Crittografia simmetrica basata su dizionario

Indice

Elenco delle figure

1 Crittografia a chiave simmetrica

Garantire la segretezza di dati sensibili è lo scopo principale della crittografia; le tecniche moderne di **cifratura** utilizzano una serie di manipolazioni matematiche che considerano il messaggio da cifrare (*testo in chiaro*) come una serie di numeri o elementi appartenenti ad uno spazio algebrico e lo trasformano in un testo inintelligibile (*messaggio cifrato*).

Per recuperare il testo in chiaro a partire dal messaggio cifrato, il sistema di crittografia deve essere reversibile e l'operazione inversa prende il nome di **decifratura**; per definizione, tutte queste operazioni si appoggiano sull'uso di *chiavi* crittografiche. L'insieme costituito dalle chiavi e dagli algoritmi di cifratura e decifratura è chiamato **crittosistema**.

Un'importante proprietà che ogni crittosistema dovrebbe avere è che lo "spazio" del messaggio cifrato contenga tutti i possibili messaggi, e che quello del testo in chiaro sia una regione "sparsa" all'interno dello spazio di tutti i messaggi "sensati"; un buon alogritmo di cifratura contiene una serie di trasformazioni che distribuiscono i messaggi uniformemente nello spazio di tutti i possibili messaggi.

La simbologia standard che si usa in crittografia è la seguente:

- **Spazio del testo in chiaro**: $\mathcal{M}$
  (una serie di stringhe costruite su un dato alfabeto).
- **Spazio dei messaggi cifrati**: $\mathcal{C}$
  (l'insieme di tutti i possibili messaggi cifrati).
- **Spazi delle chiavi**: $\mathcal{K}$ e $\mathcal{K}'$
  (l'insieme di tutte le possibili chiavi di cifratura e decifratura).
- **Algoritmo di generazione delle chiavi**: $\mathcal{G} : \mathbb{N} \mapsto \mathcal{K} \times \mathcal{K}'$.
  - $\mathcal{G}$ fornisce una coppia di chiavi $(ke, kd) \in \mathcal{K} \times \mathcal{K}'$ di lunghezza l.
- **Algoritmo di cifratura**: $\mathcal{E} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{C}$.
- **Algoritmo di decifratura**: $\mathcal{D} : \mathcal{C} \times \mathcal{K}' \mapsto \mathcal{M}$.
- **Messaggio cifrato**: $c = \mathcal{E}_{ke}(m)$, per ogni $ke \in \mathcal{K}$ e $m \in \mathcal{M}$.
- **Testo decifrato**: $m = \mathcal{D}_{kd}(c)$.

La seguente equazione esprime la reversibilità del crittosistema:

$$\mathcal{D}_{kd}\big(\mathcal{E}_{ke}(m)\big) = m$$

I **crittosistemi simmetrici** si basano sull'assunzione che $kd = ke$, cioè che la stessa chiave debba essere usata sia in cifratura che in decifratura.

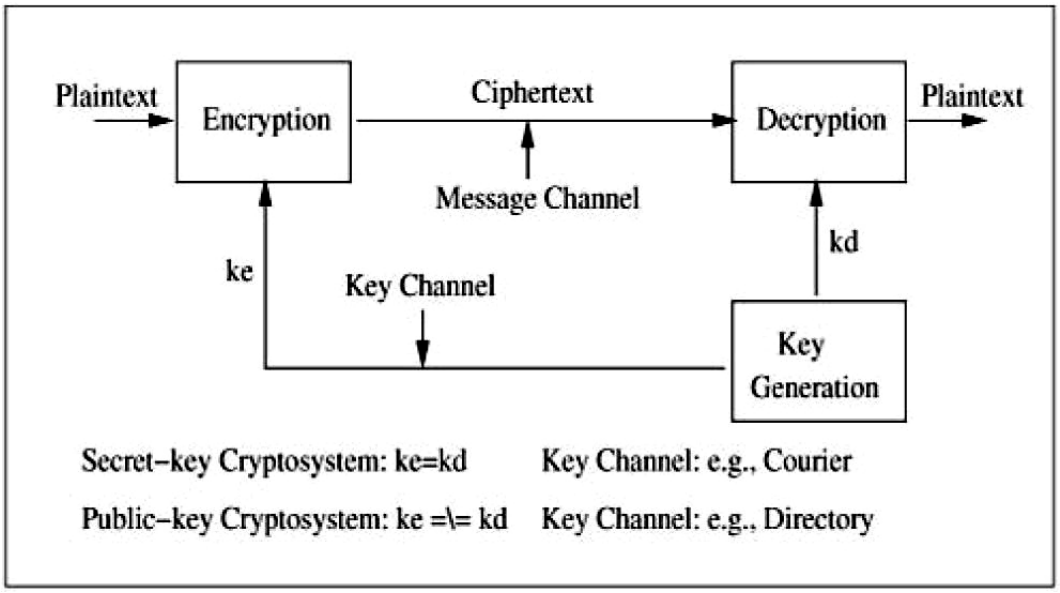

Figura 1: Schematizzazione di un crittosistema a chiave simmetrica.

Nel 1883, KERCHOFF compilò una lista di requisiti da rispettare durante la progettazione dei moderni crittosistemi ed il più importante di essi è:

> "Gli ideatori di un crittosistema devono necessariamente renderne pubblica l'architettura e la dimensione delle chiavi, perché, se un malintenzionato ne venisse a conoscenza, è preferibile non affidarsi alla sua segretezza, ma ad una robustezza testata dal maggior numero di utenti possibile."

Mettendo insieme la caratterizzazione semantica fornita da SHANNON con il principio di KERCHOFF, possiamo ricavare le seguenti linee guida per la progettazione di un buon crittosistema:

- Gli algoritmi $\mathcal{E}$ e $\mathcal{D}$ non devono contenere componenti segreti.
- Le funzioni $\mathcal{E}$ e $\mathcal{D}$ devono essere computazionalmente "facili" da calcolare (*efficienza*).
- La funzione $\mathcal{E}$ deve distribuire uniformemente i messaggi "sensati" su tutto lo spazio dei messaggi cifrati (*diffusione*).
- L'operazione di ricostruzione del testo in chiaro a partire dal messaggio cifrato deve essere un problema la cui difficoltà dipende esponenzialmente dalla dimensione dello spazio delle chiavi (*confusione*).

Tuttavia è semplice notare come queste caratteristiche si rivelino spesso inadeguate per applicazioni moderne, ed infatti nelle sezioni seguenti mostreremo altri requisiti più stringenti appositamente progettati ad hoc a seconda dei vari crittosistemi.

1.1 Cifrari monoalfabetici

In un **cifrario a sostituzione** l'algoritmo di cifratura $\mathcal{E}_k(M)$ è una funzione di sostituzione che rimpiazza ogni $m \in M$ con il corrispondente $c \in C$. La funzione utilizza come parametro la chiave segreta k e la decifratura $\mathcal{D}_k(c)$ è semplicemente la sostituzione inversa; in generale, essa rappresenta la mappatura $\pi : \mathcal{K} \mapsto \mathcal{C}$, mentre la sostituzione inversa è $\pi^{-1} : \mathcal{C} \mapsto \mathcal{M}$.

1.1.1 Cifrario a sostituzione semplice

Poniamo $\mathcal{M} = \mathcal{C} = \mathbb{Z}_{26}$ e $(A = 0, B = 1, \ldots, Z = 25)$; l'algoritmo di cifratura $\mathcal{E}_k(m)$ rappresenta quindi una generica permutazione all'interno di $\mathbb{Z}_{26}$. Lo spazio delle chiavi ha dimensione $26! > 4 \times 10^{26}$, che è piuttosto grande rispetto a quella dello spazio $\mathcal{M}$, ma nonostante ciò il cifrario è molto debole.

Ogni carattere dell'alfabeto viene associato ad un altro carattere, senza sovrapposizioni; la debolezza risiede nel fatto che la **crittoanalisi** può fare leva sull'analisi della frequenza delle lettere, dato che il linguaggio naturale presenta caratteristiche statistiche molto regolari, come vedremo in § 1.3.

Prendiamo ad esempio come testo in chiaro la frase "`oggi vado al mare`" e come chiave la seguente permutazione:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
Q	W	E	R	T	Y	U	I	O	P	A	S	D

N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
F	G	H	J	K	L	Z	X	C	V	B	N	M

Il corrispondente messaggio cifrato sarà "`guuo cqrg qs dqkt`".

1.1.2 Cifrario a scorrimento

Nella storia hanno fatto la loro comparsa un gran numero di cifrari a sostituzione, ma di sicuro i più usati sono stati i cifrari a scorrimento, dove la cifratura e la decifratura sono così definite:

$$\begin{cases} \mathcal{E}_k(m) \leftarrow m + k & (\text{mod } N) \\ \mathcal{D}_k(c) \leftarrow c - k & (\text{mod } N) \end{cases}$$

con $m, c, k \in \mathbb{Z}_N$.

Nel caso in cui $k = 3$ e l'alfabeto coincida con le 26 lettere occidentali, cioè $\mathcal{M} = \mathbb{Z}_{26}$, il cifrario a scorrimento risultante è tradizionalmente chiamato **cifrario di Cesare**, dal nome di *Giulio Cesare* che lo utilizzava frequentemente nelle sue comunicazioni segrete; in questo caso la permutazione diventa:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
C	D	E	F	G	H	I	J	K	L	M	N	O

N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
P	Q	R	S	T	U	V	W	X	Y	Z	A	B

Il messaggio in chiaro "`oggi vado al mare`" diventa "`rjjl ydgr do pduh`".

1.1.3 Cifrario affine

Se k e N sono primi tra loro, si ha che $k \cdot m$ assume valori nell'intero spazio $\mathbb{Z}_N$ del messaggio ($\forall\, m < N$), perciò con $m, c < N$ possiamo definire

$$\begin{cases} \mathcal{E}_k(m) \leftarrow k \cdot m & (\text{mod } N) \\ \mathcal{D}_k(c) \leftarrow k^{-1} \cdot c & (\text{mod } N) \end{cases}$$

In questo senso, anche $k_1 \cdot m + k_2$ rappresenta una cifratura a sostituzione, che viene infatti utilizzata nel **cifrario affine**, del quale riportiamo le funzioni di cifratura e decifratura:

$$\begin{cases} \mathcal{E}_k(m) \leftarrow k_1 \cdot m + k_2 & (\text{mod } N) \\ \mathcal{D}_k(c) \leftarrow k^{-1} \cdot (c - k_2) & (\text{mod } N) \end{cases}$$

Non è difficile vedere che applicando varie operazioni aritmetiche tra le chiavi in $\mathcal{K}$ ed i messaggi in $\mathcal{M}$, è possibile dare origine a vari casi di cifrari a sostituzione semplice, che vengono chiamati **cifrari monoalfabetici**, anch'essi estremamente suscettibili ad attacchi basati sull'analisi delle frequenze.

Nonostante ciò, proprio per la loro semplicità, i cifrari a sostituzione sono largamente impiegati anche nei moderni crittosistemi ed infatti vedremo che costituiscono il cuore di crittosistemi ben più complessi, come ad esempio il DES e l'AES; è stato dimostrato che la combinazione di un sufficiente numero di algoritmi di cifratura semplice dà origine ad un algoritmo di cifratura forte.

1.2 Cifrari polialfabetici

Un crittosistema a sostituzione viene detto **polialfabetico** se almeno un elemento del testo in chiaro viene associato a più elementi nel messaggio cifrato; l'esempio più conosciuto è il **cifrario di Vigenère**: una volta fornita come chiave una parola di lunghezza m, il testo in chiaro viene suddiviso in segmenti di lunghezza m e la cifratura avviene "sommando" la lettera in chiaro con la corrispondente lettera della chiave (la decifratura segue il procedimento inverso).

Prendendo sempre come esempio la frase "`oggi vado al mare`", possiamo impostare la segmentazione tramite la chiave "`critto`":

m	=	o	g	g	i		v	a	d	o		a	l		m	a	r	e
k	=	c	r	i	t		t	o	c	r		i	t		t	o	c	r
c	=	q	x	o	b		o	o	f	f		i	e		f	o	t	v

Il testo cifrato sarà quindi "`qxob ooff ie fotv`".
Esistono molti altre tipologie di cifrari polialfabetici, ma quelli più famosi sono il **cifrario di Beale** (la chiave viene scelta casualmente da un libro di testo) ed il **cifrario di Hill** (la chiave viene inizialmente disposta in una matrice).

1.3 Crittoanalisi dei cifrari classici

I cifrari a scorrimento sono i più facili da forzare in quanto anche un attacco a forza bruta, cioè provando tutte le 26 chiavi possibili, si dimostra fattibile anche manualmente, senza ricorrere all'uso di calcolatori; alternativamente è possibile comunque ricorrere alle frequenze delle lettere, come mostreremo in seguito.

Per quanto riguarda invece i cifrari a sostituzione di caratteri, dobbiamo tener conto che lo spazio del testo in chiaro coincide con l'alfabeto e quindi ogni carattere in chiaro corrisponderà sempre allo stesso carattere cifrato, qualsiasi sia la sua posizione e, come accennato in precedenza, un'analisi delle frequenze delle lettere permette di risalire al testo originale, a patto di conoscere a priori la lingua in cui è stato scritto il testo in chiaro; se volessimo sondare tutto lo spazio delle chiavi, ci troveremmo di fronte a $26! \approx 4.032 \times 10^{26}$ possibilità.

Come riferimento per generare statistiche attendibili è meglio scegliere un corpus rappresentativo di documenti scritti nella lingua interessata; la Fig. 2 riporta l'istogramma delle frequenze estratte dal romanzo "OceanoMare" di *Baricco*:

Confrontando l'istogramma del testo cifrato con quello del campione scelto, è possibile ricostruire la permutazione originale e completare le associazioni facendo delle inferenze linguistiche (è lo stesso procedimento che si usa quando si tenta di risolvere la popolare *Crittografia* della Settimana Enigmistica).

Dal punto di vista della sicurezza, i cifrari polialfabetici sono più robusti, perché ogni carattere può essere mappato in molteplici simboli, rendendo vano un attacco statistico come quello visto in precedenza; tuttavia, studiando l'autocorrelazione delle lettere è possibile individuare un certa periodicità che corrisponde alla lunghezza della chiave, perché più il testo è lungo, più è probabile che una lettera venga cifrata con la medesima lettera della chiave.

La Fig. 3 mette a confronto l'autocorrelazione del testo in chiaro con quella del cifrato ed appare evidente la periodicità pari alla lunghezza della chiave scelta ("`CRITTO`" = 6 lettere); dato che la cifratura avviene sommando le posizioni delle lettere, una volta suddiviso il testo cifrato in segmenti di opportuna lunghezza, è facile applicare congiuntamente i concetti visti per il cifrario a scorrimento.

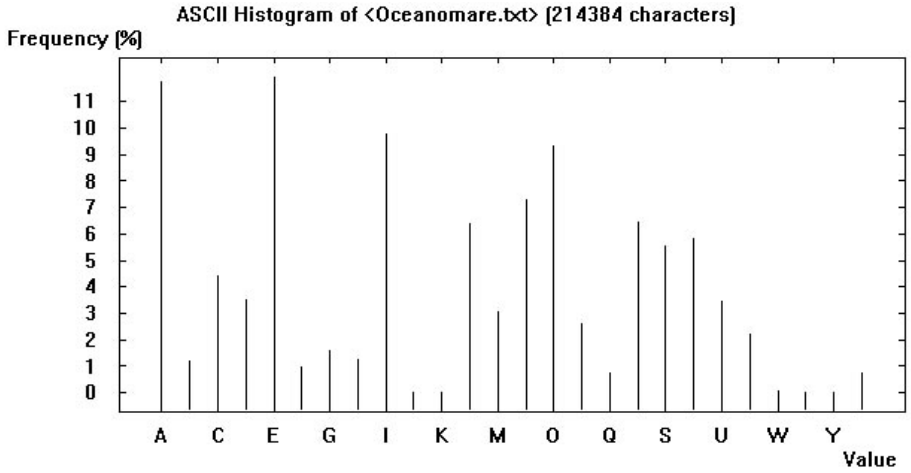

Figura 2: Istogramma delle frequenze relative delle lettere calcolato sul testo del libro "OceanoMare" di *Baricco*.

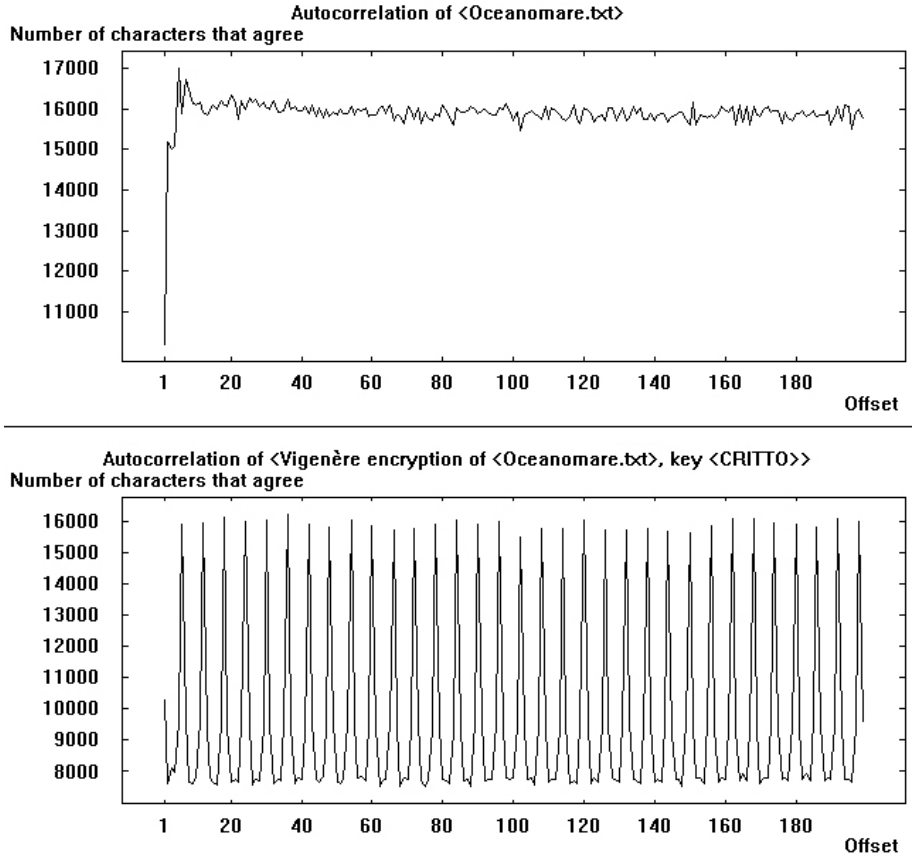

Figura 3: Comparazione tra le autocorrelazioni calcolate sul testo in chiaro e su quello cifrato ("OceanoMare" di *Baricco* cifrato con Vigenère).

1.4 Cifrari a blocchi

1.4.1 Cifrario di *Feistel*

Il cifrario o rete di *Feistel* di è un cifrario a blocchi con una struttura molto particolare che ha preso il nome dal suo inventore *Horst Feistel*. L'importanza di questo cifratore è data dal fatto che molti cifratori a blocchi utilizzano questo schema, incluso il Data Encryption Standard (*DES*). La struttura di *Feistel* ha il vantaggio che le fasi di cifratura e di decifratura sono molto simili, identiche in alcuni casi, e richiedono solamente lo schedule inverso della chiave. La rete di *Feistel* fù commercializzata per la prima volta da *IBM* con l'introduzione del cifratore *Lucifer* progettato dallo stesso Feistel e da *Coppersmith*. Questo sistema acquisì rispettabilità nel momento in cui il Governo Federale degli U.S.A. adottò il *DES*. Uno dei vantaggi che si riscontrò nell'utilizzo della rete di Feistel nel *Des* fù quello della facilità con cui si riuscì ad implementare in hardware questo crittosistema. La struttura proposta da Feistel prevedeva di avere in input un blocco di lunghezza $2w$ bit ed una chiave k. Il blocco in ingresso viene diviso in due parti L_0 ed R_0 che vengono elaborate in 16 iterazioni ogni volta con una sottochiave diversa secondo lo schema di Fig. 4

La realizzazione di una rete di Feistel dipende dalla scelta di alcuni parametri significativi:

- *Grandezza del blocco in input* La scelta di un blocco grande aumenta la sicurezza del crittosistema ma aumenta il tempo di cifratura e decifratura.
- *Grandezza della chiave* Aumenta il livello di sicurezza ma diminuisce la velocità di cifratura e decifratura.
- *Numero di Round* Un solo round avrebbe portato ad un livello di sicurezza inadeguato; solitamente il numero di round è 16.
- *Algoritmo di generazione della Sottochiave* L'utilizzo di un algoritmo con un'elevata complessità computazionale dovrebbe garantire una maggiore difficoltà nel momento in cui viene effettuata una crittoanalisi.

1.4.2 Data Encryption Standard (DES)

Il Data Encryption Standard introdotto dalla IBM nella seconda metà degli anni settanta, è diventato lo standard per le comunicazioni commerciali riservate ma non classificate. Nel *DES* i criteri di Shannon sono soddisfatti mediante una serie di permutazioni ed espansioni dei bit del messaggio (diffusione), e la combinazione e successiva compressione dei bit del messaggio e della chiave (confusione). Il nucleo di quest'ultima operazione è costituito da un insieme di funzioni raggruppate in un blocco detto *S-Box* da cui dipende crucialmente la sicurezza del cifrario.

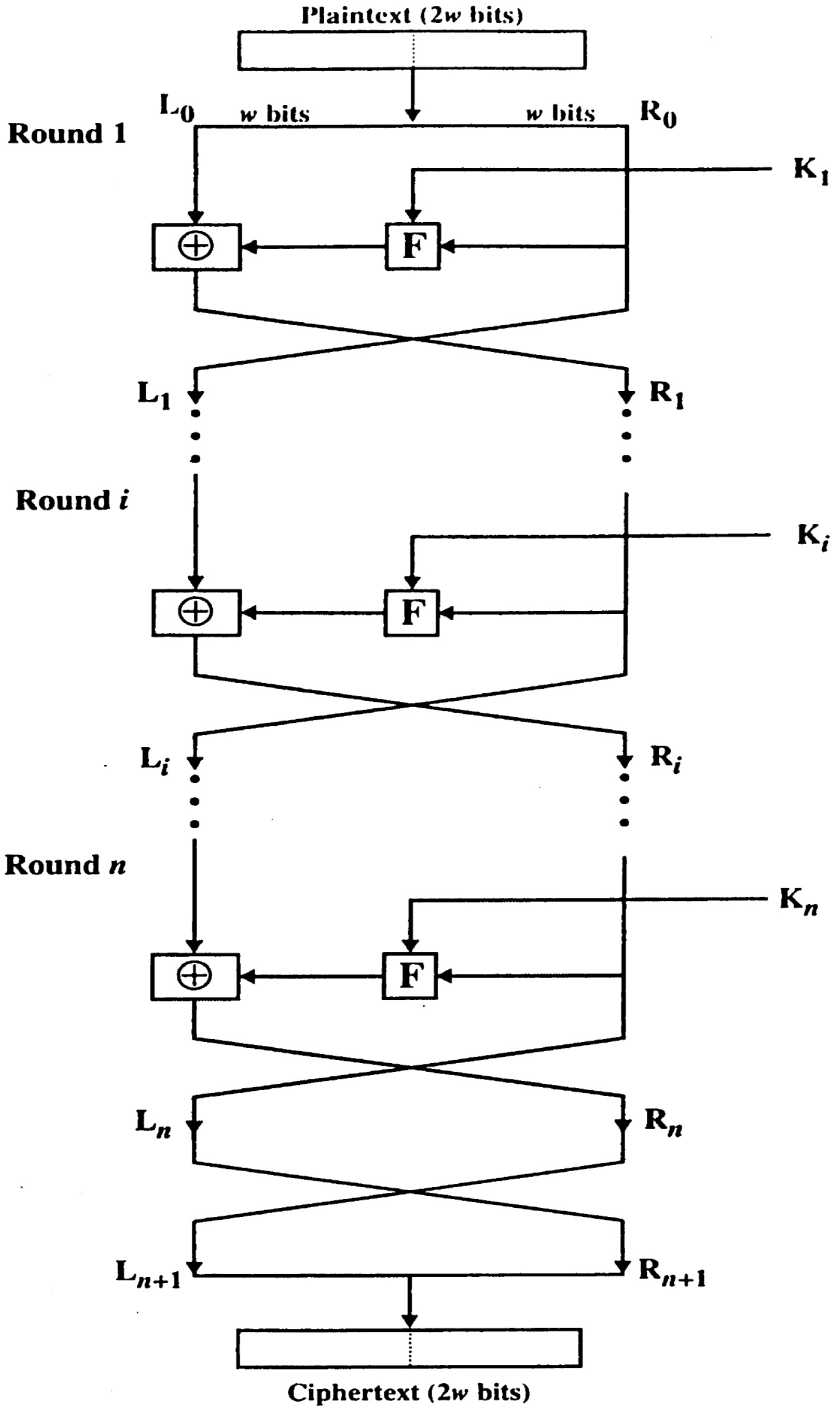

Figura 4: Struttura della rete di Feistel.

Il *DES* è un cifratore a blocchi di uso generale che presenta le seguenti caratteristiche:

- Il messaggio è suddiviso in blocchi ciascuno dei quali viene cifrato e decifrato indipendentemente dagli altri. Nel *DES* ogni blocco contiene 64 bit.
- Cifratura e decifratura procedono attraverso $r = 16$ iterazioni (round) successive in cui si ripetono le stesse operazioni.
- La chiave segreta è lunga 64 bit dei quali 56 scelti arbitrariamente e 8 di parità (il bit meno significativo di ogni byte che costituisce la chiave).
- Dalla chiave k vengono create r sottochiavi $k_0, k_1, ..., k_{r-1}$, impiegate una per round.
- Il messaggio viene diviso in due metà L e R (sinistra e destra). In ciascuna fase si eseguono le due operazioni: $L \longleftarrow R$ e $R \longleftarrow f(k_{i-1}, R, L)$ dove f è un'opportuna funzione non lineare ed $i = 1, ..., r$. Alla fine delle r iterazioni le due metà vengono nuovamente scambiate e poi concatenate per ottenere il crittogramma finale.
- La decifrazione consiste nel ripetere il processo invertendo l'ordine delle chiavi, ovvero usando la sequenza di chiavi $k_{r-1}, k_{r-2}, ..., k_0$.

Volendo schematizzare la struttura logica del *DES* possiamo far riferimento allo schema in Fig 5.

Generazione messaggio crittato L'algoritmo DES consiste di 16 iterazioni dove il blocco di bit di ingresso viene permutato mediante *PI* e suddiviso in un blocco sinistro *L[0]* e un blocco destro *R[0]* di 32 bit ciascuno. Sul blocco R_i si eseguono per tutte le iterazioni delle trasformazioni strutturalmente uguali, ciascuna con una diversa sottochiave K_i di 56 bit derivata dalla chiave iniziale K. Il blocco L_i viene semplicemente scambiato con il blocco R_i diventando al passo successivo il blocco R_{i+1}. L'i-esima iterazione riceve in ingresso tre blocchi $L_{i-1}, R_{i-1}, K_{i-1}$, rispettivamente di 32, 32 e 56 bit e produce in uscita tre nuovi blocchi L_i, R_i, K_i che costituiscono l'ingresso alla fase successiva. Dopo l'ultima fase i due blocchi $L[16]$ e $R[16]$ vengono scambiati e concatenati tra loro originando così il crittogramma finale costituito da un unico blocco di 64 bit.

Dunque volendo formalizzare questi concetti potremmo scrivere le seguenti relazioni relative al passo i-esimo:

$$L_i = R_{i-1}$$

$$L_i = R_{i-1} \oplus F(R_{i-1}, K_i)$$

In Fig 7 è possibile vedere lo schema relativo all'i-esimo round dell'algoritmo. Descriveremo quindi i dettagli dei passi che costituiscono l'algoritmo.

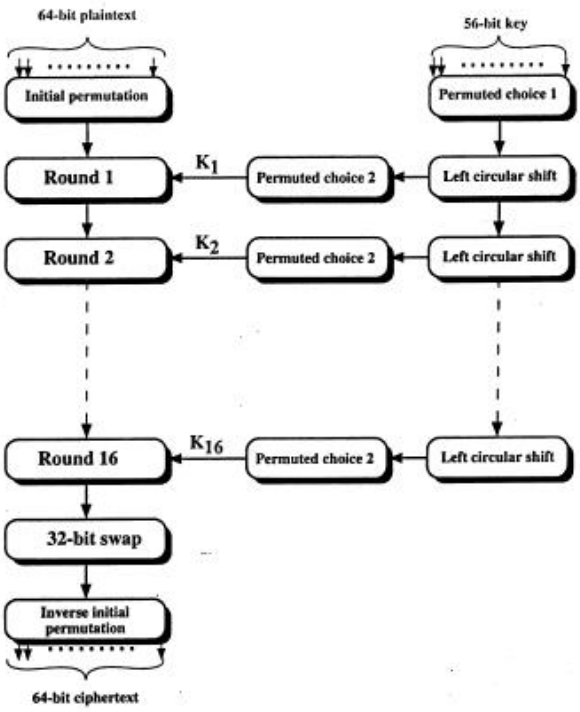

Figura 5: Struttura logica del crittosistema DES.

1. **Permutazione Iniziale:** La prima fase dell'algoritmo consiste di una permutazione iniziale *PI* dei bit del blocco che si stà cifrando, ottenendo l'input permutato. Vedi Fig 6

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

Figura 6: Matrice di permutazione iniziale PI nel DES

2. **Espansione e permutazione EP** La parte destra del blocco R_{i-1} di 32 bit viene espanso a 48 bit duplicando 16 bit in ingresso e permutata spostandone altri come mostrato in Fig. 8, per ottenere un blocco della stessa dimensione di quello estratto dalla sottochiave (di cui parleremo dopo) e poter eseguire lo *XOR* tra i due.

3. **OR esclusivo** tra la chiave K_i e l'uscita dal blocco *EP*.

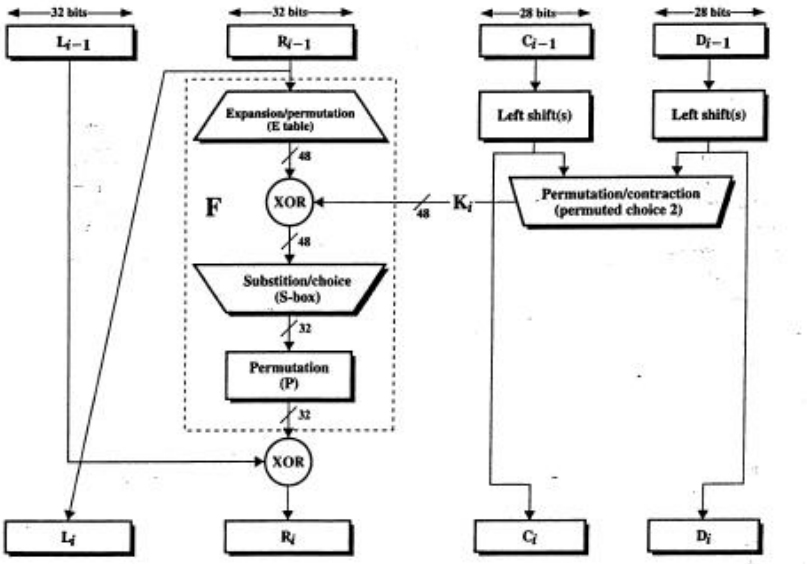

Figura 7: Iterazione i-esima dell'algoritmo DES

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

Figura 8: Sedici bit di ingresso vengono duplicati: per esempio il bit 32 è copiato nella posizione 1 e 47 dell'uscita.

4. **Sostituzione S** Questa funzione progettata con molta cura dalla *IBM* e modificata con altrettanta cura dalla *NSA*, costituisce la parte cruciale e magica su cui si basa la sicurezza del cifrario. Essa consta di otto sottofunzioni combinatorie $S_1, S_2, ..., S_8$ come mostrato in Fig. 9. L'ingresso di 48 bit viene decomposto in 8 blocchi $B_1, B_2, ..., B_8$ di 6 bit ciascuno che costituiscono l'ingresso alle sottofunzioni di pari indice. Sia $B_j = b_1b_2b_3b_4b_5b_6$. Questi bit vengono divisi in due gruppi, b_1b_6 e $b_2b_3b_4b_5$ che definiscono due numeri x, y con $0 \leq x \leq 3$ e $0 \leq y \leq 15$, utilizzati per accedere alla cella di riga x e colonna y in una tabella che definisce la sottofunzione S_j. Il numero ivi contenuto è compreso tra 0 e 15 ed è quindi rappresentato con 4 bit che costituiscono l'uscita della sottofunzione S_j realizzando una compressione da 6 a 4 bit. Complessivamente gli 8 blocchi generano una sequenza a 32 bit. La Fig. 10 è una rappresentazione della struttura *S-Box* delle sottofunzioni di sostituzione.

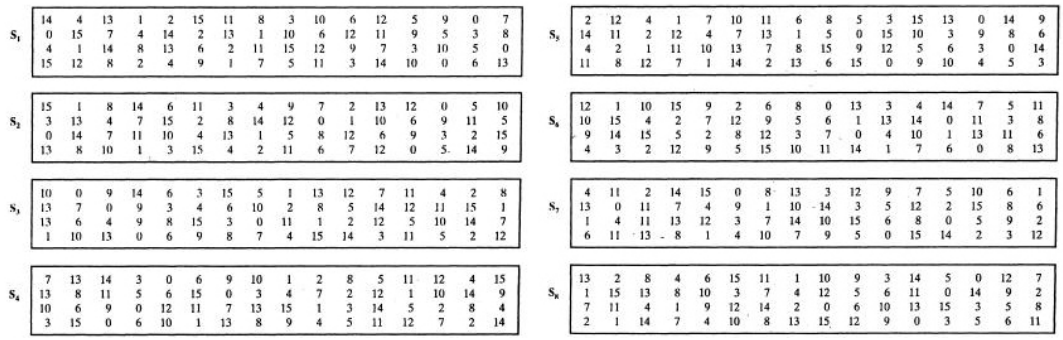

Figura 9: Le otto funzioni presenti nell'S-Box.

5. **Permutazione P** E' una permutazione di 32 bit.A questo punto viene effettuato lo *XOR* tra questa sequenza di bit e i 32 bit del blocco L_{i-1} generando il blocco finale R_i . Tale permutazione è mostrata in Fig. 11.

6. **OR esclusivo** tra il blocco in uscita da P e L_{i-1}.

7. **Scambio:** Il blocco R_{i-1} diventa il blocco L_i, ed il blocco in uscita al passo precedente diventa R_i.

Generazione delle chiavi La chiave di 64 bit del *DES* è ridotta a 56 bit ingnorando l'ottavo bit (leggendo da sinistra vestro destra lo stream di bit) di ogno byte che costituisce la chiave. Come già detto questi bit sono usati per il controllo di parità in modo da assicurarsi che la chiave non contenga errori.

Dopo che i 56 bit della chiave sono stati estratti, per ogni round del *DES*, è generata una distinta sottochiave a 48 bit. Abbiamo precedentemente indicato queste sottochiavi come k_i.

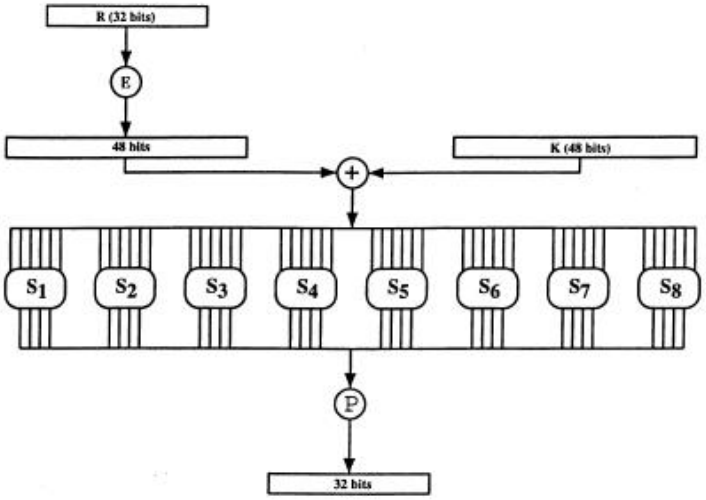

Figura 10: Schema logico dell'S-Box nel DES.

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

Figura 11: Permutazione finale del blocco generato all'i-esimo round.

Queste chiavi sono ottenute mediante il seguente procedimento.

1. **Permutation Choice One:** La chiave iniziale viene permutata mediante la funzione di Fig. 12 **a**. Questo avviene solo al primo round.

2. **Shift Sx:** La chiave a 56 bit è divisa in due blocchi C_{i-1} e D_{i-1} di 28 bit ciascuno ai quali viene applicato uno shift circolare sinistro di due posizioni su ciascun bit in funzione del round. Vedi Fig. 12 **c**

3. **Compressione Permutazione:** Dei 56 bit totali ne vengono selezionati 48 mediante una funzione *compressione permutazione* (CP). Vedi Fig. 12 **b** Per esempio il bit in posizione 33 della chiave shiftata viene spostato nella posivione 35 dell'output e il bit in posizione 18 della chiave shiftata viene ignorato.

(a) Permuted Choice One (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

(b) Permuted Choice Two (PC-2)

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

(c) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figura 12: Trasformazioni del DES alla chiave segreta.

1.4.3 Algoritmo Rijndael (AES)

***La nascita di un nuovo standard*:** Il 2 Gennaio 1997, l'Istituto Nazionale per gli Standard e la Tecnologia degli Stati Uniti (NIST) dichiarò di aver iniziato lo sviluppo di un nuovo cifratore a blocchi a chiave simmetrica che avrebbe rappresentato l'evoluzione del DES, ed il suo nome sarebbe stato "Advanced Encryption Standard" (AES).

Contrariamente a quanto era avvenuto per le fasi di progettazione del DES, che si erano svolte in un'atmosfera di grande segretezza, AES si avvalse della famosa *call for paper* pubblica del 12 Settembre 1997, che stabiliva le specifiche fondamentali che il nuovo algoritmo avrebbe dovuto rispettare:

- essere reso pubblico in tutte le sue parti;
- i blocchi essere lunghi almeno 128 bit;
- le chiavi essere lunghe 128, 192 o 256 bit;
- la robustezza essere almeno paragonabile a quella del Triple DES, ma più efficiente da calcolare.

Il 20 Agosto 1998, il NIST annunciò di aver selezionato un gruppo di 15 algoritmi candidati per l'AES scelti fra una gran quantità di progetti provenienti da sviluppatori di tutte le parti del mondo.

Il primo round di analisi e valutazione si chiuse il 15 Aprile 1999 con il restringimento a 5 candidati, cioè MARS, RC6, Rijndael, SERPENT e TWOFISH; durante il secondo round, questi finalisti furono ulteriormente analizzati, con particolare attenzione ad i seguenti aspetti:

- Crittoanalisi
- Proprietà intellettuale
- Valutazione delle performance
- Dettagli implementativi
- Considerazioni di carattere generale

Il secondo round terminò il 15 Maggio 2000 con la proclamazione del vincitore da parte del NIST: si trattava dell'algoritmo Rijndael proposto da DAEMEN e RIJMEN, due famosi crittologi belgi.

***Cifratura*:** Rijndael è un cifratore a blocchi di lunghezza variabile; anche la chiave non è di lunghezza fissa, ma può essere lunga 128, 192 o 256 bit; per semplicità, tratteremo soltanto il caso in cui sia il blocco che la chiave sono lunghi 128 bit, ma i concetti di base rimangono invariati.

Il blocco e la chiave vengono divisi in segmenti di 16 bytes (un byte contiene 8 bit, infatti $128 = 16 \times 8$):

- InputBlock $= m_0, m_1, \ldots, m_{15}$.
- InputKey $= k_0, k_1, \ldots, k_{15}$.

Questi dati vengono memorizzati nelle seguenti matrici 4×4:

$$\text{InputBlock} = \begin{pmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{pmatrix}$$

$$\text{InputKey} = \begin{pmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{pmatrix}$$

Come il DES, anche l'AES esegue ripetutamente una trasformazione, dove ogni iterazione prende il nome di *round* e nel caso del blocco a 128 bit, il numero totale di iterazioni è 10; ogni trasformazione viene indicata con **Round**$(State, RoundKey)$.

$State$ è una matrice che viene trattata sia come input che come output; $RoundKey$ è la matrice che viene derivata dalla chiave; ogni round modifica gli elementi di $State$ e nel primo round della cifratura l'input è l'$InputBlock$, cioè la matrice del messaggio in chiaro, mentre l'output dell'ultimo round è la matrice del messaggio cifrato.

Le trasformazioni eseguite in tutti i round escluso l'ultimo sono composte da quattro diverse sotto-trasformazioni:

Round$(State, RoundKey)$
{

1. SubBytes($State$);
2. ShiftRows($State$);
3. MixColumns($State$);
4. AddRoundKey($State, RoundKey$);

}

L'ultimo round, invece, è rappresentato da **FinalRound**$(State, RoundKey)$, che esegue le stesse trasformazioni di **Round**$(State, RoundKey)$, esclusa la funzione MixColumns$(State, RoundKey)$; questo è simile all'ultimo round del DES, in cui vi è uno "swap" aggiuntivo tra le due metà del blocco.

Le trasformazioni del round sono invertibili per garantire la decifratura; le corrispondenti funzioni inverse si indicano con **Round**$^{-1}(State, RoundKey)$ e **FinalRound**$^{-1}(State, RoundKey)$. Passiamo adesso all'analisi delle funzioni interne dell'algoritmo Rijndael.

***Le 4 funzioni interne*:** Le trasformazioni sono ristrette al campo finito generato dal seguente polinomio irriducibile:

$$f(x) = x^8 + x^4 + x^3 + x + 1 \quad (\text{in } \mathbb{F}_2)$$

Ogni elemento appartenente a questo campo è un polinomio in $\mathbb{F}_2$ di grado minore di 8 e tutte le operazioni su di esso avvengono modulo $f(x)$; questo campo prende il nome di *campo di Rijndael*. Considerando l'isomorfismo ad esso relativo, questo campo viene spesso indicato con $\mathbb{F}_{2^8}$, che contiene $2^8 = 256$ elementi.

Come abbiamo visto in precedenza, un blocco del messaggio (uno "stato") ed un blocco della chiave sono segmentati in bytes, ed i loro bit appartengono al *campo di Rijndael*.

SubBytes$(State)$**:** Questa funzione esegue una sostituzione non lineare su ciascun byte di $State$; ogni byte $x \in (\mathbb{F}_{2^8})^*$ diverso da 0 viene sostituito secondo la trasformazione $y = A \cdot x^{-1} + b$, dove

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{e} \quad b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Se x vale 0, allora $y = b$ è il risultato di SubBytes(); è da notare che la non linearità deriva esclusivamente dall'inversione x^{-1} e siccome la matrice A è invertibile (le sue righe sono linearmente indipendenti) anche l'intera trasformazione è invertibile.

ShiftRows(*State*): Questa funzione opera su ciascun elemento $s_{i,j}$ di *State* e per il caso blocchi da 128 bit corrisponde alla seguente permutazione:

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \Longrightarrow \begin{pmatrix} s_{0,0} & s_{1,0} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

Praticamente di tratta di un cifrario a trasposizione che per ogni riga i esegue uno shift ciclico a destra di $4 - i$ posizioni.

Mixcolumns(*State*)**:** Questa funzione viene applicata alle 4 colonne di *State*; indichiamo ogni colonna con:

$$S_j = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

ed interpretiamo ciascuna colonna come il polinomio di terzo grado

$$S_j(x) = s_3 \cdot x^3 + s_2 \cdot x^2 + s_1 \cdot x + s_0$$

I coefficienti di $S_j(x)$ sono bytes in $\mathbb{F}_{2^8}$, quindi il polinomio stesso appartiene al campo $\mathbb{F}_{2^8}$.

L'operazione che si esegue sulla colonna j consiste nel moltiplicare $s(x)$ per il polinomio $c(x)$, modulo $x^4 + 1$:

$$f(x) = c(x) \cdot S_j(x) \quad \left(\mathrm{mod}\ x^4 + 1\right)$$

dove

$$c(x) = c_3\, x^3 + c_2\, x^2 + c_1\, x + c_0 = {}'03'\, x^3 + {}'01'\, x^2 + {}'01'\, x + {}'02'$$

Anche i coefficienti di $c(x)$ appartengono a $\mathbb{F}_{2^8}$ e sono espressi in esadecimale, ma dato che $c(x)$ e $s(x)$ non appartengono al *campo di Rijndael*, nemmeno il loro prodotto ne fa parte; eseguendo il prodotto modulo un polinomio di quarto ordine si ottiene un polinomio di terzo grado, che permette quindi di operare una trasformazione da una colonna in un'altra, entrambe polinomi di terzo grado.

Fatte queste considerazioni, è facile capire che il prodotto di polinomi può

essere espresso dalla seguente combinazione lineare:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ c_3 \end{pmatrix}$$

$$= \begin{pmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ c_3 \end{pmatrix}$$

Inoltre, tenendo conto che $c(x)$ è sempre primo rispetto a $x^4 + 1$ su $\mathbb{F}_2$, l'inversione $c^{-1}(x) \pmod{x^4 + 1}$ esiste in $\mathbb{F}_{2\,[x]}$; questo equivale ad affermare che la matrice, e quindi l'intera trasformazione, è invertibile.

AddRoundKey$(State, RoundKey)$**:** Questa funzione è semplicemente una somma bit a bit degli elementi di $RoundKey$ con quelli di $State$; stavolta l'addizione è uno XOR che per definizione opera in $\mathbb{F}_2$, quindi perfettamente invertibile.

I bit di $RoundKey$ subiscono un processo di "schedulazione" da un round all'altro, cioè vengono trasformati in base alla chiave secondo uno schema reso pubblico dagli autori dell'algoritmo; per ulteriori dettagli su questa fase, fare riferimento alla documentazione ufficiale dell'AES.

Terminata quest'ultima fase, il processo di cifratura è finito e l'output finale corrisponde al messaggio cifrato.

***Decifratura*:** Abbiamo visto che le 4 operazioni di ogni round sono invertibili, quindi la decifratura consisterà nell'applicare in ordine inverso le corrispondenti funzioni inverse:

Round$^{-1}(State, RoundKey)$
{

1. AddRoundKey$^{-1}(State, RoundKey)$;
2. MixColumns$^{-1}(State)$;
3. ShiftRows$^{-1}(State)$;
4. SubBytes$^{-1}(State)$;

}

Differentemente dal cifrario di Feistel in cui la cifratura e la decifratura si basano lo stesso circuito (hardware) o codice (software), il cifrario di Rijndael deve implementare separatamente le trasformazioni dirette e inverse.

***Considerazioni generali*:** Qui di seguito è riportato un riassunto del ruolo svolto da ciascuna delle quattro funzioni interne:

SubBytes: permette di realizzare un cifrario a sostituzione non lineare che rende l'algoritmo inattaccabile alla crittoanalisi differenziale.

ShiftRows, MixColumns: consentono di ottenere un rimescolamento dei bytes del test in chiaro; in genere, i messaggi hanno un basso livello di entropia a causa dell'alta ridondanza insita nel linguaggio naturale. Cambiare le posizioni dei bytes dà origine ad una più ampia distribuzione dei caratteri nell'alfabeto e questo rappresenta un soddisfacimento della proprietà di *distribuzione* enunciata da SHANNON.

AddRoundKey: fornisce l'indispensabile distribuzione casuale dei simboli nello spazio del messaggio.

Queste funzioni vengono ripetute varie volte a partire da un minimo di 10 per il caso a 128 bit ed l'insieme di queste operazioni costituisce il cifratore AES; la seguente figura ne riporta lo schema generale di funzionamento.

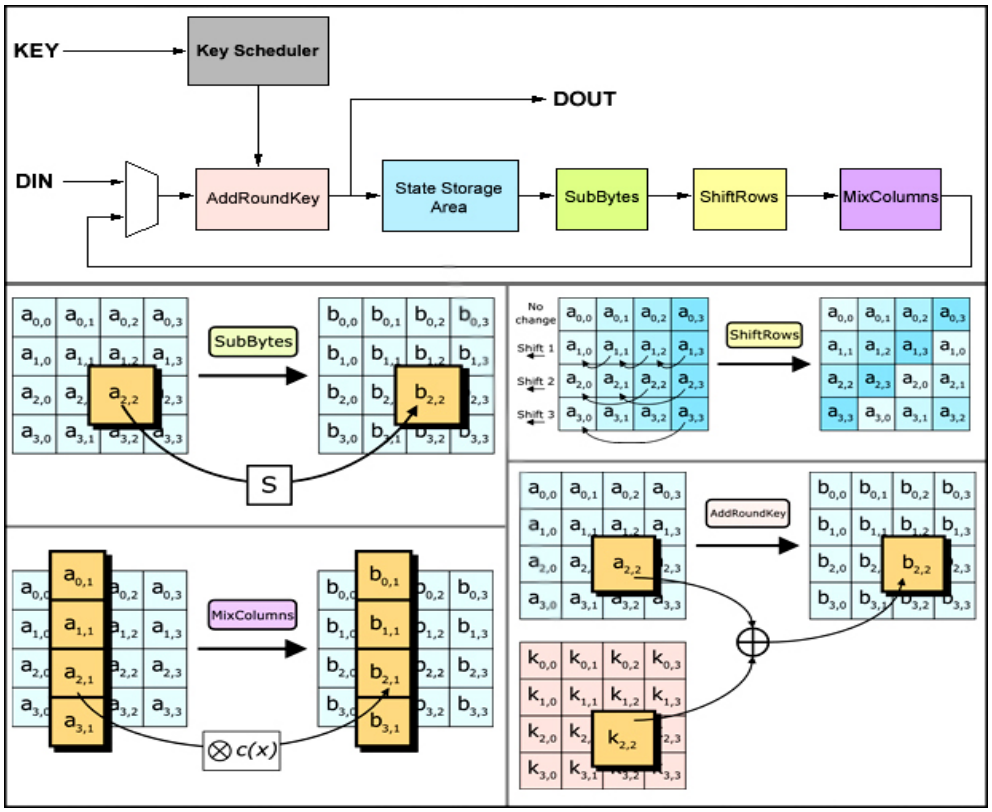

Figura 13: Schema generale di funzionamento dell'algoritmo AES.

2 Crittografia basata su dizionario

Dopo questo breve excursus sullo stato dell'arte dei crittosistemi simmetrici, passiamo all'esposizione dell'algoritmo da noi proposto; la nostra idea è stata quella di progettare un cifratore simmetrico il cui "alfabeto" è un dizionario costituito da parole generiche e/o relative ad un argomento specifico.

A differenza dei cifratori attualmente in uso che considerano ogni carattere del testo in chiaro come un "simbolo" a sé stante, il nostro approccio è basato sulla scomposizione del messaggio in parole e sulla codifica della loro *posizione* in una lista di parole prestabilita (dizionario).

Sebbene a prima vista questa soluzione possa non sembrare del tutto efficiente (ogni parola diventa un numero che va in genere ad incrementare la lunghezza del messaggio cifrato), il guadagno reale si ha in termini di robustezza; infatti, diversamente dagli altri cifratori simmetrici in cui una chiave sbagliata restituisce un messaggio in genere completamente inintelligibile, decifrando con il nostro algoritmo un testo cifrato con una chiave sbagliata, si otterrà sempre e comunque una frase contenente parole appartenenti al dizionario usato.

Il nostro algoritmo rientra nella categoria dei cifrari a blocchi di lunghezza variabile, dove però ogni blocco è costituito da una frase del testo in chiaro; questa modalità trova la sua applicazione ideale a testi generici (articoli, email, libri, ecc...), ma può essere adattata anche a qualsiasi altro testo (codici fiscali, formule, simboli matematici, ecc...), a patto di fornire un dizionario adeguato.

Passiamo ora a vedere più in dettaglio le varie fasi che compongono la modalità operativa di questo algoritmo.

2.1 Cifratura

2.1.1 Chiavi di permutazione

Il crittosistema è simmetrico, quindi si basa sull'uso della stessa chiave sia per la cifratura che per la decifratura; per garantire una buona sicurezza dell'algoritmo, abbiamo deciso di utilizzare due chiavi private a 64 bit, una primaria k_1 ed una secondaria k_2, che possono anche essere considerate come un'unica chiave a 128 bit risultante dalla concatenazione $[k_1, k_2]$, in modo da associare le cifre più significative a quella primaria.

Una volta stabilito il dizionario di riferimento, da esso vengono ricavate le posizioni (ID) delle parole del testo in chiaro e concatenate in un'unica stringa da cifrare; ogni chiave svolge una funzione distinta all'interno dell'algoritmo:

Chiave primaria: permuta l'ordine delle cifre della stringa risultante dal concatenamento di tutti gli ID.

Chiave secondaria: permuta l'ordine delle parole prima di utilizzare la chiave primaria sulle cifre degli ID.

2.1.2 Preprocessing del dizionario

Data una lista di N parole i cui ID hanno al massimo d cifre, è necessario far sì che qualsiasi numero $0 \leq x \leq 10^d - 1$ sia un ID valido, cioè "punti" ad una parola presente nel dizionario.

Per garantire questa proprietà, abbiamo deciso di replicare le w_i parole fino a riempire $N' = 10^d$ posizioni di un dizionario "maggiorato" utilizzato sia durante la cifratura che la decifratura:

$$dizionario = \overbrace{\{\underbrace{w_0, w_1, \ldots, w_{N-1}}_{\text{ID originali } (N)}, \underbrace{w_0, w_1, \ldots, w_{N'-N-1}}_{\text{ID } \textit{duplicati } (N'-N)}\}}^{\underline{\textbf{ID totali}}\ (N')}$$

Se una sola ripetizione non è sufficiente a riempire tutte le N' posizioni, ovvero quando $N < N'/2$, le parole originali continuano ad essere replicate fino a che non si è raggiunta l'ultima posizione $(N' - 1)$.

Nel caso in cui una parola corrisponda a più di un ID, uno qualsiasi di essi viene associato ad essa; come vedremo meglio nel prossimo paragrafo, questo espediente impedisce ad un crittoanalista di scartare a priori gli ID che cadono nell'intervallo $[N, N']$, perché con questo accorgimento in decifratura ogni indice $id + \alpha \cdot N \pmod{N'}$ (con $\alpha \in \mathbb{N}$) "punta" alla stessa parola.

2.1.3 Concatenamento e permutazione

Per rendere più pulita la struttura dell'algoritmo, abbiamo implementato la funzione $S(v, seed)$ che esegue una permutazione dei q elementi del vettore v in base al valore di *seed*.

Questa funzione si basa sulla generazione di q numeri casuali compresi nell'intervallo $[0, q-1]$ usando *seed* come seme generativo (internamente viene usata $F(dim, seed)$ che genera un vettore random di dim elementi basati su *seed*); il vettore v' composto da tali numeri rappresenta la permutazione di v (dati due semi uguali, il risultato della permutazione è garantito essere il medesimo).

Lo spazio delle chiavi è generalmente molto maggiore delle $q!$ permutazioni possibili dei q elementi, quindi un seme diverso da *seed* genera *tendenzialmente* un vettore $v'' \neq v'$, e possiamo dire che $S(v, seed)$ rappresenta una sorta di "one-way trap-door function"; in realtà, la probabilità che due semi diversi *seed* e *seed*$'$ generino la stessa permutazione è inversamente proporzionale a q.

Tornando alla cifratura, se la chiave secondaria $k_2 \neq 0$, le parole estratte dal testo in chiaro vengono permutate in base al valore di k_2; supponendo che il testo in chiaro m sia composto da p parole, la permutazione avrà questa forma:

$$\begin{cases} m &= w_1, w_2, \ldots, w_p \\ m' &= S(m, k_2) \end{cases}$$

Dopo aver concatenato nella stringa m' gli ID delle p parole eventualmente permutate tramite k_2, si esegue un'ulteriore permutazione delle sue cifre basandosi sulla chiave primaria k_1; il risultato sarà il messaggio cifrato c:

$$c = S(m', k_1)$$

2.2 Decifratura

Una volta recuperata la lunghezza degli ID in funzione della dimensione del vocabolario, per poter ricostruire il messaggio in chiaro a partire da quello cifrato è necessario avere a disposizione entrambe le chiavi e nel processo inverso si deve prima risalire al messaggio m':

$$m' = S^{-1}(c, k_1)$$

La funzione S^{-1} consiste nel recuperare la permutazione $S(m', k_1)$ generando il vettore v' della permutazione da applicare alle cifre in c per ottenere m'; successivamente vengono recuperate le parole corrispondenti agli ID a cui applicare l'inverso di $S(m, k_2)$ per avere come risultato m.

$$m = S^{-1}(m', k_2)$$

La seguente figura riporta lo schema funzionale dell'intero crittosistema.

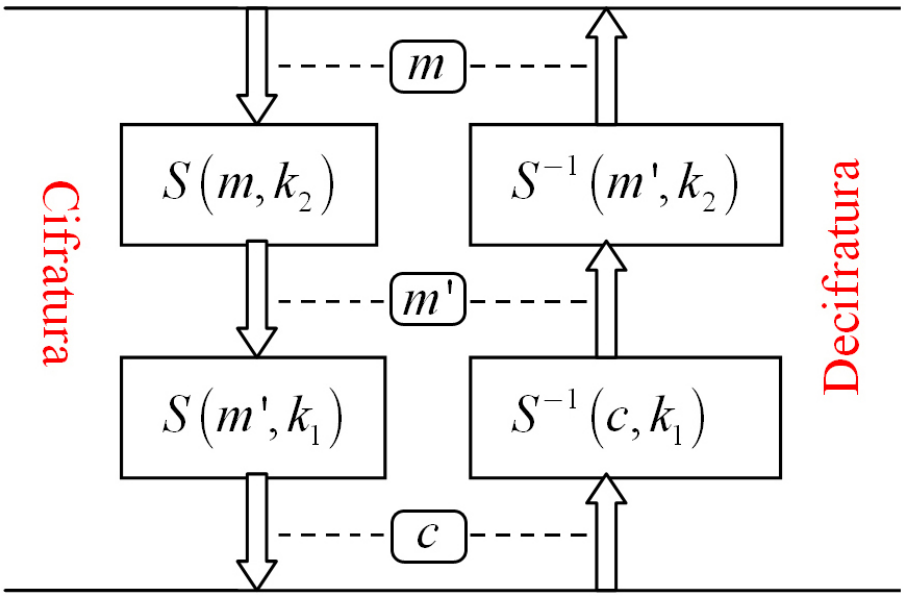

Figura 14: Schema funzionale dell'algoritmo *WordCrypt*.

2.3 Esempio di applicazione

Applichiamo il nostro algoritmo alla frase $m =$"`oggi vado al mare`" e seguiamo tutti i passi delle procedure di cifratura e di decifratura.

2.3.1 Cifratura

Supponiamo di avere a disposizione un vocabolario della lingua italiana composto da $N = 245.231$ parole, espresse per ogni genere e numero[1], comprese le preposizioni articolate ed i verbi coniugati (questa è la lista di parole che abbiamo usato per tutti i test qui presentati [file: `italiano.dat`]); N è formato da 6 cifre, quindi N' sarà pari a $10^6 = 1.000.000$ e gli ID hanno valori compresi tra 0 e 999.999, con eventuale *padding* di zeri a partire da sinistra.

La funzione che recupera gli ID delle parole dal dizionario restituisce in questo caso le seguenti posizioni:

$$\begin{cases} \texttt{oggi} &= 116904 \\ \texttt{vado} &= 238595 \\ \texttt{al} &= 005536 \\ \texttt{mare} &= 100409 \end{cases}$$

Come chiave primaria scegliamo $k_1 = 12345$ e come chiave secondaria $k_2 = 1$; la funzione $S(m, k_2)$ genera il vettore

$$v' = [3, 0, 1, 2]$$

quindi la permutazione di m sarà "`vado al mare oggi`" e la concatenazione degli ID corrisponde a

$$m' = \underbrace{238595}_{\texttt{vado}}\underbrace{005536}_{\texttt{al}}\underbrace{100409}_{\texttt{mare}}\underbrace{116904}_{\texttt{oggi}}$$

A questo punto si usa k_1 per permutare le 24 cifre di m', cioè si applica la funzione $S(m', k_1)$; internamente la funzione $F(24, 12345)$ restituisce

$$v'' = [12, 4, 20, 23, 0, 14, 18, 13, 21, 16, 11, 10, 5, 1, 9, 3, 22, 7, 8, 15, 6, 17, 2, 19]$$

e la permutazione finale sarà

$$c = \texttt{900431691063205159048505}$$

che rappresenta il nostro messaggio cifrato.

[1] **Genere** = maschile o femminile; **Numero** = singolare o plurale.

2.3.2 Decifratura

Dato c, applichiamo le operazioni descritte nel precedente paragrafo; supponiamo di essere il destinatario autorizzato del messaggio e di possedere quindi le due chiavi private k_1 e k_2.

$$\begin{cases} c &= 900431691063205159048505 \\ k_1 &= 12345 \\ k_2 &= 1 \end{cases}$$

Basandosi sulla dimensione N' del dizionario pubblico, possiamo ricavare la lunghezza d di ogni ID, che in questo caso sarà

$$\begin{cases} N' &= 1.000.000 \\ d &= \log_{10}(N') = 6 \end{cases}$$

Adesso dobbiamo calcolare $S^{-1}(c, k_1)$ e ci serviamo della stessa funzione F usata in cifratura; il messaggio è lungo 24 caratteri, quindi è composto da 4 parole i cui ID sono lunghi ciascuno 6 cifre. Generiamo il vettore v'' tramite la chiave primaria $k_1 = 12345$ con la funzione $F(24, 12345)$, il cui risultato sarà uguale a quello usato in cifratura:

$$v'' = [12, 4, 20, 23, 0, 14, 18, 13, 21, 16, 11, 10, 5, 1, 9, 3, 22, 7, 8, 15, 6, 17, 2, 19]$$

Dopodiché effettuiamo la permutazione inversa di c tramite v'' e ricaviamo

$$m' = \underbrace{238595}_{id_0}\underbrace{005536}_{id_1}\underbrace{100409}_{id_2}\underbrace{116904}_{id_3}$$

Segmentando m' nei 4 ID che lo compongono, otteniamo il vettore che servirà per calcolare $S^{-1}(m', k_2)$; generiamo v' tramite la chiave secondaria $k_2 = 1$:

$$v' = [3, 0, 1, 2]$$

che serve per permutare in senso inverso m' ed ottenere

$$m = \underbrace{116904}_{\texttt{oggi}}\underbrace{238595}_{\texttt{vado}}\underbrace{005536}_{\texttt{al}}\underbrace{100409}_{\texttt{mare}}$$

cioè il messaggio in chiaro originale.

2.4 Dizionario

2.4.1 Composizione

Risulta chiaro che la composizione del dizionario influisce in gran misura sulla cifratura, in quanto le parole in esso contenute determinano il livello di entropia lessicale delle frasi incontrate in crittoanalisi; se per assurdo esso contenesse una sola parola, gli ID sarebbero solo 10 e la ripetizione di quella parola costituirebbe lo spazio del testo in chiaro.

Il dizionario che abbiamo utilizzato durante le varie fasi di test è molto generico e contiene numerosi termini poco comuni che potrebbero essere eliminati per rendere più difficile la crittoanalisi (più frasi con parole "sensate"); tuttavia, un metodo più efficiente è quello di fondere varie liste in fase di cifratura e comunicare tale operazione anche al destinatario del messaggio.

Se analizziamo le parole contenute in un testo campione, possiamo renderci conto che la maggior parte di esse rientra in una lista di termini generici peculiari della lingua in uso, come ad esempio gli articoli, le preposizioni, gli avverbi, ecc. . . ; gli altri termini del testo saranno invece più specifici ed appartenenti a argomenti particolari, come ad esempio termini informatici, militari, medici, ecc. . . .

E' quindi ragionevole pensare di avere a disposizione un dizionario generico basato sulla maggior parte dei testi di comune impiego, ma anche una serie di dizionari specifici appositamente preparati per i vari argomenti: in fase di cifratura, il testo verrebbe analizzato tenendo traccia del numero di parole che rientrano nei dizionari specifici (ognuno di essi riceverebbe uno *score*), dopodiché si potrebbe "fondere" il dizionario generico con quelli aventi lo *score* più alto.

Questo modello di gestione dei dizionari potrebbe anche costituire un servizio centralizzato su Web tramite il quale accedere pubblicamente alle liste di parole e scegliere quelle più opportune prima di cifrare il testo; in fase di decifratura, si accederebbe al medesimo servizio per utilizzare gli stessi dizionari.

La seguente figura mostra lo schema di costruzione del dizionario finale.

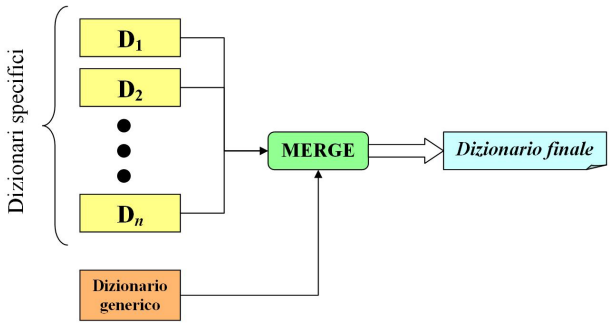

Figura 15: Schema di costruzione del dizionario usato dal crittosistema.

2.4.2 Parole “esterne”

La struttura dell'algoritmo consente anche di far fronte all'eventualità che il testo in chiaro comprenda parole non contenute del dizionario:

1. durante l'analisi preliminare della frase si aggiungono le parole non contenute nel dizionario alla lista di parole caricate da file e si esegue il preprocessing come in § 2.1.2;
2. si cifra la frase normalmente come esposto in § 2.1;
3. le parole aggiuntive vengono scritte in un file ASCII, una per riga;
4. ad ogni carattere di questo file viene applicato uno XOR con la chiave primaria ed il risultato viene inviato insieme al messaggio cifrato;
5. prima di decifrare il messaggio, il sistema esegue nuovamente lo XOR sul file aggiuntivo recuperando le parole aggiuntive usate in cifratura ed aggiornando il dizionario;
6. la decifratura avviene regolarmente come spiegato in § 2.3.2.

La natura “simmetrica” dello XOR fa sì che questo procedimento sia del tutto invertibile e sebbene un malintenzionato possa eseguire un attacco a forza bruta sul file di corredo per recuperare parole che di sicuro appartengono al testo in chiaro, deve necessariamente esplorare tutti i 64 bit della chiave primaria, cioè $(2^{63} - 1)/2 \approx 4.611 \times 10^{18}$ tentativi in media.

La seguente figura riporta lo schema del procedimento sopra esposto.

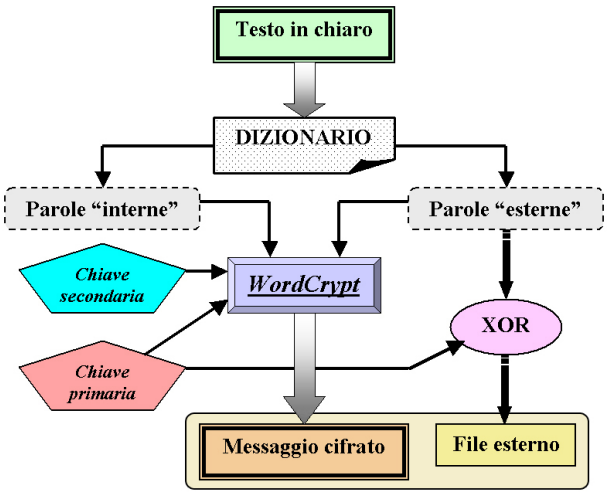

Figura 16: Schema di gestione delle parole non appartenenti al dizionario.

2.5 Crittoanalisi

2.5.1 Attacco sul messaggio cifrato

Dato un messaggio cifrato c di lunghezza l, è possibile generare le $l!$ permutazioni delle sue cifre ed analizzare le frasi ad esse corrispondenti; se $l!$ è minore dello spazio delle chiavi ($64\,\text{bit} + 64\,\text{bit} = 128\,\text{bit} = (2^{63} - 1)^2 \approx 8.5 \times 10^{37}$), quest'attacco diventa più conveniente di quello sullo spazio delle chiavi.

Per entrambi questi approcci, è tuttavia necessario un sistema in grado di valutare la correttezza grammaticale di ogni risultato per poter scegliere delle frasi candidate ad essere il testo in chiaro originale.

2.5.2 Attacco sullo spazio delle chiavi

La modalità di attacco più attuabile risulta essere quella a forza bruta sullo spazio delle chiavi, in quanto è impossibile risalire direttamente dal messaggio cifrato c al testo in chiaro m, poiché abbiamo visto che le funzioni usate in codifica sono "difficilmente" invertibili.

tramite analisi grammaticale: La robustezza di questo algoritmo risiede proprio nella difficoltà di riconoscere automaticamente la frase corretta tra l'insieme di quelle generate durante un attacco a forza bruta; una chiave sbagliata porta a ricostruire erroneamente le cifre degli ID ed ottenere in tal modo una frase con parole diverse dal testo in chiaro.

Facendo ancora riferimento al messaggio cifrato c ottenuto nell'esempio precedente, se utilizziamo come chiavi $k_1 = 54321$ (errata) e $k_2 = 2$ (errata), otteniamo la seguente frase, che ovviamente non corrisponde a quella originale:

$$m = \texttt{ricolmai ricovrire additeremo baldanzosa}$$

Le frasi ottenute, per quanto spesso grammaticalmente non accettabili, in generale potrebbero esserlo cambiando l'ordine delle parole, ma questo comporterebbe comunque un'analisi grammaticale di tutte le possibili permutazioni per scartare quelle non valide.

Se invece riusciamo a ricavare la chiave primaria, abbiamo comunque il problema di determinare l'ordine corretto delle parole determinato da k_2; nell'esempio citato, decodificando c con $k_1 = 12345$ (corretta) e $k_2 = 2$ (errata), otteniamo

$$m = \texttt{oggi al vado mare}$$

che grammaticalmente non ha senso, ma siccome le sue parole fanno parte del messaggio in chiaro originale è del tutto indistinguibile da una qualsiasi frase sbagliata, a patto di non indovinare anche la chiave secondaria.

tramite ricerca su Web: Un altro possibile approccio alla crittoanalisi di questo algoritmo è quello di sfruttare la ricerca sul Web per poter valutare la correttezza grammaticale e semantica delle frasi che si ottengono durante la prova a forza bruta delle chiavi; questo tipo di attacco si divide in due fasi distinte.

***forzatura della chiave primaria*:** Trovare la giusta k_1 equivale a recuperare le parole che appartengono al messaggio in chiaro, ma senza conoscerne l'ordinamento originale; per stimare la probabilità con cui un'insieme di parole fa parte di una frase possiamo immetterle come query su un motore di ricerca.

Prendiamo come esempio la medesima frase "`oggi vado al mare`" cifrata con $k_1 = 12345$ e $k_2 = 1$, e riportiamo le corrispondenti frasi decifrate fissando la chiave secondaria a 0, ma facendo variare le chiavi primarie da 12340 a 12350 (per semplicità, facciamo riferimento al dizionario generico):

$$\begin{cases} 12340 \implies \text{“}\texttt{suonasse scavalcassi brachicefalia lucerne}\text{”} \\ 12341 \implies \text{“}\texttt{addentrati connaturale muraglie ricollocava}\text{”} \\ 12342 \implies \text{“}\texttt{riallineamento guadagnerai righero soggiornera}\text{”} \\ 12343 \implies \text{“}\texttt{materiate maledicevamo aeriforme follatura}\text{”} \\ 12344 \implies \text{“}\texttt{fonotelegrafia latinizzamento strappalana schermiva}\text{”} \\ \boxed{12345} \implies \text{“}\texttt{vado al mare oggi}\text{”} \\ 12346 \implies \text{“}\texttt{generalizza aristotelismi procreo bambagine}\text{”} \\ 12347 \implies \text{“}\texttt{slacciandosi grandinifugo broccio ripassassi}\text{”} \\ 12348 \implies \text{“}\texttt{seziona accadra acidificazioni neurone}\text{”} \\ 12349 \implies \text{“}\texttt{imbrana inclusa emerografia evacuiate}\text{”} \\ 12350 \implies \text{“}\texttt{percepirete appendo judoiste fischiato}\text{”} \end{cases}$$

Se inseriamo ciascun risultato in un motore di ricerca[2] come serie di termini in OR[3] possiamo considerare il numero di documenti o pagine web individuati come la probabilità di trovare tali parole nel medesimo contesto; nonostante l'apparente immediatezza di questo tipo di attacco, il numero di documenti recuperati è inversamente proporzionale al numero di parole immesse, quindi più la frase è lunga, più è difficile ottenere risultati rilevanti.

Per quanto riguarda le prestazioni, l'accesso ad un motore di ricerca offline è sicuramente più efficiente di quello fatto ad un server remoto (overhead della rete, ritardi dovuti alla congestione, ecc...), anche se risulta essere la soluzione più diffusa ed accessibile alla maggior parte dei sistemi.

[2]Tutti i risultati riportati in questa relazione fanno riferimento ai documenti recuperati dal motore di ricerca Google.

[3]I motori di ricerca più diffusi usano questo tipo di ricerca come default se la frase non viene racchiusa tra apici.

***forzatura della chiave secondaria*:** Dopo aver individuato la chiave primaria, e quindi anche le parole che appartengono al testo in chiaro, è necessario recuperare la chiave secondaria, cioè il loro corretto ordinamento; sebbene k_2 vari tra 0 e $2^{63} - 1$, il numero totale delle permutazioni di p parole equivale a $p!$, che in genere è molto minore dello spazio della chiave secondaria.

Anche in questo caso, è necessario un attacco a forza bruta per ricavare le permutazioni delle parole ricavate tramite un chiave primaria; supponiamo di aver indovinato $k_1 = 12345$ e di voler trovare k_2 elencando le $4! = 24$ possibili frasi f_i composte dalle parole "`vado`", "`al`", "`mare`" e "`oggi`":

$$\begin{cases} f_0 \implies \text{“}\texttt{vado al mare oggi}\text{”} \\ f_1 \implies \text{“}\texttt{vado al oggi mare}\text{”} \\ f_2 \implies \text{“}\texttt{vado mare al oggi}\text{”} \\ f_3 \implies \text{“}\texttt{vado mare oggi al}\text{”} \\ f_4 \implies \text{“}\texttt{vado oggi al mare}\text{”} \\ f_5 \implies \text{“}\texttt{vado oggi mare al}\text{”} \\ f_6 \implies \text{“}\texttt{al vado mare oggi}\text{”} \\ f_7 \implies \text{“}\texttt{al vado oggi mare}\text{”} \\ f_8 \implies \text{“}\texttt{al mare vado oggi}\text{”} \\ f_9 \implies \text{“}\texttt{al mare oggi vado}\text{”} \\ f_{10} \implies \text{“}\texttt{al oggi vado mare}\text{”} \\ f_{11} \implies \text{“}\texttt{al oggi mare vado}\text{”} \end{cases} \quad \begin{cases} f_{12} \implies \text{“}\texttt{mare vado al oggi}\text{”} \\ f_{13} \implies \text{“}\texttt{mare vado oggi al}\text{”} \\ f_{14} \implies \text{“}\texttt{mare al vado oggi}\text{”} \\ f_{15} \implies \text{“}\texttt{mare al oggi vado}\text{”} \\ f_{16} \implies \text{“}\texttt{mare oggi vado al}\text{”} \\ f_{17} \implies \text{“}\texttt{mare oggi al vado}\text{”} \\ f_{18} \implies \text{“}\texttt{oggi vado al mare}\text{”} \\ f_{19} \implies \text{“}\texttt{oggi vado mare al}\text{”} \\ f_{20} \implies \text{“}\texttt{oggi al vado mare}\text{”} \\ f_{21} \implies \text{“}\texttt{oggi al mare vado}\text{”} \\ f_{22} \implies \text{“}\texttt{oggi mare vado al}\text{”} \\ f_{23} \implies \text{“}\texttt{oggi mare al vado}\text{”} \end{cases}$$

Per valutare la correttezza di queste frasi, e individuare dunque quella che più probabilmente corrisponde all'originale testo in chiaro, possiamo avvalerci di un motore di ricerca a cui passeremo questi risultati come serie di parole in AND[4]; il numero di documenti o di pagine web ricavate ci può dare un'idea del grado di coerenza grammaticale che possiede ciascun ordinamento.

Le Fig. 17 e 18 riportano i risultati ottenuti eseguendo le query in OR ed in AND per recuperare la chiave primaria e secondaria, rispettivamente.

[4]In genere, i motori di ricerca utilizzano questa modalità quando la frase immessa è interamente racchiusa tra apici.

Figura 17: Risultati ottenuti immettendo su Google i termini in OR per recuperare la chiave primaria.

Figura 18: Risultati ottenuti immettendo su Google i termini in AND per recuperare la chiave secondaria.

tramite statistiche su documenti: Avendo a disposizione un corpus significativo di documenti nella lingua usata in cifratura, è possibile valutare la probabilità che esse facciano parte della stessa frase. Ovviamente il calcolo delle probabilità congiunte richiederebbe uno sforzo computazionale non indifferente ed in ogni caso all'aumentare del numero di parole della frase tale probabilità tenderà a zero.

***definizioni e terminologia*:** Date due parole w' e w'' ed una frase s, definiamo $P(w', w''|s)$ la probabilità congiunta di trovare le due parole all'interno della frase s; tale valore si può stimare come il rapporto numero di frasi del dataset in cui w' e w'' compaiono insieme ed il numero di frasi in cui compare almeno una delle due parole.

Se ci vogliamo limitare a calcolare le probabilità congiunte di frasi composte da solo due parole, è chiaro che questo tipo di calcolo dovrebbe essere fatto per ogni coppia di parole; in generale questo non è sufficiente se consideriamo che l'algoritmo presentato agisce su frasi di lunghezza arbitraria e dunque le frasi decrittate possono essere ben più lunghe.

Inoltre, come possiamo dedurre dalle precedenti osservazioni, la probabilità congiunta tende a zero in funzione della lunghezza della frase, quindi un modo per superare questi limiti sarebbe quello di calcolare le probabilità congiunte solo con due, tre o al massimo quattro parole in funzione della capacità computazionale che si ha a disposizione; infatti, già con solo due parole, la complessità di calcolo di tutte le $P(w', w''|s)$ è di circa N'^2 e cresce esponenzialmente con il numero di parole del dizionario.

Per rendere più efficiente questo approccio dal punto di vista computazionale, si possono sfruttare alcuni concetti apaprtenenti al campo dell'*information retrieval*, ovvero un *search engine* minimale studiato ad hoc; a differenza di un tradizionale motore di ricerca, esso lavorerebbe ad una grana più fine rispetto al documento, ovvero l'entità fondamentale di questo sistema sarebbe la frase.

Costruendo un *indice inverso*[5] in cui al posto del documento abbiamo le frasi ed usando un *query server*[6] si potrebbe calcolare in maniera efficiente la metrica chiamata *resemblance*, che come vedremo costituisce un'ottima approssimazione delle probabilità congiunte che vogliamo calcolare.

La *resemblance* tra due insiemi A e B è definita come:

$$\frac{|A \cap B|}{|A \cup B|}$$

Date due parole w' e w'' di cui si vuole calcolare la probabiltà di trovarle nella

[5]Struttura dati in cui vengono memorizzate le informazioni necessarie ad un motore di ricerca per rispondere alle query e lo possiamo vedere come una sorta di indice analitico in cui sono contenute le posizioni delle parole all'interno dei documenti.

[6]Programma che si occupa di analizzare l'indice inverso e verificare quali sono i documenti che soddisfano la query, generalmente espressioni booleane di termini in AND e/o in OR.

stessa frase, questa può essere rappresentata mediante la resemblance delle liste di documenti che risultano dalle query (w' OR w'') e (w' AND w''); la resemblace tra w' e w'' risulterebbe:

$$\frac{\left|f\left([w', w'']\right)\right|}{\left|f(w', w'')\right|}$$

dove $f()$ indica l'elaborazione del query server (l'insieme dei documenti (frasi) che soddisfano la query) e $[w', w'']$ indica l'AND tra le parole, mentre w', w'' ne indica l'OR, esattamente come nei tradizionali motori di ricerca.

Calcolando la resemblance otterremmo il rapporto tra il numero di frasi del dataset in cui le due parole compaiono insieme (l'AND corrisponde all'intersezione tra le liste di frasi che contengono le parole) ed il numero totali di frasi in cui compare almeno una delle due parole (l'OR tra le liste dell'indice inverso corrispondenti alle due parole.).

Risulta piuttosto evidente che questo calcolo esprime molto bene il tipo di metrica che ci interessava cogliere; tuttavia essa richiede l'utilizzo di un search engine dedicato per ottenere in maniera efficiente queste statistiche che potrebbero anche essere calcolate in run-time, ovvero aumentando il numero di documenti della base documentale, non è necessario rifare tutte le statistiche mediante un preprocessing offline.

***applicazione al crittosistema*:** Una volta che è stata calcolata la resemblance tra tutte le parole presenti nelle frasi generate in decodifica, otteniamo uno score tramite il quale scartare le frasi con valore più basso, riuscendo a diminuire considerevolmente il numero di frasi da dover analizzare manualmente; infatti la frase corretta dovrebbe avere uno score elevato, in genere il maggiore fra quelli calcolati.

Nonostante questo approccio superi alcuni limiti dell'attacco tramite ricerca su Web di cui abbiamo parlato in § 2.5.2, perché indipendente dall'ordine in cui le parole compaiono nella frase, l'effetto della chiave secondaria viene ridotto, ma non del tutto annullato; in ogni caso, il problema delle frasi lunghe e delle probabilità congiunte che tendono a zero rimane aperto.

Concretizzare l'attacco tramite statistiche su documenti comporta comunque la messa a punto di un complesso sistema appositamente dedicato contenente numerosi moduli indispensabili per la realizzazione di un motore di ricerca e questo fattore di difficoltà implementativa incrementa in qualche modo la resistenza del crittosistema ad attacchi "tradizionali".

2.6 Benchmark

Escluse le operazioni indipendenti dalla quantità di dati, come ad esempio il preprocessing del dizionario, la nostra implementazione fa largo uso di funzioni di accesso a vettori (funzioni `indexOf()` e `elementAt()`) e permutazioni di array (funzione `shuffle()`) utilizzate sia in cifratura che in decifratura.

Queste operazioni hanno complessità lineare garantita dall'implementazione interna del linguaggio (JAVA™ nel nostro caso), quindi è lecito pensare che anche l'intero crittosistema abbia tale vantaggiosa caratteristica.

Per avvalorare questa supposizione, abbiamo effettuato dei test sulla velocità di cifratura in funzione del numero di parole e sulla decifratura in funzione del numero di chiavi primarie esaminate; tutti i test sono stati eseguiti su un PC con un AMD Athlon 3500+.

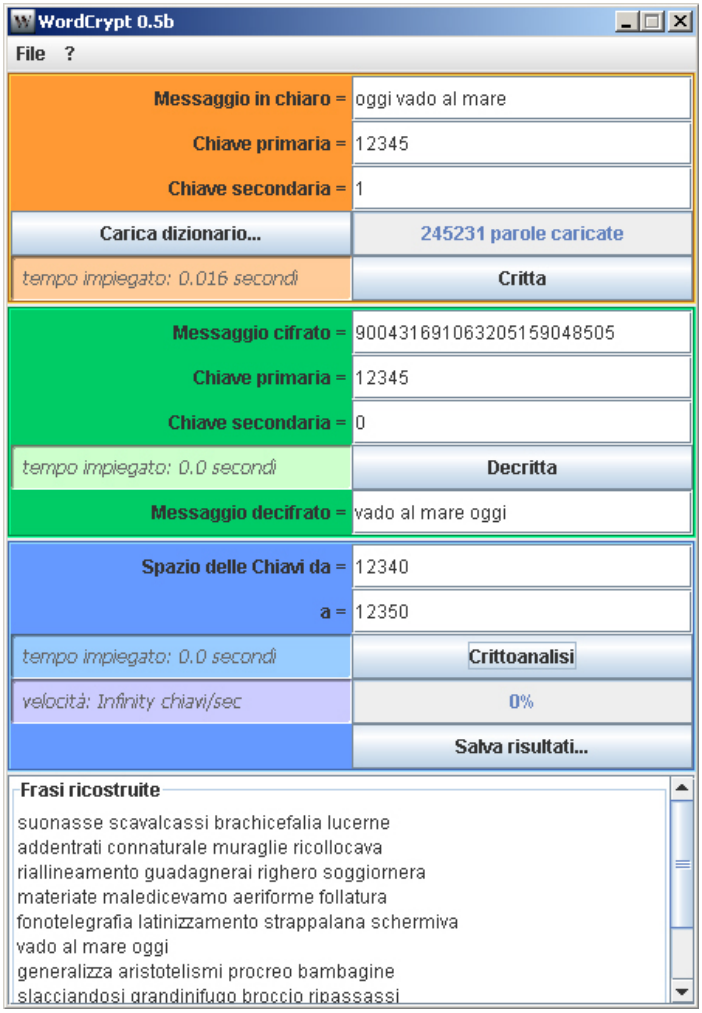

Figura 19: Screenshot dell'interfaccia grafica di *WordCrypt*.

Figura 20: Valutazione delle prestazioni della funzione `Crypt()`; la retta di colore verde rappresenta la regressione lineare che meglio approssima il rapporto tra tempo e numero di parole codificate.

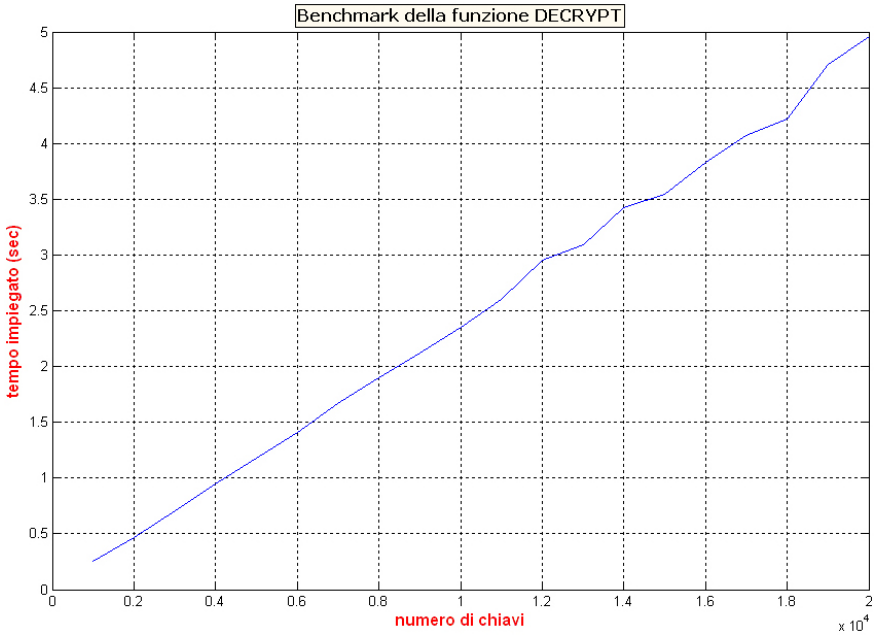

Figura 21: Valutazione delle prestazioni della funzione `Decrypt()`; senza bisogno di sovrapporre una regressione, risulta evidente la dipendenza lineare tra lo spazio delle chiavi esplorato ed il tempo impiegato.

2.7 Considerazioni finali

Come si può dedurre dagli esempi riportati, il grado di indecisione che si presenta in fase di crittoanalisi è molto alto a causa della filosofia di cifratura che sta alla base di questo algoritmo; alla luce di questo, possiamo quindi riassumere i seguenti aspetti relativi alla **sicurezza** ed alla **robustezza** del crittosistema:

- ogni operazione di decifratura genera messaggi contenenti parole che appartengono al dizionario fornito e quindi non scartabili a priori;
- durante un attacco a forza bruta non è possibile fare assunzioni sul testo in chiaro, in quanto ogni frase composta da almeno una parola del dizionario costituisce un input valido per l'algoritmo;
- per poter valutare il grado di "bontà" di una frase decifrata rispetto a quella originale sono necessari metodi euristici (motori di ricerca o moduli offline) che intrinsecamente non possono garantire l'esattezza della stima;
- l'eventuale intercettazione del dizionario da parte di un malintenzionato non compromette in alcun modo la sicurezza del crittosistema;
- utilizzando un dizionario specifico che contenga parole principalmente relative all'argomento del testo da cifrare, la crittoanalisi risulta ancora più ardua (più frasi semanticamente corrette e quindi difficilmente scartabili);
- la probabilità che lo stesso testo in chiaro cifrato con la medesima chiave generi lo stesso messaggio cifrato è direttamente proporzionale al numero di parole univoche presenti nel dizionario.

Per quanto riguarda l'**efficienza** e la **scalabilità**, possiamo inoltre fare queste osservazioni:

- data una frase di p parole, le funzioni di cifratura e decifratura hanno entrambe complessità lineare $O(p)$;
- la cifratura avviene separatamente su ogni frase, quindi l'algoritmo può essere facilmente adattato ad un intero testo considerando i segni di punteggiatura come normali parole del dizionario;
- sebbene il messaggio cifrato sia di solito più lungo del testo in chiaro, è possibile ridurne la dimensione prima dell'invio con un qualsiasi algoritmo di compressione (Huffman, LZW, GZIP, ecc...), per poi scompattarlo prima della decifratura;
- anche se non è stato argomento di questa relazione, riteniamo possibile una realizzazione hardware di questo cifratore che implementi i due circuiti necessari allo shuffle diretto ed inverso, sfruttando un CMOS per memorizzare il dizionario.