Data Encryption Standard

DES, breve storia

- 1973: Il National Bureau of Standards (NBS) pubblica un bando in cui richiede un algoritmo di cifratura:
  - La cui sicurezza risiedesse nella segretezza della chiave e non nel processo di cifratura
  - Che potesse essere realizzato efficientemente a livello hardware
- IBM propone una prima versione del DES
- NSA (National Security Agency) lo certifica ma propone delle variazioni:
  - Riduzione della lunghezza della chiave, da 128 bit a 56 bit
  - Modifica delle funzioni contenute nelle S-box

DES, breve storia

- 1977: DES viene accettato e reso pubblicamente disponibile.
  - Primo cifrario certificato ufficialmente come sicuro e noto a tutti.
  - Certificato ufficialmente ogni 5 anni
- 1987: Prime manifestazioni di sfiducia sul DES
- 1998: Il DES viene rotto
- 2000: NBS sceglie il successore del DES, denominato Advanced Encryption Standard (AES).

IL DES

cifrario simmetrico a blocchi

- I cifrari a blocchi
  - Il messaggio viene suddiviso in blocchi di n bit
  - I blocchi vengono cifrati indipendentemente l'uno dall'altro.
- I cifrari simmetrici:
  - Cifratura e decifrazione con la stessa chiave.

Le basi del DES

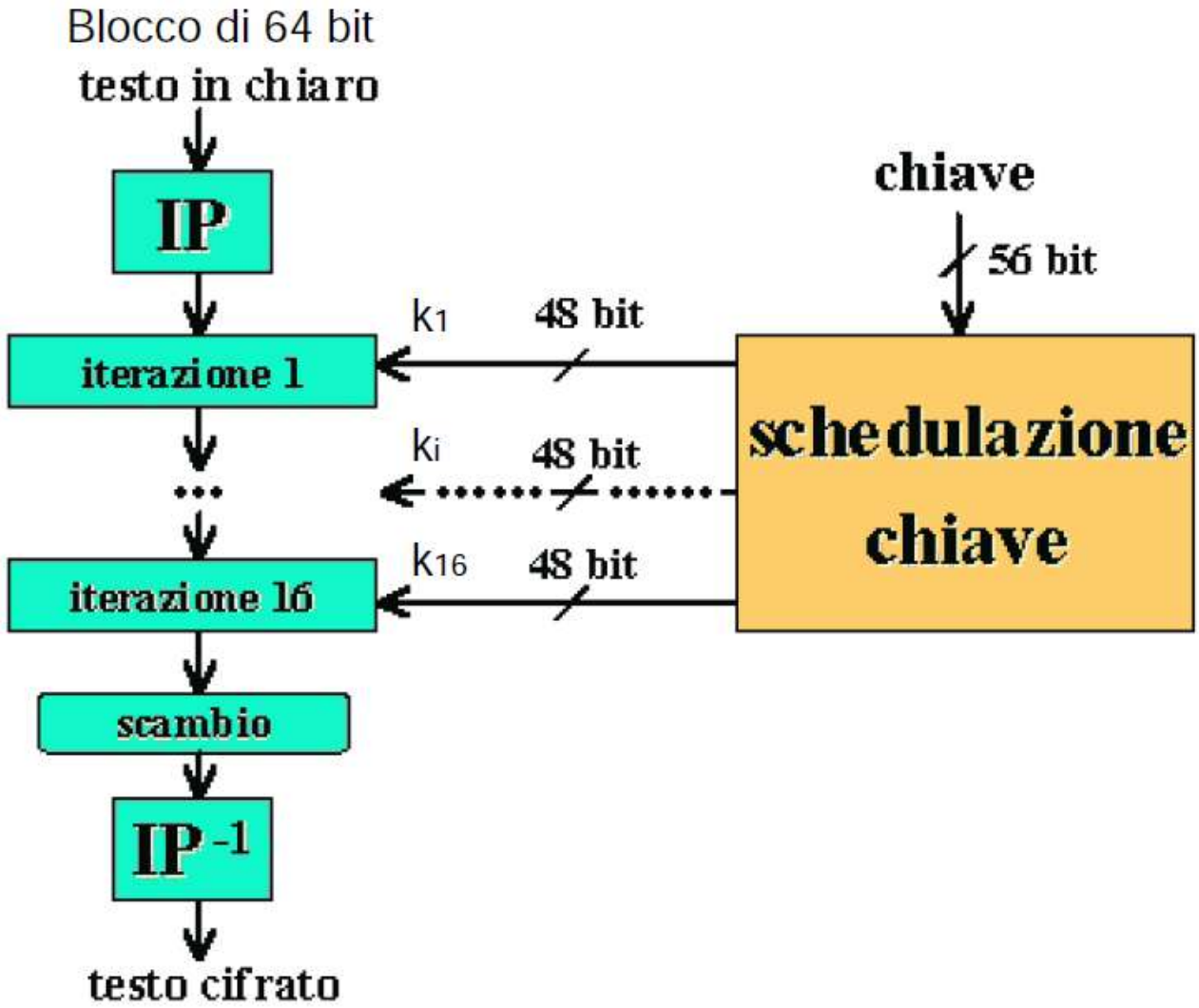
Proprietà desiderabili di un algoritmo di cifratura secondo Shannon.

- Diffusione
  - Alterare la struttura del testo in chiaro "spargendo" i caratteri su tutto il testo cifrato.
- Confusione
  - Combinare in modo complesso il messaggio e la chiave, per non permettere al crittoanalista di separare le due sequenze mediante l'analisi del crittogramma.

Nel DES diffusione e confusione sono soddisfatte attraverso una serie di permutazioni e combinazioni del messaggio con la chiave.

La Struttura

- Ad alto livello

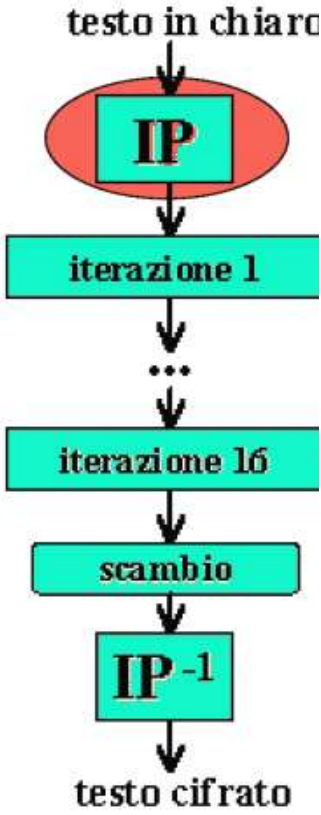


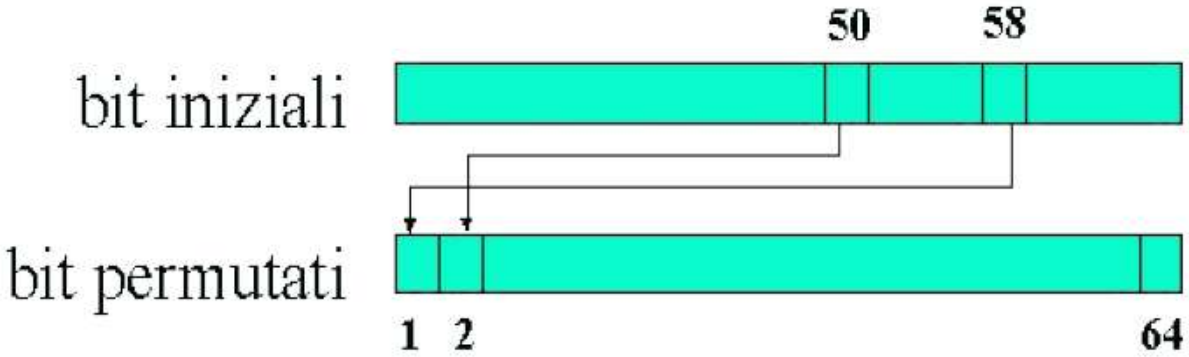
La Struttura

- Un blocco di testo in chiaro (64 bit) viene permutato dal blocco IP, il risultato di questa permutazione andrà in ingresso alla prima iterazione.
- In ogni iterazione il blocco di 64 bit viene opportunamente mescolato con una Kiesima chiave.
- In tutto 16 iterazioni.
- Dopo le 16 iterazioni al testo di 64 bit oramai cifrato viene applicata la permutazione iniziale inversa IP^-1

IP : permutazione iniziale

Blocco in ingresso								Permutazione iniziale							
0	1	0	1	0	0	1	1	58	50	42	34	26	18	10	02
0	1	0	0	0	1	0	1	60	52	44	36	28	20	12	04
0	1	0	0	0	0	1	1	62	54	46	38	30	22	14	06
0	1	0	1	0	1	0	1	64	56	48	40	32	24	16	08
0	1	0	1	0	0	1	0	57	49	41	33	25	17	09	01
0	1	0	0	1	0	0	1	59	51	43	35	27	19	11	03
0		0	1	0	1	0	0	61	53	45	37	29	21	13	05
0	1	0	1	1	0	0	1	63	55	47	39	31	23	15	07





IP

Dati permutati

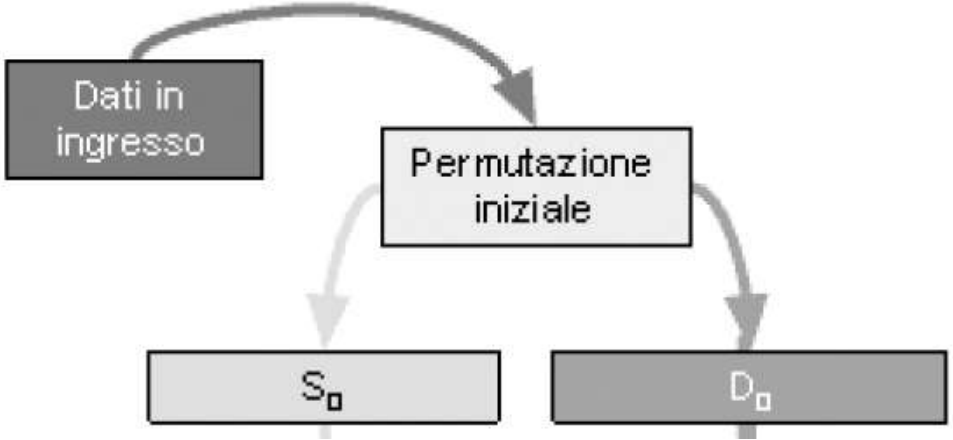
58	50	42	34	26	18	10	02
60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06
64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01
59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05
63	55	47	39	31	23	15	07

32bit

Parte sinistra
S_0

32bit

Parte destra
D_0

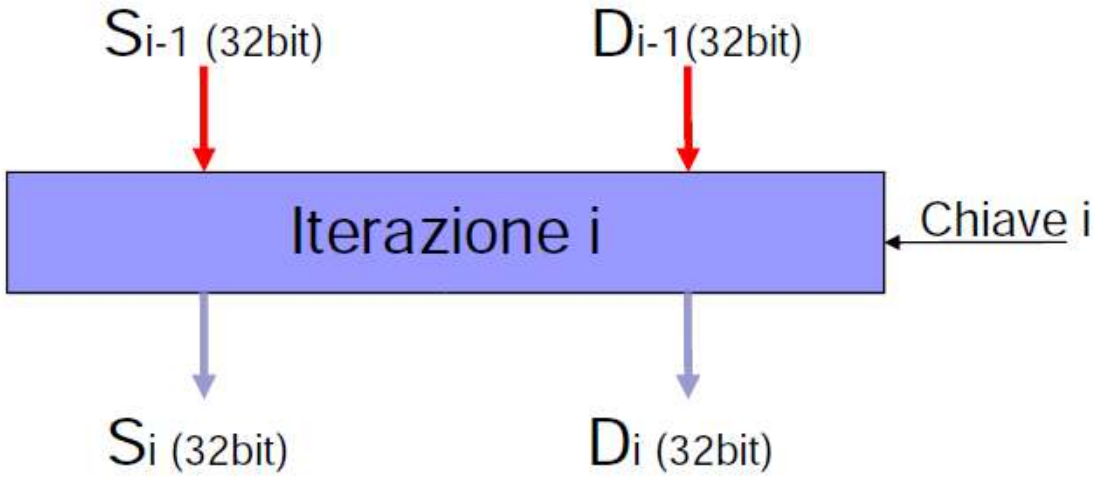


Input Iterazione 1

IP : permutazione iniziale

- Nella permutazione iniziale i bit vengono scambiati in base alla matrice di permutazione IP, ovvero in posizione [1,1] andrà il bit in posizione 58 del testo in chiaro, in seconda posizione il bit 50 e cosi via.
- Al termine della permutazione il blocco viene suddiviso in due sotto-blocchi da 32 bit ciascuno S_0 e D_0, questi saranno l'input della prima iterazione.

Iterazione

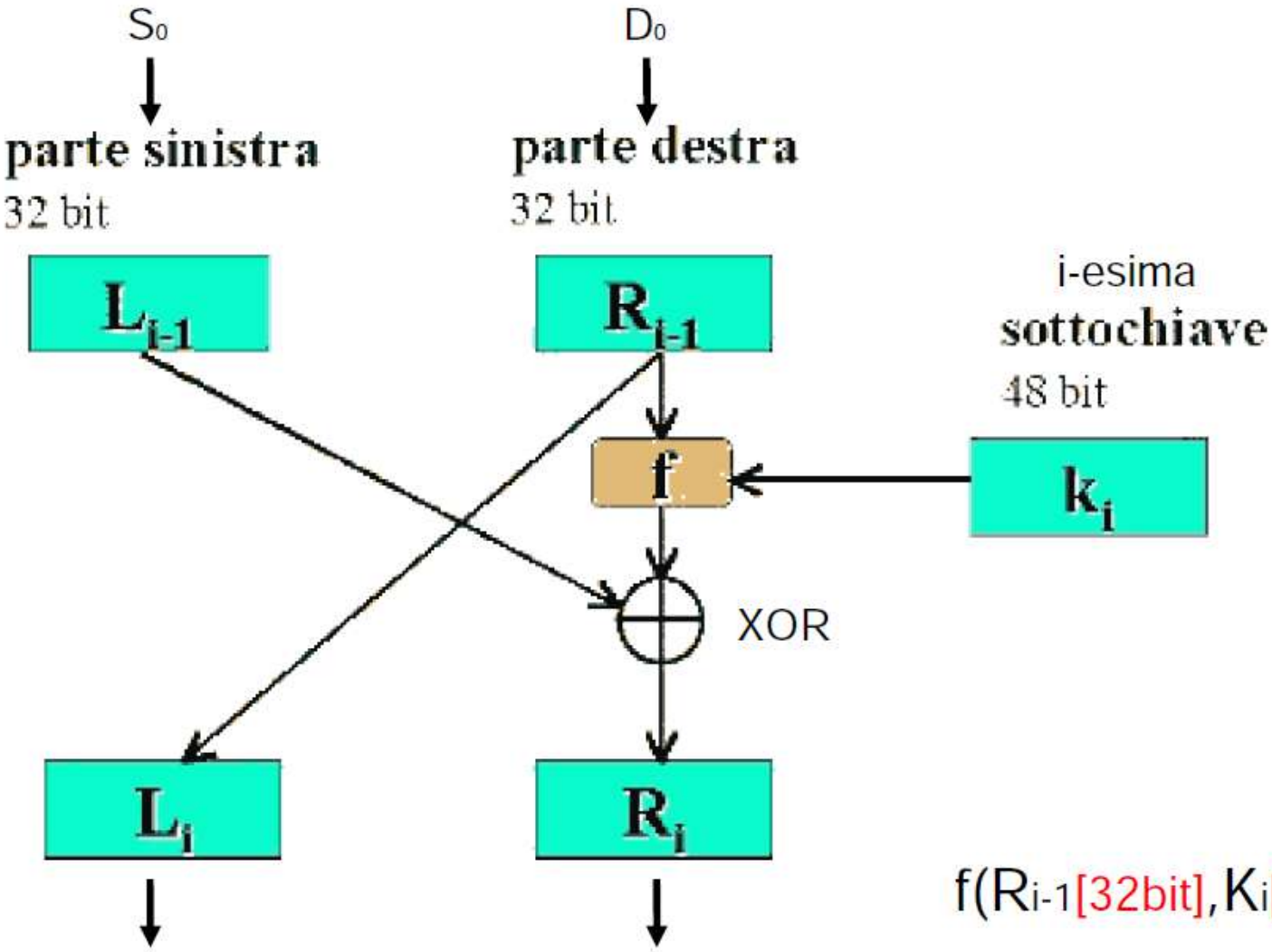

Si-1 (32bit)
Di-1(32bit)
Iterazione i
Chiave i
Si (32bit)
Di (32bit)


testo in chiaro
IP
iterazione 1
...
iterazione 16
scambio
IP -1
testo cifrato

Iterazione

- Ogni Iterazione ha la seguente struttura :
  - Input : due blocchi S_{i-1} e D_{i-1} da 32bit, una chiave Ki da 48 bit.
  - Elaborazione : combinazione di S_{i-1} e D_{i-1} con Ki.
  - Output : due nuovi blocchi S_i e D_i da 32 bit.

Iterazione



$f(R_{i-1}[32bit], K_i[48bit])$

Passi del ROUND

QUINDI:

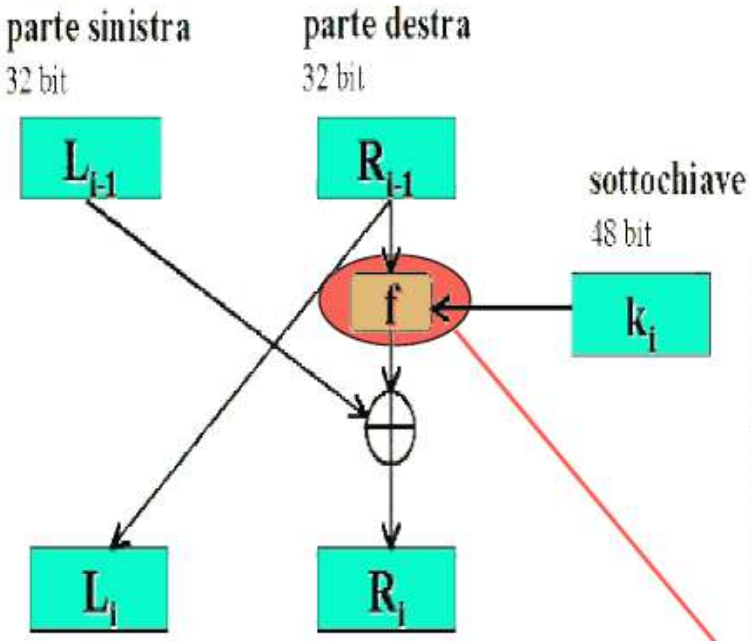
- $L_i = R_{i-1}$
- $R_i = L_{i-1}$ XOR Feistel(R_{i-1}, K_i)

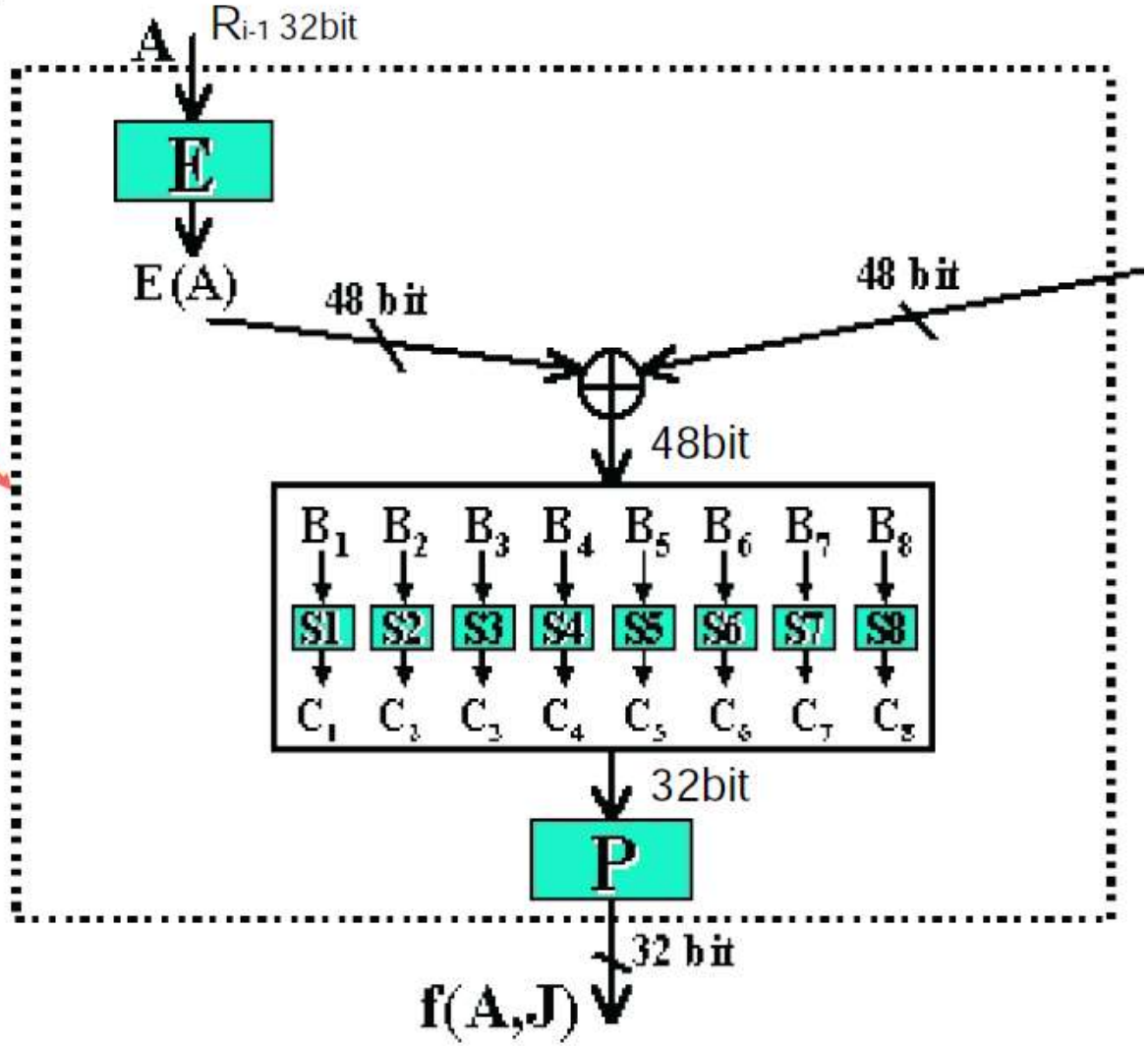
Iterazione

- Elaborazione dei blocchi nell'iterazione.
  - I due blocchi di Output Li e Ri sono ottenuti in questo modo :
    - Il nuovo blocco Li è il vecchio Ri-1
    - Il nuovo blocco Ri è ottenuto mettendo in XOR il vecchio blocco Li-1 con l'output della funzione F
  - La Funzione F combina il vecchio blocco di destra Ri-1 con la chiave Ki

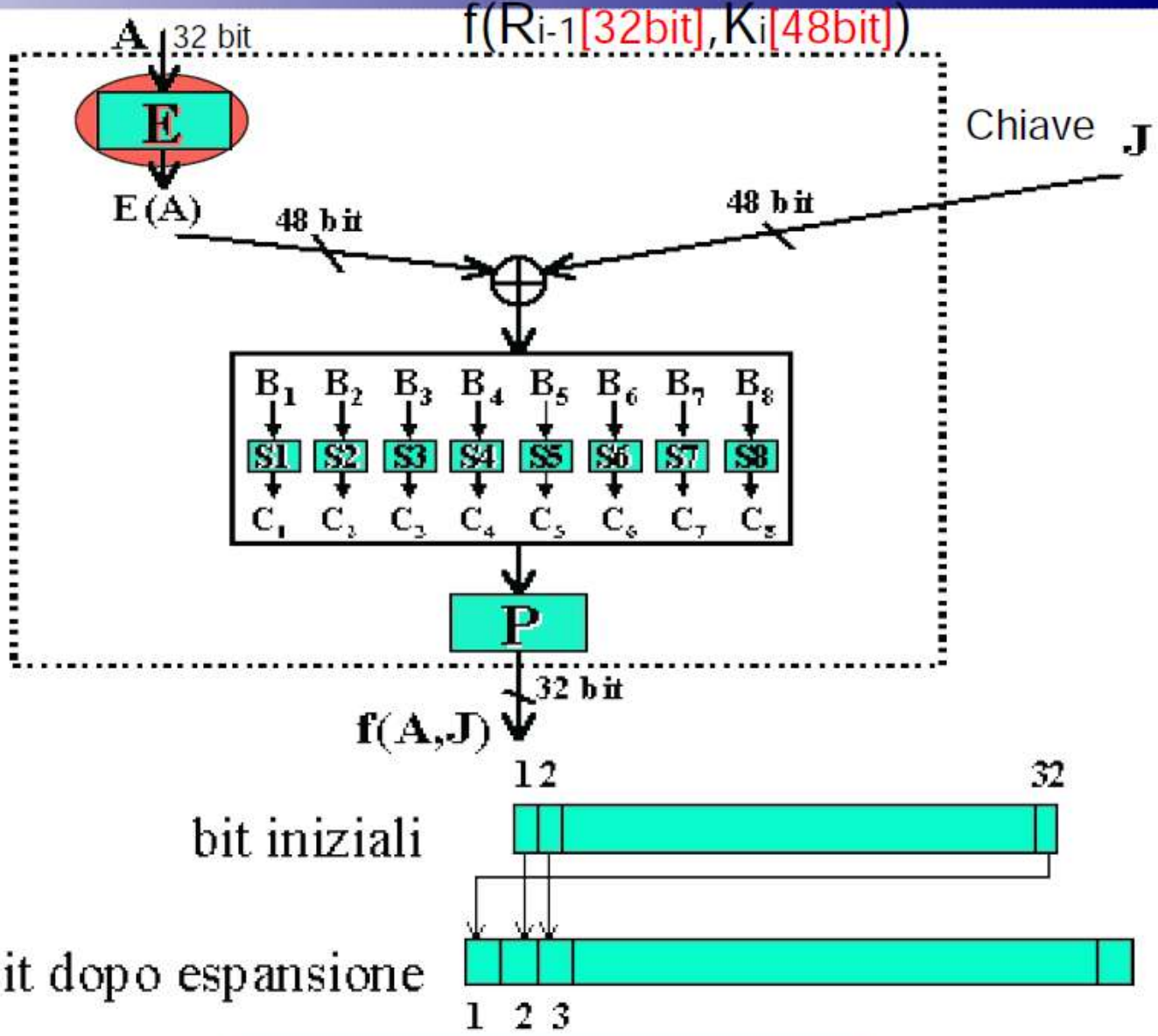
La Funzione f

$f(R_{i-1}[32bit], K_i[48bit])$





Il Blocco 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

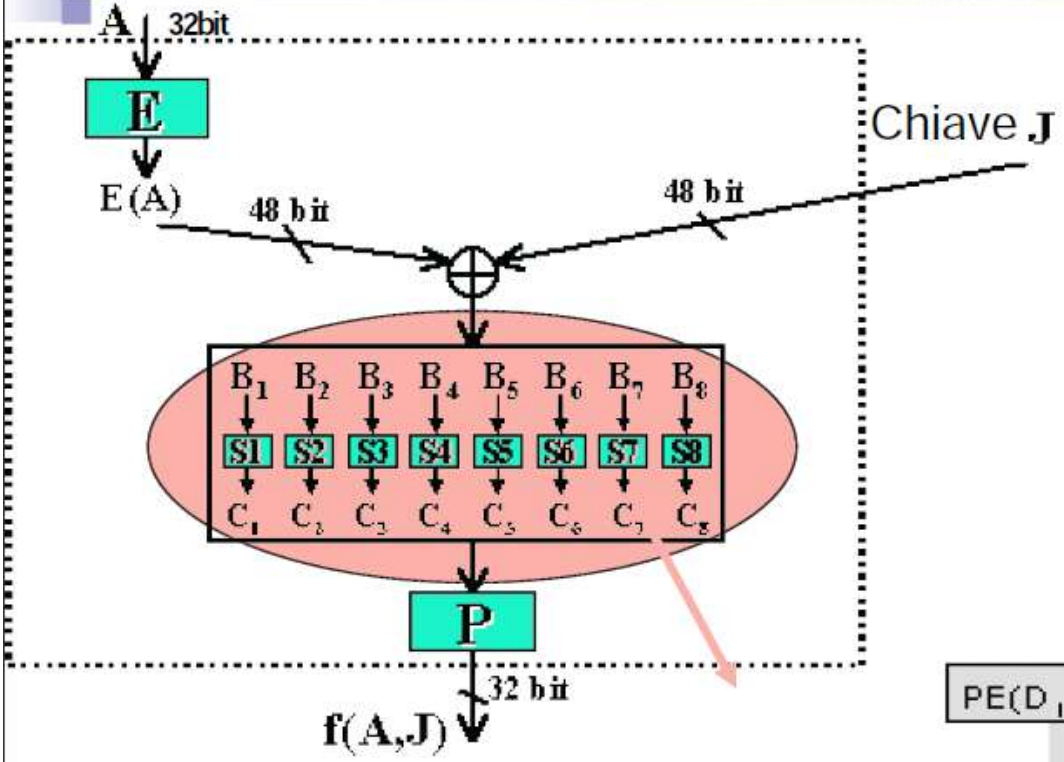
Alcuni bit vengono ripetuti

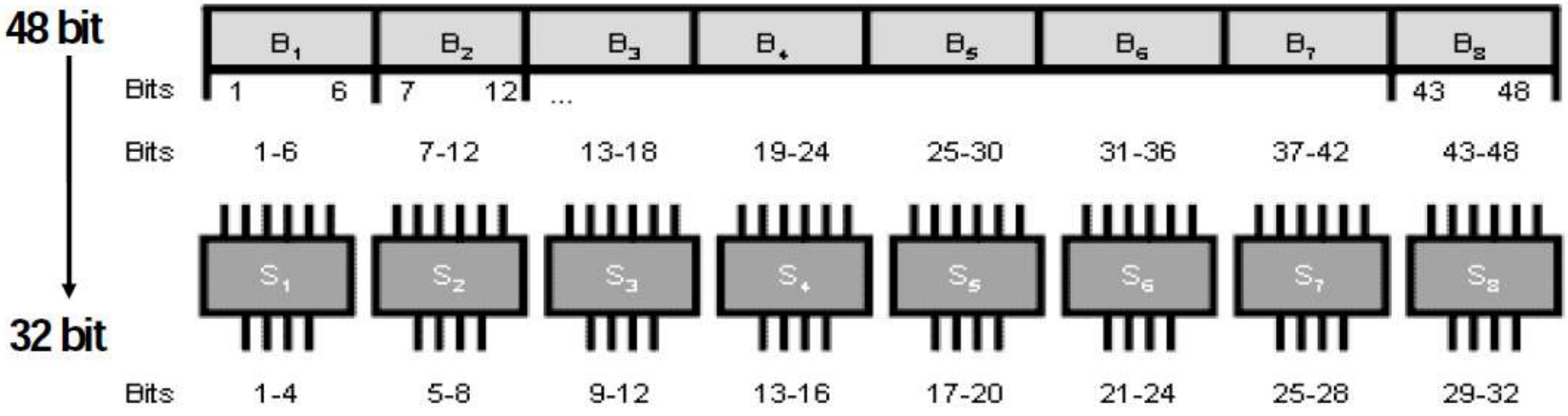
Funzione F

- Inizialmente i 32 bit del vecchio blocco di destra Ri-1 vengono espansi a 48 bit dal blocco E mediante la matrice E, semplicemente alcuni bit vengono ripetuti.
- I 48 bit ottenuti dall'espansione vengono messi in XOR con la Ki chiave anch'essa di 48 bit.
- La nuova sequenza B di 48 bit ottenuta viene suddivisa in 8 sotto-blocchi da 6 bit ciascuno.
- La stringhe Bi di 6 bit vengono date in ingresso alle otto funzioni SBOX.

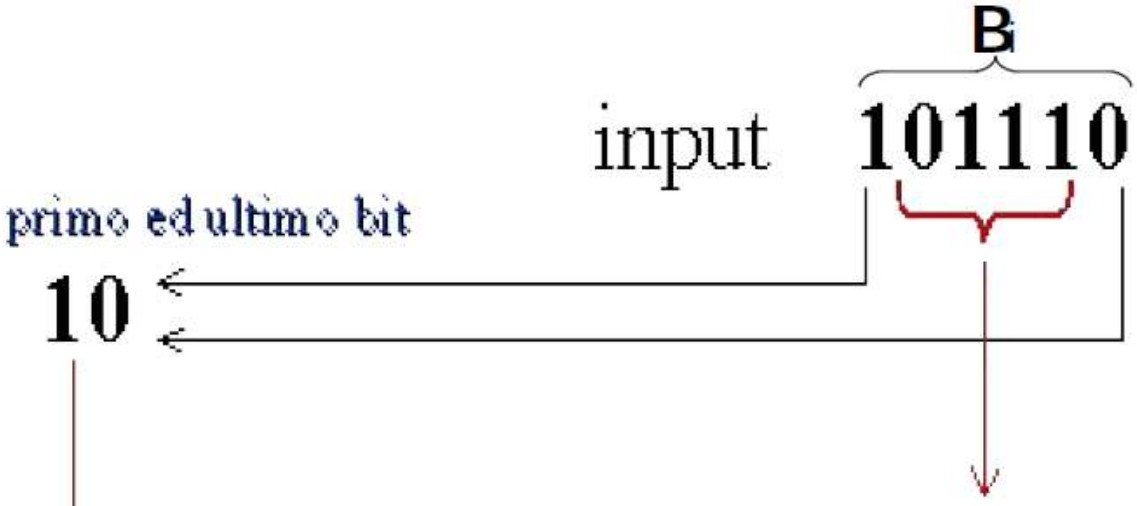
Le S-Box

Parte cruciale su cui si basa la sicurezza del cifrario





Le S-Box



	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	7	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	10	2	4	9	1	7	5	11	3	14	10	0	6	13

Box S1

output 11 in binario = **1011**

SBOX

- Le SBOX hanno il compito di trasformare un blocco in ingresso Bi di 6 bit in un blocco in uscita di 4 bit.
- Dalla figura nella slide precedente possiamo vedere come ciò avviene: dal primo e l'ultimo bit del blocco Bi in input ricaviamo l'indice di riga della SBOX, dai 4 bit centrali restanti invece ricaviamo l'indice di colonna.
- Otteniamo cosi l'elemento S[i,j]=11 ; trasformandolo in binario otteniamo i 4 bit di output.
- Mettendo in sequenza gli output delle otto SBOX otteniamo la stringa di 32bit, questa verrà sottoposta ad un'ultima permutazione P.

Le 8 S-Box

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

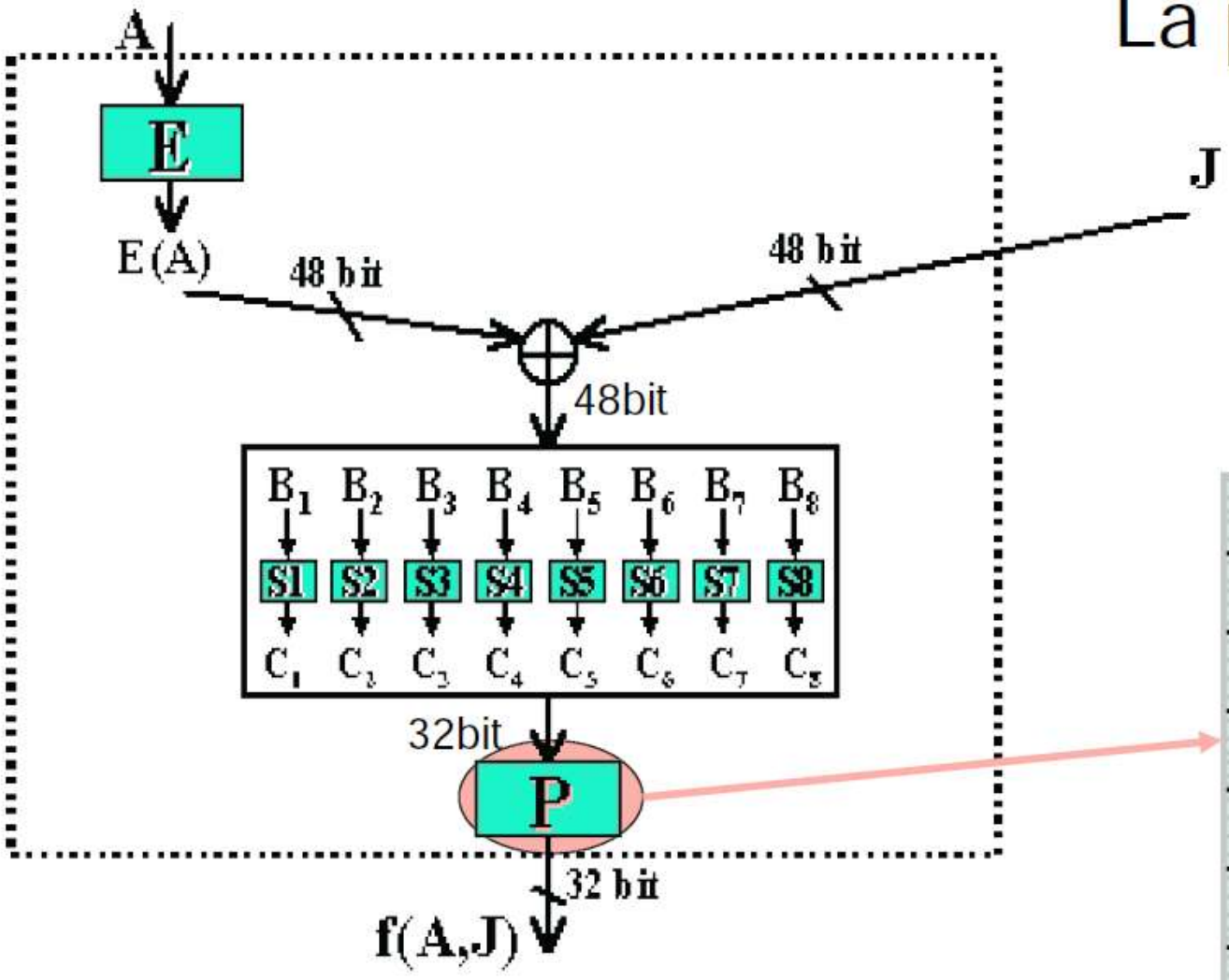
S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

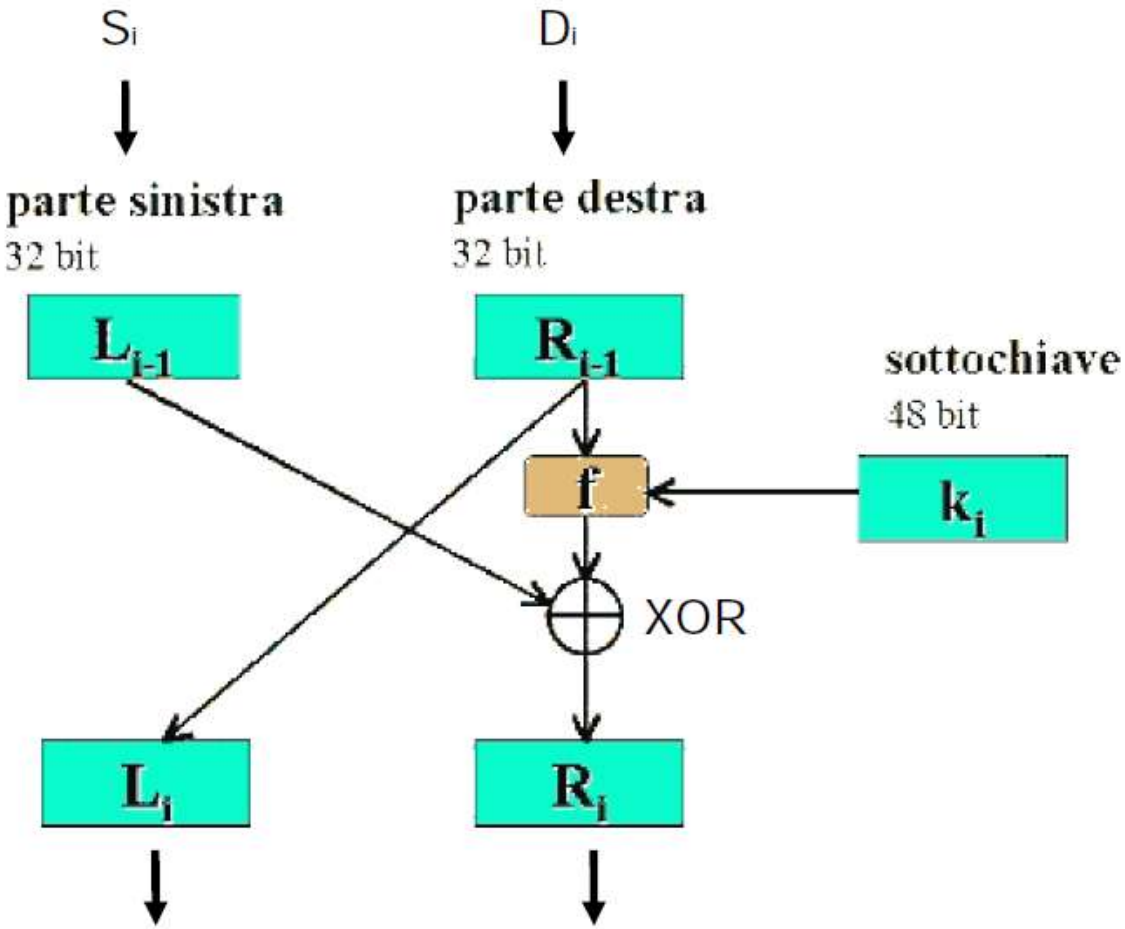
S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

La 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

Iterazione


Si
Di
parte sinistra
32 bit
parte destra
32 bit
Li-1
Ri-1
sottochiave
48 bit
f
ki
XOR
Li
Ri
Input Iterazione successiva

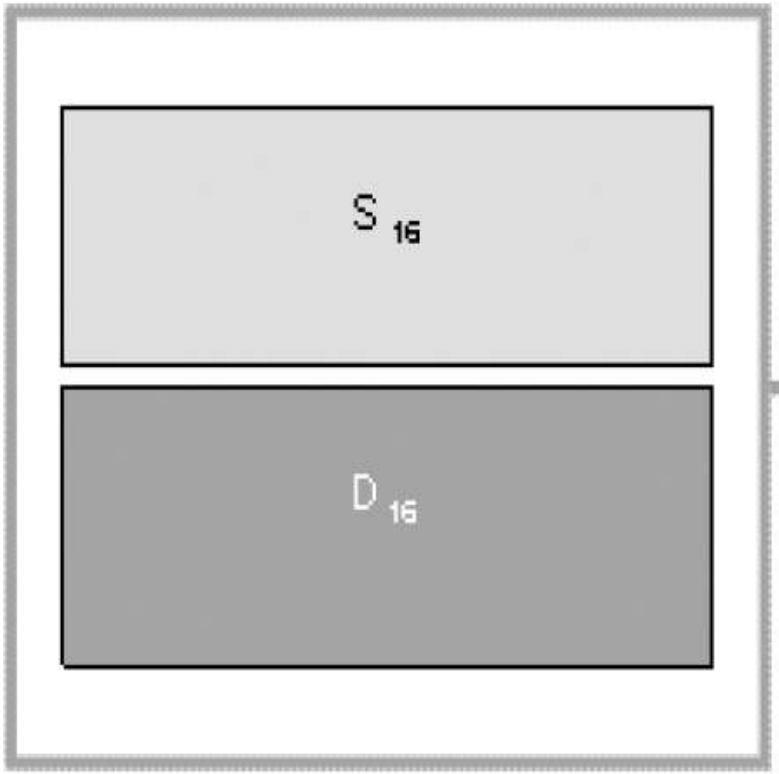

testo in chiaro
IP
iterazione 1
...
iterazione 16
scambio
IP -1
testo cifrato

Scambio

- Le parti L_{16} ed R_{16} vengono invertite

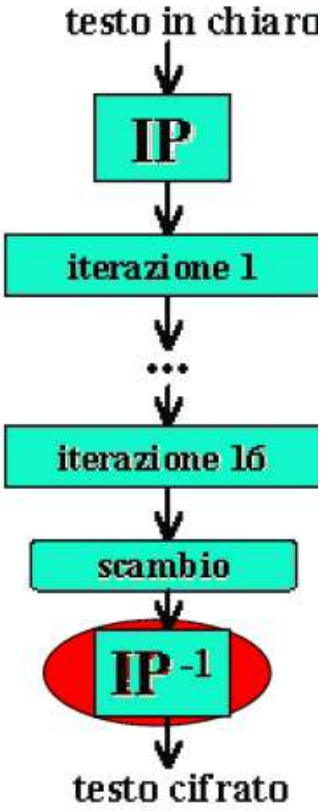
- In questo modo il blocco viene predisposto per il processo di decifrazione, come vedremo in seguito.

IP^{-1}

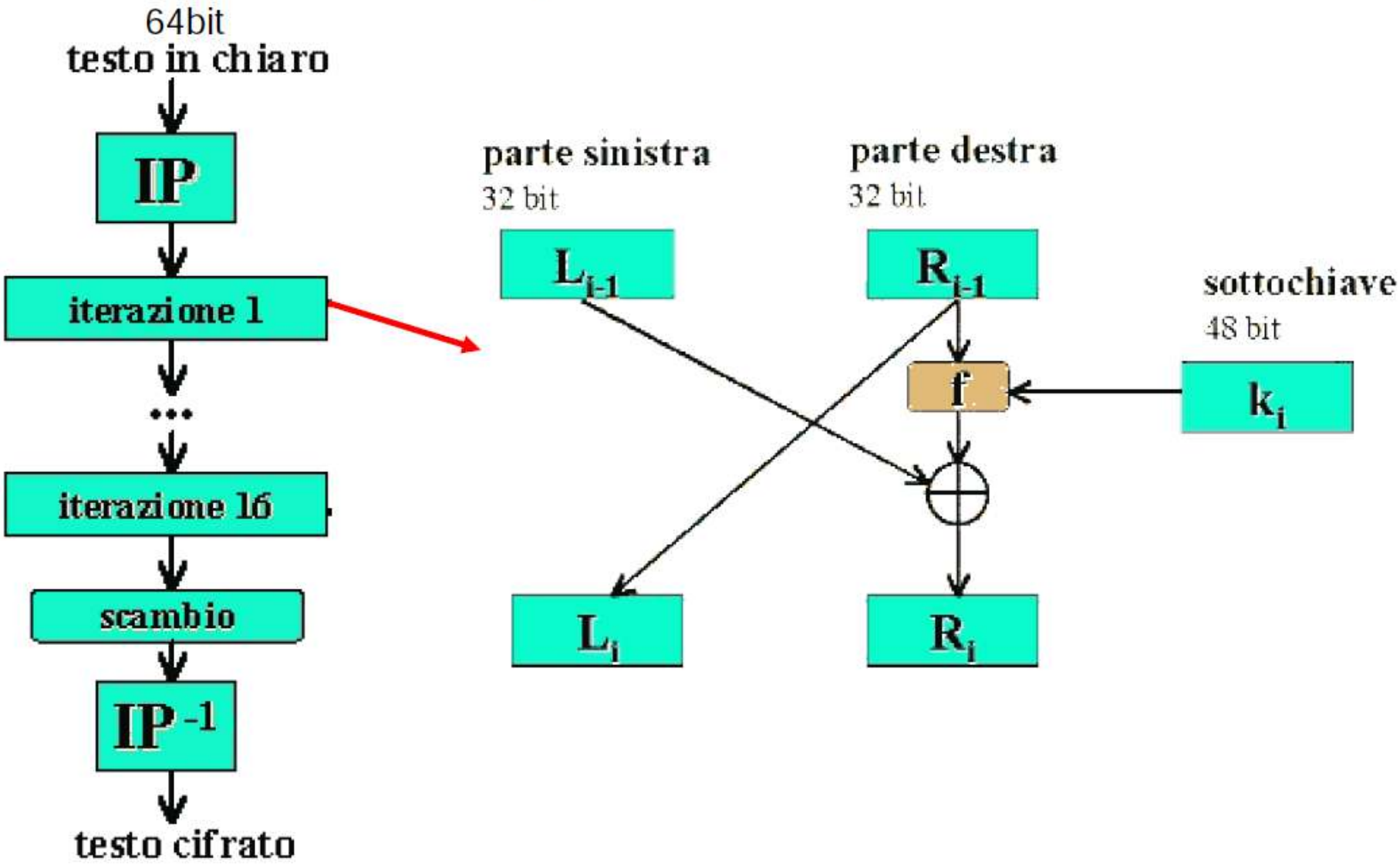


Permutazione finale

40	08	48	16	56	24	64	32
39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	41	09	49	17	57	25

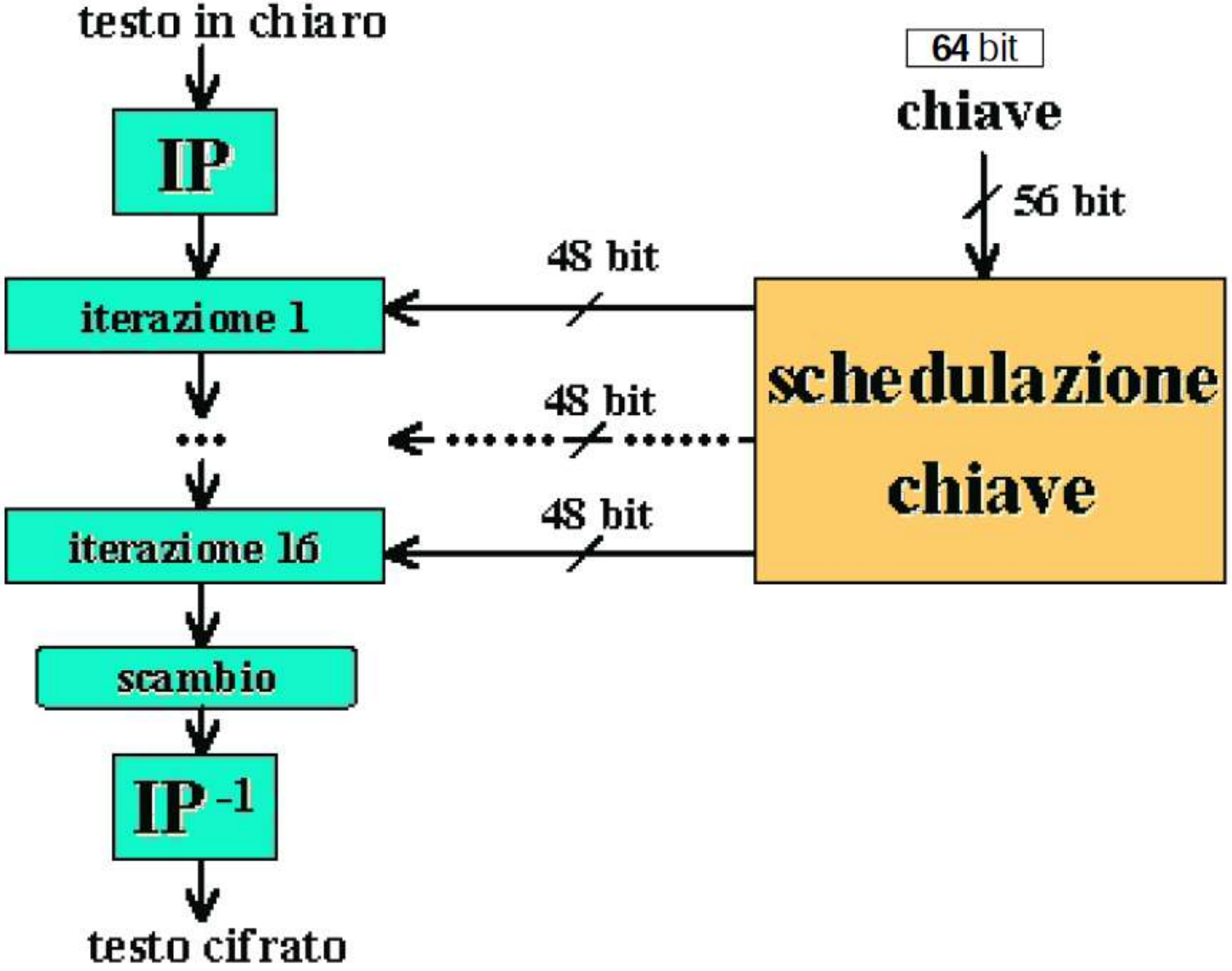


Ricapitolando…



Generazione delle chiavi e decifrazione

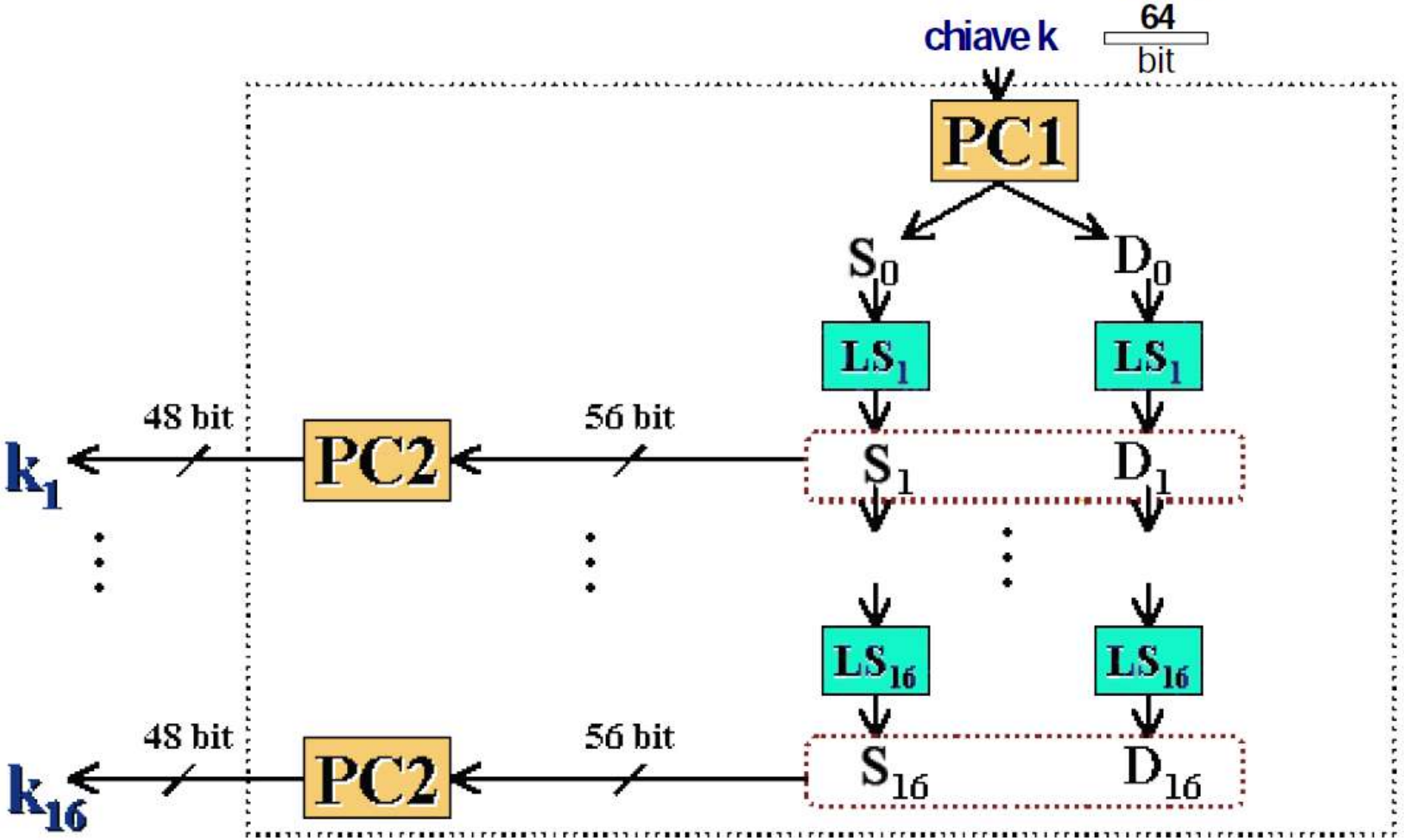
Generazione delle Chiavi (1)



Generazione delle Chiavi (2)

- Le chiavi vengono generate da uno Scheduler. Esso prende in input la chiave di 64 bit (generata in modo random automaticamente dal sistema una volta per l' intero procedimento di cifratura)
- Lo Scheduler interviene ad ogni iterazione del DES producendo una sotto-chiave diversa di 48 bit per ciascun round
- Dalla chiave di partenza vengono dunque prodotte 16 sotto-chiavi diverse

Generazione delle Chiavi (3)



Generazione delle Chiavi (3)

- Lo Scheduler si compone di diverse box, attraverso le quali la chiave viene ridotta di dimensioni (in termini di bit) e rimaneggiata per produrre le diverse sotto-chiavi

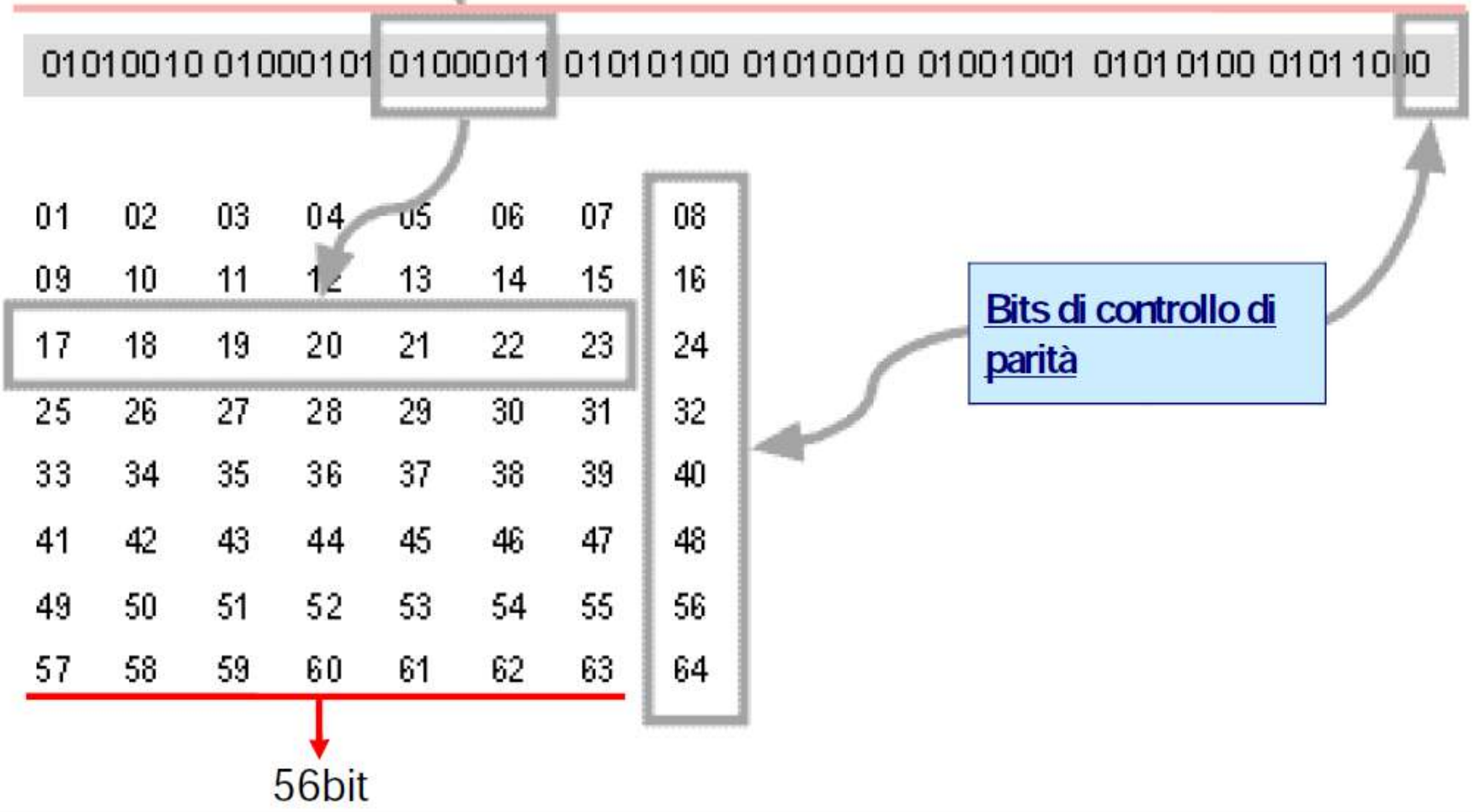
- Elementi fondamentali sono:
  - PC1 (Permutate Choise 1)
  - LS_i (Left Shift i)
  - PC2 (Permutate Choise 2)

Riduzione bit (PC1)

Dati della chiave generati in modo casuale

64 bit

- ***K*** chiave di 64 bit di cui 8 utilizzati per controllo di parità
- Le chiavi usate nei round sono tutte derivate da ***K***



Bits di controllo di parità

PC1 (1)

Chiave iniziale

01	02	03	04	05	06	07	08
09	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

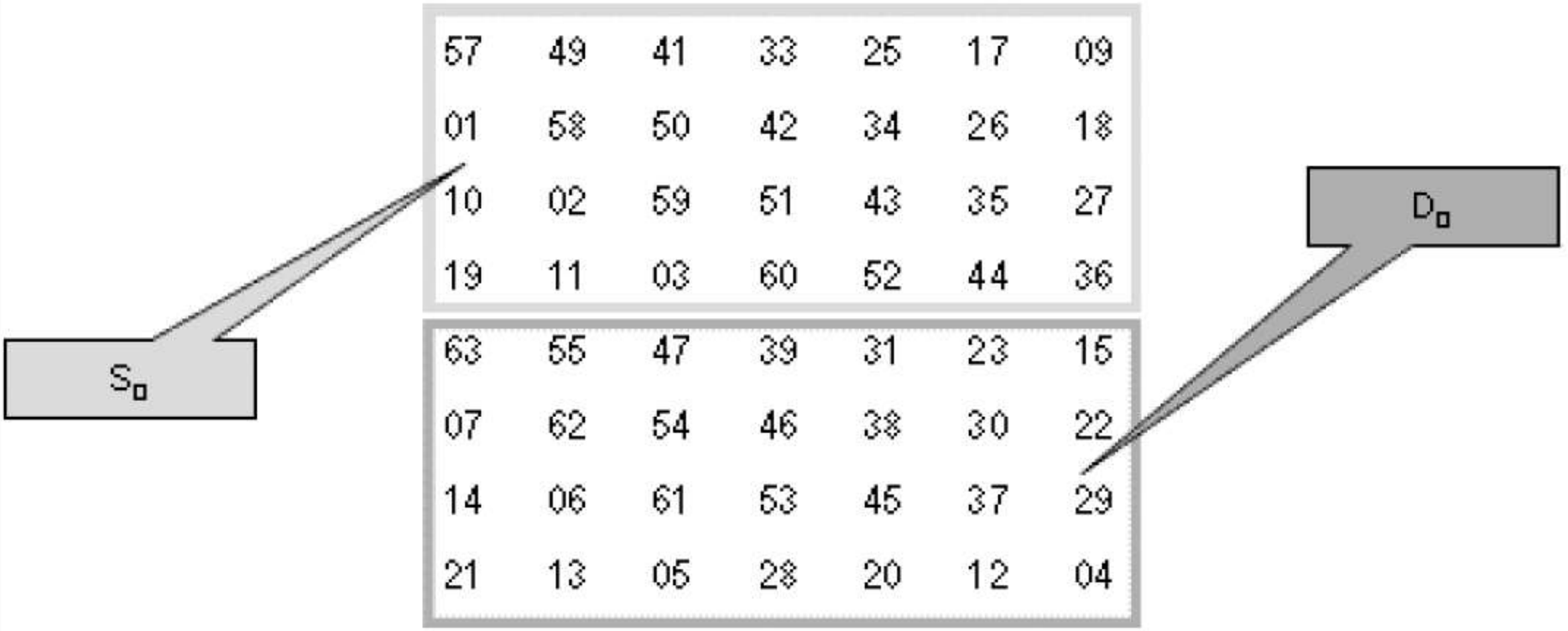
Scelta permutata 1

57	49	41	33	25	17	09
01	58	50	42	34	26	18
10	02	59	51	43	35	27
19	11	03	60	52	44	36
63	55	47	39	31	23	15
07	62	54	46	38	30	22
14	06	61	53	45	37	29
21	13	05	28	20	12	04

Bits scartati

Fig. 1

PC1 (2)



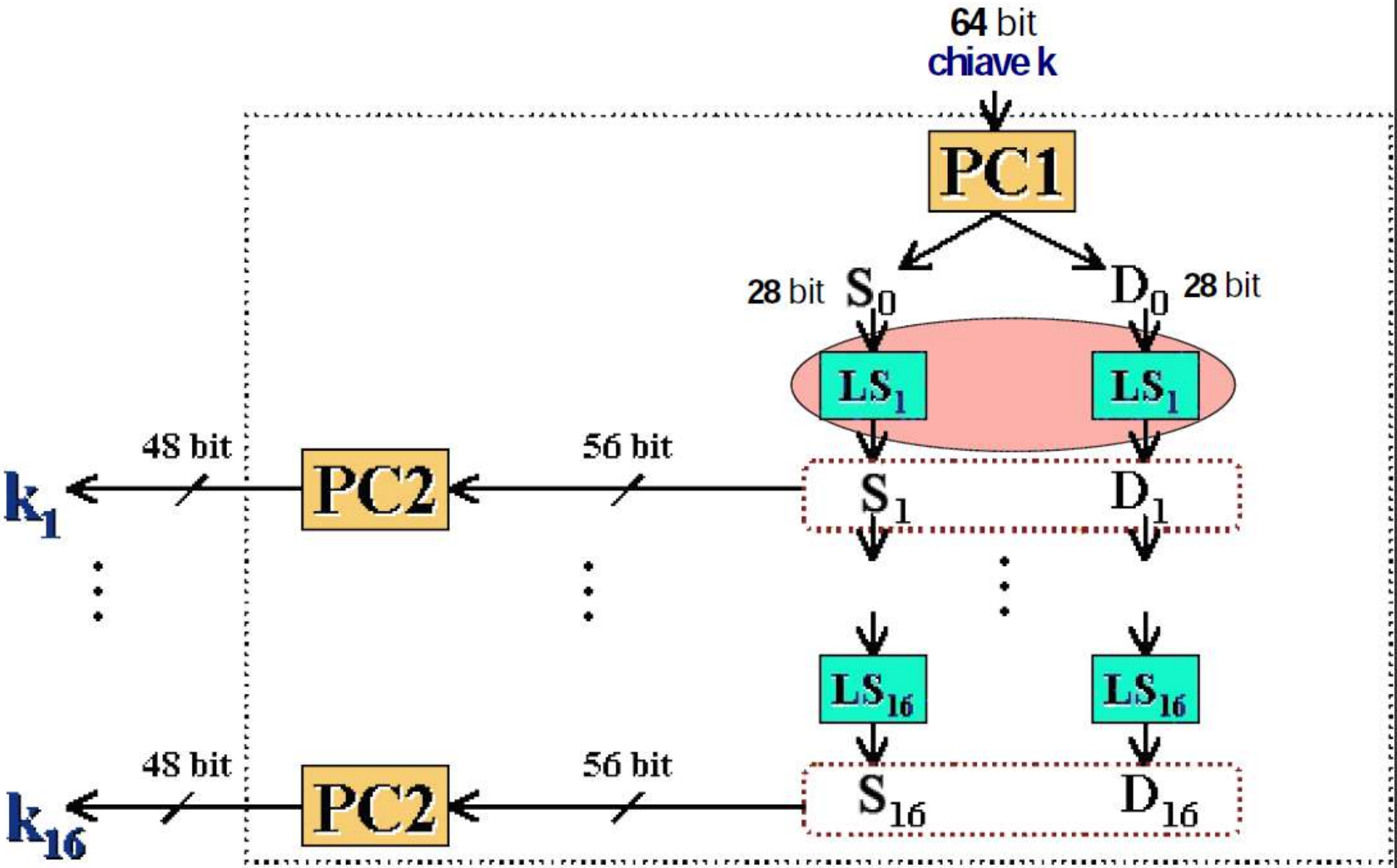
- Vengono prodotte due semi-chiavi, S_0 e D_0

PC1 (spiegazioni)

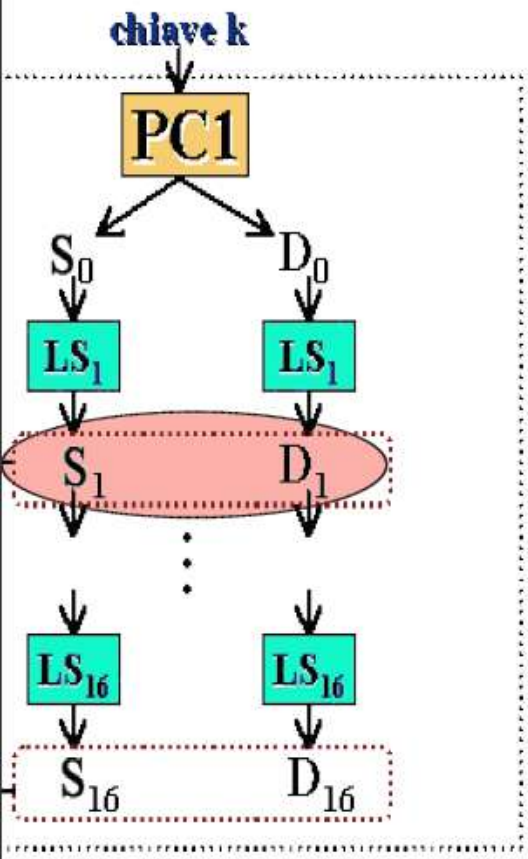
- Accetta in ingresso la chiave di 64 bit assegnata al processo di cifratura. Tale chiave contiene 8 bit per il controllo di parità (uno per ogni byte). Non contenendo informazione i bit di parità possono essere eliminati, portando la chiave da 64 a 56 bit. In una rappresentazione matriciale della chiave (matrice 8x8) i bit da scartare sono quindi 8,16,24,32,40,48,56,64. La matrice uscente è quindi 8x7.

- Esegue una permutazione di bit secondo lo schema in Fig.1. Il bit 57 ad esempio viene mappato nel bit 1 della nuova matrice, il bit 49 nel bit 2 etc..

- La matrice così ottenuta viene suddivisa in due (per righe), andando a produrre due sotto-chiavi di 28 bit ciascuna, etichettate come ***S0*** e ***D0*** (sotto-chiave Sinistra e sotto-chiave Destra)


64 bit
chiave k
PC1
28 bit
S_0
D_0
28 bit
LS_1
LS_1
48 bit
56 bit
PC2
k_1
S_1
D_1
LS_{16}
LS_{16}
48 bit
56 bit
PC2
k_{16}
S_{16}
D_{16}

LSi



Ciclo	Spostamento a sinistra
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Tabella 1

= 28

Shift ciclico

S_1

49	41	33	25	17	09	01
58	50	42	34	26	18	10
02	59	51	43	35	27	19
11	03	60	52	44	36	57

55	47	39	31	23	15	07
62	54	46	38	30	22	14
06	61	53	45	37	29	21
13	05	28	20	12	04	63

D_1

LSi dettagli

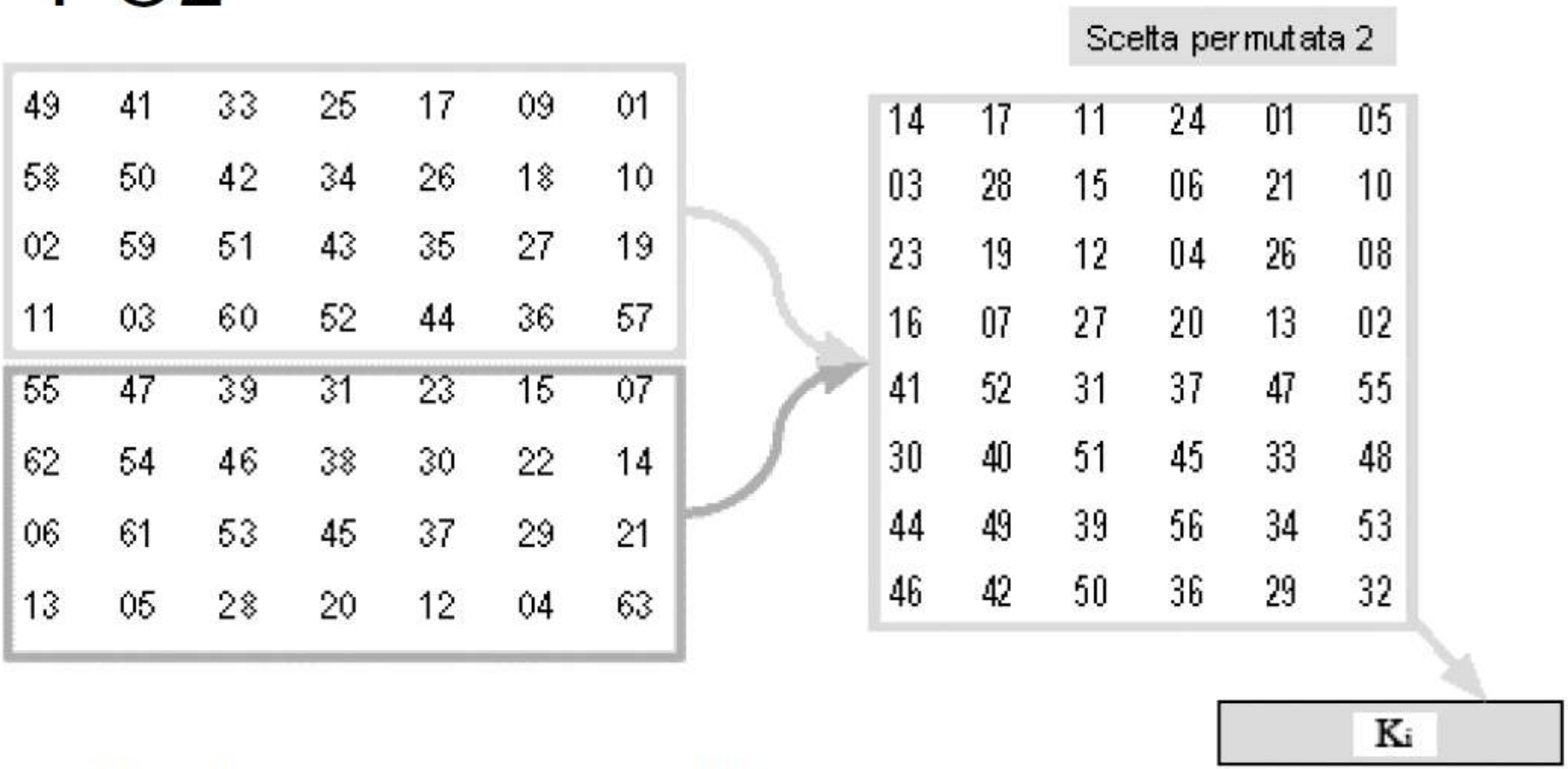
- Ad ogni iterazione le due semichiavi S_i e D_i di 28 bit vengono elaborate dal blocco LS_i in parallelo.

- LS_i effettua uno shift verso sinistra di tutti i bit delle semichiavi ad ogni iterazione in accordo con la Tabella 1. Come si può notare lo spostamento è di una posizione nelle itarazioni 1-2-9-16, di due posizioni in tutte le altre.

- Se sommiamo i numero totale di spostamenti otteniamo un numero di bit pari a 28. Ciò significa che lo shift è ciclico, ovvero alla sedicesima itarazione il primo bit della matrice di partenza è tornato in posizione 1. Questo dettaglio ci sarà utile nella fase di decifrazione.

- LS_i produce in output due nuove semichiavi S_{i+1} e D_{i+1}.

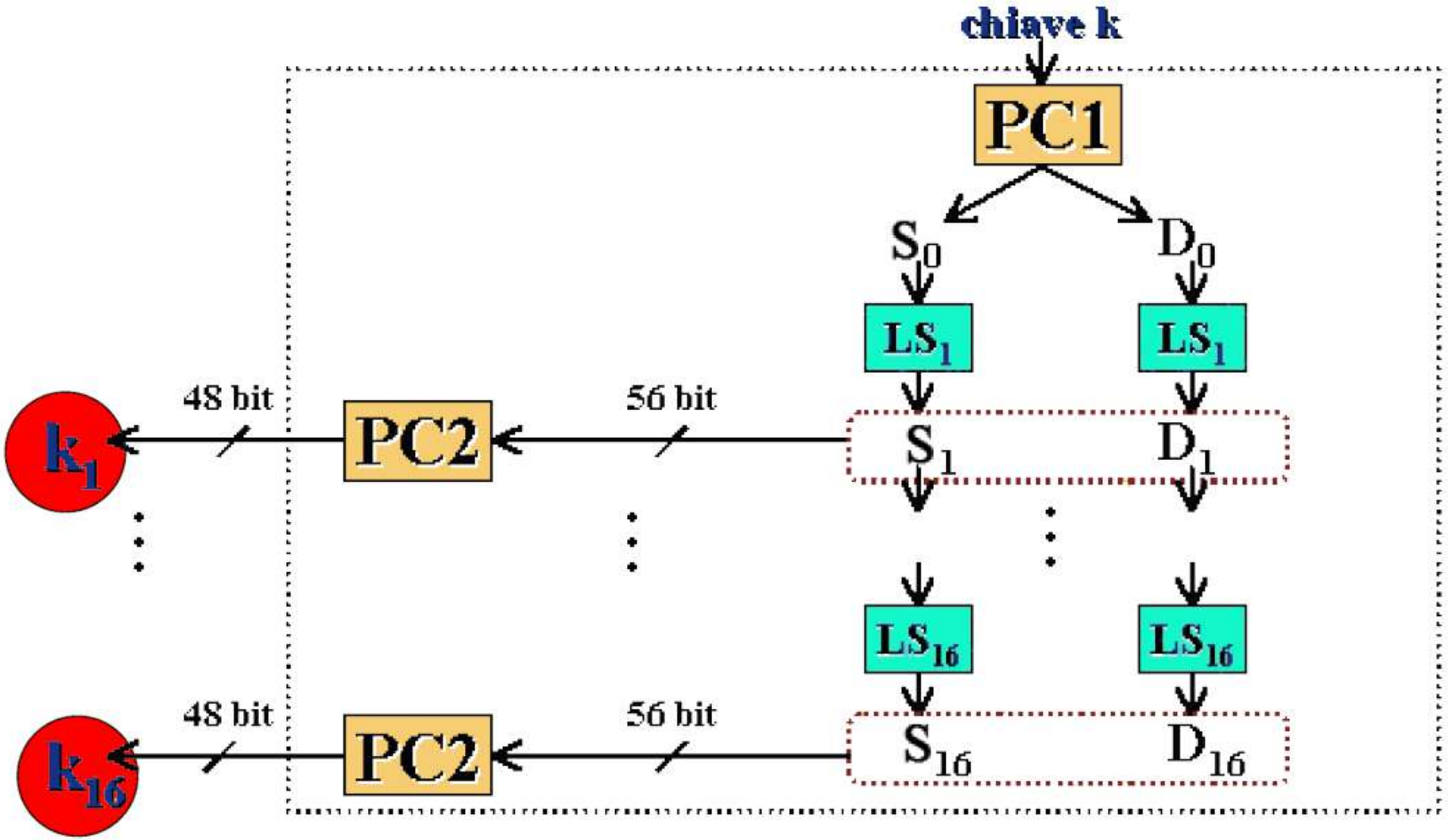
PC2



I bit che vengono soppressi dalla compressione sono quelli in posizione 9, 18, 22, 25, 35, 38, 43 e 54 della stringa input

Fig. 2

Output di PC2 (1)



Output di PC2 (2)

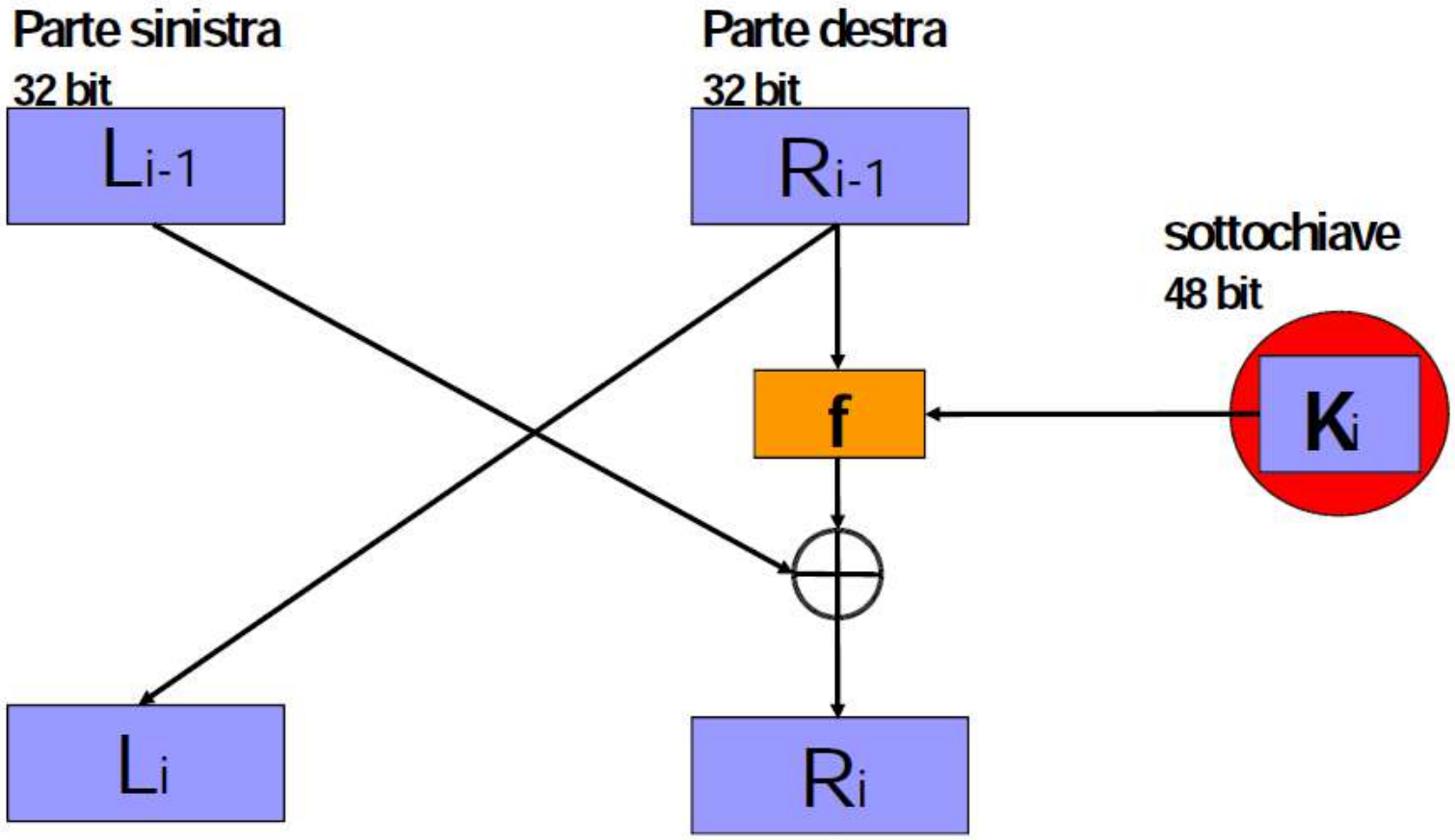
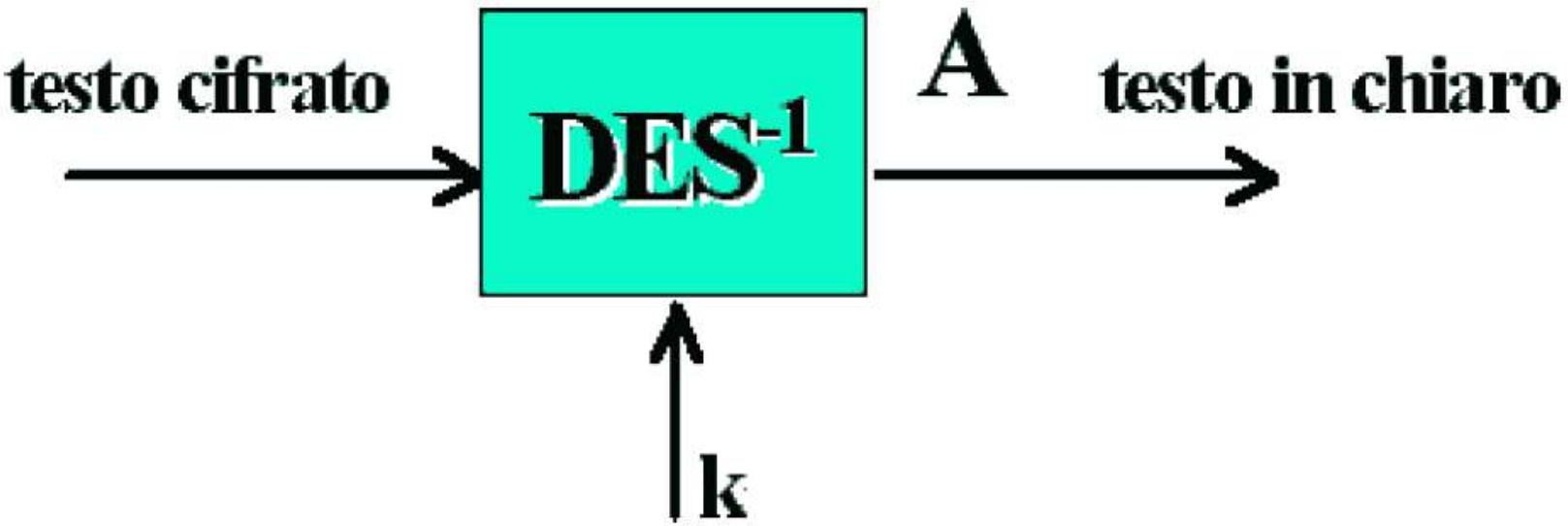


Fig. 3

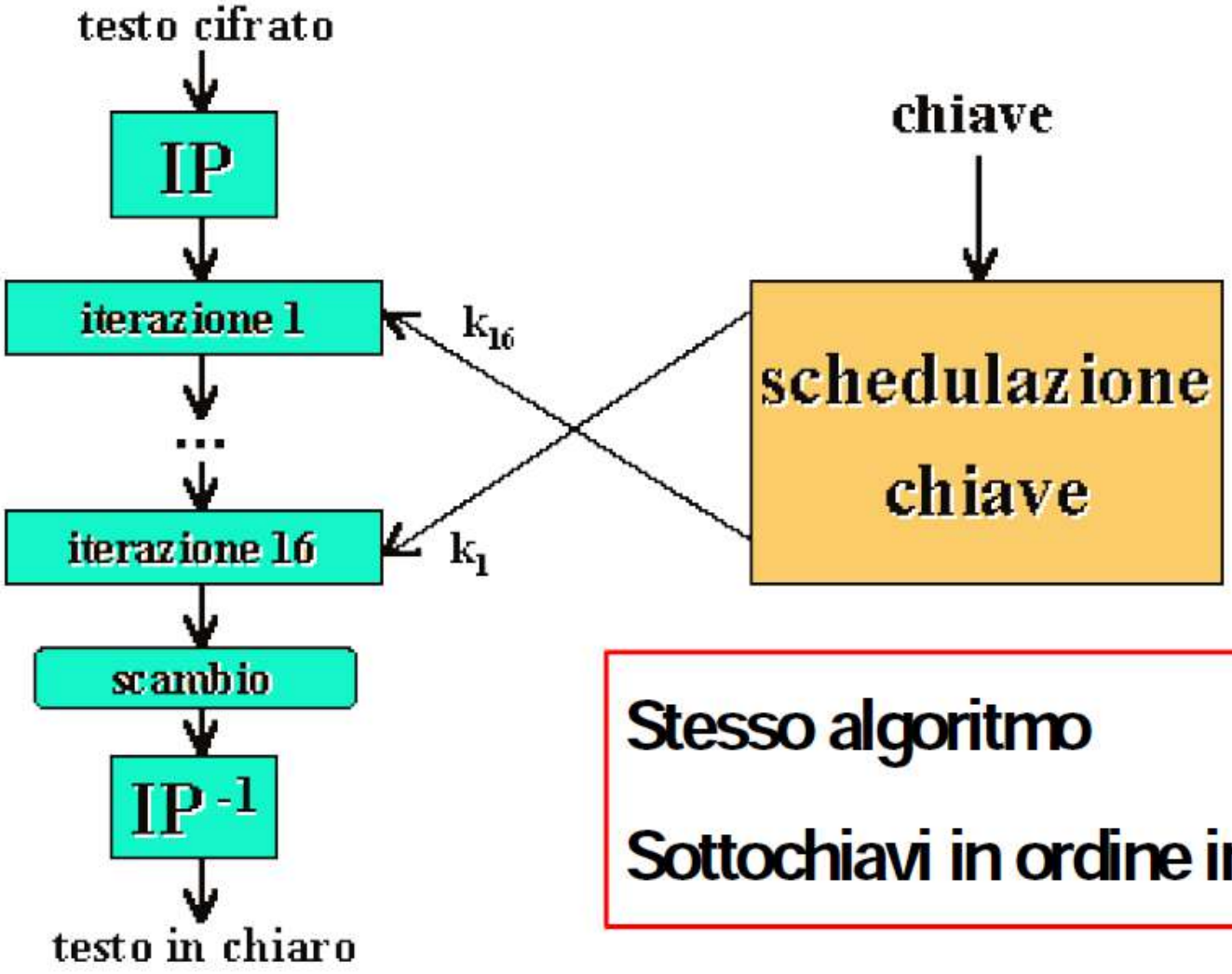
PC2 dettagli

- PC2 opera ad ogni itarazione per produrre in output una chiave da 48 bit.
- Prende in ingresso le due sotto-chiavi S_i e D_i di 28 bit ciascuna e le ricompone in una da 56 bit.
- Effettua una seconda scelta permutata (Fig. 2), che rimescola e comprime la chiave in input (vengono eliminati i bit in posizione 9, 18, 22, 25, 35, 38, 43 e 54) per ottenere in output una chiave da 48 bit.
- La chiave K_i così ottenuta va quindi in ingresso alla funzione ***f*** dell' i-esimo round del DES (Fig. 3)

Decifrazione (1)



Decifrazione (2)



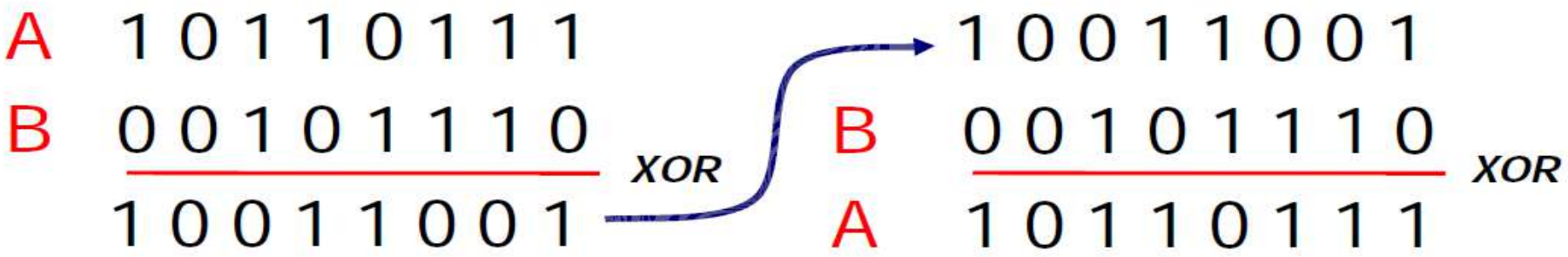
Stesso algoritmo

Sottochiavi in ordine inverso

Proprietà XOR

- E' una operazione reversibile:
  - reciproco annullamento di due ***XOR*** identici
  - non comporta perdita di informazione

(A XOR B) XOR B = A
(A XOR B) XOR A = B



C=A XOR **B** ⟹ **A = C** XOR **B**

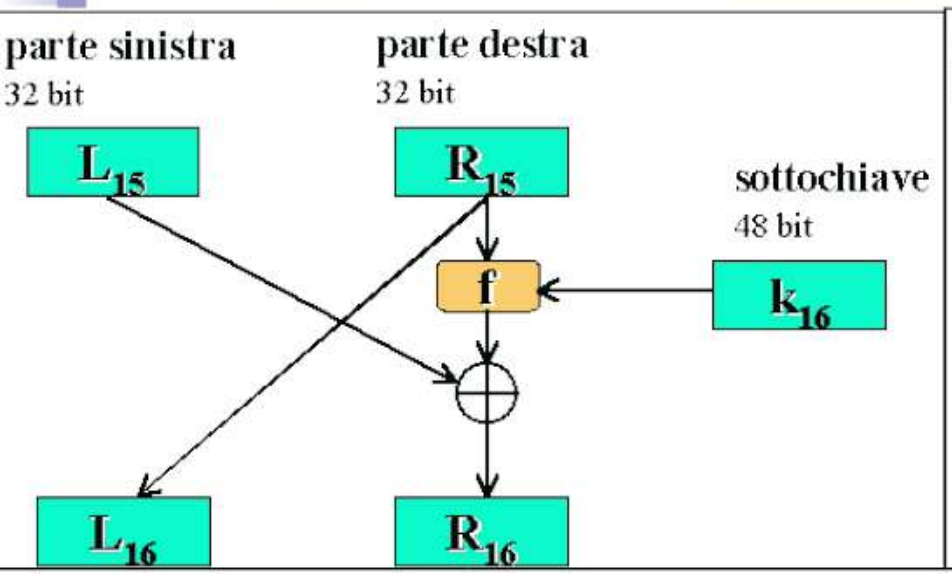


Fig.A (ultima iterazione cifratura)

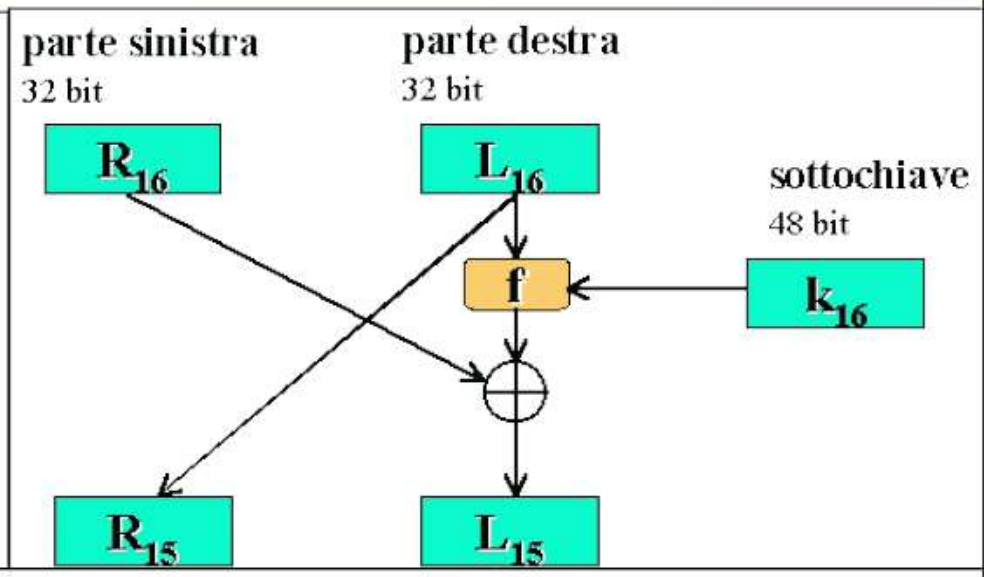


Fig.B (prima iterazione decifratura)

- Consideriamo il messaggio cifrato $R16\ L16$ a meno della permutazione finale IP^(-1). Dalla Figura A si ricavano le seguenti relazioni:

$$L16 = R15 \text{ e } R16 = L15 \oplus f(R15, K16)$$

- Da qui riscrivendo le due equazioni possiamo ricavare i *valori precedenti*:

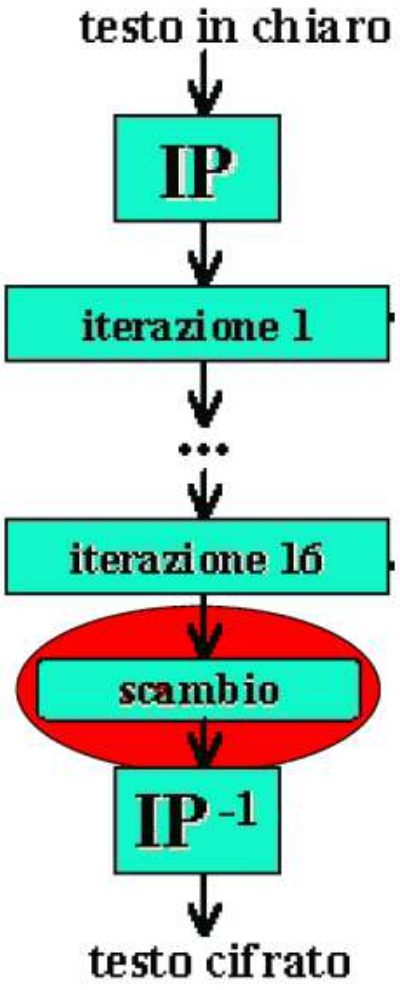
$$R15 = L16 \text{ e } L15 = R16 \oplus f(R15, K16) = R16 \oplus f(L16, K16)$$

- In conclusione: da $R16$ e $L16$ abbiamo ricavato $R15$ ed $L15$ (Figura B)

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

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

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



Chiave – Schedulazione inversa (1)

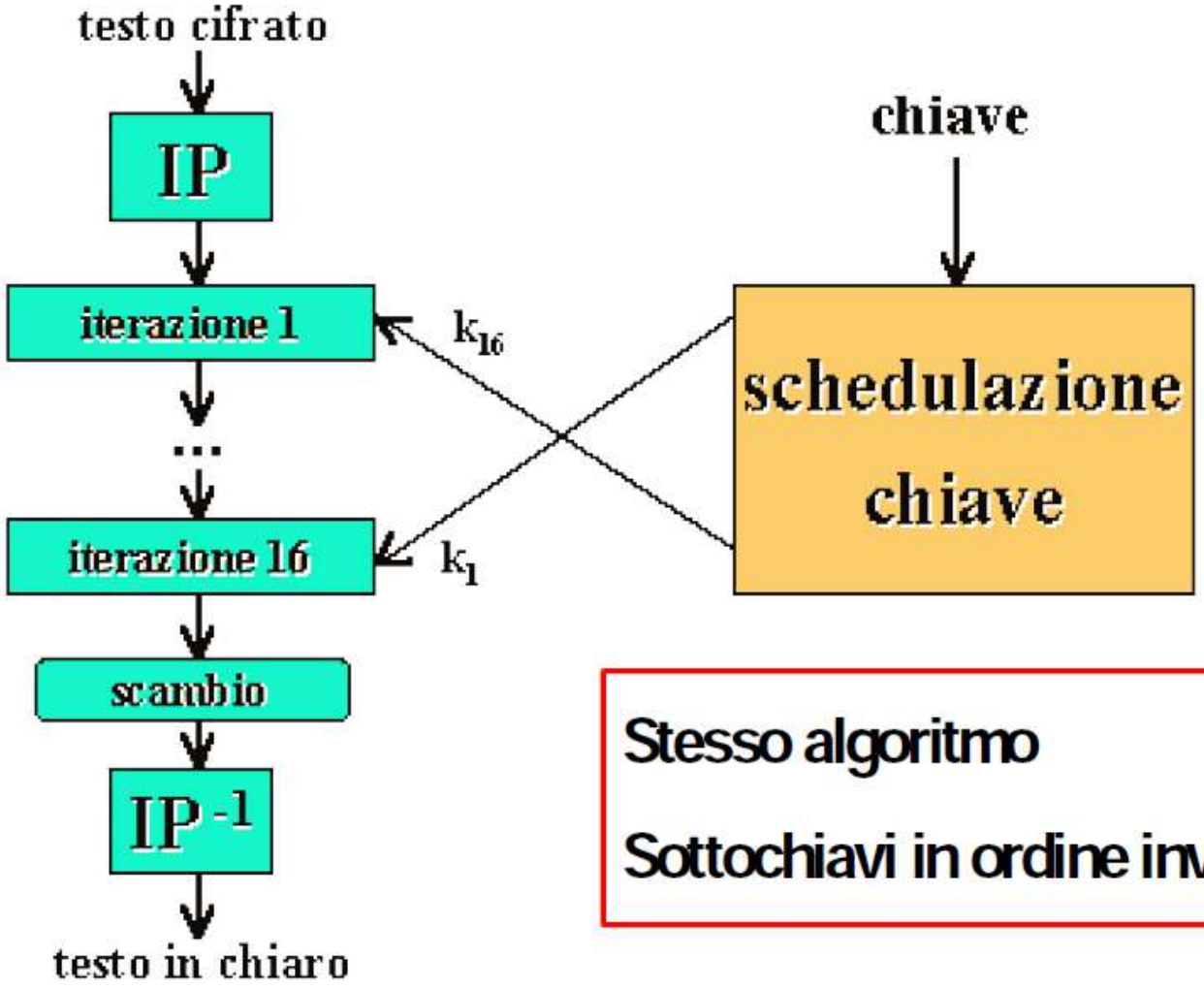
- Dal momento che per la fase di decifratura le chiavi devono essere schedulate in ordine inverso rispetto alla cifratura, dobbiamo effettuare delle variazioni al dispositivo di schedulazione definito in precedenza.

- In particolare dobbiamo fare in modo che pur prendendo in input la chiave *K*, tale dispositivo produca le chiavi nell'ordine desiderato:

  - Per ottenere ciò invertiamo lo shift che veniva applicato alle semi-chiavi *Li* e *Ri* effettuandolo verso destra anziché verso sinistra
  - Applichiamo la tabella di ***LS*** in ordine inverso
  - In questo modo al primo passo di decifrazione ottengo da ***LS*** la sedicesima sotto-chiave di cifratura.

- Per ottenere la i-esima chiave **non** devo ricalcolare tutte le i-1 che la precedono!

Chiave – Schedulazione inversa (2)



Stesso algoritmo

Sottochiavi in ordine inverso

La Sicurezza del DES

Sin dalla sua presentazione la **sicurezza del DES** è stata messa in discussione; con una chiave di 56 bit le chiavi possibili sono 2^{56} numero enorme, assolutamente fuori della portata per un essere umano, ma non di quella dei moderni computer. $2^{56} = 72.057.594.037.927.936$

In altre parole il DES non è affatto al sicuro dal più semplice degli attacchi crittanalitici, quello **esaustivo** (in inglese: ***brute-force***) che semplicemente **prova** una per una **tutte le chiavi**.

Nel 1993 Wiener presentò un progetto di computer dedicato in grado di decrittare il DES, unico difetto il costo stimato in un milione di dollari!

Nel **1998** un gruppo di tre aziende Cryptographic Research, Advanced wireless technologies, Electronic Frontier Foundation comunicò di aver realizzato ***DES Cracker*** una macchina per la ricerca delle chiavi dal costo di 250.000$ in grado di **forzare** la **chiave** del **DES** in **56 ore**. Comunicazione che mostrò definitivamente che la **chiave** di **56 bit era troppo corta**.

Il DES a 64 bit ha dovuto così cedere il passo ad un DES a **128 bit** e al **triplo DES**. Nel 2001 è stato presentato il nuovo protocollo **AES** destinato a sostituire progressivamente il DES.

Ricerca esaustiva

- Numero chiavi DES = $2^{56} \approx 7{,}2056 \cdot 10^{16}$
- Un computer a 5.000 Mhz che testa una chiave per 10 cicli di clock impiega

  144.115.188 secondi ≈ 834 giorni ≈ 2 anni e 3 mesi

  per provare $2^{55} \approx 3{,}6 \cdot 10^{16}$ chiavi

5.000.000.000/10 = 500.000.000 di chiavi al secondo
72.056.000.000.000.000/500.000.000= 144.115.188 secondi

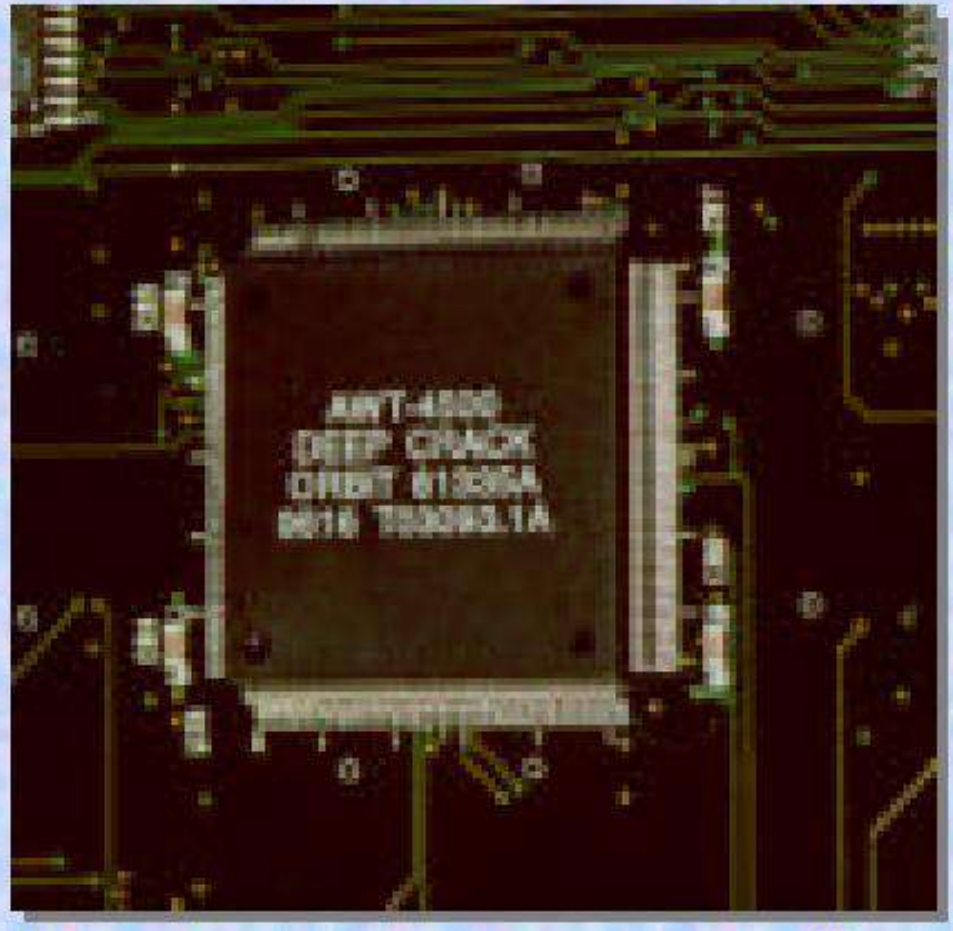
Deep Crack: Unità di ricerca

- Clock di 40Mhz
- Una decifratura in 16 cicli di clock
- Numero chiavi provate al secondo

$$\frac{40.000.000}{16} = 2.500.000$$

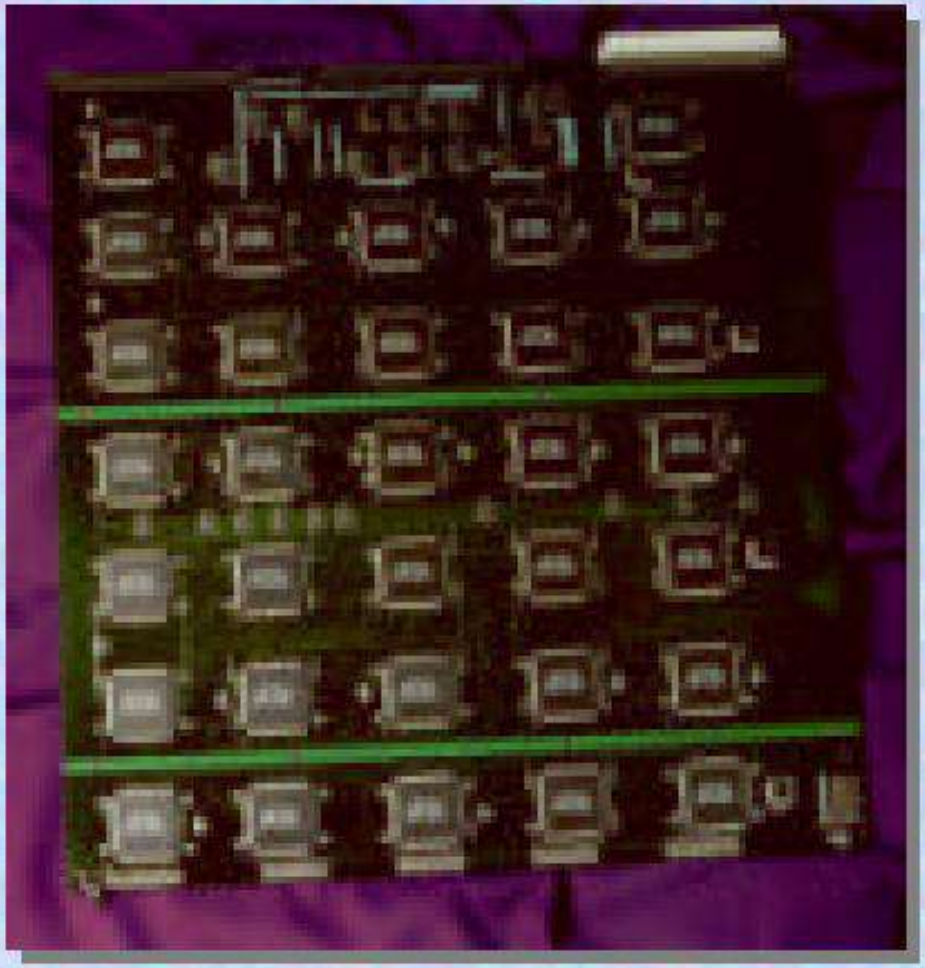
Chip

- 24 unità di ricerca
- Prova 24 · 2.500.000 = 60.000.000 chiavi al sec.
- Prova tutte le chiavi in 13.900 giorni (≈38 anni)



Board

- 64 chip
- 32 per faccia
- 40 cm X 40 cm
- Prova 64 · 60.000.000 = 3.840.000.000 chiavi al sec.
- Prova tutte le chiavi in ≈218 giorni



Chassis

- 12 schede
- Prova 12 · 3.840.000.000 = 46.080.000.000 chiavi al sec.
- Prova tutte le chiavi in ≈18 giorni

EFF DES Cracker

Prestazioni

Device	Quanti nella prossima device	Chiavi/sec	Num. medio Giorni per ricerca
Unità di ricerca	24	2.500.000	166.800
Chip	64	60.000.000	6.950
Board	12	3.840.000.000	109
Chassis	2	46.080.000.000	9,05
EFF DES Cracker		92.160.000.000	4,524

RSA

- L'algoritmo RSA (dal nome degli inventori Rivest, Shamir e Adleman) è il più famoso algoritmo di crittografia a chiave pubblica
- Inventato nel 1977, poco dopo l'algoritmo di Diffie-Hellman
- Due componenti principali
  - Algoritmo di generazione delle chiavi
  - Algoritmo crittografico vero e proprio

RSA – generazione delle chiavi

- Scegliere due numeri primi *p* e *q*
- Calcolare $n = pq$
- Scegliere *e*, coprimo e più piccolo di $(p-1)(q-1)$
- Calcolare *d* tale che $de \equiv 1 \mod (p-1)(q-1)$

- La coppia (n, e) è la chiave pubblica
- La coppia (n, d) è la chiave privata

- Non è possibile risalire facilmente dalla chiave pubblica a quella privata (e viceversa), in quanto servirebbe conoscere il numero $(p-1)(q-1)$, e questo implica fattorizzare *n* nei suoi fattori *p* e *q* (problema difficile)

Funzione di Eulero

$\Phi(n) = (p-1) \times (q-1)$.

In matematica, la **funzione ϕ di Eulero** o semplicemente **funzione di Eulero** o **toziente**, è una **funzione** definita, per ogni intero positivo **n**, come il numero degli interi **compresi** tra **1** e **n** che sono **coprimi** con **n**. Ex: **$\Phi(8) = 4$ {1,3,5,7}**.

Teorema di Eulero: Se **a** è un numero **coprimo** con **n**, allora:

$$a^{\Phi(n)} \equiv 1 \mod n$$

$d = e^{-1} \mod \Phi(n)$ inverso moltiplicativo di **e** modulo **$\Phi(n)$.**

Numeri coprimi

- Cosa significa 'coprimo'?
- *a* è coprimo di *b* se il massimo comune divisore tra *a* e *b* è 1
- Ad es. 7 e 15 sono coprimi, mentre 8 e 10 no (hanno in comune il divisore 2)

- Nota: se *a* è primo, allora è coprimo di qualsiasi numero che non sia diviso da *a*
- Ad es. 7 è coprimo di tutti i numeri che non sono multipli di 7

RSA – esempio di generazione chiavi

- Siano p=3, q=11
- n=pq=33, (p-1)(q-1)=20
- Scegliamo e = 7 (7<20, 7 coprimo di 20)
- d = 3, infatti 3*7 = 21 ≡ 1 mod 20

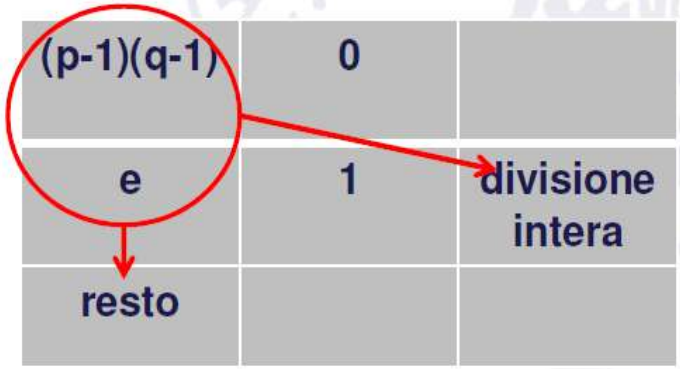
- La chiave pubblica è (33, 7)
- La chiave privata è (33, 3)

Come calcolare d?

- Metodo di Euclide esteso
- Si inizia con questa tabella:

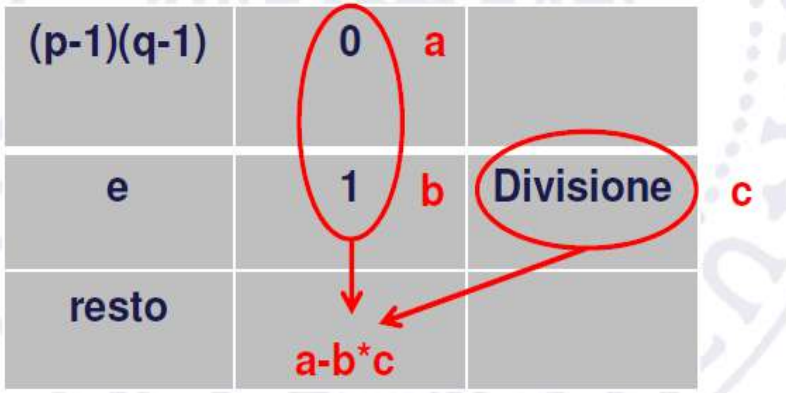
(p-1)(q-1)	0	
e	1	

- Si calcolano il risultato intero e il resto della divisione dei due numeri nella prima colonna e li si salvano nella tabella:



Come calcolare d? / 2

- Nella seconda colonna, terza riga si scrive il valore della seconda colonna, prima riga meno quello della seconda colonna, seconda riga moltiplicato per il risultato intero della divisione appena calcolato.



*Errore frequente: calcolare (a – b)*c anziché a – b*c*

Come calcolare d? / 3

- Il procedimento si ripete, aggiungendo nuove righe alla tabella e calcolandone i valori usando le due righe precedenti. Ci si ferma quando nella prima colonna compare un 1.
- Il risultato nella seconda colonna è il valore d cercato (se negativo, sommare (p-1)(q-1))

- Esempio: trovare d tale che

  $d * 7 \equiv 1 \bmod 20$

Metodo di Euclide esteso

20	0	
7	1	

Metodo di Euclide esteso

20	0	
7	1	2
6		

2 = risultato intero della divisione 20 / 7

6 = resto della divisione 20 / 7

Metodo di Euclide esteso

20	0	
7	1	2
6	-2	

$$0 - 1 * 2 = -2$$

Metodo di Euclide esteso

20	0	
7	1	2
6	-2	1
1		

7/6 = 1 col resto di 1

Metodo di Euclide esteso

20	0	
7	1	2
6	-2	1
1	3	

1 - (-2*1) = 3

d = 3

Infatti $3 * 7 \equiv 1 \bmod 20$

Metodo di Euclide esteso / 2

- Altro esempio. Trovare d tale che
  d * 23 ≡ 1 mod 120

120	0	
23	1	5
5	-5	4
3	21	1
2	-26	1
1	47	

Metodo di Euclide esteso / 3

- Altro esempio. Trovare d tale che
  d * 7 ≡ 1 mod 60

60	0	
7	1	8
4	-8	1
3	9	1
1	-17	

d = -17 + 60 = 43 (se il numero è negativo, si somma il modulo)

(un altro metodo semplice per calcolare d…)

- de mod (p-1)(q-1) = 1
- de = k(p-1)(q-1) + 1
- d = (k(p-1)(q-1) + 1) / e

Provo k=1, 2, 3 …

Finché non trovo un valore INTERO per d

Esercizio

- p=7, q=13, e=…
- Calcolare la chiave pubblica (n,e) e privata (n,d)

n = **p** x **q** = **91**
(p-1)x(q-1)=6x12=72
Scelgo **e = 11** (11<72 e coprimo con 72)
d = k x 72 + 1 / 11 per k=1,2,3,…….
k=1 => 72+1 / 11 **non INTERO**
k=2 => 144+1 / 11 **non INTERO**
……
……
k=9 => 648+1 / 11 = 649 / 11 = **59** **INTERO**

Chiave pubblica = (91,11)
Chiave privata = (91,59)

[esercizio]

- p=7, q=13, e=11
- n = p*q = 91
- (p-1)*(q-1) = 72
- d*11 mod 72 = 1
- d = -13 + 72 = 59

72	0	
11	1	6
6	-6	1
5	7	1
1	-13	

- 59*11 mod 72 = 1
- Pubblica: (91,11) privata: (91, 59)

RSA – cifratura e decifratura

- Dato un messaggio *m* (0 < m < n)
- Cifratura: calcolare $c = m^e \bmod n$
- Decifratura: calcolare $m = c^d \bmod n$

(n,d) (n,e)
rispettivamente chiave privata e pubblica del destinatario (Bob)

(nota: si basa sull'ipotesi che l'esponenziazione modulare sia un problema difficilmente invertibile – ipotesi RSA).

RSA – Esempio di cifratura e decifratura

- Chiave pubblica: (33, 7) chiave privata: (33, 3)
- m = 15

- Cifratura: $c = m^e \bmod n = 15^7 \bmod 33 = 27$
- Decifratura: $m = c^d \bmod n = 27^3 \bmod 33 = 15$

Si considera il **testo in chiaro** come una **sequenza di bit** e si divide in **blocchi** di **k** bit dove **$2^k<n$**.
Quindi k=5 ($2^5<33$)
m=01111=15 => c=27=11011

Dimostrazione del funzionamento di RSA

- **Teorema del toziente di Eulero**

Se *m* è coprimo di *pq* allora…

$$m^{(p-1)(q-1)} \equiv 1 \bmod pq$$

Dimostrazione del funzionamento di RSA

- $c^d \bmod n = [m^e \bmod n]^d \bmod n = m^{de} \bmod n$
- Siccome $de \equiv 1 \bmod (p-1)(q-1)$ allora
- $m^{de} \bmod n = m^{k(p-1)(q-1)+1} \bmod n$

$$= m \cdot [m^{(p-1)(q-1)}]^k \bmod n$$

Per il teorema del toziente(*), ricordando che $n=pq$

$$= m \cdot 1^k \bmod n$$

$$= m \bmod n$$

- Quindi $c^d \bmod n = m \bmod n = m$ (perché $m<n$)

(*) se m è coprimo di n. In realtà il procedimento vale per tutti gli m, ma la dimostrazione in questo caso va oltre gli scopi di questo corso

Applicazioni reali di RSA

- Per garantire la non invertibilità delle funzioni utilizzate, è importante usare numeri sufficientemente grandi. Nelle applicazioni attuali, solitamente n è un numero di almeno 1024 bit (poco più di 300 cifre decimali).

$2^{1024} \approx (2^{10})^{102} \approx (10^{3})^{102} \approx 10^{306}$ **(oltre 300 cifre decimali)**

Rivest, Shamir e Adleman

- Appena scoperto l'RSA, i tre crittografi lanciarono una sfida alla comunità mondiale dei matematici, **cifrando un testo** e fornendo come **chiave pubblica** di **129** cifre:
- **N** = 114 381 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242 362 562 561 842 935 706 935 245 733 897 830 597 123 563 958 705 058 989 075 147 599 290 026 879 543 541
- Chiedevano di fattorizzare il numero nei **due numeri primi** **P** e **Q** che lo componevano per riuscire a decrittare il messaggio, e il premio era di 100$.
- Tale sfida fu risolta 17 anni dopo il 26 Aprile 1994, da un gruppo di 600 volontari di tutto il mondo. I due fattori erano:
- **P** = 3 490 529 510 847 650 949 147 849 619 903 898 133 417 764 638 493 387 843 990 820 577
- **Q** = 32 769 132 993 266 709 549 961 988 190 834 461 413 177 642 967 992 942 539 798 288 533

Conclusioni RSA

- Il **testo in chiaro** viene visto come una **stringa di bit** e viene diviso in **blocchi** costituiti da **k bit**, dove **k** è il **più grande intero** che soddisfa la disequazione $2^k < n$ ($k=\log_2 n$).
- **Il codice RSA viene considerato sicuro perché non è ancora stato trovato il modo per fattorizzare numeri primi molto grandi, che nel nostro caso significa riuscire a trovare p e q conoscendo n.**
- Nel corso degli anni l'algoritmo RSA ha più volte dimostrato la sua **robustezza**: in un esperimento del 1994, coordinato da Arjen Lenstra dei laboratori Bellcore, per "rompere" una chiave RSA di 129 cifre, svelando il meccanismo con cui quella chiave generava messaggi crittografati, sono stati necessari 8 mesi di lavoro coordinato effettuato da 600 gruppi di ricerca sparsi in 25 paesi, che hanno messo a disposizione 1600 macchine da calcolo, facendole lavorare in parallelo collegate tra loro attraverso Internet.
- Data la mole delle risorse necessarie per rompere la barriera di sicurezza dell'algoritmo **RSA**, è chiaro come un **attacco alla privacy** di un sistema a **doppia chiave non sia praticamente realizzabile**. Inoltre, nell'esperimento era stata utilizzata una chiave di 129 cifre mentre i programmi di crittografia attualmente a disposizione prevedono chiavi private con una "**robustezza**" che raggiunge e **supera i 2048 bit**, risultando quindi praticamente **inattaccabili**, visto anche che l'ordine di grandezza dei tempi necessari alla rottura di chiavi di questo tipo è esponenziale e passa in fretta da qualche giorno a qualche **centinai di anni**.
- **$2^{2048} \approx (2^{10})^{204} \approx (10^3)^{204} \approx 10^{612}$ (oltre 600 cifre decimali)**

3DES (Triple DES) - 1

Per superare la **debolezza** del **DES** a 56 bit una semplice idea è quella di usare N volte il DES portando la **chiave** a **Nx56 bit**.

Nel 1999 fu introdotto il cifrario **3DES** (**Triple DES**) che consiste nel applicare **tre** volte l'algoritmo **DES** alternando la funzione di **cifratura** ***Cifra(chiave,blocco)*** e quella di **decifratura** ***Decifra(chiave,blocco)***, usando tre diverse chiavi a 56 bit.

La cifratura mediante **triplo DES** si articola nei seguenti passi:

C = E(K3,D(K2,E(K1,M)))

dove **M** è il messaggio in **chiaro**, E(K1,M) indica la cifratura di M mediante la chiave K1, D(K2,E(K1,M)) indica la decifratura del messaggio precedentemente criptato, utilizzando questa volta la chiave k2, ed infine, E(K3,D(K2,E(K1,M))) indica la cifratura del messaggio appena decriptato, però mediante la chiave K3. Per capire meglio tale sequenza di operazioni, è conveniente illustrare la seguente schematizzazione:

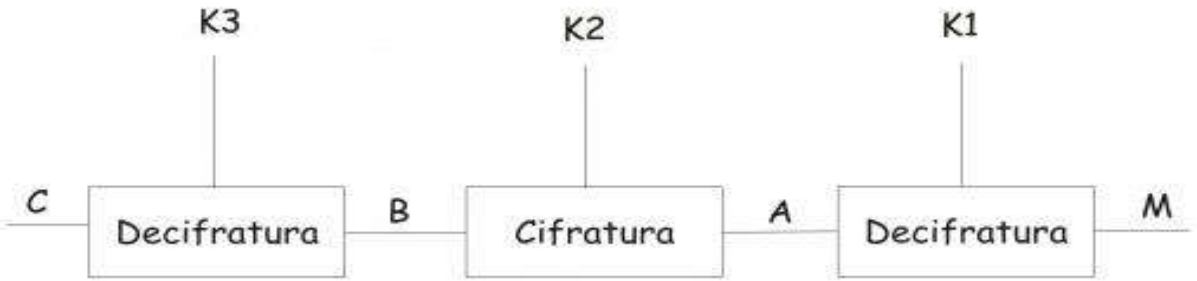
3DES (Triple DES) - 2

La **decifratura** consiste nelle stesse operazioni della **cifratura**, ma utilizza le **chiavi** in **ordine inverso**:

M = D(K1, E(K2, D(K3,C)))

dove **C** è il messaggio **cifrato**.

Tale operazione viene rappresentata di seguito:



In questo modo la chiave viene ad essere di **3x56 = 168 bit** con molta maggiore sicurezza; il vantaggio è che non è necessario progettare ed implementare nuovi algoritmi o circuiti, si riutilizzano quelli del DES, tali e quali.

In base alla **scelta** delle **chiavi** il **3DES** offre tre alternative:

- Le **tre** chiavi K1, K2 e K3 sono **diverse** ed indipendenti (sicurezza = 168 bit).
- **Due** chiavi **uguali** (K1=K3) ed una diversa (K2) (sicurezza = 112 bit).
- Le **tre** chiavi **tutte uguali** (K1=K2=K3) (sicurezza=56 bit come il DES).

Oggi il **3DES** viene utilizzato con alcune varianti nelle transazioni commerciali elettroniche (**VISA**, **Mastercard**..).

IDEA (International Data Encryption Algorithm)

Nel 1991 fu proposto il cifrario **IDEA** in sostituzione del **DES**, proprio quando si temeva che lo stesso non avrebbe resistito per molto agli attacchi dei crittoanalisti.

Il metodo progettato in **Svizzera** al politecnico di **Zurigo** ad opera di due famosi ricercatori **Xuejia Lai** e **James Massey** si basa su concetti **simili al DES** con chiave a **128 bit** dove i blocchi del messaggio a **64 bit** vengono elaborati in **8 iterazioni** usando operazioni di **XOR**, **somma** e **moltiplicazione modulo 2^{16}**.

I 64 bit del messaggio vengono divisi in 4 gruppi di 16 bit mescolati con 6 chiavi di 16 bit estratte dalla chiave di 128 bit.

L'algoritmo è uno dei più resistenti e ad oggi non risulta che sia stato violato: i progettisti di **IDEA** lo hanno realizzato in modo che fosse praticamente immune ad attacchi condotti con la **crittoanalisi differenziale** (la crittoanalisi **differenziale** di una funzione crittografica è lo studio di come le differenze nei dati forniti in ingresso alla funzione **possono incidere sulle differenze risultanti** in uscita dalla stessa: l'attaccante deve essere in grado di ottenere i messaggi cifrati relativi a testi in chiaro di sua scelta). Per attacchi a **forza bruta** si calcola che la violazione di una chiave a 128 bit impieghi nel migliore dei casi 2×10^{15} anni per la riuscita. Attualmente è uno dei cifrari a chiave segreta **più utilizzati** per quanto riguarda i software commerciali di crittografia vista la sua **velocità di codifica** e **decodifica** e la sua **elevata sicurezza**.

AES (Advanced Encryption Standard) - 1

Nel **1997** il **NIST** (**National Institute of Standards and Technology**) ha cominciato a cercare ufficialmente un successore per l'oramai anziano standard di crittografia **DES** organizzando un concorso.

Si presentarono 15 candidati: dopo lunghi test e rigorose analisi nel 2000 vinse l'algoritmo di **Rijndael** proposto da due crittografi belgi, **Joan Daemen** e **Vincent Rijmen**. L'**AES - Rijndael** fu certificato nel 2001 come nuovo standard di cifratura. Il nome **Rijndael** è una **sintesi** dei nomi dei suoi inventori.

Si basa su tre caratteristiche fondamentali:

- Resistenza contro gli attacchi.
- Velocità e compattezza del codice su un ampia gamma di piattaforme.
- Semplicità progettuale.

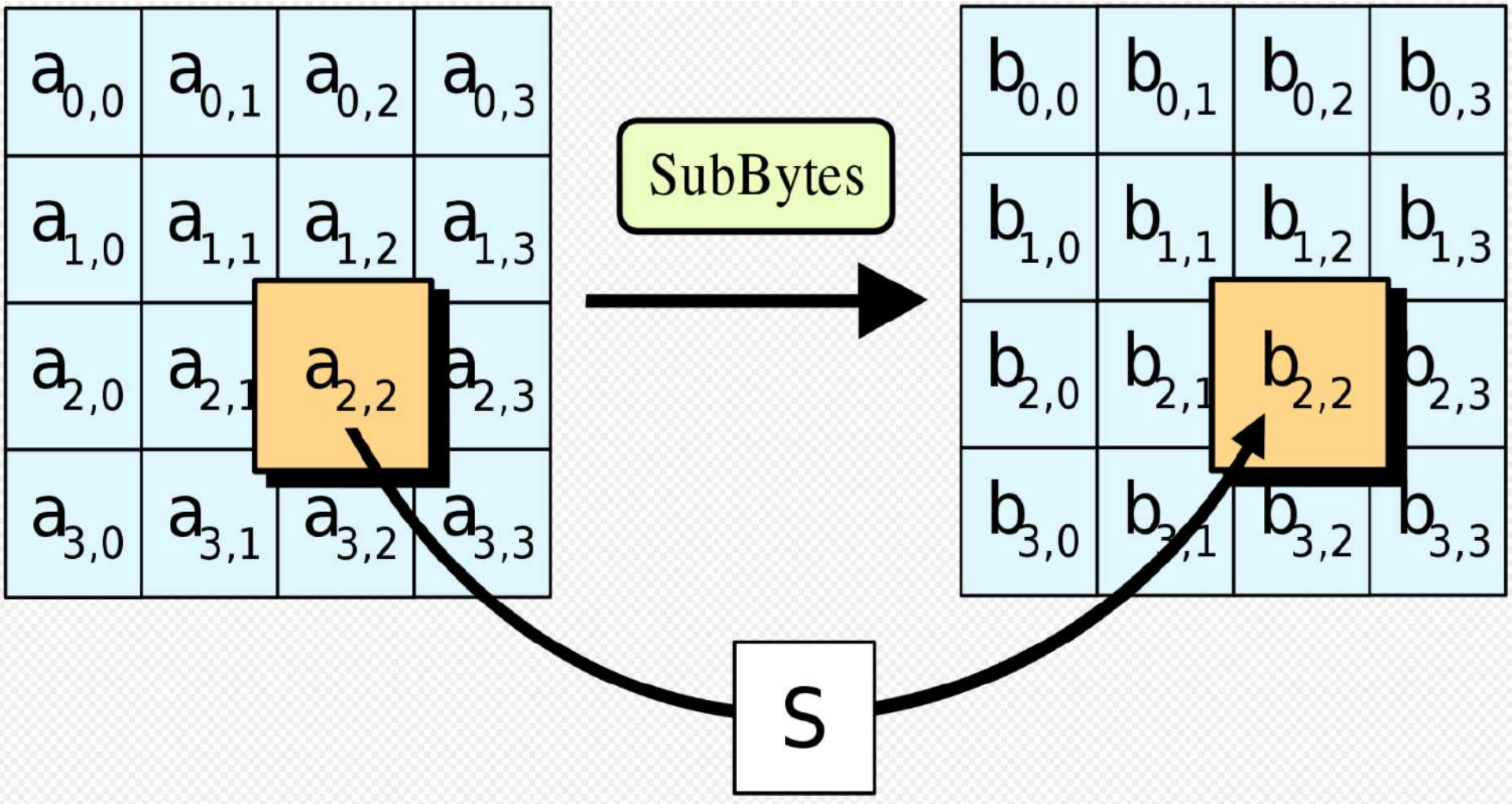
AES è un cifrario a blocchi con **lunghezza del blocco di 128 bit**, ma può avere chiavi indipendenti con lunghezza di 128, 192 e 256 bit, ed effettua una combinazione di **permutazioni** e **sostituzioni**. Come il **DES** prevede la ripetizione di numerosi cicli identici: per l'**AES** a 128, 192 e 256 bit sono previste rispettivamente 10, 12 e 14 **ripetizioni** del ciclo base (**round**).

Ogni blocco di 128 bit è diviso in 16 bytes da 8 bit, che dobbiamo immaginare disposti su una **matrice 4x4 byte**.

AES (Advanced Encryption Standard) - 2

Le **quattro operazioni** che costituiscono **ogni round** sono le seguenti:

1. Substitute Bytes: ogni **byte** viene **trasformato** mediante una permutazione non lineare di byte che vengono mappati tramite una tabella **S-box** a 8 bit definita in **AES** stesso.



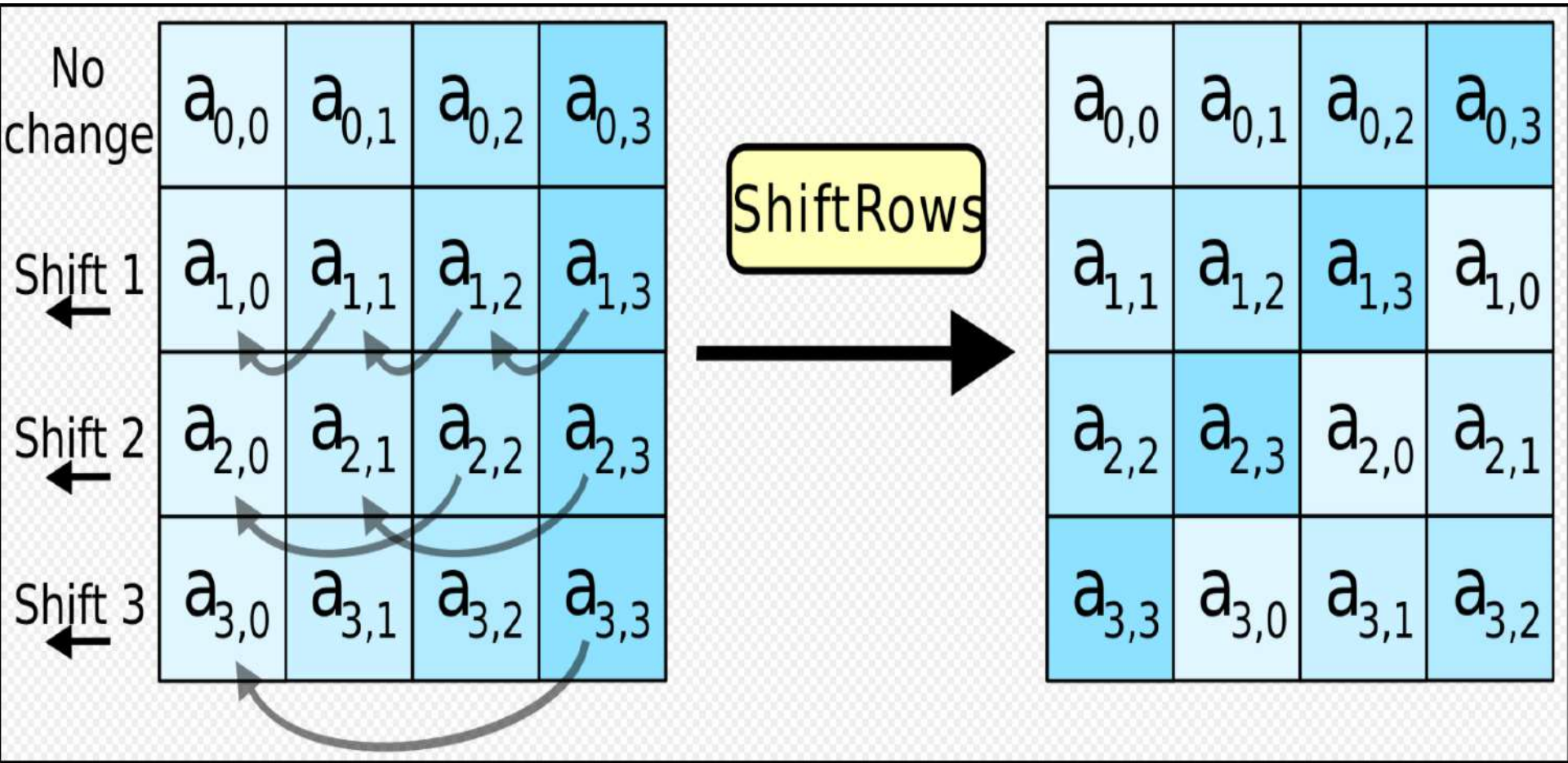
AES (Advanced Encryption Standard) - 3

Tabella **S-box** ad 8 bit usata nell'operazione di **Substitute Bytes**

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

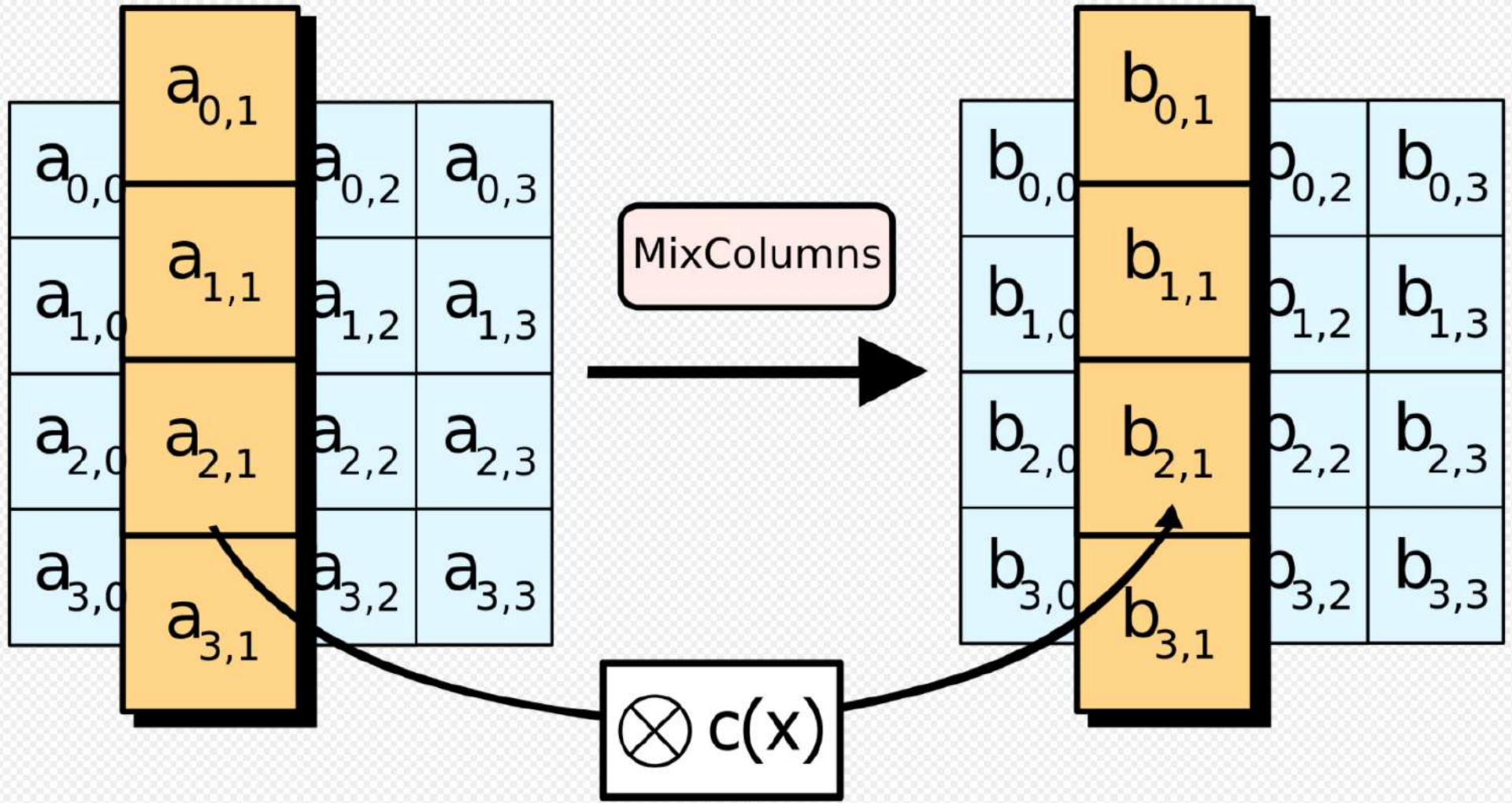
AES (Advanced Encryption Standard) - 4

2. Shift Rows : il passaggio **Shift Rows** provvede a scostare le righe della matrice di un parametro dipendente dal numero di riga. Nell'**AES** la prima riga resta invariata, la seconda viene spostata di un posto verso sinistra, la terza di due posti e la quarta di tre. In questo modo l'ultima colonna dei dati in ingresso andrà a formare la **diagonale** della matrice in uscita.



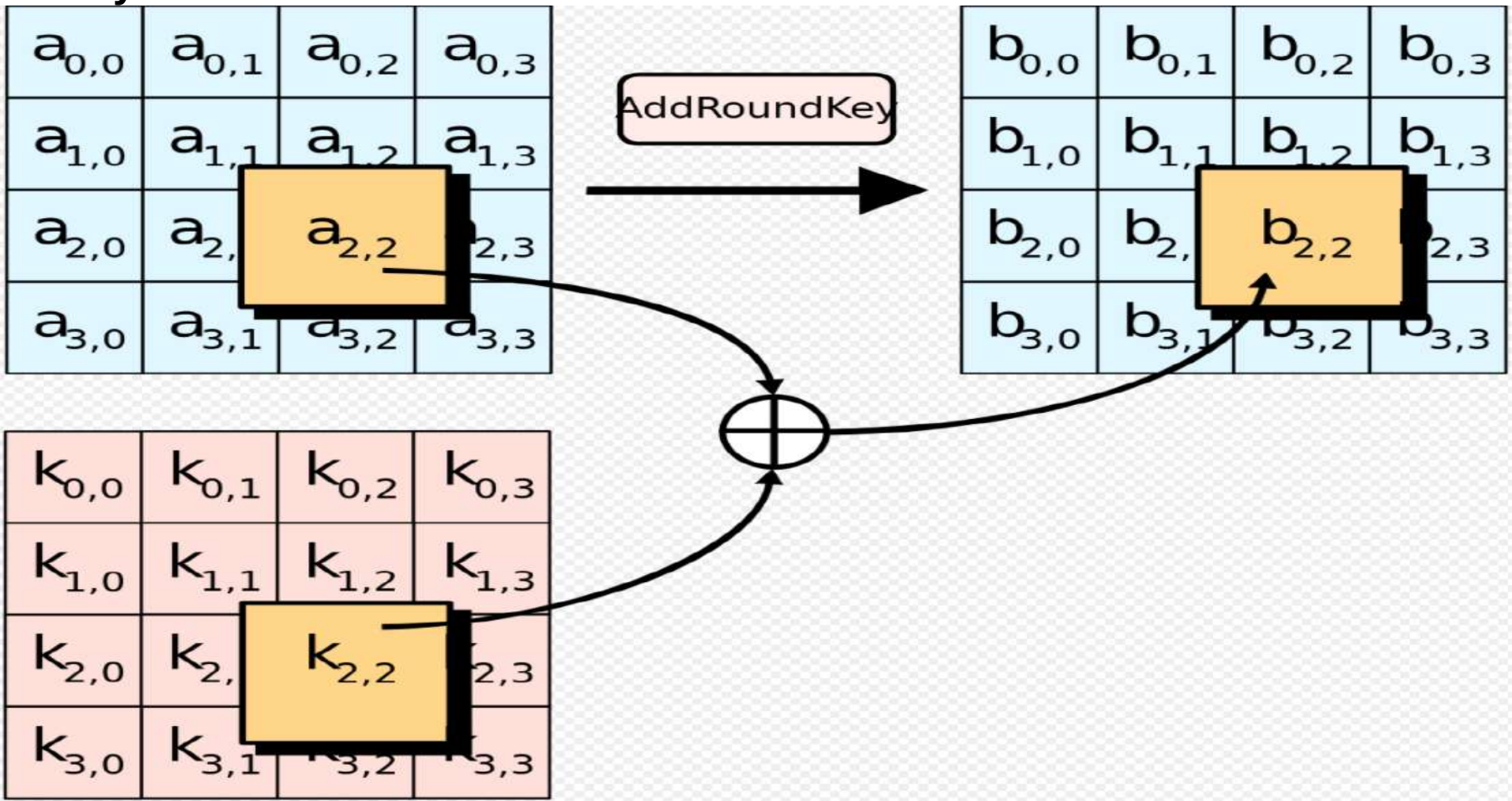
AES (Advanced Encryption Standard) - 5

3. Mix Columns: ogni colonna viene trasformata mediante un'operazione che può essere vista come una **moltiplicazione matriciale** con una particolare matrice generata da un **polinomio prefissato** c(x).



AES (Advanced Encryption Standard) - 6

4. Add Round Key: il passaggio **Add Round Key** combina con uno **XOR** la **chiave di sessione** con la **matrice** ottenuta dai passaggi precedenti . Una chiave di sessione viene **ricavata** dalla **chiave primaria** ad ogni round (con dei passaggi più o meno semplici, ad esempio uno shift di posizione dei bit) grazie al **Key Schedule.**



AES (Advanced Encryption Standard) - 7

La **National Security Agency** (**NSA**) segnalava che tutti i finalisti del processo di standardizzazione erano dotati di una sicurezza sufficiente per **diventare** l'**AES**, ma che fu scelto il **Rijndael** per via della sua flessibilità nel trattare chiavi di lunghezza diversa, per la sua semplice implementazione in **hardware** e in **software** e per le sue basse richieste di memoria che ne consentono un'**implementazione anche** in dispositivi con **scarse risorse come le smart card**. Anche considerando che la potenza dei computer aumenta nel tempo, servirà ancora molto tempo prima che una chiave da 128 bit sia attaccabile con il metodo **forza bruta**.

Nel 2011 è stato pubblicato Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger (2011) ***Biclique Cryptanalysis of the Full AES*** che richiede qualcosa come 2^{126} operazioni per forzare una chiave AES a 128 bit, 2^{190} per AES 192 bit e 2^{254} per AES a 256 bit.

Il **livello** di **sicurezza** di **AES** sembra quindi **molto elevato**; per i documenti del governo USA, **AES 128 bit** è considerato sufficiente per i documenti classificati "**Secret**", mentre per i "**Top Secret**" occorre **AES 192 bit** o meglio ancora **AES 256 bit**.

CAST

CAST (**Carlisle Adams and Stafford Tavares**)

Algoritmo con **blocchi** a 64 o 128 bit e **chiavi** da 128 a 256 bit.

Esistono infatti due versioni:

- **CAST-128**
- **CAST-256**.

Questa è una rete di **Feistel**. I passaggi sono costituiti da 3 gruppi di operazioni dove la differenza fra di essi è minima e consiste in un'unica operazione (addizione, sottrazione o XOR).

Algoritmo di Diffie-Hellman - 1

Nel **1976** due giovani ricercatori statunitensi, **Whitfield Diffie** e **Monte Hellman**, idearono un sistema per cui due individui scambiavano messaggi cifrati, senza per questo dover scambiare alcuna chiave.

Il protocollo **Diffie-Hellman** consente a due entità di **generare separatamente** una **chiave simmetrica comune** utilizzando un canale di comunicazione insicuro (**pubblico**) a partire da un valore di un **pre-master condiviso**.

Algoritmo di Diffie-Hellman - 2

Siano Alice e Bob i due interlocutori, entrambi a conoscenza di una **pre-master** che **consiste** in **due numeri**:

- una **base g** (**gruppo generatore**).
- un **numero primo p** utilizzato per il calcolo dei **moduli**.

Alice **genera** un **numero casuale a** che mantiene **segreto** ed esegue il calcolo $A = g^a \bmod p$ inviando in chiaro il valore di **A** a Bob.

Bob **genera** un **numero casuale b** che mantiene **segreto** ed esegue il calcolo $B = g^b \bmod p$ inviando in chiaro il valore di **B** a Alice.

Alice utilizza il numero ricevuto **B** per calcolare $K = B^a \bmod p$

Bob utilizza il numero ricevuto **A** per calcolare $K = A^b \bmod p$

I **due risultati coincidono** e rappresentano la **chiave simmetrica** che Alice e Bob utilizzeranno per comunicare.

Algoritmo di Diffie-Hellman - 3

Alice calcola:

$K = B^a \bmod p = (g^b \bmod p)^a \bmod p = (g^b)^a \bmod p = g^{ba} \bmod p$

Bob calcola:

$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$

Alice e Bob trovano lo stesso risultato perché

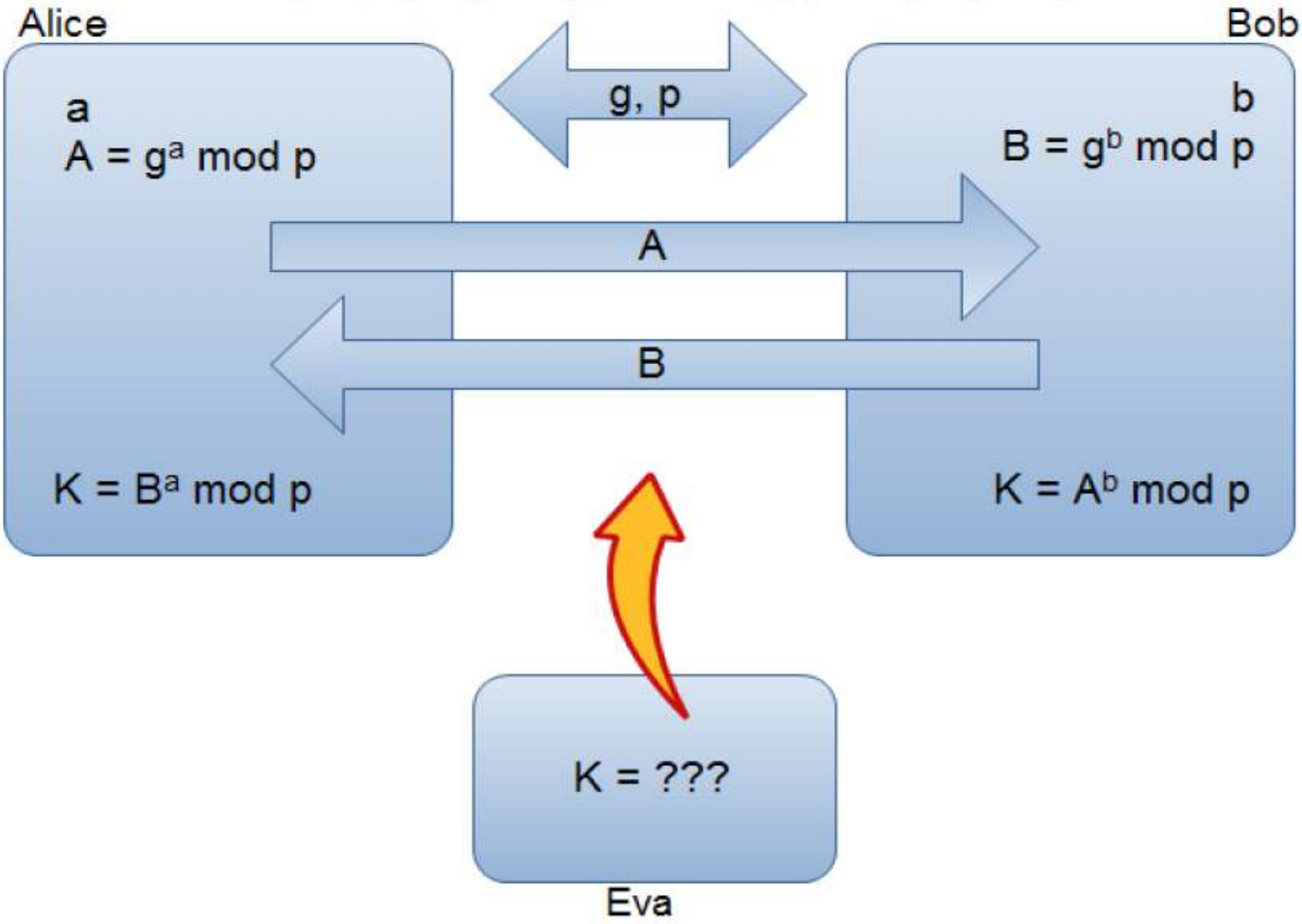
g^{ab} e g^{ba} sono uguali!

Si noti come solo a, b e $g^{ab} = g^{ba}$ sono segreti.

Tutti gli altri numeri sono mandati in chiaro, ossia pubblici.

Una volta che Alice e Bob calcolano la chiave segreta, essa può esser usata come chiave di cifratura simmetrica, conosciuta solo a loro, per mandare messaggi cifrati (3DES, IDEA, AES) tramite il canale di comunicazione in chiaro.

Algoritmo di Diffie-Hellman - 4



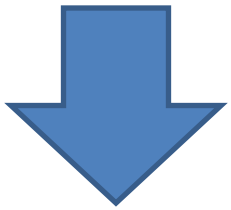
Algoritmo di Diffie-Hellman - 5

Alice e Bob hanno condiviso un **pre-master segreto** (il numero **K**) **senza comunicarlo esplicitamente**!

L'attaccante Eva può osservare **A**, **B**, **g**, **p** ma questa informazione non è sufficiente per ricavare **K**!

K è calcolabile solo conoscendo **a** o **b**, che tuttavia sono **segreti** e non vengono **mai trasmessi**.

Ricavare **a** da **A** (invertendo $A = g^a \bmod p$) o analogamente **b** da **B** (invertendo $B = g^b \bmod p$) significa risolvere un **logaritmo discreto**



computazionalmente difficile!

Per valori di **g** di oltre 300 cifre e di **a** di oltre 100, la soluzione e, quindi, il riconoscimento della chiave, risulta difficilissima.

Algoritmo di Diffie-Hellman - 6

• Molte funzioni normalmente invertibili, diventano **non invertibili** nella versione **modulare**.

• **Esempio**: il logaritmo

$$a^b = c$$

Trovare **b** dati **a** e **c** *è* computazionalmente semplice (**logaritmo**: $b = \log_a(c)$ "*logaritmo di **c** in base **a***" che è l'inverso dell'esponenziale, della potenza).

$$a^b \bmod m = c$$

Trovare **b** dati **a**, **c** ed **m** *è* computazionalmente molto difficile! (**logaritmo discreto** di un numero **c** in base **a** che è l'inverso della **potenza discreta**).

Algoritmo di Diffie-Hellman - 7

Esempio 1

g = 7, p = 11 (pubblici)

a = 3 (Alice) segreto

b = 6 (Bob) segreto

Alice calcola $A = 7^3 \bmod 11 = 2$ e lo comunica a Bob

Bob calcola $B = 7^6 \bmod 11 = 4$ e lo comunica ad Alice

Alice calcola **K** $= 4^3 \bmod 11 = 9$

Bob calcola **K** $= 2^6 \bmod 11 = 9$

Algoritmo di Diffie-Hellman - 8

Esempio 2

g = 5, p = 23 (pubblici)

a = 6 (Alice) segreto

b = 15 (Bob) segreto

Alice calcola A = 5^6 mod 23 = 8 e lo comunica a Bob

Bob calcola B = 5^{15} mod 23 = 19 e lo comunica ad Alice

Alice calcola **K** = 19^6 mod 23 = 2

Bob calcola **K** = 8^{15} mod 23 = 2

Algoritmo di Diffie-Hellman - 9

Esempio 2 - Come calcolare le potenze in modulo

$5^{15} \bmod 23 = 5^8 \times 5^4 \times 5^2 \times 5^1 \bmod 23 =$

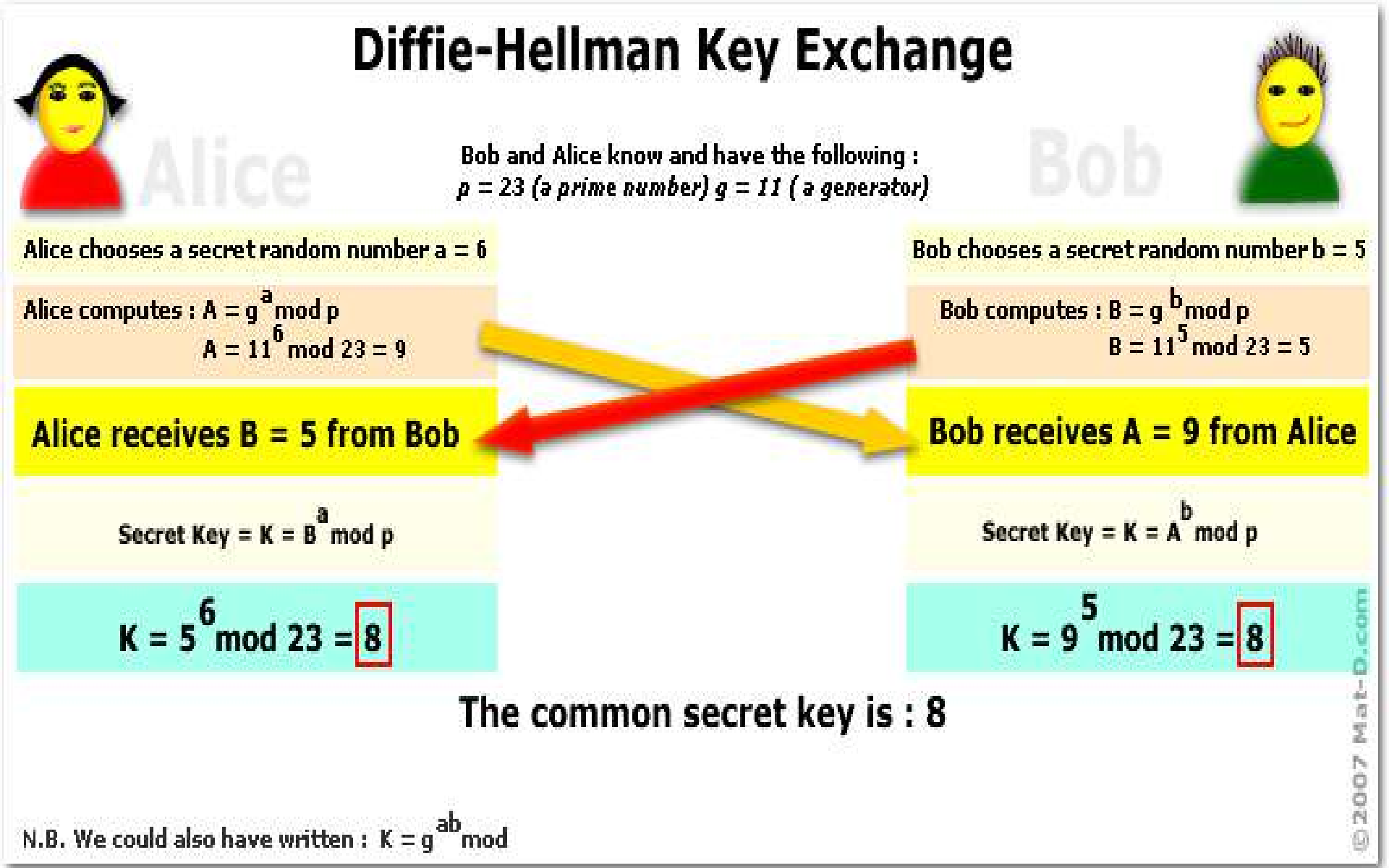
$[5^8 \bmod 23 \times 5^4 \bmod 23 \times 5^2 \bmod 23 \times 5^1 \bmod 23] \bmod 23$

- $5^1 \bmod 23 = 5$
- $5^2 \bmod 23 = 25 \bmod 23 = 2$
- $5^4 \bmod 23 = (5^2)^2 \bmod 23 = (5^2 \bmod 23)^2 \bmod 23$
  $= 2^2 \bmod 23 = 4 \bmod 23 = 4$
- $5^8 \bmod 23 = (5^4)^2 \bmod 23 = (5^4 \bmod 23)^2 \bmod 23$
  $= 4^2 \bmod 23 = 16 \bmod 23 = 16$

$5^{15} \bmod 23 = [16 \times 4 \times 2 \times 5] \bmod 23 = 640 \bmod 23 = 19$

Algoritmo di Diffie-Hellman - 10

Esempio 3



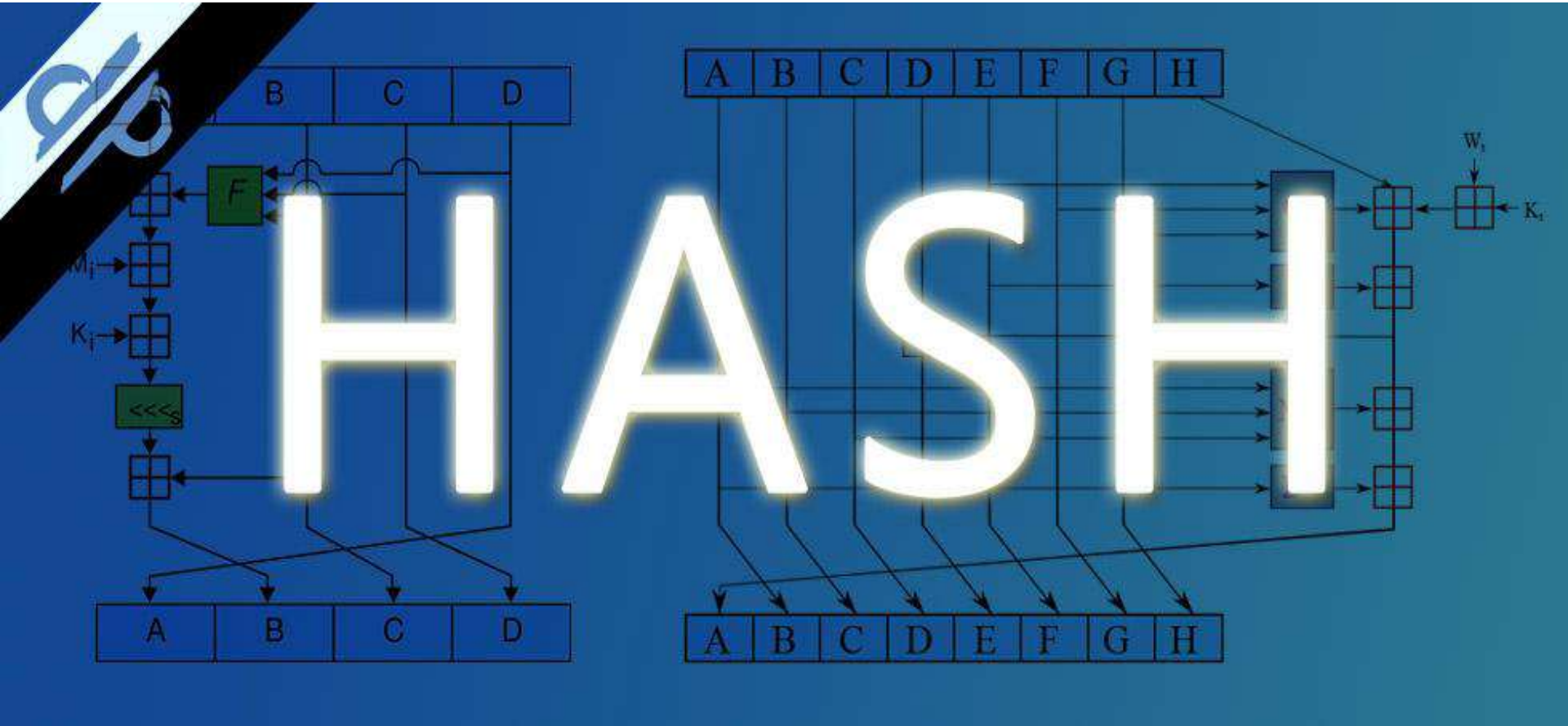
Algoritmo di Diffie-Hellman - 11

Nota: La **pre-master key** potrebbe anche essere resa **pubblica**, senza creare problemi alla crittografia.

A questo punto però l'algoritmo diventa ***anonimo*** (**non autenticato**): se uno si mette in mezzo (**men in the middle**) e mi fa credere di essere il reale destinatario, io posso comunicare con lui credendo che sia il reale destinatario.

Nel caso invece di una **pre-master key** **riservata** e **nota a priori**, **solo chi conosce** la **pre-master key** può attivare una **sessione protetta**. Questa **tecnica è utilizzata** ad esempio nel **WiFi** dal **protocollo** **WPA-PSK** (Wi-Fi Protected Access - Pre Shared Key) , dove la **Pre Shared Key** viene utilizzata sia per l'**autenticazione** sia come **pre-master key** per generare la **chiave** di **sessione** con **Diffie-Hellman**.

Funzioni di HASH



Funzioni Hash: caratteristiche

- Una **funzione** di **Hash** (o **Message Digest**) è una funzione **H** che, dato un **input M** di dimensione qualsiasi, produce un output **Y** (**Hash**) di dimensione fissa (in genere 128-160-256-512 bit): **Y = H(M)**.
- L'idea alla base è che il valore hash **H(M)** sia una **rappresentazione non ambigua** e **non falsificabile** del messaggio **M**.
- Dato un input **X** deve essere **altamente improbabile** che un altro input **Z ≠ X** generi lo stesso hash (**Collisioni**) cioè che:

  H(Z) = H(X).
- **One-Way** (**irreversibili**): cioè deve essere computazionalmente impossibile, noto **H(M)**, ricalcolare **M**.
- Inoltre la funzione deve essere **computazionalmente** semplice da eseguire, cioè calcolare **Y** come **H(M)**.

Funzioni Hash: collisioni

Le **funzioni Hash** si **classificano** in base a due proprietà che rappresentano la **resistenza alle collisioni**:

- **Resistenza Debole alle Collisioni**: Dato M, è computazionalmente impraticabile trovare un altro M´, diverso da M, tale che **H(M) = H(M´)**.

- **Resistenza Forte alle Collisioni**: Computazionalmente impraticabile trovare una coppia (M, M'), con M' ≠ M, tale che **H(M) = H(M')**.

Dalla **Resistenza alle Collisioni** dipende la **sicurezza** di una funzione Hash!

Funzioni Hash: attacchi

Le **due** proprietà sull'**assenza** di **collisioni** (**debole** e **forte**), corrispondono **due diversi tipi di attacchi** :

- **Attacco a forza bruta** (**Brute Force**):

  Trovare un **messaggio** che produca un **dato hash** (cioè un **hash** uguale a quello di un **messaggio dato**).

- **Attacco del compleanno** (**Birthday Attack**):

  Trovare **due messaggi diversi** che producano lo **stesso hash**, **indipendentemente** dal valore di questo **hash**.

MD5

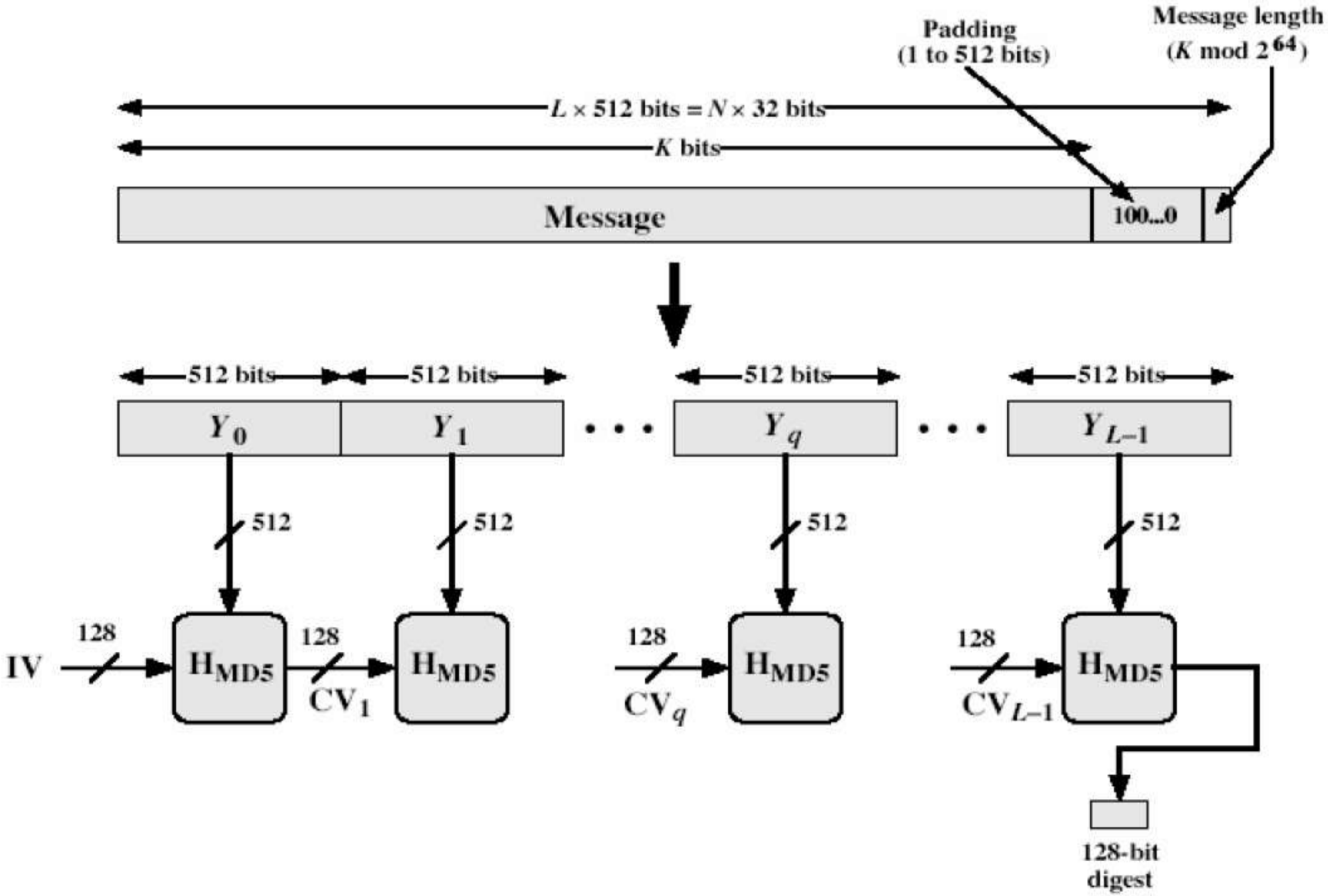
- L’algoritmo di sintesi **MD5** (**Message Digest - Rivest** 1992) è stato uno dei più usati algoritmi hash fino a qualche anno fa.
- L’algoritmo produce una **stringa di 128 bit** (32 caratteri esadecimali) a partire da un **messaggio** di lunghezza qualsiasi, con un’elaborazione piuttosto complicata in modo che ogni bit in uscita dipenda da ogni bit in ingresso.
- Il messaggio da **sintetizzare**, visto come una stringa di bit, viene diviso in **blocchi di 512 bit**; l’ultimo blocco deve essere completato con bit aggiuntivi e contiene negli ultimi 64 bit la **lunghezza originale del messaggio**.
- Viene usato un buffer di 128 bit, inizializzato a un valore **prefissato**; l’algoritmo mescola completamente ogni blocco di 512 bit con il buffer di 128 bit, attraverso un procedimento in **4 fasi** (di **16 passi ciascuna**); al termine il buffer contiene la **sintesi** del **messaggio** (**digest**).

Algoritmo MD5 - 1

- La lunghezza del messaggio è resa pari a 448 mod 512 aggiungendo un bit 1 e poi tanti 0 (**padding**).
- Viene poi aggiunta una stringa di 64 bit, contenente la **lunghezza** del **messaggio originale** (si ottiene così una stringa multipla di 512).
- Viene inizializzato il **buffer** con 4 variabili da 32 bit l'una (8 cifre esadecimali) che conterranno poi l'uscita.
- I **valori** di **inizializzazione** del buffer sono:

  A=67452301, B=EFCDAB89, C=98BADCFE, D=10325476
- Elabora il messaggio in blocchi da **512 bit**.
- Utilizza **4 fasi** (di **16 passi** ciascuna), in cui sono svolte operazioni **bitwise** sul **blocco** di **messaggio** e sul **buffer**.
- Il **codice hash** è il **contenuto** del **buffer** alla **fine** dell'**algoritmo**.

Algoritmo MD5 - 2



Applicazioni Pratiche MD5 - 1

La **crittografia** tramite algoritmo **MD5** viene applicata in tutti i settori dell'informatica che lavorano e che trattano dati sensibili.

Ad esempio, viene utilizzata per controllare che uno **scambio** di dati sia avvenuto **senza perdite** (**integrità**) , semplicemente attraverso il **confronto** del **digest prodotto** dal **file inviato** con quello prodotto dal **file ricevuto**.

È diffuso anche come supporto per l'**autenticazione** degli utenti attraverso i linguaggi di scripting web server-side (**PHP** in particolare): durante la registrazione di un utente su un portale internet, la **password** scelta durante il processo verrà **codificata** tramite **MD5** e **memorizzata** nel **database**.

Successivamente, durante il **login** la **password** immessa dall'utente subirà lo **stesso trattamento** e verrà confrontata con la copia in possesso del server per avere la certezza dell'**autenticità** del login.

```
<?php
    $my_var = 'Questa è la stringa che voglio codificare';  //Assegno ad una variabile la stringa che voglio codificare con l'algoritmo MD5
    echo md5($my_var);  //Stampo a video la codifica MD5 della stringa memorizzata nella variabile
?>
```

In C#

Classe MD5 contenuta nel namespace **System.Security.Cryptography**

```
MD5 md5Hash = MD5.Create("TextToHash")
```

Applicazioni Pratiche MD5 - 2

Molte volte la password viene inserita nel database con opportuna **cifratura simmetrica** (per **poterla decifrare**) ed il controllo con la codifica **MD5** viene eseguito **concatenando** la **password** con un **token** che il **server web** invia all'utente che si vuole autenticare.

Il **client** codifica con **MD5** il **token concatenato** con la **password** e lo invia al server (in modo che la password **non viaggi** in **chiaro** e **non** viaggi sempre la **stessa password codificata**) ed il server effettua lo stesso controllo all'atto della ricezione.

SHA (Secure Hash Algorithm)

Sono stati **sviluppati** dalla **National Security Agency** (**NSA**) e **pubblicati** dal **National Institute of Standards and Technology** (**NIST**).

Gli algoritmi nascono come modifiche del **MD5** e sono suddivisi in quattro categorie:

- **SHA-0** fu sviluppato dal NIST e dall' NSA nel 1993 (obsoleto).
- Fu poi rivisto nel 1995 e chiamato **SHA-1**. Entrambi producono codici hash di **160 bit**.
- Usano come buffer 5 variabili di 4 byte ciascuna (32x5=160) , inizializzate a valori prefissati.
- È stato l'algoritmo hash maggiormente preferito.
- Successivamente nacque la nuova versione, la **SHA-2**, che fu suddivisa in diverse famiglie a seconda della **lunghezza** in bit del **codice hash**: SHA-256, SHA-384, SHA-512. La struttura di tali algoritmi è simile a quella di **SHA-1**.
- **SHA-3** è l'ultimo membro della famiglia di standard, rilasciato dal **NIST** il 5 agosto 2015 (SHA3-224, SHA3-256, SHA3-384 e SHA3-512).

Algoritmo completamente diverso dai predecessori.

Sono usati in **SSL/TLS**.

In C# esistono classi SHA1, SHA256, SHA512.

HMAC
(Hash Message Authentication Code)

É una modalità per l'**autenticazione di messaggi** (**message authentication code**) basata su una funzione di hash, utilizzata in diverse applicazioni legate alla **sicurezza informatica**.

Tramite **HMAC** è infatti possibile garantire sia l'**integrità**, sia l'**autenticità** di un **messaggio**.

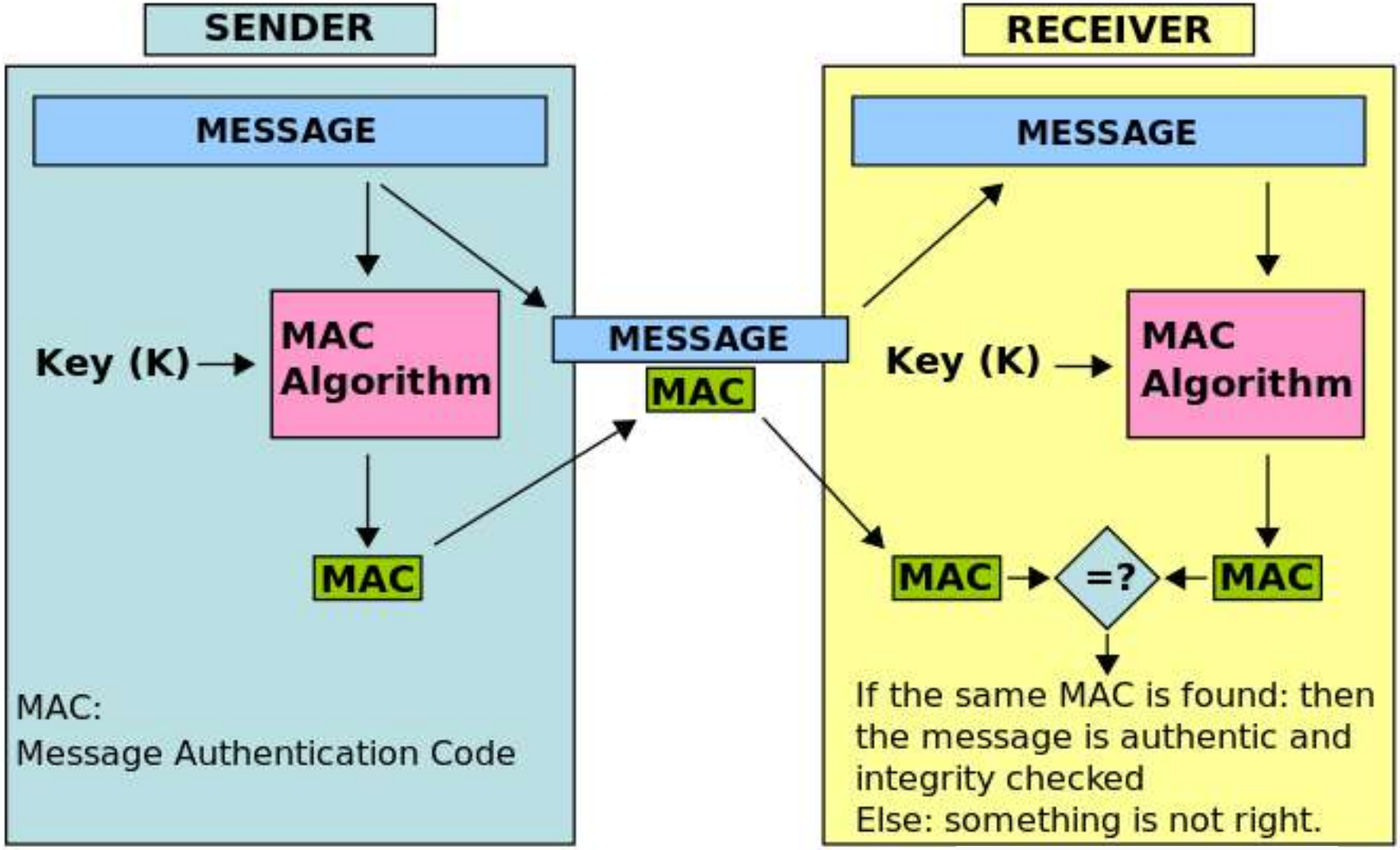
HMAC utilizza infatti una **combinazione** del **messaggio originale** e una **chiave segreta** per la **generazione** del **codice hash**.

Una caratteristica peculiare di **HMAC** è il non essere legato a nessuna funzione di **Hash** in particolare, questo per rendere possibile una sostituzione della funzione nel caso non fosse abbastanza sicura.

Nonostante ciò le funzioni più utilizzate sono **MD5** e **SHA-1**, entrambe attualmente considerate poco sicure.

È **obbligatorio** per la **sicurezza IP** (**IPsec**) usata nelle **VPN** e viene usato in **protocolli internet** come **SSL/TLS.** In questi protocolli ogni messaggio è firmato con un cosiddetto codice di autenticazione dei messaggi , o in breve **MAC**. Se mittente e destinatario sono d'accordo su una **chiave** e un codice di **hashing**, il destinatario può verificare che il messaggio provenga dal mittente e che lo stesso messaggio non sia stato modificato.

HMAC



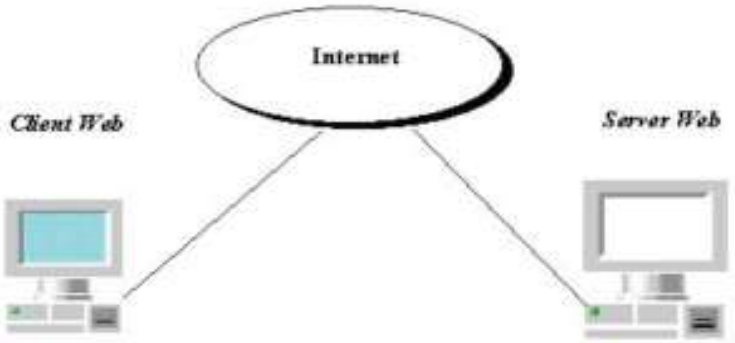
qualcosa non quadra!

Un utente malintenzionato può modificare il messaggio ma **non** conosce la **chiave**. **Non può calcolare** il MAC corretto ed il destinatario saprà che il messaggio non è autentico.

Comunicazioni sicure su Internet: HTTPS e SSL/TLS

HTTP - 1

- Come funziona nel dettaglio il Web?



- Uniform Resource Locator (**URL**) (Indirizzo simbolico pagina)
- Hyper Text Transfer Protocol (**HTTP**)
- Hyper Text Markup Language (**HTML**)

HTTPS (HTTP Secure) - 2

Hyper Text Transfer Protocol (**HTTP**) -------> **HTTPS**

HTTPS è invece *Hyper Text Transfer Protocol over Secure Socket Layer /Transport Layer Security*.

Utilizza il **protocollo SSL/TLS** (*Secure Socket Layer /Transport Layer Security* - letteralmente *livello di socket sicuro/ sicurezza del livello di trasporto*) tra il livello di Trasporto e di Sessione.

L' **HTTPS** è:

- Sintatticamente identico allo schema **http://** ma con la **differenza** che gli accessi vengono effettuati sulla porta **443** (e non sulla **80**) - **https://**
- Tra il protocollo **TCP** e **HTTP** si interpone un livello di **crittografia/autenticazione**.

HTTPS - 3

- **HTTPS verifica l'identità di un sito web e crittografa** le informazioni inviate tra il sito web e il client e viceversa. Queste includono i cookie, i percorsi degli URL e i dati inviati tramite i moduli (form). **HTTPS** è progettato per impedire che queste informazioni vengano lette o modificate mentre sono in transito.

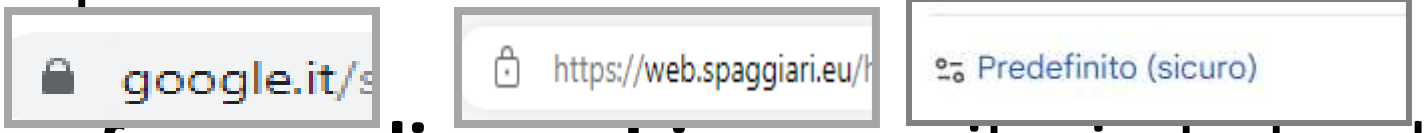
- Le **URL** di **HTTPS** iniziano con **https://** 

e utilizzano la porta **443** di default, a differenza di **HTTP** le cui URL cominciano con **http://** e utilizzano la porta **80**.

HTTPS - 4

Come **riconoscere** una connessione **HTTPS**.

Per **incentivare** la **migrazione** verso **siti sicuri**, tutti i browser visualizzano oramai lo **status** dei website:

- **Un'icona a forma di lucchetto chiuso** di colore **grigio** o **verde** significa che il **sito è sicuro** ed **ha un certificato attendibile** (in Microsoft Edge o Crhome il **lucchetto grigio** indica: sito web **crittografato** e **verificato**; il **lucchetto verde** indica un certificato di **convalida più esteso** (**EV**, **Extended Validation**) che richiede un processo di verifica dell'identità molto rigoroso); 

- **Un'icona a forma di cerchio** con il simbolo di **Informazioni** o **triangolo con !** significa **sito non sicuro**; 

SSL/TLS *Secure Socket Layer /Transport Layer Security* - 5

Si tratta di un **insieme di protocolli** crittografici che aggiungono funzionalità di **cifratura e autenticazione a protocolli preesistenti**.

SSL: versione iniziale sviluppata da **Netscape** nel **1995** , poi evoluta nel protocollo standardizzato **TLS** nel **1999**.

Protocolli SSL e TLS

Protocollo	Pubblicato	Status
SSL 1.0	Non pubblicato	Non pubblicato
SSL 2.0	1995	Deprecato nel 2011 (RFC 6176)
SSL 3.0	1996	Deprecato nel 2015 (RFC 7568)
TLS 1.0	1999	Deprecazione pianificata per il 2020[1]
TLS 1.1	2006	Deprecazione pianificata per il 2020[1]
TLS 1.2	2008	
TLS 1.3	2018	

Il protocollo **SSL/TLS** provvede alla sicurezza del collegamento garantendo:

- **Autenticazione**: sicurezza dell'identità dei soggetti che comunicano (**crittografia asimmetrica**, ovvero a **chiave pubblica e privata** (ex. RSA).
- **Riservatezza-Confidenzialità**: protezione dei dati da osservatori non autorizzati (la crittografia è usata dopo un **handshake** (**accordo**) iniziale per definire una **chiave simmetrica segreta di sessione** (**Diffie-Helmann**). In seguito, per **crittografare i dati** è usata la **crittografia simmetrica** (**3DES**, **AES**, **IDEA** ecc.)).
- **Integrità**: sicurezza che il dato ricevuto è uguale al dato inviato (il livello di Trasporto include un **controllo dell'integrità del messaggio** basato su un apposito **MAC** (**Message Authentication Code**) che utilizza funzioni **hash** sicure (**MD5** Message Digest, **SHA** Secure Hash Algorithm). In tal modo si verifica che i dati spediti tra client e server non siano stati **alterati** durante la trasmissione.

Transport Layer Security - 6

Funzionalità di TLS

- **Cifrare la connessione** al fine di garantire la **riservatezza** dei dati.
- **Autenticare** il **server** a cui si è connessi.

Funzionamento di TLS

- Il **server** invia al **client** un **certificato** attestante la **propria identità** e contenente la **propria chiave pubblica** (ad es. una chiave **RSA**).
- Il **client verifica l'identità** del **server** tramite **Certification Authority**. Poi genera una **chiave di sessione casuale** (o **Diffie-Hellman**) e la invia al server, **cifrandola** con la **chiave pubblica** del server.
- Il server **decifra** il messaggio con la propria **chiave privata** ed ottiene la **chiave di sessione**. Questa viene utilizzata per cifrare il successivo traffico dati con un **algoritmo simmetrico** (ad es. **3DES** o **AES** o **IDEA**).
- La **riservatezza** è quindi garantita dall'utilizzo **ibrido** di **cifratura simmetrica** e **asimmetrica**.
- L'**autenticazione** del server è **garantita** dal **certificato** rilasciato dalla **CA**.

Autorità di Certificazione -7

Lo **scambio** di **chiavi pubbliche** introduce un nuovo problema da risolvere.

Se ad esempio vogliamo comunicare in modo sicuro con Google, abbiamo bisogno della sua chiave pubblica e lo stesso vale per altri siti che vogliamo visitare (Facebook, Twitter, etc...).

Dato che esistono miliardi di siti web su Internet, come possiamo ottenere una **chiave pubblica** per ogni sito web che vogliamo visitare?

Ed è qui che entrano in scena le **Autorità di Certificazione** o **CA** – **Certification Authorities** come **DigiCert**, **Comodo, Symantec**, **GlobalSign**, **Google Trust Services**, **VeriSign**, **GeoTrust, Actalis**.

Lo scopo del **certificato digitale** è quello di garantire che una **chiave pubblica** sia **associata** alla vera **identità del soggetto** che la **rivendica** come **propria**.

Una **CA** è un'**organizzazione** di terze parti con 3 obiettivi principali:

- **Rilasciare certificati digitali.**
- **Confermare l'identità del proprietario del certificato.**
- **Fornire la prova che il certificato è valido.**

Nello scenario di **HTTPS** la **chiave pubblica** è rappresentata da un **certificato digitale**, più noto come **certificato SSL/TLS**. Un **certificato digitale** è un **certificato di identità elettronico,** cioè un **documento elettronico** **che associa l'identità di una persona ad una chiave pubblica**.

Certificato SSL/TLS - 8

Ora vediamo in che modo è possibile **ottenere un certificato SSL/TLS firmato da una Certification Authority**:

1. Il proprietario del sito web genera una **chiave pubblica** e una **chiave privata** (ex. **RSA**) ed invia un file di **richiesta di firma** del certificato (**CSR** Certificate Signing Request) e la sua **chiave pubblica** alla **CA**.
2. La **CA** crea quindi un **certificato personale basato sulla CSR**, dove sono indicati nome di dominio, nome del proprietario, data di scadenza e altre informazioni, appone la propria **firma digitale**, infine **crittografa** l'intero certificato con **la chiave pubblica** del sito e lo rimanda al proprietario del sito web.
3. Il **certificato** viene quindi **decifrato con la chiave privata** del proprietario del **sito web** ed, infine, **viene installato sul server**.

Il certificato è firmato digitalmente dall'autorità certificante (certification authority, CA) che quindi si fa garante dell'autenticità delle informazioni contenute nel certificato.

- **N.B.** La **firma digitale della CA sul certificato** è **crittografata** dalla **chiave privata della CA** e **può essere decifrata solamente** con la **chiave pubblica** della **CA**, chiamiamo questa **chiave Certificato Radice** (**Root Certificate**).
- Ogni dispositivo (pc, smartphone) ha, installato nel browser, un elenco di **certificati radice** delle **CA radice attendibili** e delle **CA intermedie**.

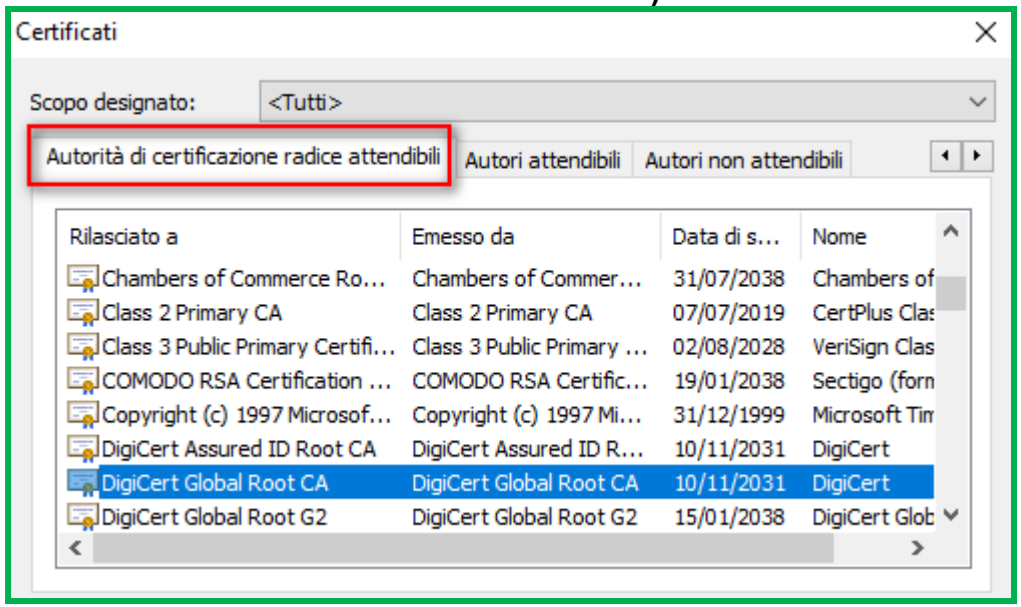
I Certificati SSL/TLS - 9

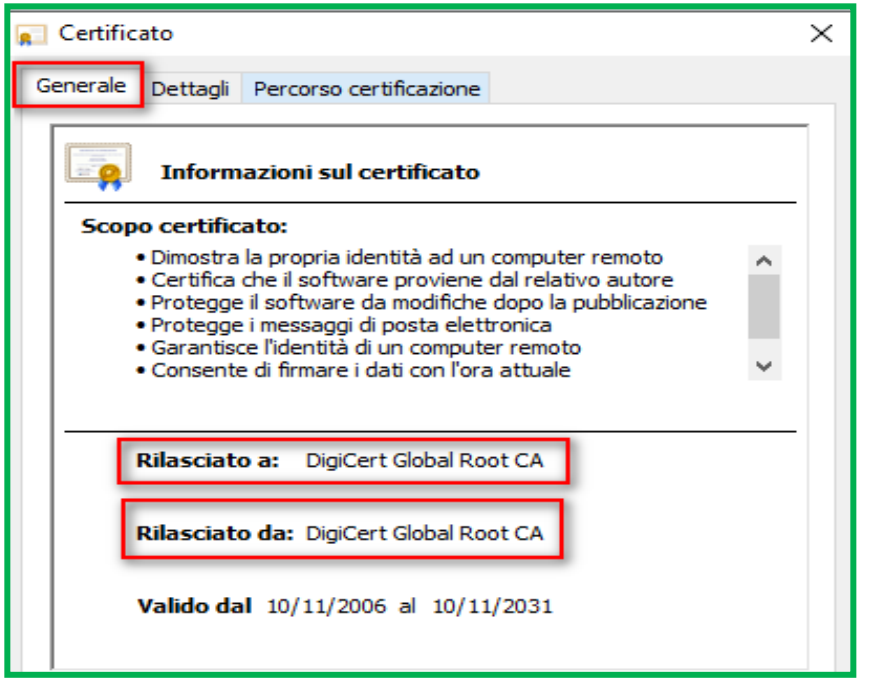
Il **Certificato Digitale** viene tipicamente distribuito mediante file **.DER** e contiene le seguenti **informazioni**:

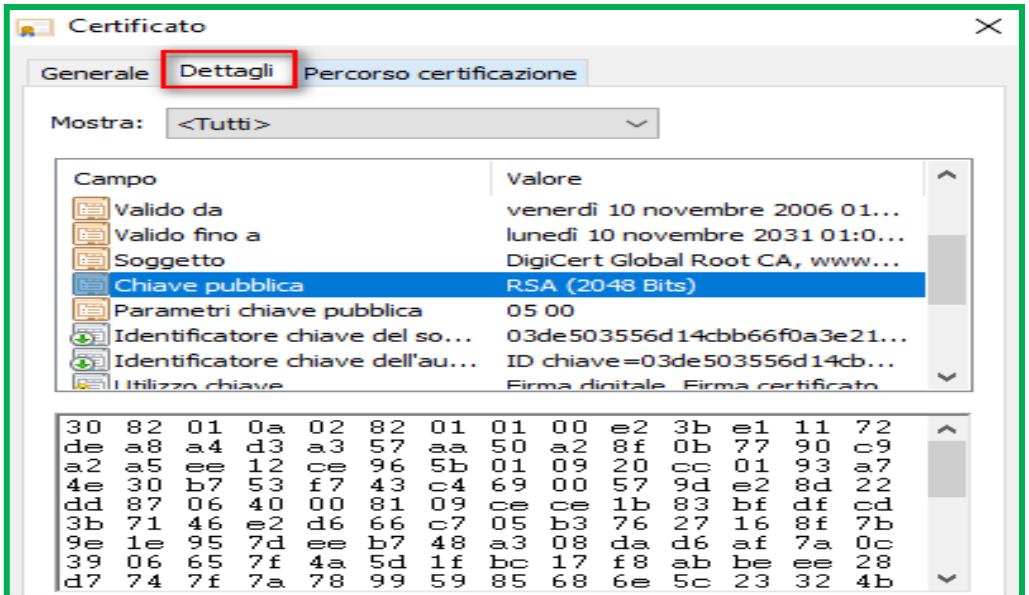
- **nome del proprietario** a cui è stato rilasciato il **Certificato**.
- **il nome del dominio** associato al **Certificato**.
- **chiave pubblica** (2048 bit) del **Certificato** (visualizzabile in forma esadecimale).
- **nome della CA** che ha firmato il **Certificato Digitale**
- **numero di serie** del Certificato.
- **validità** del Certificato (data di inizio e data di scadenza)
- la **firma digitale** della **CA** (**CERTIFICATE SIGNATURE**) e l'algoritmo utilizzato per la firma (**MD5** o **SHA**): la **CA** si fa garante dell'**autenticità** delle **informazioni** contenute nel **certificato**.

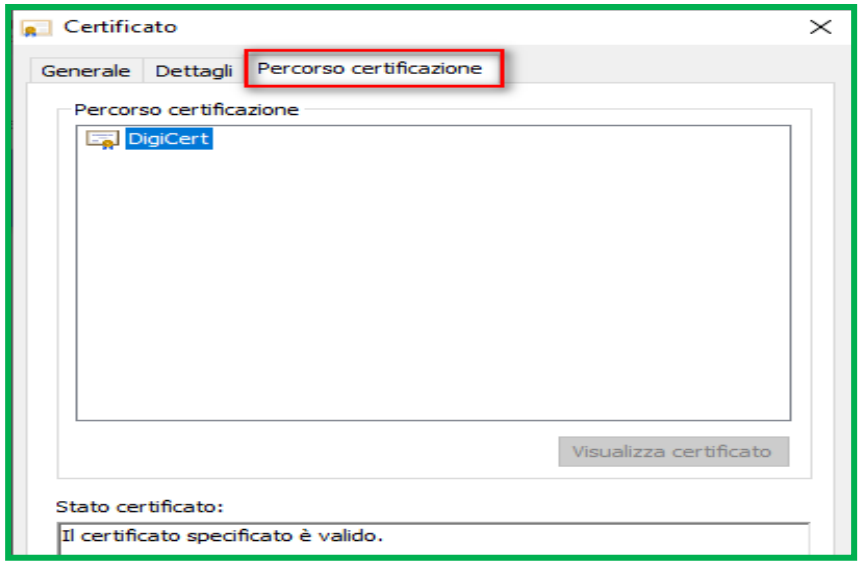
Elenco Certificati Radice - 10

Elenco di **certificati radice** sul browser **google chrome** (Impostazioni -> Privacy e sicurezza -> Sicurezza -> Gestisci certificati):



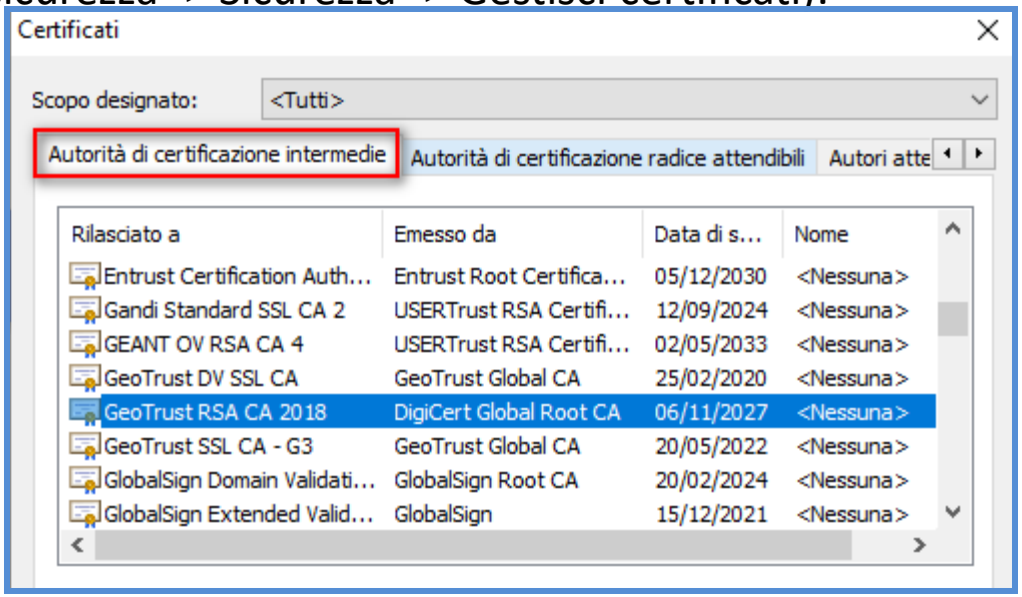


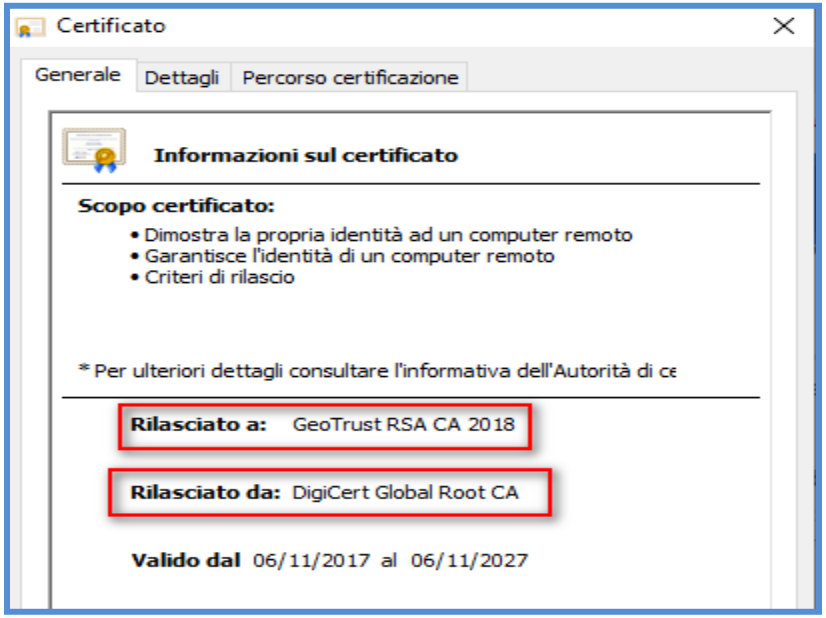


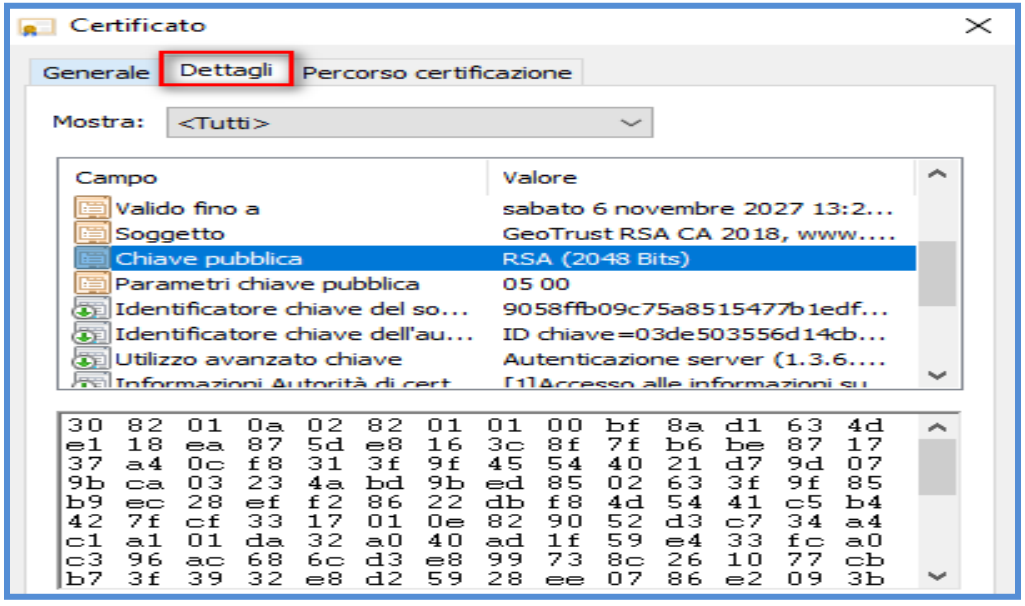


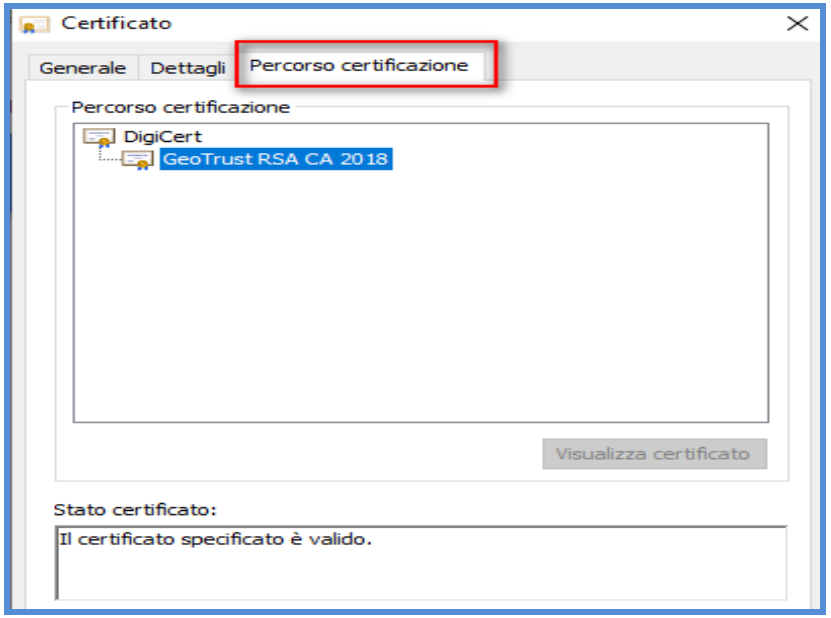
Elenco Certificazioni Intermedie - 11

Elenco di **certificati di Autorità Intermedie** sul browser **google chrome** (Impostazioni -> Privacy e sicurezza -> Sicurezza -> Gestisci certificati):









Gerarchia di Certificati - 12

Ma come **fidarsi** della **Certification Authority**?

- L'identità di ogni **CA** è **garantita** a sua volta da un certificato rilasciato da una **CA** "**superiore**"…
- Fino ad arrivare alla **root authority**, un'autorità certificante **non certificata da nessuno**, ovvero **certificata** da **se stessa**.
- Se la **root authority** è nota a livello mondiale e sottoposta a verifiche annuali da appositi organismi di controllo, possiamo ragionevolmente **fidarci** dei suoi **certificati**.

Root Authorities - 13

• Per questo motivo le **root authorities** fidate sono poche (meno di 40 in tutto il mondo), con **VeriSign** che detiene attualmente più del **50%** del mercato.

• I certificati delle root authorities sono "**self-signed**" in quanto non possono essere firmati da nessun'altra **CA**.

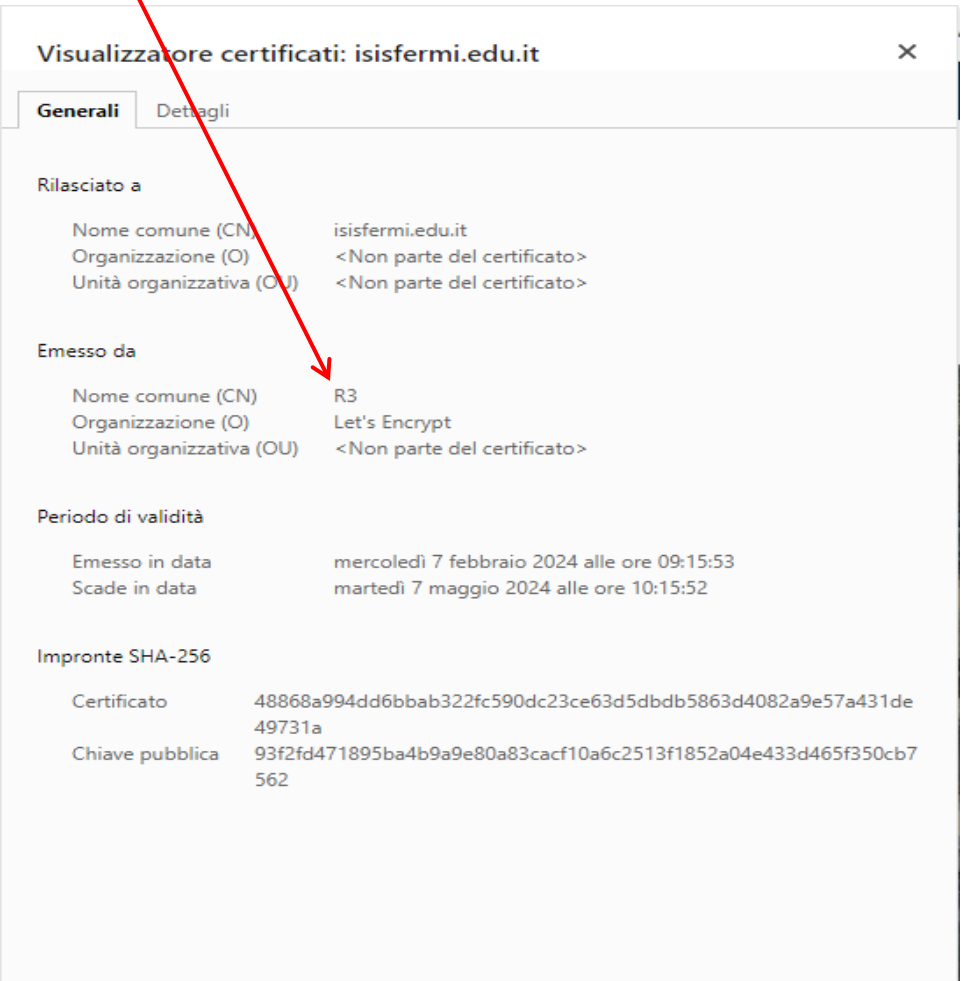
• Per garantire il processo di **verifica dei certificati**, **tutti i browser**, **client di posta elettronica**, e in generale ogni software che faccia uso di connessioni **TLS**, **sono distribuiti assieme** ai **certificati delle root authorities** (**Root Certificate - Certificati Radice**).

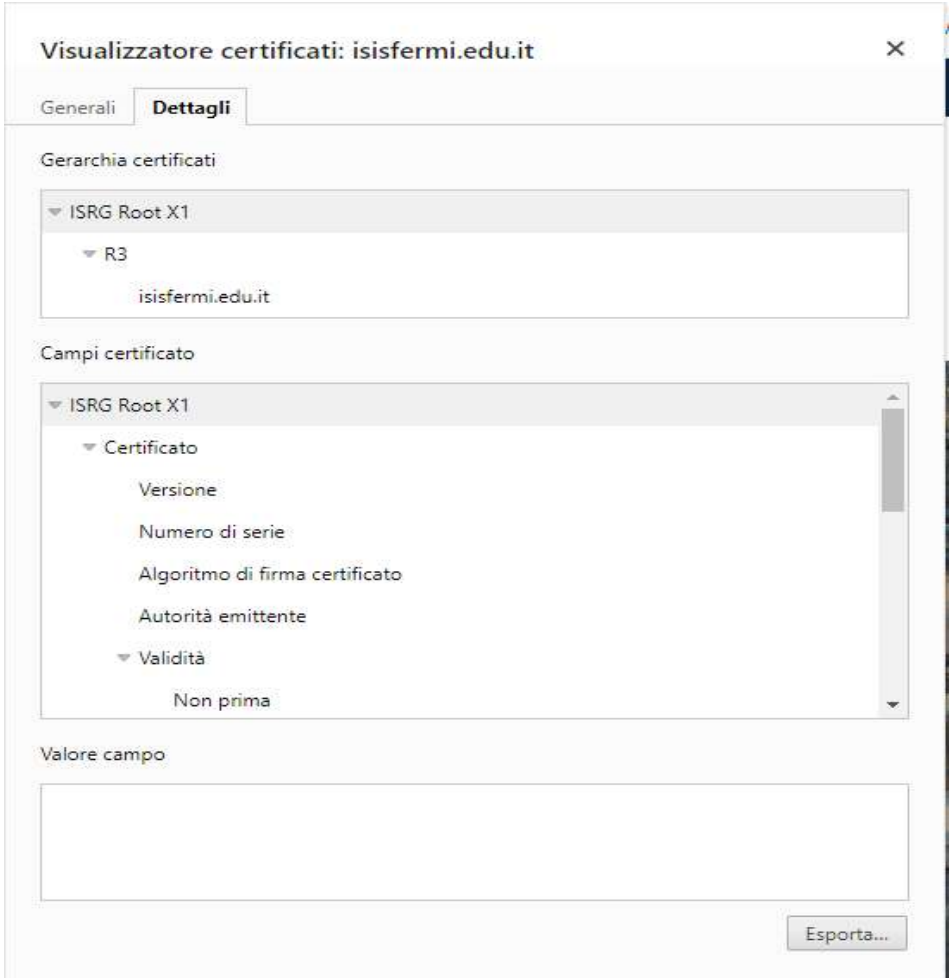
Connessione HTTPS tra Browser e Server - 14

Ora vediamo come avviene la **connessione HTTPS** tra client browser e sito web, prendendo come esempio il sito https://www.isisfermi.edu.it/

e **R3** (**Let's Encrypt**) come **CA - Certification Authority**.

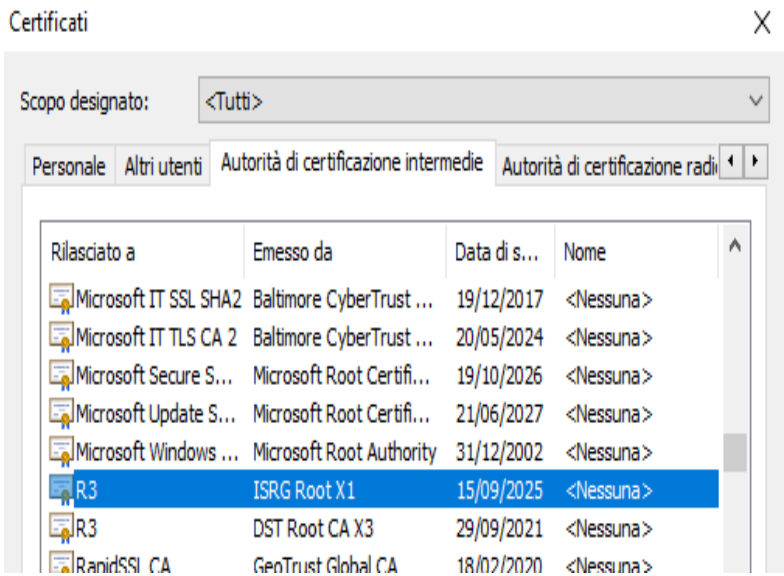
Cliccando sul simbolo  davanti alla url e poi scegliendo **Certificato**:





Connessione HTTPS tra Browser e Server - 15

ISRG Root X1 è la **root authority** https://letsencrypt.org **Let's Encrypt** è un'**autorità di certificazione** (**CA**) libera, automatizzata e gratuita, gestita per il bene pubblico. Si tratta di un servizio fornito dal Gruppo di Ricerca della Sicurezza di Internet (ISRG).
Fornisce i certificati digitali necessari per abilitare **HTTPS** (**SSL/TLS**) per i siti web, gratis, nel modo più semplice possibile per l'utente.



Durante una connessione **HTTPS** avvengono **due passaggi**:

- L'**handshake** (letteralmente **stretta di mano**) per **convalidare** il **certificato** del **sito web**.
- La **creazione di una connessione sicura** tra **client** e **sito web**.

Handshake - 16

Ecco cosa succede durante la **stretta di mano** tra **browser** e **sito web**:

- Il **Client** invia al **Server HTTPs** un **ClientHello message** contenente la versione di **TLS/SSL** impostata sul client più la cosiddetta **cipher suite**, (insieme di codici), cioè un elenco dei protocolli di **Autenticazione** (tipo **RSA**), **Crittografia** (tipo **3DES**, **AES**, **IDEA**) con relativa **profondità di chiave** e **Funzioni HASH** (tipo **MD5** e **SHA**) **supportati** dal **browser** utilizzato per la richiesta.
- Il **Server** risponde inviando in chiaro una **lista di Protocolli Scelti** (fra quelli supportati dal browser), la sua **chiave pubblica** + il **certificato digitale di autenticazione** (firmato da **R3** con la sua chiave privata).
- Per **verificare l'autenticità del certificato**, il **browser** preleva dall'elenco dei **certificati radice o intermedi** la **chiave pubblica** di **R3** e tenta di **decodificare la firma digitale del certificato** che è stata **crittografata tramite la chiave privata** di **R3**.
- Se è in grado di **decifrare** la **firma digitale** del **certificato** passa allo step successivo di creazione di una **connessione sicura col server**, altrimenti mostra all'utente un avviso di **Certificato non Valido** come si può vedere dalla figura seguente.

Avviso Fallimento Handshake - 17

La connessione non è privata

Gli utenti malintenzionati potrebbero provare a carpire le tue informazioni da **untrusted-root.badssl.com** (ad esempio, password, messaggi o carte di credito). Ulteriori informazioni

NET::ERR_CERT_AUTHORITY_INVALID

☐ Invia automaticamente a Google alcune informazioni sul sistema e alcuni contenuti delle pagine per contribuire a rilevare app e siti pericolosi. Norme sulla privacy

AVANZATE

Torna nell'area protetta

Connessione Sicura a Chiave Simmetrica - 18

Come già detto, con la richiesta della pagina di **www.isisfermi.edu.it** , il server di **Isisfermi** invia la sua **chiave pubblica** al browser.

Tutti i dati cifrati con questa **chiave pubblica** possono essere decifrati solo dalla **chiave privata** di **Isisfermi**.

- Dopo aver **convalidato** il **certificato**, il browser crea una nuova **chiave simmetrica di sessione** (o una **pre-shared key Diffie-Hellman**) facendone una copia. Queste chiavi saranno usate per crittografare e decrittografare i dati.
- Il browser quindi **crittografa** la **chiave simmetrica di sessione** con la **chiave pubblica di Isisfermi** ed invia tutto al server di **Isisfermi** .
- Il server di **Isisfermi** decodifica la **chiave simmetrica di sessione** con la sua **chiave privata**.
- Ora server e browser hanno entrambi una copia della **chiave simmetrica di sessione** creata dal browser. Nessun altro ha questa chiave, quindi solo server e browser **possono cifrare** e **decifrare** i dati in **maniera sicura**.
- Quando **Isisfermi** invia dati al browser, prima li cifra con la chiave di **simmetrica di sessione** e il browser decodifica i dati con la sua copia di questa chiave. Lo stesso avviene se è il browser ad inviare i dati.
- **Client** e **Server** trasmettono i dati codificandoli con la **chiave simmetrica** concordata. In coda ad ogni messaggio l'**End Point autenticato** (il **server**, oppure **entrambi** in caso di **autenticazione bilaterale**), aggiunge la propria **Firma Digitale** che consiste in una **impronta SHA256 univoca** cifrata con la **chiave privata**. Il ricevente, utilizzando il **Certificato Digitale** del mittente, **verifica** la **Firma Digitale** che garantisce sia **l'integrità** del **messaggio** sia **l'autenticità** del **mittente**.

Funzionamento HTTPS - 19

Riepilogo passo dopo passo.

1. **Isisfermi** richiede un **certificato** a **R3**.
2. **R3** verifica che è davvero **Isisfermi** che sta effettuando la richiesta.
3. **Isisfermi** invia a **R3** la propria **chiave pubblica**.
4. **R3** usa la propria **chiave privata** per firmare digitalmente la **chiave pubblica** di **Isisfermi**.
5. **R3** dà a **Isisfermi** la **chiave pubblica** firmata, questa rappresenta ora il **Certificato SSL/TLS** di **Isisfermi**.
6. **Lo studente** si collega col suo browser al sito web di **Isisfermi** ed inizia la fase di **handshake** (**protocolli** e **hash supportati**).
7. **Isisfermi** invia al browser il **Certificato SSL/TLS**.
8. **Utilizzando** la **chiave pubblica** di **R3** , che si trova nell'archivio dei **certificati radice** del browser, viene verificata la firma di **R3** sul **Certificato SSL/TLS** di **Isisfermi** .

Funzionamento HTTPS - 20

9. **Se tutto ok**, viene generata una **chiave segreta simmetrica di sessione** e si utilizza la **chiave pubblica** di **Isisfermi**, ovvero il **Certificato SSL/TLS**, per **crittografarla**.
10. **Viene** inviata a **Isisfermi** la **chiave segreta simmetrica di sessione cifrata** con la **chiave pubblica** di **Isisfermi** (in **alcuni** casi si utilizza **pre-shared key Diffie-Hellman**).
11. **Isisfermi decifra** la **chiave segreta di sessione** con la propria **chiave privata** e la trattiene.
12. **Il browser** dello studente e **Isisfermi** utilizzano la **chiave segreta simmetrica di sessione** condivisa per **cifrare** le **comunicazioni successive**.
13. **Così** viene raggiunta l'**autenticazione** e la **segretezza** e finché **R3** è ritenuta una **CA attendibile**, lo studente può essere certo che i **certificati di Isisfermi** saranno **sicuri** e **affidabili**.