

Crittografia

Programma:

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

Cifratura simmetrica e cifratura asimmetrica

L'obiettivo fondamentale della crittografia è permettere a due persone, solitamente chiamate Alice e Bob di comunicare su un canale di comunicazione insicuro, facendo in modo che un attaccante, solitamente chiamato Eve, possa intercettare il messaggio.

I canali di comunicazione possono essere molteplici (internet, telefono etc.). Il messaggio inviato da Alice è chiamato plaintext, un testo in inglese di struttura arbitraria. Alice cifra il messaggio utilizzando una chiave, e spedisce il ciphertext lungo il canale di comunicazione a Bob, che decifra il messaggio e ricostruisce il plaintext.

I cifrari che andiamo a vedere sono suddivisi in cifrari:

- simmetrici, in cui la chiave di cifratura è uguale alla chiave di decifrazione. I cifrari simmetrici vengono utilizzati per cifrare grosse quantità di dati (eg. testo, *des* e *aes*)
- asimmetrici, in cui la chiave di cifratura è diversa dalla chiave di decifrazione. I cifrari asimmetrici vengono utilizzati per cifrare piccole quantità di dati (eg. pw, *rsa*, *elgamal*)

Dobbiamo infine accennare le tecniche steganografiche, che sono tecniche di offuscamento del messaggio

Sicurezza di un cifrario

Uno dei principali obbiettivi di un cifrario è sicuramente la sicurezza, ma come facciamo a quantificarla? Generalmente possiamo dire che un cifrario è sicuro se un attaccante non riesce a decifrare il messaggio senza una chiave.

Uno dei principi che i cifrari seguono maggiormente è il principio di Kerkhoff.

Principio di Kerkhoff

Il principio di Kerkhoff ci dice di assumere che l'attaccante conosca sempre l'intero schema/protocollo e la chiave pubblica, ma non la chiave privata (security through obscurity non funziona).

Brutalmente: se un cifrario rimane pubblico per 30 anni, ma non viene bucato, vuol dire che è resiliente

Scenari di attacco

I principali modelli di scenario di attacco sono:

- known ciphertext only, in cui l'avversario conosce una porzione del cyphertext
- known plaintext, in cui l'avversario conosce una porzione del plaintext, e ha accesso al relativo ciphertext
- chosen plaintext, in cui l'attaccante è in grado di scegliere un plaintext, e possiede il relativo ciphertext
- chosen cypher-text, in cui l'attaccante è in grado di scegliere il ciphertext, ed è poi fornire del plaintext

Chiaramente gli attacchi chosen plaintext e chosen ciphertext sono gli attacchi che forniscono più informazioni ad un potenziale attaccante.

Le proprietà delle funzioni crittografiche

Vi sono una serie di proprietà che devono garantire delle funzioni crittografiche perché siano considerabili, ovvero:

- confidenzialità, o segretezza, ottenuta attraverso l'operazione di cifratura prima della spedizione
- integrità dei dati, il messaggio non viene alterato applicando una funzione crittografica, se il risultato è medesimo nel caso del mittente e del destinatario la cifratura è considerata integro

- autenticazione, che garantisce che solo gli interessati siano in grado di accedere alle informazioni
- Non ripudio, che garantisce che l'autore di una dichiarazione non potrà negare la paternità e la validità della dichiarazione

Cifrari storici

Passiamo ora a vedere una serie di cifrari storici, utilizzando lungo la storia, le caratteristiche ed eventuali vulnerabilità

Cifrario di Cesare o Shift Cipher

Il cifrario di Cesare basato sulla modularità aritmetica:

congruenza modulo n

Consideriamo a e b valori interi, ed m un intero positivo, possiamo scrivere

$$a \equiv b \pmod{m}$$

se m divide $b - a$

La frase $a \equiv b \pmod{m}$ è chiamata una congruenza, e si dice che " a è congruente a b modulo m ", il valore m è chiamato modulo

Il cifrario di cesare può quindi essere definito nel seguente modo:

Cryptosystem 2.1: Shift Cipher

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. For $0 \leq K \leq 25$, define

$$e_K(x) = (x + K) \pmod{26}$$

and

$$d_K(y) = (y - K) \pmod{26}$$

$(x, y \in \mathbb{Z}_{26})$.

Il cifrario di cesare lavora quindi associando un valore numerico (solitamente da 0 a 25, siccome vi sono 26 caratteri nella lingua italiana), e shiftando l'alfabeto a destra/sinistra per k valori, ove k è la chiave. Il processo di decifrazione avviene shiftando k valori indietro le corrispondenze.

Quadrato di Polibio

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Il cifrario basato sul quadrato di Polibio funziona creando una matrice $n * n$, nella quale si inseriscono le lettere all'interno, per ogni corrispondenza si prende la coppia *riga, colonna*. Intuitivamente la lunghezza del ciphertext è il doppio del plaintext, il problema principale con questo cifrario è la presenza di punti fissi.

Una naturale evoluzione

La presenza di punti fissi ci porta a cercare un modo di migliorare questi cifrari, in particolare un approccio migliore è passare da espressioni

$$y = x + k \rightarrow y = \alpha x + \beta$$

con $\alpha \neq 0, \beta \neq 0$

Per ricavare un valore dobbiamo fare

$$\frac{(y-\beta)}{\alpha} = x$$

È inoltre importante che $\text{MCD}(\alpha, 26) = 1$ ovvero che α sia invertibile, e che α e 26 siano *coprimi*

Attacco ai cifrari affini

Come attacchiamo i cifrari affini? Andiamo ad analizzare le frequenze

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

codifica della lettera R con chiave V

Il quadrato di Vigenère lavora nel seguente modo: il mittente ed il destinatario concordano una parola, chiamata chiave. La chiave è ripetuta n volte fino a ricavare una corrispondenza 1-1 con la lunghezza del messaggio da cifrare.

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 *cyphertext*.

Key: **ABCDABCDABCDABCDABCDABCDABCDABCD**
 Plaintext: **cryptoisshortfor**cryptograph**y**
 Ciphertext: **CSASTPKVSIQUTGQU**CSASTPIUAQJB

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 crittotesto, 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. Si utilizza una parola come chiave e le lettere sono inserite saltando i doppioni, mettendo in testa la chiave.

P	L	A	Y	F _A
I	R	E	X _A	M _{PLE A}
B	C	D _{EF}	G	H _{I=J}
K _{LM}	N	O _P	Q _R	S
T	U	V	W _{XY}	Z

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

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

HI

Shape: Rectangle
Rule: Pick Same Rows,
Opposite Corners

BM

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).

Consideriamo il seguente alfabeto con corrispondenti numeri

Letter	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
Number	0	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

Per codificare un messaggio, ogni blocco di n lettere è moltiplicato per una matrice $n * n$ invertibile, modulandolo a 26. per decifrare il messaggio ogni blocco è moltiplicato per l'inverso della matrice utilizzata durante la fare di cifratura. Non tutte le matrici chiavi sono ammissibili, la matrice deve esser invertibile per permettere il decifraggio da parte del destinatario. In particolare $MCD(\Delta, 26) = 1$.

Consideriamo il messaggio *ACT*, con chiave *GYB/NQK/URP*

$$\begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix} \quad \begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$$

Il processo di cifratura avviene così:

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix} = \begin{pmatrix} 67 \\ 222 \\ 319 \end{pmatrix} \equiv \begin{pmatrix} 15 \\ 14 \\ 7 \end{pmatrix} \pmod{26}$$

ove 15, 14, 7 corrisponde a POH

E la fase di decifratura avviene invertendo la matrice iniziale modulo 26, e moltiplicandolo per il ciphertext:

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}^{-1} \pmod{26} \equiv \begin{pmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{pmatrix} \begin{pmatrix} 15 \\ 14 \\ 7 \end{pmatrix} = \begin{pmatrix} 260 \\ 574 \\ 539 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix} \pmod{26}$$

che ci restituisce nuovamente ACT

ADFGX cipher

Utilizzata dai generali tedeschi durante la prima guerra mondiale, viene creata una matrice 5 x 5 in cui vengono inserite le lettere arbitrariamente.

	A	D	F	G	X
A	b	t	a	l	p
D	d	h	o	z	k
F	q	f	v	s	n
G	g	i/j	c	u	x
X	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 crittotesto prendiamo le coppie degli indici (riga, colonna), ogni lettera quindi è data da una coppia di indici.

Viene poi riordinato il crittotesto secondo una chiave di lunghezza k . Il crittotesto 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.

C	A	R	G	O
A	F	A	D	A
D	A	F	G	F
D	X	A	F	A
D	D	F	F	X
G	F	X	F	

FAXDF ADDDG DGFFF AFAXX AFAF

La codifica di CARGO

A	C	G	O	R
F	A	D	A	A
A	D	G	F	F
X	D	F	A	A
D	D	F	X	F
F	G	F	X	

NIST / NBS

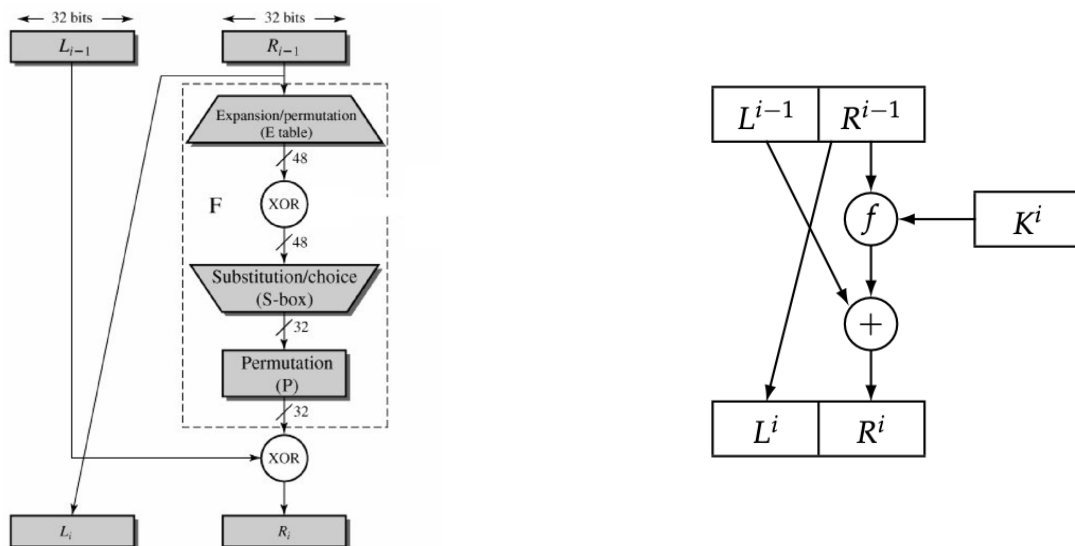
Il NIST è un'associazione governativa americana che si occupa di emettere bandi per richiedere la creazione di cifrari per il pubblico. In questo momento i bandi aperti sono: "Post-quantum competition" del 2018 ed il "Lightweight cryptography" per sistemi leggeri. Nel 1970 manda un bando per creare un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

Data Encryption Standard — DES

E' un cifrario simmetrico 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 obbiettare.

- grandezza dei blocchi di plaintext: 64
- grandezza delle chiavi: 56 (64 - 8)
- grandezza dei ciphertext: 64

Per cifrare un messaggio con DES vi sono 3 principali sali, l'initial permutation o IP, Feistel permutation, e la final permutation FP. IP e FP non hanno valore crittografico, ma furono inseriti per facilitare il caricamento dei blocchi all'interno dei sistemi a 8 bit del 1970.

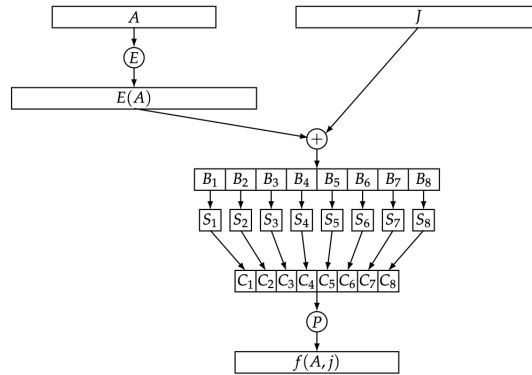


Ma dove entra in gioco la chiave?

La chiave è fornita alla funzione di Feistel, ed 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 di espansione

La funzione E rappresenta la permutazione di espansione, che aggiunge 16 bit ai 32 forniti in precedenza. Questi 48 bit sono utilizzati per effettuare uno XOR con J (la chiave del round), che ci permette di ricavare 8 blocchi. Il valore in output dipende da una tabella prefissata. Questi 8 blocchi (formati da 6 bit ciascuno) sono permutati nuovamente (all'interno di S_k).

S-Box

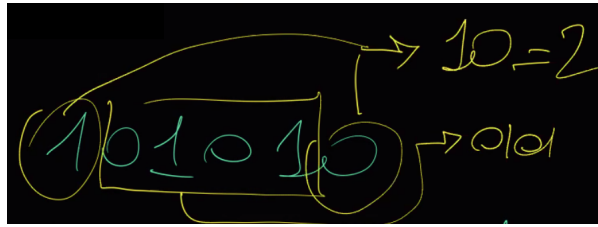
Le funzioni S_k , rappresentate dalle S-Box, eseguono un'operazione di compressione-permutazione: prendono in bit 6 bit ciascuno, ma restituiscono 4 bit, portando l'output a 32 bit. Questi 32 bit rappresentano l'output. S_k è formato da una matrice (4×16) sulla quale sono inseriti una permutazione dei valori tra 1 a 16 (fissa).

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

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

un esempio di S-Box

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.



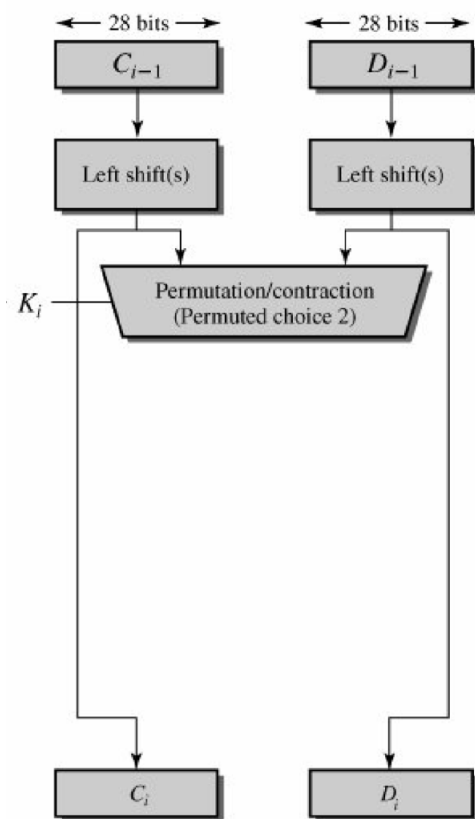
Si prende la terza e la sesta perché il binario 00 è la prima colonna, noi contiamo da 0

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.

00	14	4	13	12	15	1	8	3	10
01									
10				6					
11									

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

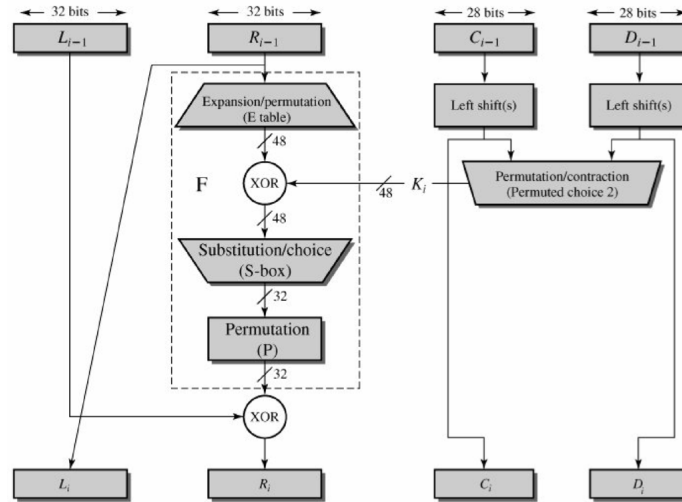
La chiave K_i



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 . Questi sono shiftati a sinistra, ed il loro valore dipende dal numero di round.

$$\begin{aligned} C_i &= \text{LS}(C_{i-1}) \\ D_i &= \text{LS}(D_{i-1}) \end{aligned}$$

Il valori shiftati vengono utilizzati come input per una permutazione/contrazione, passando da 56 a 48. L'output di questa permucontrazione 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.

Attacco a DES — Crittoanalisi differenziale

L'attacco a DES è un attacco chiamato di analisi differenziale, è un attacco deterministico se il numero di round è ≤ 3 e diventa probabilistico dai 4 in su dovuta alla presenza di round di crittografia (round di DES).

Doppio DES e triplo DES

Doppio DES e triplo DES sono tecniche di cifratura direttamente derivate da DES, sappiamo che DES utilizza una chiave di 56 bit per cifrare un qualsiasi plaintext, per rafforzare ulteriormente l'algoritmo è stato pensato di raddoppiare le operazioni di cifratura applicando DES più volte.

Rispettivamente doppio DES utilizza una chiave da 112 bits, e triplo DES una chiave da 168 bits. L'efficacia di applicazione di questi algoritmi dipende strettamente dalla scelta della chiave, in particolare se la chiave scelta è la medesima, ovvero $k_1 == k_2$, la sicurezza fornita non è di $2n$ la lunghezza della chiave, ma di $n + 1$. In particolare attraverso un attacco "meet in the middle" un attaccante può ricavare la coppia di chiavi in 2^{n+1} passi, piuttosto che 2^{2n} .

Per ovviare a questo problema, quindi, sono state definite una serie di linee guida per la selezione delle chiavi, in particolare:

- Le 3 chiavi sono indipendenti, chiamato anche 3TDEA, è l'approccio migliore, che fornisce il grado di sicurezza maggiore, ovvero $3 \times 56 = 168$ bit di chiave.
- Le chiavi k_1 e k_2 sono indipendenti, e la chiave $k_3 = k_1$, chiamato anche 2TDEA, fornisce una chiave id lunghezza inferiore, ed è un compromesso accettabile. Il NIST ha già deprecato questa opzione
- Le chiavi k_1, k_2, k_3 sono identiche, come abbiamo detto in precedenza, l'adozione di questo approccio è fortemente scoraggiato in quanto non fornisce granché sicurezza aggiuntiva rispetto al DES tradizionale.

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 possiamo trasformare la funzione di encryption in:

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

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.

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

Vediamo 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 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.

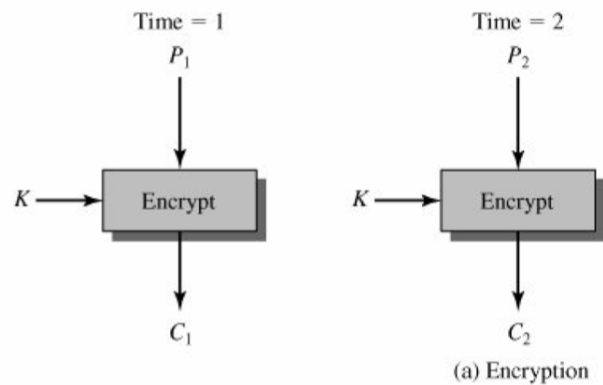
Dato:

$$P = [p_1, \dots, p_n] \text{ l'insieme delle porzioni di grandezza } n$$
$$C = [c_1, \dots, c_n] \text{ l'insieme dei ciphertext di grandezza } n$$

attraverso la funzione

$$f(P_i; DES) = C_i$$

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

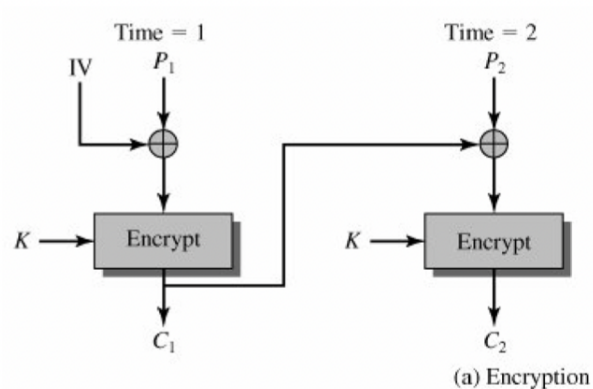


Il problema con ECB è che se più blocchi di plaintext sono uguali, il ciphertext prodotto è uguale. Questo permette su messaggi molto lunghi di essere attaccati attraverso una crittoanalisi di frequenza

CBC, o "cipher block chaining mode":

Per mitigare le falle 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 il plaintext. IV è sostituito dal ciphertext del round $P_i - 1$ dalla fase 2 in poi.



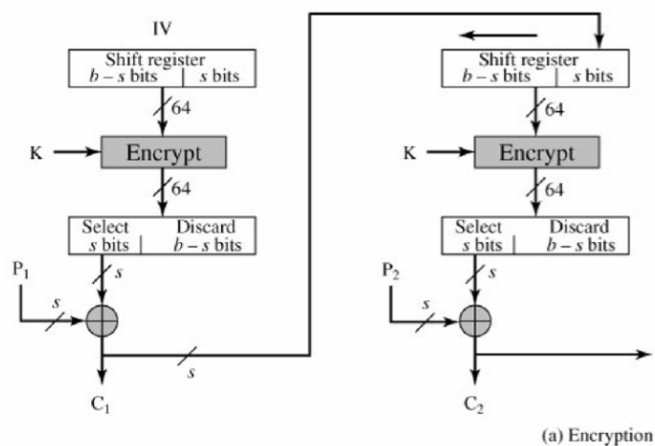
In parole povere introduciamo una chiave IV da applicare in XOR con la prima fase di encryption. Dalla seconda fase in poi l' IV è sostituito con il ciphertext del round $P_i - 1$.

The underlying problem:

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 . $n = 8$ bits dell'output sono utilizzati come input per un'operazione di XOR con il plaintext, che è stato in precedenza suddiviso in blocchi da 8 bits. 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)*

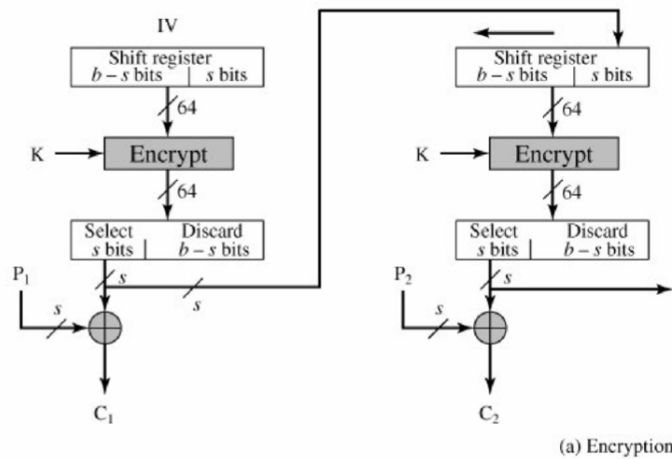
In questa modalità la grandezza dei blocchi del plaintext è s (di Select s -bits).

Se $s = 1char(8bits)$, 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 non venga espulso dall'IV.

OFB, o "output feedback mode":

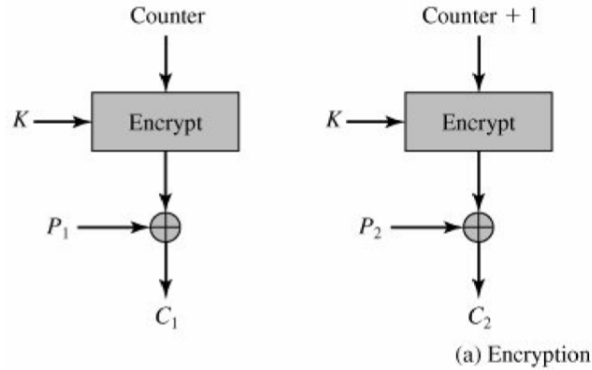
Questo schema di encryption è molto simile a CFB, ma che risolve il problema di propagazione di errore di trasmissione. Il feedback, che in CFB proveniva dopo allo XOR con il plaintext, viene piuttosto estratto in locale, prima dello XOR.



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.

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

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

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:

$m(x) * q(x) + r(x)$, e sul campo di Galois la rappresentazione di $m(x)$ 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 $m(x)$

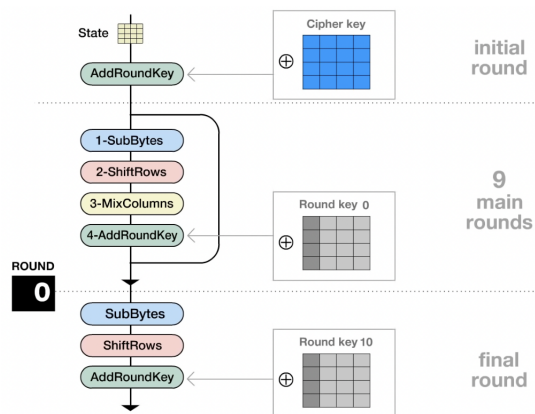
AES — Rijndael cipher

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 (10 round), 192 (12 round) e 256 (14 round) 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.

$Round_n$

Le operazioni all'interno del $round_n$ 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)

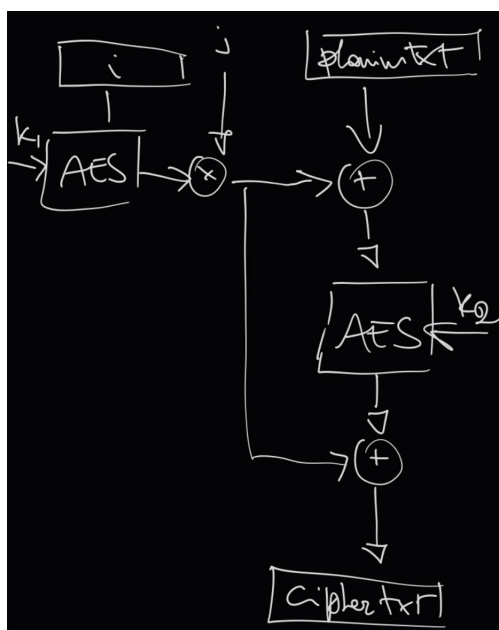
Il valore ricavato in riga 9, colonna 5 è 2A

	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 un byte a sinistra, e la terza viene shiftata di due byte a sinistra e la quarta di 3 byte a sinistra.
- Mix columns: viene moltiplicata una colonna degli stati per una matrice nota, una colonna alla volta.

AES-XTS: meccanismo di cifratura nel disco

AES XTS è la modalità di cifratura adottata per la cifratura dei dati su disco, in particolare adottata per evitare che due dati con lo stesso dato abbiamo la stessa cifratura (che ci espone ad attacchi di tipo analisi di frequenza). Consideriamo lo schema generale della modalità:



- 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.

Padding per l'ultimo blocco

E' talvolta possibile che l'ultimo blocco non sia perfettamente di grandezza 128, 192, 256, in questo caso viene svolta un'operazione speciale per generare il padding.

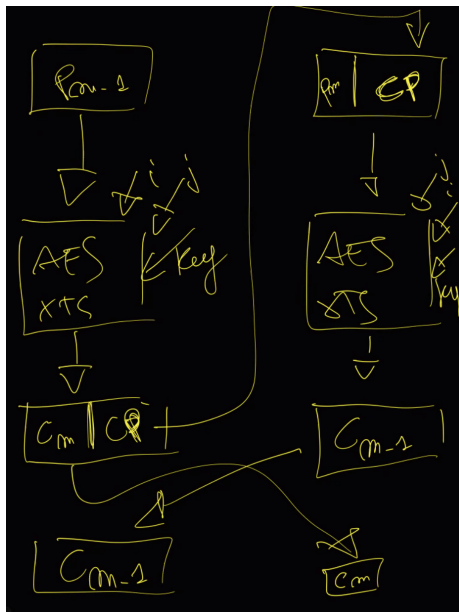
Sia un plaintext diviso in blocchi $P_1, \dots, P_{m-1}, P_m$ con P_m incompleto, in AES-XTS i blocchi $P_1, \dots, P_{m-2}$ sono gestiti normalmente, mentre i blocchi P_{m-1} e P_m sono gestiti in maniera speciale. In particolare:

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

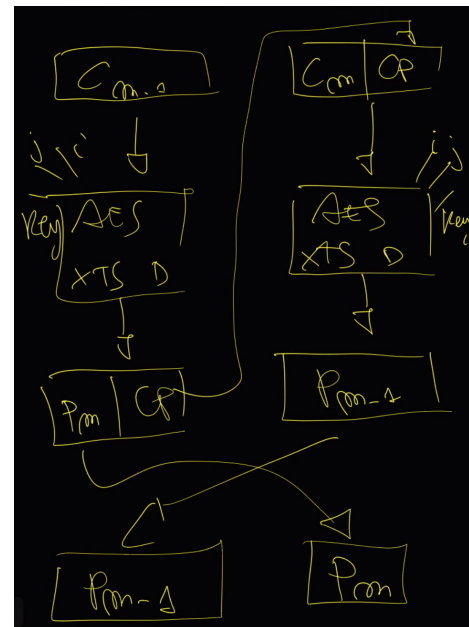
$$AES - XTS(P_{m-1}) = (C_m, CP)$$

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

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



fase di encryption



fase di decryption

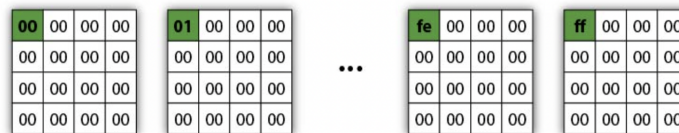
La fase di decryption funziona al contrario (figure 2)

Attacking AES (6 rounds)

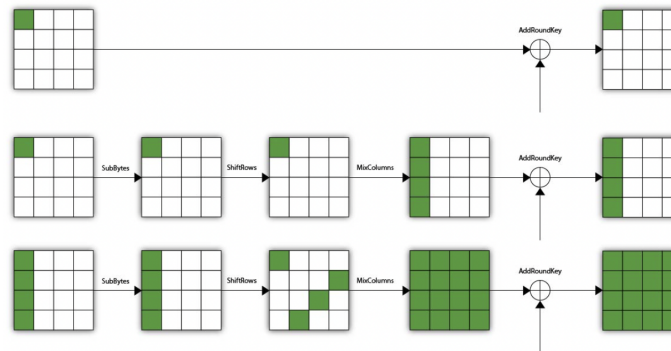
Proprietà del Δ set

Consideriamo un set di 256 plaintexts, riempiti tutti di 0. Se selezioniamo una cella all'interno dei 256 stati, e le permettiamo di assumere tutti i valori possibili in un range di un byte, ovvero da 00 a ff questa viene chiamata proprietà Δ set. $\Delta - SET$ rappresenta l'insieme dei valori da 0 a 256. Una proprietà particolare del $\Delta - SET$ è la proprietà della *somma zero*, che ci garantisce che la somma delle celle in un $\Delta - SET$ sarà uguale a 0.

La proprietà di $\Delta - SET$ è garantita fino al secondo round, ma viene persa al terzo round. In particolare selezioniamo un index che può assumere tutti questi valori una cella attiva.



Consideriamo uno stato iniziale con proprietà Δ set, quando questa viene cifrata utilizzando AES passa all'interno di una serie di trasformazioni, in particolare:



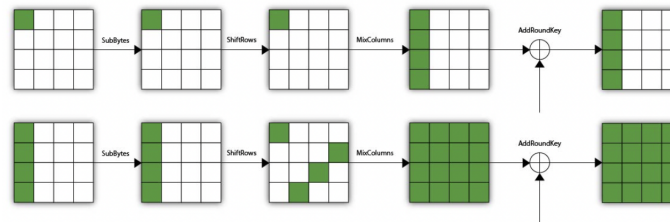
Come possiamo vedere la proprietà di Δ set è mantenuta sia all'operazione di preambolo *AddRoundKey*, che dopo tutto il primo round, fino all'operazione di *MixColumns*. L'operazione di *MixColumns* è particolare in quanto "attiva" tutte le celle nella stessa colonna della cella attiva iniziale.



Quando parliamo di "attivazione di cella" intendiamo che l'effetto della cella inizialmente attiva va ad influire sul valore di tutte le altre, la proprietà di Δ set è quindi presente su 4 celle attive. L'indice di una cella è considerata attiva se la cella può assumere tutti i differenti valori di un byte all'interno di un set di 256 differenti plaintexts. Contrariamente una cella è considerata inattiva se prende lo stesso valore all'interno del set di plaintexts.

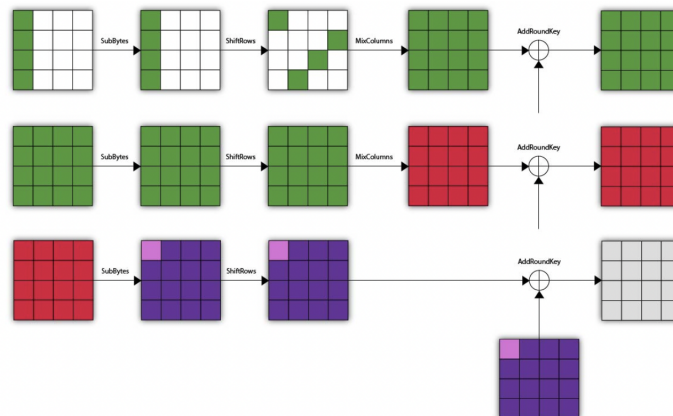
Confusion and diffusion

Se ci concentriamo sulla situazione alla fine del secondo round possiamo vedere che la proprietà di Δ set è ancora presente questa volta tuttavia è presente su tutte le celle.



Possiamo inoltre intuire che dal terzo round di AES la proprietà di Δ set è persa. Possiamo concludere che una variazione nel plaintext o della chiave va ad influire in maniera esponenziale sul ciphertext, questa proprietà è chiamata confusion e diffusion, ovvero che tutto dipende da tutto.

Attacco ad AES



Se eseguiamo un ulteriore round di AES possiamo notare che l'output non possiede la proprietà di Δ set, sorge tuttavia uno scenario di attacco, in particolare possiamo prendere una cella del round di output, ed effettuare un attacco di tipo bruteforce della chiave. Se

il valore mi restituisce di nuovo un Δ set possiamo essere certi di aver ritrovato la porzione di chiave corretta; questa operazione è svolta su tutte le celle, una alla volta, fino a ricavare la chiave intera.

Crittografia asimmetrica

Come abbiamo visto in precedenza, la crittografia asimmetrica differisce dalla crittografia simmetrica in quanto entrano in gioco ulteriori chiavi, ovvero 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

$$\text{Encryption: } E_{pub}(M) = C$$

$$\text{Decryption: } D_{priv}(C) = M$$

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).

L'algoritmo di RSA è suddivisibile in 4 macro-fasi: la generazione della chiave, la distribuzione della chiave, l'encryption e la decryption. La chiave pubblica è conosciuta da chiunque, ed è utilizzata da Bob per cifrare i messaggi mentre la chiave privata è conosciuta solo da Alice e utilizzata per decifrarli. E' un cifrario a chiavi pubbliche ed i principali parametri sono:

- le chiavi pubbliche: $public(n, e)$:
 - (solitamente) $e = 2^{16} + 1 = 65.537$
 - $n = p * q$
- le 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
 - $d * e = 1 \bmod(\phi(n))$ (d segreto)

In particolare il valore $\phi(n)$ è chiamata funzione "phi" di Eulero, e rappresenta l'insieme dei numeri k nel range di $1 \leq k \leq n$ per cui il massimo comune divisore è 1, ovvero

$$k \text{ in } 1 \leq k \leq n \text{ t.c.} \\ \gcd(n, k) = 1$$

Possiamo infine riscrivere ϕ nel seguente modo:

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

- 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

Funzionamento di RSA

1. Generiamo due numeri primi p, q (sulla quale effettuiamo un controllo di adeguatezza applicando gli algoritmi di Euclide esteso, Fermat, Miller Rabin e Pollard) di grandezza simile, arbitrariamente grande per calcolare un numero n tale che $n = p \times q$ sia sufficientemente grande, in genere di grandezza $> 1024bit$.
2. Calcoliamo $n = p \times q$ e $\phi = (p-1)(q-1)$
3. Selezioniamo un valore e , tale che $1 < e < \phi$ e tale che $gcd(e, \phi) = 1$
4. Calcoliamo l'esponente segreto d , tale che $1 < d < \phi$ e tale che $ed \equiv 1 \mod \phi$

la chiave pubblica è (n, e)

la chiave privata è (d, p, q)

Alice per effettuare la fase di encryption:

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

Bob per effettuare la fase di decryption:

$$C^d = M \mod n$$

Il valore e

Il valore e può essere 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:

$$N^e \mod n$$

Il valore $2^{16} = 65537$ in codifica binaria è: 10000000000000001

Possiamo vedere che nella compaiono molti 0, e pochi 1, questo ci permette di effettuare operazioni di esponenziazione modulare più veloce, avendo solo 2 bit con valore 1. 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 bruteforcare

Il valore d

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

$$d = e^{-1} \bmod \phi \text{ oppure} \\ d = \frac{1}{e} \bmod \phi$$

Ovvero l'inverso modulare. Notare come questo non è la divisione classica, ma l'inverso modulare d è definito come valore intero tale che $ed = 1 \bmod \phi$, che esiste sse e, ϕ sono coprimi.

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

- l'algoritmo di Euclide per l'MCD di due valori interi, per verificare se questi sono coprimi. (Utilizzato durante la determinazione di p, q)
- l'algoritmo di Euclide esteso per la determinazione dell'inverso moltiplicativo
- il teorema piccolo di Fermat e la generalizzazione di esso attraverso il teorema di Eulero per l'euler totient ($\phi(n)$).
- Il teorema cinese dei resti per lo scaricamento dello sforzo computazionale su server

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, senza conoscere nulla sui divisori dei due valori. 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 $l'MCD(1180, 482) = 2$

L'algoritmo di Euclide esteso per l'inverso moltiplicativo

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

Come calcoliamo x e y?

Partiamo da x:

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

Per calcolare y invece:

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

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

$$\begin{aligned}
x_0 &= 1 \\
x_1 &= 0 \\
x_2 &= -2(1) + 0 = -2
\end{aligned}$$

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

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

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 decifrate possibili

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

Teorema piccolo di Fermat ed il teorema di Eulero

Il teorema piccolo di Fermat dice che se p è un valore 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 modulo 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 $2^{43210} \pmod{101}$.

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

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

Teorema di Fermat:

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

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

Infine: $1024 \bmod 101 = 14$

Considerazioni su modulo ed esponente

Consideriamo a, n, x, y numeri interi, $n \geq 1$ t.c. $MCD(a, n) = 1$

Dimostriamo che se lavoriamo con le basi stiamo lavorando con $\bmod n$, mentre se stiamo lavorando con gli esponenti lavoriamo con $\phi(n)$.

$$\begin{aligned} &\text{se } x = y \bmod \phi(n) \\ &\text{allora } a^x = a^y \bmod n \end{aligned}$$

In particolare possiamo dire che $x = y \bmod \phi(n)$ se esiste un intero k per cui vale la seguente proprietà:

$$x = y + k\phi(n)$$

Di conseguenza:

$$a^x = a^{y+k\phi(n)} = a^y * (a^{\phi(n)})^k$$

Applicando il teorema di Eulero Fermat possiamo fare la seguente operazione:

$$\begin{aligned} &a^{\phi(n)} = 1 \bmod n \\ &a^y * (a^{\phi(n)})^k \text{ diventa } a^x = a^y * 1^k \bmod n \end{aligned}$$

Ma siccome 1^k è irrilevante, possiamo eliminarlo e rimaniamo con $a^x = a^y$. Concludiamo quindi che $\phi(n) = \phi(p) = p - 1$

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, ϕ) . 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 ϕ possiamo ricavare $p + q$.
Consideriamo ora che $n = p * q$ e $\phi(n) = (p-1)(q-1)$

$$\begin{aligned}
&ax^2 + bx + c \\
\text{ove } c &= p * q = n, \ a = 1, \ b = p + q \\
&x_1 x_2 = c/a \\
&x_1 + x_2 = -b/a
\end{aligned}$$

Possiamo quindi ricavare che

$$\begin{aligned}
&x^2 - x(n - \phi(n) + 1) + n = 0 \\
\text{Se risolviamo con il } \Delta &\text{ possiamo ricavare i valori } x_1 = p \text{ e } x_2 = q
\end{aligned}$$

Casi particolari su d

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

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

Ma cosa succede se non fosse così?

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

Possiamo dire che:

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

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

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 $(d-1)$

Attacchi da conoscere a carattere informativo:

Caso $m/4$ 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

Low exponent attacks

Consideriamo (e, n) ,

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

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

Common modulus attack

Consideriamo:

$$Alice = (e_A, n) \text{ --- } Bob = (e_B, n)$$

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

$$M^{e_A} = C_1 \text{ mod } n$$
$$M^{e_B} = C_2 \text{ mod } n$$

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

$$\text{Crittotesto per Alice: } M^{e_A} = C_1 \text{ mod } n$$
$$\text{Crittotesto per Bob: } M^{e_B} = C_2 \text{ mod } n$$

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

$$(C_1^{-1})^{-r} * (C_2)^s =$$
$$= ((M^{e_a})^{-1})^{-r} * (M^{e_b})^s =$$
$$= M^{e_a r} * M^{e_b s}$$

Noi tuttavia sappiamo che è valida la seguente congruenza: $1 \equiv e_a r + e_b s \text{ mod } \phi(n)$ E possiamo sostituirla all'esponente, rimaniamo quindi con:

$$= M^{e_a r} * M^{e_b s} = \\ m^1 \text{ mod } n \\ m$$

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

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

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

$$array_2 = y^e \text{ mod } n$$

Con x, y che sono gli indici negli array

Handwritten diagram illustrating the Short Plaintext Attack. It shows two arrays of size 10^9 . The first array, labeled $C \cdot x^{-e} \text{ mod } n$, contains elements $c_1, c_2, \dots, c$. The second array, labeled $y^e \text{ mod } n$, contains elements $1^e, 2^e, 3^e, \dots, 1$. The arrays are shown as horizontal boxes with a vertical line on the right indicating the end of the array.

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.

$$C \cdot n^{-e} = y^e$$

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

$$m = n \cdot y$$

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

Padding per short plaintexts — OAEP

Per aggirare 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:

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

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

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

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 aggirare 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 la conformità di 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

$$b_0 = a^m \pmod{n}$$

Se b_0 è uguale a

- ± 1 allora n è probabilmente primo
- altro, allora seguiamo 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 seguiamo 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

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

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:

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

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 $mod\ n$.

$$123456^2 \equiv x \mod n$$

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 $mod\ n$, più valori scegliamo, più precisa sarà la nostra stima. Ad esempio scegliamo 9398, 19095, 1964, 17078

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

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 .

$$x^2 \equiv y^2 \bmod n$$

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

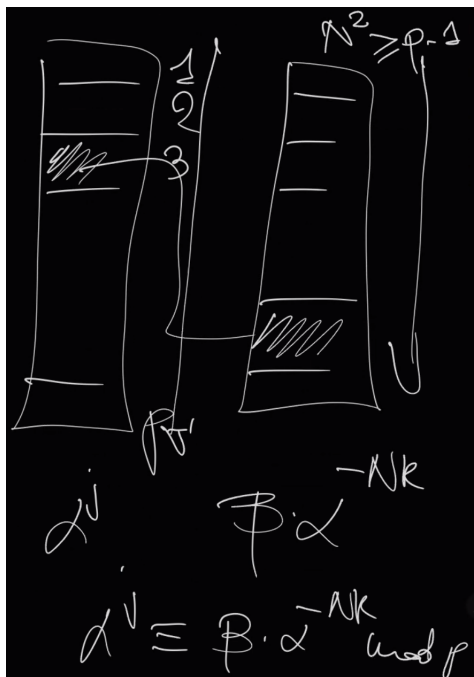
$$\alpha^x \equiv \beta \bmod p$$

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 α_j nel primo vettore, $\beta\alpha^{-Nk}$ nel secondo vettore. Cerchiamo poi due valori comuni, per poi effettuare l'uguaglianza.



In particolare l'uguaglianza

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

E' uguale a

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

Ricordiamo che:

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

Index calculus

E' una variante del quadratic sieve basato su

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

Consideriamo

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

Consideriamo a^k così definito:

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

Allora il passo computazionale è il seguente:

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

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

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

Possiamo esporre quello che ci serve:

$$\log_\alpha \beta = -r + \sum b_i \log_\alpha p_i \mod p-1$$

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:

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

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

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

$$\begin{aligned}
& \dots \\
2^1 4 &= 3^2 \text{ mod } 131 \\
& \dots \\
2^3 4 &= 3 * 5^2 \text{ mod } 131 \\
& \dots
\end{aligned}$$

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

Siccome stiamo introducendo i logaritmi passiamo a $\text{mod } p - 1$

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

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

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

Continuiamo con $\beta * \alpha^r = \Pi p_i^{b_i} \text{ mod } p$

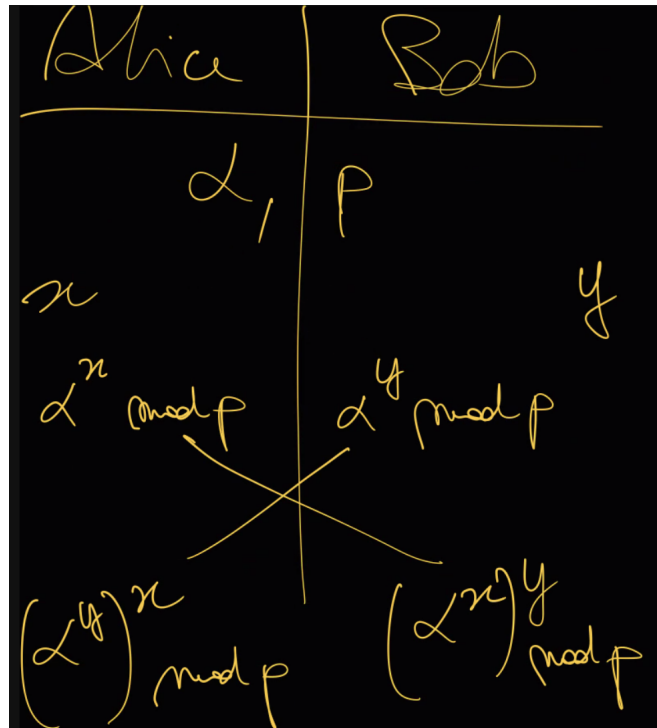
$$\begin{aligned}
37 * 2^{43} &= 3 * 5 * 7 \text{ mod } 131 \\
\log_2 37 + 43 \log_2 2 &= \log_2 3 + \log_2 5 + \log_2 7 \\
\log_2 37 &= -43 + \log_2 3 + \log_2 5 + \log_2 7 \\
x &= -r \sum b_i \log_a p_i
\end{aligned}$$

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 α, p due valori pubblici, e le due chiavi x, y possedute da Alice e Bob, come possono scambiare le chiavi su un canale insicuro?



Vulnerabilità di Diffie Hellman

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

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

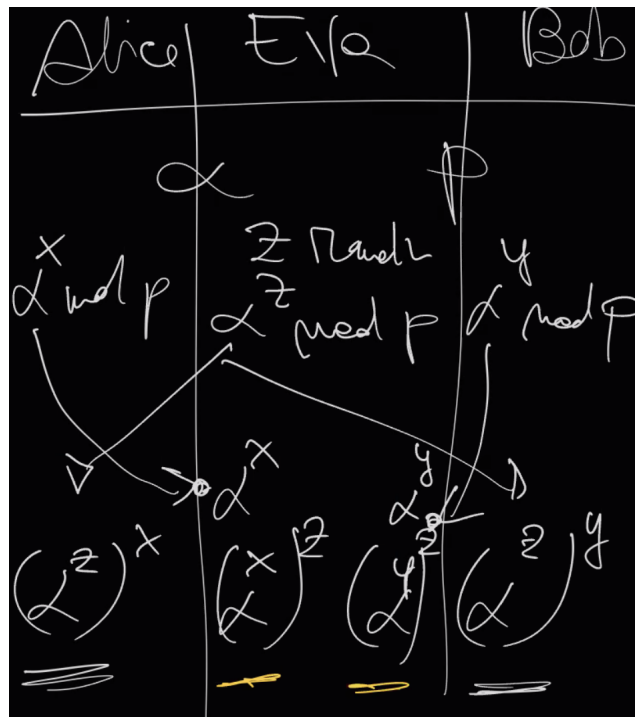
Consideriamo Eve che riesce a fare un attacco man in the middle:

Eve calcola un $\alpha^z \mod p$
 Eve intercetta α^x, α^y

Eve procede a inoltrare il valore tampered α^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 aggirare questo problema è stato implementato STS, un sistema di firme digitali per garantire l'autenticità

Bit commitment

Un'altra applicazione del logaritmo discreto l'utilizzo del bit commitment, o schema di commitment. Questo ci permette di mostrare il possesso di un dato valore senza mostrarlo, nascondendo l'informazione all'interno del problema del logaritmo discreto. Consideriamo

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

Se per esempio voglio inserire un'informazione all'interno del problema del logaritmo discreto, seleziono una posizione particolare all'interno di x , per esempio il valore del bit a posizione 1024, e gli attribuisco valore 0, se un tennista vince, e 1 se in tennista perde.

Inoltrando all'altra parte il valore β , possiamo essere certi di due fatti:

- io non sono in grado a posteriori di modificare il valore, in quanto andrebbe a cambiare fondamentalmente il problema.
- in seguito all'evento l'altra parte può andare a verificare la posizione 1024 del valore x , dopo avergli fornito il valore α , e verificare la correttezza

- l'operazione è svolta più volte, per garantire che non sia un colpo di fortuna. Funziona similmente alle funzioni zero knowledge.

ElGamal

E' un algoritmo di cifratura a chiave asimmetrica. Il funzionamento dell'algoritmo può essere suddiviso nelle fasi di computazione delle componenti pubbliche, pubblicazione delle componenti pubbliche, passo di cifratura, e passo di decifratura.

Computazione delle componenti pubbliche di Bob

Bob, prima di poter utilizzare ElGamal deve selezionare una serie di parametri, in particolare:

1. α radice primitiva
2. ϕ o p numero primo
3. a numero segreto

Con questi, Bob calcola:

$$\alpha^a \equiv \beta \mod p$$

più in particolare Bob calcola β , in quanto α e ϕ li possiede

I dati $(\alpha, \phi/p, \beta)$ rappresentano la componente pubblica di Bob.

Passo di cifratura da parte di Alice

Consideriamo ora Alice con un messaggio m , per cifrare il proprio messaggio:

1. scarica le informazioni pubbliche di Bob (α, p, β)
2. genera un valore k casuale segreto. A partire da k calcola r, t , in particolare:

$$\begin{cases} r = \alpha^k \mod p \\ t = \beta^k * m \mod p \end{cases} \quad (1)$$

3. La coppia (r, t) rappresenta il messaggio cifrato da spedire a Bob

Passo di decifratura da parte di Bob

Bob per decifrare effettua:

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

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

Dimostrazione di correttezza di ElGamal

Consideriamo:

$$t * r^{-a} =$$

Come abbiamo detto in precedenza $t = \beta^k * m \bmod p$ e $r = \alpha^k \bmod p$, possiamo quindi riscrivere l'equazione nel seguente modo:

$$(\beta^k * m)(\alpha^k)^{-a} =$$

Ma abbiamo pure detto che $\alpha^a = \beta$, possiamo quindi riscrivere l'equazione nel seguente modo:

$$[(\alpha^a)^k * m] * (\alpha^k)^{-a} =$$

Ma siccome per le proprietà delle potenze possiamo eliminare le α , rimaniamo con:

$$m \bmod p$$

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 ed i due messaggi derivanti sono:

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

Siccome r è una componente pubblica, l'attaccante può vedere senza problemi che Alice ha utilizzato un k uguale per entrambi i messaggi, in quanto i valori r sono uguali. 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 quindi 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:

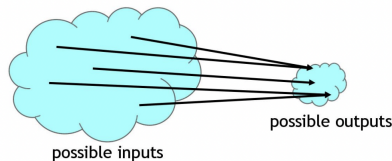
- la facilità di computazione
- l'input può essere una stringa di una qualsiasi dimensione
- l'output ha una dimensione fissa, specifica per ogni funzione hash

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 *collisione*.

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

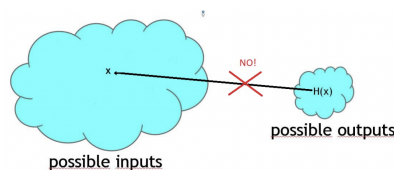


Se vengono computati i valori hash di 2^{130} stringhe, la probabilità di trovare 2 valori con lo stesso output hash è del 99%. E' tuttavia un valore così astronomicamente alto che è possibile considerarla nulla.

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

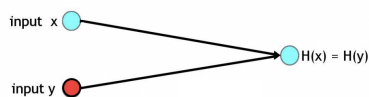
$$H(x) = H(y) \rightarrow x = y$$

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 hasharlo con $H(x)$

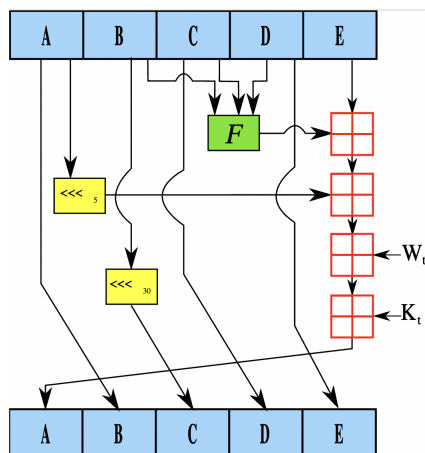
Second pre-image resistance



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

- $\lll$ 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 *XOR*ati 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:

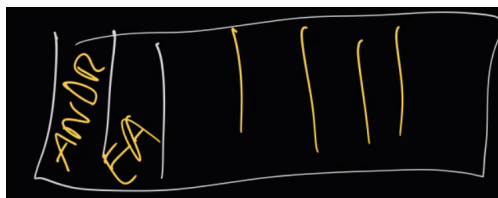
```

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:

$$\text{ANDREA} = 48 \text{ bit}$$

$$32 + 16$$

$$2^5 + 2^4$$

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

$$512 \text{ 64 bit}$$

On the weakness of PBKDF2 — LUKS

Consideriamo la modalità con cui vengono salvate le password all'interno dei sistemi Linux. La definizione della funzione di hashing è importante in quanto deve trasformare una sequenza di caratteri generalmente corta, e quindi senza molta entropia, in una chiave crittografica, in grado di resistere ad attacchi di tipo brute force e dictionary.

Entrano quindi le PBKDF, ovvero le password based key derivation functions, che data una password mi derivano un qualcosa.

PBKDF2

Password Based Key Derivation Function version 2 è una key derivation function sviluppata da RSA laboratories, introduce operazioni CPU intensive, basate su un approccio iterativo pseudorandomico (ad esempio funzioni hash, cipher o HMAC).

Una proprietà importante delle key derivation function è la "cycle free", ovvero la non-presenza di cicli determinabili o non, in quanto un potenziale attaccante se fosse in grado di riconoscerne, sarebbe in grado di saltare tale lavoro.

HMAC

Il meccanismo che entra in gioco è l'HMAC o Hash-Based Message Authentication Code è un algoritmo che ci permette di generare salvare password in maniera sicura, facendo un doppio utilizzo della funzione hash. Consideriamo:

- H : una qualsiasi funzione crittografica
- K : la chiave segreta
- $text$: l'informazione da cifrare
- $ipad$ e $opad$ due valori costanti, rispettivamente $0x36e0x5C$ ripetuti 64 volte che vengono XORati con la password

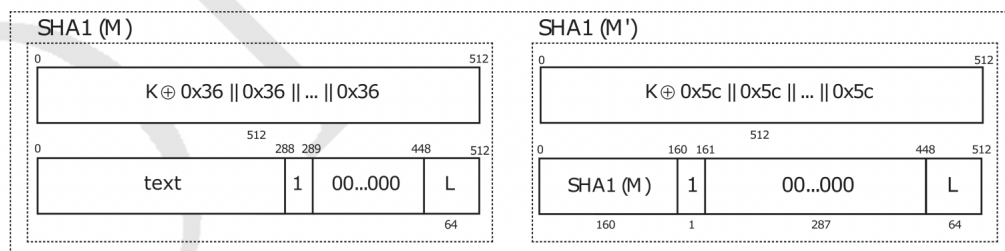


Fig. 1. HMAC-SHA-1

la password è inizialmente hashata con SHA1, che viene poi utilizzato per inizializzare il successivo blocco

La sicurezza di HMAC deriva dall'impossibilità dell'attaccante di trovare una collisione sulla seconda operazione di hashing. Anche se potenzialmente fosse in grado di trovare una collisione sulla prima operazione, è sicuramente non in grado di trovarne una sulla seconda.

Funzionamento di PBKDF2

La funzione HMAC è utilizzata c volte, il valore c dipende dalla potenza computazionale del dispositivo che sta svolgendo la funzione di encryption.

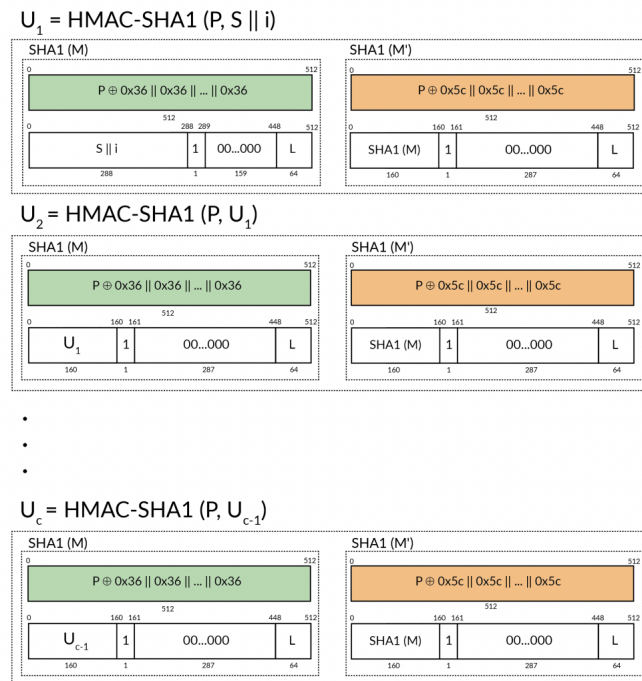


Fig. 2. PBKDF2 schema (HMAC computed in a standard mode)

Questa tecnica viene utilizzata per rallentare gli attaccanti, i conti effettuati possono essere considerati "inutili, a vuoto", che tuttavia garantiscono che un attaccante sia scoraggiato dall'eseguirli.

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 abbiamo 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

La firma digitale è un schema di verifica utilizzata per garantire l'autenticità di messaggi digitali o documenti. 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 $\text{hash}(m)$ con RSA, consideriamo:

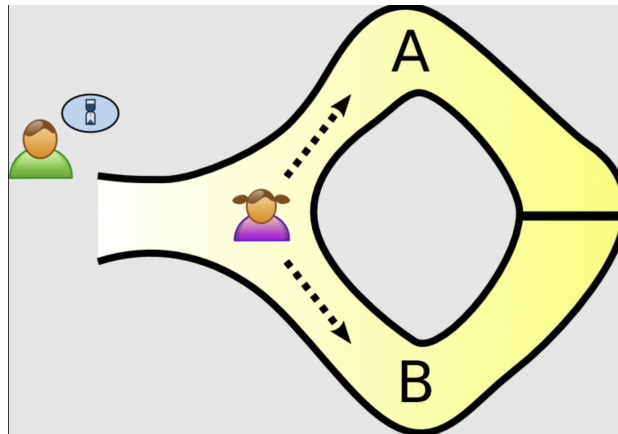
Public: (n, e)
Private: (p, q, d)

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 $c^e = m$.

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

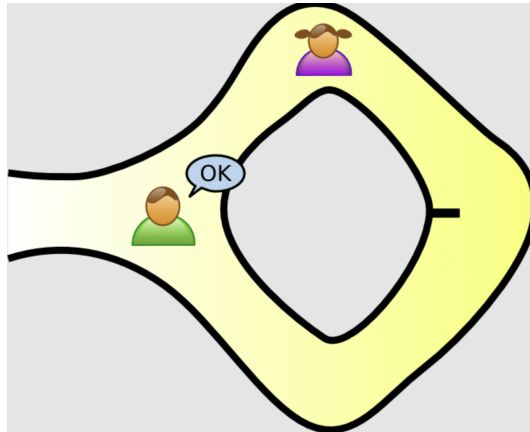
Funzioni "zero-knowledge"

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 220, 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.

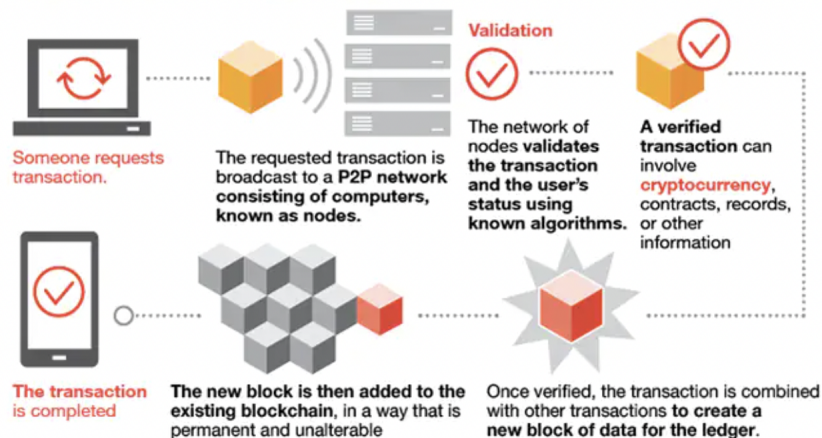
Block-chaining & Distributed Ledger Technology

Il bitcoin ed il blockchain è un argomento strettamente legato alla crittografia e alle funzioni hash. L'obiettivo del bitcoin è quello di creare un sistema di scambio e verifica basato sulla crittografia, ed il bitcoin è il più grande esempio.

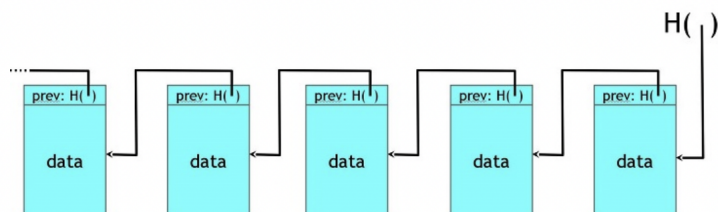
- la firma digitale per aggiungere una transazione
- controllo e aggiunta di transazioni se e solo se la maggior parte del network lo ammette
- sistema di remunerazione per i mantenitori del network, basato sulla scoperta di una collisione parziale all'interno del protocollo SHA2

Il Blockchain

How blockchain works



Il blockchain rappresenta l'insieme delle transazioni svolte all'interno del network.



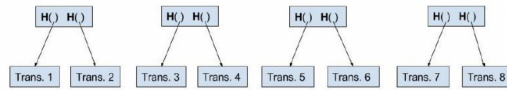
La validazione delle transazioni

Per validare delle transazioni, il network lavora nel seguente modo:

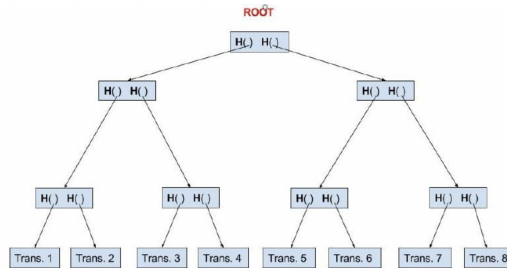
1. un sottoinsieme del network sceglie n transazioni dal paniere delle transazioni
2. viene calcolato il doppio hash delle transazioni con SHA2, utilizzando un NONCE valore casuale
3. quando viene trovato il NONCE valore corretto, il resto del network convalida l'operazione, e chi l'ha scoperto viene remunerato e i valori sono aggiunti alla blockchain

Merkle-Tree

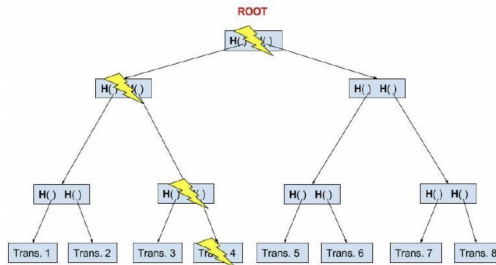
Consideriamo una serie di transazioni, da 1 a 8, calcoliamo i loro hash, e li raggruppiamo a 2 a 2.



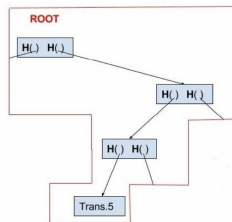
I dati dei due puntatori hash vengono combinati, e utilizzati per calcolare un nuovo puntatore hash, iterando progressivamente arriviamo prima o poi a una radice.



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 Merkel 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.

Per riassumere

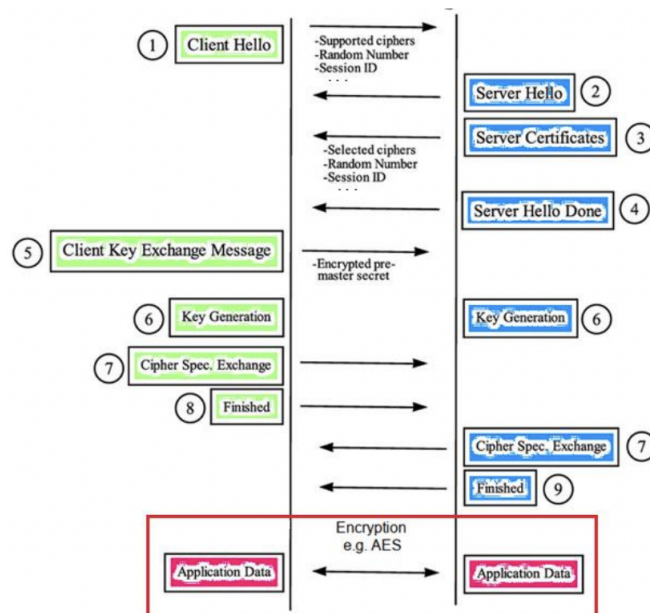
Gli hash pointers sono strutture dati che ci permettono di costruire le strutture dati necessarie per creare una blockchain, inserire transazioni, verificare la presenza di manomissioni, e verificare l'appartenenza.

SSL — TLS

SSL è un protocollo utilizzato per scambiare informazioni su internet in maniera sicura. SSL, o Secure Sockets Layer è stato sviluppato da Netscape nel 1994 per proteggere i dati in trasferimento in rete (cartolina vs busta in rete).

TLS, o Transport Layer Security è stato sviluppato da Internet Engineering Task Force (IETF) è un'evoluzione di SSL, standardizzato ed utilizzabile dal pubblico. TLS1.0 (99), inizialmente praticamente uguale a SSL3 ('96), viene poi aggiornato in TLS1.1 ('06), TLS 1.2 ('08), TLS1.3 ('18). Ogni aggiornamento va a migliorare la sicurezza e l'interoperabilità.

SSL e TLS durante la connessione producono *session states* e *connection states*, informazioni che vengono scambiati attraverso il processo descritto dal protocollo di handshake per concordare i protocolli/parametri da utilizzare durante la connessione.



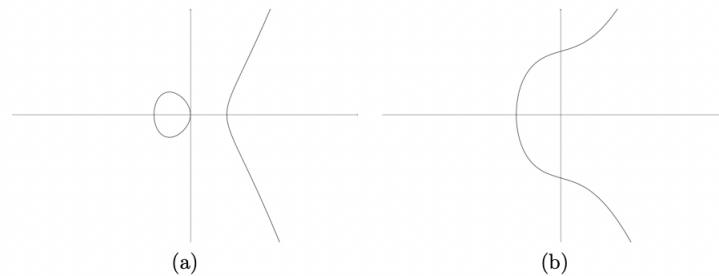
5: Diffie-Hellman

Dopo la terminazione del processo di handshake il client ed il server sono pronti a scambiare informazioni in base ai parametri che si sono appena condivisi.

All'interno del protocollo sono presenti una serie di campi, ad esempi l'alert protocol che va ad informare l'utente di eventuali problemi minori (no, revoked, expired certificates) e grandi (bad record MAC (connection tampering), handshake failure, illegal parameter).

Curve ellittiche

Le curve ellittiche sono adottate perché, a parità di grandezza delle chiavi, garantiscono un ordine di grandezza più sicurezza (4000 bit RSA vs 300 bit EC). Questo ci permette di risparmiare sulla grandezza delle chiavi, velocizzando i calcoli. (Il livello di sicurezza è uguale, è solo che con RSA mi servirebbe una chiave più lunga).



esempio di curve ellittiche

Una curva ellittica come la studiamo noi è rappresentata da una funzione, in particolare:

$$y^2 = x^3 + ax^2 + bx + c$$

$$a, b, c \in \text{Insieme}$$

Ove per *Insieme* si intende un insieme finito, in particolare un campo K finito chiamato Z_5 in modulo p . Una curva ellittica rappresenta l'insieme dei punti geometrici che risolvono l'equazione. In particolare l'insieme delle soluzioni di una curva ellittica è la seguente:

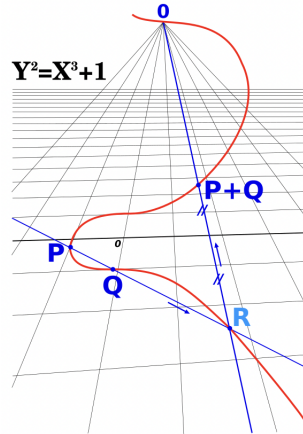
$$S = \{(x, y) \text{ ove } x, y \in K \text{ e } y^2 = x^3 + ax^2 + bx + c\}$$

Quando tuttavia le trattiamo noi, le tratteremo in termini di campo (in questo caso Z_5) ad esempio:

$$y^2 = x^3 + 2x - 1 \text{ mod } 5$$

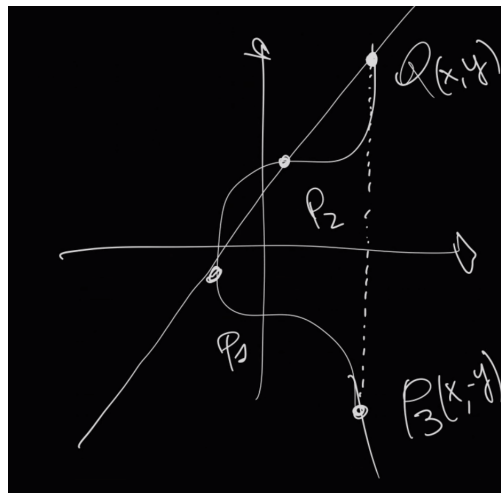
Punto all' ∞ o punto O

Il punto all'infinito è un punto speciale in ogni curva ellittica, è rappresentata dall'intersezione delle rette parallele all'asse y in prospettiva.



Legge dell'addizione e la somma di punti

Consideriamo P_1, P_2, P_3 , e $P_1 + P_2 = P_3$ questa operazione è svolta nel seguente modo:



calcolo prima Q , e P_3 è ricavato negando $l'y$

- costruiamo la retta che passa passante per P_1 e P_2
- il punto in cui interseca nuovamente la curva è chiamato Q
- calcolo la perpendicolare rispetto all'asse x che passa dal punto Q , quando nuovamente incontra la curva abbiamo trovato P_3

Matematicamente l'operazione di calcolare Q è svolta facendo il sistema tra la curva, e la retta che passa per P_1 e P_2

$$\begin{cases} y^3 = x^3 + ax^2 + bx + c \\ y = mx + q \end{cases} \quad (2)$$

Dobbiamo tuttavia considerare i casi particolari, ad esempio il caso in cui P_1 e P_2 siano identici, in questo caso la retta passante sarà semplicemente la tangente.

Un altro caso da considerare è quello in cui la retta passante è parallela all'asse y , in questo caso P_3 sarà il punto all' ∞

Altre proprietà delle curve ellittiche sono:

- $P + \infty = P$
- $P - (x, y) = P + (x, -y)$
- la commutatività: $P + Q = Q + P$
- l'associatività: $(P + Q) + R = P + (Q + R)$

Ma se siamo in grado noi di calcolare i punti, lo può fare anche un attaccante. Dal punto di vista crittografico dobbiamo andare a rafforzare il cifrario, in particolare facciamo due cose:

Utilizziamo l'equazione ed il valore p adatto

Consideriamo una curva standard:

$$y^3 = x^3 + ax^2 + bx + c \text{ mod } p$$

Nel mondo reale il valore p è un valore grande, dell'ordine di grandezza 10^{100} . Questo è fatto perché obbliga un potenziale attaccante a provare tutti i valori da 1 a 10^{100} .

Dobbiamo inoltre accennare che quando le curve ellittiche vengono utilizzate nel mondo reale, solo un piccolissimo sottoinsieme vengono utilizzate, studiate appositamente perché non siano attaccabili.

Incorporiamo il problema del logaritmo discreto

incorporiamo il problema del logaritmo discreto. Ricordiamo:

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

Il valore all'esponente x è utilizzato come numero di volte per sommare il punto P di interesse, in particolare va x a rappresentare il numero di volte che dobbiamo sommare il punto.

$$(P_1 + \dots + P_1) \text{ ove } n = x$$

Funzionamento nel mondo reale

Svolgere l'operazione di calcolo del punto P_3 nel modo canonico è un'operazione dispendiosa, andiamo quindi a considerare dei calcoli più diretti.

Consideriamo $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$

- $P_1 + P_2 = P_3$
- $x_3 = m^2 - x_1 - x_2$
- $y_3 = m(x_1 - x_3) - y_1$
- m è il coefficiente angolare della retta passante per P_1 e P_2

Ci manca solo m , che possiamo calcolare in 2 modi a seconda del caso:

- se $P_1 \neq P_2$ allora $(y_2 - y_1)/(x_2 - x_1)$
- se $P_1 = P_2$ allora $(3x_1^2 + b)/2y_1$

Esempio di codifica

Consideriamo un messaggio, ad esempio "ciao". Il messaggio è codificato in un numero secondo un criterio (uno qualsiasi, ad esempio ASCII). Il messaggio codificato viene inserito in un punto nella curva ellittica.

Dobbiamo tuttavia notare che non tutti i punti x della curva ellittica possiede un corrispettivo y , in questi casi adottiamo un algoritmo probabilistico chiamato algoritmo di Koblitz

Consideriamo:

- $\text{mod } p = 179$
- EC: $y^2 = x^3 + 2x + 7$
- Messaggio $m = 5$ ("C" nella nostra parola "Ciao")

Algoritmo di Koblitz

E' un algoritmo probabilistico che ci permette di gestire numeri che normalmente non sono utilizzati nelle curve elittiche. Fissiamo un valore k che sta a rappresentare una soglia (probabilità di errore). Il valore $k = 1/2^k$ è la probabilità di errore.

Supponiamo di fissare $k = 10$, eseguiamo la seguente operazione:

$$m * k + k < p$$

Nel nostro esempio

$$m * 10 + 10 < 179$$

Questo ci dice che possiamo rappresentare tutti i valori che sono compresi tra 0 e 16:

$$0 \leq m \leq 16 \rightarrow m = 5 \leq 16$$

Che diventa utilizzabile nella nostra curva ellittica, il valore x è quindi rappresentato da

$$x = m * k + j$$

Il valore j è un valore compreso tra $0 \leq \dots \leq j$ over $j = k - 1$

Nel nostro esempio abbiamo $5 * 10 + j$, $t * 10 + 0 = 50$, quindi $x = 50$. Questo valore è quindi utilizzabile dentro all'equazione della curva ellittica, che quindi diventa

$$y^2 = 50^3 + 2 * 50 + 7 \mod 179$$

Possiamo risolvere la funzione, facendo $y^2 = 165$, troviamo tuttavia che non esiste un y valido.

Quando non troviamo un y valido, andiamo a provare con un j diverso, normalmente il valore successivo.

Svolgiamo quindi $5 * 10 + 1 = 51$, che ci da $x = 51$. L'equazione della curva ellittica diventa quindi

$$y^2 = 51^3 + 2 * 51 + 7 \mod 179$$

Ed in questo caso la funzione è risolvibile con $y^2 = 121$ e la coppia è $(51, 11)$. La coppia va a rappresentare il messaggio m , codificato con la curva ellittica.

La componente probabilistica

La componente probabilistica entra in gioco in relazione al campo, e al valore k fissato all'inizio. La probabilità di trovare un valore j adatto da inserire è strettamente legato al valore k , che rappresenta anche il numero di tentativi massimi.

Per ricavare il messaggio iniziale svolgiamo l'operazione inversa $m = x/k$, in questo caso $51/10 = 5(, 1)$. Il valore non è precissimo, possiamo infatti vedere che tutti i valori da 50 a 59 sarebbero validi, questo dipende strettamente dal valore k .

Teorema di Hasse

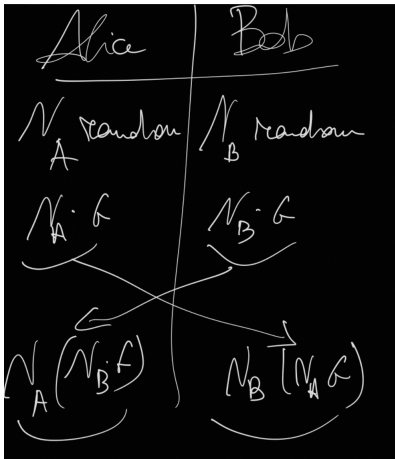
Consideriamo una EC $\mod p$, con N punti, l'algoritmo di Hasse dice che vi è una relazione tra p e N , infatti:

$$|N - p - 1| < 2\sqrt{p}$$

Questo ci permette di dire un pò di cose, considerando una curva ellittica $\text{mod } p$

- N sarà dell'ordine di grandezza di p ovvero $N \sim p$,
- Il numero di soluzioni è dato da $p/2$ numeri positivi, $p/2$ numeri negativi, ed 1 (il punto all' ∞). Il totale delle soluzioni di una qualsiasi curva ellittica è quindi rappresentata da $p/2 + p/2 + 1$

ECDH: Elliptic Curve Diffie Hellman



- i numeri N_a, N_b sono due numeri generati casualmente.
- G rappresenta il numero del problema delle curve ellittiche da sommare per sé stesso, N_a volte
- Con i valori scambiati, Bob e Alice possono utilizzarli come seed per una KGF, in quanto l'informazione riguardo al valore delle chiavi non è passato sul canale in chiaro.

Dobbiamo accennare che nuovamente questo schema di scambio di chiavi è vulnerabile ad una attacco MITM.

Elliptic curve to attack RSA

Consideriamo $\text{EC mod } p$, se facessimo:

$$\text{EC mod } n \text{ ove } n = p * q \text{ è la chiave pubblica di RSA}$$

Cosa cambierebbe? Cambierebbe che quando lavoriamo con le curve ellittiche abbiamo imposto di lavorare su un campo, ma se lavoriamo su un n di RSA lavoriamo su un anello.

Come abbiamo visto in precedenza su un campo *tutti* i valori hanno un inverso moltiplicativo, ma lo stesso non può essere detto su un anello. Questo implica che non sempre è possibile effettuare la divisione per ricavare m (m coefficiente angolare della retta).

Questo problema, tuttavia ci torna utile, come abbiamo visto, l'inverso moltiplicativo è unico quando lavoriamo con p se e solo se

$$MCD(denominatore, p) = 1$$

Quando tuttavia andiamo a lavorare con n e quindi:

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

Nel 99% dei casi il denominatore ed n sono coprimi, c'è tuttavia una bassa ma non trascurabile percentuale di casi in cui:

$$MCD(denominatore, n) \neq 1$$

Come sappiamo da RSA, tuttavia, $n = p * q$, e se $MCD(denominatore, n) \neq 1$, il valore sarà uguale a d , che corrisponde a o p o q . Il calcolo dell'altro parametro diventa quindi triviale.

Questo attacco, come abbiamo detto, è possibile solo nel caso in cui riusciamo a trovare un denominatore problematico, per cui non riusciamo a calcolare l'inverso moltiplicativo, perché ciò implica che il risultato corrisponderà ad uno dei fattori di n .

1. Abbiamo trovato un valore per cui vale $MCD(denominatore, n) \neq 1$
2. Che vuol dire che abbiamo trovato un valore per cui non esiste l'inverso moltiplicativo
3. Che vuol dire che abbiamo trovato uno dei termini di n

E nel mondo reale?

Nel mondo reale, prima di selezionare il punto P di interesse andiamo a fare le verifiche, questo viene fatto sommando più volte il punto, fino a quando troviamo un valore per cui non riusciamo a calcolare l'inverso moltiplicativo, e quindi attaccabile:

$$P + P + \dots$$

Ma quante volte dobbiamo sommare P prima di trovare un denominatore problematico? Creiamo una variabile m che rappresenta il più piccolo numero di volte che dobbiamo sommare P per sé stesso prima di trovare un problema.

Cosa possiamo intuire di m ?

- m sicuramente divide N , ove N è il numero di punti della curva ellittica in particolare $N/m = k$, N è divisibile per m .
Ricordiamo che N è un numero finito, in quanto in crittografia noi lavoriamo su un campo/anello.

Numero liscio e B-liscio

Un numero liscio è un numero che è che fattorizzabile in un prodotto di numeri primi piccoli. La terminologia "B-liscio" è utilizzato per definire una soglia dei fattori, diciamo quindi che un numero è B-liscio se i numeri primi sono minori del valore B .

- quando andiamo a selezionare un valore N di ordine di grandezza 10^{300} , la probabilità che il numero sia liscio o b-liscio non è trascurabile.

Ed un numero b-liscio è facilmente fattorizzabile.

Cosa possiamo fare allora?

1. selezioniamo una curva ellittica a caso, ad esempio:

$$y^2 = x^3 + bx + c$$

2. selezioniamo un punto P a caso

$$P(0, 0)$$

Per imporre che la curva ellittica passi per il punto P , possiamo fare la sostituzione, in particolare:

$$0^2 = 0^3 + 0 + c$$

e impostiamo un sistema in cui:

$$\begin{cases} a = 0 \\ b = 7 \text{ un numero casuale} \\ c = 0 \text{ come da risoluzione della sostituzione} \end{cases} \quad (3)$$

Inseriamo i dati che abbiamo appena calcolato nella funzione della curva ellittica, modulando a n di RSA:

$$y^2 = 7x \text{ mod } n$$

Non siamo tuttavia in grado di determinare la natura di N dato che l'abbiamo determinato casualmente, diamo per scontato che sia liscio.

Noi sappiamo che $m * P \rightarrow \infty$, è intuibile che anche i multipli di $m * P$ tendano all'infinito, ad esempio $2m * P \rightarrow \infty$, $3m * P \rightarrow \infty$, $10!m * P \rightarrow \infty$. Siccome non riusciamo a calcolare direttamente m , cerchiamo un valore con cui moltiplicare m per trovare la problematica.

$$100.000! * P$$

Se $100.000! * P$ è un multiplo di m , allora anche $100.000! * P$ farà andare la curva all'infinito, avremo quindi trovato un valore accettabile.

Questo avviene perché N è liscio, e se m divide N , anche m è liscio, e $100.000!$ contiene tutti i fattoriali piccoli.

E se così non fosse?

Andiamo a provare con altri parametri nella nostra curva ellittica, modificando a piacimento il punto P , e il parametro b .