare una espansione del testo cifrato quando la sua lunghezza non è multipla del blocco del cifrario. Questa tecnica è applicata al Cipher Block Chaining (CBC) che è stato già analizzato in precedenza, da cui otteniamo:

4.5.1 CIPHER BLOCK CHAINING (CBC) CON CIPHERTEXT STEALING CS1

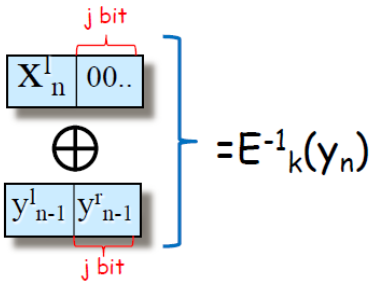
Si supponga di dare in input al cifrario un testo in chiaro e si supponga che manchino j bit per la fine del blocco, di conseguenza la lunghezza in bit del testo in chiaro non sarà un multiplo di 64. Questi verranno riempiti con una sequenza di 0, per permettere ad X_n di poter effettuare lo XOR e di conseguenza per poter ricavare Y_n.

Risulta però chiaro che si ottiene un messaggio cifrato che è di lunghezza maggiore al testo in chiaro. Per far risultare la lunghezza del testo in chiaro e quella del messaggio cifrato uguali, vengono eliminati j bit del testo cifrato dal penultimo blocco. Il risultato del messaggio cifrato è il seguente:



Il messaggio cifrato sarà costituito da blocchi di 64 bit, escluso il penultimo a cui mancano j bit finali. In questo modo, la lunghezza del testo in chiaro e del messaggio cifrato, coincidono.

La domanda che sorge spontanea ora è: nel momento in cui andiamo a decifrare, Y^R_{n-1} non è più presente nel messaggio cifrato, poiché lo abbiamo eliminato in precedenza. Come si recupera? Si può osservare che, nella fase di cifratura del testo in chiaro, l'ultima operazione di XOR viene fatta tra la Y_{n-1} (formato da Y^l_{n-1} e Y^r_{n-1})e X_n che si compone di X^l_n seguito da tanti 0 quanti sono per completare il blocco, ma uno XOR di una sequenza j di bit con tanti 0, corrispondono proprio ai bit di Y^r_{n-1} che sono stati assegnati a Y_n del messaggio cifrato. Di conseguenza nel processo di decifrazione gli ultimi j bit di Y_n sono proprio i j bit che appartengono al penultimo blocco, dunque è necessario spostarli e vengono recuperati subito. Una volta recuperati i Y^r_{n-1}, la decifratura può avvenire come stabilito dal CBC.



J a quanto corrisponde? (64 – la lunghezza del testo modulo 64)

Nel preservare la lunghezza questa risulta essere una buona tecnica. Ma ne esistono altre.

Sono state create 5 modalità di uso dei cifrari a blocchi, che mirano alla privatezza, confidenzialità. Ne esistono altre e vengono solo elencate:

- ***Authenticated encryption mode*** (Counter with CBC-MAC CCM ; Galois / Counter Mode (GCM): basata sull'autenticazione (chi ha inviato quel messaggio).
- ***Authentication modes*** (Cipher-based MAC CMAC):
- ***Format Preserving Encryption***, dati non binari: volte a salvaguardare il formato, ad esempio se abbiamo un file pdf e vogliamo cifrarlo facendo in modo che anche il cifrato sia un file pdf.
- ***Key Wrapping*** (protezione chiavi crittografiche): cifrare uno o più chiavi crittografiche.

Il DES non è stato rotto per la sua debolezza, ma perché la chiave è troppo corta (56 bit). Quanto deve essere lunga la chiave per evitare attacchi di ricerca esaustiva? Vengono elencati i vari valori:

Grandezza chiavi (bits)	Numero di possibili chiavi	Tempo medio necessario con 1 decifratura/μs	Tempo medio necessario con 10^6 decifrature/μs
32	$2^{32} \approx 4{,}3 \times 10^9$	$2^{31}\ \mu$s = 35,8 minuti	2,15 millisecondi
56	$2^{56} \approx 7{,}2 \times 10^{16}$	$2^{55}\ \mu$s = 1142 anni	10,01 ore
112	$2^{112} \approx 5{,}19 \times 10^{33}$	$2^{111}\ \mu$s = $8{,}2 \times 10^{19}$ anni	$8{,}2 \times 10^{13}$ anni
128	$2^{128} \approx 3{,}4 \times 10^{38}$	$2^{127}\ \mu$s = $5{,}4 \times 10^{24}$ anni	$5{,}4 \times 10^{18}$ anni
168	$2^{168} \approx 3{,}7 \times 10^{50}$	$2^{167}\ \mu$s = $5{,}9 \times 10^{36}$ anni	$5{,}9 \times 10^{30}$ anni
256	$2^{256} \approx 1{,}15 \times 10^{77}$	$2^{255}\ \mu$s = $1{,}83 \times 10^{63}$ anni	$1{,}83 \times 10^{57}$ anni

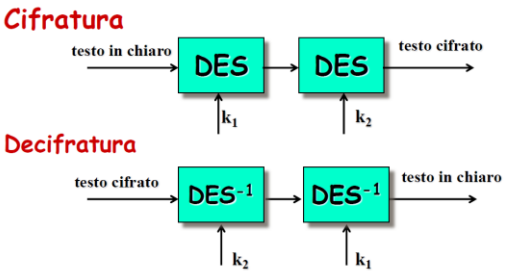
4.6 CIFRATURA MULTIPLA

Se i 56 bit sono pochi per rendere sicuro il DES, si può utilizzare il DES esistente per ottenere un cifrario più forte? L'idea è di utilizzare la *crittografia a cifratura multipla*, cioè cifrare il messaggio varie volte con chiavi differenti. Dato il DES si ottiene:

4.6.1 DES DOPPIO

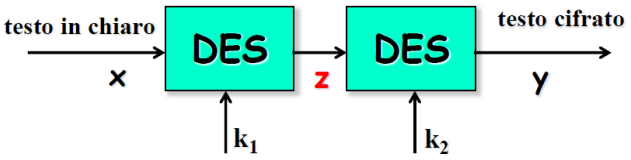
Il testo in chiaro viene cifrato due volte con 2 chiavi indipendenti tra di loro: k_1 e k_2. Se cifro in cifro in cascata la lunghezza del blocco è sempre 64 bit, mentre la chiave risulta essere k1= 56 bit, k2=56 bit → 112 bit.

Dato un testo cifrato, la sua decifratura avviene effettuando una prima decifratura con la chiave k_2 e successivamente una decifratura con k_1.



Quanto è sicuro il DES doppio? Tenendo conto che la doppia cifratura non è equivalente ad una cifratura singola poiché il DES non è un insieme chiuso per la composizione, è necessario analizzare ***l'attacco meet in the middle***.

ATTACCO MEET IN THE MIDDLE:



È possibile eseguire un attacco del genere con la modalita *Known Plaintext Attack* che può avvenire quando si conoscono le coppie (x: testo in chiaro, e il corrispettivo y: testo cifrato), ma non la chiave (k_1,k_2).

Per poter rompere il DES doppio si esegue un attacco meet in the middle, che si basa su due flussi di computazione che si incontrano nel mezzo del cifrario (nel punto **z**).

L'idea per l'attacco è la seguente:

Data una coppia nota (x,y), cifro x, che si ricorda è il testo in chiaro, con i 2^{56} valori possibili della chiave k_1. Conoscendo il corrispettivo y, decifro y con i 2^{56} valori possibili di k_2. Se tra le 2^{56} cifrature e le 2^{56} decifrature trovo un valore uguale ($Des_{ka}(x) = DES^{-1}{}_{kb}(y)$), allora viene trovata la chiave che trasforma x in y.

Dal punto di vista computazionale: vengono eseguite tutte le cifrature di x per i 2^{56} valori possibili di k_1 e posti in una tabella (di grandezza 2^{56}, dunque tempo di creazione 2^{56}). Successivamente vengono eseguite le decifrature di y per i 2^{56} valori possibili di k_2 e si cercano le corrispondenze nella tabella. La ricerca avviene in tempo 2^{56} nel caso peggiore cioè elemento non presente.

Per verificare se un elemento è presente ci vogliono 2^{56} per 2^{56} elementi e la complessità sarà 2^{56} x 2^{56} = 2^{112}. Quindi non si ha alcun vantaggio perché è uguale al tempo di attacco del DES doppio in forza bruta.

Known Plaintext Attack
Input: x, y = $DES_{k_2}(DES_{k_1}(x))$

Costruisci tabella

chiave	testo cifrato
00…00	$DES_{00…00}(x)$
00…01	$DES_{00…01}(x)$
…	…
11…11	$DES_{11…11}(x)$

for $k_b \in \{0,1\}^{56}$
do $z = DES^{-1}{}_{k_b}(y)$
if per qualche k_a, (k_a, z) è nella tabella
then return la chiave è (k_a,k_b)

Questo attacco può essere migliorato, organizzando la tabella in modo differente, per rendere efficiente la ricerca al suo interno. Può essere fatto mediante array ordinato, che consiste di ordinare la tabella non sulla chiave, ma sul testo cifrato. Una volta ordinata per testo cifrato, come tempo di ricerca occorre il logaritmo della dimensione della tabella. In questo caso $\log_2(2^{56}) = 56$. Se invece si utilizza una tabella Hash, mettendo l'Hash sul testo cifrato, anche in questo caso il tempo è costante. Cioè O(1).

Quindi, a differenza di prima in cui la ricerca in tabella è 2^{56} x 2^{56}, in questo caso è 2^{56} x costante che è uguale a 2^{56}. Riguardiamo le varie complessità:

- Complessità spazio: 2^{56} righe nella tabella.
- Complessità tempo:
  - 2^{56} cifrature per x (costruzione tabella)
  - 2^{56} decifrature per y
  - 2^{56} ricerche in tabella: O(1) se tabella hash. 56 se array ordinato.

La complessità di questo attacco è identica a quella del DES semplice, quindi il DES doppio non è adeguato.

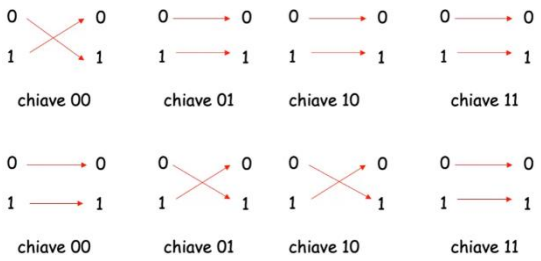
Una volta che viene ottenuta una chiave (k_1, k_2) è sicuramente la chiave cercata? Se viene ottenuta una chiave da una coppia (x,y) è necessario fare qualche calcolo: dato x,y qual è il numero medio di chiavi (k_1, k_2) tali che $y= DES_{k2}(DES_{k1}(x))$? È un problema troppo complicato calcolarlo esattamente per il DES, quindi è meglio fare ipotesi sul cifrario. Prima di analizzare ciò, si parte da un esempio semplificato per il calcolo:

Questo è un cifrario con un bit di input e un bit di output e una chiave con 2 bit, nella prima riga sono indicati i possibili valori di cifratura per un bit in chiaro e con una chiave. La seconda riga è differente dalla prima, solo per scopo di esempio.

Dato (0,0) qual è il numero medio di chiavi tali che $0= E_k(0)$? Fissato x=0 come input, ci sono 4 chiavi e 2 possibili testi cifrati di y (tenendo in considerazione solo la prima riga):

numero medio di chiavi = #chiavi/#numero y per fissato x= 4/2= 2

Questa è l'idea generale per rendere l'idea.



Ora andiamo ad applicare l'idea appena descritta al DES Doppio con attacco meet in the middle:

Fissato x, ci sono 2^{112} chiavi e 2^{64} testi cifrati y: **numero medio di chiavi=** #chiavi/#numero y per fissato x= 2^{112} / 2^{64} = 2^{48}. Questo implica che ci sono in media 2^{48} chiavi che fanno corrispondere un fissato x in un y. Quando trovo una chiave, risulta essere un problema che c'è ne sono 2^{48} perché non è detto che sia la chiave che si sta cercando.

Per aumentare la possibilità di trovare la chiave giusta si può prendere non sola una coppia (x,y) ma una seconda coppia (x', y') con la stessa coppia di chiavi (k_1, k_2). In questo caso, quando trovo una chiave per la prima coppia (x,y), vedo se è buona anche per la seconda coppia (x', y'). Se è buona anche per la seconda coppia, allora la chiave è giusta.

In questo nuovo attacco la complessità di tempo è 2^{56}, uguale a quello precedente. Analizziamo invece il calcolo:

Fissati x,x', ci sono 2^{112} chiavi e 2^{128} (64 bit per y e 64 bit per y') testi cifrati y, y': **numero medio di chiavi=** #chiavi/#numero y per fissato x = 2^{112} / 2^{128} = 2^{-16}. In questo caso la probabilità di indovinare la chiave corretta è: 1- 2^{-16}= 0.99998474 → posso essere confidente che la chiave è giusta. Se si vuole ancora più certezza è necessario dare in input un'ulteriore coppia (x'', y'').

Known Plaintext Attack
Input: $x, y = DES_{k_2}(DES_{k_1}(x))$
$x', y' = DES_{k_2}(DES_{k_1}(x'))$

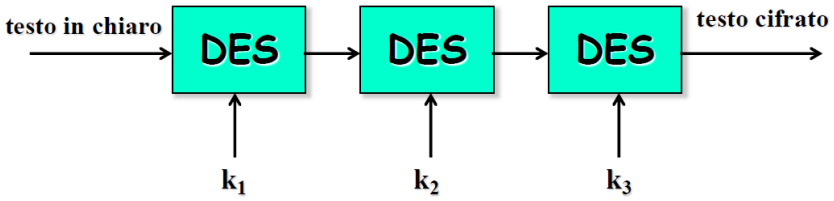
Costruisci tabella

chiave	testo cifrato
00...00	$DES_{00...00}(x)$
00...01	$DES_{00...01}(x)$
...	...
11...11	$DES_{11...11}(x)$

for $k_b \in \{0,1\}^{56}$
do $z = DES^{-1}_{k_b}(y)$
if per qualche k_a, (k_a, z) è nella tabella
e $y' = DES_{k_b}(DES_{k_a}(x'))$
then return la chiave è (k_a, k_b)

In conclusione il DES doppio è "equivalente" ad un cifrario con una chiave di 56 bit, e non 112. La sua semplice rottura ha implicato il suo scarso utilizzo. Da questo deriva il **DES TRIPLICATO.**

4.6.2 DES TRIPLICATO

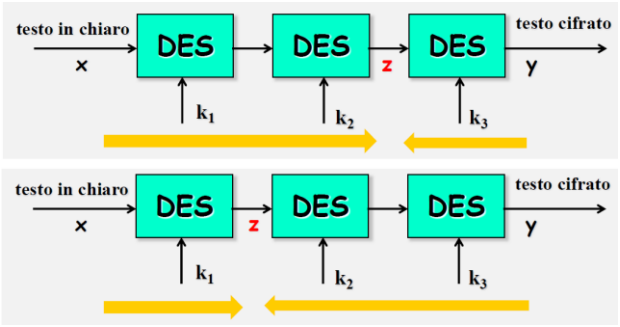


Il testo chiaro viene cifrato per 3 volte, con 3 chiavi differenti. In questo cifrario la lunghezza blocco è 64 bit mentre la chiave (k_1, k_2, k_3) è lunga 56 + 56 + 56= 168 bit.

Anche in questo caso la decifratura avviene invertendo le 3 operazioni di cifrazione e le 3 chiavi, come per il DES doppio.

ATTACCO MEET IN THE MIDDLE:

L'attacco è quasi simile al DES doppio. Simile perché in questo caso non si ha un unico centro, poiché il centro effettivo è rappresentato dal secondo DES. Possiamo considerare 2 centri: uno dopo il primo DES e uno dopo il secondo DES.



Quindi l'attacco può avvenire in 2 modi. Consideriamo il primo, in cui il nostro meet in the middle è fra il secondo e il terzo DES (come visibile nella prima foto). In questo caso la tabella è ottenuta in questo modo: fissato x, cifro 2 volte con le chiavi k_1 e k_2 e ottengo i z che posiziono in tabella. Una volta ottenuta la tabella, prendo un y e decifro con la chiave k_3 e ottengo tutti i z, se il valore è presente nella tabella ho trovato la chiave, altrimenti si continua la ricerca. L'analisi della complessità è:

- Complessità spazio: 2^{112} righe nella tabella.
- Complessità tempo:
  - 2^{112} cifrature per x (costruzione tabella)
  - 2^{56} decifrature per y
  - 2^{56} ricerche in tabella

La complessità totale è 2^{112}.

Known Plaintext Attack
Input: $x, y = DES_{k_3}(DES_{k_2}(DES_{k_1}(x)))$

Costruisci tabella

chiave	testo cifrato
00...00	$DES_{00...00}(DES_{00...00}(x))$
00...01	$DES_{00...00}(DES_{00...01}(x))$
...	...
11...11	$DES_{11...11}(DES_{11...11}(x))$

for $k_c \in \{0,1\}^{56}$
do $z = DES^{-1}_{k_c}(y)$
if per qualche k_a,k_b, ($k_a k_b$, z) è nella tabella
then return la chiave è (k_a,k_b,k_c)

Se invece si attacca posizionando z fra il primo e il secondo DES, ottengo la tabella cifrando una sola volta x con la chiave k_1, y viene decifrato 2 volte con le chiavi k_3 e k_2 ed ottengo il valore z associato. Se presente in tabella, trovo una chiave. L'analisi della complessità è:

- Complessità spazio: 2^{56} righe nella tabella.
- Complessità tempo:
  - 2^{56} cifrature per x (costruzione tabella)
  - 2^{112} decifrature per y
  - 2^{112} ricerche in tabella

Known Plaintext Attack
Input: $x, y = DES_{k_3}(DES_{k_2}(DES_{k_1}(x)))$

Costruisci tabella

chiave	testo cifrato
00…00	$DES_{00\ldots00}(x)$
00…01	$DES_{00\ldots01}(x)$
…	…
11…11	$DES_{11\ldots11}(x)$

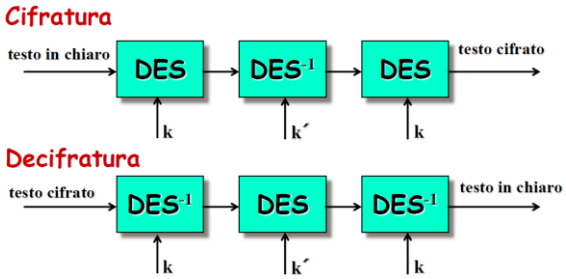
for $k_c, k_b \in \{0,1\}^{56} \times \{0,1\}^{56}$
  do $z = DES^{-1}_{k_c}(DES^{-1}_{k_c}(y))$
    if per qualche k_a, (k_a, z) è nella tabella
      then return la chiave è (k_a, k_b, k_c)

La complessità totale è 2^{112}.

Per il DES triplicato, la complessità di rottura è 2^{112}, di conseguenza è "equivalente" ad un cifrario con una chiave di 112 bit, e non 168 bit.

Un'ulteriore proposta fatta per sopperire alle lacune del DES triplicato è il **DES triplo** (anche se con nome molto simile, non si confondano le due idee).

4.6.3 DES TRIPLO



Viene spesso chiamato $EDE_{k,k'}$ (acronimo per Encrypt Decrypt Encrypt), TDEA, 3TDEA, 2TDEA. In questo cifrario la lunghezza di blocco è 64 bit. La chiave (k, k') è lunga 56+56 = 112 bit.

Il testo in chiaro viene cifrato con una chiave k, successivamente viene decifrato (retrocompatibilità con lo standard precedente, non viene alterata la sicurezza, complessità) con una chiave k' e poi cifrato con la chiave k uguale alla prima chiave.

La decifratura è banale.

La questione importante è che il DES triplo ha una chiave di 112 bit. Ma se viene effettuato un attacco di tipo meet in the middle, cosa succede? Anche in questo caso, come per il DES triplicato si possono analizzare i 2 casi.