

# Sicurezza dei sistemi informativi

## Definizioni

Computer Security (NIST) : La protezione offerta ad un sistema informatico al fine di raggiungere gli obiettivi applicabili di preservare l'integrità, la disponibilità e a confidenzialità delle risorse del sistema di informazioni.

Si hanno quindi 5 obiettivi fondamentali da voler ottenere:

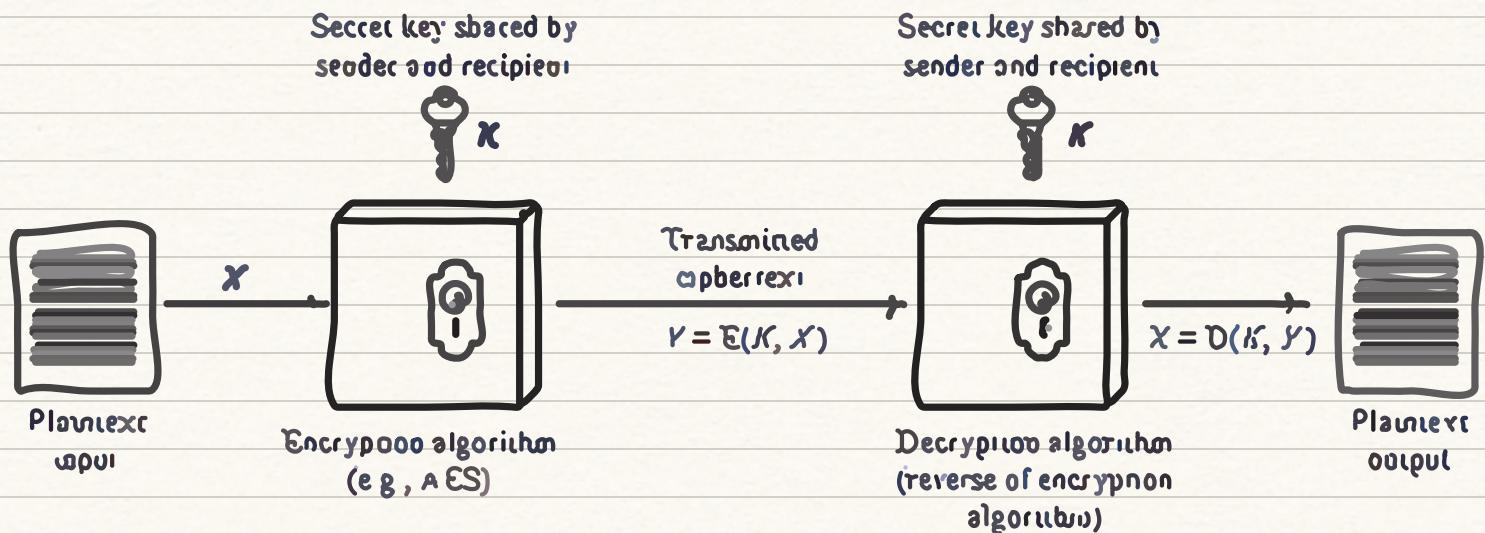
- **Confidenzialità** (Confidentiality): Informazioni accessibili solo agli interessati.
- **Integrità** (Integrity): Protezione da modifiche o distruzione delle informazioni.
- **Disponibilità** (Availability): Accessibilità ai dati in modo tempestivo e affidabile, la perdita di questa proprietà potrebbe comportare un fermo di produzione.
- **Autenticità** (Authenticity): Proprietà che indica che determinate informazioni siano vere (rispetto all'integrità che punta alla protezione da modifiche e distruzione).
- **Responsabilità** (Accountability): Proprietà mirata all'associazione di determinate operazioni alle persone/entità che le mettono in atto.

## Tecniche classiche di cifratura

Quando internet (TCP/IP) è stato progettato non si teneva conto della sicurezza, successivamente a causa della diffusione di questa tecnologia è nata la necessità di dover integrare le proprietà descritte precedentemente, il tutto continuando a mantenere la compatibilità dello stack protocollare TCP/IP.

Un modello di cifratura tra i più diffusi è sicuramente il **Modello di Cifratura Simmetrico**, esso prevede cinque elementi:

- **Testo in chiaro** (Plaintext): Il messaggio originale che si vuole nascondere.
- **Algoritmo di cifratura** (Encryption algorithm): L'algoritmo di crittografia utilizzato per la trasformazione del testo in chiaro in testo cifrato.
- **Testo cifrato** (Ciphertext): Il messaggio cifrato prodotto come output dell'algoritmo di cifratura.
- **Algoritmo di decrittografia** (Decryption algorithm): L'algoritmo utilizzato per trasformare il testo cifrato in testo in chiaro.
- **Chiave segreta** (Chiave segreta): La chiave segreta che viene utilizzata dai due algoritmi per cifrare e decifrare il testo. Le trasformazioni degli algoritmi dipendono strettamente dalla chiave.



La crittografia simmetrica (o a chiave singola) è quella più efficiente e più veloce, oltre al fatto che si utilizza solo una chiave.

Vista la loro efficienza vengono utilizzate dagli sistemi di cifratura moderni insieme agli algoritmi a chiave asimmetrica (o pubblica) per avere migliori prestazioni.

Per utilizzare in modo sicuro questo tipo di crittografia bisogna che:

- Un attaccante non sia in grado di risalire al testo in chiaro o alla chiave attraverso l'analisi crittografica, quindi deve essere difficile anche avendo in possesso di tanti ciphertext differenti.
- Il mittente e il destinatario devono potersi scambiare la chiave segreta in modo sicuro, per poi mantenerla un segreto.

Tipi di attacchi:

- **Approccio ad analisi crittografica** (cryptanalysis), analisi del testo cifrato e dell'algoritmo sulla base di alcune conoscenze del plaintext
- **Approccio a forza bruta** (brute force) si tenta ogni possibile chiave su un frammento di testo, per questo le chiavi devono essere grandi, aumenta il numero di combinazioni possibili.

## Cifrario di Cesare

Una tecnica semplicissima, dato un numero intero come chiave effettua una trasposizione della lettera dell'alfabeto con la k-esima successiva. Per esempio k=3:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12



Quindi **a** diventa **d**, **b** diventa **e**, e così via... Arrivati alla fine dell'alfabeto si ricomincia dall'inizio come un buffer circolare.

Problema di questo cifrario è lo spazio delle chiavi molto ridotto, risulta vulnerabile ad attacchi bruteforce, ma anche ad analisi crittografica.

## Cifratura Monoalfabetica

Invece di utilizzare la tabella dell'alfabeto con il suo ordine, si effettua una permutazione della tabella stessa, la chiave sarà quindi la nuova stringa contenente l'ordine delle lettere dell'alfabeto permutate. Si andrà ad effettuare una sostituzione posizionale tra il testo e la chiave:

**CVIMUDFZPTBAREOLNHGSQ è LA KEY**

Voglio criptare CIAO

ABCDEFGHIJKLMNPQRSTUVWXYZ

CVIMUDFZPTBAREOLNHGSQ

Vediamo che a la lettera C corrisponde la I, ecc...

Ciphertext: IPCR

Rispetto al cifrario di cesare abbiamo uno spazio delle chiavi molto più grande, passiamo da **25** a **26! = 4x10^26**.

Più difficile da attaccare tramite bruteforce ma rimane soggetto ad attacchi ad analisi crittografica, è infatti possibile (avendo abbastanza ciphertext) andare a comparare le distribuzioni delle varie lettere e confrontarle con quelle in chiaro della lingua utilizzata nel testo in chiaro.

La capacità di mascherare queste distribuzioni è detta proprietà di diffusione.

## Cifratura Playfair

Sostituisce al cifrario di cesare la tabella dell'alfabeto aggiungendo una chiave arbitraria all'inizio e completando la tabella con le lettere non utilizzate nella chiave.

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

Il testo in chiaro viene crittografato due lettere alla volta, si usano le seguenti regole:

- Se la coppia ha due lettere uguali le due lettere cifrate devono essere separate da un carattere di riempimento (X ad esempio)
- Due lettere di testo in chiaro che cadono nella stessa riga della matrice devono essere sostituite con l'elemento a destra.
- Due lettere di testo in chiaro che cadono nella stessa colonna devono essere sostituite con l'elemento sottostante.

È più robusto del cifrario di cesare ma continua ad essere soggetta a criptoanalisi.

## Cifratura Hill

Questo algoritmo di cifratura prende in ingresso una sequenza di m lettere in chiaro e le sostituisce con m lettere di testo cifrato.

La sostituzione è determinata da m equazioni lineari in cui, a ciascun carattere in ingresso viene assegnato un peso/parametro numerico. Per  $m = 3$

$$\begin{aligned}c_1 &= (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \text{ mod } 26 \\c_2 &= (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \text{ mod } 26 \\c_3 &= (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \text{ mod } 26\end{aligned}$$

Dove  $kij$  sono i pesi e  $pj$  sono i 3 caratteri in chiaro. La chiave è la matrice K.

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \text{ mod } 26$$

$$C = KP \text{ mod } 26$$

Per decifrare basta calcolare la matrice inversa.

Meno soggetta a bruteforce e aggiunge la proprietà di diffusione, è soggetta ad attacchi di tipo **known plaintext**.

## Cifratura Vigenère

Le cifrature monoalfabetiche hanno la debolezza di non nascondere la frequenza dell'alfabeto originario, la soluzione a questo problema è quello di andare ad effettuare sostituzioni differenti man mano che si procede con il testo in chiaro. Questo approccio è chiamato **Cifratura a Sostituzione Polialfabetica**.

Il cifrario più noto è il **Vigenère Cipher** che ha resistito agli attacchi per oltre 300 anni.

Per comprendere e utilizzare lo schema si costruisce una matrice chiamata Tabella di Vigenère.

		Testo in chiaro																									
		A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z
Chiave:	A	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z
	B	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	
D	D	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	
E	E	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	
F	F	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	
G	G	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	
H	H	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	
J	J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	
J	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	
K	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	
L	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	
M	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	
N	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	
O	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	
P	P	O	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	U	
W	W	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	
X	X	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
Y	Y	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	
Z	Z	A	B	C	D	E	F	G	H	J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	

Utilizzo una parola come chiave: **LEONE**

Ottengo il testo in chiaro da cifrare: **ATTACCHIAMOALLALBA**

Ripeto la chiave in modo che abbia la stessa lunghezza del testo in chiaro da cifrare:

P: **ATTA CCHI AMO ALL ALBA**

K: **LEONE LEONE LEONE LEO**

Per ottenere il testo cifrato si prosegue come segue:

- Per la prima lettera **A** usiamo la chiave **L**:

- selezioniamo la riga della chiave
- selezioniamo la colonna della lettera in chiaro

- Il risultato è che ad **A** corrisponde la lettera cifrata **L**

Il testo cifrato sarà:

P: ATTA CCHI AMO ALL ALBA

K: LEONE LEONE LEONE LEO

C: LXHNGN ...

Per decifrare si prosegue come segue, facciamo una prova sulla seconda lettera:

- Abbiamo a disposizione la lettera cifrata **X** e la chiave **E**
- Seleziono la riga corrispondente alla chiave
- Nella riga selezionata cerco la lettera corrispondente a quella da decifrare
- Trovata la corrispondenza nella riga vado a vedere la colonna in cui si trova e la lettera associata, che sarà la lettera in chiaro.

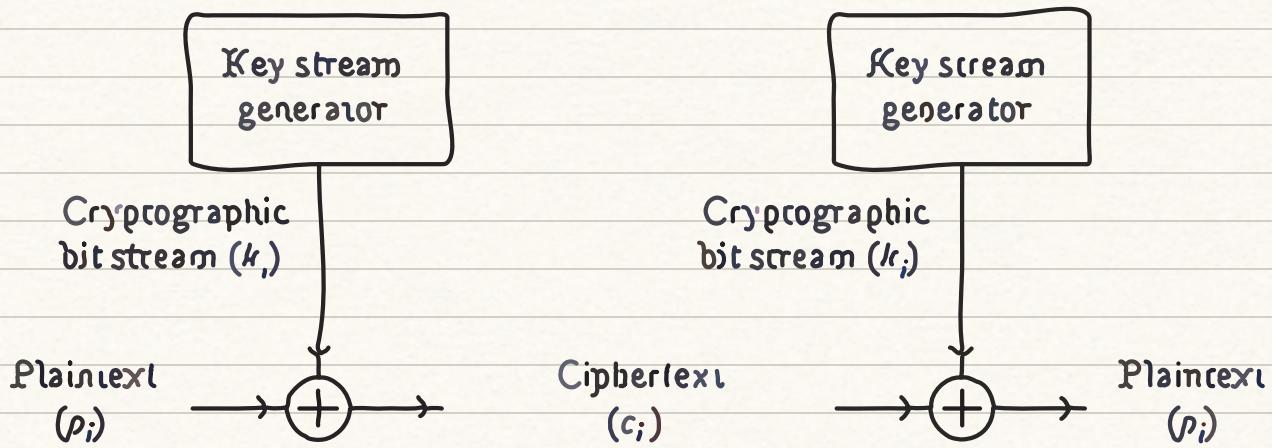
Quindi selezionando la riga **E**, individuando in essa la lettera cifrata **X** e vedendo la lettera corrispettiva di quella colonna si evince che la lettera in chiaro è **T**

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

## Cifratura di Vernam

Questo metodo di cifratura nasce per fronteggiare i rischi dell'analisi crittografica, quindi l'idea è quella di applicare la cifratura al livello di bit invece che al livello di simbolo. Prevede l'utilizzo di una chiave lunga quanto tutto il plaintext e si basa sull'operatore **XOR**:

$$c_i = p_i \oplus k_i \rightarrow \text{cifratura dell}'i\text{-esimo bit}$$
$$p_i = c_i \oplus k_i \rightarrow \text{decifratura dell}'i\text{-esimo bit}$$



## One-Time Pad

L'idea alla base di questa tecnica crittografica prevede l'utilizzo della cifratura di Vernam, si continua ad usare una chiave lunga quanto il plaintext, l'unica differenza sta nel fatto che la chiave deve essere cambiata ogni volta che si va ad effettuare una cifratura.

Risulta evidente il costo di One-Time Pad, non solo chiavi lunghe quanto il testo in chiaro... ma bisogna cambiarla di continuo, ovviamente le chiavi devono essere scambiate in modo sicuro tra le due parti (aspetto per nulla scontato).

## Tecniche di trasposizione

Oltre alle tecniche di sostituzione ci sono le tecniche di trasposizione, invece di sostituire la lettera con un corrispettivo nel dominio dell'alfabeto, vado a effettuare delle permutazioni interne al mio testo in chiaro.

**Rail Fence:** Prevede la scrittura del plaintext su sequenze di diagonali, ma viene poi letto per righe:

m e m a t r h t g p r y  
  ↓ ↓ ↓ ↓  
e t e f e t e o a a t

**Row Transposition Cipher:** La chiave dell'algoritmo è quello di scrivere il plaintext per righe e leggerlo per colonne, la chiave dell'algoritmo sta appunto nell'ordine in cui leggere le colonne.

Key	4 3 1 2 5 6 7
Input	t t n a a p t
	m t s u o a o
	d w c o i x k
	n l y p e t z
Output	NSCYAUOPTTWLIMDNAOIEPAXTOKZ

## Enigma

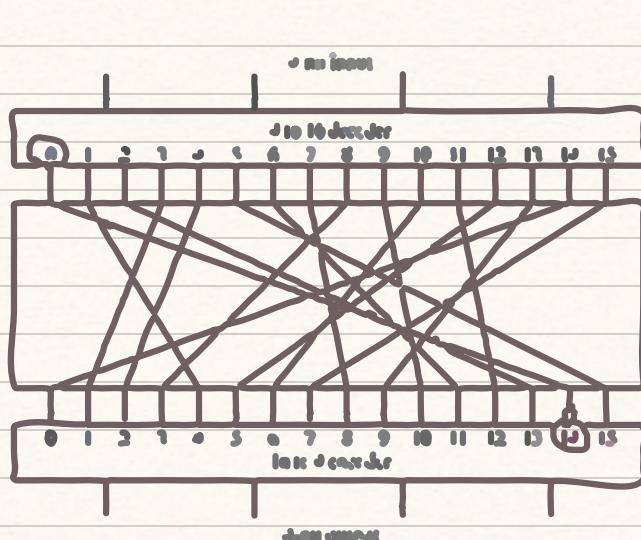
Enigma è una macchina basata su rotori (numero variabile da 3 a 5) utilizzata dall'esercito tedesco.

Ognuno di questi rotori effettuava una sostituzione, quindi si aveva in cascata un numero n di sostituzioni, dopo ogni lettera cifrata questi rotori giravano, ad esempio il primo rotore girava ad ogni step il secondo ogni 5 ed il terzo ogni 20. Queste rotazioni andavano a definire ogni volta nuove possibili sostituzioni. Utilizzando 3 rotori si avevano 17576 possibili alfabeti.

Ovviamente la chiave era rappresentata dai rotori utilizzati, dal loro ordine di applicazione e dal loro indice iniziale (posizione iniziale del singolo rotore). Fu attaccato tramite un attacco del tipo known plaintext, (**il saluto al fuhrer**).

## Block Ciphers

Le cifrature a blocchi elaborano il messaggio da criptare dividendolo in blocchi di lunghezza fissa, essi vengono poi elaborati singolarmente. Una cifratura a blocchi ideale prevede che dato un blocco di  $n$  bit, quindi con  $2^n$  possibili combinazioni, per ogni combinazione in chiaro vi sia una corrispondenza univoca di combinazione cifrata (mapping 1 a 1).



Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1001
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

Questa cifratura ha tanti svantaggi:

- Bisogna utilizzare un  $n$  molto grande (altrimenti è paragonabile ad una semplice soluzione).
- La chiave è legata al mapping, spesso è una tecnica usata in hardware e per questo infatti si dispone di tanti chip, ognuna con una chiave/mapping differente.

## Cifratura di Feistel

Sistemi crittografici che combinano tecniche di sostituzione e permutazione sono chiamati sistemi prodotto, questi sono alla base di molti cifrari moderni. Feistel ha proposto proprio un sistema basato sul sistema prodotto, alternando permutazioni e sostituzioni così da annullare e nascondere le relazioni statistiche tra ciphertext e plaintext.

Feistel si basa sui concetti fondamentali di Claude Shannon, ovvero i concetti di Diffusione e Confusione (Diffusion and Confusion).

L'obiettivo di Shannon era quello di andare a proporre dei fondamenti teorici che permettessero di realizzare dei buoni schemi di cifratura.

Le reti che si basano su questi concetti sono chiamate **Reti SPN - Substitution Permutation Networks**, sono dei sistemi prodotto che combinano tecniche di sostituzione ( S-BOX ) con tecniche di permutazione ( P-BOX ).

Partiamo dal chiarire che:

- **confusione** non coincide con **sostituzione**
- **diffusione** non coincide con **permutazione**

**Diffusione:** Consiste in un complesso meccanismo che annulla la struttura statistica del testo in chiaro presente nel testo cifrato.

**Confusione:** Consiste in un complesso meccanismo di sostituzione che permette di rendere impossibile l'individuazione della chiave anche se si hanno informazioni statistiche sul testo cifrato.

Vediamo la struttura del cifrario di Feistel

**Input** del cifrario sono:

- Plaintext di  $2w$  bit
- Chiave  $k$

**Output** del cifrario sono:

- Ciphertext

Vengono effettuate **n fasi** in ognuna delle quali vengono eseguite delle operazioni dipendenti da:

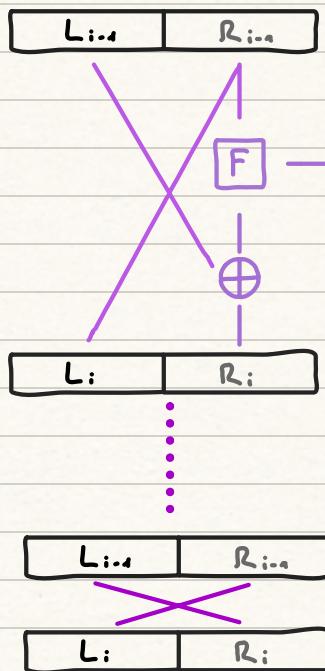
- Una sottochiave  $k_i$  della chiave  $k$
- Una funzione di fase  $F$

Aumentando la dimensione di  $W$  o di  $k$  aumenta la sicurezza a discapito delle prestazioni.

I parametri che definiscono la rete sono:

- Dimensione del blocco **2w**
- Dimensione della chiave **k**
- Numero di fasi **n**
- Algoritmo di generazione delle sottochiavi  **$k_i$**
- Funzione di fase **F**

La singola fase  $i$ -esima si svolge in questo modo:



Nota:

- La funzione  $F$  rimane la stessa per ogni fase
- La sottochiave  $k_i$  cambia per ogni fase e viene generata da un apposito algoritmo, parametri in ingresso di questo algoritmo sono i numero di fase e  $k$  chiave

- L'ultima fase prevede che non vi sia nient'altro che uno scambio tra  $L$  ed  $R$ .

Per decrittografare il ciphertext basta ripetere i passi al contrario utilizzando al contrario le sottochiavi, invece di partire da  **$k_1$**  parto da  **$k_n$** .

# Cifratura DES - Data Encryption Standard

Lo schema di crittografia più utilizzato è DES, fu adottato nel 1977. In DES i dati vengono crittografati in **blocchi da 64 bit** utilizzando una **chiave di 56 bit**. Questo algoritmo, basandosi sulla **struttura di Feistel** permette una decifratura a partire dalla stessa chiave.

Alla base di DES sta un Algoritmo sviluppato da IBM chiamato LUCIFER, esso utilizzava chiavi a 128 bit per cifrare blocchi di 64 bit, l'intento di IBM era quello di creare un sistema ingrado di funzionare in Hardware, per questo motivo la chiave venne ridotta da 128 bit a 56 bit. A seguito di un bando di gara he richiedeva di trovare un sistema di crittografia che potesse essere riconosciuto come standard, IBM partecipò e LUCIFER a 56 bit divenne DES.

Controversie:

- Al giorno d'oggi ha uno spazio di chiavi troppo piccolo, soggetto a possibile bruteforce
- Essendo che l'implementazione interna delle S-BOX è rimasta privata e non di pubblico dominio, si suppone che queste info siano state usate a favore di chi ne era a conoscenza per fare analisi crittografica.

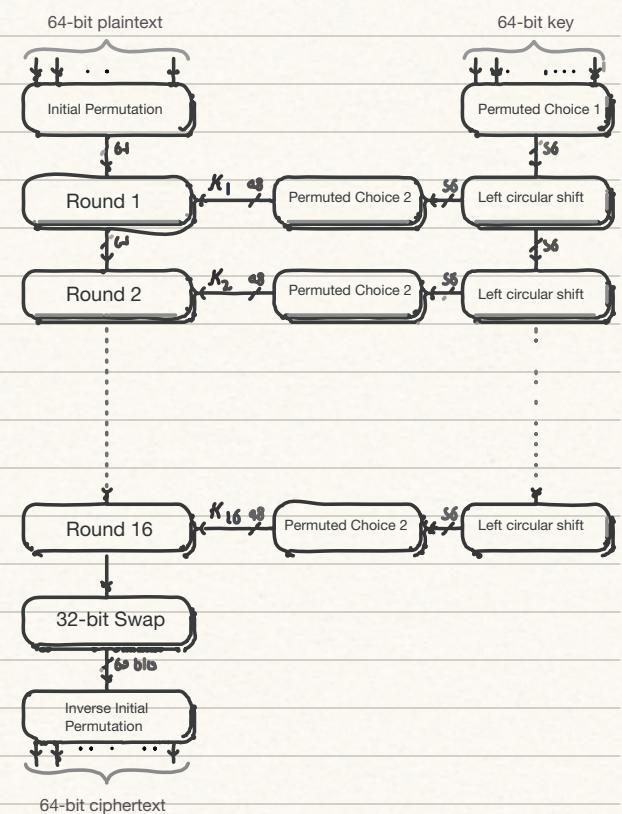
## Schema di Cifratura DES

Osservando lo schema, l'elaborazione avviene in tre fasi distinte:

- **Initial Permutation:** Il testo in chiaro viene sottoposto ad una prima permutazione IP.
- **16 Round:** Questa fase prevede 16 round dove viene applicata la struttura di Feistel con una particolare funzione F ed XOR di  $k_i$ .
- **Swap e Permutazione inversa:** Questa fase prevede uno swap delle due parole di 32 bit e relativa permutazione inversa  $IP^{-1}$ .

Trattandosi di un applicazione del cifrario di Feistel ( a meno delle permutazioni iniziali e finali ) la decifratura verrà fatta dando in pasto alla struttura il ciphertext e le sottochiavi in ordine inverso.

Per quanto riguarda l'algoritmo di generazione delle sottochiavi basti sapere per adesso che data una chiave di 64 bit ad ogni fase viene generata una sottochiave di 48 bit.



## Permutazione iniziale (IP) e inversa ( $IP^{-1}$ )

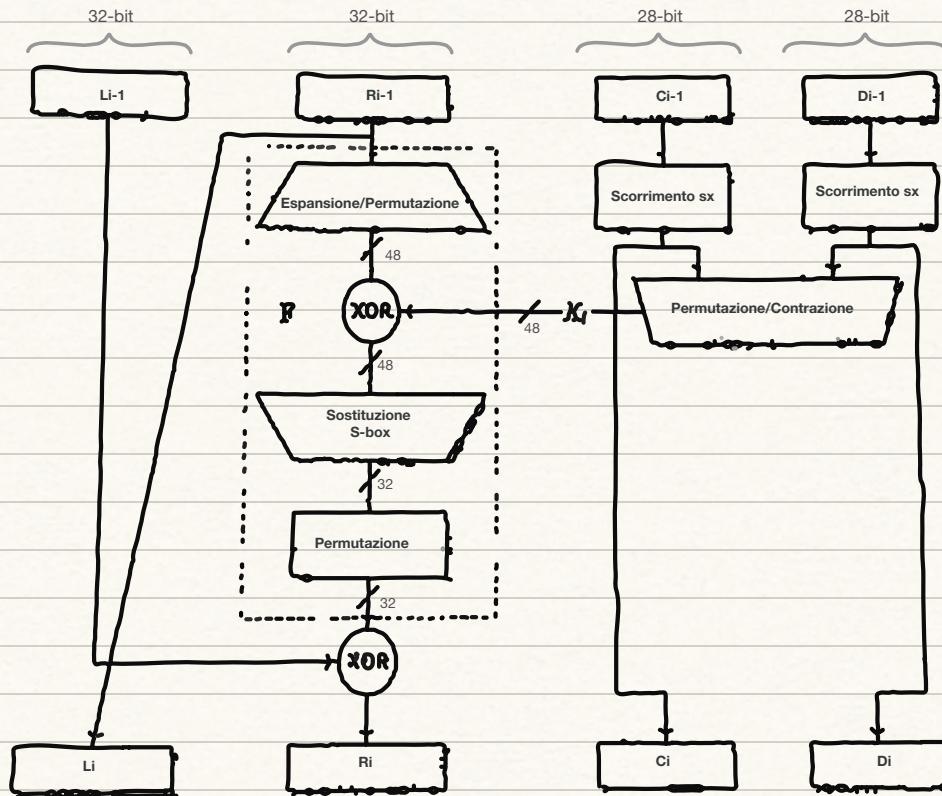
La permutazione iniziale e la sua inversa sono definite tramite due tabelle del tipo:

IP TABLE	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

La tabella inversa effettua le permutazioni al contrario, ovvero secondo la tabella precedente, il bit in posizione 1 verrà mappato in posizione 58, quindi nella tabella  $IP^{-1}$  il bit in posizione 58 sarà mappato in posizione 1.

## ROUND DES

La struttura del generico Round DES è del tutto riconducibile alla struttura di un cifrario di Feistel:



Come in Feistel i blocchi dell'i-esimo round sono ricavati nel seguente modo:

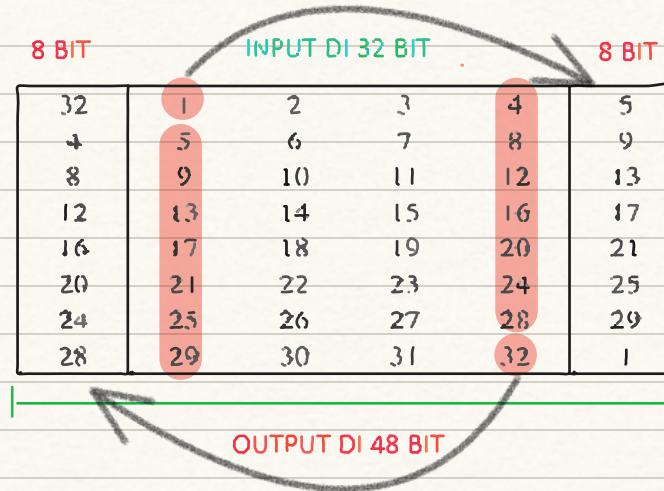
$$L_i = R_{i-1} \rightarrow \text{blocco sinistro}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \rightarrow \text{blocco destro}$$

Bisogna solo andare a specificare come è implementata la funzione **F** e l'algoritmo di generazione delle sottochiavi.

Il primo passo che viene fatto nella funzione **F** è applicare un **espansione/permutazione**, si ha infatti che per effettuare l'operazione di **XOR** con la sottochiave di 48 bit dobbiamo espandere i 32 bit del blocco destro.

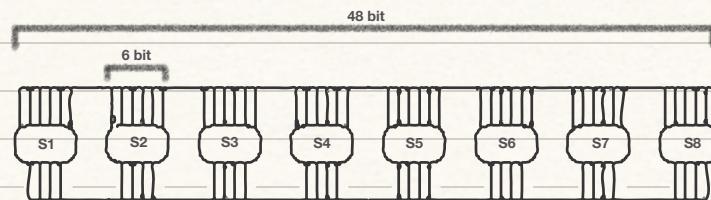
Per effettuare l'espansione si va ad utilizzare una **P-box** di questo tipo:



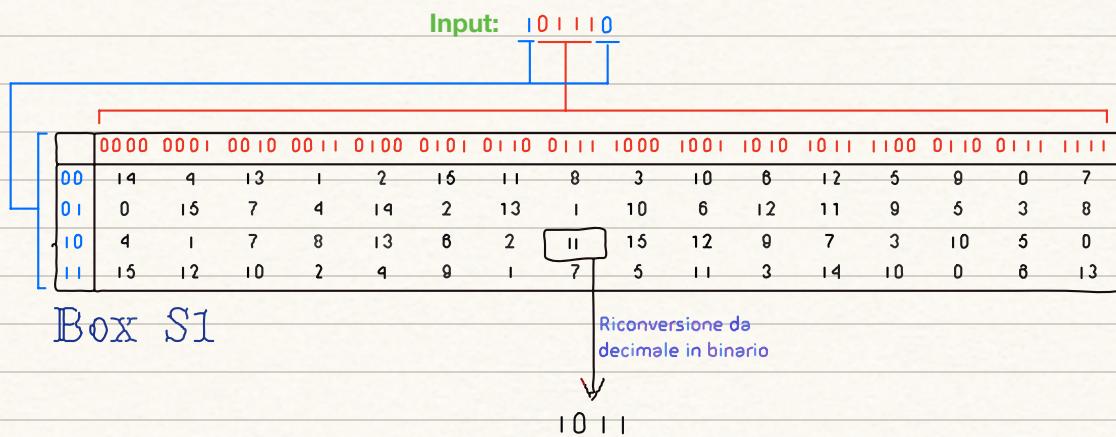
Si aggiungono due colonne esterne copiate dall'input, ma shiftate una verso il basso ed una verso l'alto.

Ovviamente rimane una P-box, solo che alcuni bit vengono mappati su più posizioni.

Successivamente all'applicazione della sottochiave con l'**XOR**, viene effettuata l'operazione di **sostituzione S-box**, in questo caso bisogna riportare l'output ad una dimensione a 32 bit partendo da un input di 48.

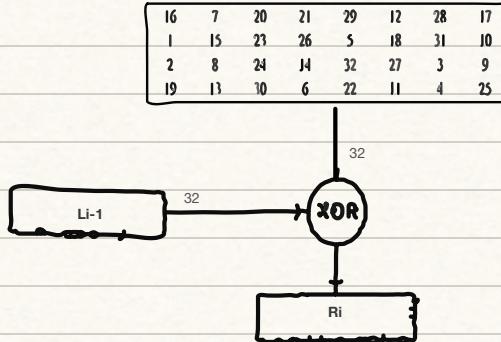


Si utilizzano 8 S-box (ognuna caratterizzata da una tabella di sostituzione differente), ogni una di queste prende in ingresso 6 bit e ne restituisce in output 4, il mapping per la sostituzione (reversibile) è definito dalla seguente tabella:



Si conclude il round con:

- P-box che effettua una permutazione dei 32 bit
- Applicazione dell'operatore **XOR** tra l'output e **Li-1**



## Generazione delle sottochiavi

Des prende in ingresso una chiave di 64 bit, di questi ne vengono usati solo 56, ogni 7 bit ne viene scartato 1.

1	2	3	4	5	6	7	8
9	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

Successivamente viene utilizzata una P-box (Permuted Choice 1) che oltre ad effettuare una permutazione va a suddividere la stringa da 56 bit in due da 28.

Queste due stringhe di bit vengono chiamate **Ci** e **Di**.

C <sub>0</sub> 28 bit	57	49	41	33	25	17	9
	1	58	50	42	34	26	18
D <sub>0</sub> 28 bit	10	2	59	51	43	35	27
	19	11	3	60	52	44	36
C <sub>1</sub> 28 bit	63	55	47	39	31	23	15
	7	62	54	46	38	30	22
D <sub>1</sub> 28 bit	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Entrambe le due stringhe di bit entrano in un blocco di **Left Circular Shift** (D e C vengono shiftati in maniera distinta).

In base al numero di ROUND lo shift può avvenire di 1 o al massimo 2 bit.

Numero fase	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit ruotati	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Dopo lo shift circolare **C** e **D** vengono date in ingresso ad una P-box 6 x 8 (Permuted Choice 2), quindi prende in ingresso 56 bit e ne da in output 48... Questo viene fatto escludendo dal mapping alcuni bit:

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

Ad esempio, in questa P-box non sono presenti 8 bit : **9-18-22-25-35-38-43-54**

L'output di questa P-box sarà la sottochiave da 48 bit **ki**, mentre **C** e **D** shiftati saranno l'input da 56 bit per la generazione della prossima sottochiave del round successivo.

## Conclusioni DES

Una proprietà desiderabile in un algoritmo di crittografia è che una piccola variazione del testo in chiaro o della chiave dovrebbe produrre una variazione significativa nel testo cifrato, questo è chiamato effetto valanga.

Attacchi possibili:

- **Timing Attack:** Questo attacco si basa sul fatto che molto spesso il tempo di codifica di un 1 o di uno 0 è differente, queste informazioni potrebbero essere utilizzate per risalire alla chiave. Gli autori di DES sostengono che sia piuttosto immune ad attacchi di questo tipo, tuttavia si pensa che esso non sia totalmente immune. Implementazioni più sofisticate come Triple-DES o AES dovrebbero diminuire la probabilità di successo di un attacco di questo tipo.
- **Cryptoanalisis:** Per provare a rompere DES sono stati creati due approcci di analisi crittografica avanzati, questi sono l'**analisi crittografica differenziale** ed **analisi crittografica lineare**; attraverso questi due approcci è stato possibile abbassare la complessità delle operazioni da  $2^{55}$  a  $2^{43}$ .
  - **Analisi crittografica differenziale:** Questa tecnica tenta di scoprire la chiave osservando come piccole variazioni negli input (plaintext + key) si riflettono nell'output, nonostante l'effetto valanga ci sono modelli prevedibili che possono essere sfruttati.
  - **Analisi crittografica lineare:** Con questo tipo di analisi l'obiettivo dell'attacco è la ricerca di approssimazioni lineari capaci di descrivere le trasformazioni eseguite in DES.

## Cifratura Double-DES

Per realizzare questa cifratura si utilizza DES per due volte in cascata, si avranno quindi due chiavi ed un unico plaintext.



Viceversa la decrittografia viene espressa da:



Nota bene l'ordine di applicazione delle chiavi, deve essere inverso.

In questo modo l'ordine delle chiavi passa da  $2^{56}$  a  $2^{112}$ .

Tutto bello... però questo cifrario è vulnerabile ad un attacco **Meet-in-the-Middle**.

### Attacco Meet-in-the-Middle

Si tratta di un attacco di tipo **known-plaintext**.

La cifratura è retta dalla relazione:

$$C = E(K_2, E(K_1, P))$$

Funzionamento dell'attacco:

- Si esegue la crittografia del plaintext per ogni valore di  $k_1$ , quindi si preparano  $2^{56}$  testi cifrati e li si mettono in una tabella.
- Si esegue la decrittografia del ciphertext  $C$  per ogni valore di  $k_2$  e li si mettono in un'altra tabella.
- Si cerca la corrispondenza tra le due tabelle, quelle saranno le chiavi  $k_1$  e  $k_2$ .

Operando in questo modo si è passato da una complessità di  $2^{112}$  a  $2^{56+2^{56}} = 2^{57}$ .

## Cifratura Triple-DES

La probabilità di trovare la chiave con l'attacco Meet-in-the-Middle su un sistema di cifratura Double-DES è molto alta, per questo motivo si utilizza Triple-DES per evitare la simmetria del sistema. Esistono due varianti di Triple-DES:

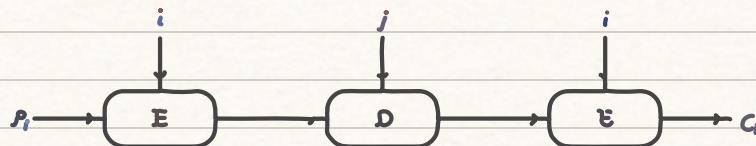
- Con **due chiavi** di cifratura **k1** e **k2**.
- Con **tre chiavi** di cifratura **k1**, **k2** e **k3**.

Aggiungendo una fase di cifratura fa aumentare lo spazio delle chiavi e la complessità, rimane comunque attaccabile con Meet-in-the-Middle anche se risulta abbastanza sicuro poiché per l'attacco si ha una complessità di  $2^{112}$ .

### 3DES con due chiavi

Il sistema prevede una sequenza di crittografia-decrittografia-crittografia **E-D-E** (Encrypt-Decrypt-Encrypt).

$$C = E(K_1, D(K_2, E(K_1, P)))$$
$$P = D(K_1, E(K_2, D(K_1, P)))$$



### 3DES con tre chiavi

Il sistema 3DES con tre chiavi fa utilizzo dello stesso schema **EDE** ma ad ogni step utilizza tre chiavi differenti **k1, k2** e **k3**.

3DES è abbastanza diffuso e sembra essere molto sicuro, tuttavia al crescere della potenza potrebbe non esserlo più.

## Standard AES

Advanced Encryption Standard nacque nel 2001 come standard per sostituire DES, può supportare blocchi di dimensione variabile ma lo standard vincola alle lunghezze 256, 192 e 128 bit.

AES fa utilizzo dell'**aritmetica dei campi finiti**, di seguito alcuni cenni:

In AES tutte le operazioni vengono eseguite su byte da 8 bit, in particolare le operazioni aritmetiche di addizione, moltiplicazione e divisione vengono effettuate sul campo finito **GF(2<sup>8</sup>)**.

Il motivo per il quale si vuole utilizzare un campo finito è che le operazioni effettuate tra numeri binari da 8 bit possano restituire un risultato che continua ad essere appartenente al campo stesso... Facciamo un esempio:

Supponiamo di avere il numero 2 che in binario a 8 bit è rappresentato da 00000010, sappiamo che non esiste nell'insieme dei numeri interi un inverso moltiplicativo b tale che **2b mod 2<sup>8</sup> = 1**. Vogliamo quindi lavorare con un insieme di numeri interi compresi tra 0 e 2<sup>8</sup>-1, e effettuando operazioni su di essi il risultato deve rientrare in questo insieme.

C'è un modo per definire un campo finito contenente 2<sup>n</sup> elementi; tale campo è denominato **GF(2<sup>n</sup>)**.

Si consideri l'insieme S di tutti i polinomi di grado n-1 o meno con coefficienti binari, quindi la formula generale è:

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

Con **a<sub>i</sub> = {0,1}**.

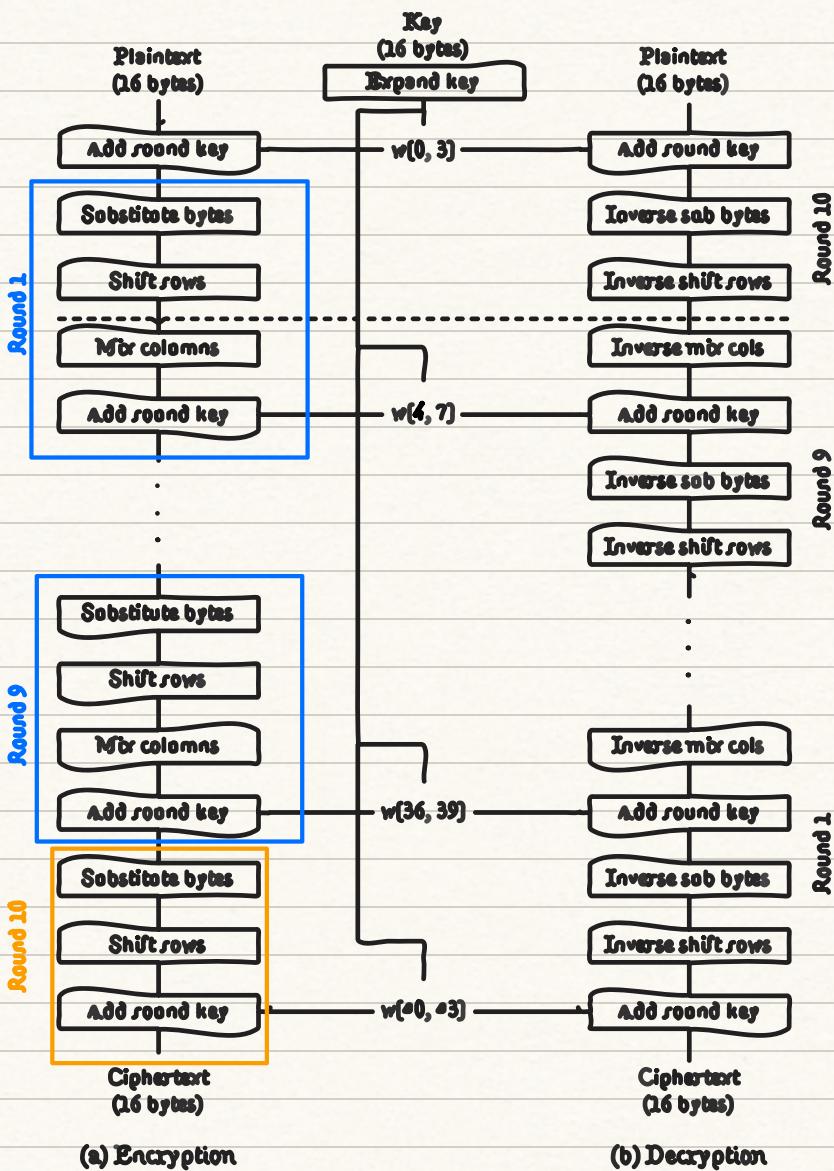
Per n=3 ci sono 8 polinomi differenti nell'insieme:

0    1    x    x+1    x<sup>2</sup>    x<sup>2</sup>+1    x<sup>2</sup>+x    x<sup>2</sup>+x+1

L'aritmetica segue le solite regole dell'aritmetica polinomiale, tranne per le due seguenti regole:

- L'aritmetica viene eseguita in modulo 2, è uguale all'operazione **XOR**
- Se effettuando la moltiplicazione risulta un polinomio di grado maggiore di n-1 **q(x)**, allora viene scelto un polinomio irriducibile **m(x)** di grado n, poi dividiamo per m(x) e manteniamo il resto.

## Struttura generale

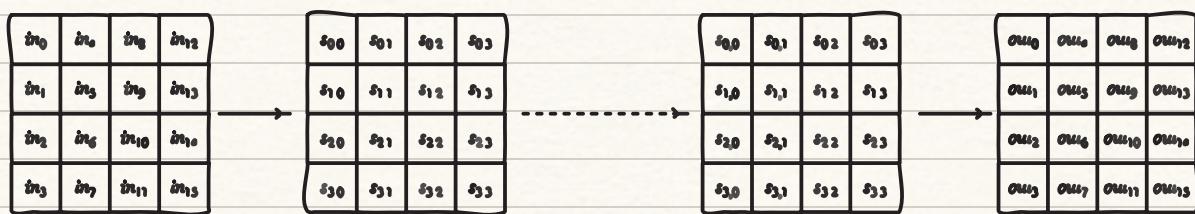


Come si vedde dallo schematico, ogni blocco (sia per la cifratura che per la decifratura) prende sempre in ingresso 16 byte / 128 bit.

Anche la chiave prende in ingresso 16 byte ed effettua un espansione generando diverse sottochiavi.

Dallo schema si può vedere la forte somiglianza con la struttura di DES anche se AES fa utilizzo di funzioni piuttosto diverse.

Prima di entrare nello schematico il blocco in input viene modificato, ovvero viene effettuata una trasformazione iniziale che mette i 16 byte in una matrice 4x4, quindi un byte per casella, questa rappresenta lo **State** che verrà modificato a ciascun round, il **final State** sarà ritrasformato in un singolo blocco da 128 bit e sarà l'output cifrato.



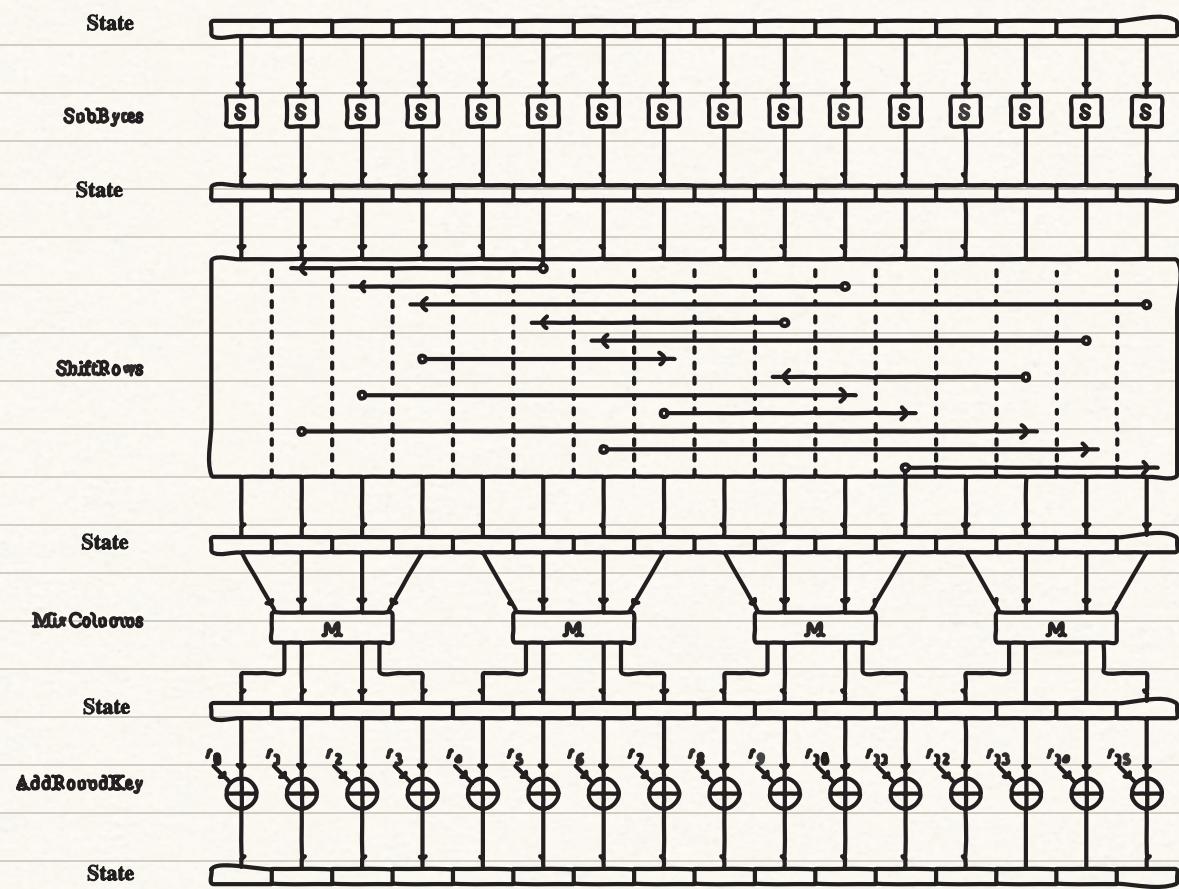
Per poter effettuare le varie operazioni, la chiave di 16 byte /128 bit in forma matriciale 4x4 viene espansa in un array di word dove ciascuna word occupa 4 byte. Quindi a partire da 16 byte si generano 44 word da 4 byte, ovvero 176 byte.



## Round AES

Una generica fase crittografica di AES prevede 4 operazioni:

- **Substitution bytes** (Sostituzione): Si utilizzano dell S-box
- **Shift rows** (Permutazione): Viene effettuata una permutazione di righe
- **Mix Column** (Sostituzione): Viene effettuata una sostituzione sulla base dell'aritmetica GF( $2^{8}$ )
- **Add round key** (Sostituzione): Viene effettuata una operazione di XOR bit a bit del blocco corrente con una porzione della chiave espansa



## Substitution Bytes

In ingresso abbiamo 16 byte, ognuno di questi byte viene dato in ingresso ad una **S-box** che è rappresentata da una matrice 16x16.

Si utilizzano i primi 4 bit per indicare la riga ed i restanti 4 bit per selezionare la colonna; In questo modo avremo in output un valore da 0 a 255 rappresentabile con 8 bit, quindi un byte entra, un altro esce.

Inout {95}															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C8	30	11	67	23	FE	01	AB
1	C1	P2	C9	7D	F1	59	07	F0	AD	D9	A2	A5	YC	1d	72
2	87	FD	93	20	76	3E	F7	CC	30	A5	E5	F1	71	D8	31
3	1U	C7	23	C3	18	96	05	0A	17	12	81	E2	EB	27	B2
4	110	83	2C	1A	J8	6E	5A	F0	S2	38	D6	03	29	E7	2F
5	S3	D1	1U	ED	21	FC	B1	SB	0A	CB	9E	3y	-P	0C	Sb
6	DU	EF	18	F8	03	0D	33	83	75	Fy	02	7F	S11	3C	9F
7	S1	A3	0U	5F	02	90	3h	F5	BC	8h	0A	21	1U	FF	F3
8	CO	1C	1J	EC	SF	97	03	17	C1	A7	7E	30	6h	50	10
9	60	8L	0f	DC	22	90	88	05	EE	B8	10	DE	S2	08	03
A	EU	32	3A	0A	0y	06	20	SC	C2	03	AC	62	91	91	E1
B	E7	Ch	37	6D	K0	DS	0E	09	6C	S6	F7	E^	65	7A	^E
C	3A	78	25	2E	JC	46	Bu	CA	EP	DD	7J	IF	4B	BD	JB
D	7U	3E	89	66	d3	0J	F6	0E	61	30	S7	B9	d6	Cl	10
E	E1	F6	48	11	6Y	09	8E	4W	98	1E	87	E9	CE	S1	2h
F	8C	11	89	0D	3F	E6	U2	6h	11	9y	2D	0F	80	50	BB

Ovviamente esiste una matrice di trasformazione diretta per la cifratura ed una di trasformazione inversa per la decifratura.

Ecco come vengono costruite queste matrici:

- Creiamo la matrice 16x16, ed associamo ad ogni posizione **{XY}** l'inverso moltiplicativo nel campo finito **GF(2^8)**
- Per tutti i byte della S-box viene applicata la seguente trasformazione matriciale:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

**X**                                   **C**

Per quanto riguarda la trasformazione inversa si utilizza un'altra equazione matriciale che risulta essere inversa della trasformazione diretta, quindi date le matrici **X** e **C** esistono due matrici **Y** e **D** che generano la funzione di trasformazione inversa.

## Shift Rows

Dopo l'operazione di ByteSubstitution, la matrice state subisce l'operazione di **ShiftRows** che è l'unica permutazione all'interno del round AES.

Per la **trasformazione diretta** avviene uno shift circolare (**byte level**) a **sinistra** della matrice:

- **Prima riga:** Nessuno shift
- **Seconda riga:** Shift di **1 posizione** dei byte
- **Terza riga:** Shift di **2 posizione** dei byte
- **Quarta riga:** Shift di **3 posizione** dei byte

Ingresso				Uscita			
A	B	C	D	A	B	C	D
A	B	C	D	B	C	D	A
A	B	C	D	C	D	A	B
A	B	C	D	D	A	B	C

Per la **trasformazione inversa** basta applicare lo shift a **Destra** allo stesso modo.

## Mix Column

Pure questa è un'operazione di sostituzione, data la matrice state, ciascun byte di una colonna viene mappato in un nuovo valore che è funzione di tutti e 4 i byte della stessa colonna in input.

Per fare questo viene moltiplicata la matrice State per un'altra matrice X:

$$\begin{matrix} \text{X} & \text{State} \\ \left[ \begin{matrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{matrix} \right] & \times \left[ \begin{matrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{matrix} \right] = \left[ \begin{matrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{matrix} \right] \end{matrix}$$

Nello sviluppo del prodotto riga per colonna si fa utilizzo dell'aritmetica in **GF(2^8)**.

Per la trasformazione inversa, si procede allo stesso modo ma si fa utilizzo di una matrice inversa **X^-1**.

L'importanza di questa operazione sta nel fatto che, dopo alcuni round, tutti i bit di output dipendono da tutti i bit in input (**Diffusione**).

## Add Round Key

Nell'operazione di **Add Round Key** viene effettuato un'operazione di **XOR bit a bit** tra i 16 byte della matrice state ed i 16 byte della chiave di fase. In questo caso l'operazione è la stessa sia per la trasformazione inversa che diretta in quanto XOR è invertibile.

La sicurezza di questa fase è data dalla complessità dell'**algoritmo di espansione della chiave**.

## Espansione della chiave

L'algoritmo di espansione della chiave richiede come input una chiave di 16 byte / 128 bit.

Questi **16 byte** vengono suddivisi in **4 word** da 4 byte, l'algoritmo di espansione genera un array lineare di 44 word.

Il numero 44 perchè per ogni round vengono utilizzate 4 word, ricordando che prima di entrare nel primo round viene applicata un operazione di **AddRoundKey** e che i round sono (di default) 10.

La chiave viene espansa nel seguente modo:

- La chiave viene copiata nelle prime 4 word della chiave espansa
- La **w<sub>i</sub>** successiva viene calcolata in due modi:
  - Se **i** è un multiplo di 4(o **N** numero di word):
    - Viene effettuato l'operazione di **XOR** tra la word **w<sub>i-4</sub>** e l'output di una funzione **g** che prende in ingresso **w<sub>i-1</sub>**
  - Se **i** non è multiplo di 4 (o **N**):
    - Viene effettuato un semplice **XOR** tra la word **w<sub>i-4</sub>** e **w<sub>i-1</sub>**

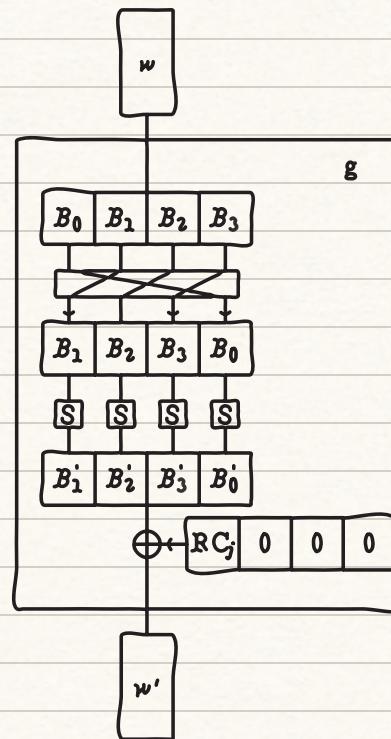
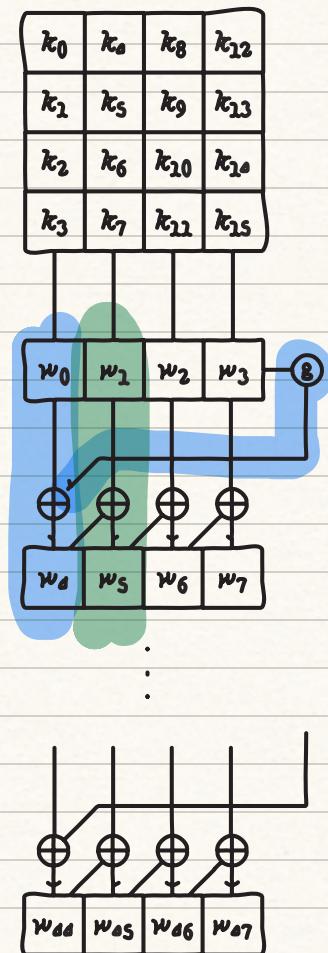
Si ha quindi che l'applicazione della funzione **g** ogni 4 word applica complessità, questo algoritmo è stato progettato per resistere all'analisi crittografica.

Ecco il dettaglio di come funziona la funzione **g**:

Come già detto prende in ingresso una word, (in particolare **w<sub>i-1</sub>**) quindi 4 byte;

Ecco cosa succede:

- **RotWord**: Viene effettuato uno scorrimento circolare a sinistra di un byte
- **SubWord**: Viene applicata una sostituzione tramite S-box (Identico a **Byte Substitution**)
- Viene applicata un'operazione di **XOR** con una costante di fase **RC<sub>j</sub>** il cui valore dipende dal numero di fase **j**. I valori assunti da **RC<sub>j</sub>** sono comunque valori a 8 bit, i 24 bit meno significativi sono impostati a 0.



## Cifratura inversa equivalente

Uno degli svantaggi di AES è dato dal fatto che la sequenza di cifratura è diversa da quella della decifratura, non è infatti possibile utilizzare lo stesso modulo software/hardware per effettuare la decifratura, infatti:

- Avendo uno schema di cifratura **SubByte - ShiftRows - MixColumn - AddRoundKey**
- Per decifrare correttamente bisogna prendere le prime due operazioni e invertirle, poi invertire pure le ultime due:  
**Inverse ShiftRow - Inverse SubByte - AddRoundKey - Inverse MixColumn**

Ovviamente oltre un aspetto di ordine delle operazioni bisogna effettuare le operazioni inverse come visto nella spiegazione delle varie fasi.

## AES-192 & AES-256

Come già detto è possibile avere un grandezza maggiore dei blocchi crittografati, è possibile applicare quanto visto finora utilizzando invece di 16 byte un numero di 24 byte per **AES-192** (sia per il plaintext che per la chiave), 32 byte per **AES-256**.

Ovviamente cambierà anche l'espansione della chiave, dove la funzione g viene applicata invece che ogni 4 word ogni 6 o ogni 8, in base alla versione di AES.

## Tecniche di concatenazione per Block Cipher

Nella pratica, per utilizzare un cifrario a blocchi è spesso necessario dover gestire una quantità arbitraria di dati, o comunque scegliere come cifrare un macro blocco maggiore della dimensione del blocco da dare in input allo schema di cifratura.

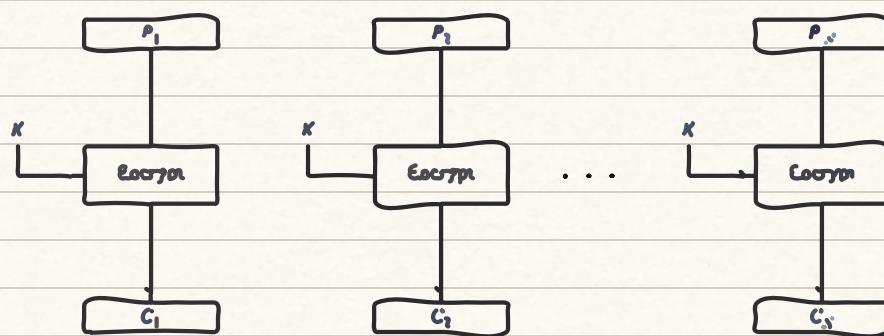
Per applicare un codice a blocchi in una varietà di applicazioni, sono state definite dal NIST cinque modalità operative; Una modalità operativa è una metodologia per applicare l'algoritmo di cifratura ad un caso d'uso reale e ad un'applicazione.

Esse sono quindi pensate per qualsiasi codice a blocchi simmetrico, inclusi DES e AES.

### Electronic CodeBlock (ECB)

Tecnica più semplice e più ovvia, il plaintext viene suddiviso in blocchi ed ogni blocco viene gestito indipendentemente dagli altri utilizzando la stessa chiave.

$$C_i = E_k(P_i)$$



Il vantaggio di questa tecnica sta nella **indipendenza di ogni blocco dagli altri**, infatti se dovesse avvenire un errore di trasmissione in un generico blocco  $i$ -esimo, i successivi non saranno invalidati (ottimo per **UDP** che non ha un controllo sugli errori).

Ovviamente un ciphertext difficilmente sarà un multiplo della dimensione del blocco, per questo si utilizzano dei **padding** di riempimento identificando una codifica per segnalare la fine del testo e l'inizio del padding.

La **decifratura** avviene allo stesso modo.

Un **problema** di ECB è che lo schema di cifratura per blocchi grandi risulta essere ricorrente e potrebbe dare informazioni sul testo in chiaro, per esempio:

1) **Maria e simpatica**

**000101 111100 101010**

2) **Luigi e anipatico**

**110101 111100 111111**

**Potrei mandare un messaggio del tipo:**

**000101 111100 111111**

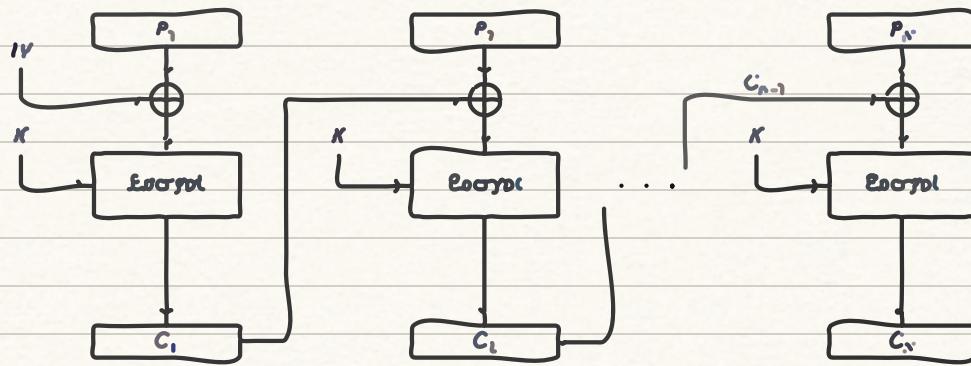
**che equivale a dire "Maria e anipatico".**

## Cipher Block Chaining (CBC)

Visti i problemi di ECB, si pensò ad una nuova modalità operativa in grado di produrre testi cifrati differenti relativi a testi in chiaro che si ripetono.

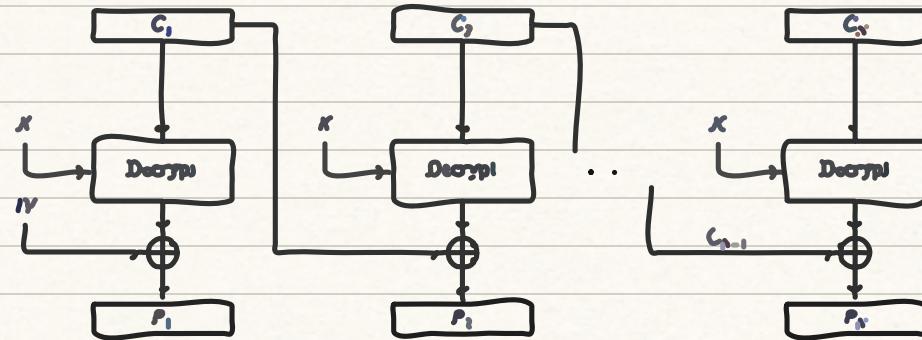
In CBC non si ha più l'indipendenza tra i vari blocchi, infatti si utilizza uno schema che fa uso del precedente blocco cifrato in XOR con il testo in chiaro corrente.

$$C_i = E_k(P_i \oplus C_{i-1})$$



Ovviamente alla prima iterazione non abbiamo un ciphertext dell'iterazione precedente, per questo si utilizza infatti un **Initialization Vector (IV)** che di norma non rappresenta un elemento di sicurezza e potrebbe anche essere trasmesso in chiaro.

La decifratura avviene al contrario:



CBC ha principalmente due problemi, il primo è l'utilizzo di **IV**, un attaccante potrebbe concentrarsi sulla cattura del primo blocco in quanto potrebbe facilmente ottenere IV poiché trasmesso in chiaro.

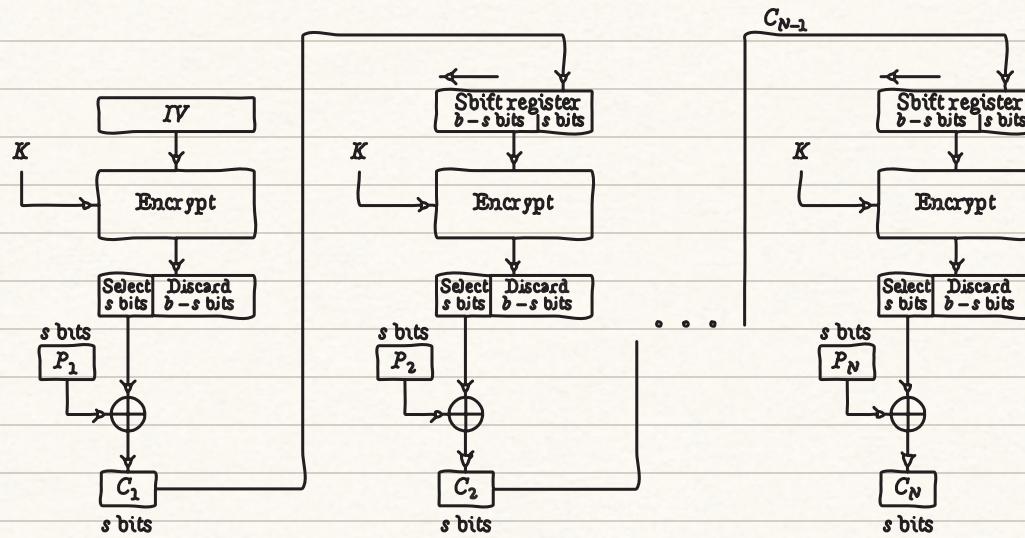
Un secondo problema è la dipendenza tra i vari blocchi, se si corrompe un blocco perdo tutti i successivi.

# Cipher FeedBack (CFB)

Questa modalità operativa nasce per l'esigenza di utilizzare cifrari a blocchi per dei flussi stream senza l'utilizzo di padding.

Una proprietà auspicabile nelle cifrature stream è quello di avere un ciphertext grande quanto il testo in chiaro, ovvero... sarebbe uno spreco inviare 8 bit cifrati per un plaintext di 7 bit.

L'idea alla base è quella di cifrare sempre un blocco di **b** bit fissi e di applicare in **XOR** gli **s** bit che rappresentano l'unità minima di dimensione dello stream, può essere un bit o uno o più byte, e successivamente trasmettere solo la parte di ciphertext contenente l'informazione di **s** bit. Per aggiungere la dipendenza tra i vari blocchi cifrati si utilizza pure il feedback.

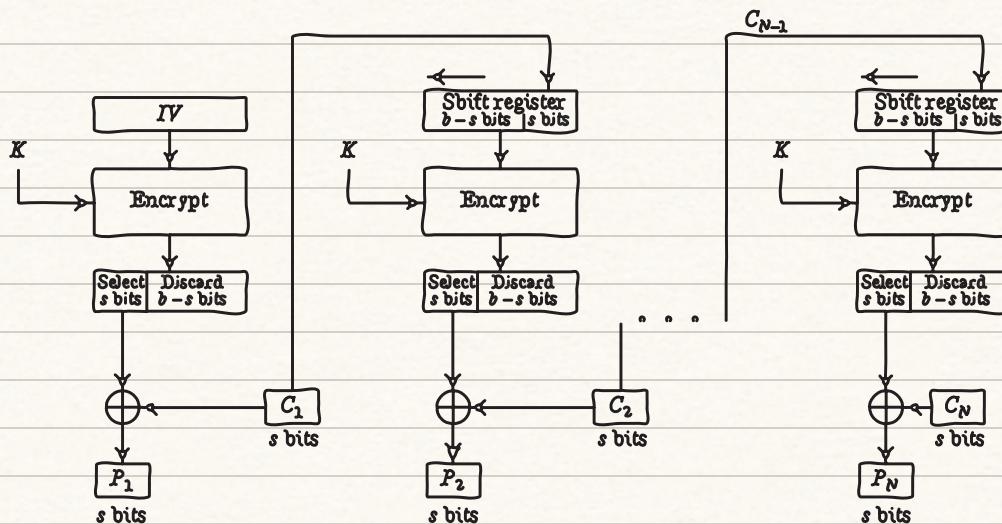


Quindi quello che succede è:

- Inizializzo IV in uno shift register di **b** bit
- Effettuo la cifratura di questo vettore
- Dall'output cifrato considero solo la quantità di **s** bit interessati che devo trasmettere/nascondere
- Ottenuti gli **s** bit cifrati **C1**, aggiorno lo shift register inserendo **C1** (Lo shift register alla prima iterazione contiene il valore IV, alla seconda iterazione conterrà n **b-s** bit meno significativi di IV concatenato ad **C1**)
- Ricomincio il ciclo con il contenuto dello shift register differente e dipendente dal precedente blocco.

In questo modo abbiamo la possibilità di cifrare un plaintext di arbitraria lunghezza (comunque minore della dimensione del blocco di cifratura) senza l'introduzione di padding.

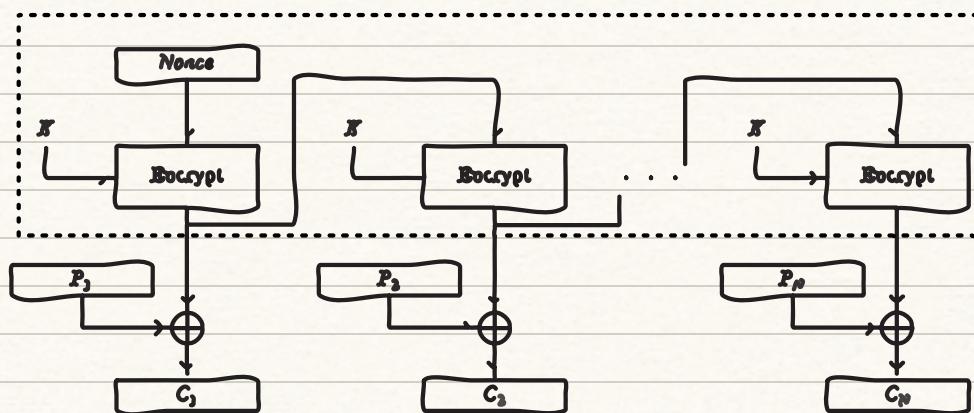
Ecco lo schema di decifratura:



## Output FeedBack (OFB)

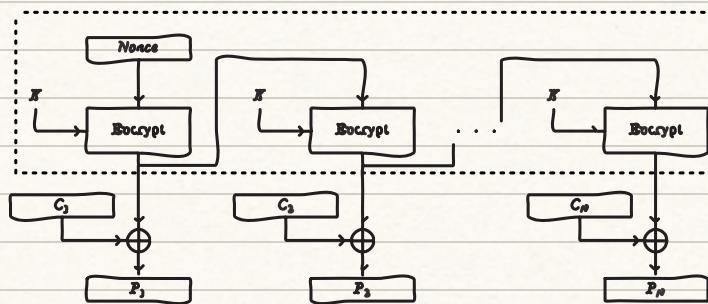
Per risolvere il problema di CFB, l'Output Feedback (OFB). Questa modalità operativa è simile nella struttura a CFB ed anche lei è può essere utilizzata per cifrare flussi stream.

In OFB, invece di mandare in input al blocco successivo il testo cifrato mandiamo l'output del blocco di cifratura.



Qualora il plaintext **P** fosse inferiore di dimensione all'output del blocco di cifratura, si scartano gli **b-s** bit meno significativi e si effettua l'**XOR** con il plaintext.

In questa modalità operativa si risolve il problema di ECB pur mantenendo la indipendenza dei blocchi ad eventuali errori, inoltre lo schema di decifratura è lo stesso.

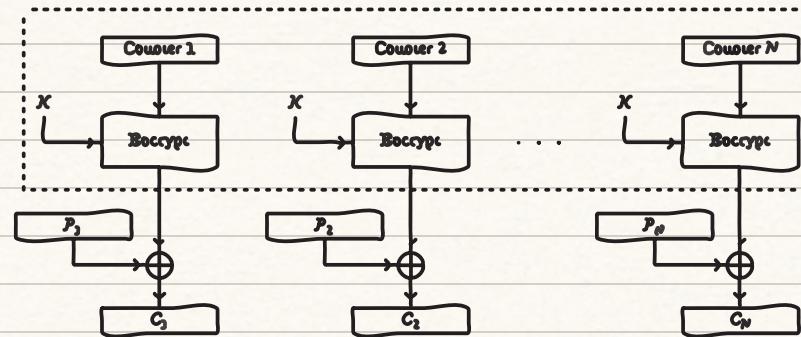


## Counter (CTR)

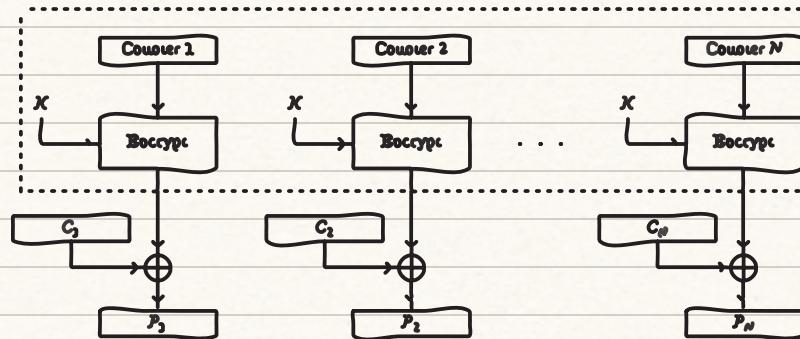
La modalità operativa Counter è probabilmente la più robusta tra quelle citate fino ad ora, per questo motivo trova utilizzo in molti standard di sicurezza come **IPSec**.

Viene utilizzato un contatore corrispondente alle dimensioni del blocco di testo in chiaro, un requisito è che il valore di questo numero sia diverso ad ogni cifratura, è un implementazione comune quella di incrementare ad ogni iterazione... da qui il nome CTR.

La crittografia prevede che vengano inizializzati tanti contatori quanti sono i blocchi in cui è suddiviso il messaggio (un vantaggio di questa modalità operativa è che è possibile lavorare in parallelo), cifrato il contenuto del contatore si effettua un XOR con il Plaintext ed il gioco è fatto.



La decrittografia avviene allo stesso modo:



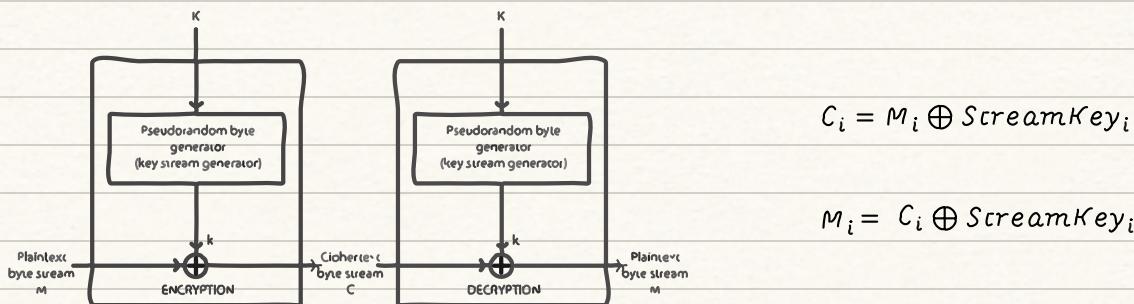
Vantaggi:

- Efficienza sia hw che sw, possibilità di lavorare in parallelo
- La semplicità è un altro grande vantaggio di CTR, oltre al fatto che è possibile utilizzare lo stesso modulo hw/sw per la cifratura e decifratura

## Stream Cipher

Una tipica cifratura a flussi deve poter cifrare in tempo reale unità di testo in tempo reale senza richiedere una dimensione minima per un blocco, in generale vengono progettati per cifrare flussi di bit, byte o multipli di byte.

Vi è una chiave che funge da input per un generatore di numeri pseudocasuali, il numero generato viene applicato in XOR al plaintext per mascherarlo.



Nei computer non è possibile generare numeri casuali, infatti un **generatore di numeri pseudocasuali** da l'impressione di dare in output una sequenza casuale, ma non è così:

- È una sequenza e si ripete periodicamente ogni TOT di simboli generati, tanto è maggiore il periodo tanto migliore e sicuro è il generatore.
- Ovviamente per dare l'impressione di essere casuale la probabilità di ottenere in output un simbolo deve essere uguale a quello degli altri, distribuzione uniforme.
- La sequenza in uscita è definita dalla chiave in ingresso all'algoritmo, questa chiave si chiama **seme**, inoltre la sequenza sarà replicabile a partire dal seme.

## RC4

Questo è uno dei cifrari a stream più utilizzati, nasce per cifrare singoli byte.

Questo genere di cifrari funziona sfruttando lo XOR della chiave con lo stream in ingresso, questa cifratura può paradossalmente cifrare testi all'infinito.

Fulcro di tutto è la generazione di numeri random in funzione della chiave/seme.

RC4 è utilizzato sia nei protocolli di sicurezza web **SSL/TLS** che nei protocolli di sicurezza LAN Wireless **WEP** e **WPA**.

### Algoritmo

- L'algoritmo usa una chiave di lunghezza variabile tra 1 e 256 byte, questa chiave viene messa all'interno di un vettore temporaneo **T** di 256 caratteri di un byte. Qualora la chiave fosse di dimensione inferiore a 256 essa viene ripetuta fino a riempire l'intero vettore, troncando comunque al raggiungimento della 256-esima posizione.
- Si inizializza un vettore state **S** di 256 caratteri di un byte, in ogni posizione **S[i] = i**, avremo quindi **S = [ 0, 1, 2, 3, 4 ... , 254, 255 ]**
- Sulla base dei valori in **T** avvengono delle permutazioni in **S**, in questo modo avremo un vettore **S** non più ordinato, ma permutato in funzione della chiave in **T**.
- Da adesso avviene la generazione del flusso a partire dal vettore di stato **S**, da ora in poi **T** e la chiave non saranno più utilizzati.
  - Questa fase prevede le seguenti operazioni principali:
    - Swap di due elementi di **S** tramite l'utilizzo di due indici **i** e **j** aggiornati all'interno del ciclo stesso.
    - Calcolo di un terzo indice **t** utilizzato per selezionare quale dei 256 elementi di **S** dare in output, il valore in output sarà **k**.
    - Alla fine viene effettuata l'operazione di XOR tra il byte in chiaro ed il numero pseudocasuale **k**.

Esistono metodi di **attacco** ad **RC4**, essi però non sono fattibili utilizzando una chiave di almeno **128 Bit**.

## Cifratura asimmetrica

La crittografia asimmetrica, detta anche a chiave pubblica, è un tipo di crittografia nella quale sia la crittografia che la decrittografia utilizzano due chiavi differenti, una chiave pubblica ed una privata. Questo tipo di crittografia permette di cifrare con una chiave **k1** e decifrare con l'altra **k2**.

La crittografia asimmetrica può essere utilizzata per garantire:

- Segretezza
- Autenticazione
- O entrambe

La crittografia a chiave pubblica pone le sue fondamenta su concetti avanzati della teoria dei numeri (non sono quindi utilizzate tecniche di sostituzione e permutazione)

I principali vantaggi del suo utilizzo sono:

- **Agevolazioni nella distribuzione della chiave**, è possibile comunicare le chiavi in modo sicuro e in chiaro. Attenzione, non significa che possono essere trasmesse in chiaro senza problemi, bisogna prendere delle precauzioni.
- **Agevolazioni nella firma digitale**, si vuole garantire che un messaggio digitale sia stato effettivamente prodotto da una determinata persona.

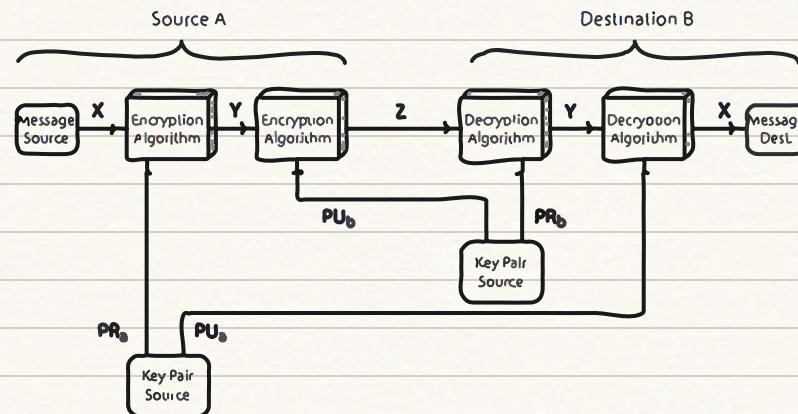
Proprietà importanti della cripitura simmetrica sono:

- È computazionalmente impossibile determinare una chiave **k1** conoscendo la sua opposta **k2** e l'algoritmo di cripitura.
- Inoltre è possibile utilizzare una qualsiasi delle due chiavi per cifrare un messaggio, la decifratura potrà essere effettuata solo conoscendo l'altra chiave.

In base a come vengono applicate le chiavi è possibile ottenere proprietà di **segretezza** o **autenticità**:

- **Cifratura con chiave pubblica destinatario**: Il messaggio potrà essere decifrato solo applicando la chiave privata del destinatario, quindi solo il destinatario sarà in grado di ottenere il plaintext. Proprietà di **segretezza** ottenuta, si garantisce pure la proprietà di **integrità** poiché non sarà possibile modificare il messaggio se non si è a conoscenza della chiave privata.
- **Cifratura con chiave privata mittente**: Il messaggio potrà essere decifrato solo applicando la chiave pubblica del mittente, questo non offre segretezza ma bensì l'autenticità poiché quel messaggio potrà averlo prodotto solo il mittente, chiunque è a conoscenza della chiave pubblica del mittente potrà leggere il messaggio essendo sicuro della sua provenienza. Anche in questo caso viene offerta integrità poiché, se si volesse modificare il messaggio si dovrebbe decifrare (possibile farlo), modificare e cifrare nuovamente (questa ultima operazione è impossibile non essendo a conoscenza della chiave privata del mittente).

Ci chiediamo se è possibile ottenere le tre proprietà di **Segretezza, Autenticità e Integrità**, in realtà è possibile a patto di utilizzare tutte e 4 le chiavi:



Di seguito i passaggi da effettuare:

- Lato mittente A:
  - Cifro con la chiave privata del mittente A
  - Cifro con la chiave pubblica del mittente B
- Lato destinatario B:
  - Decifro con la chiave privata del mittente B (Potrà farlo solo B, segretezza garantita)
  - Decifro con la chiave pubblica del mittente A (Mi permette di verificare che solo A avrebbe potuto cifrare il messaggio, autenticità garantita)

In generale l'**integrità** è garantita poiché entrambe le versioni banali visti precedentemente già lo facevano.

Questa soluzione è molto potente e sicura, tuttavia ha un costo computazionale molto elevato, per questo motivo questo procedimento è spesso utilizzato per scambio di chiavi di sessione che verranno utilizzate per dei cifrari simmetrici a blocchi, questi infatti sono molto più performanti ed efficienti.

## Algoritmo RSA

L' algoritmo RSA nasce nel 1977, è un metodo di cifratura asimmetrica e basa i suoi principi su funzioni che fanno uso di numeri primi. RSA è una cifratura a blocchi, il **plaintext** e il **ciphertext** sono interi con dimensione compresa tra **0** e **n-1**, per un dato valore n comunemente dell'ordine di 1024 bit. Todo l'algoritmo e la sua sicurezza si basa sulla difficoltà di fattorizzare numeri grandi, ovvero: Si devono trovare due numeri primi (molto grandi) che moltiplicati generano un certo numero.

Quando si ha a che fare con numeri primi piccoli trovarli è molto semplice, la complessità cresce esponenzialmente all'aumentare della dimensione dei fattori. La ricerca dei numeri primi viene solitamente fatta con la tecnica di **Eratostene**, è un algoritmo impegnativo per complessità, pertanto ci si chiede se esiste un'alternativa più efficiente e la risposta è assolutamente NO!! (Esistono però algoritmi che con una certa approssimazione mi dicono se il numero potrebbe essere primo o meno)

Un altro elemento chiave per la sicurezza è l'**esponenziazione**, RSA fa utilizzo di esponenti nel campo di Galois.

Quindi i concetti di primalità ed esponenziazione rendono molto difficile la rottura di RSA.

### Cifratura

Il plaintext in RSA viene crittografato a blocchi, ciascun blocco è un valore binario minore di un certo numero n. Quindi, in pratica, il blocco deve avere una dimensione di k bit tale che :

$$k \leq \log_2 n$$

$$2^k \leq n \leq 2^{k+1}$$

Significa che il numero n deve essere codificato da esattamente k bit, ne meno ne più

Dopo che il sender ha il blocco da cifrare **M** e la chiave pubblica **PU = {e,n}**, è possibile calcolare il ciphertext C nel seguente modo:

$$C = M^e \text{ mod } n \quad \text{con } M \in [0, n[$$

Come si può notare viene svolta sia un'operazione di esponenziazione che di modularità.

## Decifratura

Dall'altra parte, il receiver conosce un valore **d** che in coppia con **n** identifica la chiave privata **PR = {d,n}**.

Attraverso la chiave sarà possibile decifrare il messaggio **C** e ottenere **M**.

$$M = C^d \bmod n \quad \text{con } M \in [0, n[$$

## Generazione delle chiavi RSA

Concentriamoci adesso sul come vengono generate le chiavi, ecco i passaggi:

- Ogni utente genera due numeri primi grandi che chiameremo **p** e **q**
- Si genera il numero **n** che verrà utilizzato in modulo come  $n = p * q$
- Noto **n** è molto difficile calcolare **p** e **q**, mentre noti **p** e **q** è molto facile calcolare **n**.
- I valori **e** e **d**, generati a partire da **p** e **q** sono inversi moltiplicativi di **mod phi(n)**, ovvero:

$$ed \bmod \phi(n) = 1$$

- Esprimendo **d** in funzione di **e** abbiamo la possibilità di scegliere **e** e calcolare **d** in funzione di esso.

$$ed \bmod \phi(n) = 1 \Rightarrow ed \equiv 1 \bmod \phi(n) \Rightarrow d \equiv e^{-1} \bmod \phi(n)$$

- Quindi scegliamo una chiave random e imponendo le seguenti condizioni:

- **e** deve essere:  $1 < e < \phi(n)$
- Massimo comune divisore tra **e** e il quoziente di eulero **phi(n)** deve essere uguale ad 1:  $\text{gdc}(e, \phi(n)) = 1$

- Possiamo quindi calcolare **d** con la formula trovata, imponendo però che:  $0 \leq d \leq n$

A questo punto abbiamo calcolato **n**, **e**, **d**. Avremo:

- La chiave pubblica **PU = {e,n}** viene resa pubblica
- La chiave privata **PR = {d,n}** viene mantenuta segreta

## Inefficienza di RSA

In RSA la crittografia e decrittografia richiedono esponenziazione e moltiplicazione di numeri molto grandi, di base sappiamo che la moltiplicazione è tra le operazioni più onerose per un compilatore (dopo la divisione), mentre l'esponenziazione viene tradotto in cicli di moltiplicazioni, quindi molto inefficiente. Esistono degli algoritmi per diminuire la complessità di queste operazioni (come l'algoritmo Square and Multiply).

Inoltre risulta non banale la generazione della chiave pubblica e privata, vengono impiegati algoritmi più efficienti per migliorare questo aspetto, una soluzione tra questi è il **Teorema Cinese del Resto**.

## Attacchi ad RSA

Per attaccare RSA vengono solitamente adottati i seguenti approcci:

- **Attacchi a brute-force**: Viene effettuata una ricerca della chiave attraverso la forza bruta
- **Attacchi matematici**: Si cerca di fattorizzare **n** o di calcolare **phi(n)**
- **Attacchi a tempo**: L'attacco più pericoloso, vista la sua natura matematica, si cerca di capire il plaintext o la chiave in base ai tempi necessari per criptare, infatti operare su un **1** richiede tempi differenti rispetto che ad operare su uno **0**.
- **Attacchi a testo cifrato scelto**: Si monitorano e cercano alcuni pattern ripetitivi che permettono di avere informazioni sulla chiave.

## Algoritmo Diffie-Hellman

Lo schema di Diffie-Hellman non è un algoritmo di cifratura, esso viene utilizzato per scambiare in modo sicuro una chiave che deve essere mantenuta segreta.

Questo algoritmo prevede la presenza di due numeri pubblicamente noti:

- Un numero primo  $q$
- Un intero  $a$  che è radice primitiva di  $q$ :
  - Se le potenze di  $a$  calcolate in modulo  $q$  generano tutti gli interi da 1 a  $q-1$

Si supponga che gli utenti **A** e **B** vogliano scambiarsi una chiave:

- L'utente **A** seleziona un intero casuale  $X_A < q$  e calcola:  $Y_A = a^{X_A} \text{ mod } q$
- L'utente **B** seleziona un intero casuale  $X_B < q$  e calcola:  $Y_B = a^{X_B} \text{ mod } q$
- Entrambi mantengono privati i valori  $X$  e comunicano il valore  $Y$  alla controparte.
- L'utente A calcola la chiave  $K = (Y_B)^{X_A} \text{ mod } q$
- L'utente B calcola la chiave  $K = (Y_A)^{X_B} \text{ mod } q$

In questo modo, un attaccante non potrà sapere la chiave  $K$ , infatti i valori pubblici saranno  $Y_A$ ,  $Y_B$ ,  $a$  e  $q$ . L'unico modo per ottenere la chiave sarà calcolare  $X_A$  o  $X_B$  tramite un algoritmo discreto, questa è un'operazione di altissima complessità (per valori grandi è ritenuta addirittura impossibile).

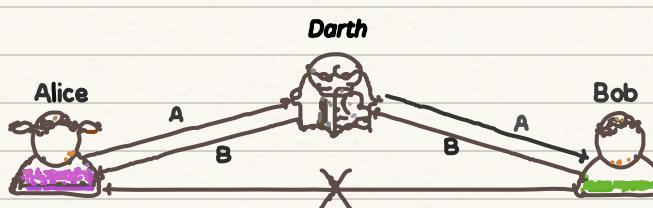
## Attacco Man-in-the-middle

**Diffie-Hellman**, ma anche **RSA**, sono vulnerabili all'attacco **Man In The Middle MITM**, questo avviene perché in questi algoritmi non viene gestita l'autenticazione, ma si risolve tutto attuando delle procedure e protocolli di auth che assicurano l'identità di un host remoto.

Si supponga che **Alice** e **Bob** vogliano scambiarsi le chiavi (tramite Diffie Hellman), al canale ha accesso un terzo attore malevolo che chiameremo **Darth**. L'attacco procede come segue:

- **D** si prepara all'attacco generando due **chiavi private casuali**  $X_{d1}$  e  $X_{d2}$  e quindi calcolando le corrispondenti **chiavi pubbliche**  $Y_{d1}$  e  $Y_{d2}$
- **A** trasmette  $Y_A$  a **B**
- **D** intercetta il messaggio, si conserva  $Y_A$  e trasmette  $Y_{d1}$  a **B**
- **B** trasmette  $Y_B$  a **A**
- **D** intercetta il messaggio, si conserva  $Y_B$  e trasmette  $Y_{d2}$  a **A**
- **B** crede di aver ricevuto  $Y_A$  (ma ha ricevuto  $Y_{d1}$ ), calcola la chiave  $k_1$  con  $Y_{d1}$  e  $X_B$
- **A** crede di aver ricevuto  $Y_B$  (ma ha ricevuto  $Y_{d2}$ ), calcola la chiave  $k_2$  con  $Y_{d2}$  e  $X_A$
- **D** è in grado di calcolare  $k_1$  e  $k_2$  essendo a conoscenza di  $Y_A$ ,  $Y_B$ ,  $X_{d1}$ ,  $X_{d2}$

Da questo momento in poi utilizzando le chiavi **D** potrà intercettare e ritrasmettere tutto il traffico, risulterà totalmente trasparente ad **A** e **B**.



## ElGamal

È uno schema a chiave pubblica strettamente correlato alla tecnica di **Diffie-Hellman**, viene utilizzato in una serie di standard tra cui lo standard di firma digitale **DSS** e lo standard di posta elettronica **S/MIME**.

Vengono definiti a livello globale **q** ed **a**, con radice primitiva di q. L'utente A genera una coppia di chiavi privata e pubblica:

- Sceglie la chiave privata:  $1 < x_A < q-1$
- Calcola la chiave pubblica:  $y_A = a^{x_A} \text{ mod } q$

L'utente **B** il quale ha accesso alla chiave pubblica di **A** può criptare il messaggio nel seguente modo:

- Rappresenta il messaggio come un numero **M < q-1**. Ogni blocco da cifrare deve essere minore di q
- Sceglie un numero intero random **k < q-1**
- Viene calcolata una chiave **one-time**  $K = (Y_A)^k \text{ mod } q$
- **M** viene criptato come la coppia di valori **{C1,C2}** dove:  $C1 = a^k \text{ mod } q$        $C2 = KM \text{ mod } q$

Successivamente l'utente **A** recupera prima la chiave e successivamente decifra il messaggio:

- Recupera la chiave:  $K = (C_1)^{x_A} \text{ mod } q$
- Decifra il messaggio:  $M = (C_2 K^{-1}) \text{ mod } q$

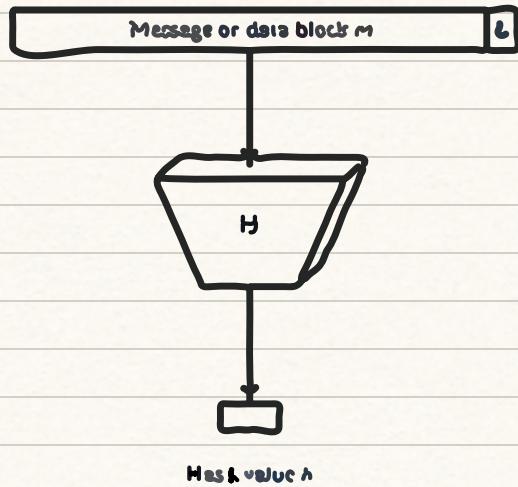
Chiunque abbia accesso alla chiave pubblica di A può cifrare un messaggio, inoltre si noti che K funziona come una chiave una tantum, inoltre se un messaggio deve essere suddiviso in più blocchi saranno necessarie più chiavi.

## Hash Functions e Autenticazione dei Messaggi

Un valore hash viene generato da una funzione **H** con la seguente forma:

$$h = H(M)$$

dove **h** è l'uscita, **H** la funzione di hashing ed **M** il messaggio in input. L'elemento chiave di una funzione hash è che per un messaggio di **qualsiasi lunghezza in input** si ottiene, applicando **H**, una **uscita di lunghezza nota**, cioè la lunghezza dell'uscita ha un valore fisso indipendentemente dalla dimensione dell' ingresso.



Quindi una funzione hash associa a messaggi di lunghezza arbitraria una stringa di lunghezza fissa. È intuitivo considerare che la funzione **H** comporta **perdita di informazione**. L'algoritmo è costruito in maniera tale che una modifica a qualsiasi bit in **M** risulta con alta probabilità in una modifica al codice hash in output.

I requisiti di una funzione di hash sono:

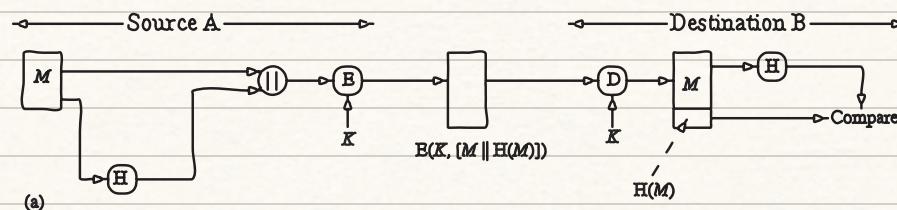
- **H** può essere applicata a un blocco di dati di dimensione qualsiasi
- **H** produce un output di lunghezza fissa
- **H(x)** è relativamente facile da calcolare, consentendo sia un implementazione sia **Hardware che Software**
- Per un determinato valore di **h** è computazionalmente impossibile trovare un valore **x** tale che **H(x)=h**, questa si chiama proprietà di **monodirezionalità**.
- Per un determinato blocco **x** è computazionalmente impossibile trovare un valore **y diverso da x** tale che **H(x) = H(y)**, questa proprietà si chiama **resistenza debole alle collisioni**.
- È computazionalmente impossibile trovare una coppia **(x,y) diversi** tra loro ma tale che **H(x) = H(y)**, questa proprietà è solitamente chiamata **resistenza forte alle collisioni**.

Una **semplice** funzione di **hash** è l'utilizzo dell'operatore **XOR**, questa tecnica risulta molto banale ma anche molto poco sicura poiché produce un semplice blocco di parità . L'idea è quella di dividere l'input in blocchi da **m** bit ed effettuare l'**XOR** bit a bit tra i vari blocchi.

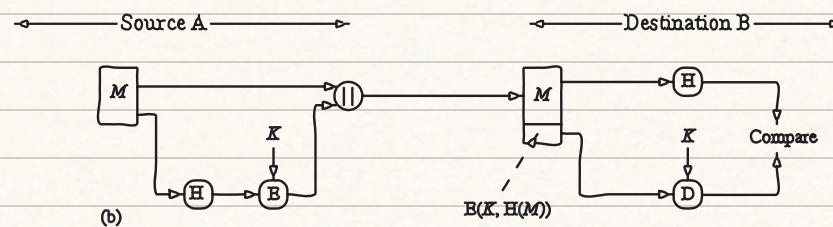
# Modalità operative per le funzioni di Hash

Attraverso le funzioni di hash, in combinazione con la crittografia, è possibile ottenere in uno scambio di messaggi le proprietà di **autenticazione, integrità e segretezza**, di seguito quattro possibili approcci:

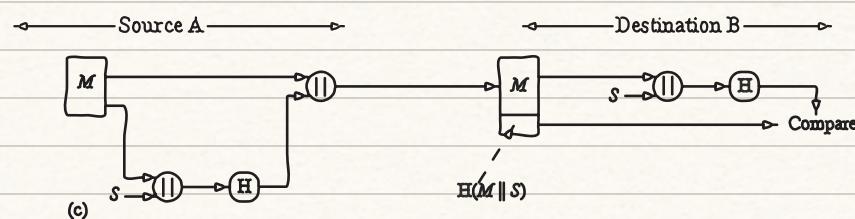
- **Crittografia del messaggio e del codice hash:** Il messaggio e il codice hash di questo vengono concatenati e crittografati. Poichè solo **A** e **B** condividono la chiave segreta il messaggio deve provenire per forza da **A**, per questo abbiamo sia **autenticazione** che **integrità**, oltre che **segretezza**.



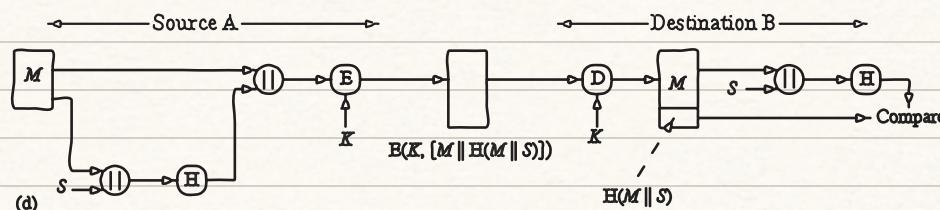
- **Crittografia del codice hash e chiave segreta condivisa:** Viene crittografato solo il codice hash utilizzando la crittografia simmetrica. Questo riduce il carico di elaborazione per le applicazioni dove non è necessario garantire segretezza. È importante notare che la combinazione tra hash e crittografia produce una funzione che genera un codice **MAC**. Il senso di crittografare l'hash è che da solo non proteggerebbe i messaggi da modifiche, è vero che l'hash ci permette di verificare l'integrità del messaggio, ma non ci assicura l'autenticazione, poichè senza cifratura dell'hash chiunque potrebbe intercettare il messaggio, modificarlo e ricalcolare l'hash, applicando la crittografia si ottiene l'autenticazione poichè solo **A** potrà aver generato quell'hash in quanto protetto.



- **Crittografia del codice hash del messaggio più il valore segreto:** La tecnica presuppone che le due parti comunicanti condividano un valore segreto **S**. Si calcola il valore hash sulla concatenazione di **M** e **S**, successivamente si invia la coppia Hash e messaggio. In questo caso non si utilizza crittografia, poichè l'elemento di sicurezza che ci permette di avere l'autenticazione è il valore segreto **S**, qualora un attaccante volesse modificare il messaggio, per farlo avrebbe necessità di conoscere **S**. Poiché solo **A** conosce **S** abbiamo integrità e autenticazione.



- **Crittografia dei risultati di (c):** Per aggiungere la proprietà di segretezza all'approccio (c) si può cifrare tutto l'output (hash + messaggio). Garantisce integrità, segretezza e autenticazione (per 2 motivi, primo per la presenza del parametro segreto **S**, secondo per l'utilizzo della chiave di cifratura).



## Hash per firma digitale

Un'altra importante applicazione, simile all'applicazione di autenticazione dei messaggi, è la firma digitale. Il funzionamento è simile a quello del MAC. Nel caso della firma il valore hash di un messaggio viene crittografato con la chiave privata di un utente. Chiunque conosca la chiave pubblica dell'utente può verificare l'integrità del messaggio associato alla firma digitale. In questo caso, un utente malintenzionato che desidera modificare il messaggio deve conoscere la chiave privata del mittente. Ecco come viene utilizzato il codice hash per fornire una firma digitale:

- **Crittografia del codice hash con chiave privata:** Viene crittografato solo il codice hash utilizzando la chiave privata del mittente, viene assicurata l'autenticazione e viene garantita la firma digitale poiché solo il mittente può avere prodotto il codice hash crittografato.
- **Crittografia dei risultati di (a) con chiave simmetrica:** Se si richiede la segretezza tutto l'output di (a) viene crittografato con una chiave simmetrica segreta posseduta tra mittente e destinatario.

## Attacchi alle funzioni Hash

Come per gli algoritmi di crittografia possiamo distinguere due macro-famiglie di attacchi:

- **A forza bruta:** Non dipende dall'algoritmo specifico ma solo dalla lunghezza in bit del valore di hash, più quest'ultimo è lungo meno è probabile un attacco di questo genere. Ovviamente l'obiettivo dell'attaccante è trovare una collisione tra due valori in input, in modo che generino un hash comune.
- **A crittoanalisi:** Basa la sua efficacia sui punti deboli dell'algoritmo. In genere per questo tipo di attacchi è richiesto uno sforzo e uno studio maggiore rispetto alla forza bruta.

Un attacco importante è **l'attacco a compleanno**, esso basa il suo fondamento sul paradosso del compleanno.

Sia  $n = 2^m$  il numero di possibili output della funzione di hash, sia  $H(Y)$  il valore di hash dato, dato un altro  $X$ , la probabilità che  $H(X) = H(Y)$  è di  $1/n$ .

Ovvero si ha che qualunque valore esistente verrà mappato in un dominio con numero di possibili valori ristretto a  $n$ , quindi la probabilità di beccare quel preciso valore di hash sarà  $1/n$ , la probabilità di non trovare la collisione sarà di  $1-1/n$ .

Scelti  $k$  valori casuali di  $X$ , la probabilità di non beccare nessuna corrispondenza su questi  $k$  campioni sarà di  $(1-1/n)^k$ , si può dimostrare che per  $k$  non molto grandi si ha la probabilità di trovare una corrispondenza pari all'incirca a  $k/n$ .

Si ha infatti che per ottenere una probabilità del 50% di beccare una collisione bastino  $\sqrt{n}$ .

La fattibilità di questo attacco è mitigabile da una buona lunghezza del codice hash.

# SHA - Secure Hash Algoritm

Negli ultimi anni, la funzione hash più utilizzata è stata SHA, si hanno diverse versioni di SHA:

- **SHA-0:** Prima versione
- **SHA-1:** Versione rivista
- **SHA-2:** Versione con un digest molto più lungo 256/224 o 512/384, contro i 160 bit dei precedenti.

Quindi possiamo identificare due versioni di SHA-2:

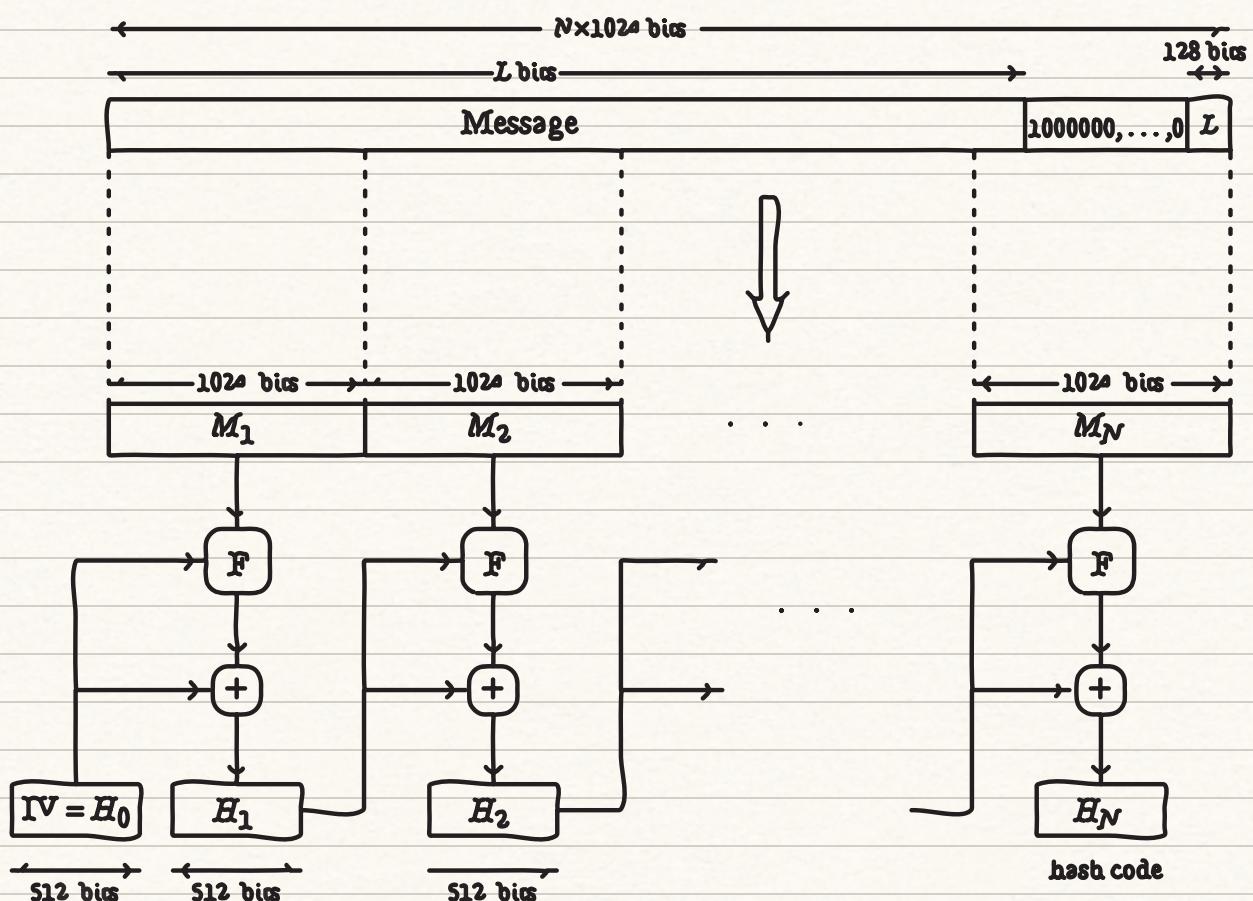
- **SHA-256/224**
- **SHA-512/384**

Di seguito come è implementato **SHA-512** (SHA-256 risulta uguale a meno di una differenza nelle dimensione dei blocchi):

L'algoritmo prende in ingresso un messaggio di lunghezza arbitraria, e produce un digest in output di 512 bit.

• L'input viene esteso in modo che la sua lunghezza sia congruente a 896 mod 1024, questo perchè dopo l'aggiunta del padding l'ultimo blocco deve avere dimensione pari a 896 visto che dovrà essere aggiunta anche una stringa di bit di 128 bit che indica la lunghezza L del messaggio (Per questo motivo la lunghezza massima di un messaggio può essere di  $2^{128}$ , numero comunque gigantesco). Il padding è un bit ad 1 susseguito da una lista di 0 a riempimento, successivamente abbiamo la lunghezza (originale) espressa da 128 bit.

- Viene suddiviso in blocchi da 1024 bit.
- Viene inizializzato un buffer H che fungerà da stato intermedio tra i round sui blocchi e da output. Questo buffer è formato da 8 registri da 64 bit (A,B,C,D,E,F,G,H), i dati sono memorizzati in formato big-endian.



$+ = \text{word-by-word addition mod } 2^{64}$

- Ogni fase prevede un output:

$$H_i = H_{i-1} + F(M_i, H_{i-1})$$

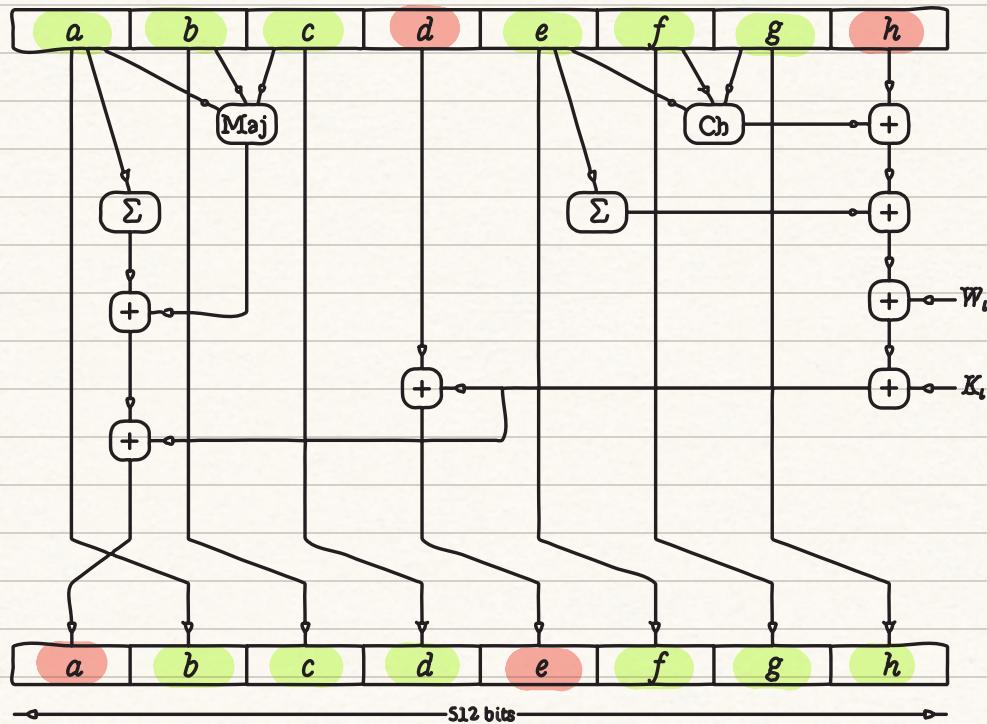
- Abbiamo quindi:

- Un'operazione di addizione, questa non è una normale somma, ma una somma a livello di registro in modulo 64. Per esempio:

$$A_i = (A_0 - A_{i-1}) \bmod 2^64$$

Vale lo stesso ragionamento per gli altri buffer B,C,D...

- Una funzione F che prende in ingresso sia il blocco corrente che l'output del blocco precedente. Questa funzione consiste in un ciclo di 80 step, il generico step può essere descritto dal seguente schematico:



Primo aspetto da notare è che per i buffer evidenziati in verde avviene un semplice shift a destra, quindi  $a \rightarrow b$ ,  $b \rightarrow c \dots$

Per quanto riguarda *a*, *e* in output, essi sono calcolati a partire dai buffer dello step precedente e dagli ulteriori ingressi *Wt* e *Kt*.

Di seguito il dettaglio dei vari blocchi operazionali:

$$\Sigma(a) = \text{ROTR}(a, 28) \oplus \text{ROTR}(a, 34) \oplus \text{ROTR}(a, 39)$$

$$\Sigma(e) = \text{ROTR}(e, 14) \oplus \text{ROTR}(e, 18) \oplus \text{ROTR}(e, 41)$$

$$\text{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$$

$$\text{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$$

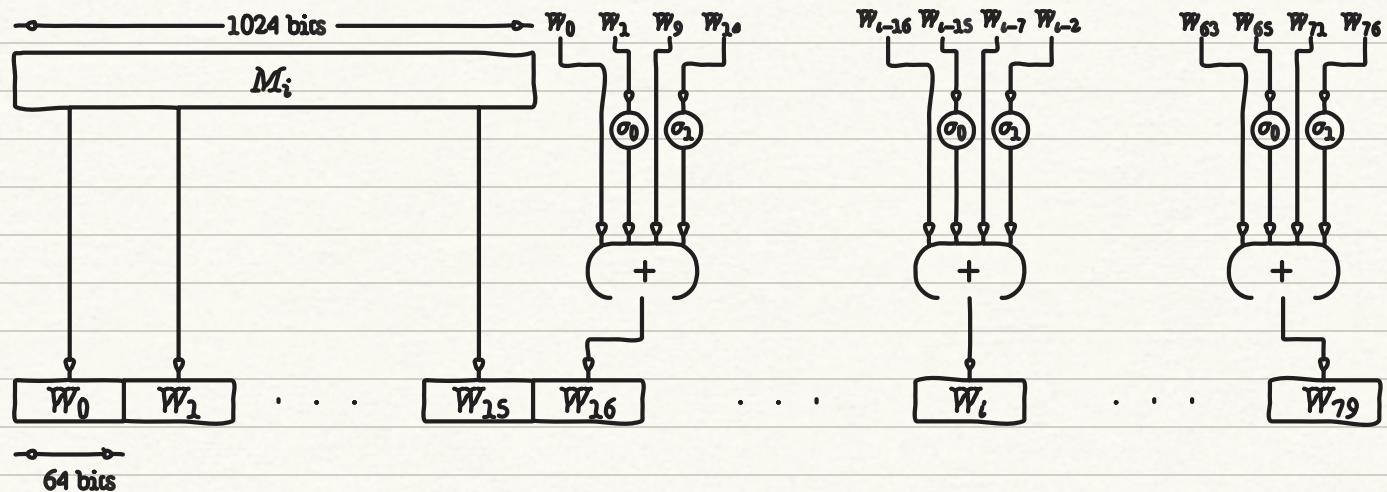
$$+ = \text{somma modulo } 2^{64}$$

$$Kt = \text{costante additiva a 64 bit}$$

$$Wt = \text{word a 64 bit derivante dal blocco } M_i \text{ corrente di 1024 bit}$$

Se Kt è una costante, Wt no, ecco come viene calcolato:

- Avendo 80 step dobbiamo ottenere 80 word da 64bit, le prime 16 word vengono derivate dal messaggio  $M_i$  corrente, le successive [W15, W16... W79] sono calcolate in funzione delle prime 16.



Abbiamo che il generico  $W_i$  sarà dato dalla somma dei precedenti  $W_j$  con due di questi sottoposti a due funzioni che fanno utilizzo di operatori ROTR e SHR.

Più precisamente:

$$W_i = W_{i-16} + \sigma_0(W_{i-15}) + W_{i-7} + \sigma_1(W_{i-2})$$

$$\sigma_0(x) = \text{ROTR}(x, 1) \text{ XOR } \text{ROTR}(x, 8) \text{ XOR } \text{SHR}(x, 7)$$

$$\sigma_1(x) = \text{ROTR}(x, 19) \text{ XOR } \text{ROTR}(x, 61) \text{ XOR } \text{SHR}(x, 6)$$

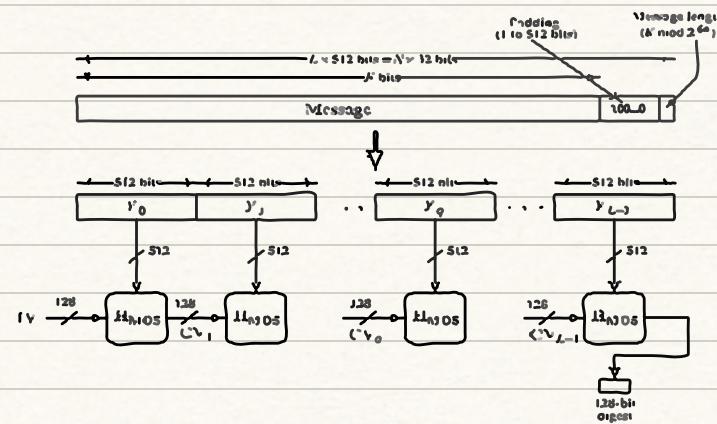
Con ROTR operazione di shift circolare a destra, SHR operazione di shift a destra con riempimento di zeri.

## MD5

MD5 nasce per sostituire i percussori MD2 e MD4. MD5 produce un codice hash di 128 bit prendendo un input variabile gestito in blocchi da 512 bit. Fino ad oggi è stato uno degli algoritmi più utilizzati, ultimamente sono sorti dubbi sulla sua vulnerabilità agli attacchi a brute force.

La lunghezza del messaggio deve essere congruente  $448 \bmod 512$ , alla fine dell'ultimo blocco viene aggiunta una stringa di 64 bit contenente la lunghezza mod 64 del messaggio originale.

In MD5 si fa utilizzo di 4 buffer (A,B,C,D) da 32 bit, in base al numero di blocchi da 512 bit derivati dall'input si andranno a svolgere un certo numero di fasi, ogni fase prende in ingresso il contenuto dei buffer (output della fase precedente) ed il blocco corrente, la prima fase prende in ingresso il buffer che conterrà un IV. L'output dell'ultima fase sarà il digest.



La Funzione HM5 che viene eseguita ad ogni fase consiste in:

- 16 Step del tipo:

$$a = b + ((a + g(b, c, d) + X[k] + T[i]) \ll\ll s)$$

$$k = 1, \dots, 16$$

$g(b, c, d)$  è una funzione non lineare diversa ad ogni fase

$T[i]$  è una costante derivata dalla funzione seno

Ad ogni fase viene aggiornato un solo buffer, successivamente avviene uno shift.

