

# Crittografia

December 10, 2021

Mercoledì: 13 - 14.30, Venerdì: 13 - 14.30

Modalità d'esame: orale - 16233

Programma:

- Accenni storici
- Cifrario simmetrico con una chiave
- Cifrario asimmetrico con due chiavi, una per aprire una per chiudere
- Funzioni hash

In un cifrario simmetrico, la chiave di encoding è uguale alla chiave di decoding; nel cifrario asimmetrico le chiavi di encoding e decoding sono differenti.

Dato un mittente A, ed un ricevente B, il mittente A cifra il plain text in cypher text utilizzando le chiavi di encoding. Il cypher text viene trasmesso nel canale di comunicazione, che verrà poi passato nel processo di decoding da parte di B, per ricavare nuovamente il plain text iniziale.

Scenari di attacco:

- 1) Cyphertext only
- 2) Known plaintext
- 3) Chosen plaintext
- 4) Chosen cyphertext

- I cifrari simmetrici vengono utilizzate per cifrare grosse quantità di dati (eg. *testo, des e aes*)
- I cifrari asimmetrici vengono utilizzati per cifrare piccole quantità di dati (eg. *pw, rsa, elgamal*)
- Le tecniche steganografiche sono tecniche di offuscazione del messaggio

Principio di Kerkhoff: un cifrario è forte sse l'attaccante conosce tutto tranne che la chiave.  
*Brutalmente: se un cifrario rimane pubblico per 30 anni, ma non viene bucato, vuol dire*

*che è resiliente.*

4 proprietà delle funzioni crittografiche:

- Confidenzialità: ottenuta attraverso l'operazione di cifratura prima della spedizione
- Integrità dei dati: applicando una funzione crittografica, se il risultato è medesimo nel caso del mittente e del destinatario la cifratura è considerata integro (hashing functions)
- Autenticazione: garantire che solo gli interessati siano in grado di accedere alle informazioni
- Non ripudio: l'autore di una dichiarazione non potrà negare la paternità e la validità della dichiarazione stessa

Cifrari storici:

### **Cifrario di Cesare**

Basato sulla traslazione delle lettere, dato un k trasliamo l'alfabeto di k valori

### **Quadrato di Polibio**

Viene creato una matrice, si inseriscono le lettere all'interno della matrice e per ogni corrispondenza si prende la coppia colonna riga. Il cyphertext è grande 2x il plaintext. Il problema di questo cifrario è che ha punti fissi.

Un approccio migliore è passare da espressioni  $y = x + k$  a  $y = \alpha x + \beta$  con  $\alpha \neq 0, \beta \neq 0$   
Per ricavare un valore dobbiamo fare  $(y - \beta)/\alpha = x$ .

E' inoltre importante che  $\text{MCD}(\alpha, 26) = 1$  ovvero che  $\alpha$  sia invertibile, e che  $\alpha$  e 26 siano coprimi; il numero totale di chiavi possibili è  $12 * 26$ .

Come attacchiamo i crittosistemi affini? Andiamo ad analizzare le frequenze.

Dato il teorema di Kerkhoff l'attaccante conosce tutte le informazioni riguardo il cifrario, ma non la chiave. In italiano le lettere che compaiono di più sono E (che compare in media 11,79 %), la A (11,74%), la I (11,28 %) etc.

Più il testo è lungo, più possiamo essere certi della corrispondenza (maggiori il campione, migliore l'approssimazione). Il problema del cifrario affine è quindi che data una lettera, la corrispondenza cifrata è sempre la stessa.

### **Quadrato di Vigenère**

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

codifica della lettera R con chiave V

Viene introdotta una chiave (parola), concordata tra mittente e destinatario. La chiave è ripetuta  $n$  volte fino a ricavare una corrispondenza 1-1 con il messaggio da cifrare. In questo modo le stesse lettere hanno potenzialmente più corrispondenze. Per ricavare le singole lettere andiamo a ricavare la corrispondenza con colonna  $Cyphertext_i = (Plaintext_i, Chiave_i)$

attackatdawn

LEMONLEMONLE

LXFOPVEFRNHR

Come attacchiamo questo cifrario? Il metodo più famoso è il metodo di Kasiski Attraverso l'*analisi di Kasiski* possiamo sfruttare il fatto che alcune parole, per probabilità, vengono cifrate con le stesse lettere. Questo ci permette di trovare gruppi ripetuti nel *ciphertext*.

Key:	<b>ABCDABCDABCDABCD<b>ABCD<b>ABCDABCD</b>ABCD</b>ABCD<b>ABCD</b>ABCD</b> ABCD
Plaintext:	<b>cryptoisshortfor<b>crypt<b>ography</b></b></b>
Ciphertext:	<b>CSASTPKVSIOUTGQU<b>CSAST<b>PIUAQJB</b></b></b>

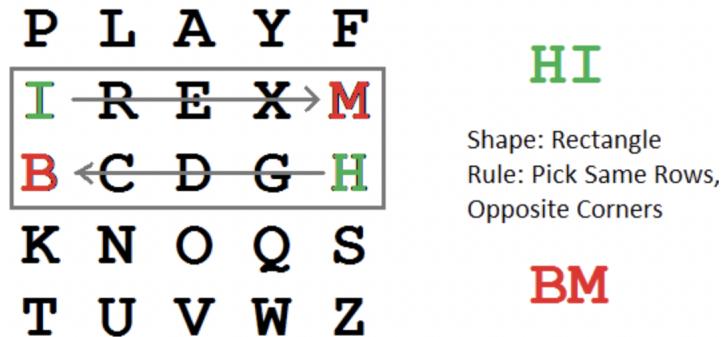
Una volta identificati questi gruppetti, li raccogliamo e possiamo essere certi che questi sottoinsieme saranno un divisore della chiave. Possiamo quindi effettuare un'analisi di frequenza su queste, più lunga il testo, migliore sarà la precisione dell'analisi di frequenza.

Una variante è il cifrario di Cardano, in cui la chiave è inserita prima del critttesto, tecnica molto vulnerabile in quanto è l'attaccante deve solo indovinare la lunghezza.

**Playfair cipher** Il cifrario è basato su una matrice 5x5 (essendo l'alfabeto di 26, una lettera ha una corrispondenza con un'altra, solitamente i con j) in cui vengono inserite le lettere. La chiave è scelta, può essere una parola. Le lettere sono inserite saltando i doppiomi, mettendo in testa la chiave.

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

Il cyphertext è creato a coppia di lettere, si forma un rettangolo/quadrato utilizzando i vertici opposti delle due lettere scelte. I vertici opposti rappresentano le lettere cifrate.



I vertici opposti del rettangolo delle lettere **HI** sono **BM**, la codifica è **BM**

### Hill cipher

Viene definita una matrice  $k$  di dimensione  $n * n$ , le parole vengono cifrate a coppie di  $n$  (matrice 2 x 2, cifrate 2 alla volta, matrice 3 x 3, cifrate 3 alla volta).

Un esempio può essere quella di prendere 2 lettere alla volta, codificate in numero, valore che è poi utilizzato per fare la moltiplicazione riga per colonna, il valore è passato a

$\text{mod}(26)$  per ricavare una lettera. Per decodificare facciamo la moltiplicazione per l'inverso della matrice chiave.

Non tutte le matrici chiavi sono ammissibili, la matrice deve esser invertibile per permettere il decifraggio da parte del destinatario. In particolare  $MCD(\Delta, 26) = 1$ .

**ADFGX cipher** Utilizzata dai generali tedeschi durante la prima guerra mondiale, viene creata una matrice  $5 \times 5$  in cui vengono inserite le lettere arbitrariamente.

	<b>A</b>	<b>D</b>	<b>F</b>	<b>G</b>	<b>X</b>
<b>A</b>	b	t	a	l	p
<b>D</b>	d	h	o	z	k
<b>F</b>	q	f	v	s	n
<b>G</b>	g	i/j	c	u	x
<b>X</b>	m	r	e	w	y

a	t	t	a	c	k	a	t	o	n	c	e
AF	AD	AD	AF	GF	DX	AF	AD	DF	FX	GF	XF

Per ricavare il critttesto prendiamo le coppie degli indici (riga, colonna), ogni lettera quindi è data da una coppia di indici.

Viene poi riordinato il critttesto secondo una chiave di lunghezza  $k$ . Il critttesto viene suddiviso in righe di  $k$  lunghezza. Una volta messe in pila di lunghezza  $k$ , le colonne vengono riordinate secondo l'ordine alfabetico delle lettere chiave.

<b>C A R G O</b>						<b>A C G O R</b>					
<hr/>						<hr/>					
A F A D A						F A D A A					
D A F G F						A D G F F					
D X A F A						X D F A A					
D D F F X						D D F X F					
G F X F						F G F X					

FAXDF ADDDG DGFFF AFAXX AFAF

La codifica di CARGO

Con questo abbiamo finito con i cifrari storici.

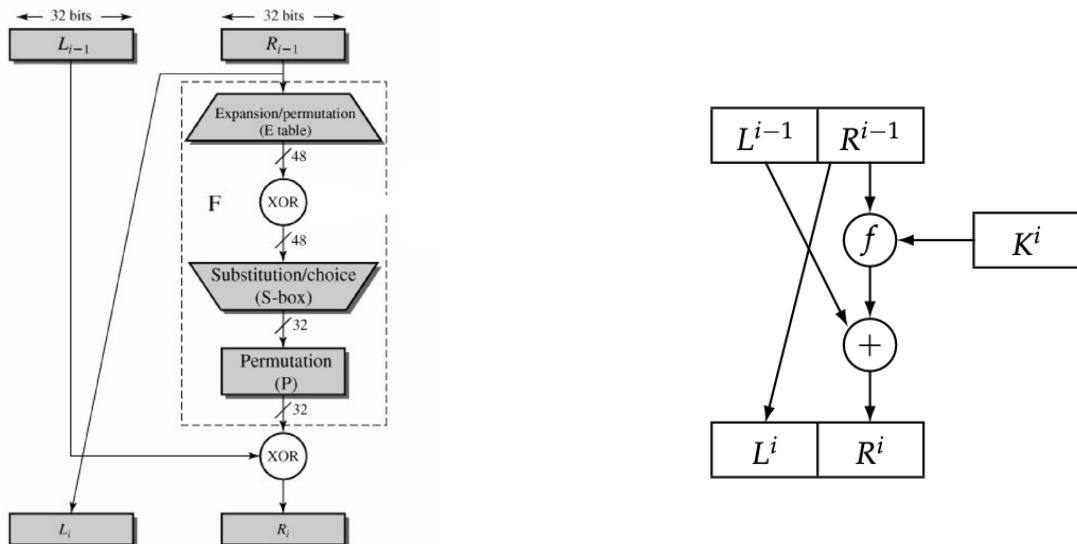
## NIST / NBS

Il NIST è un'associazione governativa americana che si occupa di emettere bandi per richiedere la creazione di cifrari per il pubblico. In questo momento i bandi aperti sono: "Post-quantum competition" del 2018 ed il "Lightweight cryptography" per sistemi leggeri. Nel 1970 manda un bando per creare un cifrario simmetrico utile alle grandi aziende americane per cifrare i dati sicuri, in particolare viene sviluppato DES, un cifrario simmetrico.

## DES

E' un cifrario a 16 round, basato su blocchi di 64, dotato di  $64 - 8$  (*bit di controllo*) = 56 bit di chiave che produce un ciphertext di lunghezza 64. Un attacco di brute-forcing svolge al massimo  $2^{56}$  tentativi, che già nel 1970 porta molti ad obiettare.

Il plaintext è permutato per 16 volte, l'inverso della permutazione è poi applicato sul plaintext permutato.



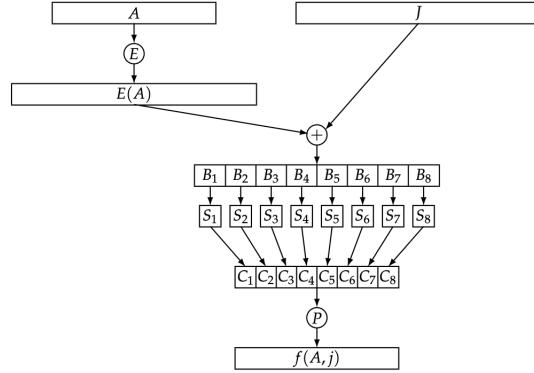
F è la funzione di Feistel

## Feistel function

*Ma dove entra in gioco la chiave?* La chiave è fornita alla funzione di Feistel, è generata per ogni dato round di cifratura.

*La funzione Feistel*

Quando parliamo di funzioni di Feistel parliamo di funzioni la cui sicurezza è acquisita effettuando  $n$  round di permutazioni



La funzione  $E$  è rappresentata permutazione di espansione, che aggiunge 8 bit ai 32 forniti in precedenza. Questi 48 bit sono utilizzati per effettuare uno XOR con  $J$  (la chiave del round), che mi permette di ricavare 8 blocchi. Questi 8 blocchi (formati da 6 bit ciascuno) sono permutati nuovamente (all'interno di  $S_k$ ). Le funzioni  $S_k$  eseguono *un'operazione di compressione*: prendono in bit 6 bit ciascuno, ma restituiscono 4 bit, portando l'output a 32 bit. Questi 32 bit rappresentano l'output.

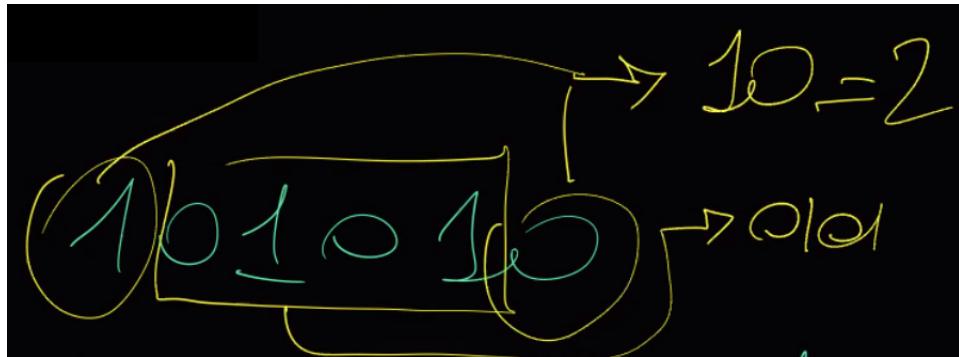
## S-Box

La S-Box è un componente di compressione,  $S_k$  è formato da una matrice ( $4 * 16$ ) sulla quale sono inseriti una permutazione dei valori tra 1 a 16 (fissa).

$S_1$	<table border="1"> <tbody> <tr><td>14</td><td>4</td><td>13</td><td>1</td><td>2</td><td>15</td><td>11</td><td>8</td><td>3</td><td>10</td><td>6</td><td>12</td><td>5</td><td>9</td><td>0</td><td>7</td></tr> <tr><td>0</td><td>15</td><td>7</td><td>4</td><td>14</td><td>2</td><td>13</td><td>1</td><td>10</td><td>6</td><td>12</td><td>11</td><td>9</td><td>5</td><td>3</td><td>8</td></tr> <tr><td>4</td><td>1</td><td>14</td><td>8</td><td>13</td><td>6</td><td>2</td><td>11</td><td>15</td><td>12</td><td>9</td><td>7</td><td>3</td><td>10</td><td>5</td><td>0</td></tr> <tr><td>15</td><td>12</td><td>8</td><td>2</td><td>4</td><td>9</td><td>1</td><td>7</td><td>5</td><td>11</td><td>3</td><td>14</td><td>10</td><td>0</td><td>6</td><td>13</td></tr> </tbody> </table>	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
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																																																		
$S_2$	<table border="1"> <tbody> <tr><td>15</td><td>1</td><td>8</td><td>14</td><td>6</td><td>11</td><td>3</td><td>4</td><td>9</td><td>7</td><td>2</td><td>13</td><td>12</td><td>0</td><td>5</td><td>10</td></tr> <tr><td>3</td><td>13</td><td>4</td><td>7</td><td>15</td><td>2</td><td>8</td><td>14</td><td>12</td><td>0</td><td>1</td><td>10</td><td>6</td><td>9</td><td>11</td><td>5</td></tr> <tr><td>0</td><td>14</td><td>7</td><td>11</td><td>10</td><td>4</td><td>13</td><td>1</td><td>5</td><td>8</td><td>12</td><td>6</td><td>9</td><td>3</td><td>2</td><td>15</td></tr> <tr><td>13</td><td>8</td><td>10</td><td>1</td><td>3</td><td>15</td><td>4</td><td>2</td><td>11</td><td>6</td><td>7</td><td>12</td><td>0</td><td>5</td><td>14</td><td>9</td></tr> </tbody> </table>	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
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																																																		

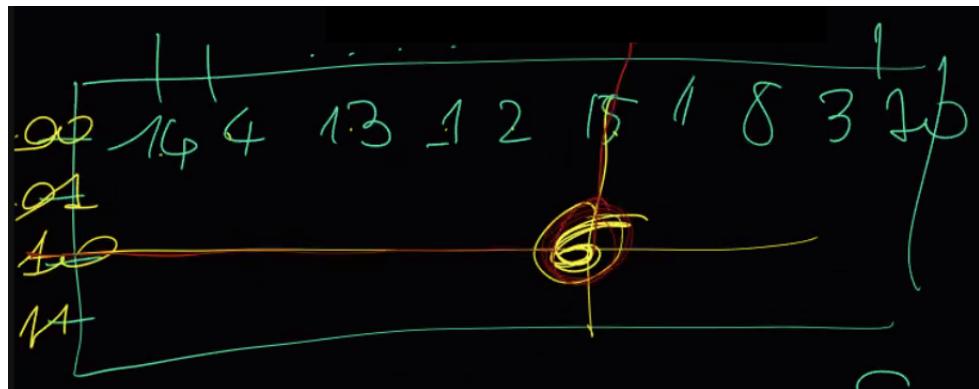
*un esempio di S-boxes*

Dato una sequenza di input di bit di lunghezza 6 (l'input di ogni s-box), il primo ed l'ultimo valore sono utilizzati per indicizzare la riga, ed i 4 valori interni indicizzano la colonna.



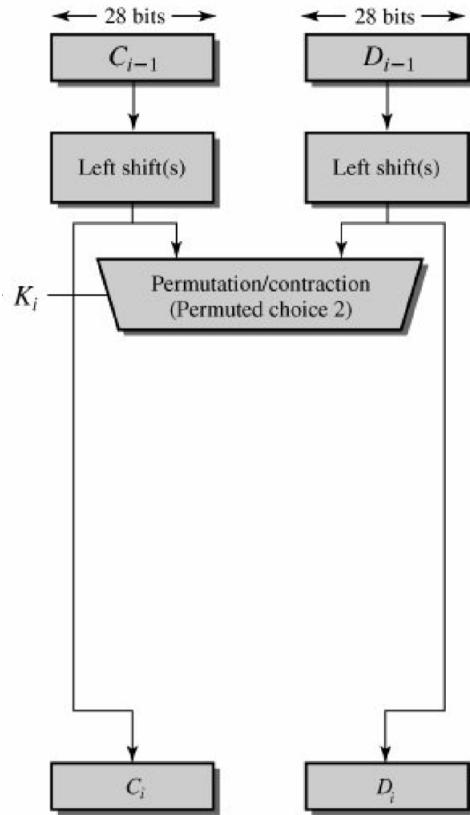
*Si prende la terza e la sesta perché il binario 00 è la prima colonna, noi contiamo da 0*

Il valore ricavato incrociando riga e colonna codificato in binario rappresenta l'output della S-Box. Questo è possibile perché nella matrice vi sono solo valori tra 0 e 15, che sono codificabili in binario su 4 valori.



*il valore di output di questa S-box sarà  $0110_2 = 6_{(10)}$*

## La chiave $K_i$

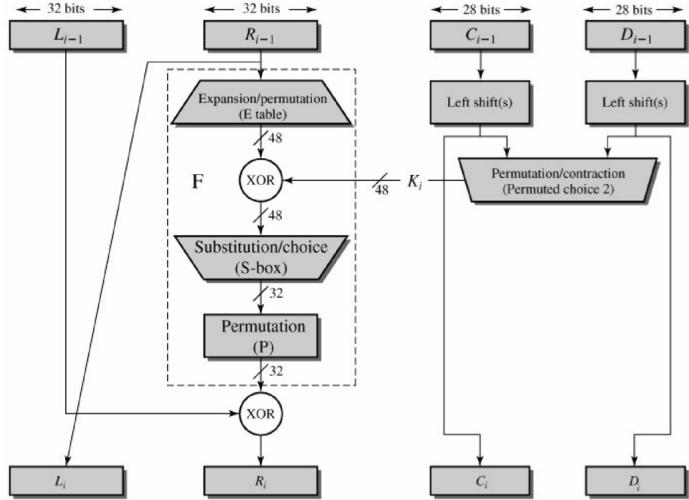


La chiave  $K_i$  è formata da 64 bit, dalla quale vengono rimossi 8 bit di parità. Sui bit rimanenti è effettuata una permutazione. I 56 bit dopo la permutazione sono suddivisi in 2 blocchi da 28, chiamati rispettivamente  $C_0$  e  $D_0$ .  $C_0$  e  $D_0$  sono shiftati a sinistra, a seconda del round.

$$C_i = \text{LS}(C_{i-1})$$

$$D_i = \text{LS}(D_{i-1})$$

Il valori shiftati vengono utilizzati come input per una permutazione/contrazione, passando da 56 a 48. L'output di questa permutazione rappresenta la chiave, e gli input della prossima  $C_i$  e  $D_i$ .



*Overview di DES*

Vi sono dei vincoli riguardo a DES, per esempio:

E' importante che dato un S-Box, l'output di tale S-Box non possa in qualsiasi modo approssimabile.

## S-DES — Simplified DES

I dati:

$plaintext = 12bits$

$L_i = 6bits$

$R_i = 6bits$

$k = 9bits$

Il numero di round è diminuito, in particolare 3 (II, III, IV), tratteremo quindi:

$L_1, R_1 \rightarrow L_2, R_2 = I\ round$

$L_2, R_2 \rightarrow L_3, R_3 = II\ round$

$L_3, R_3 \rightarrow L_4, R_4 = III\ round.$

*La funzione di Feistel semplificata:*

$R_{(i-1)}$  (6 bits) è passato in espansione (8 bits), e fatta passata in XOR con la chiave (8 bits) (che perde un bit), per poi essere inserita come input (di 4 bits) in 2 S-Box di compressione (matrici  $2 * 8$ ).

L'output delle S-boxes sono di 3 bits ciascuna, il risultato è  $f(R_{i-1}, k_i)$ .

*La funzione di espansione  $E(x)$ :*

La funzione di espansione ripete dei bit dell'input. In particolare dato un input:

(1-2-3-4-5-6) (6 bits), l'output sarà (1-2-4-3-4-3-5-6) (8 bits)  
in cui  $k$  rappresenta la posizione  $k_i$

Le S-Box semplificate sono:

$S_1$
$\begin{bmatrix} 101 & 010 & 001 & 110 & 011 & 100 & 111 & \infty \\ \infty & 100 & 110 & 010 & \infty & 111 & 101 & 011 \end{bmatrix}$
$S_2$
$\begin{bmatrix} 1\infty & \infty & 110 & 101 & 111 & \infty & 011 & 010 \\ 101 & 011 & \infty & 111 & 110 & 010 & 001 & 1\infty \end{bmatrix}$

Per ricavare il valore di output è ricavato prendendo: il 1° bit per la riga, e gli altri 3 per la colonna

*La compressione della chiave:*

La chiave è ricavata copiando la chiave a partire dalla posizione 2, ricopiando dall'inizio quando necessario.

$$k = 001001101$$

$$k_2 = 010011010$$

Il plaintext iniziale è:  $L_1R_1 = 000111 - 011011$

Il secondo plaintext è:  $L_1R_1 = 101110 - 011011$

Qual'è  $L_4R_4$ ?

L'operazione di crittoanalisi è eseguibile solo se il numero di round è basso (in particolare 3). Se i round sono 4, siamo sicuri che in mezzo c'è almeno un round di crittografia, se siamo a livello 5 sappiamo che sono almeno 2, e così via. L'attacco non è eseguibile deterministicamente, ma probabilisticamente.

*Le modalità di cifratura ECB, CBC, CFB, OFB, CTR*

Sono le modalità di utilizzo dei cifrari a blocchi, applicabili su qualsiasi cifrario a blocchi:

- ECB, o "electronic codebook mode":

ECB è applicato dividendo il plaintext  $P$  in blocchi di equa grandezza, che vengono poi codificati in cyphertext attraverso una cifrario a scelta. La dimensione del blocco

è dipendente dal cifrario scelto

ECB è ideale quando la quantità di dati da codificare è bassa, se per esempio si vuole trasmettere una chiave DES, ECB è ideale.

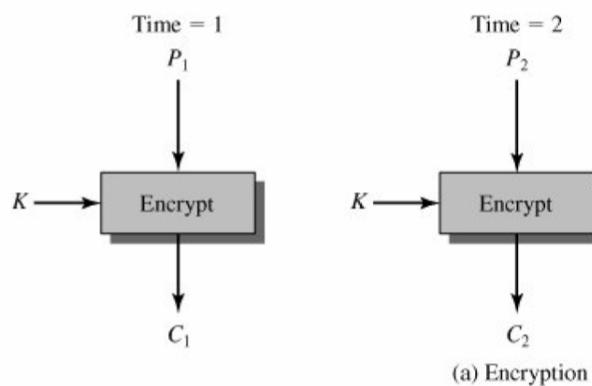
Dato:

$P = [p_1, \dots, p_n]$  l'insieme delle porzioni di grandezza n

$C = [c_1, \dots, c_n]$  l'insieme dei ciphertext di grandezza n

attraverso la funzione  $f(P_i; DES) = C_i$  possiamo codificare il plaintext in ciphertext

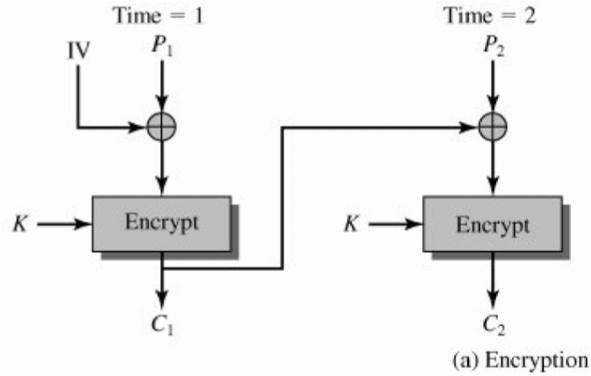
Per ricavare il plaintext utilizziamo la funzione di decryption su ogni blocco di ciphertext.



Il problema con ECB è che se più blocchi di plaintext sono uguali, il ciphertext prodotto è uguale. Questo permette su messaggi molto lunghi di essere attaccati attraverso una crittoanalisi di frequenza

- CBC, o "cipher block chaining mode":

Per mitigare le fallo di sicurezza di ECB, vogliamo introdurre un meccanismo per produrre ciphertext differenti per plaintext diversi. Con CBC introduciamo un round di XOR con una chiave IV (Initialization vector)(pubblica o privata) prima della fase di encryption, per offuscare ulteriormente la procedura. IV è sostituito dal ciphertext del round  $P_i - 1$  dalla fase 2 in poi.

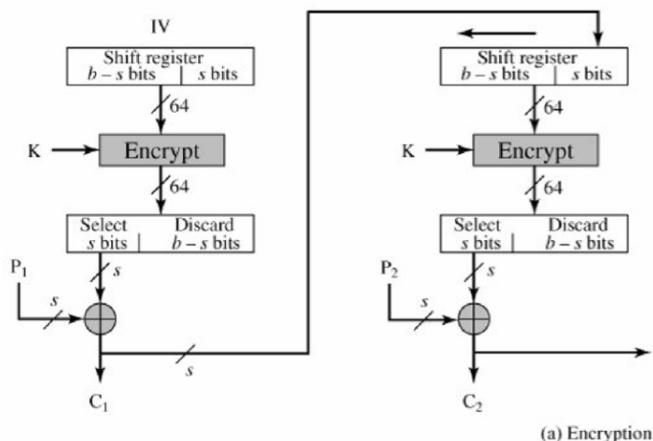


Introduciamo una chiave  $IV$  da applicare in XOR con la prima fase di encryption. Dalla seconda fase in poi l'IV è sostituito con il ciphertext del round  $P_i - 1$ .

Il problema con queste modalità è che l'operazione di cifratura avviene su singoli blocchi, piuttosto che su singoli caratteri. Sarebbe comodo poter trasmettere singoli blocchi cifrati alla volta.

- CFB, o "cipher feedback mode":

In questo schema di encryption, il vettore di inizializzazione è encryptato con la chiave  $k$ .  $n$  bits dell'output sono utilizzati come input per un'operazione di XOR con il plaintext. L'output è inserito in coda al vettore di inizializzazione prossimo, shiftando l'IV precedente.



da notare che l'output di encrypt sono 64 bit.  
si prende il primo blocco e quindi un char (8bits)

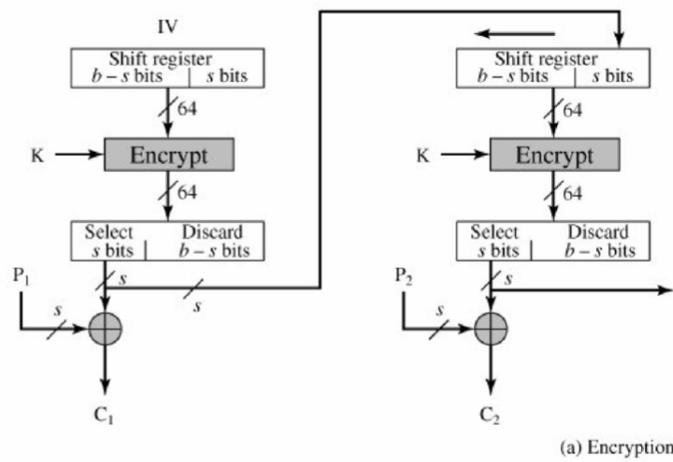
In questa modalità la grandezza dei blocchi del plaintext è  $s$  (di Select  $s$ -bits).

Se  $s = 1\text{char}(8\text{bits})$ , i blocchi del plaintext sono grandi  $s$ .

Il problema con questa modalità è che se compare un errore di trasmissione, esso si propaga, fintanto che il blocco corrotto venga espulso dall'IV.

- OFB, o *"output feedback mode"*:

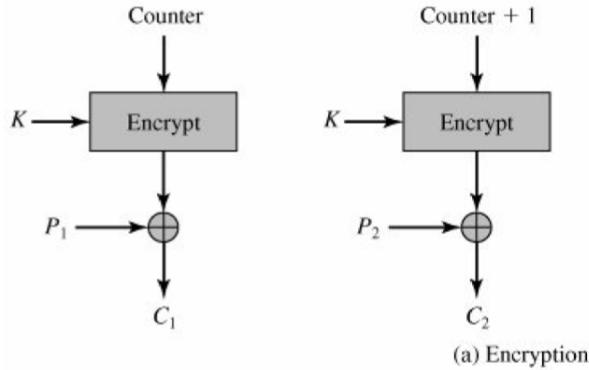
Questo schema di encryption è molto simile a CFB, ma che risolve il problema di propagazione di errore di trasmissione. Il feedback, che in CFB proveniva dopo allo XOR con il plaintext, viene piuttosto estratto in locale, prima dello XOR.



Il problema con questa modalità è che le operazioni avvengono necessariamente sequenzialmente, piuttosto che in parallelo.

- CTR, o *"counter mode"*:

Viene introdotto un counter che ha grandezza uguale al plaintext (che tuttavia deve essere necessariamente diverso rispetto ad ogni plaintext che si vuole encryptare). Le due parti devono concordarsi sul valore del counter iniziale, ma siccome il valore del counter è deterministico ed indipendente l'uno dall'altro, possiamo parallelizzare le operazioni di encryption.



- XTS, o *XEX-based tweaked-codebook mode with ciphertext stealing*  
Modalità di cifratura applicata su dischi.

Non possiamo utilizzare  $2^{256}$  perché 256 non è un numero primo. E' basato su un polinomio irriducibile di grado 8 sul campo di Galois che ha grado massimo 7, ovvero

$$GF(2^8) : ax^7 + bx^6 + cx^5 + dx^4 + ex^3 + fx^2 + gx^1 + h \text{ (range dei byte)}$$

Il polinomio  $m(x) = ax^8 + bx^7 + cx^6 + dx^5 + ex^4 + fx^3 + gx^2 + hx^1 + i$  funge da modulo. Il grado è 8 perché quando andiamo ad effettuare il modulo, il risultato sarà sicuramente di grado  $< 8$  che rientra ancora nel campo di Galois. I termini  $a, b, \dots, i$  appartengono a  $Z_2 (0, 1)$

Consideriamo  $(x^7 + 1)x^7 = x^{14} + x^7$ , come possiamo vedere il risultato sfiora oltre il range dei byte gestiti. Come nel cifrario di cesare effettuiamo un modulo, rispetto ad un polinomio  $m(x)$  di grado 8

$$x^{14} + x^7 \bmod(m(x))$$

*Ma quale  $m(x)$  dobbiamo utilizzare?*

Selezioniamo un  $m(x)$  di grado 8 irriducibile (sono circa 30), ad esempio:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Questo polinomio irriducibile è quello che è stato selezionato per AES.

Consideriamo  $m(x)$ , la sua scomposizione è:

$$m(x) * q(x) + r(x), \text{ e sul campo di Galois la rappresentazione di } m(x) \text{ è } r(x)$$

*Cosa succede se ad AES applichiamo un polinomio  $m'(x)$ , ovvero un altro polinomio di grado 8 irriducibile ?*

In questo caso otterremmo un  $r'(x)$ , ovvero una rappresentazione diversa di  $m(x)$

### *Double-DES — Doppio-DES*

Consideriamo un ciphertext cifrato con 2 round di DES, prima con una chiave  $k_1$  e poi con una chiave  $k_2$ . Lo sforzo computazionale per decryptare il ciphertext richiederebbe  $2^{56} + 2^{56}$  tentativi ovvero  $2^{57}$ , rispettivamente per ogni round.

$$C = E(k_2; E(k_1; p))$$

*Esiste una chiave  $k_3$  che possa passare direttamente da plaintext a ciphertext?*

- La risposta è no

Il problema con questa implementazione è che le chiavi da mantenere diventano 2:  $k_1, k_2$ . Per arginare questo problema la funzione di encryption diventa:

$$C = E_{k_1}(D_{k_2}(E_{k_1}(plaintext)))$$

*Ove  $E$  è l'operazione di encryption, mentre  $D$  è l'operazione di decryption*

Questa tecnica è chiamata *Triple-DES o 3-DES* ed è il meccanismo alla base dei pagamenti elettronici. La decryption è effettuata con la chiave  $k_2$ , che però è diversa dalla chiave  $k_1$ , aggiungendo ulteriormente entropia.

Se  $k_1 == k_2$  l'operazione di encryption è svolta una sola volta (single DES), che ci permette di comunicare anche con dispositivi datati che non supportano l'encryption basato su più chiavi.

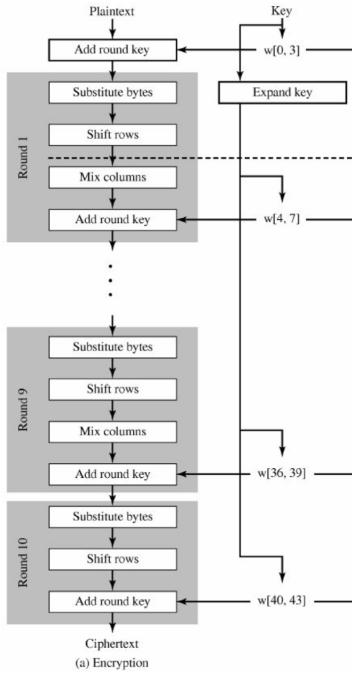
## **AES — Rijndael cipher**

Nel 1997 il NIST indice un nuovo contest per determinare un successore a DES, che verrà chiamato AES. Questo è fatto perché DES, avendo una chiave di 56-bit, era diventato vulnerabile ad attacchi brute force.

I 5 cifrari concorrenti arrivati alla finale sono: *Rijndael, Serpent, Twofish, RC6, MARS*, Rijndael è selezionato nel 2001 in base a test di performance, efficienza su hardware datato, e supporto a chiavi di grandezza superiore e variabile.

In particolare AES:

- il plaintext ha blocchi di grandezza fissa 128 bits
- la chiave a grandezza variabile: 128, 192 e 256 bits
- il ciphertext ha grandezza 128 bits



## Add-round key

E' l'operazione di "whitening" e ha l'obiettivo di neutralizzare il potenziale attacco basato su plaintext particolari (tutti 0, tutti 1 etc), consiste in un'operazione di XOR con la chiave colonna per colonna.

*Round<sub>n</sub>*

Le operazioni all'interno del *round<sub>n</sub>* sono svolte 9 volte e sono:

- Substitute bytes: corrisponde alle s-box in DES. in base ai valori presenti nello stato, vengono presi i valori all'interno di una matrice. Le s-box in AES sono di dimensione  $16 * 16$

*Ma come vengono calcolati i valori da inserire all'interno dell's-box?*

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

- Calcoliamo l'inverso moltiplicativo del valore  $yx$  (ove.  $riga = y$  e  $colonna = x$ ) sul campo di Galois  $GF(2^8)$

- Il byte risultante lo moltiplico per la matrice M

La matrice M è una matrice che è composta da n righe, costruite shiftando progressivamente verso destra il valore la seguente sequenza di bit:

```

1000 1111
1100 0111
...
fino a raggiungere
0001 1111

```

- Lo sommo per un valore costante C di valore 63

Il valore risultante è il valore da inserire all'interno della s-box in riga  $y$  e colonna  $x$ .

Ricaviamo il valore a partire da 95:

- L'inverso moltiplicativo di 95 è  $(95)^{-1} = 8A$
- Moltiplico l'inverso moltiplicativo per la matrice M (prodotto riga per colonna con la codifica in binario di  $8A$ )
- Faccio la somma con il valore 63 codificato in binario (11000110)

Il valore ricavato in riga 9, colonna 5 è 2A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

- Shift rows: la prima riga viene lasciata invariata, la seconda viene shiftata di due byte, e la terza viene shiftata di 3 byte.
- Mix columns: viene moltiplicata una colonna degli stati per una matrice nota, una colonna alla volta.
- Add round key: un altro round di ARK, tuttavia

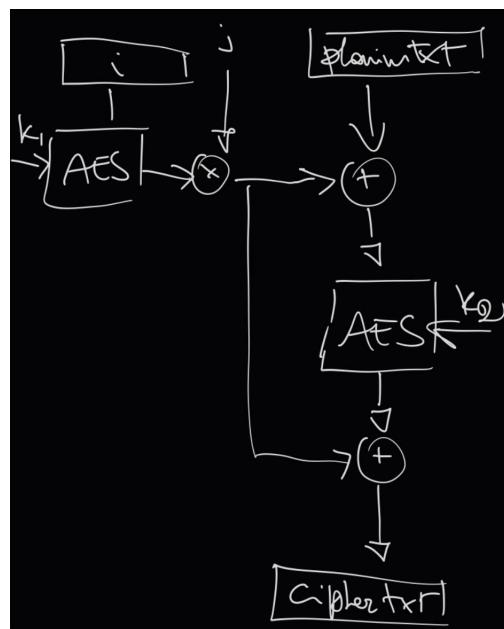
$Round_9 + 1 — Round_{10}$

Dopo aver svolto 9 volte le operazioni nel round, vengono effettuate nuovamente *substitute bytes e shift rows*.

L'output del decimo round è il ciphertext.

*Round keys — Key schedule*

### AES-XTS



- AES  $k_1$  e  $k_2$  sono le due porzioni di chiavi di chiave AES applicate.  $k_1$  e  $k_2$  hanno dimensione 128, se la chiave AES ha 256 di dimensione. (Sono come il left and right)

- $i$  e  $j$  sono rispettivamente il settore e la traccia del disco e hanno grandezza 128 bit. L'operazione che si svolge con  $j$  è quella di prodotto (settore cifrato \* traccia (non cifrata)). L'output è un polinomio in campo  $GF(2^{128})$  con coefficienti su  $Z_2$ , con modulo  $m(x) = x^{128} + x^7 + x^2 + x + 1$

- Questo output è utilizzato per fare lo XOR con il plaintext, svolgendo un'operazione di whitening, prima di passare all'operazione di  $AES_{k_2}$ , passando per un whitening ulteriore abbiamo l'output.

Siccome i blocchi sono formati da 128 bit, è possibile che ci siano blocchi incompleti. In questi casi sono introdotti dei padding.

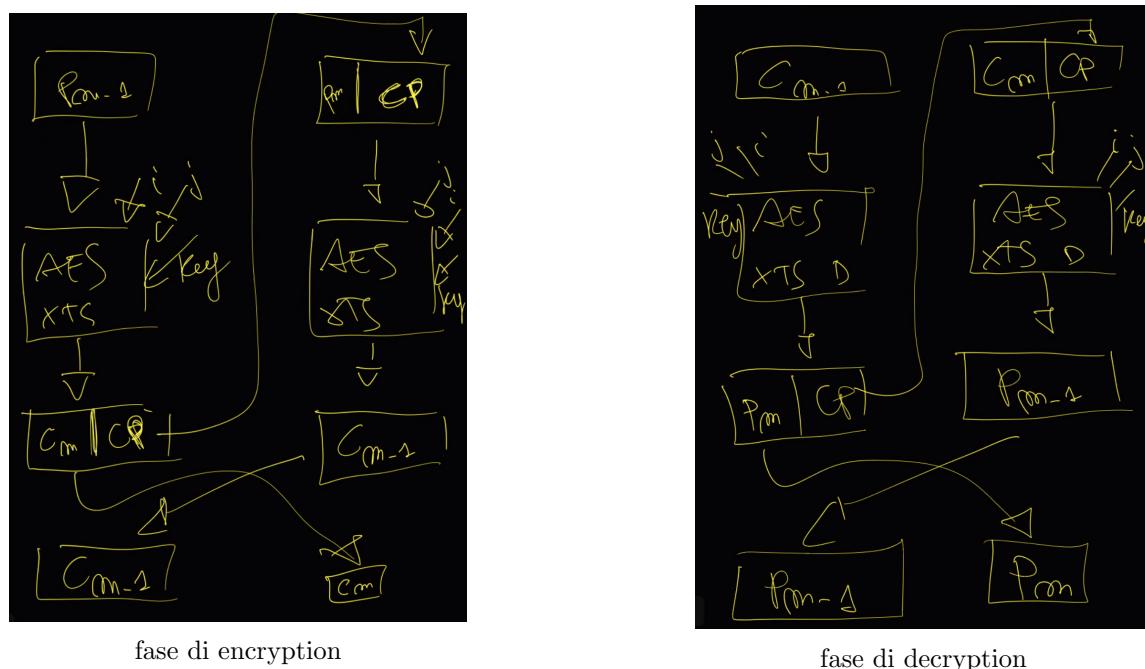
Sia un plaintext diviso in blocchi  $P_1, \dots, P_{m-1}, P_m$  con  $P_m$  incompleto, in AES-XTS i blocchi  $P_1, \dots, P_{m-2}$  sono gestiti normalmente, mentre i blocchi  $P_{m-1}$  e  $P_m$  sono gestiti in maniera speciale. In particolare:

- Il blocco  $P_{m-1}$  è passato in AES-XTS, generando

$$AES - XTS(< P_{m-1} >) = (C_m, CP)$$

- il blocco  $P_m$  è incompleto, ed è complementato da un padding, in particolare n digits dal ciphertext  $CP$  del blocco precedente.

$$AES - XTS(< P_m, CP_n >) = C_{m-1}$$

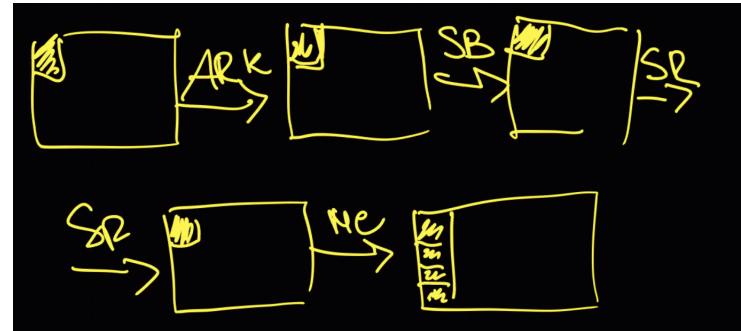


La fase di decryption funziona al contrario (figure 2)

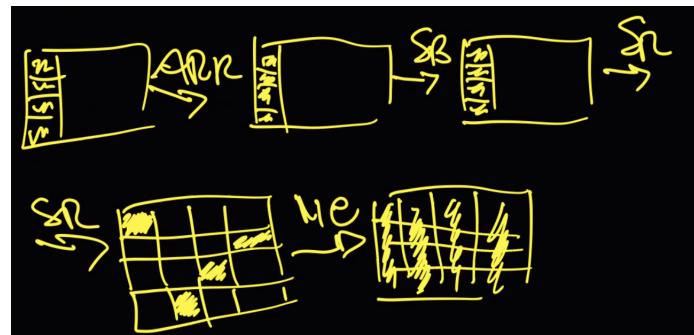
## Attacking AES (6 rounds)

La prima cella all'interno della matrice 4x4 è designata e chiamata attiva, gli altri valori sono costanti.  $\Delta - SET$ : l'insieme dei valori da 0 e 256  $P_1, \dots, P_{256}$  ovvero 00, 01, ..., FE, FF Somma zero (proprietà che ci permette di attaccare AES fino al terzo round): se prendiamo tutti i 256 crittostesti, lo xor su tutt i singoli byte in tutte le posizione la somma è uguale

a 0, è una proprietà che vale fino al terzo round (dal round  $\geq 4$  non vale più). Le operazioni che vengono svolte sono in ordine : *Add Round Key*, *Substitute Bytes*, *Shift Rows* ed infine *Mix columns*. Possiamo tuttavia notare che le tutte le operazioni non vanno a modificare la posizione dell'active cell

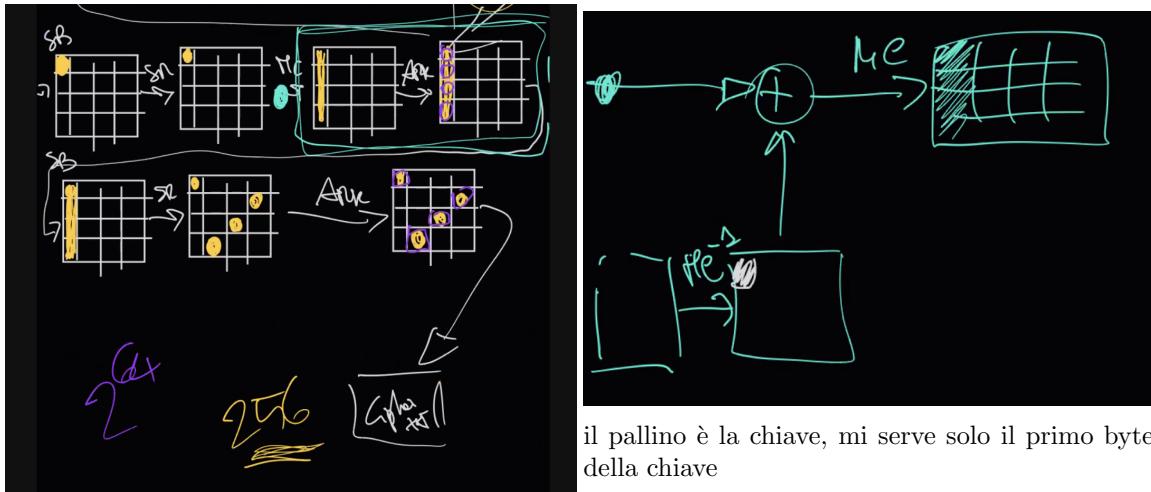


Tuttavia, dal secondo round in poi, possiamo vedere che lo stato della cella attiva 1 va ad influire sul valore di tutte le altre.



Una minima variazione del plaintext o della chiave va ad influire in maniera esponenziale sul ciphertext. Queste proprietà sono chiamate confusion and diffusion (tutto dipende da tutto). Un bit della chiave va ad influire tutto lo stato di AES

$$\begin{aligned} MC(\text{stato}) \oplus ARK = \\ MC(\text{stato}) \oplus MC(MC^{-1}(ARK)) = \\ MC(\text{stato} \oplus MC^{-1}(ARK)) \end{aligned}$$



Lo sforzo computazionale è sceso da  $2^{64}$  a  $2^8 * 5 = 2^{40}$ , uno nel penultimo round, 4 nei round. Per 5 round lo sforzo computazionale è  $2^{32} * 2^{40} = 2^{72}$ . Guardare sullo stallings

## Crittografia asimmetrica

Come abbiamo visto in precedenza, la crittografia asimmetrica differisce dalla crittografia simmetrica in quanto entrano in gioco più chiavi, le chiavi private. Le chiavi che entrano in gioco nei processi di cifratura e decifratura sono quindi differenti.

Le chiavi nella crittografia asimmetrica sono di piccola grandezza, in quanto il mantenimento comporta uno sforzo computazionale non indifferente

$$\begin{aligned} \text{Encryption: } & E_{pub}(M) = C \\ \text{Decryption: } & D_{priv}(C) = M \end{aligned}$$

Il processo di scambio di chiavi pubbliche avviene attraverso un processo di *handshake*.

## RSA

E' un algoritmo di cifratura asimmetrica introdotto negli anni 70, il nome proviene dal nome degli inventori (Rivest-Shamir-Adleman). E' un cifrario a chiavi pubbliche, i principali soggetti sono:

- chiavi pubbliche:  $public(n, e)$ :
  - (solitamente)  $e = 2^{16} + 1$
  - $n = p * q$

- chiavi private:  $private(p, q, d)$ :
  - $p$  e  $q$  sono due numeri (segreti) primi arbitrariamente grandi, in genere  $> 1024$  bit.
  - Questi valori devono essere grandi, per fornire un alto grado di entropia
  - $d * e = 1 \text{ mod } \phi(n)$  (d segreto)
  - $e$  è un numero casuale tale che  $MCD(e, \phi(n)) = 1$

Le componenti pubbliche sono  $n$  ed  $e$ , mentre le componenti private sono  $p, d, e$

Alice: a fase di encryption avviene

$$M^e = C \text{ mod } n \text{ ove } M < n$$

Bob: la fase di decryption avviene:

$$C^d = M \text{ mod } n$$

### Il valore e

Il valore  $e$  è un valore casuale, che tuttavia deve essere scelto tale che valga la seguente condizione  $MCD(e, \phi(n)) = 1$  (con l'applicazione dell'algoritmo di Euclide)

*Ma perché viene scelto  $2^{16} + 1$  solitamente?*

Consideriamo dove viene applicato:  $m^e \text{ mod } n$ , il valore  $2^{16} = 65537$  in codifica binaria è: 1000000000000001

Possiamo vedere che la compaiono molti 0, e pochi 1, ogni volta che compare 1 dobbiamo calcolare, quindi se minimizziamo il valore possiamo ridurre il numero di calcoli da effettuare. L'1 finale ci serve perché dobbiamo utilizzare un numero dispari. Il valore è in particolare  $e$  perché deve essere coprimo con  $d$ , verificabile calcolando l'inverso moltiplicativo

Il valore 65537 è scelto perché è un valore sufficientemente grande, che supera di un numero arbitrario di volte la  $n$  nel modulo, aumentando lo sforzo computazionale richiesto al momento del calcolo della radice. In particolare quando andiamo a fare il bruteforcing, più grande è il valore  $e$ , più alto il valore che dobbiamo bruteforcicare

### Il valore d

Il valore  $d$  è calcolato utilizzando l'algoritmo esteso di Euclide

### La fase di decryption ed encryption

Viene utilizzato il  $\phi(n)$

Per applicare l'algoritmo RSA abbiamo bisogno di un alcuni strumenti:

### Coprimi: "l'algoritmo di Euclide" e "piccolo teorema di Fermat"

Per verificare che due valori  $a, b$  siano coprimi, ovvero che  $MCD(a, b) = 1$  possiamo:

- Per valori non troppo grandi: per fattorizzare due valori  $a, b$  arbitrariamente grandi dobbiamo verificare tutti i numeri primi fino a radice di  $a$ .
- Per valori grandi utilizzare *l'algoritmo di Euclide*

### L'algoritmo di Euclide per il calcolo dell'MCD

L'algoritmo di Euclide è un algoritmo efficiente utilizzato per calcolare l'MCD di due valori interi. Se l'output dell'algoritmo di Euclide per due valori  $a, b$  è 1 allora possiamo concludere che due valori sono coprimi.

$$\begin{aligned} a &= bq_1 + r_1 \\ b &= r_1q_2 + r_2 \\ r_1 &= r_2q_3 + r_3 \\ &\dots \\ r_{k-2} &= r_{k-1}q_k + r_k \\ r_{k-1} &= r_kq_{k+1} + 0 \end{aligned}$$

Se ci troviamo in questa situazione l'algoritmo di Euclide ha trovato resto uguale a 0.

L'MCD è quindi  $MCD(a, b) = r_k$

Proviamo a fare il calcolo con  $a = 1180, b = 482$

$$\begin{aligned} 1180 &= 482 * 2 + 216 \\ 482 &= 216 * 2 + 50 \\ 216 &= 50 * 4 + 16 \\ 50 &= 16 * 3 + 2 \\ 16 &= 2 * 8 + 0 \end{aligned}$$

Possiamo concludere che  $MCD(1180, 482) = 2$

### L'algoritmo di Euclide esteso per l'inverso moltiplicativo

consideriamo  $MCD(a, b) = ax + by$

*Come calcoliamo x e y?*

Partiamo da x:

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= -q_{j-1} * x_1 + x_0 \\ &\dots \\ x_j &= -q_{j-1} * x_{j-1} + x_{j-2} \end{aligned}$$

Per calcolare y invece:

$$\begin{aligned}
y_0 &= 1 \\
y_1 &= 0 \\
&\dots \\
y_j &= -q_{j-1} * y_{j-1} + y_{j-2}
\end{aligned}$$

Considerando  $MCD(1180, 482) = 2 = 1180y + 482x$

$$\begin{aligned}
x_0 &= 0 \\
x_1 &= 1 \\
x_2 &= -2(1) + 0 = -2 \\
x_3 &= -2(-2) + 1 = 5 \\
x_4 &= -4(5) - 2 = -22 \\
x_5 &= -3(-22) + 5 = 71 \\
&\text{Ricaviamo che } x = 71
\end{aligned}$$

$$\begin{aligned}
y_0 &= 1 \\
y_1 &= 0 \\
y_2 &= -2(0) + 1 = 1 \\
y_3 &= -2(1) + 0 = -2 \\
y_4 &= -4(-2) + 1 = 9 \\
y_5 &= -3(9) - 2 = -29 \\
&\text{Ricaviamo che } y = -29
\end{aligned}$$

Allora abbiamo che  $MCD(1180, 482) = -29(1180) + 71(482) = 2$

L'inverso moltiplicativo deve essere = 1, ovvero  $ax + by = MCD(a, b) = 1$  Poniamo noi l'inverso moltiplicativo = 1 perché il processo di decifratura è unica, se avessimo 2 ci sarebbero più decifrazioni possibili

$$\begin{aligned}
by &\equiv 1 \pmod{a} \\
by &\equiv 1 + tot * a \text{ ove } tot = x \\
y * b &\equiv 1 \pmod{a}
\end{aligned}$$

### **Teorema piccolo di Fermat ed il teorema di Eulero**

Il teorema piccolo di Fermat dice che se  $p$  è un numero primo, per ogni intero  $a$ , il valore  $a^p - a$  è un intero multiplo di  $p$ .

$$a^p \equiv a \pmod{p} \text{ oppure } a^{p-1} \equiv 1 \pmod{p}$$

Attraverso il *teorema di Eulero* possiamo generalizzare il teorema di Fermat, in particolare: per ogni numero  $n$  ed ogni intero  $a$  coprimo rispetto a  $n$  abbiamo:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Ove  $\phi(n)$  rappresenta la funzione di Eulero, che conta il numero di interi fra 1 e  $n$  coprimi rispetto ad  $n$ . Se  $n = p$  è un numero primo allora

$$\phi(p) = p - 1$$

Vediamo un esempio di applicazione del piccolo teorema di Fermat.  
Supponiamo di voler calcolare  $2^{43210} \bmod 101$ .

Trasformiamo l'esponente: 43210 in  $432 * 100 + 10$

$$(2^{100})^{432} * 2^{10} \bmod 101$$

Teorema di Fermat:

$$a^{p-1} \equiv \bmod p$$

$$\text{Allora } 1^{432} * 2^{10} \bmod 101$$

$$\text{Infine: } 1024 \bmod 101 = 14$$

## Attacking RSA

Siccome il cifrario RSA è stato introdotto nel 1970, durante gli anni sono stati eseguiti numerosi attacchi, che vertono sulle problematiche algoritmiche, matematiche ed ingegneristiche.

### $\phi(n)$ accessibile

Consideriamo un caso in cui l'attaccante sia in grado di accedere al valore  $\phi(n)$ , magari attraverso ad un dump della memoria dopo che un programmatore l'ha utilizzato, senza cancellare il valore. Siccome i valori della chiave pubblica sono  $(n, e)$ , abbiamo i seguenti valori disponibili:  $(n, e, \phi)$ . Se effettuiamo:

$$\begin{aligned} n - \phi(n) + 1 &= \\ = pq - [(p-1)(q-1)] + 1 &= \\ = pq - [pq - p - q + 1] + 1 &= \\ = p + q & \end{aligned}$$

Deduciamo che se troviamo  $\phi$  possiamo ricavare  $p + q$ .

Consideriamo ora che  $n = p * q$  e  $\phi(n) = (p-1)(q-1)$

$$ax^2 + bx + c$$

$$\text{ove } c = p * q = n, a = 1,$$

$$b = p + q, \text{ che ricaviamo a partire da } n - \phi(n) + 1$$

$$x_1 x_2 = c/a$$

$$x_1 + x_2 = -b/a$$

Possiamo quindi ricavare che

$$x^2 - x(n - \phi(n) + 1) + n = 0$$

Se risolviamo con il  $\Delta$  possiamo ricavare i valori  $x_1 = p$  e  $x_2 = q$

## Caso valido su d

Riguardareeeeeee

Se  $MCD(a,n) = 1$  Allora esiste un valore s,t tle che  $s \cdot a + t \cdot n = 1$  Allora in questo caso esiste

## Casi particolari su d

Per l'algoritmo di Euclide esteso abbiamo messo come condizione che

$$MCD(a, n) = d = 1$$

Ma cosa succede se non fosse così?

$$\begin{aligned} MCD(a, n) &= d! = 1 \\ ax &\equiv b \pmod{n} \end{aligned}$$

Possiamo dire che:

- se d non divide b, allora non esiste soluzione per  $ax \equiv b \pmod{n}$
- se d divide b, allora esiste soluzione per  $ax \equiv b \pmod{n}$ , ma non è unica, la congruenza iniziale avrà  $d$  soluzioni. Per decifrare provare più soluzioni, scartando quelle che non mi interessano

$$\begin{aligned} ax &\equiv b \pmod{n} \\ ax/d &\equiv b/d \pmod{n/d} \end{aligned}$$

Chiamiamo la prima soluzione  $x_0$ , per calcolare le successive soluzioni possiamo

$$x_0 + n/d, x_0 + 2n/d, \dots, (d-1)(n/d)$$

Le soluzioni sono quindi  $(d-1)$

## m/4 cifre esposte di p, q, d

Supponiamo di avere

$$n = p * q \text{ di } n \text{ cifre}$$

Se siamo in grado di recuperare le prime  $m/4$  cifre di  $p$  o di  $q$  esiste un attacco in tempo polinomiale in grado di fattorizzare  $n$ . Questo attacco è stato successivamente esteso, in particolare se sono in grado di recuperare le prime  $m/4$  cifre di  $d$  sono ancora una volta in grado di rompere RSA

## Common modulus attack

Consideriamo:

$$Alice = (e_A, n) — Bob = (e_B, n)$$

Cosa succede se sia Alice che Bob utilizzano uno stesso  $p, q$ , e di conseguenza  $n$ ?

$$\begin{aligned} M^{e_A} &= C_1 \bmod n \\ M^{e_B} &= C_2 \bmod n \end{aligned}$$

Consideriamo un terzo soggetto Carl, che deve inoltrare un messaggio ad Alice e Bob, cifra i messaggi:

$$\begin{aligned} \text{Crittotesto per Alice: } M^{e_A} &= C_1 \bmod n \\ \text{Crittotesto per Bob: } M^{e_B} &= C_2 \bmod n \end{aligned}$$

Se Eve è in grado di recuperare i crittostesti  $C_1$  e  $C_2$ , e nota che gli  $n$  di Bob e Alice sono congruenti Consideriamo

$$MCD(C_1, n) = 1$$

$$\begin{aligned} & \left( C_1^{-1} \right)^{-\mathcal{E}} \cdot \left( C_2 \right)^S = \\ &= \left( \left( M^{e_A} \right)^{-1} \right)^{-\mathcal{E}} \cdot \left( M^{e_B} \right)^S = \\ &= M^{e_A \cdot \mathcal{E}} \cdot M^{e_B \cdot S} \end{aligned}$$

$$\begin{aligned}
 J &= [e_A^r + e_B s]^{-1} \pmod{\phi(n)} \\
 &= m \pmod{n} \\
 &\equiv m^{-1}
 \end{aligned}$$

Come possiamo vedere Eve è stata in grado di recuperare il messaggio  $m$  sfruttando una vulnerabilità  $n$  comune

### Low exponent attacks

Consideriamo  $(e, n)$ ,

$$\begin{aligned}
 d * e &\equiv 1 \pmod{\phi(n)}, \text{ con } d \in [1, \dots, \phi(n)] \\
 \text{Se } d &< 1/3\sqrt[4]{n}
 \end{aligned}$$

Esiste un algoritmo in grado di attaccare la chiave in pochi millisecondi

### Short plaintext attack

Attacco probabilistico attuabile nel caso in cui un plaintext sia troppo piccolo, in particolare un plaintext cryptato con DES avrà ordine di grandezza di  $m = 10^{17} \equiv 2^{56}$ . Il valore  $m$  è ri-scrivibile come

$$m = a * b$$

$a, b$  avranno, con una probabilità non trascurabile, un ordine di grandezza di  $10^9$  (valore preso accettabile per un computer diffuso) Li suddivido ora in due array di  $10^9$  elementi, in particolare ricavati rispettivamente con

$$array_1 = c * x^{-e} \pmod{n}$$

$$array_2 = y^e \pmod{n}$$

Con  $x, y$  che sono gli indici negli array

$$C \cdot x^{-e} \pmod{m} \quad C(10^9)^c$$


  
 $\boxed{C.1 \mid C.2 \mid \dots \mid C.n}$        $10^9$

$$y^e \pmod{N}$$


  
 $\boxed{1^e \mid 2^e \mid 3^e \mid \dots \mid n^e}$        $10^9$

In una posizione arbitraria in ogni array sarà possibile trovare 2 celle con lo stesso valore, uguagliando il contenuto dei due array è possibile ricavare il messaggio.

$$C \cdot n^e = y^e$$

$$C = (n \cdot y)^e$$

$$\underline{m} = n \cdot y$$

In particolare ho trovato i fattori  $x * y$ , ricavando così il valore  $m$

### Padding per short plaintexts — OAEP

Per raggiungere il problema di plaintext troppo corto viene aggiunto un padding, ovvero una stringa definita, inserita in coda al plaintext. In particolare per RSA viene utilizzato l'*OAEP* (optimal asymmetric encryption padding), il cui valore è generato da una funzione Feistel (la parte della generazione del padding è ignorata). Per criptare plaintext corti, quindi, viene generato un padding di lunghezza  $x$ .

Dato un messaggio troppo corto  $M$  di lunghezza  $i$ , il messaggio unito col padding  $P$  con lunghezza  $k$  diventa:

$$M_i \rightarrow M_i P_k$$

$M_i P_k$  è poi utilizzato per produrre il ciphertext

### Teorema cinese dei resti

Permette di fare il calcolo dell'inverso attraverso la risoluzione di sistemi, che permette di scaricare il carico computazionale al server Consideriamo un set di numeri interi o naturali:

$$\begin{aligned} & \text{for } i \ P_i/Z_i = r_1 r_2 \dots r_{i-1} r_i \\ & \text{for } i \ Q_i/Y_i = Z_i^{-1} \bmod r_i \\ & x = a_1 y_1 z_1 + \dots + a_n y_n z_n \end{aligned}$$

Questo viene fatto perché il calcolo dell'inverso moltiplicativo nel metodo tradizionale è un'operazione dispendiosa. La risoluzione del sistema risulta meno dispendiosa, e ci permette anche a scaricare il server ad un'altra location.

### **Lemma del teorema cinese dei resti**

Consideriamo due interi  $m, n$  coprimi fra loro, ovvero:

$$m, n \text{ t.c. } MCD(m, n) = 1$$

Se  $c$  un altro intero è multiplo di  $m$  e  $n$

$$\text{ovvero } c = m * k, c = n * l$$

allora

$$c \text{ è multiplo di } (m * n) * x = c$$

### **Lemma del teorema cinese dei resti**

Consideriamo:

$$(m^e)^d = m \bmod n$$

Consideriamo 2 dei 4 casi possibili su  $m$

$$MCD(p, m) \neq 1 = d$$

Allora  $p$  ed  $m$  hanno qualcosa in comune, in particolare  $d = p$ , e  $p|m$ , di conseguenza

$$M \equiv 0 \bmod p$$

Consideriamo  $e * d = 1 \bmod \phi(n)$

$$M^{ed} \equiv M^{k(p-1)(q-1)+1} \bmod p$$

*TO REVIEW*

### **Timings attacks**

Un attacco notato per la prima volta da uno studente della Stanford 20 anni dopo l'implementazione di RSA. Questo attacco espone informazioni private misurando l'ammontare di tempo richiesto o analizzando l'oscillazione durante le operazioni svolte sulla chiave privata.

Possiamo raggirare questa problematica offuscando l'implementazione, rendendo il tempo d'esecuzione costante (per esempio introducendo una serie di nop), in alternativa attraverso la tecnica di sliding windows, si effettuano i calcoli su un subset di bits.

## Chiavi RSA

Consideriamo i valori  $p, q$ , due numeri primi, abbiamo detto che questi valori devono essere arbitrariamente grandi per garantire un buon grado di sicurezza

*Ma come verifichiamo che un numero sia adatto?*

Siccome non possiamo utilizzare un algoritmo deterministico per determinare  $n$ , in quanto il calcolo dei numeri primi non è deterministico, utilizziamo un approccio probabilistico. Andiamo a verificare il valore  $n$  con una serie di test per escludere valore inadatti.

In modo per determinare un valore  $n$  adatto è il seguente:

- determiniamo un lower bound di  $10^{100}$ , determiniamo un valore  $P_0 = \cong 10^{40}$
- verifichiamo con  $P_0$  con Miller Rabin
- determiniamo un valore  $k$  tale che  $P_0 * k \cong 10^{100}$ ,  $k$  sarà dell'ordine di  $10^{60}$
- $p_0 * k + 1 = p$ , applichiamo Miller Rabin  $n$  volte, se passa,  $p$  è primo

Questo ci permette di evitare l'attacco di Pollard in quanto il valore è  $p - 1$  è sicuramente un valore non basso.

## Test di primalità di Fermat

Consideriamo

$$\begin{aligned} n, x, y \text{ tale che } x^2 &\equiv y^2 \pmod{n} \\ \text{se } n \neq \pm y &\rightarrow n \text{ è composto} \end{aligned}$$

In particolare

$$MCD(x - y; n) \text{ fornisce un fattore di } n$$

Consideriamo ora

$$\begin{aligned} n &\text{ un intero arbitrariamente grande} \\ \pm a &< n - 1 \\ \text{se } a^{n-1} &= d \pmod{n} \\ \text{Ovvero che } a^{n-1} &\neq 1 \pmod{n} \end{aligned}$$

allora possiamo concludere che  $n$  è un numero composto, per il piccolo teorema di Fermat. se invece:

$$\text{se } a^{n-1} = 1 \pmod{n}$$

Non possiamo concludere nulla su  $a$ . Il teorema piccolo di Fermat è quindi utilizzato per verificare che un numero è composto.

Siccome non siamo stati in grado di determinare nulla su  $a$ , utilizziamo un altro test, in particolare il test di Miller-Rabin

## Test di primalità di Miller Rabin

Consideriamo un valore primo dispari  $n$  da verificare

$$n - 1 = 2^k * m$$

Consideriamo:

$$a < n - 1$$

Andiamo ad applicare l'algoritmo di Miller Rabin

$$b_0 = a^m \bmod n$$

Se  $b_0$  è uguale a

- $+/- 1$  allora  $n$  è probabilmente primo
- altro, allora proseguiamo con  $b_1$

$$b_1 = b_0^2 \bmod n$$

Se  $b_1$  è uguale a

- $+1$  allora  $n$  è composto
- $-1$  allora  $n$  è probabilmente primo
- altro, allora proseguiamo con  $b_2$

In particolare  $b_2$  viene calcolato nella stessa maniera in cui è calcolato  $b_1$ , elevando alla seconda il valore  $b_{m-1}$  e modulando.

$$b_k = (b_{m-1})^2 \bmod n$$

L'algoritmo termina quando calcoliamo il valore  $b_{k-1}$ , in particolare se  $b_{k-1}$  è uguale a

- $+1$  allora  $n$  è composto
- $-1$  allora  $n$  è probabilmente primo
- altro, allora scegli un altro valore  $n$  da verificare

L'algoritmo di Miller Rabin ha una probabilità di errore del 25%, un risultato su 4 è errato.

Tuttavia 25% non è soddisfacente, per mitigare la probabilità l'algoritmo di Miller Rabin è applicato più volte su  $j$  valori diversi, in particolare se la probabilità è 25% per una verifica, per 10 verifiche la probabilità di errore diventa

$$(1/4)^{10}$$

I valori che passano il test di Fermat sono chiamati *pseudoprimi*, mentre quelli che passano sia Fermat che Miller Rabin sono chiamati *pseudoprimi forti*. Consideriamo

$$\begin{aligned} n = 13 &\rightarrow n - 1 = 12 = 2^2 * 3 \text{ prendiamo } a = 11 \\ b_0 &= 11^3 \bmod 13 \quad b_0 = 5 \quad b_1 = 25 \bmod 13 - > 12 \quad b_2 = 144 \bmod 13 \end{aligned}$$

Le banche utilizzano Fermat, Miller Rabin, Solovay–Strassen per effettuare test di primalità.

## Fattorizzazione di n

Dato un valore  $n$  composto, come possiamo fattorizzarlo nei valori  $p, q$ ? Introduciamo la tecnica di fattorizzazione di Fermat. Consideriamo  $n, i, j$  tale che

$$n + i^2 = j^2$$

Allora possiamo fattorizzare  $n$ , infatti:

$$\begin{aligned} n &= j^2 - i^2 = \\ &= (j+i)(j-i) \end{aligned}$$

Ad esempio consideriamo  $n = 295927$

$$n + 1^2 = j^2$$

se è possibile abbiamo finito, senò proseguiamo:

$$\begin{aligned} n + 2^2 &= j^2 \\ n + 3^2 &= 295936 = 544^2 \\ n &= (544 + 3)(544 - 3) = \\ &= 547 * 541 \end{aligned}$$

Abbiamo trovato che  $p = 547, q = 541$ . La difficoltà sta nel trovare i valori  $i, j$ , in quanto la ricerca di tali valori è lenta.

## Fattorizzazione di n — algoritmo di Pollard

Consideriamo

$$n = p * q$$

Se è possibile scrivere  $p - 1$  come un prodotto di numeri primi piccoli, allora il valore  $p$  è facilmente attaccabile con l'algoritmo di Pollard. Ad esempio:

$$p - 1 = 2^{101} * 3^7 * 5^{10} * 7^{701}$$

Come possiamo vedere i tali valori sono "piccoli", il valore  $p$  non è adatto.

## Quadrato di Sieve

Il quadrato di Sieve è un algoritmo per la fattorizzazione (sieve = setaccio). Consideriamo

$$n = p * q$$

Selezioniamo  $k$  valori su cui applicare  $\text{mod } n$ .

$$123456^2 \equiv x \pmod{n}$$

Supponiamo che  $x = 2^3 * 3^4 * 11 * 23^2$ , continuiamo provando con  $n$  valori  $x_n$

$$n = 3837523$$

Selezioniamo  $k$  valori su cui applicare  $\text{mod } n$ , più valori scegliamo, più precisa sarà la nostra stima. Ad esempio scegliamo 9398, 19095, 1964, 17078

$$\begin{aligned} 9398^2 &= 5^5 * 19 \text{ mod } n \\ 19095^2 &= 2^2 * 5 * 11 * 13 * 19 \text{ mod } n \\ 1964^2 &= 3^2 * 13^3 \text{ mod } n \\ 17078^2 &= 2^6 * 3^2 * 11 \text{ mod } n \\ &\dots \end{aligned}$$

I valori a sinistra sono tutti valori al quadrato, moltiplicandoli tra di loro avremo sicuramente un valore quadrato, in particolare:

$$(9398 * 19095 * 1964 * 17078)^2$$

Il valore all'interno delle parentesi lo chiamiamo  $x$ . I valori a destra invece abbiamo  $n$  valori con la stessa base, progressivamente moltiplichiamo tra loro e ricaviamo:

$$2^8 * 3^4 * 5^6 * 11^2 * 13^4 * 19^2$$

Notiamo che tutti questi valori hanno una potenza quadrata, possiamo quindi riscriverla così:

$$(2^4 * 3^2 * 5^3 * 11 * 13^2 * 19)^2$$

Il valore all'interno delle parentesi lo chiamiamo  $y$ .

$$x^2 \equiv y^2 \text{ mod } n$$

Allora  $n$  è composto e posso trovare uno dei due fattori di  $n$  facendo:  $x \neq \pm y \rightarrow$   
 $(MCD(x - y; n) = d \neq 1)$   
 Valore che è uguale a  $p$  o  $q$

## Problema del logaritmo discreto

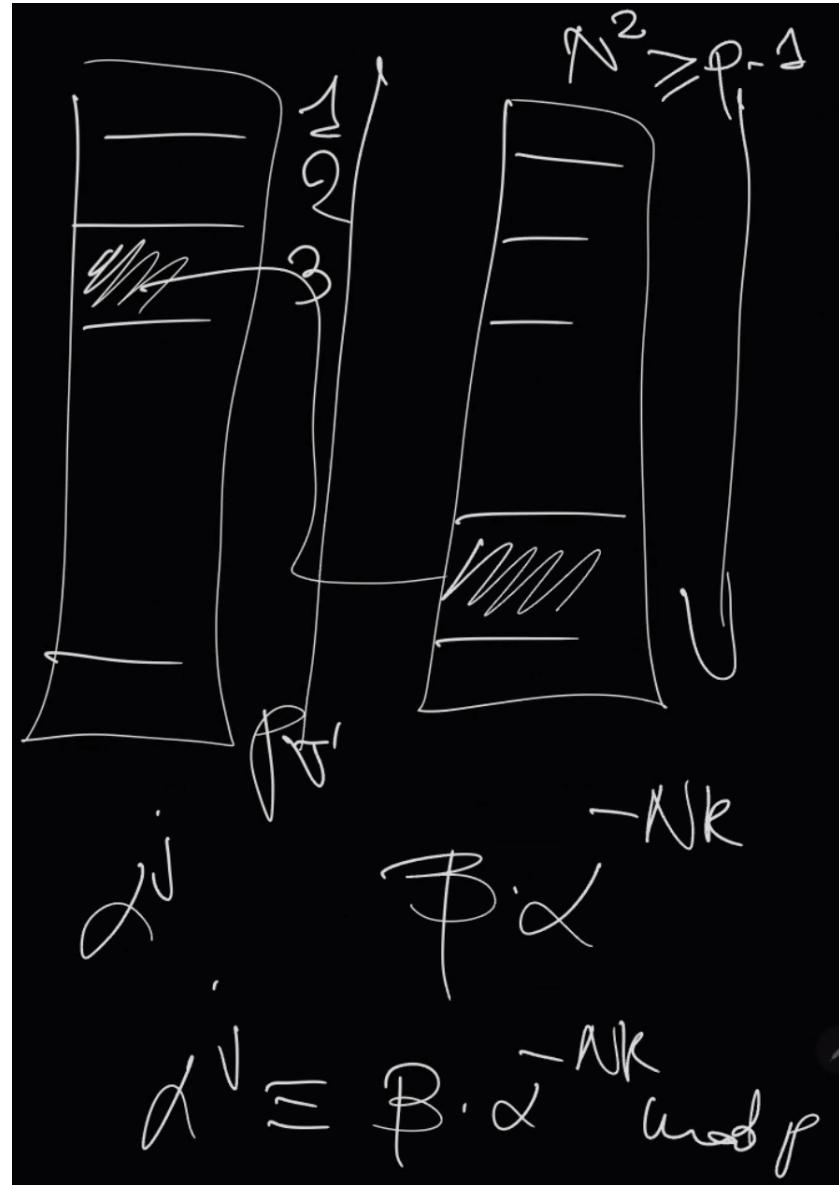
$$\alpha^x \equiv \beta \text{ mod } p$$

Come faccio a trovare  $x$ ? La cosa più intuitiva sarebbe utilizzare il logaritmo

## Baby step giant step

Il primo approccio è quello di utilizzare il *baby step giant step*

Creiamo due vettori di grandezza  $p$  (ovvero il modulo) in cui vengono inseriti gli elementi  $\alpha_j$  nel primo vettore,  $\beta\alpha^{-Nk}$  nel secondo vettore. Cerchiamo poi due valori comuni, per poi effettuare l'uguaglianza.



In particolare l'uguaglianza

$$\alpha^j \equiv \beta * \alpha^{Nk} \pmod{p}$$

E' uguale a

$$\alpha^{j+NK} \equiv \beta \pmod{p}$$

Ricordiamo che:

$$\begin{aligned}\alpha^x &\equiv \alpha^y \pmod{p} \\ x &\equiv y \pmod{p-1} \\ \alpha^{p-1} &\equiv 1 \pmod{p}\end{aligned}$$

### analisi su parità o meno di x

Per dividere in 2 il sample size *su carta*

### Index calculus

E' una variabile del quadratic sieve basato su

- un passo di precomputazione
- calcolo dei logaritmi
- un passo di controllo proprietà, in particolare se è scomponibile

Consideriamo

$$\begin{aligned}\beta &\equiv \alpha^x \pmod{p} \\ \log_\alpha \beta &= x \pmod{p-1}\end{aligned}$$

Consideriamo  $a^k$  così definito:

$$a^k = \Pi p_i^{a_i} \pmod{p} \text{ ove } \Pi \text{ è la produttoria}$$

Allora il passo computazionale è il seguente:

$$k = \log_\alpha(\Pi p_i^{a_i}) = \sum a_i \log_\alpha p_i \pmod{p-1}$$

Questo ci permette di mettere in evidenza i logaritmi. Il valore  $\beta$  lo moltiplichiamo per un valore  $a$  elevato ad un valore  $r$  random, che ci permette di espanderlo nel seguente modo:

$$\begin{aligned}\beta * \alpha^r &= \Pi p_i^{b_i} \pmod{p} \\ \log_\alpha \beta + r \log_\alpha \alpha &= \sum b_i \log_\alpha p_i \pmod{p-1}\end{aligned}$$

Possiamo esporre quello che ci serve:

$$\log_{\alpha}\beta = -r + \sum b_i \log_{\alpha} p_i \bmod p - 1$$

Abbiamo quindi trovato  $x$ , dato che  $x = \log_{\alpha}\beta$ . Dobbiamo scegliere un  $r$  adeguato che mi generi un  $p_i$  sufficientemente piccolo per garantire che l'abbia generato in precedenza

$$\begin{aligned} p &= 131 \quad \alpha = 2 \\ B &= \text{soglia} = 10 \quad \beta = 37 \end{aligned}$$

La soglia va ad influire sui valori possibili di  $p_i$  in particolare questi valori devono essere minori della soglia, definita in base alle capacità computazionali del sistema, in questo caso:

$$p_i = 2, 3, 5, 7$$

### Vediamo un esempio pratico

Impostiamo il problema:

$$\begin{aligned} \beta &= \alpha^x \bmod p \\ 37 &= 2^x \bmod 131 \end{aligned}$$

Calcolo gli  $\alpha^k$ , tale che  $\alpha$  è scomponibile in un prodotto numeri primi minori della soglia, ovvero il passo di precomputazione  $a^k = \prod p_i^{a_i} \bmod p$ :

$$\begin{aligned} 2^1 &= 2 \bmod 131 \\ \dots \\ 2^8 &= 5^3 \bmod 131 \\ \dots \\ 2^{12} &= 5 * 7 \bmod 131 \\ \dots \\ 2^{14} &= 3^2 \bmod 131 \\ \dots \\ 2^{34} &= 3 * 5^2 \bmod 131 \\ \dots \end{aligned}$$

Calcoliamo ora  $k$ :  $k = \log_{\alpha}(\prod p_i^{a_i})$ .

Siccome stiamo introducendo i logaritmi passiamo a  $\bmod p - 1$

$$\begin{aligned} 1 &\equiv \log_2 2 \bmod 130 \\ 8 &\equiv 3 \log_2 5 \bmod 130 \\ 12 &\equiv \log_2 5 + \log_2 7 \bmod 130 \\ 14 &\equiv 2 \log_2 3 \bmod 130 \\ 34 &\equiv \log_2 3 + 2 \log_2 5 \bmod 130 \end{aligned}$$

Possiamo procedere con l'esplicitazione dei logaritmi, se questi sono invertibili

$$\begin{aligned}
\log_2 2 &= 1 \bmod 130 \\
\log_2 5 &= 5 * 87 = 46 \bmod 130 \\
\log_2 3 &= 72 \bmod 130 \\
\log_2 7 &= 12 - 46 = -34 = 96 \bmod 130
\end{aligned}$$

Continuiamo con  $\beta * \alpha^r = \prod p_i^{b_i} \bmod p$

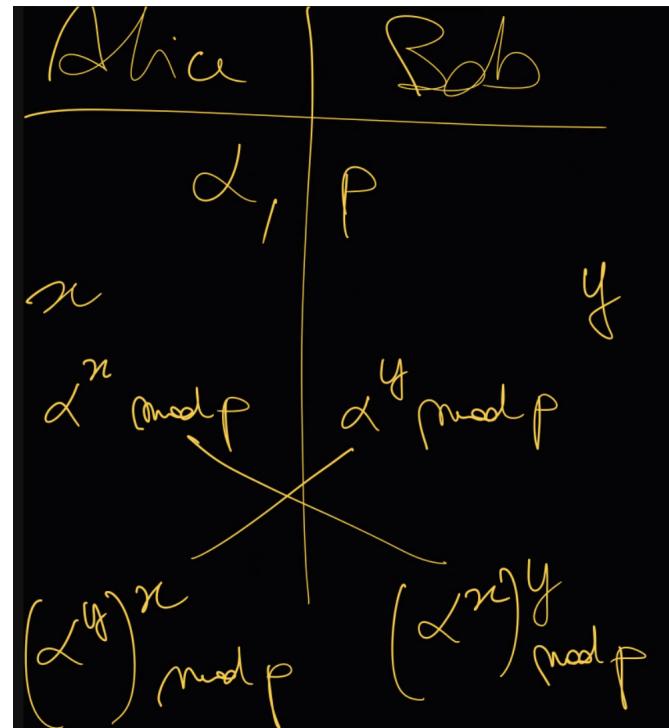
$$\begin{aligned}
37 * 2^{43} &= 3 * 5 * 7 \bmod 131 \\
\log_2 37 + 43 \log_2 2 &= \log_2 3 + \log_2 5 + \log_2 7 \\
\log_2 37 &= -43 + \log_2 3 + \log_2 5 + \log_2 7 \\
x &= -r \sum b_i \log_p \alpha_i
\end{aligned}$$

## Logaritmo discreto

Casi d'uso del logaritmo discreto

### Diffie Hellman

L'algoritmo del logaritmo discreto è utilizzato ogni giorno per lo scambio delle chiavi secondo l'algoritmo di Diffie-Hellman, che permette di scambiare le chiavi su un canale insicuro. Supponiamo  $\alpha, p$  due valori pubblici, e le due chiavi  $x, y$  possedute da Alice e Bob, come possono scambiare le chiavi su un canale insicuro?



## Problemi con Diffie Hellman

Consideriamo Alice, Bob ed un attaccante Eve. L'algoritmo di Diffie Hellman ci dice che:

Alice spedisce  $\alpha^x \bmod p$   
Bob spedisce  $\alpha^y \bmod p$

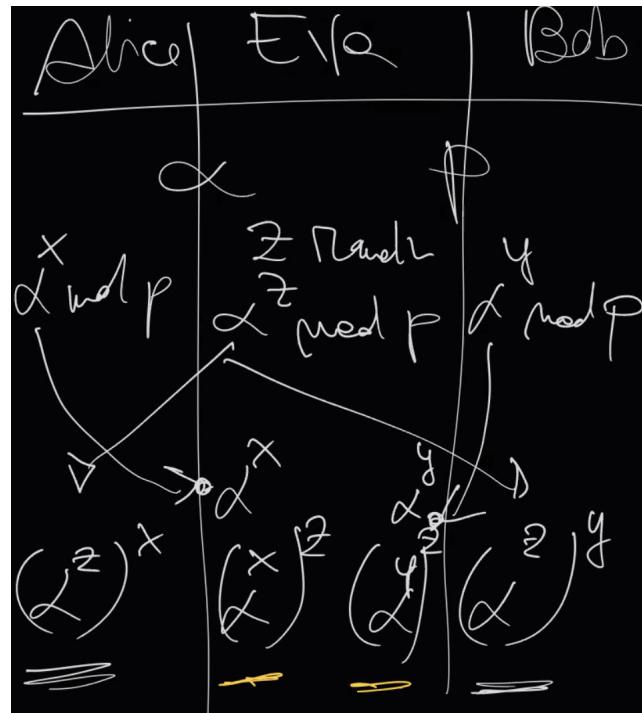
Consideriamo Eve che riesce a fare un attacco man in the middle:

Eve possiede  $\alpha^z \bmod p$   
Eve intercetta  $\alpha^x, \alpha^y$

Eve procede a inoltrare il valore tampered  $\alpha^z$  ad Alice e Bob, che rispettivamente svolgono:

Alice:  $(\alpha^z)^x$  — Bob:  $(\alpha^z)^y$

Eve è in grado di intercettare i due valori  $x$  e  $y$ , e ciò gli permette di leggere e contraffare tutti i messaggi.



Per raggiungere questo problema è stato implementato STS, un sistema di firme digitali per garantire l'autenticità

## Bit commitment

Un'altra applicazione del logaritmo discreto è durante l'utilizzo del bit commitment, o schema di commitment. questo ci permette di mostrare il possedimento di un dato valore senza mostrarlo. E' poi possibile mostare il dato valore in futuro.

$$\alpha^x \equiv \beta \pmod{p}$$

## ElGamal

E' un algoritmo di cifratura a chiave asimmetrica. Consideriamo

- Alice con un messaggio  $m$
- Bob che sceglie  $\alpha$  radice primitiva,  $\phi$  numero primo,  $a$  numero segreto

Bob computa:

$$\alpha^a \equiv \beta \pmod{p}$$

Valori da pubblicare:  $(\alpha, p, \beta)$

Alice per cifrare il proprio messaggio scarica le informazioni pubbliche di bob  $(\alpha, p, \beta)$ , genera un valore  $k$  casuale segreto. A partire da  $k$  calcola  $r, t$ , in particolare:

$$r = \alpha^k \pmod{p}$$
$$t = \beta^k * m \pmod{p}$$

La coppia  $(r, t)$  rappresenta il messaggio cifrato da spedire a Bob. Bob per decifrare effettua:

$$t * r^{-a} \equiv m \pmod{p}$$

Stiamo incorporando il messaggio all'interno del problema del logaritmo discreto, che l'attaccante non riesce a bucare.

## Dimostrazione di corretta di ElGamal

Consideriamo:

$$t * r^{-a} = (\beta^k * m)(\alpha^k)^{-a} =$$
$$= [(\alpha^a) * m^k] * (\alpha^k)^{-a} = m \pmod{p}$$

## Problematiche con ElGamal

### Valore k comune

Consideriamo Alice con due messaggi  $m_1, m_2$ , cosa succede se Alice cifra due messaggi con  $k$  uguale. Alice cifra:

$$(r, t_1) | (r, t_2)$$

Siccome  $r$  è una componente pubblica, l'attaccante può vedere senza problemi che Alice ha utilizzato un  $k$  uguale per entrambi i messaggi. L'attaccante conosce  $t_1, t_2$ , ma non i valori a destra dell'uguale:

$$\begin{aligned}t_1 &= \beta^k * m_1 \bmod p \\t_2 &= \beta^k * m_2 \bmod p\end{aligned}$$

L'attaccante tuttavia sa che:

$$t_1/m_1 = \beta_k = t_2/m_2$$

Può computare il messaggio facendo:

$$m_2 = t_2 * m_1 / t_1$$

## Funzioni Hash

Una funzione hash è una funzione matematica/crittografica. Le principali caratteristiche delle funzione hash sono:

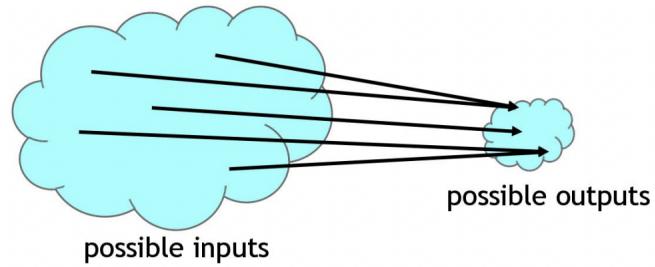
- la facilità di computazione
- l'input può essere una stringa di una qualsiasi dimensione
- l'output ha una dimensione fissa, specifica per ogni funzione hash

Alcuni esempi di funzioni hash sono SHA1, SHA256, MD5, SHA3 etc.

### Collision free property

Una funzione di hash è perfetta se è iniettiva, ovvero se  $u = v$ ,  $h(u) \neq h(v)$ , più una funzione di hash sparpaglia gli elementi, meglio è. E' tuttavia possibile che due elementi abbiano una stessa chiave, in questo caso parliamo di *collistione*.

Le collisioni comportano che un output della hash function abbia 2 input medesimi. E' tuttavia possibile trovarli?

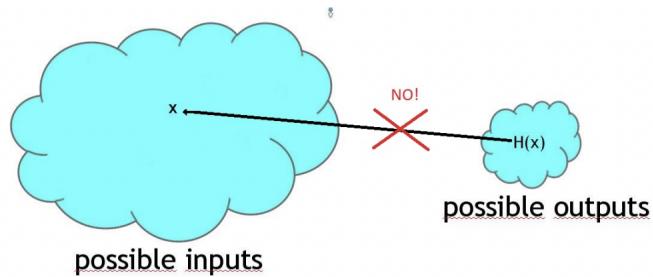


Se vengono computati i valori hash di  $2^{130}$  stringhe, la probabilità di trovare 2 valori con lo stesso output hash è del 99%. E' tuttavia un valore così astronomicamente alto che è possibile considerarla nulla.

Possiamo quindi considerare che le funzioni hash siano collision free (non proprio, ma non importa). Possiamo quindi considerare che:

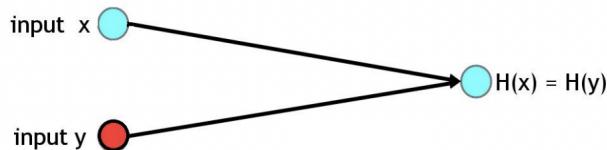
$$H(x) = H(y) \rightarrow x = y$$

### Pre-image resistance



Consideriamo un valore hash  $H(x)$ , è computazionalmente impossibile ricavare  $x$ , ovvero la funzione non è reversibile (a partire da un valore hashato, non posso risalire alla fonte). In particolare la funzione hash è chiamata one-way. Possiamo quindi concludere che per nascondere un messaggio  $x$ , è possibile nasconderlo con  $H(x)$

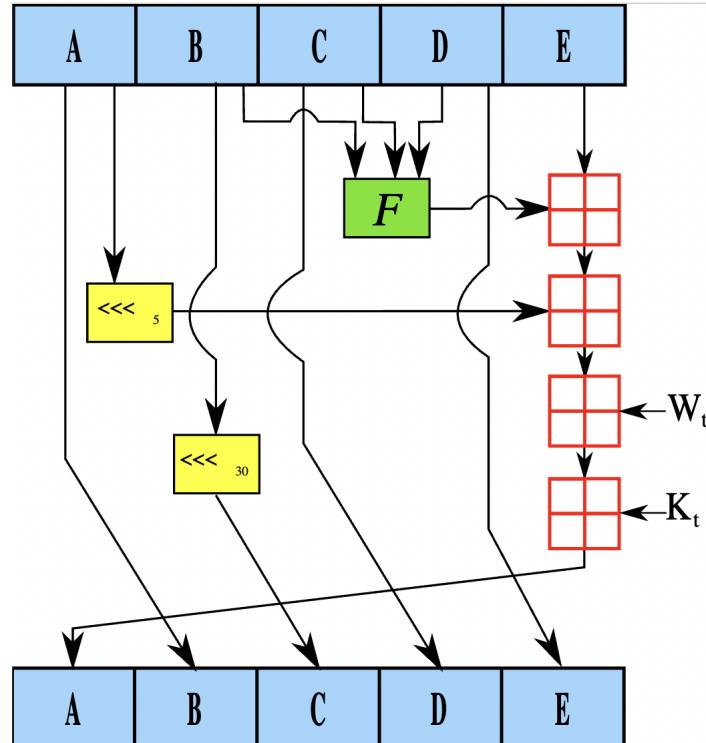
### Second pre-image resistance



Dato un valore  $x$ , è computazionalmente impossibile trovare un valore  $y$  tale che  $H(x) = H(y)$ .

## SHA1

SHA1 è una funzione hash crittografica che produce hash values di lunghezza 160. SHA1 lavora svolgendo 80 round della seguente funzione:



Ogni 20 round la funzione  $f$  ed il valore  $k$  cambiano

- I blocchi A, B, C, D sono blocchi di 32 bit  
In particolare i blocchi A, B, C, D sono inizializzati con valori in big endian con valore esadecimale in ordine crescente:

```

h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

```

I valori costanti presenti nei blocchi cambiano dal passo  $n_2$  in poi. Dopo 80 round di SHA1, questo output è utilizzato come valore costante iniziale per i blocchi successivi

- $F$  è una funzione lineare
- $<<<$  rappresenta una funzione di shift, rispettivamente di 5 e 30 posizioni

- $W_t$  rappresenta il plaintext del round  $t$ , di grandezza 512 bit.

Consideriamo un messaggio  $m$  (la divina commedia) di lunghezza  $n$  ( $n$  lunghezza arbitraria), il messaggio  $m$  è diviso in  $k$  blocchi di grandezza 512, questi 512 bit sono divisi in blocchi  $W_j$  da 32 bit ciascuno.

In particolare se il messaggio è lungo 512 bit, questo ci produrrà 16 blocchi che rappresentano rispettivamente un valore di  $W_j$  da 32 bit ( $16 * 32 = 512$ ).

*Ma io ho bisogno di fare 79 rounds, e ne ho solo 16, cosa faccio?*

```
Message schedule: extend the sixteen 32-bit words into eighty 32-bit words:
for i from 16 to 79
    Note 3: SHA-0 differs by not having this leftrotate.
    w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1
```

Dal round 16 in poi, quindi, vengono presi 3 blocchi, e XORati tra di loro, per poi effettuare un *leftrotate*

- $K_t$  è una costante, valore che dipende dal numero di round in cui ci troviamo

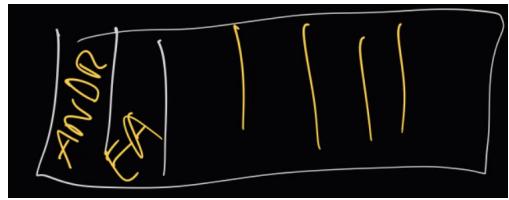
```
Main loop:[3][57]
for i from 0 to 79
    if 0 ≤ i ≤ 19 then
        f = (b and c) or ((not b) and d)
        k = 0x5A827999
    else if 20 ≤ i ≤ 39
        f = b xor c xor d
        k = 0x6ED9EBA1
    else if 40 ≤ i ≤ 59
        f = (b and c) or (b and d) or (c and d)
        k = 0x8F1BBCDC
    else if 60 ≤ i ≤ 79
        f = b xor c xor d
        k = 0xCA62C1D6
```

- Infine svogliamo una semplice somma per rendere SHA1 immune ad attacchi semplici:

```
Add this chunk's hash to result so far:
h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e
```

## SHA1 — password hashing

Consideriamo di voler hashare una password, queste essendo solitamente corte potremmo trovarci in una situazione in cui non riusciamo a riempire neanche il primo blocco  $W_k$ :



Utilizziamo quindi un padding, in particolare vengono aggiunti un bit di 1 ed  $n$  bit 0, in particolare  $n$  fino al 448 bit. Nei rimanenti 64 bit inseriamo:

ANSWER = 48 bit

In particolare il 5to ed il 4to bit da destra vengono settati ad 1 in quanto la lunghezza della parola Andrea è  $48bit = 32 + 16 = 2^5 + 2^4$ , ed i rimanenti bit sono settati a 0, fino a 64.

44868

Attacco a SHA1 — SHAttered: first public collision

On 23 February 2017, the CWI (Centrum Wiskunde & Informatica) and Google announced the SHAttered attack, in which they generated two different PDF files with the same SHA-1 hash in roughly  $2^{63.1}$  SHA-1 evaluations. This attack is about 100,000 times faster than brute forcing a SHA-1 collision with a birthday attack, which was estimated to take  $2^{80}$  SHA-1 evaluations. The attack required "the equivalent processing power of 6,500 years of single-CPU computations and 110 years of single-GPU computations"

## Attacchi a funzioni hash

### Paradosso del compleanno

Utilizzato per attaccare le funzioni hash e le firme digitali, il paradosso afferma che la probabilità che almeno due persone in un gruppo compiano gli anni lo stesso giorno è largamente superiore a quanto potrebbe dire l'intuito: infatti già in un gruppo di 23 persone la probabilità è circa 0,51 (51%); con 30 persone essa supera 0,70 (70%), con 50 persone tocca addirittura 0,97 (97%).

Calcoliamo la probabilità secondo la quale 23 non compiano gli anni lo stesso giorno:

$$PND = (1 - 1/365) * (1 - 2/365) * \dots * (1 - 22/365)$$

Se ora calcoliamo

$$1 - PND$$

Possiamo calcolare la probabilità di collisione.

Se  $PND$  è 0,493 allora  $1 - PND = 0,507$ , ovvero c'è una probabilità del 50% che due persone abbiano lo stesso compleanno. In particolare la probabilità è 70% per 30 persone, 89% per 40 persone. La probabilità della collisione è approssimabile con

$$1 - e^{-r^2/2N}$$

ove  $N$  = dimensione del campione (365)  
e  $r$  = dimensione scelta (23, 30, 40)

Un altro esempio applicabile è l'analisi delle targhe delle automobili, quant'è il numero di macchine che dobbiamo esaminare perché due abbiano la porzione delle cifre uguale?

### Consideriamo con le funzioni hash

$N$  = è il nostro digest

$r$  = numero di hash che dobbiamo calcolare per trovare una collisione

Questo tuttavia non ci permette di attaccare SHA1, in quanto gli hash che calcoliamo noi sono randomici. Per attaccare SHA1 effettuiamo un attacco simile a baby step giant step, calcolando

$$P = 1 - e^{-\lambda}$$
$$\lambda = r^2/N$$

## Firma digitale

Consideriamo un contratto con messaggio  $m$ , effettuiamo:

$$m \rightarrow \text{hash}(m) \rightarrow \text{FirmaRSA}(\text{hash}(m))$$

Il crittografo in questo caso deve difendersi sia dall'attacco a  $\text{hash}(m)$  che da attacchi a  $\text{FirmaRSA}(\text{hash}(m))$ . Andiamo quindi a firmare l' $\text{hash}(m)$  con RSA, consideriamo:

$$\begin{aligned} \text{Public: } & (n, e) \\ \text{Private: } & (p, q, d) \end{aligned}$$

E procedura di encryption  $m^e = c$ , e procedura di decryption  $c^d = m$ , andiamo ad applicare una variante. Consideriamo che  $d$  è la chiave privata, posseduta solo dal proprietario. Consideriamo procedura di encryption  $m^d = c$  e procedura di decryption  $c^e = m$ .

$$\begin{aligned} m \rightarrow \text{hash}(m) = h \\ h^d \rightarrow \text{firma} \end{aligned}$$

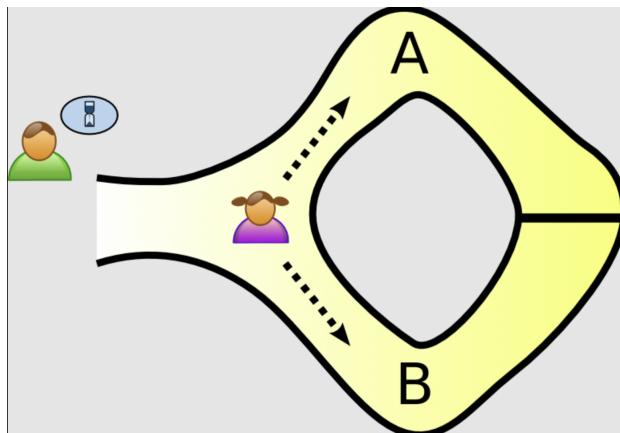
### MSG recovery schemes RSA — Signatures with appendix El Gamal

*TBD to be completed*

$$\text{El - Gamal : } m, \text{sig}(h(m))$$

### Funzioni "zero-knowledge"

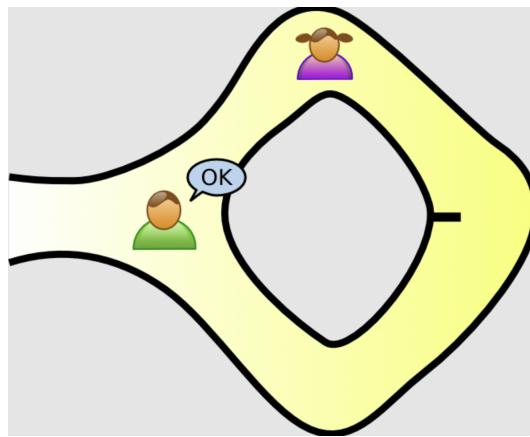
It is common practice to label the two parties in a zero-knowledge proof as Peggy (the prover of the statement) and Victor (the verifier of the statement).



In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a ring, with the entrance on one side and the magic door blocking the opposite side. Victor wants to know whether Peggy knows the secret word; but Peggy,

being a very private person, does not want to reveal her knowledge (the secret word) to Victor or to reveal the fact of her knowledge to the world in general.

They label the left and right paths from the entrance A and B. First, Victor waits outside the cave as Peggy goes in. Peggy takes either path A or B; Victor is not allowed to see which path she takes. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Providing she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path.



However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path by which she had entered. Since Victor would choose A or B at random, she would have a 50% chance of guessing correctly. If they were to repeat this trick many times, say 20 times in a row, her chance of successfully anticipating all of Victor's requests would become vanishingly small (1 in 2<sup>20</sup>, or very roughly 1 in a million).

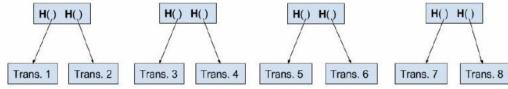
Thus, if Peggy repeatedly appears at the exit Victor names, he can conclude that it is extremely probable that Peggy does, in fact, know the secret word.

## Block-chaining

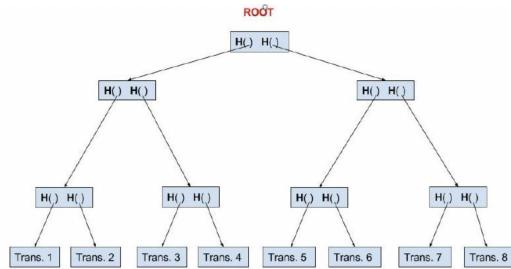
<https://www.finriskalert.it/wp-content/uploads/visconti.pdf>

## Merkle-Tree

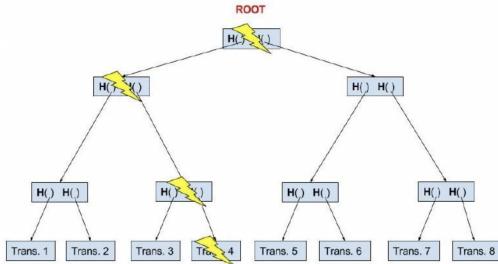
Consideriamo una serie di transazioni, da 1 a 8, calcoliamo i loro hash, e li raggruppamo a 2 a 2.



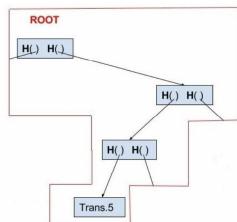
I dati dei due puntatori hash vengono combinati, e utilizzati per calcolare un nuovo puntatore hash, interando progressivamente arriviamo prima o poi a una radice.



Questo albero è chiamato un Merkle tree. Il Merkle Tree permette di verificare una potenziale manomissione, se per esempio un attaccante modifica una transazione, o un nodo intermedio, il valore della radice cambia.



Un'altra proprietà del Merkle Tree è che ci permette di verificare l'appartenenza. Se vogliamo per esempio verificare pubblicamente che una transazione sia presente all'interno di un albero, conoscendo solo la radice, mostriamo i nodi dalla radice alla transazione.



Questo ci permette di mostrare l'appartenenza all'albero, senza mostrare il valore della transazione. Per effettivamente mostrare la proprietà di una transazione viene utilizzata la firma digitale.

**Protocolli**

**Curve ellittiche**