

Teoria dell'informazione e crittografia

Riassunto videolezioni

Contents

Lezione 2	4
Codifica di canale	4
Ridondanza	6
Canali - Modelli di rumore	6
Burst di errori	10
Lezione 3	11
Codici pesati	11
Lezione 4	16
Codici triangolari	16
Codici cubici	17
Codici ipercubici	18
Codici di Hamming	18
Codici di Hamming ottimali	24
Lezione 5	26
Interpretazione geometrica	26
Cubi	26
Sfere	29
Codifica di sorgente	30
Lezione 6	32
Albero di decodifica	33
Disuguaglianza di Kraft	36
Lezione 7	42
Disuguaglianza di McMillan	43
Codici a blocchi accorciati	45
Codici di Huffman	46

Lezione 8	50
Huffman con $r > 2$	52
Robustezza codici di Huffman	56
Lezione 9	59
Teoria dell'informazione - Entropia	59
Lezione 10	64
Sorgente Bernoulliana	64
Disuguaglianza di Gibbs	65
Codifica di Shannon-Fano	68
Estensione di una sorgente	71
Entropia in sorgenti estese	72
Lezione 11	73
Primo teorema di Shannon-Fano	73
Canali di comunicazione	74
Probabilità all'indietro	76
Lezione 12	78
Entropia congiunta	79
Mutua informazione	82
Capacità del canale	84
Lezione 13	85
Secondo teorema di Shannon	87
Dimostrazione - Secondo teorema di Shannon	87
Preliminari matematici	87
Dimostrazione	89
Lezione 14	91
Inverso Teorema di Shannon	94
Lezione 14 - Parte 2	96
Crittografia	96
Principio di Kerkhoffs	96
Lezione 15	98
Cifrari storici	98
Lezione 16	104
Vigenère	105
DES	106
Lezione 17	108
AES	108
Crittosistemi simmetrici	109

Lezione 18	112
Substitution-Permutation Network (SPN)	112
Design delle S-box	114
Cifrario a flusso	115
RC4	115
Lezione 19	118
ANSI X9.17	121
Crittosistemi a chiave pubblica	121
Lezione 20	123
Algoritmo di Diffie-Hellman	124
Crittosistema El Gamal	125
RSA	126
Lezione 21	130
Firme digitali e funzioni di hash	130
Schema El Gamal	131
Funzioni di Hash	133

Lezione 2

Supponiamo di avere una sorgente S che emette un simbolo ogni colpo di clock. Questo simbolo subisce una codifica di sorgente (per compattare la rappresentazione dell'informazione) e successivamente, prima di immettere nel canale, bisogna effettuare una codifica di canale. Il canale è rumoroso per definizione, quindi bisogna aggiungere al simbolo delle cifre di controllo. Il messaggio quindi sarà composto da m bit di messaggio e k bit di controllo, il totale farà n .

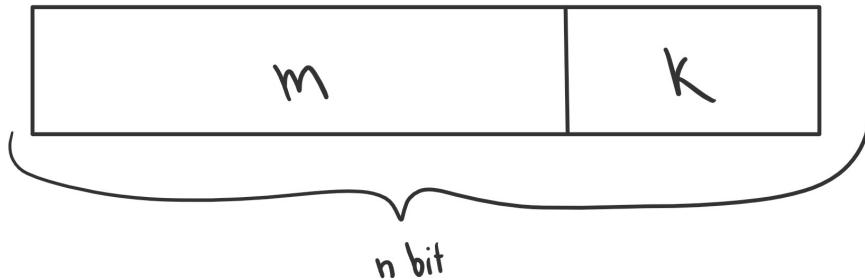


Figure 1: Struttura messaggio da inserire nel canale

Le cifre di controllo servono ad aumentare la ridondanza del messaggio, in questo modo per il ricevente sarà più facile capire se ci sono stati errori.

La **ridondanza** serve per capire meglio il messaggio (esempio, se togli una lettera dall'alfabeto italiano le parole si capiscono comunque, però con quella lettera è più veloce capire).

Controllo di parità singolo - Codifica di canale

Si aggiunge una cifra alla fine del messaggio. Ci sono due tipi di parità:

- Parità pari: il bit equivale a 1 se il numero di 1 nel messaggio è dispari (in questo modo si arriva ad avere un numero pari di 1). il bit equivale a 0 se il numero di 1 nel messaggio è pari (il numero è già pari, quindi non aggiungo altri 1)
- Parità dispari: stesso ragionamento ma al contrario.

Di base useremo la parità pari, hanno stesse proprietà (in certi casi si usa quella dispari, es. mando pacchetto di zeri su linea con 0 volt).

Per il calcolo del bit di parità si usa una funzione di parità:

$$y = \text{Parity}(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n \quad (1)$$

In questo modo quando il ricevente legge il messaggio, controlla che il numero di 1 sia pari. Se non lo è, allora c'è stato un errore singolo.

La stessa funzione può essere vista come:

$$y = \text{Parity}(x_1, x_2, \dots, x_n) = x_1 + x_2 + \dots + x_n \bmod 2 \quad (2)$$

Questo è utile quando si vuole trattare 0 e 1 con i loro significati numerici.
Isomorfismo fra queste due forme:

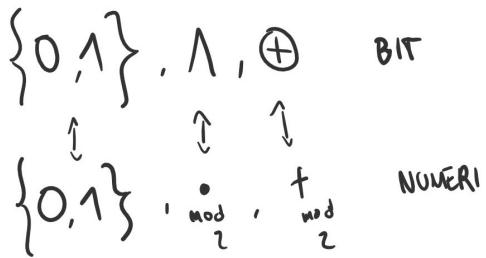


Figure 2: Isomorfismo bit/numeri

Per fare il calcolo della funzione $\text{Parity}(x)$ di solito si basa l'hardware (visto che è un operazione molto diffusa, ogni hardware ha supporto per questo tipo di calcoli). In caso l'hardware supporti solo ad esempio lo xor fra due soli bit, posso scomporre in questo modo ed effettuarne uno ad uno:

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = (x_1 \oplus x_2) \oplus (x_3 \oplus x_4) \oplus (x_5 \oplus 0) \quad (3)$$

n.b.

$$x \oplus 0 = x$$

$$x \oplus 1 = \neg x$$

Si potrebbe anche utilizzare un automa a stati finiti per modellare il problema:

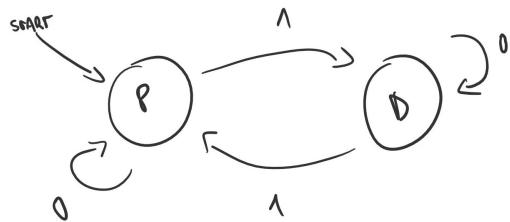


Figure 3: Automa a stati finiti per il calcolo pari/dispari

Ridondanza

Con ridondanza intendiamo, data la Figura 1, questo rapporto:

$$R = \frac{\text{numero di cifre totali}}{\text{numero di cifre di controllo}} = \frac{n}{m} \geq 1 \quad (4)$$

Con $k = 0$ la ridondanza vale 1, e praticamente stiamo inserendo il messaggio senza bit di parità.

In altri termini si dice che:

$$R = 1 + \text{eccesso di ridondanza} \quad (5)$$

$$R = 1 + \frac{k}{m}$$

Quando si definisce un metodo per il controllo degli errori, si cerca di avere un eccesso di ridondanza minore possibile (meno bit si usano nel canale, meno errori avverranno; ma in generale meglio ottimizzare).

Quanto vale la ridondanza nel caso dei controlli di parità?

$$R_{\text{parity}} = \frac{n}{n-1} = \frac{(n-1)+1}{n-1} = 1 + \frac{1}{n-1} \quad (6)$$

Quindi l'eccesso di ridondanza è $\frac{1}{n-1}$.

Ovviamente, osservando come l'eccesso di ridondanza sia uguale a $\frac{k}{m}$, a parità di m , il numero di bit del messaggio, il modo di ridurre al minimo la ridondanza è quello di minimizzare il numeratore, quindi la k . In gergo si dice che ogni cifra di controllo dovrà *coprire* il maggior numero di bit di messaggio possibile, per avere una bassa ridondanza.

Canale - modelli di rumore

Il canale più semplice e facile da gestire è il **rumore bianco**, definito da queste proprietà:

- La probabilità che avvenga un errore in ciascuna posizione è uguale a p per ogni bit. Di conseguenza la probabilità che la trasmissione avvenga in modo corretto è $1 - p$.
- Il fatto che avvenga un errore in posizione i non influisce sulle altre probabilità, quindi si parla di eventi statisticamente indipendenti (condizione che rende poco realistico il rumore bianco, es. sbalzo di corrente).

Nell'intero pacchetto, qual è la probabilità di avere k errori, dato $0 \leq k \leq n$? Ragioniamo prima sul valore di p

- $p = 0$? No, impossibile che in un canale ci sia completa assenza di errore (qualunque supporto lo utilizzi, col tempo si deteriora).
- $p = 1$? No, se no basta una porta not al ricevitore per avere tutti i bit corretti (come prima).

- $p > \frac{1}{2}$? Potrebbe aver senso, però se ho un canale che sbaglia più della metà delle volte, basta una porta not per ridurre questa quantità fino a una quantità minore di $\frac{1}{2}$
- $p = \frac{1}{2}$? Così ottengo un generatore di numeri casuali perfetto in quanto ogni bit viene trasformato in modo casuale in 0 o in 1 (cosa impossibile).

Possiamo concludere dicendo che p è sempre $0 < p < \frac{1}{2}$.

Tornando alla domanda di prima, quanto è la probabilità di avere k errori? Partiamo dalla probabilità di avere 1 errore, esso può avvenire in una posizione qualsiasi. Essendo eventi stocastici e statisticamente indipendenti, la probabilità di un errore è uguale a

$$p(1 \text{ errore}) = p(E_1 \vee E_2 \vee \dots \vee E_n) \quad (7)$$

Con E_k s'intende l'evento che avvenga un errore in posizione k .

Quanto è la probabilità che avvenga un errore *solo* nella posizione 2?

$$\begin{aligned} p(\text{errore in pos 2}) &= (1-p)p(1-p)(1-p)\dots(1-p) \\ &= p(1-p)(1-p)\dots(1-p) \\ &= p(1-p)^{n-1} \end{aligned} \quad (8)$$

Ovviamente al posto della seconda posizione, questa regola vale per ogni posizione (proprietà commutativa della moltiplicazione).

Quindi la probabilità di avere un solo errore:

$$\begin{aligned} p(1 \text{ errore}) &= p(\text{errore in pos 1}) + p(\text{errore in pos 2}) + \dots + p(\text{errore in pos n}) \\ &= np(1-p)^{n-1} \end{aligned} \quad (9)$$

E la probabilità di ottenere 2 errori?

Vediamo prima la probabilità di ottenere 2 errori in 2 posizioni fissate:

$$\begin{aligned} p(\text{errori in pos 1 e 2}) &= p \cdot p(1-p)(1-p)\dots(1-p) \\ &= p^2(1-p)(1-p)\dots(1-p) \\ &= p^2(1-p)^{n-2} \end{aligned} \quad (10)$$

Come posso creare le combinazioni di queste posizioni?

$$\binom{n}{2} = \frac{n(n-1)}{2} \quad (11)$$

Quindi questo è il numero di modi in cui posso disporre gli errori nel pacchetto

Da qui, la probabilità di ottenere 2 errori è:

$$p(2 \text{ errori}) = \binom{n}{2} p^2(1-p)^{n-2} \quad (12)$$

Riprendendo la probabilità di avere un errore, essa si può riscrivere come:

$$p(1 \text{ errore}) = \binom{n}{1} p^1 (1-p)^{n-1} \quad (13)$$

Ora, generalizzando, si può rispondere alla domanda: qual è la probabilità che si verifichino k errori in un messaggio di n bit (errori disposti in qualsiasi posizione all'interno del pacchetto)?

$$p(k \text{ errori}) = \binom{n}{k} p^k (1-p)^{n-k} \quad \text{per } 0 \leq k \leq n \quad (14)$$

Vediamo ora come si comporta la funzione $p(\text{errori})$ dati un n e un p fissati.

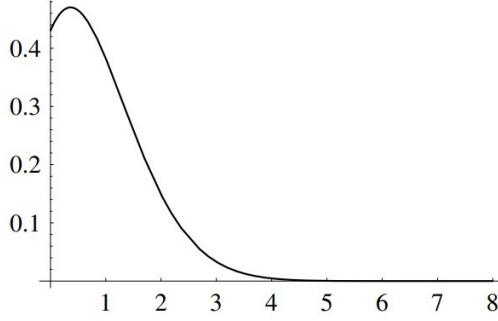


Figure 4: Grafico di $\binom{n}{k} p^k (1-p)^{n-k}$ con $n = 8$ e $p = 0.1$

Ma se all'interno del pacchetto avviene un numero pari di errori? Il controllo di parità fallisce in quanto il numero di 1 è pari (numero pari di 1 iniziale + numero pari di errori = numero pari di 1). Viceversa se il numero di errori è dispari, allora il controllo di parità se ne accorge.

Quanto è la probabilità che il controllo di parità fallisca? In altri termini: quanto è la probabilità che avvenga un numero pari di errori?

Procediamo per passi; recuperiamo la probabilità di commettere un errore dall'equazione (13):

$$\begin{aligned} p(1 \text{ errore}) &= \binom{n}{1} p^1 (1-p)^{n-1} \\ &= np(1-p)^{n-1} \end{aligned} \quad (15)$$

Ricordiamo un'approssimazione sfruttando le serie di Taylor:

$$(1+x)^\alpha \approx 1 + \alpha x \quad \text{per } |x| < 1 \quad (16)$$

Possiamo considerare n come molto minore di $\frac{1}{p}$, quindi scrivere:

$$\begin{aligned}
& np(1-p)^{n-1} \\
& \approx np[1-p(n-1)] \\
& = np[1-p(n-1)] \\
& = np[1-np+p] \\
& = np[1-np+p] \\
& np - n^2p^2 + np^2
\end{aligned} \tag{17}$$

Ora trascuriamo i termini con p^2 in quanto essendo minore di 1, dominerà p ; quindi

$$\begin{aligned}
p(1 \text{ errore}) & \approx np \\
& = \binom{n}{1} p
\end{aligned} \tag{18}$$

Per due errori:

$$\begin{aligned}
p(2 \text{ errori}) & \approx np \binom{n}{2} p^2 \\
& = \binom{n}{2} p^2
\end{aligned} \tag{19}$$

Generalizzando per k errori:

$$p(k \text{ errori}) \approx np \binom{n}{k} p^k \tag{20}$$

Quindi, tornando al problema di prima, dobbiamo fare un'osservazione:

$$\sum_{k=0}^n \binom{n}{k} p^k (1-p)^{n-k} = 1 \tag{21}$$

Se sommo tutte le probabilità, fare zero errori, fare un errore, fare due errori ecc. ovviamente ho come somma 1 (evento stocastico).

Sfrutto questa proprietà:

$$(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k} \tag{22}$$

... Lezione 02 parte 3 min 33 +- ...

Quindi la probabilità che il controllo di parità fallisca è

$$p(\text{controllo parità fallisce}) = \frac{1 + (1-2p)^n}{2} - (1-p)^n \tag{23}$$

Burst di errori

Gli errori nella realtà avvengono in maniera non indipendente, di solito avvengono errori su sequenze di bit adiacenti. Definiamo L come lunghezza del **burst** di errori.

La parola **ciao** in ASCII è rappresentata in questo modo:

$$\begin{aligned} \text{C} &= 1000011 \\ \text{I} &= 1001001 \\ \text{A} &= 1000001 \\ \text{O} &= 1001111 \end{aligned}$$

... fine lezione 02 parte 3

Lezione 3

Codici pesati

Dato un messaggio da spedire nel canale, si associa ad esso un insieme di valori. Ad esempio, dato un messaggio $m = m_1 \ m_2 \ \dots \ m_n$, codifichiamo ogni m_i nel seguente modo:

$$\begin{array}{l} 26 \text{ lettere A-Z} \\ 1 \text{ spazio } _ \\ 10 \text{ cifre 0-9} \\ \downarrow \\ 37 \text{ simboli} \end{array}$$

Successivamente si associa ad ogni messaggio un insieme di valori di *peso*:

$$\begin{array}{l} \text{msg: } m_1, m_2, \dots, m_n, c \\ \text{pes: } n+1, n, \dots, 2, 1 \end{array}$$

Facciamo un esempio, mettiamo di voler spedire la parola **ciao**, utilizzando la codifica mostrata in precedenza.

	C	I	A	0	check
msg	2	8	0	14	check
pes	5	4	3	2	1

Il mittente pesa ogni carattere moltiplicando la codifica per il peso:

$$2 \cdot 5 + 8 \cdot 4 + 0 \cdot 3 + 14 \cdot 2 + \text{check} \cdot 1$$

Successivamente si pone questa espressione uguale a $0 \bmod 37$ (questo è il numero totale di caratteri).

$$2 \cdot 5 + 8 \cdot 4 + 0 \cdot 3 + 14 \cdot 2 + \text{check} \cdot 1 = 0 \bmod 37$$

$$70 + \text{check} = 0 \bmod 36$$

$$\text{check} = 4$$

Successivamente spedisce il messaggio con il vettore dei pesi.

Il ricevitore si calcola il peso (usando il valore di *check* spedito dal mittente) e controlla che il peso sia uguale a $0 \bmod 37$.

Da notare che 37 è primo, quindi ci va bene perchè lavoriamo in un campo. Ad esempio i conti in modulo 6 hanno i divisori di zero ($3 \cdot 2 = 0 \bmod 6$). Se non è primo piuttosto aumentiamo il numero di caratteri (piuttosto non li usiamo, ma ci semplifica tutto avere un numero primo).

I codici pesati sono utili per individuare errori come ad esempio lo scambio di caratteri, l'inserimento di caratteri da parte del canale ecc...

Ad esempio, se all'interno di un messaggio al posto di **ab** viene inviato **ba**, allora la differenza fra le somme pesate è (fissato $k+1$ come peso di **a**, k come peso di **b**).

$$\begin{aligned}
 a(k+1) + bk &\rightarrow b(k+1) + ak \\
 a(k+1) + bk - b(k+1) + ak &= \\
 = b(k+1) + ak - a(k+1) - bk & \\
 = bk + b + ak - ak - a - bk & \\
 = b - a
 \end{aligned}$$

Quindi il messaggio viene accettato solo se la differenza è uguale a 0, quindi quando $b - a = 0 \text{ mod } 37$, ma per essere 0 allora i due caratteri devono essere uguali. Se due simboli uguali vengono scambiati quindi, il messaggio viene accettato lo stesso.

Vediamo un algoritmo per calcolare i pesi:

```

1: procedure CALCOLACHECKDIGIT
2:   sum  $\leftarrow 0$ 
3:   ssum  $\leftarrow 0$ 
4:   while not EOF do
5:     read symbol
6:     sum  $\leftarrow$  sum + symbol mod 37
7:     ssum  $\leftarrow$  ssum + sum mod 37
8:   temp  $\leftarrow$  ssum + sum mod 37
9:   c  $\leftarrow 37 - \text{temp}$  mod 37
10:  return c

```

Una simulazione dell'algoritmo con la parola **ciao** è la seguente:

msg	sum	ssum
C = 2	2	2
I = 8	10	12
A = 0	10	22
O = 14	24	46 \equiv 9

$$\text{temp} \leftarrow 9 + 24 \equiv 33$$

$$c \leftarrow 37 - 33 = 4 \equiv \text{E}$$

Quindi il messaggio spedito sarà **ciaoE**.

Il ricevente ricalcolerà la tabella usando il messaggio appena arrivato:

msg	sum	ssum
C = 2	2	2
I = 8	10	12
A = 0	10	22
O = 14	24	46 ≡ 9
E = 4	28	37 ≡ 0

La somma delle somme finale è 0, quindi il messaggio è ok. Se non fosse 0, allora il messaggio è stato modificato, ma non posso sapere dove.

Astraiamo il procedimento appena utilizzato:

msg	sum	ssum
w	w	2
x	w + x	2w+x
y	w + x + y	3w + 2x + y
z	w + x + y + z	4w + 3x + 2y + z
c	w + x + y + z + c	5w + 4x + 3y + 2z + c ≡ 0 mod 37

Si noti come la somma delle somme assegna un peso decrescente a tutti i caratteri: 5 a w , 4 a x e così via...

Il codice ISBN è un esempio di codice pesato su base 11; proviamo a verificare il seguente codice: 1-58488-508-4

msg	sum	ssum
1	1	1
5	6	7
8	14 ≡ 3	10
4	7	17 ≡ 6
8	15 ≡ 4	10
8	12 ≡ 1	11 ≡ 0
5	6	6
0	6	12 ≡ 1
8	14 ≡ 3	4
4	7	11 mod 11 ≡ 0

Quindi il codice è corretto. Nel caso l'ultima cifra fosse un 10 si inserisce una x.

Es. 2.4 - 1, pag.24

Caso: rumore bianco

$p = 0.001$ (probabilità errore in ogni posizione)

$n = 100$ (dimensione pacchetto)

Quanto è la probabilità che ci siano 0 errori?

$$\begin{aligned} p(0 \text{ errori}) &= (1 - p)^n \\ &= \left(1 - \frac{1}{1000}\right)^{100} \\ &= e^{100 \cdot \ln\left(1 - \frac{1}{1000}\right)} \\ &\approx e^{100 \cdot \ln\left(-\frac{1}{1000}\right)} \\ &= e^{-\frac{1}{10}} \end{aligned}$$

Es. 2.4 - 2, pag.25

Caso: rumore bianco

$p = 0.001$ (probabilità errore in ogni posizione)

$n = 100$ (dimensione pacchetto)

Quanto è la probabilità di non riuscire ad individuare un errore?

Per rispondere a questa domanda, possiamo rispondere a: quanto è la probabilità che il controllo di errore singolo sbagli?

Ricordando l'equazione (23):

$$\begin{aligned} p(\text{controllo parità fallisce}) &= \frac{1 + (1 - 2p)^n}{2} - (1 - p)^n \\ &= \frac{1 + (1 - \frac{2}{1000})^{100}}{2} - \left(1 - \frac{1}{1000}\right)^{100} \\ &\approx 0.0045 \end{aligned}$$

Codici a correzione d'errore

I codici a correzione d'errore, a differenza del controllo di parità, permette al ricevente di rilevare dove è avvenuto un errore ed eventualmente di correggerlo. Questo ovviamente richiede di aggiungere delle cifre di controllo ulteriori.

Codici rettangolari - 1 Errore

Il messaggio inserito nel canale viene rappresentato come un rettangolo (logicamente parlando, non fisicamente). Rappresentiamo i bit di messaggio in questo modo:

$$\begin{array}{ccccccc} o & o & o & \dots & o & x \\ o & o & o & \dots & o & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ o & o & o & \dots & o & x \\ x & x & x & \dots & x & \end{array}$$

Nell'esempio il carattere o rappresenta un bit di messaggio, x rappresenta un carattere di controllo. Ogni cifra di controllo sulle righe codifica la riga corrispondente, quelle sulle colonne codificano la colonna.

In questo modo, se un bit di messaggio viene modificato, posso individuare subito l'errore in quanto verranno sbagliati i bit di controllo sulla relativa riga e colonna. Ad esempio, se un bit in posizione i, j viene alterato, allora i due controlli di parità sulla riga i e sulla colonna j falliranno, ma il ricevente intuisce subito quale bit ha subito modifiche.

Per capire quanto il metodo è efficiente, andiamo a calcolarne la ridondanza:

$$\begin{aligned} R &= \frac{m \cdot n}{(m-1)(n-1)} & (24) \\ &= 1 + \frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)} \end{aligned}$$

Quindi l'eccesso di ridondanza equivale a $\frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)}$. Quanto è l'eccesso di ridondanza più piccola possibile in questa classe di codici (codici rettangolari)? L'idea è quella di minimizzare la funzione

$$\frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)}$$

Si dovrebbero quindi calcolare le derivate parziali rispetto a n e a m e vedere quando si annullano, ma è facilmente intuibile che questa quantità è la minore possibile quando $m = n$.

Per cui i più efficienti sono i codici **quadrati**, ovvero il sottoinsieme di codici rettangolari in cui il numero di righe è uguale al numero di colonne.

La loro ridondanza vale

$$R = \frac{n^2}{(n-1)^2} = 1 + \frac{2}{n-1} + \frac{1}{(n-1)^2}$$

I peggiori invece, sono i codici con una singola riga (i bit di controllo sono di più dei bit di messaggio!)

Lezione 4

Seguendo l'esempio dei codici rettangolari, proviamo a disporre ora i bit di messaggio in altre forme logiche, in modo da coprire il più gran numero di bit di messaggio con ogni bit di controllo.

Codici triangolari - 1 Errore

Proviamo a creare una forma logica di un codice triangolare:

$$\begin{array}{ccccc} o & o & o & o & x \\ o & o & o & x & \\ o & o & x & & \\ o & x & & & \\ x & & & & \end{array}$$

Ovviamente il numero di bit di messaggio deve consentire una costruzione di questo genere (dagli antichi greci: numeri triangolari).

Come prima, ogni bit di controllo copre la relativa colonna e riga.

Quanto vale la ridondanza di questo tipo di codici?

Il numero di bit totali è

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} \quad (25)$$

mentre il numero di bit di messaggio è

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{(n-1)n}{2} \quad (26)$$

Per cui passiamo a calcolare la ridondanza:

$$\begin{aligned} R_{triangolare} &= \frac{\frac{n(n+1)}{2}}{\frac{(n-1)n}{2}} \\ &= \frac{\cancel{n}(n+1)}{\cancel{n}} \cdot \frac{2}{(n-1)\cancel{n}} \\ &= \frac{(n+1)}{(n-1)} \\ &= \frac{(n-1)+2}{(n-1)} \\ &= 1 + \frac{2}{(n-1)} \end{aligned} \quad (27)$$

Da qui, l'eccesso di ridondanza vale

$$\frac{2}{(n-1)} \quad (28)$$

Confrontiamolo con l'eccesso di ridondanza dei codici quadrati:

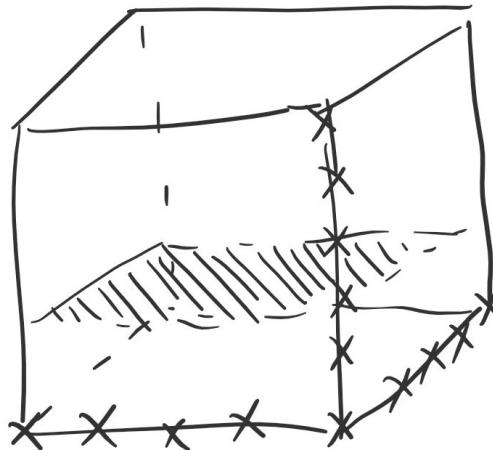
$$ecc_{quadrato} = \frac{2}{n-1} + \frac{1}{(n-1)^2}$$

$$ecc_{triangolo} = \frac{2}{n-1}$$

Si nota ovviamente come i codici triangolari siano sicuramente più efficienti.

Codici cubici - 1 Errore

Un'altra disposizione che si può seguire per ordinare i bit di messaggio è a forma cubica.



Facendo dei conti a *spanne*, ci sono n^3 posizioni, le cifre di controllo sono $3(n-1) + 1 = 3n - 2$.

La ridondanza più o meno quindi vale

$$R = \frac{(n-1)^3 + 3n - 2}{(n-1)^3} \approx 1 + \frac{3}{n^2} \quad (29)$$

Seguendo questo ragionamento, potrei pensare di aumentare le dimensioni

Codici ipercubici

Seguendo il modello dei *conti a spanne*, diciamo più o meno che le cifre di messaggi è $(n - 1)^4$, mentre le check digit saranno $4(n - 1) + 1 = 4n - 3$.

La ridondanza quindi varrà:

$$R = 1 + \frac{4n - 3}{(n - 1)^4} \approx 1 + \frac{4}{n^3} \quad (30)$$

Da qui si nota come si può passare a ragionare in k dimensioni, e si possono tirare delle conclusioni:

$$\begin{aligned} & (n - 1)^k \text{ bit di messaggio} \\ & k(n - 1) + 1 = kn - k + 1 \text{ bit di controllo} \\ & R = 1 + \frac{kn - k + 1}{(n - 1)^k} \approx 1 + \frac{k}{n^{k-1}} \text{ ridondanza} \end{aligned} \quad (31)$$

Da qui si nota come la ridondanza decresce in maniera polinomiale, possiamo fare di meglio?

Codici di Hamming

Codici ottimali per individuare e correggere 1 errore (al massimo) in un canale caratterizzato da rumore bianco. Cosa significa codici ottimali? Il rapporto fra cifre di controllo e cifre di messaggio è esponenziale, meglio di così non si può fare.

Da ora in poi ci riferiremo con k alle cifre di messaggio, m alle cifre di controllo. n sarà la loro somma.

Le combinazioni (configurazioni) di cifre di controllo sono 2^m . Da notare come deve sempre valere $2^m \geq n + 1$, questo è necessario in quanto ci dev'essere una configurazione per ogni posizione di errore, più uno (configurazione per rappresentare zero errori). Altrimenti non abbiamo abbastanza configurazioni per rappresentare tutte le situazioni (errore in posizione 3, errore in posizione 8, ecc).

Supponiamo di voler mandare un messaggio di 7 bit, ci chiediamo quanti di questi bit possono essere di messaggio e quanti di controllo?

$$2^m \geq 8 \rightarrow m \geq 3 \rightarrow m = 3$$

$$k = 7 - 3 = 4$$

Oppure posso chiedermi: devo inviare 4 cifre di messaggio sul canale, di quante cifre di controllo ho bisogno?

$$2^m \geq n + 1 \rightarrow 2^m \geq m + k + 1 \rightarrow 2^m \geq m + 5$$

Quando è vero $2^m \geq m + 5$? Proviamo ad assegnare dei valori a m :

m	2^m	$m + 5$
1	2	6
2	4	7
3	8	8
4	16	9

Il valore più piccolo che ci va bene è proprio 3, quindi l'ideale è utilizzare 3 cifre di controllo.

Ciascuna cifra di controllo fa il controllo di parità su diverse cifre di messaggio. Facciamo un esempio:

Ho 9 cifre di messaggio, avrò quindi 3 cifre di controllo, come si suddividono le cifre per il calcolo delle parità?

$$c_1 = x_1 \oplus x_2 \oplus x_5 \oplus x_7 \quad (32)$$

$$c_2 = x_5 \oplus x_7 \oplus x_8 \oplus x_9$$

$$c_3 = x_1 \oplus x_2 \oplus x_8 \oplus x_9$$

Ma quanto vale la somma di c_1 e c_2 (ricordando che $x \oplus 0 = x$)?

$$c_1 \oplus c_2 = x_1 \oplus x_2 \oplus x_8 \oplus x_9$$

che è esattamente uguale a c_3 !

Allo stesso modo considerando vettori binari di n elementi, una somma in modulo due e un prodotto modulo due, posso effettuare il calcolo in questo modo:

$$v_1 = (1, 1, 0, 0, 1, 0, 1, 0, 0) \quad (33)$$

$$v_2 = (0, 0, 0, 0, 1, 0, 1, 1, 1)$$

↓

$$v_1 + v_2 = (1, 1, 0, 0, 0, 0, 0, 1, 1)$$

Questo insieme, composto da $(\{0, 1\}^n, +_2, \cdot)$ è uno spazio vettoriale, quindi le tre equazioni di parità considerate sono linearmente dipendenti, e questo ovviamente è da evitare visto che la terza equazione non da informazione in quanto è ricavabile dalle altre due.

Ripendendo l'esempio del pacchetto di 7 bit, scriviamo questa tabella che rappresenta le posizioni:

pos	pos in binario
1	001
2	010
3	011
4	100
5	101
6	110
7	111
0 (nessun errore)	000

Da notare come i bit necessari per coprire tutte le posizioni sono 3, esattamente come il numero di bit di controllo. L'insieme delle posizioni in valore binario assume il nome di *sindrome*. Se la sindrome vale 101 allora è avvenuto un errore in posizione 5. Non ci resta che capire come assegnare le posizioni per ogni bit.

Definiamo le cifre di controllo come le colonne della tabella precedente, quindi:

1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
	c_3	c_2	c_1

Supponiamo che sia avvenuto un errore in posizione 5, quindi la terza e la prima cifra di controllo devono 'attivarsi', quindi valere 1, mentre la seconda deve rimanere a 0 (in questo modo il valore è 101 che rappresenta la posizione 5).

Quindi il bit numero 5 deve far parte delle equazioni che calcolano c_3 e c_1 , e non deve far parte dell'equazione che calcola c_2 . Per calcolare c_1 quindi, mi basta vedere su che posizioni ci sono degli 1, nel nostro esempio calcoleremo:

- c_1 sfruttando le posizioni 1, 3, 5 e 7
- c_2 sfruttando le posizioni 2, 3, 6 e 7
- c_3 sfruttando le posizioni 4, 5, 6 e 7

Quindi sfruttando queste c , se avviene un errore in posizione 5, la tripla varrà 101, visto che la posizione 5 è inclusa in c_1 e in c_3 .

Le equazioni così formate saranno linearmente indipendenti (dimostrazione balza).

In generale le cifre di controllo vengono posizionate seguendo il primo numero che codificano, quindi:

- c_1 in posizione 1
- c_2 in posizione 2
- c_3 in posizione 4

Quest'ordine, al crescere delle cifre di messaggio, crescerà seguendo l'ordine di 2^n (quindi posizione 1, 2, 4, 8, 16...).

Da notare come la distanza fra una cifra di controllo e l'altra cresce in maniera esponenziale, questo a confermare il fatto che i codici di hamming sono ottimali.

Vediamo un esempio:

- Messaggio di 4 bit: 0110
- $k = 4$

Sfruttando la disequazione $2^m \geq n + k + 1$, otteniamo che $m = 3$, quindi il numero totale di bit sarà 4+3.

Ora disponiamo le cifre di controllo, ricordando che la loro posizione vale 2^i con i che parte a 0 a m .

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ c_1 & c_2 & & c_3 & & & \end{array}$$

Quindi le 4 cifre di messaggio verranno distribuite lungo le posizioni rimanenti:

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ c_1 & c_2 & m_1 & c_3 & m_2 & m_3 & m_4 \end{array}$$

A questo punto si passa a calcolare il valore delle varie c_i in questo modo:

$$\begin{aligned} c_1 \oplus m_1 \oplus m_2 \oplus m_4 &= 0 \\ c_2 \oplus m_1 \oplus m_3 \oplus m_4 &= 0 \\ c_3 \oplus m_2 \oplus m_3 \oplus m_4 &= 0 \end{aligned}$$

In ogni riga il c_i viene calcolato in modo da avere un numero pari di 1.

Il messaggio per ora è uguale a:

$$c_1 \ c_2 \ 0 \ c_3 \ 1 \ 1 \ 0$$

Quindi passo al calcolo di c_1 :

$$\text{parity}(c_1 \ 0 \ 1 \ 0) = 0$$

Per avere parità pari c_1 deve valere 1, quindi il messaggio diventa:

$$1 \ c_2 \ 0 \ c_3 \ 1 \ 1 \ 0$$

E così via per le altre due cifre di controllo:

$$\text{parity}(c_2 \ 0 \ 1 \ 0) = 0$$

↓

$$c_2 = 1$$

$$\text{parity}(c_3 \ 1 \ 1 \ 0) = 0$$

↓

$$c_3 = 0$$

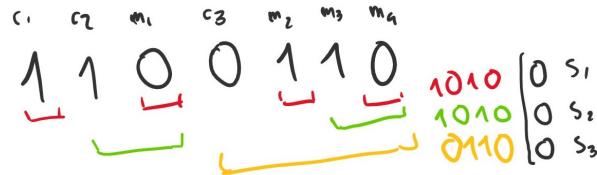
Il messaggio che verrà inviato nel canale sarà quindi:

$$1100110$$

Quando il mittente riceve il messaggio, dovrà controllare se sono avvenuti degli errori. Il messaggio, abbiamo detto, è così composto:

$$c_1 \ c_2 \ 0 \ c_3 \ 1 \ 1 \ 0$$

Il mittente calcolerà il valore della sindrome controllando le parità delle varie posizioni, un trucco per vedere visivamente in che modo vengono effettuati i calcoli è il seguente:



Quindi partendo dalla fine, il primo bit della sindrome viene calcolando facendo la parità sull'ultima cifra, terz'ultima e così via scalando sempre di 1, il secondo bit invece ne prende due alla volta e scala di due, il terzo ne prende quattro e scala di quattro. Seguendo sempre quindi lo schema di 2^n .

Se capitasse un errore? Mettiamo che avvenga un errore in posizione 5, il messaggio passa da:

1 1 0 0 1 1 0

a

1 1 0 0 0 1 0

La cosa che farà il mittente sarà quindi quello di calcolare la sindrome.

$$s_1 = \text{parity}(1000) = 1$$

$$s_2 = \text{parity}(1010) = 0$$

$$s_3 = \text{parity}(0010) = 1$$

Il valore finale sarà quindi $s_1 s_2 s_3$ uguale a 101, quindi il mittente riconosce che è avvenuto un errore in posizione 5.

E se la cifra modificata fosse di controllo? Proviamo con la seconda cifra:

1 0 0 0 1 1 0

Calcolo della sindrome:

$$s_1 = \text{parity}(1010) = 0$$

$$s_2 = \text{parity}(0010) = 1$$

$$s_3 = \text{parity}(0110) = 0$$

Quindi questo tipo di codici riconosce errori anche sulle cifre di controllo! Ma nel caso di due errori nel messaggio? Proviamo a modificare due cifre del pacchetto:

1 0 0 0 0 1 0

Calcolo della sindrome:

$$s_1 = \text{parity}(1000) = 1$$

$$s_2 = \text{parity}(0010) = 1$$

$$s_3 = \text{parity}(0010) = 1$$

Essa dice che è avvenuto un errore in posizione 7, cosa falsa. Si noti quindi come il codice di Hamming copre un singolo errore, nel caso di errori multipli essi non sono riconoscibili.

Come si gestisce il caso di 4 cifre di controllo?

$$m = 4$$

$$2^m \geq n + 1$$

$$16 \geq n + 1$$

$$n \leq 15$$

$$n = 15 \text{ (ottimale)}$$

$$k = n - m = 15 - 4 = 11$$

Vediamo come si distribuiscono le cifre di controllo nel messaggio:

c_1	c_2	m_1	c_3	m_2	m_3	m_4	c_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Come si calcolano i bit di controllo? Come prima:

$$c_1 \oplus m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} = 0$$

$$c_2 \oplus m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} = 0$$

$$c_3 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} = 0$$

$$c_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} = 0$$

Codici di Hamming ottimali

Come si può dedurre da quello spiegato sopra, i codici di Hamming definiti "ottimali" sono quelli che utilizzano tutte le configurazioni della sindrome sono utilizzate. Questo equivale a dire che:

$$2^m = n + 1 \tag{34}$$

Isolando la n , si ottiene che $n = 2^m - 1$. Questo per mostrare che i codici ottimali hanno sempre un numero di bit totali uguale a una potenza di 2 meno uno ($3, 7, 15, 31, 63, 127\dots$). Quando vale la ridondanza dei codici di Hamming ottimali?

$$R = \frac{n}{k} \tag{35}$$

$$R = \frac{k + m}{k}$$

$$R = 1 + \frac{m}{k}$$

Qual è la relazione fra k e m ? Per i codici quadrati, cubici, ecc abbiamo notato un rapporto polinomiale.

$$2^m = m + k + 1$$

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

A sinistra prevale 2^m , quindi:

$$2^m \approx k$$

Per cui possiamo riscrivere la ridondanza in questo modo:

$$R = 1 + \frac{m}{2^m}$$

oppure

$$R = 1 + \frac{\log_2 k}{k}$$

Che decresce esponenzialmente.

I casi limite di questo tipo di codici sono $m = 2$ (2 cifre di controllo e 1 di messaggio) e $m = 1$ (1 cifra di controllo), cosa che deriva dal fatto che le funzioni esponenziali crescono inizialmente in modo lento, e poi esplodono. Più sono lunghi i pacchetti, più le cifre di controllo sono "dilatate" nel messaggio (all'inizio adiacenti $[2^0 - 1]$, poi staccate di 1 $[2^1 - 1]$, poi di 3 $[2^2 - 1]$, poi di 7 $[2^3 - 1]$ e così via).

Caso particolare - Codici di Hamming a 2 cifre

Nonostante possano sembrare molto ridondanti, i codici di Hamming a due cifre di controllo possono avere un'applicazione efficace. In questo caso la banda del canale viene utilizzata solo a $\frac{1}{3}$ della sua disponibilità in quanto per mandare un bit ne invio un totale di 3, in particolare invio 000 per inviare 0, 111 per inviare 1.

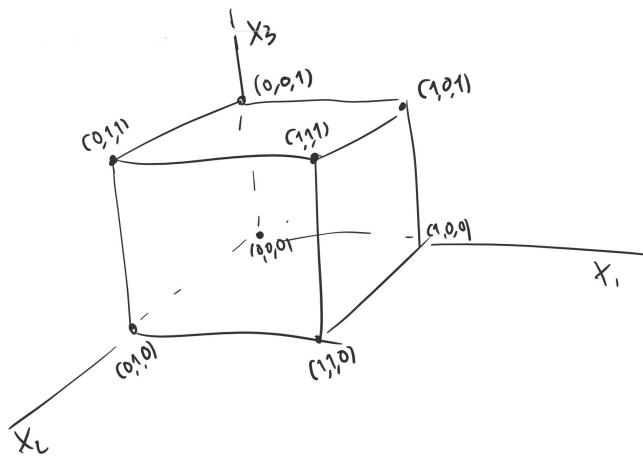
Su uno Space Shuttle, le possibilità di errore sono tante (raggi solari, temperature ecc), tutti i componenti sono triplicati. Se un pacchetto arriva con 0 e con 1 (esempio, arriva 001) allora si ragiona in probabilità, come si è visto in precedenza la probabilità che avvengano due errori (nel modello di rumore bianco), è molto più bassa di quella che avvenga un errore solo. Quindi per decodificare si fa osservare quale bit è più "usato", ad esempio se arriva 101 allora viene decodificato come 1, 001 come 0, 100 come 0 e così via.

Lezione 5

Interpretazione geometrica

Cubi

Abbiamo detto che l'insieme $\{0, 1\}^n, +_2, \cdot >$ è uno spazio vettoriale. Supponiamo di avere $n = 3$, quindi dal canale entrano ed escono vettori di 3 bit. Diciamo che ognuno di questi bit rappresenta un lato di un cubo in tre dimensioni, successivamente "appoggiamo" questo cubo all'interno di uno spazio come \mathbb{R}^3 .



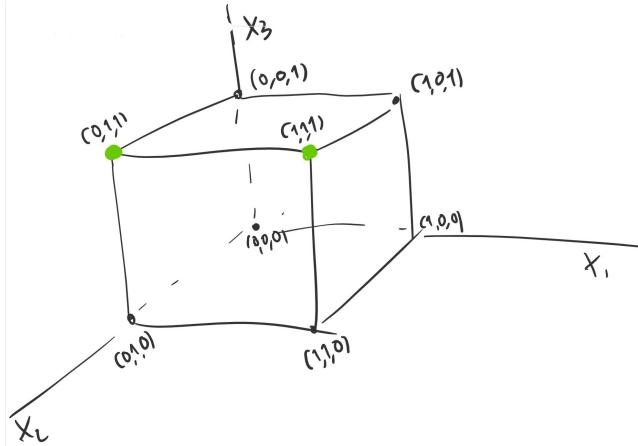
L'origine corrisponderà al messaggio $(0, 0, 0)$, ogni asse rappresenterà quindi un bit.

L'idea è ovviamente utilizzabile in n dimensione, usando quindi dei cubi di n dimensioni.

I vertici del cubo rappresentano quindi tutte le possibili sequenze di messaggi. Un sottoinsieme di questi vettori, o vertici, rappresenta i messaggi "validi", quindi senza errore. Ma cosa vuol dire dal punto di vista geometrico che un messaggio non è valido? Supponiamo di mandare un messaggio 010 , e che il rumore lo trasformi in 011 . Un errore significa che ci siamo "mossi" da un vertice ad un altro del cubo. Un errore alla prima posizione ci fa muovere lungo l'asse x_1 , un errore alla seconda posizione lungo l'asse x_2 e così via.

Il mittente e il destinatario devono quindi mettersi d'accordo su quali vertici siano "accettabili".

Mettiamo che i vertici accettabili siano 011 e 111



Ora mettiamo che nel canale il messaggio da 011 si sposti lungo la direzione x_1 , quindi il messaggio diventa 111. Esso è accettabile però!

Definiamo la distanza di Hamming: dati due vettori u e v come il numero che bit che bisogna cambiare in u per ottenere v (e viceversa).

Ad esempio la distanza di Hamming fra 011 e 111 è 1.

Possiamo definire questa distanza anche usando la somma bit a bit. $(0, 1, 1) \oplus (1, 1, 1) = (1, 0, 0)$ La distanza di Hamming è uguale a:

$$d_H(u) = \sum_{i=1}^n u_i \quad (36)$$

Che rappresenta il numero di 1 all'interno dello xor bit a bit fra i due vettori.

Tornando al nostro cubo, diciamo che due vertici adiacenti non possono essere entrambi validi visto che, a causa di un errore, ci si sposta da uno all'altro; il mittente non rileverà l'errore!

Se nel mio codice la distanza di Hamming fra codici validi è uguale a 1, non riuscirò mai a riconoscere errori.

Il numero di "passi" che devo fare da un vertice all'altro del cubo è uguale alla distanza di Hamming che c'è fra i due valori rappresentati dai vertici.

Se selezionassi le parole valide usando distanza di Hamming = 2? Posso vedere la distanza fra i vertici in questo modo:

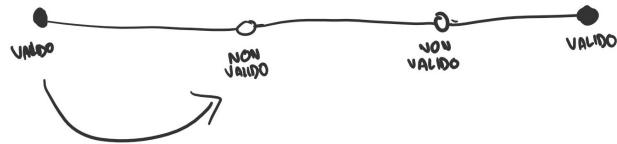


Nel caso di errore singolo, il messaggio inviato si "sposterebbe" in un vertice adiacente (quindi con distanza di Hamming = 1) che però non verrà accettato dal mittente (visto che i validi sono a distanza 2).

E per quanto riguarda distanza di Hamming = 3?



Nel caso di errore singolo, ci si sposterebbe in questo modo:



A questo punto il mittente si chiede: è più probabile che il messaggio arrivi dal vertice a sinistra (con un errore) o dal vertice di destra (con due errori)? Ovviamente è più probabile il vertice sinistro, quindi con questa distanza posso rilevare l'errore e anche correggerlo. Nel caso di due errori, però, il ricevente sbaglierebbe a correggere l'errore; seguendo l'esempio precedente andrebbe al vertice destro.

Vediamo quindi anche il caso di distanza di Hamming = 4:



È facile notare come si può correggere facilmente un errore, ma in caso di errore doppio ci si troverà ad un punto equidistante dai due estremi (ovviamente nel caso di tre errori se ne introdurrà un quarto correggendo in maniera sbagliata).

E così via, riassumiamo il tutto con questa tabella:

d_H	errori rilevati	errori corretti
1	0	0
2	1	0
3	1	1
4	2	1
5	2	2
:	:	:

Sfere

Proviamo a cambiare punto di vista, riprendiamo la rappresentazione su segmenti:

Immaginiamo che il messaggio valido inserito nel canale sia il centro di una sfera. Se il messaggio che esce fa parte di un'unica sfera, lo si può correggere "riportandolo" al centro della sfera. Se invece il messaggio fa parte di più sfere, rilevo l'errore ma non so a quale centro appartenga.

Ma come definisco una sfera nello spazio vettoriale? Dico che la sfera S è

$$S(x, r) = \{y \in \{0, 1\}^n | d_H(x, y) \leq r\} \quad (37)$$

Possiamo vedere l'effetto degli errori come una somma xor:

$$(1, 1, 0) \oplus (0, 1, 0) = (1, 0, 0)$$

In questo caso è avvenuto un posizone 2, quindi è come sommare un vettore con un 1 in posizione 2.

Se capissi qual è il vettore errore mi basterebbe ri-sommarla, per ottenere il messaggio di partenza:

$$(1, 1, 0) \oplus (0, 1, 0) = (1, 0, 0)$$

$$(1, 0, 0) \oplus (0, 1, 0) = (1, 1, 0)$$

Riconsideriamo i codici di Hamming con distanza 3, quindi a rilevamento e correzione d'errore singolo. Ci chiediamo quanto vale il volume della sfera centrata nel vettore che inviamo di raggio 1.

$$V(S(x, 1)) = 1 + n \quad (38)$$

In generale:

$$\begin{aligned} V(S(x, r)) &= 1 + n + \binom{n}{2} + \binom{n}{3} + \dots \binom{n}{r} \\ V(S(x, r)) &= \sum_{i=0}^r \binom{n}{i} \end{aligned} \quad (39)$$

Quanti messaggi validi posso creare in questo modo? Data distanza di Hamming $d_H = 3$, messaggi di n bit, quindi uno spazio grande 2^n e n sfere grandi $(n+1)$, dico che:

$$\frac{\text{volume spazio}}{\text{volume sfera}} \geq \text{numero massimo di sfere / messaggi} \quad (40)$$

Applicato al nostro esempio:

$$\begin{aligned}\frac{2^n}{n+1} &\geq 2^k \quad n = k+m \\ 2^n &\geq 2^k(n+1) \\ 2^k 2^m &\geq 2^k(n+1) \\ 2^m &\geq n+1\end{aligned}$$

Che è la stessa conclusione a cui è arrivato Hamming per i codici ottimali!

Codifica di sorgente

L'obiettivo della codifica di sorgente è quello di creare delle codeword ottimali, ad esempio codificando con codeword più piccole i caratteri più utilizzati (codice morse).

Ci sono diversi codici per la codifica di sorgente, distinguibili in due:

- Codici a blocchi: tutte le codeword hanno la stessa lunghezza ($\ell_1 = \ell_2 = \dots = \ell_q$)
 - Vantaggi: il ricevente ha 'vita facile' per decodificare in quanto sa subito quando inizia e quando finisce la codeword (sa già la lunghezza)
 - Svantaggi: se un simbolo esce con probabilità alta, allora abbiamo un grande spreco (uso gli stessi bit che userei con un simbolo usato raramente)

La lunghezza media delle codeword ($\sum_{i=1}^q p_i = 1$) è ovviamente:

$$L = \sum_{i=1}^q p_i \ell_i = \sum_{i=1}^q p_i \ell = \ell \sum_{i=1}^q p_i = \ell \quad (41)$$

uguale per tutti. Questo tipo di codice conviene quando la distribuzione delle probabilità per i simboli di uscire è uniforme.

- Lunghezza variabile: le codeword hanno lunghezze diverse, di solito i caratteri che escono con probabilità p_i maggiori sono codificati in codeword più brevi.
 - Vantaggi: C'è meno spreco di spazio, le codeword si adattano alle probabilità dei simboli
 - Svantaggi: Per il ricevente non è semplice capire quando una codeword finisce.

A causa di questo svantaggio si introducono due sottoclassi dei codici a lunghezza variabile:

- Univocamente decodificabili: quando il ricevente riceve una sequenza di codeword (es $w_1, w_2 \dots w_k$) di lunghezza diversa, allora esiste un unico modo di spezzare la sequenza di codeword
- Non univocamente decodificabili: al contrario, non esiste un unico modo di spezzare la sequenza ricevuta (ad esempio posso spezzare una sequenza sia in $s_1s_1s_2$ che in s_3s_2).

Nel corso useremo **solo** codici univocamente decodificabili.

Lezione 6

Cominciamo con un esempio: supponiamo di avere una sorgente caratterizzata da quattro simboli, codificati in questo modo:

$$s_1 \rightarrow 0$$

$$s_2 \rightarrow 01$$

$$s_3 \rightarrow 11$$

$$s_4 \rightarrow 00$$

A questo punto supponiamo che il ricevente riceva la stringa 0011, qual è la sequenza di codeword che ho ricevuto? Una soluzione potrebbe essere s_1, s_1, s_3 , ma anche s_4, s_3 . Questa codifica e questa sequenza mette in crisi il ricevitore, ed è uno dei casi da evitare (non univocamente decodificabile).

Tra i codici **univocamente decodificabili** possiamo distinguere due ulteriori sottocasi:

- Instantanei: il ricevente appena finito di ricevere una codeword la decodifica senza ambiguità.
- Non instantanei: non vale la regola precedente.

Vediamo un esempio di codice univocamente decodificabile non instantaneo:

$$s_1 \rightarrow 0$$

$$s_2 \rightarrow 01$$

$$s_3 \rightarrow 011$$

$$s_4 \rightarrow 111$$

Immaginiamo di dover decodificare la sequenza 01111111, all'inizio potrei star leggendo s_1, s_2, s_3 , dipende da cosa arriva dopo. Dopo arriva 1, e anche qui c'è ambiguità fra s_2 e s_3 , e così via.

Partendo dal fondo però, quando la trasmissione è finita, si può decodificare il messaggio partendo dalla fine:

011 111 111

$s_3 \ s_4 \ s_4$

Il lato negativo è che devo attendere la fine della trasmissione per decodificare il messaggio. I codici instantanei invece, non sono caratterizzati da questa pecca. Un codice instantaneo implica che **nessuna codeword è prefissa di un'altra**. Nel codice di prima, s_1 è prefisso di s_2 , ecc.

Vediamo questo esempio:

$$s_1 \rightarrow 0$$

$$s_2 \rightarrow 10$$

$$s_3 \rightarrow 110$$

$$s_4 \rightarrow 111$$

E decodifichiamo 1011001110 (s_2, s_3, s_1, s_4, s_1) si noti come partendo dall'inizio si può riconoscere immediatamente le codeword, senza attendere il ricevimento della sequenza completa.

Albero di decodifica

Il lavoro del ricevente per codificare si può rappresentare con un albero di decodifica. All'inizio egli si trova alla radice dell'albero, poi si sposta all'interno di esso seguendo i simboli sulle frecce:

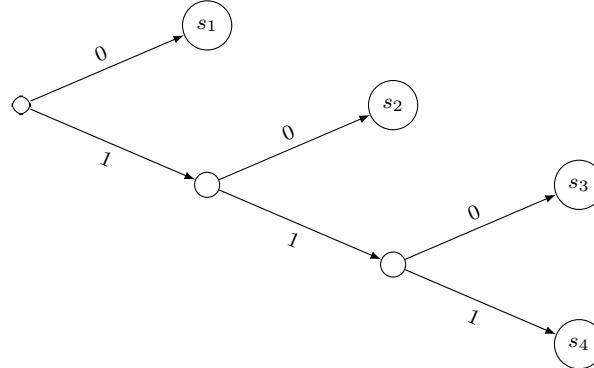


Figure 5: Albero di decodifica per $s_1 \rightarrow 0, s_2 \rightarrow 10, s_3 \rightarrow 110, s_4 \rightarrow 111$

Questa rappresentazione è molto utile in quanto si riconosce immediatamente l'istantaneità dei codici istantanei, inoltre è possibile passare dall'albero di decodifica al codice e viceversa.

Un albero è definito vuoto oppure un nodo che punta ad altri alberi (ovviamente in termini ricorsivi).

Un'altra cosa da notare è che gli alberi di decodifica sono utilizzabili solo per codici istantanei, cosa succede se non lo sono? Proviamo con:

$$\begin{aligned}s_1 &\rightarrow 0 \\ s_2 &\rightarrow 01 \\ s_3 &\rightarrow 011 \\ s_4 &\rightarrow 111\end{aligned}$$

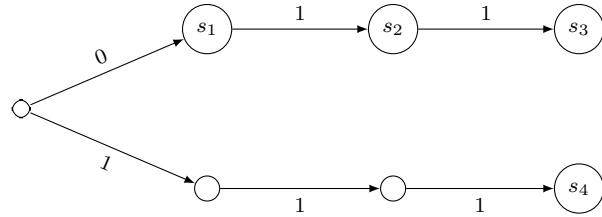


Figure 6: Albero di decodifica per $s_1 \rightarrow 0$, $s_2 \rightarrow 01$, $s_3 \rightarrow 011$, $s_4 \rightarrow 111$, codice non istantaneo

Si nota facilmente come durante il ricevimento non si può stabilire univocamente a che s_i si sta facendo riferimento (una codifica non deterministica, o meglio, lo diventa solo alla fine).

L'esempio di codice istantaneo di prima, $s_1 \rightarrow 0$, $s_2 \rightarrow 10$, $s_3 \rightarrow 110$, $s_4 \rightarrow 111$ è chiamato **comma code**, in quanto il messaggio termina o quanto arrivano tre 1 oppure quando arriva uno 0. Il relativo albero verrà disegnato "a pettine".

Vediamo un altro esempio di comma code a cinque codeword:

$$\begin{aligned}s_1 &\rightarrow 0 \\ s_2 &\rightarrow 10 \\ s_3 &\rightarrow 110 \\ s_4 &\rightarrow 1110 \\ s_5 &\rightarrow 1111\end{aligned}$$

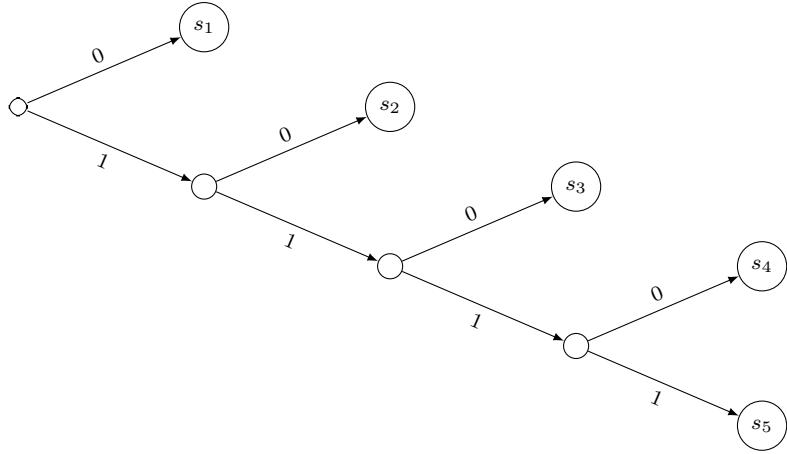


Figure 7: Albero di decodifica per $s_1 \rightarrow 0$, $s_2 \rightarrow 10$, $s_3 \rightarrow 110$, $s_4 \rightarrow 1110$, $s_5 \rightarrow 1111$

Vediamo ora un altro esempio (non comma code):

$$\begin{aligned}
 s_1 &\rightarrow 00 \\
 s_2 &\rightarrow 01 \\
 s_3 &\rightarrow 10 \\
 s_4 &\rightarrow 110 \\
 s_5 &\rightarrow 111
 \end{aligned}$$

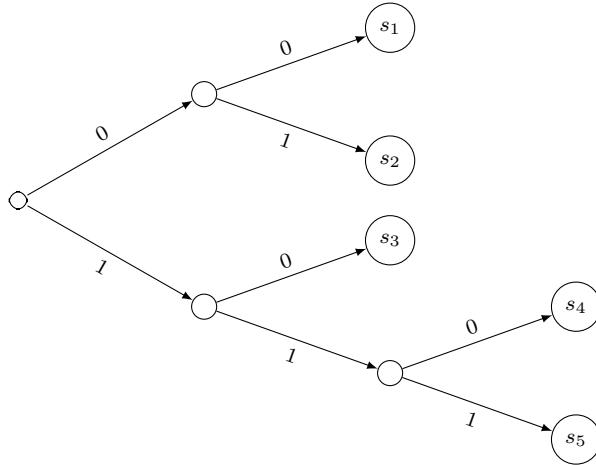


Figure 8: Albero di decodifica per $s_1 \rightarrow 00$, $s_2 \rightarrow 01$, $s_3 \rightarrow 10$, $s_4 \rightarrow 110$, $s_5 \rightarrow 111$

Dopo aver definito queste due codifiche a cinque simboli, quale sarebbe da preferire? Il criterio che utilizziamo è la lunghezza media, quindi senza sapere le probabilità dei simboli della sorgente non possiamo decidere a priori qual è più vantaggioso.

Supponiamo di avere queste probabilità: $p_1 = 0.9$, $p_2 = p_3 = p_4 = p_5 = 0.025$. Andiamo quindi a calcolare le due lunghezze medie:

$$L_{pettine} = 0.9 \cdot 1 + 0.025 \cdot 2 + 0.025 \cdot 3 + 0.025 \cdot 4 + 0.025 \cdot 4 = 1.225 \frac{\text{bit}}{\text{simbolo}}$$

$$L_{altroalbero} = 0.9 \cdot 2 + 0.025 \cdot 2 + 0.025 \cdot 2 + 0.025 \cdot 3 + 0.025 \cdot 3 = 2.05 \frac{\text{bit}}{\text{simbolo}}$$

Quindi utilizzando questa distribuzione di probabilità il comma code è da preferire. Usando la distribuzione uniforme: $p_1 = p_2 = p_3 = p_4 = p_5 = 0.2$ avremo che:

$$L_{pettine} = 0.9 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 4 + 0.2 \cdot 4 = 2.4 \frac{\text{bit}}{\text{simbolo}}$$

$$L_{altroalbero} = 0.2 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 3 = 2.05 \frac{\text{bit}}{\text{simbolo}}$$

La conclusione è che senza probabilità non possiamo trarre conclusioni sul tipo di albero da preferire per ottimizzare la lunghezza media delle codeword.

Disuguaglianza di Kraft

Una domanda che potremmo porci è: quali sono le condizioni per cui posso costruire un codice istantaneo?

Teorema 1 Condizione necessaria e sufficiente affichè esista un codice istantaneo r -ario per una sorgente S di q simboli con codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q$ è che valga:

$$K = \sum_{i=1}^q \frac{1}{r^{\ell_i}} \leq 1 \quad (42)$$

Questo teorema non dice: questo è un codice istantaneo se e solo se..., ma invece dice che esiste un codice istantaneo se e solo se...

L'utilità è la seguente: ho una sorgente con q simboli, voglio costruire un codice istantaneo r -ario con codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q$, posso? Allora calcolo K e vedo se il valore viene ≤ 1

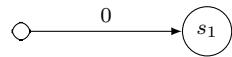
Dimostrazione 1 La condizione necessaria e sufficiente richiede di dover dimostrare queste due cose:

- Per ogni codice istantaneo r -ario per q simboli e codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q \rightarrow k \leq 1$
- $k \leq 1 \rightarrow$ esiste codice istantaneo r -ario per q simboli e codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q$

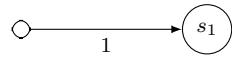
Partiamo dalla prima:

Utilizziamo come valore di $r = 2$. Parlare di codici istantanei è uguale al considerare i relativi alberi di decodifica. Possiamo quindi dimostrare per induzione sulla profondità dell'albero.

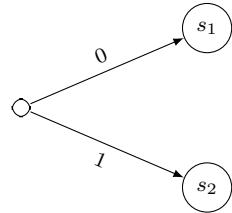
Caso base (albero di decodifica di profondità 1) I tre casi possibili sono i seguenti:



oppure



oppure



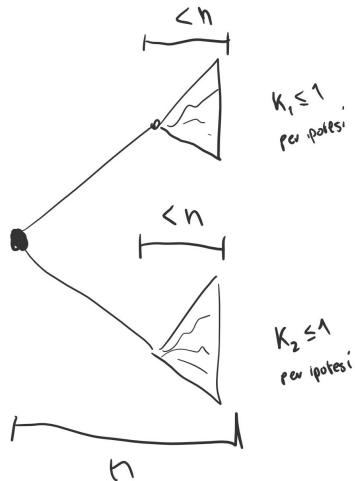
Andiamo quindi a calcolare i $K = \sum_{i=1}^q \frac{1}{r^{\ell_i}}$ dei tre alberi appena costruiti:

1. $K = \frac{1}{2^1} = \frac{1}{2} \leq 1$
2. $K = \frac{1}{2^1} = \frac{1}{2} \leq 1$
3. $K = \frac{1}{2^1} + \frac{1}{2^1} = 1 \leq 1$

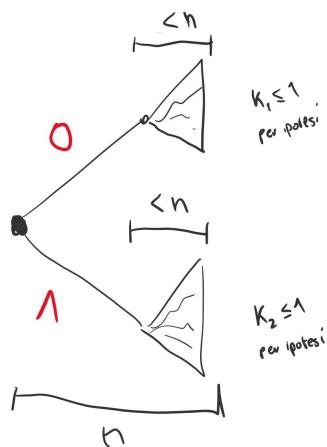
Quindi per il passo base abbiamo dimostrato.

Passo induttivo:

Supponiamo che il teorema valga per alberi di profondità $< n$ e dimostriamo che valga per alberi di profondità n (dimostro per $< n$ e non $n - 1$ perché magari il sottoalbero sopra ha profondità 1, l'altro $n - 1$, ma deve valere per entrambi). La situazione è quindi la seguente:



Quindi per ipotesi i due sottoalberi profondi al massimo $n - 1$ hanno $K \leq 1$; supponendo di 'attaccarli' ad un nuovo nodo significa appendere come prefisso di tutte le codeword un altro simbolo.



Come cambiano le K quando allungo di uno le lunghezze delle codeword?

Tutte le ℓ_i diventano $\ell_i + 1$:

$$K_1 = \sum_{i=1}^q \frac{1}{r^{\ell_i+1}} = \sum_{i=1}^q \left(\frac{1}{r^{\ell_i}} \cdot \frac{1}{r} \right) = \frac{1}{r} \sum_{i=1}^q \frac{1}{r^{\ell_i}}$$

Quindi il contributo dato da ognuno dei due sottoalberi al calcolo di K totale è dato da $\frac{1}{2}K_i$ (per $r = 2$). Il K relativo all'albero totale quindi diventa:

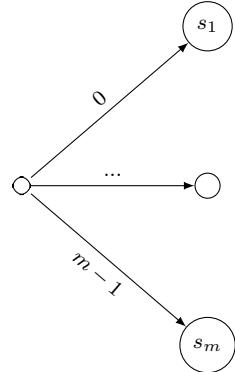
$$K = \frac{1}{2}K_1 + \frac{1}{2}K_2 = \frac{1}{2}(K_1 + K_2)$$

Sappiamo che $K_1 \leq 1$ e che $K_2 \leq 1$, quindi la loro somma sarà sicuramente ≤ 2 . La somma, moltiplicata per $\frac{1}{2}$ è ≤ 1 ; quindi possiamo scrivere:

$$K \leq 1$$

che è quello che volevamo dimostrare.

In caso n non fosse $n = 2$ bisogna solo creare più alberi di profondità 1 nel caso base:



Dato che $m \leq r$ (delle r foglie possibili ne ha solo m)

$$K = m\left(\frac{1}{r}\right) = \frac{m}{r} \leq 1$$

Nel passo induttivo il numero di sottoalberi sarà m , e quando attacchiamo i sottoalberi alla nuova radice il nuovo K verrà

$$K = \frac{1}{r}K_1 + \frac{1}{r}K_2 + \dots + \frac{1}{r}K_m$$

Dimostriamo ora la seconda parte, ovvero "k ≤ 1 → esiste codice istantaneo r-ario per q simboli e codeword di lunghezze ℓ₁, ℓ₂, ..., ℓ_q".

Partiamo dall'ipotesi che K ≤ 1, successivamente definiamo ℓ come:

$$\ell = \max\{\ell_1, \ell_2, \dots, \ell_q\}$$

Inoltre chiamo t_j il numero di codeword di lunghezza j.

Riscriviamo K in questo modo:

$$K = \sum_{i=1}^q \frac{1}{r^{\ell_i}} = \sum_{i=1}^{\ell} t_i \cdot \frac{1}{r^i} \leq 1$$

Penso riscriverlo in questo modo perché è come se stessi raccogliendo le codeword di lunghezza uguale t_i. Moltiplico per r^ℓ entrambi i membri:

$$\sum_{i=1}^{\ell} t_i r^{\ell-1} \leq r^{\ell}$$

Questo equivale a:

$$t_1 r^{\ell-1} + t_2 r^{\ell-2} + \dots + t_{\ell-1} r + t_{\ell} \leq r^{\ell}$$

Isolo t_ℓ:

$$t_{\ell} \leq r^{\ell} - t_1 r^{\ell-1} - t_2 r^{\ell-2} - \dots - t_{\ell-1} r$$

So che t_ℓ è maggiore o uguale a zero (numero di codeword di lunghezza ℓ)

$$0 \leq t_{\ell} \leq r^{\ell} - t_1 r^{\ell-1} - t_2 r^{\ell-2} - \dots - t_{\ell-1} r$$

Quindi ho posto un vincolo al numero massimo di codeword di lunghezza ℓ. Ora considero solo questo pezzo:

$$0 \leq r^{\ell} - t_1 r^{\ell-1} - t_2 r^{\ell-2} - \dots - t_{\ell-1} r$$

Divido tutto per r e porto a sinistra l'ultimo termine t_{ℓ-1}r:

$$0 \leq t_{\ell-1} \leq r^{\ell-1} - t_1 r^{\ell-2} - t_2 r^{\ell-3} - \dots - t_{\ell-2} r$$

Posso rifarlo per tutti i t:

$$0 \leq t_{\ell-2} \leq \dots$$

E mi fermo a

$$0 \leq t_2 \leq r^2 - t_1 r \tag{43}$$

$$0 \leq t_1 \leq r \tag{44}$$

Quindi sto ponendo dei limiti alle codeword di lunghezza 1, di lunghezza 2, ecc...

A questo punto, partendo dall'ultima equazione, risalgo.

Quante codeword di lunghezza 1 posso creare? (Equazione 44) Boh, di sicuro un numero compreso fra 0 e r.

Poi mi chiedo quante di lunghezza 2 (Equazione 43)? Boh quelle rimaste sono $r - t_1$, da queste posso appendere un carattere qualsiasi, quindi r . Per cui ne posso creare al massimo $r^2 - t_1 r$

Proseguendo in questo modo verifico tutte le disuguaglianze, partendo dalla sola ipotesi che $K \leq 1$

Appunti "Disugliagliaza di Kraft MacMillan.pdf"

Lezione 7

Kraft da condizione necessaria e sufficiente per fare in modo che esista un codice istantaneo r -ario di q simboli avente codeword di lunghezza $\ell_1, \ell_2, \dots, \ell_q$.

MacMillan dimostra che lo **stesso** teorema è estendibile ai codici univocamente decodificabili.



Figure 9: I codici istantanei sono un sottoinsieme dei codici univocamente decodificabili

Importante



Non è vero che il codice è istantaneo/univocamente decodificabile se $K \leq 1$
ma è vero che *esiste* un codice istantaneo con quelle caratteristiche
(r -ario con q caratteri con lunghezze $\ell_1, \ell_2, \dots, \ell_q$) se e solo se $K \leq 1$

Riprendiamo come esempio questo codice non istantaneo:

$$s_1 \rightarrow 0$$

$$s_2 \rightarrow 01$$

$$s_3 \rightarrow 011$$

$$s_4 \rightarrow 111$$

Rigirando le cifre codificate:

$$\begin{aligned}s_1 &\rightarrow 0 \\ s_2 &\rightarrow 10 \\ s_3 &\rightarrow 110 \\ s_4 &\rightarrow 111\end{aligned}$$

Otteniamo un codice istantaneo!

Effettivamente la K di Kraft accede solo alle lunghezze delle codeword (ℓ_i) e il numero di simboli con cui scriviamo le codeword (r). Questo per dire che le caratteristiche del codice possono renderlo istantaneo, ma sta poi a chi codifica farlo diventare istantaneo.

Disuguaglianza di McMillan

Teorema 2 Condizione necessaria e sufficiente affinché **esista** un codice univocamente decodificabile r -ario per una sorgente di q simboli con codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q$ è che valga:

$$K = \sum_{i=1}^q \frac{1}{r^{\ell_i}} \leq 1 \quad (45)$$

1. Per ogni codice univocamente decodificabile r -ario per una sorgente di q simboli con codeword di lunghezze $\ell_1, \ell_2, \dots, \ell_q \rightarrow K \leq 1$.
2. Se $K \leq 1$ allora esiste un codice univocamente decodificabile r -ario con lunghezze $\ell_1, \ell_2, \dots, \ell_q$.

Dimostrazione 2 Dividiamo l'enunciato del teorema nelle due considerazioni fatte sopra.

Parte 2:

Se $K \leq 1 \rightarrow$ (per Kraft) esiste un codice istantaneo r -ario con lunghezze $\ell_1, \ell_2, \dots, \ell_q$. Lo stesso codice è univocamente decodificabile (istantanei sono un sottoinsieme di univocamente decodificabili).

Parte 1:

Eleviamo K^n , dove $n > 1$ intero. Quindi avremo che:

$$K^n = \left[\sum_{i=1}^q \frac{1}{r^{\ell_i}} \right]^n$$

Definiamo $\ell = \max\{\ell_1, \ell_2, \dots, \ell_q\}$, quindi riscriviamo la sommatoria come:

$$K^n = \sum_{t=n}^{n\ell} \frac{N_t}{r^t}$$

N.B. passaggio simile a quello effettuato nella dimostrazione della seconda parte di Kraft

Da qui noto che il numeratore Nt è il numero di codeword di lunghezza t , ma questo numero non può essere maggiore di r^t visto che rappresenta tutti i possibili vettori lunghi t dove ogni elemento è un carattere dell'alfabeto r -ario.

$$Nt \leq r^t$$

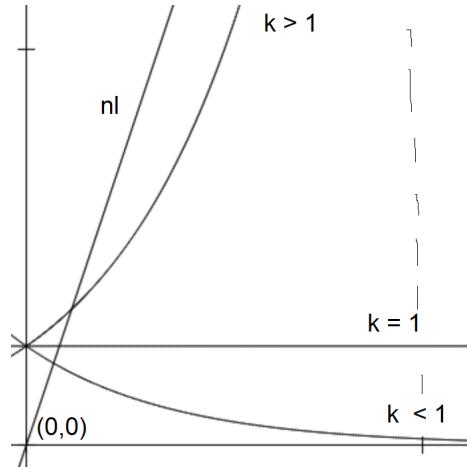
Quindi posso maggiorare la sommatoria:

$$K^n \leq \sum_{t=n}^{n\ell} \frac{Nt}{r^t} = \sum_{t=n}^{n\ell} 1 = n\ell - n + 1 = n\ell - (n - 1)$$

$(n - 1)$ è negativo visto che per ipotesi $n > 0$, quindi:

$$K^n \leq n\ell$$

Analizziamo quindi i diversi casi:



In maniera asintotica la diseguaglianza è rispettata solo da $k = 1$ e $k < 1$, quindi questo conclude la dimostrazione.

Una domanda che potrebbe sorgere è: ha senso utilizzare codici univocamente decodificabili ma non istantanei? No perché lo sforzo per trovarli è uguali, quindi inutile perder tempo su codici non istantanei.

Date le probabilità dei simboli qual è il modo più veloce per creare un codice istantaneo che minimizzi $L = \sum_{i=1}^q p_i \ell_i$?

Codici a blocchi accorciati

Si parte da codeword di lunghezza uguale per poi accorciare. Facciamo un esempio:

$$r = 2$$

$$q = 5$$

Per rappresentare 5 simboli ci servono 3 bit; tutte le possibili combinazioni sono:

000
001
010
011
100
101
110
111

Dobbiamo utilizzarne solo 5, quindi scartiamone 3

000 s_1
001
010 s_2
011
100 s_3
101
110 s_4
111 s_5

A questo punto creiamo un albero di decodifica:

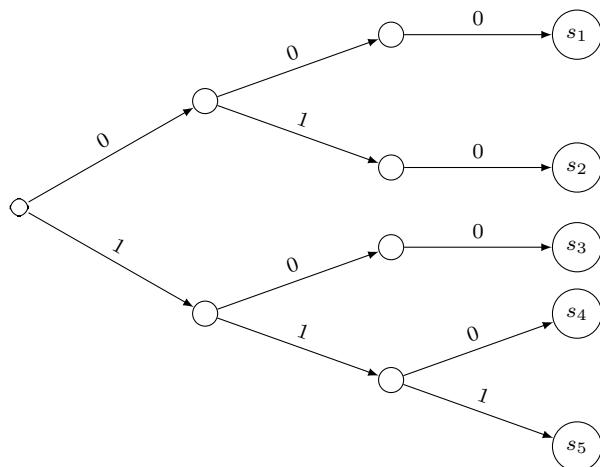


Figure 10: Albero di decodifica codice a blocchi

Partendo dall'albero di questo codice a blocchi, posso effettuare un processo di "sfoltimento" sui nodi di decisione con solo un figlio:

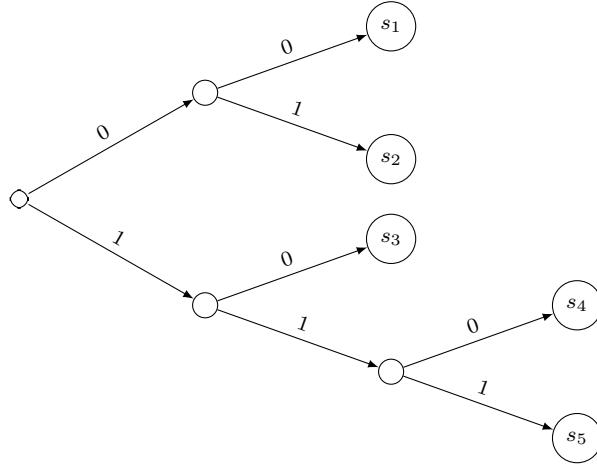


Figure 11: Albero sfoltito a partire dal precedente

Ma come facciamo ad assicurarci che questa sia la migliore codifica per questo problema?

Codici di Huffman

I codici restituiti dall'algoritmo di Huffman sono **ottimali**, quindi il valore di $L = \sum_{i=1}^q p_i \ell_i$ è minimo.

Ordiniamo le probabilità dalla più grande alla più piccola (ordine non crescente):

$$p_1 \geq p_2 \geq \dots \geq p_q$$

allora un codice ottimale deve associare queste probabilità alle lunghezze poste in ordine crescente:

$$\ell_1 \leq \ell_2 \leq \dots \leq \ell_q$$

Se questo non è verificato allora il codice non è ottimale.

Dimostrazione 3 *Assumiamo che le probabilità siano ordinate in ordine non crescente.*

Esistono due indici m e n con $m < n$ tali per cui $p_m > p_n$ e $\ell_m > \ell_n$ (quindi viola la regola di prima). Se questo è vero, posso scambiare le codeword e ottenere una lunghezza media più piccola, quindi il codice iniziale da cui siamo partiti

non era ottimale. Il contributo nella L prima dello scambio era $p_m \ell_m + p_n \ell_n$, questo contributo diventerà $p_m \ell_n + p_n \ell_m$. La differenza fra queste quantità vale

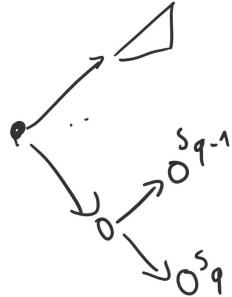
$$\begin{aligned} & p_m \ell_n + p_n \ell_m - p_m \ell_m + p_n \ell_n \\ &= p_m(\ell_n - \ell_m) - p_n(\ell_n - \ell_m) \\ &= (p_m - p_n)(\ell_n - \ell_m) \end{aligned}$$

Queste due quantità sono rispettivamente > 0 e < 0 , quindi la differenza è negativa, quindi scambiando le codeword avrò una lunghezza media minore. Quindi questo dimostra che sono partito da un codice non ottimale in quanto le lunghezze delle codeword non sono in ordine non decrescente.

Ora riconsideriamo

$$\begin{aligned} p_1 &\geq p_2 \geq \dots \geq p_{q-1} \geq p_q \\ \ell_1 &\leq \ell_2 \leq \dots \leq \ell_{q-1} \leq \ell_q \end{aligned}$$

I due simboli meno probabili avranno codeword di uguale lunghezza, per costruzione dell'albero (si apre sempre in due, quindi \downarrow)



Quindi per concludere: se le probabilità e le lunghezze seguono la regola precedente e questa regola appena spiegata è rispettata, allora il codice è ottimale. Ma come creo un codice ottimale?

Facciamo un esempio di esecuzione di Algoritmo di Huffman.
 $r = 2$, Abbiamo cinque simboli che escono con le seguenti probabilità:

$$\begin{aligned} p_1 &= 0.4 \\ p_2 &= 0.2 \\ p_3 &= 0.2 \\ p_4 &= 0.1 \\ p_5 &= 0.1 \end{aligned}$$

L'algoritmo di Huffman ha un'esecuzione di tipo greedy e funziona così:

- Prendo i due simboli meno probabili e creo un simbolo 'virtuale' combinandoli, quindi dall'esempio prima otterrò:

$$\begin{aligned} p_1 &= 0.4 \\ p_2 &= 0.2 \\ p_3 &= 0.2 \\ p_{4,5} &= 0.1 + 0.1 = 0.2 \end{aligned}$$

- Itero il passaggio precedente mantenendo traccia delle varie unioni:

$$\begin{array}{c|c|c} p_1 = 0.4 & p_1 = 0.4 & p_1 = 0.4 \\ p_2 = 0.2 & p_{3,4,5} = 0.4 & p_{2,3,4,5} = 0.6 \\ p_3 = 0.2 & p_2 = 0.2 & \\ p_{4,5} = 0.2 & & \end{array}$$

- Mi fermo quando ottengo r probabilità e assegno un carattere ciascuno:

$$\begin{aligned} p_1 &= 0.4 \rightarrow 0 \\ p_{2,3,4,5} &= 0.6 \rightarrow 1 \end{aligned}$$

- Ora itero all'indietro, appendendo un carattere alla volta nelle probabilità virtuali (appendo alla fine, se no creo prefissi e il codice non è più istantaneo), quando una virtuale ha lunghezza uguale a un altro simbolo la metto sotto.

$$\begin{array}{c|c|c|c} p_{2,3,4,5} = 0.6 \rightarrow 0 & p_1 = 0.4 \rightarrow 1 & p_1 = 0.4 \rightarrow 1 & p_1 = 0.4 \rightarrow 1 \\ p_1 = 0.4 \rightarrow 1 & p_{3,4,5} = 0.4 \rightarrow 00 & p_2 = 0.2 \rightarrow 01 & p_2 = 0.2 \rightarrow 01 \\ & p_2 = 0.2 \rightarrow 01 & p_3 = 0.2 \rightarrow 000 & p_3 = 0.2 \rightarrow 000 \\ & & p_{4,5} = 0.2 \rightarrow 001 & p_4 = 0.1 \rightarrow 0010 \\ & & & p_5 = 0.1 \rightarrow 0011 \end{array}$$

Si noti come la regola delle probabilità in ordine inverso rispetto alle lunghezze sia rispettata in tutti i passaggi.

Quanto vale la lunghezza media del codice appena calcolato?

$$L = \sum_{i=1}^q p_i \ell_i = 0.4 \cdot 1 + 0.2(2 + 3) + 0.1 \cdot (4 + 4) =$$

$$= 0.4 + 0.2 \cdot 5 + 0.1 \cdot 8 = 0.4 + 1 + 0.8 = 2.2 \frac{\text{bit}}{\text{simbolo}}$$

Proviamo a effettuare un'altra esecuzione variando una regola, nel caso due simboli abbiano stessa probabilità, metto sopra quello virtuale (prima era il contrario):

$$\begin{array}{l|l|l|l} p_1 = 0.4 \rightarrow 00 & p_1 = 0.4 \rightarrow 00 & p_{2,3} = 0.4 \rightarrow 1 & p_{1,4,5} = 0.6 \rightarrow 0 \\ p_2 = 0.2 \rightarrow 10 & p_{4,5} = 0.2 \rightarrow 01 & p_1 = 0.4 \rightarrow 00 & p_{2,3} = 0.4 \rightarrow 1 \\ p_3 = 0.2 \rightarrow 11 & p_2 = 0.2 \rightarrow 10 & p_{4,5} = 0.2 \rightarrow 01 & \\ p_4 = 0.1 \rightarrow 010 & p_3 = 0.2 \rightarrow 11 & & \\ p_5 = 0.1 \rightarrow 011 & & & \end{array}$$

Le codeword ottenute sono ovviamente diverse, ma la lunghezza media rimarrà uguale:

$$\begin{aligned} L &= \sum_{i=1}^q p_i \ell_i = 0.4 \cdot 2 + 0.2(2+2) + 0.1 \cdot (3+3) = \\ &= 0.8 + 0.2 \cdot 4 + 0.1 \cdot 6 = 0.8 + 0.8 + 0.6 = 2.2 \frac{\text{bit}}{\text{simbolo}} \end{aligned}$$

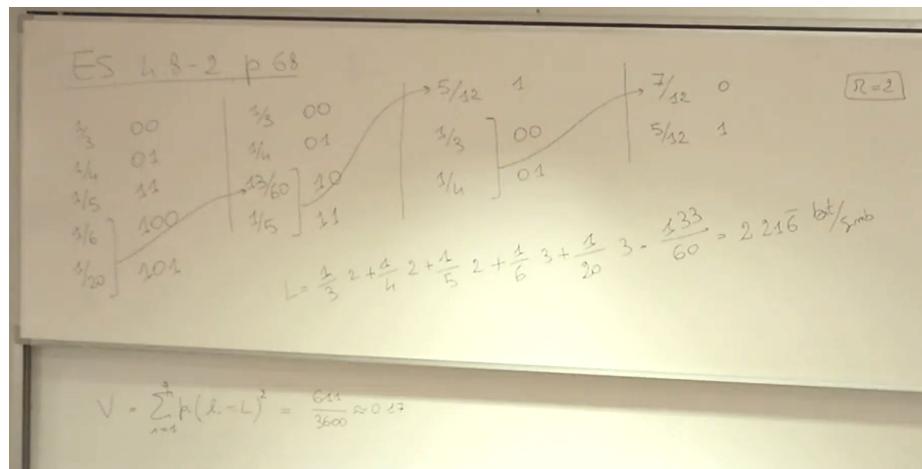
Quello che cambia sarà la varianza.

Lezione 8

Portando le probabilità virtuali uguali sopra quindi abbiamo detto si ridurrà la varianza (scelta che preferiamo per robustezza ecc).

Esercizio 4.8-2 pag.68 Creare codice ottimale per le seguenti probabilità:

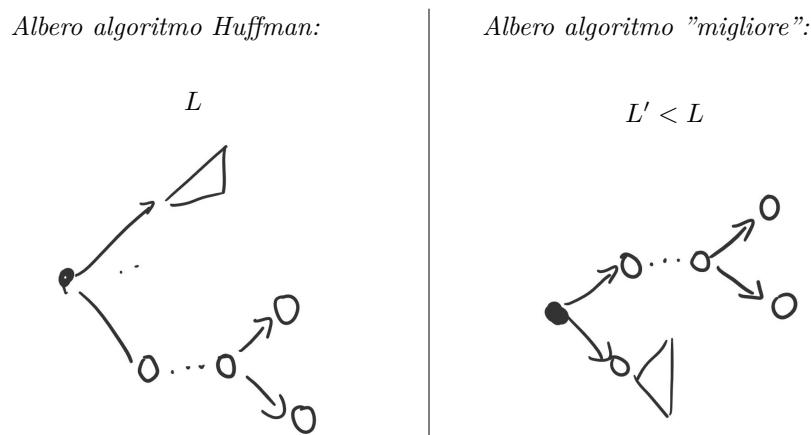
$$\frac{1}{3} \mid \frac{1}{4} \mid \frac{1}{5} \mid \frac{1}{6} \mid \frac{1}{20}$$



Non è possibile costruire codice binario istantaneo migliore (in termini di lunghezza media) di quello prodotto dall'algoritmo di Huffman

Dimostrazione 4 Per assurdo:

Consideriamo delle probabilità p_1, p_2, \dots, p_q e una L dell'algoritmo di Huffman. Successivamente consideriamo una L' ipoteticamente migliore, quindi diciamo che $L' < L$. Consideriamo l'albero i due alberi costruiti:



Ora soffermiamoci sui due codici più lunghi. Ricordiamo che in un codice ottimale, le probabilità sono ordinate in modo decrescente e le lunghezze in modo crescente. In altri termini:

$$p_1 \geq p_2 \geq \dots \geq p_{q-1} \geq p_q$$

$$\ell_1 \leq \ell_2 \leq \dots \leq \ell_{q-1} \leq \ell_q$$

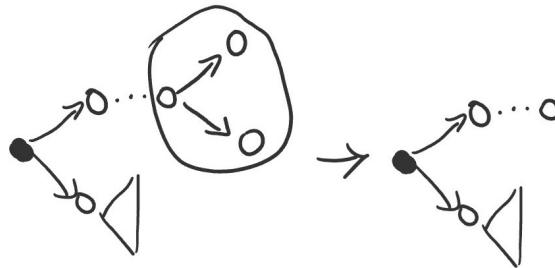
Ricordiamoci che le ultime due codeword in ordine di lunghezza devono essere uguali (pensa all'albero, se non sono uguali vuol dire che ho un nodo di decisione con un solo figlio), quindi:

$$\ell_{q-1} = \ell_q$$

Nel calcolo di L avrò che negli ultimi due termini andrò a sommare $(\ell_{q-1} \cdot p_{q-1}) + (\ell_q \cdot p_q)$, ma visto che $\ell_{q-1} = \ell_q$ è come se stessi facendo

$$\ell_q(p_{q-1} + p_q)$$

Soffermiamoci su questi due caratteri e consideriamo l'ultimo sottoalbero; sostituiamolo con un singolo nodo "virtuale" (come accade nell'algoritmo di huffman):



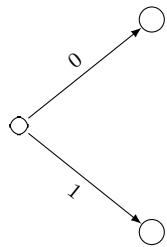
Avendo effettuato questa operazione, nella lunghezza media il contributo diventa:

$$\ell_q - 1(p_{q-1} + p_q)$$

N.B. sottraggo uno in quanto scendo di un livello nell'albero, quindi le codeword avranno un carattere in meno

Questa operazione accorcia le lunghezze medie dei due algoritmi in maniera uguale. A questo punto avrò un albero che rappresenta una sorgente di $q - 1$ simboli. Continuo iterativamente ad accorciare gli alberi risultanti, facendo ridurre la lunghezza media in modo uguale nei due algoritmi. Alla fine per Huffman arriverò all'albero con due soli simboli, la cui lunghezza media è 1, a questo punto l'altro algoritmo dovrebbe avere un'albero con lunghezza media di lunghezza < 1 , ma che non ha senso!

Albero minimale di Huffman con $L = 1$



Albero minimale dell'algoritmo migliore con $L < 1$

?

Quindi l'esistenza di tale algoritmo è assurda

Algoritmo di Huffman con $r > 2$

Ora proviamo a lavorare con $r = 3$, con cifre 0, 1, 2.

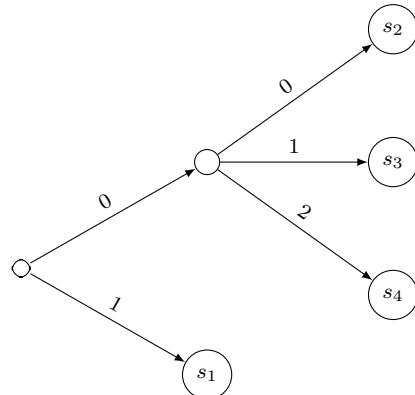
Supponiamo di avere una sorgente di quattro simboli con probabilità:

$$\begin{aligned} p_1 &= 0.4 \\ p_2 &= 0.3 \\ p_3 &= 0.2 \\ p_4 &= 0.1 \end{aligned}$$

A questo punto lanciamo l'algoritmo di Huffman:

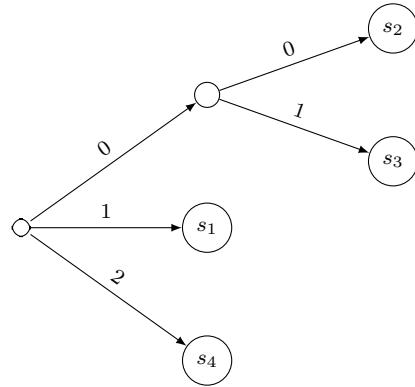
$$\begin{aligned} p_1 &= 0.4 \rightarrow 1 \\ p_2 &= 0.3 \rightarrow 00 \\ p_3 &= 0.2 \rightarrow 01 \\ p_4 &= 0.1 \rightarrow 02 \end{aligned}$$

$$\begin{aligned} p_{2,3,4} &= 0.6 \rightarrow 0 \\ p_1 &= 0.4 \rightarrow 1 \end{aligned}$$



$$L = 1 \cdot 0.4 + 2 \cdot (0.3 + 0.2 + 0.1) = 1.6 \frac{\text{cifre ternarie}}{\text{simbolo}}$$

Però questo albero non è ottimale! Lo si può correggere in questo modo:



$$L = 1 \cdot (0.4 + 0.1) + 2 \cdot (0.3 + 0.2) = 1.5 \frac{\text{cifre ternarie}}{\text{simbolo}}$$

C'è qualche problema quindi con l'algoritmo applicato ad un $r \neq 2$. In particolare vorrei che dalla radice uscissero tutti i simboli di gamma.

Nell'esempio precedente, potrei lavorare in questo modo:

$$\begin{aligned} p_1 &= 0.4 \rightarrow 0 \\ p_2 &= 0.3 \rightarrow 2 \\ p_3 &= 0.2 \rightarrow 10 \\ p_4 &= 0.1 \rightarrow 11 \end{aligned}$$

$$\begin{aligned} p_1 &= 0.4 \rightarrow 0 \\ p_{3,4} &= 0.3 \rightarrow 1 \\ p_2 &= 0.3 \rightarrow 2 \end{aligned}$$

Ma quindi quanti simboli devo raccogliere ad ogni iterazione?

L'operazione appena effettuata sopra può essere vista come un raccoglimento triplo, includendo un simbolo fittizio con probabilità 0:

$$\begin{aligned} p_1 &= 0.4 \rightarrow 0 \\ p_2 &= 0.3 \rightarrow 2 \\ p_3 &= 0.2 \rightarrow 10 \\ p_4 &= 0.1 \rightarrow 11 \\ \underline{p_5 = 0.0 \Rightarrow 12} \end{aligned}$$

$$\begin{aligned} p_1 &= 0.4 \rightarrow 0 \\ p_{3,4,5} &= 0.3 \rightarrow 1 \\ p_2 &= 0.3 \rightarrow 2 \end{aligned}$$

Quindi posso definire un'altro insieme $q' \geq q$ che include anche i simboli fittizi, per un totale di $q' - q$ simboli fittizi.

$$q' = k(r - 1) + r$$

con k che rappresenta il numero di passi (iterazioni) dell'algoritmo, ma non conosco questa quantità!

$$\begin{aligned} q' &= k(r-1) + r = (k+1)(r-1) + 1 \\ &\equiv 1 \pmod{r-1} \end{aligned}$$

Quindi devo trovare il più piccolo $q' \geq q$ tale che $q' \equiv 1 \pmod{r-1}$.

Proviamo con l'esempio di prima: devo trovare il più piccolo $q' \geq 4$ tale che $q' \equiv 1 \pmod{2}$, quindi $q' = 5$.

Il numero di simboli fintizi è quindi $q' - q = 1$.

Vediamo un esempio con una sorgente un po' più grossa con $r = 4$:

$$\begin{aligned} p_1 &= 0.22 \\ p_2 &= 0.20 \\ p_3 &= 0.18 \\ p_4 &= 0.15 \\ p_5 &= 0.10 \\ p_6 &= 0.08 \\ p_7 &= 0.05 \\ p_8 &= 0.02 \end{aligned}$$

Secondo la regola di prima dobbiamo trovare il più piccolo $q' \geq 8$ tale che $q' \equiv 1 \pmod{3}$. Il q' cercato è 10 in quanto è ≥ 8 e diviso 3 ha resto 1; in conclusione il numero di simboli fintizi sarà $q' - q = 10 - 8 = 2$.

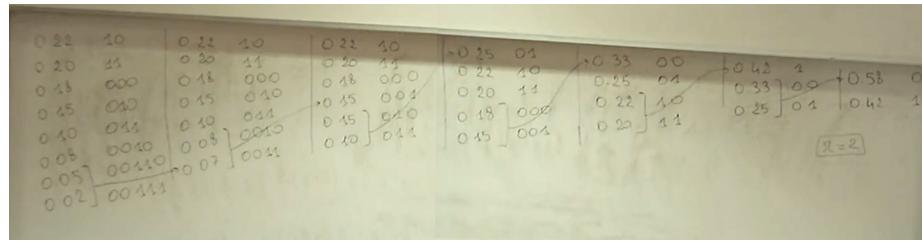
I simboli finali saranno:

$$\begin{aligned} p_1 &= 0.22 \\ p_2 &= 0.20 \\ p_3 &= 0.18 \\ p_4 &= 0.15 \\ p_5 &= 0.10 \\ p_6 &= 0.08 \\ p_7 &= 0.05 \\ p_8 &= 0.02 \\ p_9 &= 0.00 \\ p_{10} &= 0.00 \end{aligned}$$

Applichiamo quindi l'algoritmo di Huffman a questo insieme di probabilità:

$p_1 = 0.22 \rightarrow 1$	$p_1 = 0.22 \rightarrow 1$	$p_{4,5,6,7,8,9,10} = 0.4 \rightarrow 0$
$p_2 = 0.20 \rightarrow 2$	$p_2 = 0.20 \rightarrow 2$	$p_1 = 0.22 \rightarrow 1$
$p_3 = 0.18 \rightarrow 3$	$p_3 = 0.18 \rightarrow 3$	$p_2 = 0.20 \rightarrow 2$
$p_4 = 0.15 \rightarrow 00$	$p_4 = 0.15 \rightarrow 00$	$p_3 = 0.18 \rightarrow 3$
$p_5 = 0.10 \rightarrow 01$	$p_5 = 0.10 \rightarrow 01$	
$p_6 = 0.08 \rightarrow 02$	$p_6 = 0.08 \rightarrow 02$	
$p_7 = 0.05 \rightarrow 030$	$p_{7,8,9,10} = 0.07 \rightarrow 03$	
$p_8 = 0.02 \rightarrow 031$		
$\cancel{p_9 = 0.00 \rightarrow 032}$		
$\cancel{p_{10} = 0.00 \rightarrow 033}$		

Ora proviamo ad utilizzare $r = 2$ cifre sugli stessi simboli, e poi confrontiamo le lunghezze medie dei codici:



$$\begin{aligned} L_{\text{binario}} &= 0.42 \cdot 2 + 0.43 \cdot 3 + 0.08 \cdot 4 + 0.07 \cdot 4 = \\ &= 0.84 + 1.29 + 0.32 + 0.35 = 2.13 + 0.67 = 2.8 \frac{\text{bit}}{\text{simbolo}} \end{aligned}$$

Mentre per $r = 4$:

$$\begin{aligned} L_{\text{quattro}} &= 0.60 \cdot 1 + 0.33 \cdot 2 + 0.07 \cdot 3 = \\ &= 0.60 + 0.66 + 0.21 = 1.26 + 0.21 = 1.47 \frac{\text{cifre quaternarie}}{\text{simbolo}} \end{aligned}$$

Effettivamente potremmo passare da un codice all'altro utilizzando una tabella per la conversione dei codici:

Γ	binario
0	00
1	01
2	10
3	11

$p_1 = 0.22 \rightarrow 1$	$p_1 = 0.22 \rightarrow 01$
$p_2 = 0.20 \rightarrow 2$	$p_2 = 0.20 \rightarrow 11$
$p_3 = 0.18 \rightarrow 3$	$p_3 = 0.18 \rightarrow 11$
$p_4 = 0.15 \rightarrow 00$	$p_4 = 0.15 \rightarrow 0000$
$p_5 = 0.10 \rightarrow 01$	$p_5 = 0.10 \rightarrow 0001$
$p_6 = 0.08 \rightarrow 02$	$p_6 = 0.08 \rightarrow 0010$
$p_7 = 0.05 \rightarrow 030$	$p_7 = 0.05 \rightarrow 001100$
$p_8 = 0.02 \rightarrow 031$	$p_8 = 0.02 \rightarrow 001101$

E viene un codice diverso da quello calcolato prima, la lunghezza media sarà il doppio di quella precedente (ogni simbolo di Σ viene convertito in due simboli binari), quindi avrà che $L = 2 \cdot 1.47 = 2.94 \frac{\text{bit}}{\text{simbolo}}$, che è maggiore di $2.8 \frac{\text{bit}}{\text{simbolo}}$ calcolato applicando Huffman con $r = 2$.

Possiamo quindi affermare che questo tipo di "scorciatoia" non produce un codice ottimale.

Come si dimostra che il codice prodotto sia ottimale (con $r = 4$)? Allo stesso modo, si individuano le r codeword più piccole e le si raggruppano ottenendo un albero r -ario (aggiungendo il giusto numero di simboli fittizi).

Robustezza codici di Huffman

Come mai si preferiscono codici con meno varianza (quindi a parità di probabilità metto il simbolo virtuale sopra)?

Data una sorgente S noi stimiamo le probabilità di uscita dei simboli s_i osservando in un lasso di tempo più o meno lungo. In pratica calcolo dei p'_i che sarebbero delle p_i sommate ad un errore. Ovviamente deve valere che

$$\sum_{i=1}^q p'_i = 1$$

Ma dato che $p'_i = p_i + e_i$:

$$\sum_{i=1}^q p'_i = \sum_{i=1}^q p_i + \sum_{i=1}^q e_i = 1$$

Quindi

$$\begin{aligned} 1 &= 1 + \sum_{i=1}^q e_i \\ \sum_{i=1}^q e_i &= 0 \end{aligned} \tag{46}$$

Quindi gli errori in media si "compensano" a vicenda (la coperta è corta: se la tiro da una parte si scopre l'altra ;)) La media degli errori è uguale a 0:

$$\frac{1}{q} \sum_{i=1}^q e_i = 0 \tag{47}$$

La varianza di questi errori vale:

$$\sigma^2 = \frac{1}{q} \sum_{i=1}^q e_i^2 \quad (48)$$

E la lunghezza calcolata con le probabilità stimate?

$$L' = \sum_{i=1}^q p'_i \ell_i = \sum_{i=1}^q p_i \ell_i + \sum_{i=1}^q e_i \ell_I = L + \sum_{i=1}^q e_i \ell_i$$

Quindi l'obiettivo è quello di minimizzare $\sum_{i=1}^q e_i \ell_i$. Consideriamola come una funzione f di q variabili:

$$\begin{aligned} \min f(e_1, e_2, \dots, e_q) &= \sum_{i=1}^q e_i \ell_i \\ \text{t.c. } \sum_{i=1}^q e_i \ell_i &= 0 \quad \wedge \\ \frac{1}{q} \sum_{i=1}^q e_i^2 - \sigma^2 &= 0 \end{aligned}$$

Per risolvere questo problema di minimo vincolato si usano i moltiplicatori di Lagrange:

$$\mathcal{L}(l_1, \dots, l_q) = \sum_{i=1}^q e_i \ell_i - \lambda \sum_{i=1}^q e_i - \mu \left(\frac{1}{q} \sum_{i=1}^q e_i^2 - \sigma^2 \right)$$

Per risolvere questo problema uso le derivate:

$$\frac{\partial \mathcal{L}}{\partial e_i} = \ell_i - \lambda - \frac{2\mu}{q} e_i = 0 \quad \forall i \in \{1, 2, \dots, q\}$$

Derivo rispetto a \mathcal{L} :

$$\sum_{i=1}^q \ell_i - \lambda q - \frac{2\mu}{q} \sum_{i=1}^q e_i = 0$$

Ma ricordiamo che $\sum_{i=1}^q e_i = 0$, quindi

$$\lambda = \frac{1}{q} \sum_{i=1}^q \ell_i$$

Derivo rispetto a e_i :

$$\sum_{i=1}^q e_i \ell_i - \lambda \sum_{i=1}^q e_i - \frac{2\mu}{q} \sum_{i=1}^q e_i^2 = 0$$

Ma ricordiamo che $\sum_{i=1}^q e_i = 0$ e che $\frac{1}{q} \sum_{i=1}^q e_i^2 = 0$, quindi

$$\sum_{i=1}^q \ell_i - 2\mu\sigma^2 = 0$$

$$\mu = \frac{1}{2\sigma^2} \sum_{i=1}^q e_i \ell_i$$

Ora sostituiamo i moltiplicatori appena trovati all'interno di

$$\frac{\partial \mathcal{L}}{\partial e_i} = \ell_i - \lambda - \frac{2\mu}{q} e_i = 0$$

e otterremo:

$$\begin{aligned} \sum_{i=1}^q \ell_i^2 - \frac{1}{q} (\sum_{i=1}^q \ell_i)^2 - \frac{1}{q\sigma^2} (\sum_{i=1}^q e_i \ell_i)^2 &= 0 \\ (\sum_{i=1}^q e_i \ell_i)^2 &= \sigma^2 [q \sum_{i=1}^q \ell_i - (\sum_{i=1}^q \ell_i)^2] \\ \sigma^2 q^2 [\frac{1}{q} \sum_{i=1}^q \ell_i^2 - (\frac{1}{q} \sum_{i=1}^q \ell_i)^2] &= \sigma^2 q^2 [var(\ell_i)] \end{aligned}$$

Quindi otteniamo che la quantità che vogliamo minimizzare è proporzionale alla varianza delle ℓ_i moltiplicata per il numero di simboli della sorgente al quadrato e il quadrato di sigma. Quindi più piccola è la varianza di ℓ_i più è piccola la sommatoria degli errori che vogliamo minimizzare (che rappresenta l'errore con cui conosco le probabilità p'_i).

Quindi varianza minore rappresenta una minor differenza $L - L'$. In questo modo rendiamo 'robusto' il codice di Huffman: rendiamo la lunghezza media meno sensibile agli errori. All'orale interessano i concetti, non saper rifare tutto il calcolo (però sapere il procedimento e le conclusioni)

Lezione 9

Teoria dell'informazione - Entropia

Prendiamo una sorgente $S = s_1, s_2, \dots, s_q$ caratterizzato dalle probabilità p_1, p_2, \dots, p_q . Definiamo una funzione $I(s_i)/I(p_i)$ come la quantità di informazione data da un simbolo. La quantità di informazione è proporzionale alla "sorpresa" che abbiamo nel leggere il simbolo.

Dominio:

$$I : S \rightarrow \mathbb{R}$$

Inversamente proporzionale alla probabilità:

$$I(p_i) = \frac{1}{p_i}$$

Abbiamo bisogno inoltre della proprietà additiva: l'informazione di due eventi stocastici indipendenti si sommano:

$$I(p_i) + I(p_j) = \frac{1}{p_i} + \frac{1}{p_j} = \frac{p_j + p_i}{p_i p_j}$$

Ma quello che accade nella realtà è che la probabilità che avvengano due eventi è

$$I(p_i p_j) = \frac{1}{p_i p_j}$$

Quindi la funzione $I = \frac{1}{p_i}$ non gode della proprietà additiva, quindi la scartiamo.

Cerchiamo di capire come si comporta la funzione che stiamo cercando:

1. $I(p) \geq 0$ (*positività*)
2. $I(p_1 p_2) = I(p_1) + I(p_2)$ (*additività*)
se gli eventi sono indipendenti
3. I è una funzione continua di p

Notiamo che $I(p^n) = nI(p)$

Base induzione:

$$I(p) = I(p)$$

Passo induttivo:

Supponiamo che valga $I(p^{n-1}) = (n-1)I(p)$,

$$I(p^n) = I(pp^{n-1}) = I(p) + I(p^{n-1}) = I(p) + (n-1)I(p) = nI(p)$$

Ora definiamo $y = p^n$, da cui ricaviamo che $p = y^{\frac{1}{n}}$. Poi calcoliamo $I(y) = I(p^n) = nI(p) = nI(y^{\frac{1}{n}})$.

$$I(y^{\frac{1}{n}}) = \frac{1}{n}I(Y) \quad \forall n \in N$$

Ora estendiamo per tutti i numeri razionali:

$$I(y^{\frac{m}{n}}) = I((y^{\frac{1}{n}})^m) = mI(y^{\frac{1}{n}}) = \frac{m}{n}I(y)$$

Osserviamo queste proprietà, c'è una funzione che le rispetta tutte: il logaritmo.

$$I(p) = k \cdot \log p$$

per $k = -1$ ottengo:

$$I(p) = \log \frac{1}{p}$$

La proprietà di additività è rispettata:

$$I(p_1) + I(p_2) = \log \frac{1}{p_1} + \log \frac{1}{p_2} = \log \frac{1}{p_1 p_2} = I(p_1 p_2)$$

Quale base utilizziamo per il logaritmo? Essa ci fornisce l'unità di misura con cui calcoliamo la quantità di informazione. La base più utilizzata in natura è e , mentre nel mondo digitale si utilizza molto la base 2.

Quanto si calcola in \ln l'unità di misura dell'informazione è *nat*, in base 2 l'unità di misura è il *bit*, in base 10 si misura in *Hartley*. In ogni caso posso cambiare base velocemente moltiplicando per una costante C .

Ma il logaritmo è l'unica funzione che va bene per rappresentare la funzione I ?

Dimostrazione 5 Fissiamo $I(p) = \log_2 \frac{1}{p}$.
Abbiamo già visto questa proprietà:

$$I(p^n) = nI(p)$$

Supponiamo che esista una funzione g per cui $g(p^n) = ng(p)$ che sia continua, e che restituiscia valori ≥ 0 .

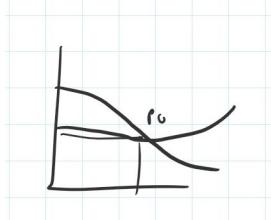
Proviamo a calcolare la differenza fra il logaritmo e questa ipotetica funzione:

$$g(p^n) - C \log_2 \frac{1}{p^n} = n[g(p) - C \log_2 \frac{1}{p}]$$

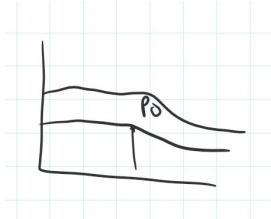
Poniamo $C = \frac{g(p_0)}{\log_2 \frac{1}{p_0}}$, con $p_0 \neq 0, 1$, quindi diverso dagli estremi. In questo modo la differenza fra le due diventa zero.

Ora possono succedere due cose:

- Due funzioni completamente diverse:



- Due funzioni che sembrano diverse ma sono la stessa funzione:



Sfrutto una proprietà dall'analisi matematica per cui ogni numero reale può essere scritto come:

$$\forall z \exists n \mid z = p_0^n$$

Quindi per $p = p_0$ posso scrivere:

$$g(z) - C \log_2 \frac{1}{z} = 0$$

Uguale a 0 in quanto $C = \frac{g(p_0)}{\log_2 \frac{1}{p_0}}$ annulla l'equazione sopra

$$g(z) = C \log_2 \frac{1}{z}$$

Quindi questa funzione si dimostra essere un logaritmo

Quindi abbiamo definito l'unica funzione possibile per definire la quantità di informazione per un evento.

Consideriamo una sorgente S che emette q simboli con p_1, p_2, \dots, p_q . Abbiamo quindi detto che:

$$I(s_i) = I(p_i) = \log_r \frac{1}{p_i}$$

Quanto è la quantità di informazione media data una sorgente? Chiamo questo valore come $H(s)$. Per calcolare questo valore peso ogni quantità di informazione:

$$H(s) = p_1 I(s_1) + p_2 I(s_2) + \dots + p_q I(s_q)$$

Riscriviamo in forma compatta:

$$H(s) = \sum_{i=1}^q p_i I(s_i)$$

$$H(s) = \sum_{i=1}^q p_i \log \frac{1}{p_i}$$

oppure scritto come

$$H(s) = - \sum_{i=1}^q p_i \log p_i$$

Questa formula appena scritta è l'**entropia** di una sorgente ed è definita come:

$$H_r(s) = \sum_{i=1}^q p_i \log_r \frac{1}{p_i}$$

L'unità di misura dell'entropia è data dalla base del logaritmo,

$$H_r(s) = H_2(s) \log_r 2$$

Ma posso cambiare velocemente base tramite una costante.

Se una sorgente fa uscire messaggi lunghi N con simboli s_q , mi aspetterò di avere

- $N \cdot p_1$ occorrenze di s_1
- $N \cdot p_2$ occorrenze di s_2
- ..
- $N \cdot p_q$ occorrenze di s_q

La probabilità P di avere un messaggio fatto in questo modo sarà:

$$P = p_1^{Np_1} \cdot p_2^{Np_2} \cdot \dots \cdot p_q^{Np_q} = [p_1^{p_1} p_2^{p_2} \dots p_q^{p_q}]^N$$

Quando esce questo messaggio, la quantità di informazione che otteniamo è $\log \frac{1}{p}$, che è quindi:

$$\begin{aligned}\log \frac{1}{p} &= \log \frac{1}{[p_1^{p_1} p_2^{p_2} \dots p_q^{p_q}]^N} = \log \left[\frac{1}{p_1^{p_1} p_2^{p_2} \dots p_q^{p_q}} \right]^N = N \log \left[\frac{1}{p_1^{p_1}} \cdot \frac{1}{p_2^{p_2}} \cdot \dots \cdot \frac{1}{p_q^{p_q}} \right] \\ &= N \sum_{i=1}^q \log \left[\frac{1}{p_i} \right]^{p_i} = N \sum_{i=1}^q p_i \log \frac{1}{p_i}\end{aligned}$$

Quindi l'entropia possiamo vederla come la quantità di informazione mediamente emessa dalla sorgente:

$$H(s) = \frac{\log \frac{1}{p}}{N}$$

Vediamo ora come è fatta questa funzione entropia:

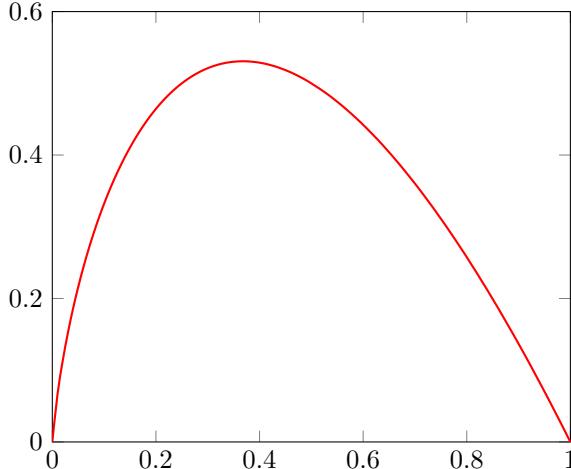


Figure 12: Grafico funzione $H_2(s)$

In $p = 0$ abbiamo un punto di discontinuità a cui ci si avvicina con pendenza infinita, quindi definiamo formalmente un'altra funzione per cui se $p = 0$ allora la funzione vale 0 (un evento impossibile non ci da informazione).

Il punto massimo vale $\frac{1}{e}$.

L'entropia $H(s)$ è sempre un valore ≥ 0 .

Quando vale 0? Essendo una somma di termini ≥ 0 , allora essa si annulla solo se tutti i termini sono uguali a 0, che è il caso in cui la sorgente ha come probabilità $p_1 = 1, p_2 = 0, p_3 = 0, \dots, p_q = 0$, quindi quando esce sempre lo stesso simbolo.

Lezione 10

Sorgente Bernoulliana

Data una sorgente $S = \{s_1, s_2\}$ in cui s_1 ha probabilità p e s_2 ha probabilità $1 - p$, la sua entropia vale

$$H_2(p) = p \log_2 \frac{1}{p} + (1 - p) \log_2 \frac{1}{1 - p}$$

Questa è una funzione a una variabile (e non due). In generale $H_r(p_1, p_2, \dots, p_q)$ è una funzione a $q - 1$ variabili. Proviamo a disegnare la funzione entropia della sorgente Bernoulliana:

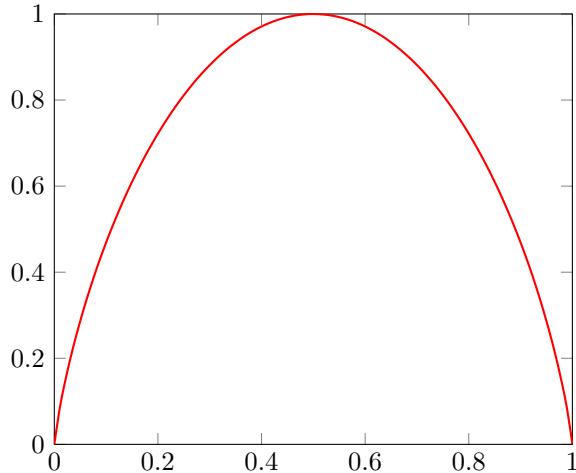


Figure 13: Grafico sorgente Bernoulliana

Sembra una parabola ma non lo è (in 0 la derivata ha pendenza infinita). Il valore massimo (1) si ottiene per $p = \frac{1}{2}$, il minimo (0) quando $p = 0 \vee p = 1$. Dove si trova invece il valore massimo per il caso generale $H_r(p_1, p_2, \dots, p_q)$?

Disuguaglianza di Gibbs

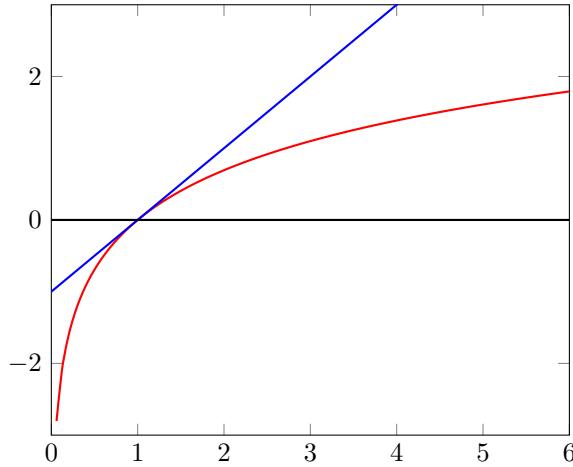


Figure 14: Grafico $y = x - 1$ (in blu) e $y = \ln(x)$ (in rosso)

Notiamo che $\ln(x) \leq x - 1 \quad \forall x > 0$ e vale $\ln(x) = x$ se $x = 1$.

Prendiamo due distribuzioni di probabilità

$$x_1, x_2, \dots, x_q$$

$$y_1, y_2, \dots, y_q$$

La diseguaglianza di Gibbs ci dice che:

$$\sum_{i=1}^q x_i \log_2 \left(\frac{y_i}{x_i} \right) \leq 0$$

Dimostrazione non richiesta all'orale, però la scrivo dai

Dimostrazione 6 Partiamo da:

$$\sum_{i=1}^q x_i \log_2 \left(\frac{y_i}{x_i} \right) = \frac{1}{\ln 2} \sum_{i=1}^q x_i \ln \left(\frac{y_i}{x_i} \right)$$

Ora sfruttiamo la maggiorazione di prima:

$$\frac{1}{\ln 2} \sum_{i=1}^q x_i \ln \left(\frac{y_i}{x_i} \right) \leq \frac{1}{\ln 2} \sum_{i=1}^q x_i \left(\frac{y_i}{x_i} - 1 \right)$$

$$\frac{1}{\ln 2} \sum_{i=1}^q x_i \ln \left(\frac{y_i}{x_i} \right) \leq \frac{1}{\ln 2} \sum_{i=1}^q (y_i - x_i)$$

$$\frac{1}{\ln 2} \sum_{i=1}^q x_i \ln\left(\frac{y_i}{x_i}\right) \leq \frac{1}{\ln 2} \left[\sum_{i=1}^q y_i - \sum_{i=1}^q x_i \right]$$

Ma essendo le due sommatorie $\sum_{i=1}^q y_i - \sum_{i=1}^q x_i$ e $\sum_{i=1}^q y_i - \sum_{i=1}^q x_i$ sono uguali a 1, allora:

$$\frac{1}{\ln 2} \sum_{i=1}^q x_i \ln\left(\frac{y_i}{x_i}\right) \leq \frac{1}{\ln 2} [0 - 0]$$

$$\sum_{i=1}^q x_i \ln\left(\frac{y_i}{x_i}\right) \leq 0$$

Questo funziona solo se $x_i = y_i$ per tutte le i , quindi se e solo se le due distribuzioni di probabilità sono uguali.

Ora possiamo usare la diseguaglianza per trovare il valore massimo dell'entropia. Calcoliamo

$$\begin{aligned} H_2(S) - \log_2 q &= \sum_{i=1}^q p_i \log_2 \frac{1}{p_i} - (\log_2 q) \sum_{i=1}^q p_i \\ &= \sum_{i=1}^q p_i \left[\log_2 \frac{1}{p_i} - \log_2 q \right] = \sum_{i=1}^q p_i \log_2 \frac{1}{p_i q} = \sum_{i=1}^q p_i \log_2 \frac{\frac{1}{q}}{p_i} \end{aligned}$$

Applico Gibbs e scopro che:

$$\sum_{i=1}^q p_i \log_2 \frac{\frac{1}{q}}{p_i} \leq 0$$

E quindi:

$$H_2(S) \leq \log_2 q$$

e l'entropia raggiunge il valore massimo (quindi $\log_2 q$) solo quando i valori di probabilità sono uguali, quindi quando la distribuzione è uniforme.

Quindi l'entropia $H_2(S)$:

- È sempre maggiore o uguale di zero, in particolare è = 0 quando la distribuzione di probabilità assegna ad una p_i valore 1 e al resto valore 0
- È sempre minore o uguale a $\log_2 q$, e raggiunge questo valore massimo quando la distribuzione di probabilità è uniforme.

Ora supponiamo di avere una sorgente $S = \{s_1, s_2, \dots, s_q\}$ che codifica un codice istantaneo r -ario con codeword di lunghezza $\ell_1, \ell_2, \dots, \ell_q$. Proviamo a mettere in relazione la lunghezza media con l'entropia della sorgente. Essendo il codice istantaneo vale la diseguaglianza di Kraft:

$$\sum_{i=1}^q \frac{1}{r^{\ell_i}} \leq 1$$

Ora definiamo per $i = 1, 2, \dots, q$

$$Q_i = \frac{r^{-\ell_i}}{K}$$

che sarebbe uno dei termini della somma fratto la somma totale

$$0 \leq Q_i = \frac{r^{-\ell_i}}{K} \leq 1$$

Posso dire che

$$\{Q_1, Q_2, \dots, Q_q\}$$

è una distribuzione di probabilità, e vale $\sum_{i=1}^q Q_i = 1$. (P.S. questo conto qua non serve orale)

Ora uso queste due distribuzioni per scrivere la diseguaglianza di Gibbs

$$\sum_{i=0}^q p_i \log_2 \frac{Q_i}{p_i} \leq 0$$

Ora lavoriamoci su:

$$\begin{aligned} & \sum_{i=0}^q p_i \log_2 \frac{Q_i}{p_i} = \\ &= \sum_{i=0}^q p_i \left[\log_2 \frac{1}{p_i} - \log_2 \frac{1}{Q_i} \right] \\ &= H_2(S) - \sum_{i=0}^q p_i \log_2 \frac{1}{Q_i} \leq 0 \\ &= H_2(S) \leq \sum_{i=0}^q p_i \log_2 \frac{1}{Q_i} \\ &= H_2(S) \leq \sum_{i=0}^q p_i \log_2 \frac{K}{r^{-\ell_i}} \\ &= H_2(S) \leq \sum_{i=0}^q p_i [\log_2 K - \log_2 r^{-\ell_i}] \\ &= H_2(S) \leq \log_2 K \sum_{i=0}^q p_i + \log_2 r^{-\ell_i} \\ &= H_2(S) \leq \log_2 K \sum_{i=0}^q p_i + \log_2 r \sum_{i=1}^q p_i \ell_i \\ &= H_2(S) \leq \log_2 K + \log_2 r \cdot L \end{aligned}$$

Noto che $K \leq 1$, quindi il logaritmo di un valore minore di 1 è ≤ 0 , quindi posso minorare ulteriormente

$$= H_2(S) \leq \log_2 r \cdot L$$

Riscrivo tutto come:

$$\frac{1}{\log_2 r} \cdot H_2(S) \leq L$$

La quantità $\frac{1}{\log_2 r}$ fa un cambio di base nel logaritmo dell'entropia, quindi:

$$H_r(S) \leq L$$

Questa diseguaglianza mi dice che data una sorgente S istantanea con una lunghezza media L , essa non può essere più piccola dell'entropia misurata in base r . Quindi la lunghezza media non può essere più piccola dell'entropia. Quando rappresento in maniera compatta è come se le stesse comprimendo delle sequenze, se scendo sotto una certa soglia vuol dire che sto perdendo informazione. Quindi l'entropia rappresenta anche una soglia minima per quanto riguarda la lunghezza media delle codeword.

In pratica più mi avvicino all'entropia più sto comprimendo la lunghezza media.

La seguente diseguaglianza vale anche per i codici univocamente decodificabili (thanks McMillan) visto che la regola sulla K vale anche su di essi.

Quando però questa diseguaglianza diventa una uguaglianza (quindi ottengo la compressione massima)?

Nel passaggio da:

$$= H_2(S) \leq \log_2 K + \log_2 r \cdot L$$

a

$$= H_2(S) \leq \log_2 r \cdot L$$

Dobbiamo fare in modo che $\log_2 K$ sia uguale a 0, ovvero quando $K = 1$.

Codifica di Shannon-Fano

Shannon e Fano partono da:

$$H_r(S) \leq L$$

che è un confronto fra medie (somma pesata vs somma pesata), ma vale la stessa cosa anche singolarmente?

$$\log_r \frac{1}{p_i} \leq \ell_i$$

Se fosse vero allora posso conoscere la lunghezza di una codeword a partire da p_i . Possiamo anche dire che ℓ_i è l'unico intero compreso in questo intervallo:

$$\log_r \frac{1}{p_i} \leq \ell_i < \log_r \frac{1}{p_i} + 1$$

Chiamo queste lunghezze ℓ_i *lunghezze di Shannon-Fano*. Ma che tipo di codici ottengo con queste lunghezze? Codici istantanei; per dimostrarlo parto dalla disuguaglianza sopra (orale chiede):

$$\begin{aligned} \log_r \frac{1}{p_i} &\leq \ell_i < \log_r \frac{1}{p_i} + 1 \\ r^{\log_r \frac{1}{p_i}} &\leq r^{\ell_i} < r^{\log_r \frac{1}{p_i} + 1} \\ \frac{1}{p_i} &\leq r^{\ell_i} < \frac{r}{p_i} \\ p_i &\geq \frac{1}{r^{\ell_i}} > \frac{p_i}{r} \end{aligned}$$

Ora sommo tutte le i :

$$\begin{aligned} \sum_{i=0}^q p_i &\geq \sum_{i=0}^q \frac{1}{r^{\ell_i}} > \sum_{i=0}^q p_i \frac{1}{r} \\ 1 &\geq K > \frac{1}{r} \end{aligned}$$

Quindi con queste lunghezze vale Kraft, quindi sono in grado di costruire un codice istantaneo. La codifica di Shannon-Fano ci dice date le probabilità come sono definite le lunghezze, ma come trovo poi le codeword? Con l'albero di decodifica.

Ma allora perchè studiare Huffman? Esso ritorna un codice ottimale, vale anche per Shannon-Fano?

Siamo sicuri che

$$L_{\text{Huffman}} \leq L_{\text{Shannon-Fano}}$$

Andiamo a calcolare la lunghezza media di un codice creato tramite Shannon-Fano:

$$\begin{aligned} \log_r \frac{1}{p_i} &\leq \ell_i < \log_r \frac{1}{p_i} + 1 \\ p_i \log_r \frac{1}{p_i} &\leq p_i \ell_i < p_i \log_r \frac{1}{p_i} + p_i \\ \sum_{i=0}^q p_i \log_r \frac{1}{p_i} &\leq \sum_{i=0}^q p_i \ell_i < \sum_{i=0}^q p_i \log_r \frac{1}{p_i} + \sum_{i=0}^q p_i \end{aligned}$$

$$H_r(S) \leq L_{\text{Shannon-Fano}} \leq H_r(S) + 1$$

E questo lo abbiamo già visto prima, però ci dice anche che non saranno mai più grandi di 1 dall'ottimo. In pratica magari non saranno sempre ottimali, però nel caso non lo fossero non sono così tanto peggiori dell'ottimale. In conclusione Huffman è più lento ma ottimale, Shannon Fano più veloce ma non sempre ottimale (ma non così male).

Vediamo un esempio di applicazione della codifica.

Sorgente con 6 simboli: $p_1 = p_2 = \frac{1}{4}$ $p_3 = p_4 = p_5 = p_6 = \frac{1}{8}$ con $r = 2$. Dobbiamo trovare il più piccolo ℓ_1 tale per cui $\log_2 \frac{1}{p_i} \leq \ell_i$.

$$\log_2 \frac{1}{\frac{1}{4}} \leq \ell_1$$

$$\log_2 4 \leq \ell_1$$

$$2 \leq \ell_1$$

$$\ell_1 = 2$$

$$\ell_2 = 2$$

Quindi ho trovato le prime due.

$$\log_2 \frac{1}{\frac{1}{8}} \leq \ell_3$$

$$\log_2 8 \leq \ell_3$$

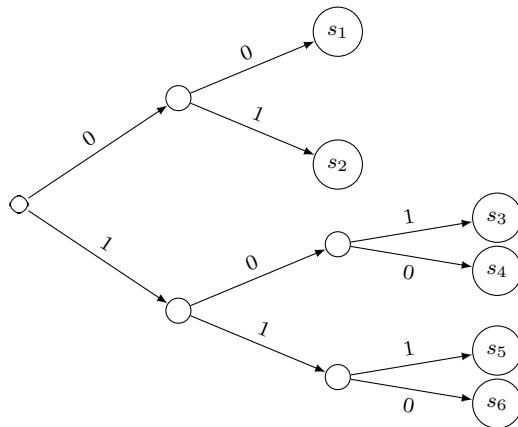
$$\ell_3 = 3$$

$$\ell_4 = 3$$

$$\ell_5 = 3$$

$$\ell_6 = 3$$

Quindi costruisco l'albero a partire dalle lunghezze trovate:



Vediamo ora un altro esempio basato su una sorgente Bernoulliana:

$$p_1 = 1 - \frac{1}{2^k} \quad (\text{n.b. k non è quella di kraft è un generico numero intero } > 1)$$

$$p_2 = \frac{1}{2^k}$$

Huffman assegnerebbe 0 al primo simbolo e 1 al secondo simbolo, quindi $L_{\text{Huffman}} = 1$.

Vediamo ora la $L_{\text{Shannon-Fano}}$, quindi calcoliamo le lunghezze singole:

$$\log_2 \frac{1}{p_1} \leq \ell_1$$

Penso vedere $\frac{1}{p_1} = \frac{2^k}{2^k - 1} = 1 + \frac{1}{2^k - 1} \leq 2$

$$\log_2 \frac{1}{p_1} \leq \log_2 2$$

$$\log_2 \frac{1}{p_1} \leq 1 \leq \ell_1$$

$$\ell_1 = 1$$

Ora la seconda:

$$\log_2 \frac{1}{p_2} = \log_2 2^k = k \leq \ell_2$$

$$\ell_2 = k$$

$$\ell_2 \geq 2$$

Ma se $k = 1000$ allora ho la seconda codeword lunga 1000, vale ancora:

$$H_r(S) \leq L_{\text{Shannon-Fano}} \leq H_r(S) + 1$$

?

Sì certo perchè è una media pesata, quindi la seconda codeword la moltiplico per $\frac{1}{2^{1000}}$

Estensione di una sorgente

Ho una sorgente S che emette i simboli s_1, s_2, \dots, s_q con probabilità p_1, p_2, \dots, p_q . Scelgo un numero $n \geq 1$ intero e definisco una nuova sorgente S^n che chiamo n -sima estensione della sorgente S .

$$S^n = S \times S \dots \times S$$

Gli elementi della sorgente S^n sono quindi i simboli dell'alfabeto, verranno

chiamati σ e σ_i è l' i -esimo simbolo della sorgente S^n è un vettore di n elementi di S .

$$\sigma_i = (S_{i1}, S_{i2}, \dots, S_{in})$$

Dove per ogni componente ho q possibilità. q^n possibili simboli. Quanto è la probabilità che esca σ_i ?

$$p(\sigma_i) = p_{i1} \cdot p_{i2} \cdot \dots \cdot p_{in}$$

Caso particolare è $n = 1$, quindi vettori composti da un solo elemento ($\sigma_i = (s_1)$) e q^1 simboli.

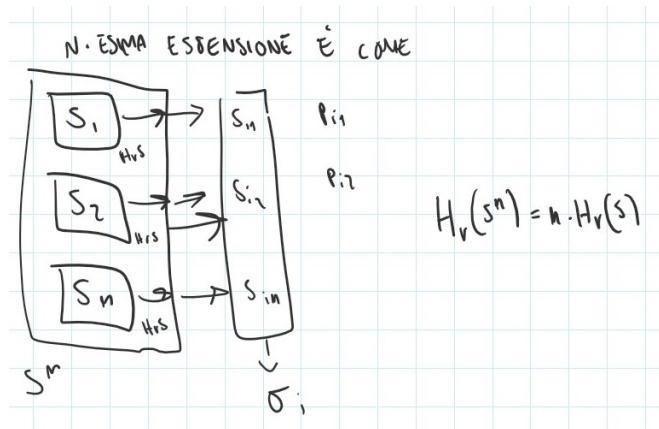
Entropia in sorgenti estese

Quanto vale l'entropia di una sorgente S^n ? Chiamiamo $Q_i = p(\sigma_i) = p_{i1} \cdot p_{i2} \cdot \dots \cdot p_{in}$.

Allora diciamo che:

$$\begin{aligned} H_r(S^n) &= \sum_{i=1}^{q^n} Q_i \log_r \left(\frac{1}{Q_i} \right) = \sum_{i=1}^{q^n} Q_i \log_r \left(\frac{1}{p_{i1} \cdot p_{i2} \cdot \dots \cdot p_{in}} \right) \\ &= \sum_{i=1}^{q^n} Q_i \sum_{k=1}^n \log_r \frac{1}{p_{ik}} \\ &= \sum_{k=1}^n \sum_{i=1}^{q^k} (p_{i1} \cdot p_{i2} \cdot \dots \cdot p_{in}) \log_r \frac{1}{p_{ik}} \\ &\quad \dots \\ H_r(S^n) &= n \cdot H_r(S) \end{aligned}$$

L'ennesima estensione dell'entropia può essere vista come n copie della sorgente che lavorano in parallelo:



Lezione 11

Riprendiamo dalla seguente disuguaglianza:

$$H_r(S) \leq L_{sf} < H_r(S) + 1$$

Primo teorema di Shannon-Fano

Proviamo ad applicarla all' n -esima estensione della sorgente: (P.S. la L è relativa a Shannon-Fano, L_n è la lunghezza media delle codeword che codificano i simboli di S^n)

$$H_r(S^n) \leq L_n < H_r(S^n) + 1$$

Da prima però sappiamo quanto vale $H_r(S^n)$:

$$n \cdot H_r(S) \leq L_n < n \cdot H_r(S^n) + 1$$

$$H_r(S) \leq \frac{L_n}{n} < H_r(S^n) + \frac{1}{n}$$

Quest'ultima disuguaglianza è uguale a quella iniziale con $n = 1$, aumentando la n l'intervallo si riduce linearmente, "si schiaccia" verso l'ottimo (teorema dei carabinieri).

La L_n è il numero medio di bit che mi servono per rappresentare σ_i ; ma diamo un'occhiata a σ_i :

$$\sigma_i = (s_{i1}, s_{i2}, \dots, s_{in})$$

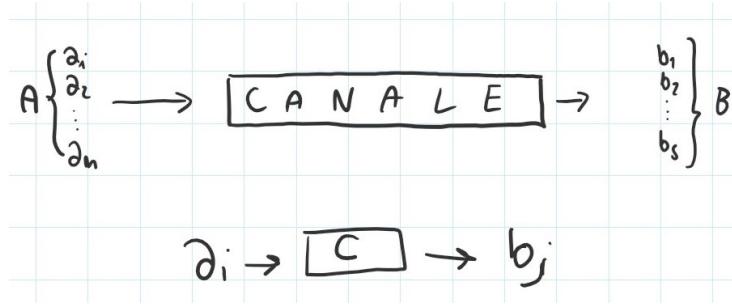
Se per rappresentare σ_i serve L_n , per rappresentare ogni s_{ij} mi servono $\frac{L_n}{n}$ lunghezze di codeword.

Quindi le lunghezze di Shannon-Fano hanno il vantaggio di avere codeword più vicine all'ottimo al crescere di n , ovviamente però non posso creare estensioni troppo grandi perché dovrei rappresentare un numero enorme di simboli.

Bello dal punto di vista teorico, ma praticamente è un disastro. Questo risultato è il primo teorema di Shannon-Fano chiamato *Noiseless coding theorem*. Fine codifica di sorgente

Canali di comunicazione

Un canale di comunicazione l'acqua, un filo elettrico, il tempo, ecc... Una definizione di un modello astratto semplice da gestire ma complicato per poterlo analizzare: Esso è costituito da:

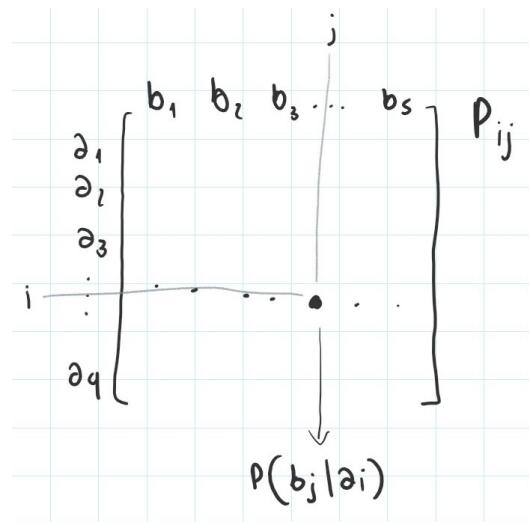


- Simboli in input
- Simboli in output
- Probabilità condizionate

Ogni volta che inserisco un simbolo a_i appartenente all'alfabeto A , il canale restituisce un simbolo b_j appartenente all'alfabeto B . Il rumore agisce sul simbolo a_i , facendo uscire un simbolo a seconda delle probabilità:

$$P(b_j|a_i)$$

Per rappresentare tutte le probabilità condizionate uso una matrice di canale:



Quindi inserito un simbolo a_i uscirà un simbolo b in output basato sulle probabilità della riga i . Ovviamente dato un simbolo ne uscirà un altro sicuramente, quindi:

$$\sum_{j=1}^s p(b_j|a_i) = \sum_{j=1}^s p_{ij} = 1$$

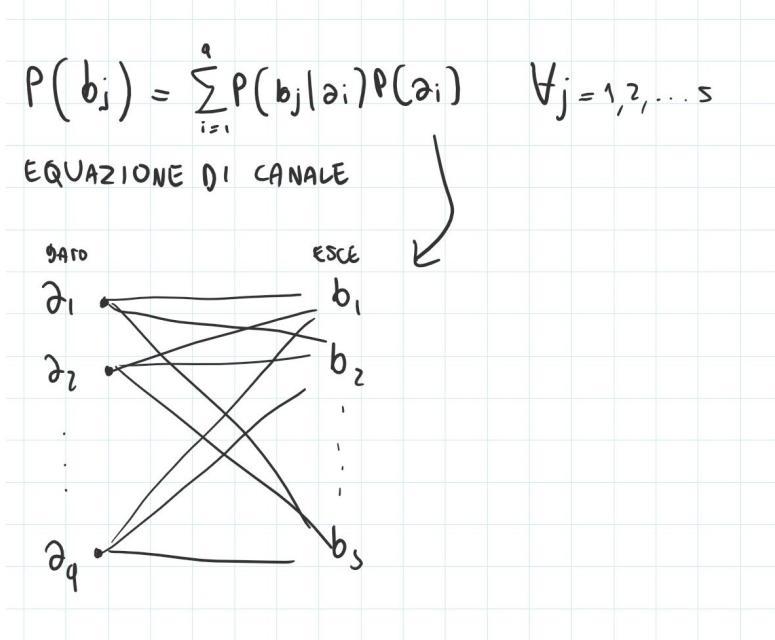
In altre parole ogni riga contiene una distribuzione discreta di probabilità, quindi la matrice è definita stocastica.

Se una riga ha tutti 0 e un 1 allora sappiamo che il simbolo su quella riga avrà sempre lo stesso simbolo in output (magari il canale è costruito in un certo modo, boh).

$p(a_i)$ è la probabilità con cui scegliamo il simbolo a_i da inserire all'interno del canale. Quindi diciamo che l'utente inserisce nel canale il simbolo a_i con questa probabilità. Ci chiediamo quanto è la probabilità che esca b_j (con $j = 1, 2, \dots, s$). Definiamo l'equazione di canale:

$$p(b_j) = \sum_{i=1}^q p(b_j|a_i)p(a_i)$$

Quindi la probabilità di b_j è la somma di più probabilità, in particolare ogni termine è la probabilità che esca b_j dato che è stato inserito in ingresso a_1 oppure a_2 oppure $a_3\dots$. Il concetto viene più chiaro tramite questo grafo:



Vediamo ora due casi particolari (estremi, nella pratica non dovrebbero mai accadere):

- Canale con assenza di rumore:

Ogni volta che inserisco a_i ho probabilità 1 che esca un certo b_j . Quindi la matrice di canale avrà tutte righe composte da 0 e 1. L'equazione di canale diventa:

$$p(b_j) = \sum_{i=1}^q p(b_j|a_i)p(a_i) = 1 + 0 + \dots + 0 = 1$$

- Canale completamente rumoroso:

Ogni simbolo inserito in input è completamente ignorato e l'output è totalmente casuale. In pratica le b_j hanno una distribuzione di probabilità che dipende dal rumore e non da a_i . L'equazione di canale è:

$$p(b_j) = \frac{1}{s} \quad \forall j$$

Probabilità condizionate all'indietro

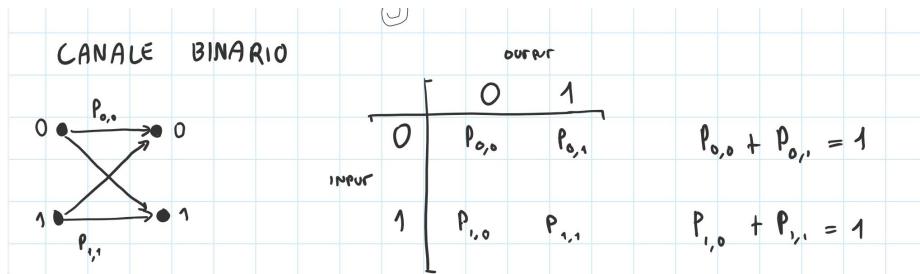
Quando inserisco un simbolo in input a_i sappiamo che la probabilità che esca b_j è $p(b_j|a_i)$, ma quanto è la probabilità 'al contrario'? Ovvero, quanto è la probabilità che sia stato inviato a_i , visto che mi è arrivato b_j ? Teorema di Bayes:

$$p(a_i|b_j) = \frac{p(b_j|a_i)p(a_i)}{p(b_j)}$$

E usando l'equazione di canale:

$$= \frac{p(b_j|a_i)p(a_i)}{\sum_{i=1}^q p(b_j|a_i)p(a_i)}$$

Consideriamo il canale più semplice (canale binario):



La probabilità $p_{0,0} = p_{1,1}$ viene chiamata P , essa rappresenta la probabilità che inserito un bit dal canale esca lo stesso bit. Chiamo $Q = 1 - P$ la probabilità che la trasmissione sia errata.

La matrice di canale diventa:

$$\begin{pmatrix} P & Q \\ Q & P \end{pmatrix}$$

Questo tipo di canale è chiamato *canale binario simmetrico* (CBS) in quanto la matrice che lo definisce è simmetrica.

Usando tanti canali CBS, voglio inviare un messaggio lungo n bit, qual è la probabilità che il primo bit arrivi a destinazione sbagliato?

Q .

E che il secondo sbagli? Q . E così via. I canali sono paralleli quindi gli eventi sono indipendenti (se uno sbaglia gli altri non vengono influenzati). Questa descrizione è già stata vista: è il rumore bianco. L'ennesima estensione della CBS (CBS^n) è il canale che rappresenta il rumore bianco.

Chiamo a l'input del CBS e b l'output. Chiamo inoltre p la probabilità di scegliere $a = 0$; $p(a = 0)$. Quindi $1 - p$ è $p(a = 1)$. Chiamiamo x la $p(b = 0)$ e $1 - x = p(b = 1)$. Le equazioni di canale ci dicono che:

$$p(b = 0) = pP + (1 - p)Q$$

$$p(b = 1) = (1 - p)P + pQ$$

Le probabilità all'indietro sono:

$$p(a = 0|b = 0) = \frac{pP}{pP + (1 - p)Q}$$

Lezione 12

Riprendendo le probabilità all'indietro di prima:

$$\begin{aligned} p(a = 0|b = 0) &= \frac{pP}{pP + Q(1 - p)} \\ p(a = 1|b = 1) &= \frac{P(1 - p)}{pQ + P(1 - p)} \\ p(a = 1|b = 0) &= \frac{Q(1 - p)}{pP + Q(1 - p)} \\ p(a = 0|b = 1) &= \frac{pQ}{pQ + P(1 - p)} \end{aligned}$$

Vediamo un esempio: $P = \frac{9}{10}$, $Q = 1 - P = \frac{1}{10}$. L'utente digita con probabilità $\frac{19}{20}$ il bit 0, mentre con $\frac{1}{20}$ il bit 1. Quindi:

$$p(a = 0) = \frac{19}{20}$$

$$p(a = 1) = \frac{1}{20}$$

Vediamo le probabilità all'indietro, quindi come il ricevente deve interpretare il messaggio:

$$\begin{aligned} p(a = 0 | b = 0) &= \frac{171}{172} \\ p(a = 1 | b = 1) &= \frac{9}{28} \\ p(a = 1 | b = 0) &= \frac{1}{172} \\ p(a = 0 | b = 1) &= \frac{19}{28} \end{aligned}$$

Quindi se il ricevente ha inviato 0 confronta le probabilità che il mittente abbia inviato effettivamente 0 oppure 1:

$$p(a = 0 | b = 0) = \frac{171}{172} \quad vs \quad p(a = 1 | b = 0) = \frac{1}{172}$$

Domina la probabilità che abbia inviato 0, quindi leggo il messaggio proprio come 0.

Ora mettiamo che arrivi un 1:

$$p(a = 0 | b = 1) = \frac{19}{28} \quad vs \quad p(a = 1 | b = 1) = \frac{9}{28}$$

La probabilità che abbia inviato 0 è ancora più alta! Quindi il mittente leggerà sempre 0!

Come mai succede questo?

Il problema è che

$$\begin{aligned}
 & \begin{cases} p(a=0|b=0) > p(a=1|b=1) \\ p(a=0|b=1) > p(a=1|b=0) \end{cases} \\
 & \begin{cases} Pp > Q(1-p) \\ Qp > P(1-p) \end{cases} \\
 & \begin{cases} Pp > (1-P)(1-p) \\ Qp > P(1-p) \end{cases} \\
 & \begin{cases} \cancel{Pp} > 1 - P + \cancel{Pp} - p \\ Qp > P(1-p) \end{cases} \\
 & \begin{cases} 0 > 1 - P - p \\ Qp > P(1-p) \end{cases} \\
 & \begin{cases} 0 > 1 - P - p \\ (\cancel{1-p})p > P(\cancel{1-p}) \end{cases} \\
 & \begin{cases} p > 1 - P \\ p > P \end{cases} \\
 & \begin{cases} p > Q \\ p > P \end{cases}
 \end{aligned}$$

Il problema qui è che l'utente ha utilizzato una distribuzione di probabilità per i simboli molto diversa dalla distribuzione uniforme.

Entropia congiunta

Possiamo vedere l'uscita del canale come se fosse una sorgente, quindi posso chiedermi quanto vale la quantità di informazione relativa a b_j e rispetto a a_i , e quale rapporto c'è fra le due?

Ci sono quindi tre punti di vista: vedo solo A , vedo solo B o le vedo entrambe e mi chiedo in che rapporto sono.

Partendo da A :

$$H_r(A) = \sum_{i=1}^q p(a_i) \log_r \frac{1}{p(a_i)}$$

possiamo quindi scrivere anche B :

$$H_r(B) = \sum_{j=1}^s p(b_j) \log_r \frac{1}{p(b_j)}$$

Dunque si può anche scrivere una entropia condizionata: un entropia sui simboli di A dato che dal canale è uscito un simbolo b_j .

$$H_r(A|b_j) = \sum_{i=1}^q p(a_i | b_j) \log_r \frac{1}{p(a_i | b_j)}$$

Allo stesso modo posso anche calcolare $H_r(A|B)$, che rappresenta la quantità di informazione che mediamente è uscita dal canale dato che ho visto uscire uno qualsiasi dei simboli di B

$$H_r(A|B) = \sum_{j=1}^s p(b_j) H_r(A | b_j)$$

$$H_r(A|B) = \sum_{j=1}^s \sum_{i=1}^q p(a_i, b_j) \log_r \frac{1}{p(a_i | b_j)}$$

Questa quantità viene detta **equivocazione**. Essa rappresenta l'effetto del rumore nel canale, un legame statistico fra i simboli di A e i simboli di B che escono

Ora vediamo dall'altro verso, posso quindi considerare $H(B | a_i)$ quindi una quantità di informazione media su B dato che so che è stato inserito un simbolo a_i nel canale.

$$H_r(B|a_i) = \sum_{j=1}^s p(b_j | a_i) \log_r \frac{1}{p(b_j | a_i)}$$

Quindi ora come prima facciamo una media pesata (questa volta sugli a_i):

$$H_r(B|A) = \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(b_j | a_i)}$$

L'ultima entropia da definire è l'entropia congiunta:

$$H_r(A, B) = \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(a_i, b_j)}$$

Vediamo ora diversi casi sulle probabilità:

- $p(a_i, b_j) = p(a_i)p(b_j) \forall i, j$

Questo accade solo quando le due probabilità sono indipendenti, quindi quando inserire a_i non influenza l'uscita di b_j , cosa che avviene solo nei

canali completamente rumorosi.

In questo caso l'entropia congiunta diventa:

$$\begin{aligned}
 H_r(A, B) &= \sum_{i=1}^q \sum_{j=1}^s p(a_i)p(b_j) \log_r \frac{1}{p(a_i)p(b_j)} \\
 H_r(A, B) &= \sum_{i=1}^q \sum_{j=1}^s p(a_i)p(b_j) [\log_r \frac{1}{p(a_i)} + \log_r \frac{1}{p(b_j)}] \\
 H_r(A, B) &= \sum_{j=1}^s p(b_j) \sum_{i=1}^q p(a_i) \log_r \frac{1}{p(a_i)} + \sum_{i=1}^q p(a_i) \sum_{j=1}^s p(b_j) \log_r \frac{1}{p(b_j)} \\
 H_r(A, B) &= 1 \cdot H_r(A) + 1 \cdot H_r(B)
 \end{aligned}$$

Quindi nel caso di canale completamente rumoroso ho che l'entropia congiunta $H_r(A, B)$ che è la quantità di informazione media che ottengo guardando contemporaneamente la sorgente A e la sorgente B (terzo punto di vista di prima).

- $p(a_i, b_j) = p(a_i)p(b_j | a_i) \forall i, j$

Questo è il caso 'normale', quindi c'è rumore bianco:

$$H_r(A, B) = \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(a_i)} + \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(b_j | a_i)}$$

Ora sfruttiamo una proprietà statistica per cui:

$$\sum_{j=1}^s p(a_i, b_j) = p(a_i)$$

Questa proprietà si chiama *marginale* di a_i . Quindi proseguo in questo modo:

... conti ...

$$H_r(A, B) = H_r(A) + H_r(B|A)$$

Quindi la somma della quantità di informazione effettivamente inserita nel canale più l'equivocazione. Quest'ultima agisce come rumore. Se non ci fosse rumore l'entropia $H_r(A, B) = H_r(A)$.

C'è una simmetria in tutte queste formule fra A e B , scambiandoli i risultati non cambiano, infatti possiamo esprimere l'entropia congiunta anche in questo modo:

$$H_r(A, B) = H_r(B) + H_r(A|B)$$

Mutua informazione

Quanto è la quantità di informazione che viene trasmessa all'interno del canale? Quella che viene trasmessa dall'inizio alla fine del canale. In altre parole la quantità di informazione massima che un canale può trasportare. Questa caratteristica non dipende da come l'utente usa il canale ma è una caratteristica intrinseca dello stesso. Essa però non potrà mai superare la capacità del canale.

$$\begin{aligned} I(a_i; b_j) &= \log_r \frac{1}{p(a_i)} - \log_r \frac{1}{p(a_i | b_j)} = \log_r \frac{p(a_i | b_j)}{p(a_i)} \\ &= \log_r \frac{p(a_i | b_j) \cdot (b_j)}{p(a_i) \cdot p(b_j)} = \log_r \frac{p(a_i, b_j)}{p(a_i) \cdot p(b_j)} \end{aligned}$$

Inoltre

$$= I(a_i; b_j) \leq I(a_i) = \log_r \frac{1}{p(a_i)}$$

Questo perché

$$= I(a_i; b_j) = \log_r \frac{p(a_i | b_j)}{p(a_i)} = \log_r p(a_i | b_j) + I(a_i)$$

$p(a_i | b_j) \leq 1$ e il logaritmo di un valore ≤ 1 è negativo, quindi sto riducendo il valore.

Se le distribuzioni di probabilità di a_i e b_j sono indipendenti, allora $I(a_i; b_j) = 0$ visto che nelle equazioni di prima questo termine:

$$\log_r \frac{p(a_i | b_j)}{p(a_i) \cdot p(b_j)}$$

si annulla.

Ora proviamo a calcolare le medie come fatto prima con le entropie:

$$I(A; b_j) = \sum_{i=1}^q p(a_i | b_j) I(a_i; b_j) = \sum_{i=0}^q p(a_i | b_j) \log_r \frac{p(a_i | b_j)}{p(a_i)}$$

Ora posso fare lo stesso sui simboli di B :

$$I(a_i; B) = \sum_{j=1}^s p(b_j | a_i) \log_r \frac{p(b_j | a_i)}{p(b_j)}$$

Ora arriviamo alla quantità che ci interessa, ovvero la mutua informazione del sistema:

$$I(A; B) = \sum_{i=1}^q p(a_i) I(a_i; B) = \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{p(a_i, b_j)}{p(a_i)p(b_j)} = I(B; A)$$

La mutua informazione del sistema gode di tre proprietà:

- Quantità sempre maggiore o uguale a 0 ($I(A; B) \geq 0$)
La mutua informazione è un prodotto fra due distribuzioni di probabilità: $p(a_i, b_j)$ e $\log_r \frac{p(a_i, b_j)}{p(a_i)p(b_j)}$ quindi usiamo la disuguaglianza di Gibbs.
- Quantità uguale a 0 se e solo se A e B sono indipendenti ($I(A; B) = 0$)
- La mutua informazione di $A; B$ è uguale alla mutua informazione di $B; A$ ($I(A; B) = I(B; A)$)

Vediamo ora le relazioni che ci sono fra mutua informazione e le varie entropie definite prima

$$\begin{aligned}
I(A; B) &= \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) [\log_r p(a_i, b_j) - \log_r p(a_i) \log_r p(b_j)] = \\
&= - \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(a_i, b_j)} + \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(a_i)} + \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{1}{p(b_j)} \\
&= -H_r(A, B) + H_r(A) + H_r(B) \geq 0
\end{aligned}$$

Posso quindi dire due cose:

$$I(A; B) = -H_r(A, B) + H_r(A) + H_r(B) \geq 0$$

e un'altra considerazione sulle entropie:

$$H_r(A, B) \leq H_r(A) + H_r(B)$$

Quindi la quantità di informazione media non supera in ogni caso la somma fra le singole.

Inoltre ricordiamo che:

$$H_r(A, B) = H_r(A) + H_r(B|A)$$

$$H_r(A, B) = H_r(B) + H_r(A|B)$$

Quindi

$$I(A; B) = H_r(B) - H_r(B|A) \geq 0$$

$$I(A; B) = H_r(A) - H_r(A|B) \geq 0$$

Da qui ricaviamo anche che

$$H_r(B|A) \leq H_r(B)$$

$$H_r(A|B) \leq H_r(A)$$

Capacità del canale

La capacità del canale è la massima quantità di informazione che può transitare nel canale. La mutua informazione mi dice che:

$$I(A; B) = \sum_{i=1}^q \sum_{j=1}^s p(a_i, b_j) \log_r \frac{p(a_i, b_j)}{p(a_i)p(b_j)}$$

Però questa quantità dipende da a_i e io non voglio scrivere la capacità del canale basandomi sulle probabilità con cui l'utente inserisce un simbolo (la capacità di un tubo non dipende da quanta acqua ci metti dentro). Ma non posso togliere $p(a_i)$ visto che $p(b_j)$ dipende da essa. Quindi la mutua informazione non può essere usata.

Definiamo la capacità del canale come:

$$C = \max(I(A; B))$$

Questa è una formula astratta ma che da un'idea di quanto sia difficile calcolare la vera capacità di un canale.

Ci sono dei canali più "semplici" da trattare? C'è una categoria di canali chiamata *canali uniformi*.

$$\begin{aligned} I(A; B) &= H_r(B) - H_r(B|A) = \\ &= H_r(B) - \sum_A \sum_B p(a, b) \log_r \frac{1}{p(b|a)} \\ &= H_r(B) - \sum_A p(a) \sum_B p(b|a) \log_r \frac{1}{p(b|a)} \end{aligned}$$

Ora soffermiamoci su $\sum_B p(b|a) \log_r \frac{1}{p(b|a)}$: ci sono dei canali in cui questa somma ha un valore indipendente da a , se vediamo ad esempio la matrice associata al canale ogni riga ha una somma sempre uguale, quindi ogni riga è una permutazione dell'altra. Questi canali sono i canali uniformi.

$$\begin{aligned} \sum_B p(b|a) \log_r \frac{1}{p(b|a)} &= W \\ &= H_r(B) - \sum_A p(a) \cdot W \\ &= H_r(B) - W \end{aligned}$$

Lezione 13

Abbiamo detto che la capacità è difficile da calcolare perché dipende dalla distribuzione dei simboli di ingresso e di uscita (che poi dipendono da quelli in entrata).

Quindi abbiamo detto magari ci sono dei canali in cui questi conti sono più semplici: i canali uniformi.

La matrice di questi canali contiene righe che sono l'una la permutazione dell'altra, ad esempio il canale binario simmetrico:

$$\begin{pmatrix} P & Q \\ Q & P \end{pmatrix}$$

Perchè il canale uniforme ci semplifica il conto? Perchè la mutua informazione del sistema può essere scritta come:

$$\begin{aligned} I(A; B) &= H_r(B) - H_r(B | A) = H_r(B) - \sum_A \sum_B P(a, b) \log_r \frac{1}{P(B | A)} \\ &= H_r(B) - \sum_A p(a) \underbrace{\sum_B P(b | a) \log_r \frac{1}{P(b | a)}}_* \end{aligned}$$

Andiamo a considerare la sommatoria (*), noi stiamo considerando la matrice di canale (dove ogni cella è $p(b | a)$), quindi il primo termine della somma, ci stiamo muovendo su una riga fissata (quella a_i).

Dato che il canale è uniforme, il risultato su ogni riga è uguale in quanto ogni riga è una permutazione dell'altra (li sommo solo in ordine differente). Diciamo che:

$$w = \sum_B P(b | a) \log_r \frac{1}{P(b | a)} = H_r(B | a)$$

Quindi w ha un valore fissato che è uguale per tutte le righe e non dipende da a , quindi ottengo:

$$\begin{aligned} &= H_r(B) - w \sum_A p(a) \\ &= H_r(B) - w = I(A; B) \end{aligned}$$

e questo è più semplice da calcolare rispetto a $H_r(B) - H_r(B | A)$.

Facciamo un esempio con un canale binario simmetrico.

$$\begin{pmatrix} P & Q \\ Q & P \end{pmatrix}$$

La capacità è uguale a $H_2(B) - w$ dove

$$w = P \log_2 \frac{1}{P} + Q \log_2 \frac{1}{Q} =$$

$$= P \log_2 \frac{1}{P} + (1 - P) \log_2 \frac{1}{1 - P} = H_2(P)$$

$$= Q \log_2 \frac{1}{1 - Q} + Q \log_2 \frac{1}{Q} = H_2(Q)$$

Entrambi i valori sono uguali all'entropia di una bernoulliana.

$$I(A; B) = H_2(B) - H_2(P) = H_2(B) - H_2(Q)$$

Ricordiamo che

$$C = \max(H_2(B) - H_2(P))$$

Siccome $H_2(P)$ è una costante, occupiamoci solo di massimizzare $H_2(B)$. Abbiamo già visto che questa funzione ha valore massimo 1 quando $x = \frac{1}{2}$. Quanto è però il valore di p tale per cui $H_2(B)$ ha come $x = \frac{1}{2}$?

$$x = \frac{1}{2} = pP + (1 - p)Q$$

È quindi la probabilità che venga mandato un 0 e arrivi uno 0 più la probabilità che venga mandato 1 e arrivi 0.

$$\begin{aligned} &= pP + (1 - P) - p(1 - P) \\ &= 1 - P - p + 2pP \\ &= 1 - P - p(1 - 2P) \\ &\quad \downarrow \\ &p(1 - 2P) = \frac{1}{2} - P \\ &p(\cancel{1 - 2P}) = \frac{\cancel{1 - 2P}}{2} \end{aligned}$$

Posso semplificare in quanto $(1 - 2P) = 0$ solo in caso di $P = \frac{1}{2}$ e quindi di canale totalmente rumoroso.

$$p = \frac{1}{2}$$

Quindi esiste un valore di p che mi consente di ottenere $x = \frac{1}{2}$ per cui $H_2(B)$ ottiene valore massimo, quindi ho che:

$$C = 1 - H_2(P)$$

$$C = 1 - H_2(Q)$$

Secondo teorema di Shannon

Questo teorema mette insieme tutto quello che abbiamo detto, cioè raccoglie cosa vuol dire fare trasmissioni all'interno di un canale rumoroso applicando codici a correzione d'errore e cercando di sfruttare il canale al miglior modo possibile, quindi cercando di avvicinarsi al più possibile alla capacità di canale.

Teorema 3 *Dato un canale rumoroso è sempre possibile avvicinarsi in maniera arbitraria alla capacità del canale mantenendo contemporaneamente arbitrariamente bassa la probabilità d'errore.*

Quindi abbiamo un canale rumoroso, e questo canale ha una certa capacità (il massimo al variare delle $p(a)$ della mutua informazione del sistema).

Questo teorema dice che è possibile avvicinarsi arbitrariamente al valore della capacità (tipo asintoto). Avvicinarsi vuol dire trovare delle codeword, o una codifica, per cui ci si avvicina. Avvicinandomi alla capacità del canale inserisco sempre più informazione, e mandandone sempre di più ovviamente sbaglia sempre di più, e sbagliando di più si aumenta la probabilità che il ricevente sbagli a correggere una codeword (due errori e si corregge male).

La 5 si riferisce alla probabilità che il ricevente *corregga male* una codeword.

Dimostrazione - Secondo teorema di Shannon

Preliminari matematici:

- $$\sum_{k=0}^{\lambda n} \binom{n}{k} \leq 2^{nH(\lambda)} \quad \text{per } 0 < \lambda < 1 \quad \text{e } \lambda n \text{ intero}$$

Dove $H(\lambda)$ è l'entropia di una sorgente bernoulliana con due simboli di probabilità λ e $1 - \lambda$.

- Legge debole dei grandi numeri:

Supponiamo di avere n variabili casuali indipendenti fra loro (x_1, x_2, \dots, x_n) tutte con la stessa distribuzione di probabilità caratterizzata da media μ e varianza σ^2 .

Allora abbiamo che per ogni $\epsilon > 0$ e per ogni $\delta > 0$ esiste un interno n_0 tale che per ogni $n \geq n_0$ vale che:

$$P\left(\left|\frac{1}{n} \sum_{i=1}^n X_i - \mu\right| \leq \epsilon\right) \geq 1 - \delta$$

oppure in maniera equivalente:

$$P\left(\left|\frac{1}{n} \sum_{i=1}^n X_i - \mu\right| > \epsilon\right) < \delta$$

Notiamo che $\frac{1}{n} \sum_{i=1}^n X_i$ è la media campionaria dei miei n esperimenti, mentre μ è la media teorica. Quindi sottraendoli in valore assoluto mi chiedo quanto sono distanti (teoria e pratica).

La probabilità che questa distanza superi un valore ϵ piccolo a piacere può essere resa piccola a piacere δ .

Nel nostro caso ogni esperimento sarà l'invio di un bit all'interno di un canale binario simmetrico.

Metto n copie del canale binario simmetrico (n -esima estensione) indipendenti fra loro, quindi la probabilità teorica di ottenere errori è $Q^n(\mu)$.

In pratica inviare pacchetti tramite n -esima estensione CDS (CDS^n) equivale a trasmettere pacchetti di n bit col modello del rumore bianco.

Assumeremo che $Q < \frac{1}{2}$ (se fosse maggiore metto una porta `not`, se uguale ho canale completamente rumoroso).

- Regola di decisione: essa è la regola/funzione che il ricevente applica per cercare di capire quale secondo lui è il simbolo che è stato spedito. Esempio (probabilità all'indietro $P(a_i | b_j)$):

$$\begin{array}{ccc} & b_1 & b_2 & b_3 \\ a_1 & \left(\begin{array}{ccc} 0.6 & 0.5 & 0.4 \\ 0.4 & 0.5 & 0.6 \end{array} \right) \\ a_2 & & & \end{array}$$

Assumiamo che la distribuzione delle a_i sia uniforme, quindi $p(a_1) = p(a_2) = \frac{1}{2}$.

La funzione di decisione di massima verosimiglianza $d : B \rightarrow A$ è definita in questo modo:

- $d(b_1) = a_1$
- $d(b_3) = a_2$
- $d(b_2) = a_1 \vee d(b_2) = a_2$

Da notare come questa regola non abbia un unico risultato (b_2). La regola di verosimiglianza funziona in questo modo:

$$d(b_j) = a^* \quad \text{t.c.} \quad P(a^* | b_j) \geq P(a_i | b_j) \quad \forall i$$

Applicando Bayes sto dicendo che:

$$\frac{p(b_j | a^*) p(a^*)}{p(b_j)} \geq \frac{p(b_j | a_i) p(a_i)}{p(b_j)}$$

Abbiamo detto che, non sapendo a priori la distribuzione dei simboli di ingresso, assumo che sia uniforme; quindi:

$$p(b_j | a^*) \frac{1}{2} \geq p(b_j | a_i) \frac{1}{2}$$

$$p(b_j | a^*) \geq p(b_j | a_i)$$

- Capacità di un CBSⁿ: $n \times C$, dove $C = n(1 - H_2(P)) = n(1 - H_2(Q))$

Dimostrazione:

N.B. a_i e b_j sono pacchetti di n bit.

I messaggi validi fra tutte le n -ple di bit di A è $|A| = M \leq 2^n$. Diciamo inoltre che sono equiprobabili, quindi $p(a_i) = \frac{1}{M}$, quindi la quantità di informazione $I(a_i) = \log_2 \frac{1}{M} = \log_2 M$.

Il nostro obiettivo è quello di avvicinarci alla capacità del canale, che è:

$$\text{Capacità CBS}^n = n \times C$$

Scelgo un $\epsilon_1 > 0$ piccolo a piacere, e chiedo che la quantità di informazione si avvicini per meno di una quantità proporzionale a ϵ_1 alla capacità, quindi:

$$I(a_i) = nC - \epsilon_1$$

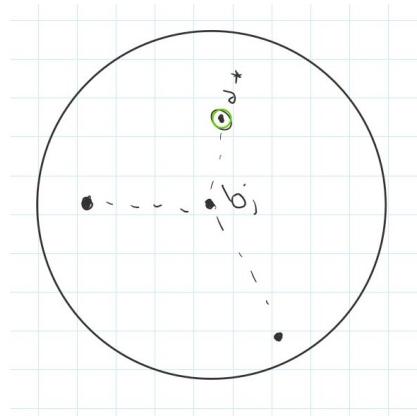
Anche se in realtà questa ϵ_1 dovrebbe funzionare (è più *fair*) in relazione al numero di estensioni n , quindi:

$$I(a_i) = n(C - \epsilon_1)$$

Da questa relazione possiamo ricavare M :

$$M = 2^{n(C - \epsilon_1)} = \frac{2^{nC}}{2^{n\epsilon_1}}$$

Ora mettiamoci nei panni del ricevente: una volta ricevuto b_j si costruisce una iper-sfera centrata in b_j e avrà un raggio lungo nQ in quanto mi aspetto nQ errori. Ora osservo che b_j non sia un codice valido, quindi devo scegliere il messaggio più vicino a b_j all'interno della sfera.



Quindi, data la distanza D fra b_j e a^* la probabilità

$$p(b_j | a^*) = Q^D(1 - Q)^{n-D}$$

per $Q < \frac{1}{2}$ è decrescente al crescere di D , quindi la probabilità che ottenga questo b_j da a_i è tanto più bassa quanto sono lontani a_i e b_j , quindi la distanza non è solo topologica.

In altre parole scelgo una a^* che mi minimizza questa quantità: $Q^D(1 - Q)^{n-D}$. Il problema è che può essere complicato trovare quello più vicino e che può esserci una a^* che ha la stessa distanza fra due b_j (le regole di massima verosimiglianza non sono uniche).

Lezione 14

Quindi scartiamo la regola di verosimiglianza, anzi la semplifichiamo (la dimostrazione funzionerebbe a maggior ragione anche per la regola più complessa).

La probabilità di commettere un errore è

$$P_E = P[a_i \notin S(r)]$$

Dove S è la sfera centrata in b_j e r è il raggio $n(Q + \epsilon_2)$ (sommo ϵ_2 così per la legge debole dei grandi numeri posso rendere trascurabile la probabilità che stia fuori).

ϵ_2 è la dimensione della corona circolare fuori dalla sfera centrata in b_j !

$$P_E = P[a_i \notin S(r)] + P[a_i \in inS(r)] \cdot P[\text{almeno un altro msg valido } \in S(r)]$$

Quindi ci dev'essere un unico messaggio valido all'interno della sfera, se c'è diciamo che è a^* e finito, altrimenti è avvenuto un errore (o c'è un unico messaggio valido ma non è quello spedito, oppure è il messaggio valido ma ce n'è anche un altro).

Ora maggiore P_E sapendo che $P[a_i \in inS(r)] \leq 1$:

$$P_E \leq P[a_i \notin S(r)] + P[\text{almeno un...} \in S(r)]$$

$$P[\text{almeno un...}] \leq \sum_{a \in A - \{a_i\}} P[a \in S(r)]$$

$$P_E \leq P[a_i \notin S(r)] + \sum_{a \in A - \{a_i\}} P[a \in S(r)]$$

Quest'ultima è la quantità che voglio rendere minima. Parto dal primo termine e uso la legge debole dei grandi numeri:

$$\forall \epsilon_2, \delta > 0, \exists n_0 \text{ t.c. } \forall n > n_0$$

↓

$$P[\text{numero errori} > nQ + n\epsilon_2] < \delta$$

$$P\left[\left|\frac{\text{numero errori} - nQ}{n}\right| \leq \epsilon_2\right] > 1 - \delta$$

$$P\left[\left|\frac{\sum_i X_i - nQ}{n}\right| > \epsilon_2\right] < \delta$$

$$P\left[\left|\frac{\frac{\sum_i X_i}{n} - Q}{n}\right| > \epsilon_2\right] < \delta$$

$$P\left[\frac{1}{n} \sum_i X_i - Q > \epsilon_2\right] < \delta$$

Così è in forma per la legge: media campionaria meno media teorica.

Devo dimostrare che la somma, passo alla media e calcolo la probabilità d'errore calcolata come media di probabilità di errore su tutti i codici.

$$\tilde{P}_E < \delta + \sum_{a \in A \setminus \{a_i\}} \tilde{P}[a \in S(r)]$$

Ma $\tilde{P}[a \in S(r)]$ comprende tutte le scelte di A , quindi non dipende più da a :

$$\tilde{P}_E < \delta + \sum_{a \in A \setminus \{a_i\}} \leq \delta + (M - 1)\tilde{P}[a \in S(r)]$$

$$\tilde{P}_E < \delta + \sum_{a \in A \setminus \{a_i\}} \leq \delta + (M)\tilde{P}[a \in S(r)]$$

$$\text{con } a \neq a_i$$

Quindi ora bisogna calcolare \tilde{P} , che è la probabilità media che a appartenga alla sfera di raggio r (visto nell'interpretazione geometrica!).

$$\tilde{P}[a \in S(r)] = \frac{N(r)}{2^n}$$

Stiamo lavorando in uno spazio di Hamming, quindi abbiamo che $N(r)$ è 1 (centro sfera) più $\binom{n}{1}$ (numero di punti a distanza 1)...

$$N(r) = 1 + \binom{n}{1} + \binom{n}{2} \dots$$

$$N(r) = \sum_{k=0}^r \binom{n}{k}$$

r è uguale a $n(Q + \epsilon_2)$, sappiamo che $Q < \frac{1}{2}$ e che ϵ_2 può essere piccolo a piacere, quindi metto $Q + \epsilon_2 < \frac{1}{2}$.

Ora sfrutto una proprietà matematica definita inizialmente:

$$\sum_{k=0}^{\lambda n} \binom{n}{k} \leq 2^{nH(\lambda)} \quad \text{con } 0 < \lambda < 1$$

Ora pongo $\lambda = Q + \epsilon_2$, quindi :

$$r = \lambda n$$

Quindi

$$N(r) = \sum_{k=0}^r \binom{n}{k} \leq 2^{nH(Q + \epsilon_2)}$$

Ricordiamo che:

$$\tilde{P}[a \in S(r)] = \frac{N(r)}{2^n}$$

Quindi abbiamo che

$$\tilde{P}[a \in S(r)] \leq 2^{-n[1-H(Q+\epsilon_2)]}$$

Quindi tornando a:

$$\tilde{P}_E < \delta + \sum_{a \in A} \underbrace{\tilde{P}[a \in S(r)]}_{\{a_i\}} \leq \delta + (M)\tilde{P}[a \in S(r)]$$

otteniamo che

$$\tilde{P}_E \leq \delta + M2^{-n[1-H(Q+\epsilon_2)]}$$

Ora notiamo che:

$$1 - H(Q + \epsilon_2) = \underbrace{1 - H(Q)}_{=C} + H(Q) - H(Q + \epsilon_2)$$

$$1 - H(Q + \epsilon_2) = C - [H(Q + \epsilon_2) - H(Q)]$$

Ora analizziamo la differenza $H(Q + \epsilon_2) - H(Q)$:

$$H(Q + \epsilon_2) \leq H(Q) + \epsilon_2 \frac{dH}{dx}|_Q$$

(spiegato in min 41 col grafico)

$$\frac{dH}{dx}|_Q = \log \frac{1}{Q} - \log \frac{1}{1-Q} = \log \frac{1-Q}{Q}$$

Siccome $Q < \frac{1}{2}$ abbiamo che $2Q < 1$, quindi $Q+Q < 1$, e ho che $Q < 1-Q$, divido per Q e ho che $\frac{1-Q}{Q} > 1$, quindi:

$$\log \frac{1-Q}{Q} > 1$$

Quindi concludendo,

$$1 - H(Q + \epsilon_2) \geq C - \underbrace{\epsilon_2 \log \frac{1-Q}{Q}}_{\text{Costante per costante, quindi } \epsilon_3}$$

$$1 - H(Q + \epsilon_2) \geq C - \epsilon_3$$

Ora riprendiamo

$$\tilde{P}_E \leq \delta + M2^{-n[1-H(Q+\epsilon_2)]}$$

e sostituiamo la potenza:

$$\tilde{P}_E \leq \delta + M 2^{-n[C-\epsilon_3]}$$

Ricordiamo che M è il numero di messaggi validi scelto calibrando ϵ_1 per avvicinarci il più possibile alla capacità di canale.

$$\begin{aligned}\tilde{P}_E &\leq \delta + 2^{n(C-\epsilon_1)} \cdot 2^{-n[C-\epsilon_3]} \\ \tilde{P}_E &\leq \delta + 2^{-n(\epsilon_1-\epsilon_3)}\end{aligned}$$

Dove (ϵ_1 e ϵ_3 si scelgono in modo che la loro differenza dev'essere maggiore di 0). In questo modo al crescere di n , $2^{-n(\epsilon_1-\epsilon_3)}$ si avvicina a 0. Quindi esiste almeno un codice che si avvicina alla capacità di canale, lavorando su ϵ_1 , mantenendo arbitrariamente bassa la probabilità di errore, lavorando su ϵ_2 (o ϵ_3). Questo avviene mediamente, quindi a maggior ragione esiste almeno un codice che si avvicina alla capacità di canale e mantiene bassa la probabilità di errore.

Il problema è che il teorema ci dice che esiste, ma non ci dice come costruirlo. Inoltre è un risultato asintotico, quindi per estensioni di n molto grandi.

Poi è dimostrato per rumore bianco, ma in realtà la struttura dei canali è diversa (burst di errori ecc...).

Ad esempio se ogni posizione del pacchetto ha probabilità di errori diverse, allora al posto di una sfera viene un ellisse strana e i conti diventano complicati.

Si usano i LDPC (Low Density Parity Code), presi tantissimi bit in ingresso (ordine di decine di migliaia) e si costruiscono un po' di equazioni di parità (come Hamming, sistema di equazioni). Si usano poche equazioni di parità (poca densità) e si fanno un po' di esperimenti al pc per capire quanta densità dare.

Inverso Teorema di Shannon

Innanzitutto mostriamo la disuguaglianza di Fano:

$$H(A|B) \leq H(P_E) + P_E \log(M - 1)$$

Dove $H(P_E)$ è l'entropia di una sorgente bernoulliana che ha due simboli: uno che ha P_E probabilità di uscire e l'altro $1 - P_E$.

Si sceglie un numero di messaggi $M = 2^{(C+\epsilon)}$ (andiamo quindi oltre alla capacità di canale, sommando ϵ). Ogni messaggio ha probabilità uguale.

$$I(a_i) = \log M = n(C + \epsilon) = nC + \underbrace{n\epsilon}_{\text{quantità che supera la capacità di canale}}$$

La mutua informazione del sistema:

$$H(A) - H(A|B) \leq nC$$

perchè la capacità è il massimo della mutua informazione, quindi non può superarla!

Ricordiamo che

$$H(A) = \sum_A \frac{1}{M} \log M$$

$$H(A) = \frac{1}{M} \log M \sum_A 1$$

$$\sum_A 1 = M$$

$$H(A) = \frac{1}{M} \log M \cdot M$$

$$H(A) = \log M$$

Se ogni volta che viene scelto un simbolo la quantità di informazione è sempre $\log M$ allora la quantità di informazione è $\log M$
manca un po di rob

Lezione 14 - Parte 2

Crittografia

Che cos'è la crittografia? Lo studio delle tecniche che permettono di elaborare, memorizzare e trasmettere dati in presenza di agenti ostili (una persona o dispositivo che cerca di leggere dati salvati, disturbare le computazioni ecc..).

Il nome deriva dalla composizione di due parole greche (*kryptòs*: nascosta e *graphia*: scrittura). Da non confondere con la steganografia (nascondere messaggi dove solo in pochi sanno dove andare a cercare). Il messaggio in crittografia è pubblico a tutti, ciò che è nascosto è il significato del messaggio (o il contenuto).

Prima del 1976 esistevano solo crittosistemi simmetrici, da quell'anno nasce la crittografia a chiave pubblica.

La crittografia ha due obiettivi:

- Studiare e implementare *crittosistemi*
- Analizzare crittosistemi con l'obiettivo di trovare vulnerabilità (*crittoanalisi*)

Tipi di cifrari:

- Simmetrici: chiavi per cifrare e decifrare uguali (DES, 3DES, AES)
- Stream
- Chiave pubblica: RSA, ElGamal,...

Primitive crittografiche:

- Funzioni hash
- Generatori di numeri pseudo-casuali

Protocolli:

- Firme digitali
- Scambio di chiavi
- Autenticazioni
- Scambio di segreti

Principio di Kerkhoffs

Il principio di Kerkhoffs dice che è inutile tener nascosto l'algoritmo di cifratura e quello di decifratura. La parte da tener nascosta è la più piccola possibile, la chiave.

La segretezza non dovrebbe risiedere nelle funzioni $E()$ e $D()$, ma piuttosto in una piccola quantità di informazione chiamata chiave

Gli algoritmi sono visibili a tutti quindi (se lo rompi pubblichi e diventi famoso).

Un **crittosistema** è una quintupla formata dai seguenti elementi:

- PT: spazio di tutti i testi in chiaro (plaintext)
- CT: spazio di tutti i testi cifrati (ciphertext)
- K: spazio delle chiavi (keyspace)
- Funzioni di cifratura ($E: PT \times K \rightarrow CT$)
- Funzioni di decifratura ($D: CT \times K \rightarrow PT$)

Come notazione si dice che $E_k(x)$ dove k è la chiave e x è il testo da cifrare.

Lo spazio delle chiavi non dovrebbe essere troppo piccolo, altrimenti la chiave può essere trovata attraverso un attacco di forza bruta.

Lezione 15

Le funzioni di cifratura e di decifratura che definiscono un crittosistema devono godere di diverse proprietà:

- $\forall m \forall k D_k(E_k(m)) = m$ In altre parole la funzione di cifratura e di decifratura sono una l'opposta dell'altra
- $\forall m \forall k$, la computazione di $c = E_k(m)$ e $D_k(c) = m$ dev'essere *facile* (per Alice e Bob dev'essere facile decifrare, cifrare).
- Dev'essere *estremamente difficile* trovare $c = E_k(m)$ senza conoscere k .

P.s.*facile* è un problema risolvibile da una MDTD in tempo polinomiale, *difficile* non è risolvibile in tempo polinomiale (non si conosce un algoritmo) (NP).

Cifrari storici

Cifrario di Cesare

Il primo cifrario storico conosciuto è quello di Cesare (scritto da Polibio forse). È un crittosistema simmetrico, quindi la chiave per cifrare/decifrare è la stessa; o meglio, la chiave per decifrare può essere ricavata *facilmente* a partire dalla chiave per cifrare (e viceversa).

Nel cifrario di Cesare $PT = CT = \Sigma$ dove $\Sigma = \{A, B, C, \dots, Z\}$, mentre $K = \{0, 1, 2, \dots, 25\}$. Ogni lettera è cifrata indipendentemente dalle altre.

Come funziona?

- **Cifratura:** data una $k \in K$, per cirfrare una lettera del plaintext σ si computa la funzione $E_k(\sigma)$ avanzando di k posizioni lungo l'alfabeto, partendo da σ e riprendendo da A quando si raggiunge la Z
- **Decifratura:** data una chiave $k \in K$ e una lettera c del ciphertext, si va indietro di k posizioni lungo l'alfabeto (il contrario del precedente).

Esempi:

- $E_3(\text{TRYAGAIN}) = \text{WUBDJDLQ}$
- $E_{25}(\text{IBM}) = \text{HAL}$

Esso è un crittosistema per sostituzione monoalfabetica: ogni lettera del testo in chiaro viene sostituita sempre con la stessa lettera nel testo cifrato (per una data k), ad esempio con $k = 2$ la a diventa sempre una c.

Crittoanalisi: lo spazio delle chiavi è troppo piccolo (solo 25 chiavi possibili)! Quindi un attacco di forza bruta è possibile.

Alcune proprietà del cifrario di Cesare:

- Commutatività: $E_3D_7E_6D_{11} = E_3E_6D_7D_{11} = E_{17} = D_9$

- Per tutte le $k \in \{1, 2, \dots, 25\}$ vale la seguente proprietà:
 - $D_k = E_{26-k}$
 - $E_k = D_{26-k}$
 - $D_k E_k = E_0 = D_0$

Cifrario di Hill

Questo crittosistema si basa su algebra lineare. Associa alle lettere da A a Z dei numeri da 0 a 25, tutte le operazioni aritmetiche sono quindi effettuate $mod 26$. Come funziona? Si sceglie un intero $d \geq 2$ (nell'esempio si utilizza proprio 2). La chiave di cifratura sarà una matrice quadrata M di ordine d , invertibile in \mathbb{Z}_{26} (ovvero deve esistere la matrice M^{-1} dove i conti vengono fatti in resto 26). Per controllare che una matrice sia invertibile in \mathbb{Z}_{26} si controlla che $\det(M) \neq 0$. La chiave di decifratura infatti è M^{-1} .

Diciamo che $PT = CT = \Sigma^d$ dove $\Sigma = \{0, 1, 2, \dots, 25\}$.

- **Cifratura:** il plaintext m è diviso in blocchi di d lettere ($m = P_1 P_2 \dots P_k$ con $|P_i| = d \forall i = 1, 2, \dots, l$). Se la lunghezza di m non è multipla di d , si aggiungono dei caratteri *dummy* (fittizie).

Ogni blocco successivamente è cifrato separatamente: $C_i = M \cdot P_i \forall i = 1, 2, \dots, l$. Il ciphertext è la concatenazione dei diversi blocchi C_1, C_2, \dots, C_l .

Esempio: $m = \text{HELP}$,

$$M = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

$$M^{-1} = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Abbiamo quindi:

$$P_1 = \begin{bmatrix} H \\ E \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \end{bmatrix} \quad P_2 = \begin{bmatrix} L \\ P \end{bmatrix} = \begin{bmatrix} 11 \\ 15 \end{bmatrix}$$

$$C_1 = M \cdot P_1 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 4 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} H \\ I \end{bmatrix}$$

$$C_2 = M \cdot P_2 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 11 \\ 15 \end{bmatrix} = \begin{bmatrix} 0 \\ 19 \end{bmatrix} = \begin{bmatrix} A \\ T \end{bmatrix}$$

Quindi, $c = C_1 C_2 = \text{HIAT}$

- **Rompere il sistema:** Mettiamo che Eve scopre che $d = 2$, ma non conosce M . A un certo punto vede che il testo cifrato $c = \text{HIAT}$ e dopo un po' scopre che quel messaggio voleva dire **HELP**.

A questo punto sa che $HI = [7 \ 8] = [H \ E] \cdot M = [7 \ 4] \cdot M$ e che $AT = [0 \ 19] = [L \ P] \cdot M = [11 \ 15] \cdot M$. Da qui:

$$M \cdot \begin{bmatrix} 7 & 11 \\ 4 & 15 \end{bmatrix} = \begin{bmatrix} 7 & 0 \\ 8 & 19 \end{bmatrix}$$

E da qui scopre che:

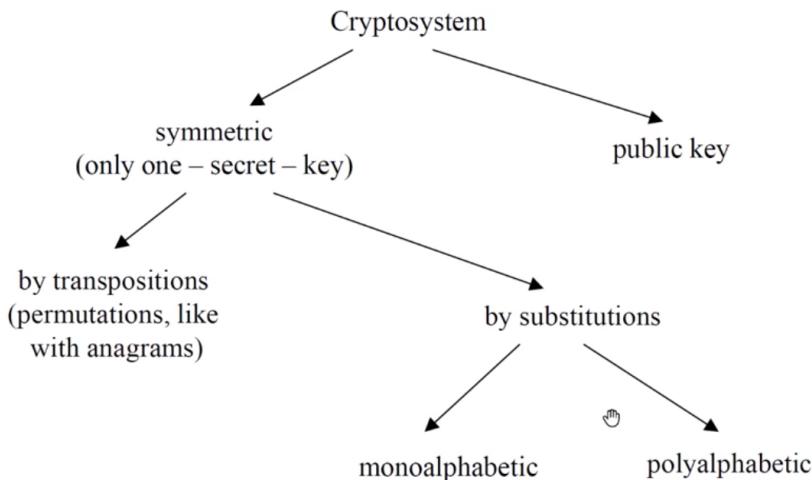
$$M = \begin{bmatrix} 7 & 0 \\ 8 & 19 \end{bmatrix} \cdot \begin{bmatrix} 7 & 11 \\ 4 & 15 \end{bmatrix}^{-1} = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

Quindi questo crittosistema non va molto bene in quanto si riesce a ricavare la chiave osservando un testo cifrato e un testo in chiaro. In particolare Eve è stata fortunata perché i due vettori sono linearmente indipendenti, quindi creano uno spazio vettoriale di dimensione 2.

I tipi di attacchi possibili per Eve (in ordine di difficoltà):

- **Cryptotext only:** Eve conosce solo il testo cifrato c e vuole trovare m (situazione peggiore per Eve, ma più frequente)
- **Known plaintext:** Eve conosce alcune coppie (c_i, m_i) . Un nuovo testo cifrato c arriva e vuole trovare la corrispondente m .
- **Chosen plaintext:** Eve conosce alcuni testi in chiaro a sua scelta (ad esempio può cifrare alcuni testi con una chiave pubblica), da qui può fare analisi statistiche sulle coppie (c_i, m_i) .
- **Chosen ciphertext:** Eve può decifrare ogni testo cifrato per un certo periodo di tempo (magari per qualche ora riesce ad utilizzare il sistema di Bob)

Tassonomia dei Crittosistemi



- Crittosistemi per sostituzione: ogni lettera di c è sostituita con una lettera di Σ
 - Monoalfabetici: è sostituita sempre con la stessa lettera
 - Polialfabetici: la sostituzione cambia durante il processo di cifratura
- Crittosistemi di trasposizioni: le chiavi sono:
 - Troppo facili da trovare oppure
 - Troppo difficili da ricordare

Possiamo usare un crittosistema polialfabetico da solo, oppure comporre trasposizione e sostituzione (product ciphers).

Per quanto un'informazione può rimanere segreta? Dipende da diversi fattori:

- Chi è l'avversario: conoscenza matematica, potenza computazionale, budget, quanto guadagna a rompere...
- Avanzamento della tecnologia
- Dimostrazione di ipotesi matematiche non dimostrate oggi (es. oggi non è dimostrato problema fattorizzazione)

Crittoanalisi dei crittosistemi monoalfabetici

Ogni linguaggio naturale ha una distribuzione di probabilità per le lettere che lo definisce, ad esempio in inglese:

High	Medium	Low
E 12.31%	L 4.03%	B 1.62%
T 9.59%	D 3.65%	G 1.61%
A 8.05%	C 3.20%	V 0.93%
O 7.94%	U 3.10%	K 0.52%
N 7.19%	P 2.29%	Q 0.20%
I 7.18%	F 2.28%	X 0.20%
S 6.59%	M 2.25%	J 0.10%
R 6.03%	W 2.03%	Z 0.09%
H 5.14%	Y 1.88%	

Una strategia per rompere un sistema monoalfabetico è contare la frequenza con cui compaiono le lettere, mettiamo ad esempio che in un testo cifrato la lettera P compaia 12 volte su 100: è molto probabile che essa sia una E o una T e così via (difficilmente, anzi impossibile sia una Z!). Questo ovviamente nell'ipotesi di conoscere la lingua che sto cercando di decifrare.

Quindi la crittoanalisi di crittosistemi monoalfabetici si basa sull'analisi delle frequenze delle lettere. Questo ragionamento si basa sul fatto che la sostituzione delle lettere non cambia le frequenze.

Per difendersi da un attacco di questo tipo cosa si fa?

- Lettere nulle: si aggiungono nel plaintext delle lettere a bassa frequenza, non fanno parte del messaggio
- Omofone: assumiamo di utilizzare $\Sigma = \{A, B, \dots, Z\}$ e $\Gamma = \{0, 1, 2, \dots, 99\}$ come alfabeti per PT e CT :
 - Siccome la lettera E occorre il 12% delle volte, associamo ad essa 12 simboli di Γ (chiamati gli omofoni di E)
 - Quando cifriamo E si sceglie casualmente uno di questi simboli (\rightarrow cifratura casuale)
 - La scelta può essere anche fatta "a mano" in modo da appiattire le frequenze

Crittoanalisi dei crittosistemi polialfabetici

Le lettere in questo caso sono sempre sostituite in maniera diversa, ma se consideriamo vettori di lettere (coppie, triple, n-uple...), allora esse sono sostituite sempre nello stesso modo.

Ad esempio nel crittosistema di Hill, la **H** in **HA** viene cifrata in modo diverso dalla **h** in **HE**, però queste due ultime coppie sono cifrate sempre nello stesso modo. Quindi da un certo punto di vista diventa un monoalfabetico.

Lezione 16

Crittosistema Playfair '800

Esso è un crittosistema polialfabetico che cifra a coppie. La chiave è formata da un quadrato di 5×5 lettere inglese (tranne la J). Sì, c'è solo una chiave. Il testo in chiaro m è spartito in blocchi di due lettere (questi blocchi non possono contenere doppie)

- **Cifratura:** ogni blocco è crittato separatamente:
 - Se due lettere sono sulla stessa riga allora ci si sposta ciclicamente verso la destra (ogni lettera diventa la sua adiacente a destra)
 - Se due lettere sono sulla stessa colonna allora ci si sposta verso il basso
 - Altrimenti esse individuano un rettangolo, quindi prendo gli altri due vertici del rettangolo
- **Decifratura:** si fa lo stesso procedimento però all'inverso

Per creare una chiave si parte da una frase:

COURSE ON CRYPTOGRAPHY

si rimuovono le lettere che occorrono più volte:

COURSENYPTGAH

poi completiamo inserendo le altre lettere:

COURSENYPTGAHBDFIKLMQVWXZ

È facile da ricordare e da ricostruire, però è facile da indovinare (la prima parte è una parola, le ultime sono le rimanenti dell'alfabeto). Dal punto di vista della crittoanalisi: ogni paio di lettere individua sempre lo stesso blocco (si possono fare analisi delle frequenti ecc.).

Crittosistema Vigenère '500

Funziona come il cifrario di Cesare, però la chiave cambia dopo aver cifrato ogni lettera. La chiave è composta da una parola, ripetuta più volte fino ad arrivare alla lunghezza del testo in chiaro m .

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

- **Cifratura:** ogni paio di lettere di m e k individua una riga e una colonna, si prende il carattere che è contenuto nella cella a cui si riferiscono. Esempio: m : PURPLE, k : CRYPTO, c : RLPEES. Ad esempio il primo carattere sarà quello individuato dalla riga che inizia per P e la colonna che inizia con C.
- **Decifratura:** le stesse operazioni all'inverso.
- **Crittoanalisi:** la i-esima lettera di m è cifrata come Cesare, la cui chiave è la i-esima lettera della chiave. Se la lunghezza della chiave è l , allora le lettere di m in posizione i , $i + l$, $i + 2l\dots$ sono tutte cifrate con la stessa lettera di k . Ad esempio per $l = 5$, scrivo i caratteri del testo cifrato in righe lunghe l , quindi avrò (in termini di posizioni):

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

A questo punto ogni colonna è cifrata con la stessa chiave di Cesare, per cui si può effettuare un'analisi delle frequenze. Questo rompe il sistema

anche se il numero delle chiavi è $26^5 = 11.881.376$. Come si trova però la lunghezza della chiave? Cerchiamo delle occorrenze di alcune sequenze, PUXUL + 15 lettere + PUXUL. La lunghezza della chiave potrebbe essere un divisore di 20.

Macchina Enigma

Ci sono tre rotori (versione più semplice) ...
troppo lungo da spiegare

DES

Negli anni 60-70 viene creato un crittosistema standard (casini fra banche ognuno aveva il suo ecc) da IBM, a partire dal vecchio Lucifer. Nasce DES (Data Encryption Standard) che è un crittosistema simmetrico. (non è da sapere il funzionamento). La lunghezza della chiave è di 56 bit, quindi ha 2^{56} possibili valori, il che lo rende oggi suscettibile da un attacco a forza bruta. Nella chiave vengono aggiungi bit di parità in posizioni 8, 16, 24, 32, ..., 64. (non ha senso, si mischia il livello crittografia e codifica). L'algoritmo ha due fasi:

1. Si generano le **chiavi derivate**: i bit della chiave vengono disposti in questo modo: non è comodo, ma DES nasce per avere un'implementazione

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

di tipo hardware...

A questo punto divido in due blocchi: le prime quattro righe diventano il blocco C_0 , le restanti quattro diventano D_0 . A questo punto posso ottenere C_1 e D_1 ruotando verso sinistra i bit di C_0 . In generale posso ottenere C_n e D_n a partire da C_{n-1} e da D_{n-1} . Poi si selezionano alcune permutazioni e si generano le chiavi derivati K_n . Poi si divide il messaggio

in chiaro in blocchi di 64 bit e comincia a fare scambi, xor, permutazioni ecc

Proprietà DES:

- Molto veloce (specialmente con hardware dedicato, le operazioni sono solo XOR, permutazioni, rotazioni, utilizzo di matrici di dimensioni fissate (lookup tables))
- La crittoanalisi di DES porta alla risoluzioni di equazioni non lineari (problemi simili al SAT, quindi NP-completi).
- Le S-boxes di DES (quelle della NSA e non quelle di Lucifer) resistono ad attacchi di crittoanalisi lineare e crittoanalisi differenziale.
- Quindi dal punto di vista matematico è ancora considerato sicuro, non più dal punto di vista informatico
- La chiave di 56 bit è troppo corta
- Esiste una macchina dedicata costruita con hardware di consumo (La DES Cracking Machine) che esplora l'intero spazio delle 2^{56} chiavi (1800 processori, trova la chiave segreta in due giorni).

A metà degli anni 90 una competizione internazionale fu organizzata per sostituire DES e creare un nuovo standard. Nel frattempo la comunità di crittografi consiglia di utilizzare il sistema 3DES:

1. Si cifra w usando k_1 , ottenendo c_1 ,
2. Si decifra c_1 usando k_2 , ottenendo c_2 ,
3. Si cifra c_2 usando k_1 , ottenendo c

Lezione 17

AES

La competizione degli anni '90 viene vinta da Rijndael e il suo AES, esso diventa standard dal 2002. Cifra blocchi di 128 bit per volta e le chiavi possono essere di diverse lunghezze (AES-128, AES-192 e AES-256). È più efficiente di DES e la sua implementazione software è più semplice. Per semplicità vediamo solo AES-128, ma gli altri cambiano solo su pochi dettagli.

- **Cifratura:** l'algoritmo opera su una matrice di stato 4x4 di 16 bytes (128 bits). Inizialmente la matrice contiene il blocco del messaggio in chiaro (riempita per colonne).

Vengono generate le chiavi derivate e si effettua uno XOR fra i bit della matrice e quelli della **Roundkey**. Per nove volte (ogni chiave derivata) si effettuano queste operazioni:

1. **SubBytes** (sostituzione dei byte utilizzando una S-box)

Essa è una operazione invertibile ma non lineare. Si fissa $b = (b_7, b_6, \dots, b_0)$ e si definiscono: Poi, se $b \neq 0$ allora $b = b^{-1}$ e si ritorna $A \cdot b + c$.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

La non linearità sta nel calcolo dell'inverso di b (resistenza ad attacchi ad approssimazione). Vedo i termini di b come i coefficienti di un'equazione di settimo grado ($b_7x^7 + b_6x^6 + \dots + b_1x + b_0$), poi faccio la divisione con resto con il polinomio irriducibile $x^8 + x^4 + x^3 + x + 1$ nel campo $GF(2^8)$ ovvero con resto 0 o 1. In questo campo ci sono vettori di 8 elementi, la somma è lo xor, il prodotto poi si divide per lo stesso vettore.

2. **Shiftrows** (rotazione delle righe)

La prima riga rimane alterata, la seconda la terza e la quarta viene ruotata di 1, 2, 3 posizioni rispettivamente

3. **MixColumns** (operazioni sulle colonne)

Moltiplica ogni colonna per una matrice fissata e che diventa la nuova colonna.

4. AddRoundKey (come prima, si effettua uno xor)

Come si generano le chiavi derivate? Spieghiamo per AES-128. Bisogna ottenere 11 chiavi, ognuna di 16 bytes. L'algoritmo lavora su parole di 4 bytes, quindi dobbiamo ottenere 44 parole che inseriamo in un array $w[0..43]$. Si definiscono 10 costanti $Rcon[1], Rcon[2], \dots, Rcon[10]$. Successivamente si segue questo pseudocodice:

```

for  $i = 0$  to  $3$ 
     $w[i] = (key[4i], key[4i+1], key[4i+2], key[4i+3])$ 

for  $i = 4$  to  $43$ 
     $temp = w[i-1]$ 
    if  $i \equiv 0 \pmod{4}$  then
         $temp = \text{SubWord}(\text{RotWord}(temp)) \oplus Rcon[i/4]$ 
    endif 
     $w[i] = w[i-4] \oplus temp$ 

return ( $w[0], \dots, w[43]$ )

```

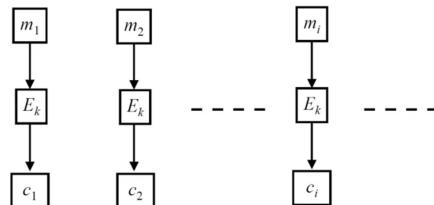
Crittosistemi simmetrici

Abbiamo una sequenza di testi di chiaro m_1, m_2, \dots e una chiave iniziale k , vogliamo avere una sequenza di testi cifrati c_1, c_2, \dots .

Modi di operazione:

- ECB (Electronic CodeBook)

Il più semplice, ma anche il meno sicuro: ho m_1 , lo cifro usando

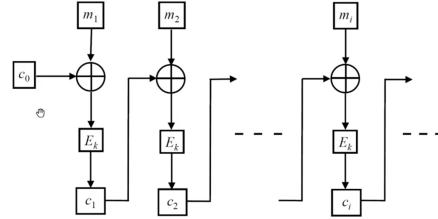


l'algoritmo E_k e ottengo c_1 (e così via per tutte le m). Quando Eve comincia ad ascoltare (mettiamo che si perda alcuni pezzi), non ha vita difficile visto che ogni operazione di cifratura è indipendente dalla precedente. Oppure esempio pinguino Linux, cifro blocchi uguali in

modo uguale, il risultato sarà strano ma avrà la forma del pinguino (il pattern non viene rotto). Ha però vantaggi: la modifica di un blocco di codice a causa del rumore, non rovina i successivi (adatto quindi a canali rumorosi).

- CBC (Cipher Block Chaining)

Effettua una concatenazione con quello che è successo in precedenza.



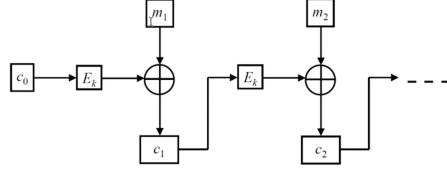
In particolare il messaggio cifrato $c_i = E_k(c_{i-1} \oplus m_i)$. È il modo di operazione più usato (se cifri l'immagine del pinguino viene un casino). Il problema è che se il rumore rovina un blocco allora tutti i successivi verranno sbagliati. Quindi il canale di comunicazione dev'essere affidabile.

- OFB (Output Feedback)

Viene prodotta una sequenza di chiavi derivate, che vengono utilizzate per effettuare la cifratura. In pratica cifro m_1 facendo lo xor con z_1 e così via. L'idea è fare lo xor con dei bit casuali per ottenere un risultato casuale. Parto da z_0 che trovo in maniera casuale. Se E_k è fatto bene lo posso usare come generatore di numeri casuali. Questo modo è molto veloce in quanto fa solo lo xor con le chiavi derivate (es guardo un video in HD). È affidabile anche in canali rumorosi in quanto l'aver perso un blocco m_i non pregiudica quello successivo. Però è vulnerabile ad un attacco che modifica lo stream. (es. se si parla di cifre, prezzi, posso invertire un numero).

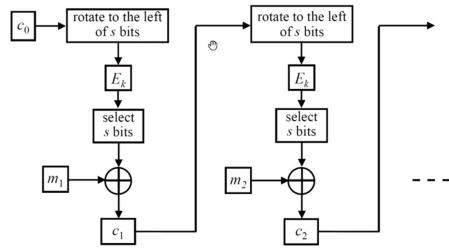
- CFB (Cipher FeedBack)

I blocchi del testo vengono cifrati facendo uno xor bit a bit con l'output dell'algoritmo di cifratura applicato al blocco precedente. Il canale dev'essere affidabile. Di pro c'è che se l'ultimo blocco arriva corretto allora anche i precedenti lo sono stati.



- s-bit CFB

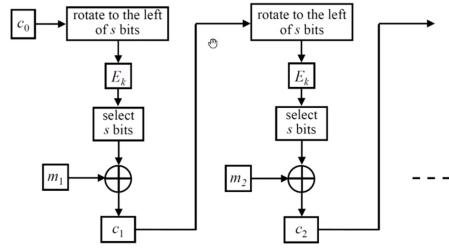
Funziona come CFB, però si applicano delle trasformazioni in più



(rotazioni e selezioni). Però più lento.

- CTR (Counter Mode)

Usato dalle ATM (Asynchronous Transfer Mode) (es. Fastweb), ma



anche da IPSec (VPN). Si generano chiavi derivate diverse sfruttando i diversi valori dei contatori. Molto efficiente

Lezione 18

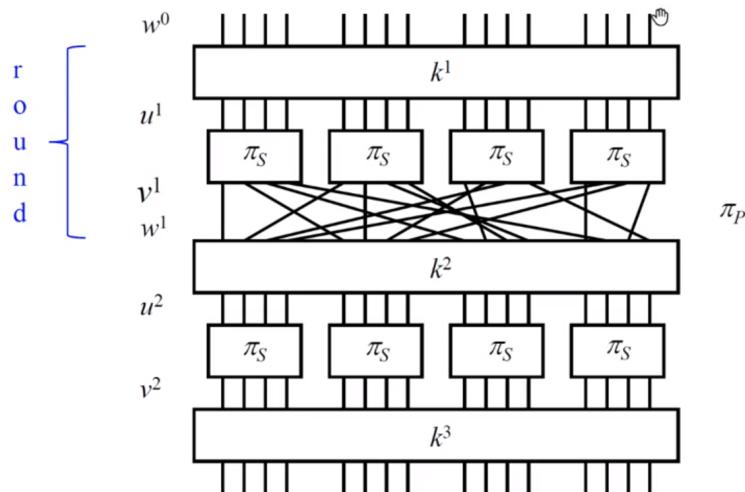
Substitution-Permutation Network (SPN)

Confusione e diffusione: principi introdotti da Shannon, l'obiettivo è quello di rendere l'analisi statistica del testo cifrato c molto difficile. L'idea è quella che le proprietà statistiche di c devono essere indipendenti (al limite della fattibilità) dal valore di k . Definisce quindi il concetto di **diffusione** (spalmare), secondo cui la struttura statistica di c è distribuita lungo tutta la lunghezza della chiave, ogni simbolo di m ha effetto su *più simboli* di c , e i simboli di c sono determinati da più simboli di m . Per ottenere questo risultato si effettua una permutazione, seguita dall'applicazione di una funzione non lineare.

La **confusione**(ingarbugliare): la modifica di m e/o di k deve avere un output "imprevedibile", che non hanno relazioni statistiche fra di loro. Per ottenere questo effetto si usa una complessa sostituzione.

Effetto valanga: un piccolo cambiamento su m o k (anche un singolo bit), produce una modifica significativa (anche metà dei bit) di c .

Per mettere tutto insieme si usa una rete di permutazione e sostituzione (SPN, Substitution-Permutation Network).

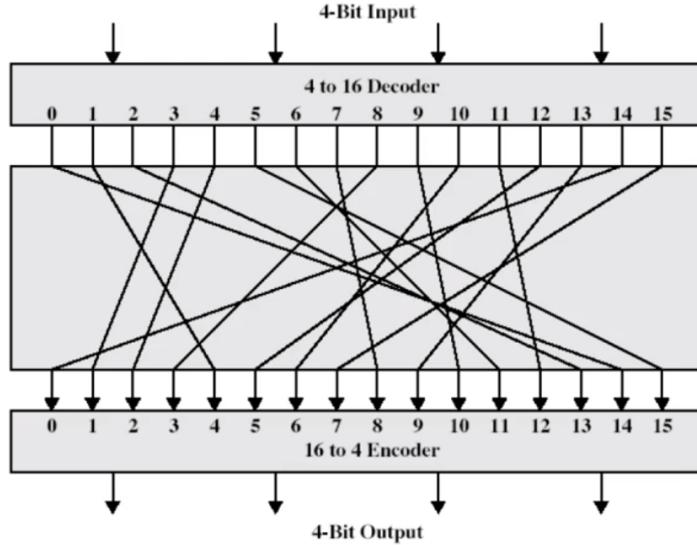


...

Struttura di Feistel

Le operazioni di cifratura devono essere invertibili. Dato un blocco di n bit, la cifratura può essere espressa come la composizione di:

- Un decoder da n a 2^n
- Una sostituzione monoalfabetica di 2^n simboli



- Un encoder da 2^n a n

Il problema è che dovrei salvare le 2^n sostituzioni, che diventano un casino. L'idea è quella di implementare queste operazioni usando dei moduli piccoli e facili da implementare (product ciphers)

La struttura di Feistel è un caso particolare di SPN:

- input: testo in chiaro m e una chiave k
- dalla chiave k si generano le chiavi derivate k^1, k^2, \dots, k^n
- m è diviso in due metà: L_0 e R_0
- L_0, R_0 sono trasformate in $L_1, R_1, L_2, R_2, \dots, L_n, R_n$ in n round

In ogni round:

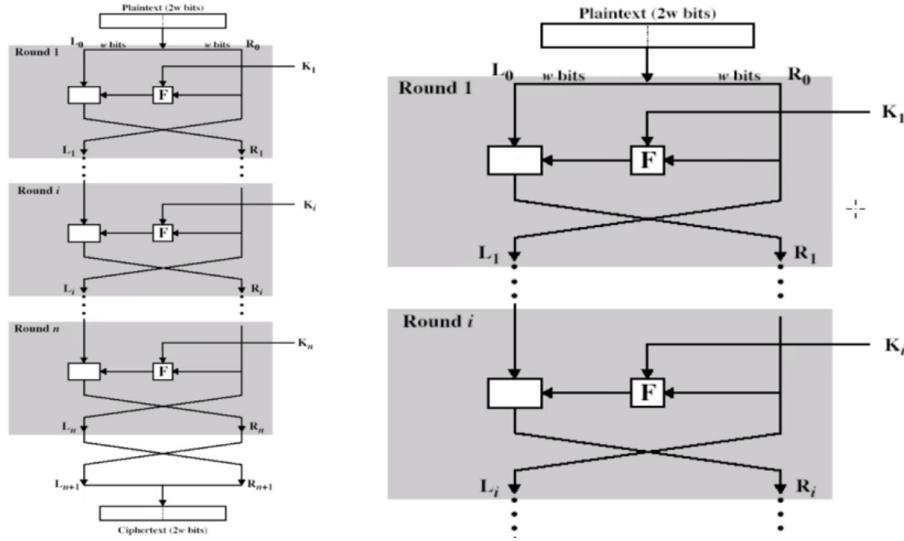
- input: L_{i-1}, R_{i-1}, k^i , output: L_i, R_i
- sostituzione: una round function F è applicata a R_{i-1} e k^i , poi si effettua lo xor fra il risultato e L_{i-1}
- permutazione: L_{i-1} e R_{i-1} sono scambiati: $L_i = R_i$, $R_i = L_{i-1} \oplus F(R_{i-1}, k^i)$.

Dopo tutti i round, si elimina l'ultima permutazione (che non aggiunge sicurezza ma velocizza la decifratura).

Più le chiavi e i blocchi sono grandi, più il sistema è sicuro (ma lento).

Più round ci sono (effetto valanga) più il sistema è sicuro (ma lento).

Più è complicata F più il sistema è sicuro (ma lento).



Più la generazione delle chiavi derivate è complicata, più il sistema è sicuro (ma lento).

Una qualità della Struttura di Feistel è che è completamente simmetrico, quindi la cifratura e la decifratura sono processi uguali (cambia solo l'ordine di utilizzo delle chiavi).

Crittoanalisi lineare

(faccio veloce, non chiede orale)
Coppersmith nel '94 dice che

Nessun bit di output di una S-box dev'essere troppo vicino a una funzione lineare dei bit di input. In particolare, se selezioniamo un qualsiasi bit di output in qualunque sottoinsieme dei sei bit di input, la porzione di input per cui questo bit di output è uguale allo xor di questi bit di input non dovrebbe essere vicina a 0 o a 1, ma piuttosto vicina a $\frac{1}{2}$

Timing attack

In base al tempo con cui vengono effettuate certe operazioni, traggo delle conclusioni. Sul DES si può trovare il numero di 1 nella chiave.

Design delle S-box

Le S-box del DES sono fatte a mano, facendo dei tentativi e cercando di capire se i tentativi funzionano o no (molto lungo, nessuno fa più così). Oppure le faccio a caso e scelgo solo quelle che soddisfano dei test statistici. Oppure le S-box di AES sono definite sfruttando criteri matematici, in particolare la wide-trail

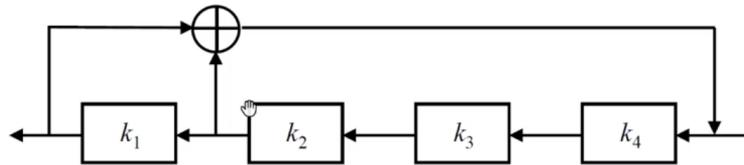
strategy: posso provare a sostituire con funzioni lineari, ma queste coinvolgerebbero così tanti bit che renderebbero infattibile l'attacco di crittoanalisi lineare.

Cifrario a flusso

Un cifrario che cifra un bit o un byte alla volta, cambiando continuamente la chiave, usando un keystream. La chiave segreta k è usata come *seed* per un PRNG (Pseudo-Random Number Generator) che produce le chiavi derivate, ognuna di un bit o byte. Successivamente ogni bit o byte m è calcolato facendo lo xor con l'output del generatore pseudo-casuale.

Il cifrario a flusso è sicuro se il generatore di numeri pseudo-casuali è buono (xor con bit casuali produce bit casuali). Essi sono anche più veloci (usano meno codice), però non possiamo riutilizzare le chiavi (altrimenti creo delle relazioni lineari fra la porzione di chiave utilizzata e i testi in chiaro-cifrati, quindi il flusso di chiavi dev'essere aperiodico (al limite del fattibile)). La sicurezza quindi è basata fortemente sull'algoritmo con cui genero un keystream robusto. Il seme (chiave) dev'essere abbastanza lungo (almeno 128 bits) per evitare attacchi di forza bruta.

Per costruire le chiavi derivate uso LFSR (Linear Feedback Shift Registers), registro di scorrimento con feedback lineare.



Ad ogni colpo di clock il contenuto di k_4 va a k_3 , quello di k_3 a k_2 e così via bla bla

Produce un output però che è generato usando equazioni lineari. Se Eve conoscesse $2m$ chiavi derivate, chiamate z_1, z_2, \dots, z_{2m} allora può scrivere un sistema di equazioni per ricavare la chiave. Quindi non usarli mai da soli.

RC4

Cifrario a flusso più famoso, usato in SSL/TLS che sono gli standard per la comunicazione attraverso browser web, servizi di e-commerce ecc.

Come funziona?

- Cifra un byte per volta
- Usa una permutazione casuale
- Usa un vettore di stato S di 256 bit

- Usa una chiave k che va da 1 a 256 byte per inizializzare S ; dopodichè k non è più utilizzata
- Il periodo del keystream arriva a 10^{100}
- Richiede solo 8-16 operazioni macchina

Vediamo ora passo passo:

1. Inizializzazione di S :

Si mettono in S tutti i numeri da 0 a 255, uso un vettore temporaneo T in cui salvo la chiave k tante volte quanto necessario per riempirlo (se la chiave è 256 byte allora appunto).

```
for  $i = 0$  to 255 do
   $S[i] = i$ 
   $T[i] = k[i \bmod \text{keylen}]$ 
```

Successivamente ogni elemento di S viene scambiato con un altro elemento, la cui posizione è determinata da S e T

```
 $j = 0$ 
for  $i = 0$  to 255 do
   $j = (j + S[i] + T[i]) \bmod 256$ 
  swap( $S[i], S[j]$ )
```

2. Generazione del keystream:

Si parte da $S[0]$ e si arriva a $S[255]$, poi di nuovo da $S[0]$ e così via. Ad ogni interazione l'elemento $S[i]$ è cambiato con un altro elemento $S[j]$, dove il valore j dipende da $S[i]$. La somma $S[i] + S[j] \pmod{256}$ da la posizione dell'elemento z

```
 $j = 0$ 
 $i = 0$ 
while true do
   $i = (i + 1) \bmod 256$ 
   $j = (j + S[i]) \bmod 256$ 
  swap( $S[i], S[j]$ )
   $t = (S[i] + S[j]) \bmod 256$ 
   $z = S[t]$ 
```

3. Cifratura e decifratura:

La cifratura è lo xor fra il testo e z : $c_i = m_i \cdot z$, la decifratura il contrario: $m_i = c_i \cdot z$. Non si conoscono attacchi che rompano l'algoritmo (a parte l'implementazione WEP)

La sicurezza di Shannon

Shannon si chiede, esiste un crittosistema sicuro incondizionatamente? Anche AES-256, se ha un computer della madonna ce la fa dopo un po'. Basta avere una coppia testo + testo cifrato e provare tutte le 2^{256} .

Chiamiamo X una variabile casuale che assume valori in PT (spazio dei testi in chiaro), Y una variabile casuale che assume valori in CT (spazio dei testi cifrati). Alice sceglie $x \in PT$, lo cifra, ottiene $y \in CT$ e lo manda a Bob. Per semplicità mettiamo che Alice scelga x con una probabilità normale uniforme. Eve conosce solo $P[X = x]$. Se però intercetta il messaggio cifrato y , conosce $P[X = x]$ e $P[X = x|Y = y]$ (probabilità all'indietro).

Per Shannon il crittosistema è sicuro se Eve non impara nulla dopo aver conosciuto la probabilità all'indietro:

$$\forall x \in X, y \in Y$$

$$Prob[X = x] = Prob[X = x|Y = y]$$

che equivale a:

$$H(X) = H(X|Y)$$

che sarebbe $I(X; Y) = 0$ (la mutua informazione del sistema vale 0), quindi X e Y sono indipendenti, quindi Eve non ottiene alcuna informazione da y .

Questo vuol dire che il canale sarebbe completamente rumoroso. Un crittosistema di questo tipo esiste ed è il **One Time Pad**. Esso ha una chiave lunga tanto quanto il testo e dev'essere utilizzata una sola volta. Si cifra e decifra facendo lo xor fra la chiave e il testo. Una proprietà di OTP è $\forall y, \forall k, \exists x : E_k(x) = y$, in altre parole la y può essere generata da ogni x utilizzando una chiave adatta. Quindi la y non ci dà alcun informazione. Basta che la chiave sia di un bit più corta per far sì che non sia più incondizionatamente sicuro, visto che $Prob[X = x|Y = y] = \frac{1}{2^{n-1}} \neq Prob[X = x]$.

Ovviamente ha dei contro:

- Bisogna produrre grandi quantità di chiavi casuali (ogni chiave poi sarà buttata)
- Bisogna distribuire e proteggere le chiavi
- Difficile da gestire

Lezione 19

Generatori di numeri pseudo-casuali

Tanti algoritmi crittografici hanno bisogno di tanti bit casuali, in natura si possono trovare ma recuperarli è molto dispendioso. Utilizziamo quindi dei generatori di numeri pseudo-casuali. Essi funzionano così: dato un seme (seed) casuale, un PRNG (Pseudo-Random Number Generator) produce una sequenza di bit che è indistinguibile da una vera sequenza di bit casuali. Indistinguibile significa che nessun algoritmo (che può essere eseguito in tempo polinomiale) può decidere se la sequenza è casuale o meno.

Un generatore di numeri casuali è un algoritmo che, dati k bit in input di seme casuale, ritorna $l(k) > k$ bit, tale per cui:

- Per tutti gli algoritmi $D \in P$ (chiamati *distinguisher*, quelli che hanno capiscono se è calcolata o casuale)
- Per tutti i polinomi p
- Per tutti gli interi k sufficientemente grandi

vale che:

$$|Prob[x \leftarrow \{0, 1\}^k; r \leftarrow G(x) : D(r) = 1] - Prob[r \leftarrow \{0, 1\}^{l(k)} : D(r) = 1]| < \frac{1}{p(k)}$$

ovvero prende x a caso di k bit, dopodichè lo dà in input al generatore di numeri pseudo casuali G , e da il risultato in r , allora mi chiedo quanto è la probabilità che dato r al *distinguisher* esso se ne accorga. L'altra probabilità invece prende r come $l(k)$ numeri davvero casuali, e faccio la stessa cosa. Se la differenza in valore assoluto (la distanza) fra le due quantità diventa sempre più piccola al crescere del seme k , quindi tende a 0 più velocemente di $\frac{1}{p(k)}$ allora esso è definito un generatore di numeri casuali.

Linear congruential generator

Si fissano due costanti a e b (più un modulo m), tali per cui $0 \leq a, b < m$. Dato un seme intero s per cui $0 \leq s < m$, allora il sistema:

$$\begin{cases} x_0 = s \\ x_i = ax_{i-1} + b \pmod{m} \quad \forall i \geq 1 \end{cases}$$

è chiamato un generatore congruenziale lineare. La scelta di a , b e m è critica per ottenere sequenze difficili da prevedere.

Non è un buon generatore in quanto utilizza operazioni lineari. Se Eve riesce ad ottenere quattro valori prodotti dal PRNG, diciamo x_0, x_1, x_2 e x_3 , allora può risolvere il sistema di equazioni per trovare i valori di a, b, m .

$$\begin{cases} x_1 \equiv ax_0 + b \pmod{m} \\ x_2 \equiv ax_1 + b \pmod{m} \\ x_3 \equiv ax_2 + b \pmod{m} \end{cases}$$

Se abbiamo un PRNG $H : \{0,1\}^k \rightarrow \{0,1\}^{k+1}$, allora possiamo produrre un qualsiasi altro PRNG $H : \{0,1\}^k \rightarrow \{0,1\}^{l(k)}$ in questo modo:

$$\begin{aligned} & G(x_0) \\ & x_1 \sigma_1 = H(x_0) \\ & x_2 \sigma_2 = H(x_1) \\ & \dots \\ & x_{l(k)} \sigma_{l(k)} = H(x_{l(k)-1}) \\ & \text{output}(\sigma_1, \sigma_2, \dots, \sigma_{l(k)}) \end{aligned}$$

Ovviamente $x_i \in \{0,1\}^k$, $\sigma_i \in \{0,1\}$ e $\forall i \in \{1, 2, \dots, l(k)\}$.

Ora ci rimane il problema di trovare la funzione H , che viene chiamata *one-way*, ovvero una funzione difficile da invertire. La parte dell'input che resiste all'attacco si chiama **hard-core** bit. Questi bit devono essere in numero polinomiale (quindi anche lineare) rispetto alla dimensione dell'input: n sono già tutti, quindi $\frac{n}{4}$ ci sta.

L'hard-core bit può essere visto come l'output di un predicato hard-core. Presa una permutazione one-way ($f : \{0,1\}^n \rightarrow \{0,1\}^n$), e $B : \{0,1\}^n \rightarrow \{0,1\}$ un predicato B che può essere calcolato in tempo polinomiale. B è un predicato hard-core per f se per ogni algoritmo A e per ogni polinomio P esiste un $n_{A,p}$ tale per cui:

$$\forall n \geq n_{A,p} \quad \text{Prob}[x \leftarrow \{0,1\}^n; b \leftarrow A(f(x)) : b = B(x)] \leq \frac{1}{2} + \frac{1}{p(n)}$$

Se la probabilità di indovinare l'hard-core bit non si discosta di molto da $\frac{1}{2}$ allora l'attaccante non sta capendo niente.

Usando f e B possiamo costruire un PRNG $H : \{0,1\}^k \rightarrow \{0,1\}^{k+1}$ come segue:

$$H(X) = f(x) || B(x)$$

Come facciamo a creare dei predicatori hard-core? Se g è un generatore di Z_p^* (classi di resto da 1 a p) esso è un gruppo ciclico, quindi esiste un generatore g . Definisco la funzione di esponenziazione modulare: $y = g^z \pmod{p}$; essa è una funzione one-way, dati y, g, p allora z è difficile da trovare. Si è dimostrato che il bit più significativo di z è un hard-core bit.

Quindi possiamo costruire un PRNG basato sull'esponenziazione modulare in questo modo:

$$H(z) = g^z \parallel \text{msb}(z)$$

(msb = Most Significative Bit)

Così allungo di 1 sempre in maniera imprevedibile.

Un altro predicato hard-core: dati due vettori $x = (x_1, x_2, \dots, x_n)$ e $y = (y_1, y_2, \dots, y_n)$ di n elementi. Il prodotto modulo 2 di x e y :

$$\langle x, y \rangle = \sum_{i=1}^n x_i y_i \bmod 2 = \oplus_{i=1}^n (x_i \wedge y_i)$$

Allora il **teorema di Goldreich e Levin** dice che: se si prende una permutazione one-way f e si costruisce una funzione $g(x, y) = f(x) \parallel y$, con $|x| = |y| = n$, allora il prodotto interno modulo due diventa un predicato hard-core:

$$B(x, y) = \langle x, y \rangle$$

Quindi data una permutazione one-way possiamo costruire un PRNG!

Ma ci chiediamo, come facciamo a dimostrare se un algoritmo è un buon generatore di PRNG? Si fanno dei test sulle sequenze prodotte dall'algoritmo e si controlla se ci sono delle relazioni statistiche fra gli elementi della sequenza (li plotto, vedo se gli 0 e 1 sono 50%). Dal punto di vista teorico c'è un test universale che formalizza la nozione di *impredicibilità*. Data una funzione $G : \{0, 1\}^n \rightarrow \{0, 1\}^{l(k)}$. G è impredicibile se e solo se $\forall i \in \{1, 2, \dots, l(k)\}, \forall$ polinomi p , e \forall algoritmi $A, \exists k_{A,p}$ tale che $\forall k \geq k_{A,p}$:

$$\text{Prob}[x \leftarrow \{0, 1\}^k; (\sigma_1, \dots, \sigma_{l(k)} = G(x); \sigma_i^{\text{new}} \leftarrow A(\sigma_1, \dots, \sigma_{i-1}) : \sigma_i^{\text{new}} = \sigma_i] \leq \frac{1}{2} + \frac{1}{p(k)}$$

In altre parole do i primi $i - 1$ elementi all'algoritmo A dell'attaccante e vedo se riesce a prevedere l'ultimo bit σ_i^{new} . Se questa probabilità tende a $\frac{1}{2}$ allora l'output di G è impredicibile. Questa è teoria (asintotica), ma come creiamo bit casuali in pratica?

ANSI X9.17

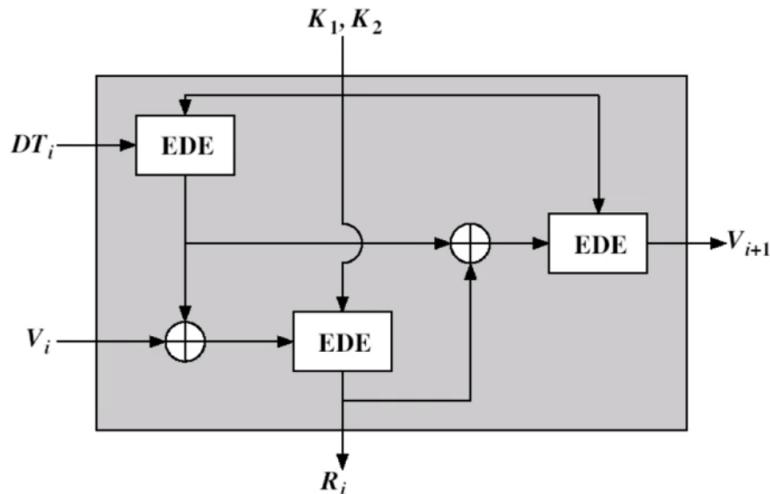
Esso è uno dei migliori generatori di numeri pseudo-casuali. Utilizza 3-DES.

- **Input:**

Un seme di 64 bit e una rappresentazione 64 bit della data e ora corrente. Due chiavi K_1, K_2 ognuna di 64 bit, esse verranno usate nei tre moduli 3DES in modalità EDE (Encryption, Decryption, Encryption).

- **Output:**

Due valori: R_i è un blocco di 64bit pseudocasuale, V_{i+1} : il nuovo valore del seme.



La crittoanalisi è molto complicata.
BBS balza

Crittosistemi a chiave pubblica

Il problema dei sistemi simmetrici è che Alice e Bob devono conoscere la stessa chiave k . Serve quindi un canale privato (quindi sicuro) per comunicarsi la chiave, e questo non è sempre possibile. La crittografia a chiave pubblica risolve questo problema: non c'è bisogno di un canale privato.

L'idea è che ogni utente crea attraverso un algoritmo una coppia di chiavi, la chiave di cifratura è pubblica, la chiave di decifratura rimane segreta. Quindi tutti possono cifrare un messaggio, ma solo io che conosco la chiave segreta posso decifrarlo.

Vediamo due soluzioni:
Bob ha due algoritmi:

- E_B : cifratura

- D_B : decifratura

tali per cui D_B decifra i messaggi cifrati con E_B . Se Alice vuole mandare un messaggio m a Bob, gli chiede E_B e calcola $E_B(m)$. Successivamente lo invia e Bob calcola $D_B(E_B(m)) = m$.

Se Eve intercetta $E_B(m)$ dovrebbe risolvere un problema che abbiamo assunto essere infattibile. Il problema è che in una grande rete di utenti Alice deve chiedere a tutti i loro algoritmi di cifratura.

Seconda soluzione:

Alice e Bob scelgono un algoritmo di cifratura e un algoritmo di decifratura. I due algoritmi di decifratura/cifratura commutano, quindi possono essere scambiati.

1. Alice calcola $E_A(m)$ e lo manda a Bob
2. Bob lo cifra con E_B e lo rimanda ad Alice
3. Alice riceve $E_B(E_A(m))$, applica D_A e manda a Bob
4. Bob applica D_B e ottiene $D_B(D_A(E_B(E_A(m)))) = D_A(E_A(m)) = m$

Lo svantaggio è che i messaggi vanno avanti e indietro due volte, però ognuno tiene il suo algoritmo di cifratura e può cambiarlo quando vuole.

Il problema è che il canale non è autenticato: quando Alice riceve la chiave pubblica di Bob, come fa ad essere sicura che non abbia ricevuto invece quelle di Eve?

Definizione formale di one-way function: f può essere computata in tempo polinomiale da una MDT, ed esistono due polinomi $p()$ e $q()$ tali per cui $p(|x|) \leq |f(x)| \leq q(|x|)$ (quindi l'output di f non dev'essere troppo corto).

Per ogni algoritmo A dell'attaccante che può essere eseguito in tempo polinomiale su una macchina probabilistica TM e per ogni polinomio $p()$ esiste un numero $n_{A,p}$ tale per cui:

$$\forall n \geq n_{A,p} \quad \text{Prob}[x \leftarrow \{0,1\}^n; x' \leftarrow A(f(x)) : f(x) = f(x')] \leq \frac{1}{p(n)}$$

Il problema è che se $P = NP$ allora le funzioni one-way non esistono! Se però si dimostra che una one-way function esiste allora $P \neq NP$

Lezione 20

Esponenziazione modulare

Riprendiamo con un esempio: il gruppo moltiplicativo sarà $\mathbb{Z}_5^* = 1, 2, 3, 4$, che è contenuto in $\mathbb{Z}_5 = 0, 1, 2, 3, 4$.

2 è un generatore di \mathbb{Z}_5^* , infatti:

$$\mathbb{Z}_5^* = \{2^0, 2^1, 2^2, 2^3\} = \{1, 2, 4, 3\}$$

Quindi possiamo definire la seguente funzione di esponenziazione modulare:

$$f(z) = 2^z \bmod 5$$

Nessuno è riuscito a trovare un algoritmo in P per trovare z a partire da $f(z)$. Dato x , però, posso calcolare $g^x \bmod p$ in tempo polinomiale (rispetto all'input). Ha senso considerare l'esponente x fra 0 e $p - 2$ (minore di 0 va all'inverso, maggiori di $p - 2$, riconsidero gli stessi elementi visto che c'è modulo). La dimensione dell'input è il numero di bit che servono per rappresentare gli elementi di \mathbb{Z}_p^* , quindi circa $n = \log_2 p$.

Possiamo quindi scrivere:

$$x = \sum_{j=0}^{n-1} x_j 2^j$$

Dove $(x_{n-1}, \dots, x_1, x_0)$ è la rappresentazione binaria di x .

Uno pseudocodice potrebbe essere:

```
result = 1
while x > 0 do
    result = (result * g) mod p
    x = x - 1
return result
```

Però questo algoritmo chiede un numero esponenziale di iterazioni in quanto $x \approx 2^n$.

Però vale che:

$$g^x = g^{\sum_{j=0}^{n-1} x_j 2^j} = \prod_{j=0}^{n-1} g^{x_j 2^j} = \prod_{j=0}^{n-1} (g^{2^j})^{x_j} \bmod p$$

Osservando l'ultima parte della formula posso calcolarmi tutti i valori di g^{2^j} per $j \in \{0, 1, \dots, n-1\}$ e moltiplico fra loro solo quello per cui vale $x_j = 1$. Questo può essere fatto in maniera polinomiale attraverso il metodo *square and multiply*.

```
result = 1
for j = n - 1 downto 0 do
```

```

result = (result * result) mod p
if  $x_j = 1$  then
    result = result * g mod p
return result

```

L'inverso dell'esponenziazione modulare è chiamato *logaritmo discreto*. Il problema è così definito:

- Numero primo p
- Generatore di \mathbb{Z}_p^* chiamato g
- $y \in \mathbb{Z}_p^*$

Si computa una $x \in \{0, 1, \dots, p-2\}$ tale per cui $g^x \equiv y \pmod{p}$.

La computazione del logaritmo discreto è un problema intrattabile in tempo polinomiale. Per avere un'idea della difficoltà del problema, prova a calcolare a mano i logaritmi base 3 dentro a \mathbb{Z}_{113}^* .

Abbiamo quindi trovato una scatola dove nascondere un valore x che non può più essere aperta; in altre parole abbiamo trovato una one-way function. Manca però una **trapdoor** che consenta a Bob di trovare una soluzione in modo facile.

Un altro esempio di funzione che sembra essere one-way è la moltiplicazione fra numeri naturali: dati due numeri primi p e q , calcolare il prodotto $n = pq$. Il problema inverso è la fattorizzazione: dato un numero interno n che è il prodotto di due numeri primi, trovarli (ogni numero ammette un'unica rappresentazione in fattori primi).

Un esempio di fattorizzazione:

```

RSA-768=12301866845301177551304949583849627207728535695953347921973224521
517264005072636575187452021997864693899564749427740638459251925573263034
537315482685079170261221429134616704292143116022212404792747377940806653
51419597459856902143413 =
334780716989568987860441698482126908177047949837137685689124313889828837
93878002287614711652531743087737814467999489 × 3674604366679959042824463
37996279526322791581643430876426760322838157396665112792333734171433968
10270092798736308917

```

L'algoritmo "banale" è quello che prova a dividere n dato in input per tutti gli interi da 2 a \sqrt{n} , ma che equivale a un tempo di:

$$O(\sqrt{n}) = O(\sqrt{2^m}) = O(2^{\frac{m}{2}})$$

che è esponenziale rispetto a m (il numero di bit).

Algoritmo di Diffie-Hellman

Vogliono sfruttare queste "mancanze" matematiche in crittografia. Alice e Bob scelgono (in maniera pubblica) un numero primo q e un generatore g del gruppo

ciclico \mathbb{Z}_q^* . Poi Alice sceglie a caso un x_A , con $0 < x_A < q - 1$, e lo tiene segreto. Bob allo stesso modo sceglie un $0 < x_B < q - 1$ e lo tiene segreto.

Ora:

1. Alice calcola $g^{x_A} \bmod q$ e lo manda a Bob. In altre parole nasconde x_A dentro una "cassetta".
2. Bob calcola $g^{x_B} \bmod q$ e lo manda ad Alice.
3. Alice calcola $(g^{x_B})^{x_A} = g^{x_A \cdot x_B} = k$
4. Bob calcola $(g^{x_A})^{x_B} = g^{x_B \cdot x_A} = k$
5. k è il valore della chiave segreta

Eve vede solamente q, g, g^{x_A}, g^{x_B} , ma per trovare x_A o x_B deve risolvere un logaritmo discreto! Anche calcolare $g^{x_A \cdot x_B}$ partendo da g^{x_A} e g^{x_B} , ma anche questo sembra un problema intrattabile.

Esempio

Alice e Bob si mettono d'accordo utilizzando $q = 25307$ e $g = 2$ (che è un generatore di \mathbb{Z}_{25307}^*). Alice sceglie $x_A = 3578$ e Bob $x_B = 19956$. Alice computa $g^{x_A} = 2^{3578} = 6113 \bmod 25307$ e lo manda a Bob. Allo stesso modo Bob calcola $g^{x_B} = 2^{19956} = 7984 \bmod 25307$ e lo manda ad Alice. Quando Bob riceve $g^{x_A} = 6113$ da Alice, egli calcola $k = (g^{x_A})^{x_B} = 6113^{19956} = 3694 \bmod 25307$, e come lui anche Alice calcola $(g^{x_B})^{x_A} = 7984^{3578} = 3694 \bmod 25307$, che è la stessa chiave k calcolata da Bob. Ora Alice e Bob possono usare k come chiave segreta in un crittosistema simmetrico

Crittosistema El Gamal

Ogni utente sceglie un numero primo q , e considera un generatore g del gruppo ciclico \mathbb{Z}_q^* . Come prima sceglie un numero a tale che $0 < a < q - 1$ e computa $g^a \bmod q$. La chiave segreta sarà a , e verrà nascosta dentro $g^a \bmod q$. La chiave pubblica è $k_p(q, g, g^a)$. Per ottenere la chiave segreta dalla chiave pubblica dovremmo computare $\log_g g^a \bmod q$.

Alice vuole mandare un messaggio m a Bob (questa m dev'essere un elemento di \mathbb{Z}_q^* quindi dev'essere minore di $\log_2 q$ bit, altrimenti lo divide in blocchi). Lei prende la chiave pubblica di Bob $k_{pB} = (q, g, g^a)$. Ora sceglie un valore $0 < \ell < q - 1$ e calcola $\gamma = g^\ell \bmod q$ e $\delta = m \cdot (g^a)^\ell \bmod q$. Poi manda a Bob il testo cifrato $c = (\gamma, \delta)$. Il valore ℓ rende i testi cifrati non correlabili fra loro (anche se sono sempre uguali). La decifratura funziona in questo modo:

$$\delta = m \cdot (g^a)^\ell \bmod q.$$

$$m = g^{-a\ell} \cdot \delta$$

$$m = g^{-a\ell} \cdot \delta = \gamma^{-a} \cdot \delta = \gamma^{q-1-a} \cdot \delta$$

Eve, come prima, dovrebbe risolvere un logaritmo discreto per rompere il crittosistema.

Questi crittosistemi a chiave pubblica però come si può vedere sono molto più lenti di quelli simmetrici (centinaia di volte più lenti). Quindi si creano dei crittosistemi ibridi che sfruttano le potenzialità di entrambi: si usa un crittosistema a chiave pubblica per decidere una chiave segreta (chiave di sessione), e poi si utilizza questa in un crittosistema simmetrico. In questo modo si utilizza il crittosistema lento solo per passarsi qualche centinaia di bit.

RSA

Dati due numeri primi molto grandi p e q della stessa dimensione più o meno (almeno 1024 bit ciascuno), ma valori molto diversi.

Si pone $n = pq$ (n chiamato *modulo* di RSA). Ora si crea la funzione toziente di Nepero $\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$.

$\phi(n)$ è il numero di interi compresi fra 1 e n che sono *coprimi* con n ($\phi(n) = |\{x : 1 \leq x \leq n \text{ and } \text{MCD}(x, n) = 1\}|$).

In altre parole $\phi(n)$ è il numero di elementi contenuti in \mathbb{Z}_N^* .

Si sceglie un d a caso tale che $1 < d < \phi(n)$ e che $\text{MCD}(d, \phi(n)) = 1$, quindi d è invertibile modulo $\phi(n)$ (quindi è invertibile in $\mathbb{Z}_{\phi(N)}$)

Usando l'algoritmo di Euclide esteso si computa e tale per cui $e \cdot d \equiv 1 \pmod{\phi(n)}$ (che vuol dire $e \equiv d^{-1} \pmod{\phi(n)}$)

La chiave pubblica è data dalla coppia (n, e) , mentre la chiave segreta sarà d . Ovviamente i valori di p, q , e $\phi(n)$ devono rimanere segreti.

- **Cifratura:**

$$c = m^e \pmod{n}$$

Dove m ed e fanno parte della chiave pubblica. Se la lunghezza del messaggio m da cifrare ha una dimensione più grande di n dev'essere diviso in blocchi.

- **Decifratura:**

$$c^d \pmod{n} = m$$

Essa funziona in questo modo:

$$c^d \pmod{n} \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Ora, siccome $ed \equiv 1 \pmod{\phi(n)}$, per definizione di congruenza esiste una k intera tale per cui $ed = 1 + k\phi(n)$, da cui:

$$c^d \pmod{n} \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \pmod{n}$$

Per il piccolo teorema di Fermat sappiamo che $m^{\phi(n)} \equiv 1 \pmod{n}$, quindi:

$$c^d \pmod{n} \equiv m \cdot (1)^k \equiv m \cdot 1 \equiv m \pmod{n}$$

Si sfrutta quindi l'esponenziazione modulare che è una permutazione one-way (la trapdoor è la chiave segreta).

Esempio

Bob sceglie $p = 101$ e $q = 113$, da cui $n_B = pq = 11413$ e $\phi(n_B) = (p-1)(q-1) = 11200$.

Ora deve scelgere una d_B fra 2 e $\phi(n_B) - 1 = 11199$ tale che l'MCD fra d_B e $\phi(n_B)$ sia uguale a 1.

Mettiamo che scelga $d_B = 6597$.

Usando l'algoritmo di Euclide esteso, Bob si calcola

$$e_B \equiv d_B^{-1} \pmod{\phi(n_B)} = 3533 \pmod{11200}$$

Quindi la chiave pubblica è data dalla coppia $(n_B, e_B) = (11413, 3533)$, la chiave privata è $d_B = 6597$.

Se Alice volesse mandare il messaggio $m = 9726$ e Bob, allora si calcola $c = 9726^{3533} \pmod{11414} = 5761$ e manda c a Bob.

Egli recupera m calcolando:

$$c^{d_B} = 5761^{6597} \pmod{11413} = 9726$$

Ovviamente nella realtà le dimensioni dei numeri sono di almeno 1024 bit.

Quanto è difficile per Eve conoscere $\phi(n)$? È facile se conosciamo la fattorizzazione di n :

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

È difficile se partiamo da n senza conoscere la sua fattorizzazione: in altre parole rompere RSA è equivalente a fattorizzare n .

Se Eve riuscisse a scoprire $\phi(n)$ può fattorizzare n :

- Sa che $n = p \cdot q$ e che $\phi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p+q) + 1$
- Calcola $p + q = n - \phi(n) + 1$
- Ora che conosce la somma e il prodotto di p e di q , li calcola risolvendo la seguente equazione quadratica $x^2 - (p+q)x + n = 0$.

I valori di p e q non dovrebbero essere troppo vicini fra loro (altrimenti si avvicinano alla \sqrt{n} e quindi provando a cercare intorno a quel valore troviamo uno dei due fattori e fattorizziamo n). Vediamo un caso in cui p e q sono vicini:

- Assumiamo che $p > q$
- Se i due sono vicini, allora $\frac{p-q}{2}$ è piccolo, e $\frac{p+q}{2}$ è leggermente più grande di \sqrt{n} .
- Dalla sequenza uguaglianza (che è valida in generale):

$$\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$$

Deduciamo che $\frac{(p+q)^2}{4}$ è un quadrato perfetto.

- Allora cerchiamo degli interi $x > \sqrt{n}$ tali per cui $x^2 - n$ è un quadrato perfetto che chiamiamo y^2 .
- Da $y^2 = x^2 - n$ otteniamo:

$$n = x^2 - y^2 = (x + y)(x - y)$$

E quindi abbiamo fattorizzato! $p = x + y$ e $q = x - y$

Dovremmo anche occuparci del valore di $\phi(n)$: assumiamo che il MCD è grande, allora

$$u = \text{mcm}(p-1, q-1) = \frac{(p-1)(q-1)}{\text{MCD}(p-1, q-1)} = \frac{\phi(n)}{\text{mcm}(p-1, q-1)}$$

Dalle proprietà dei campi se $d' \equiv e^{-1} \pmod{u}$ si può utilizzare d' per decifrare al posto di d . Ma siccome u è piccolo, si può trovare a tentativi. Quindi è meglio se $p-1$ e $q-1$ abbiano divisori grandi.

Un altro errore è utilizzare lo stesso modulo n per un gruppo di utenti. Alice manda

$$c_1 = m^{e_1} \pmod{n}$$

$$c_2 = m^{e_2} \pmod{n}$$

Se $\text{MCD}(e_1, e_2) = 1$ (sono coprimi fra loro) allora Eve può usare il teorema di Euclide esteso e calcola r e s tali per cui $re_1 + se_2 = 1$. Una volta calcolato r e s Eve può calcolare:

$$c_1^r c_2^s \equiv m^{re_1} m^{se_2} \equiv m^{re_1 + se_2} \equiv m \pmod{n}$$

Anche utilizzare esponenti piccolo per valori diversi del modulo può essere un problema: mettiamo che un utente vuole mandare m a tre utenti A, B, C usando tre moduli diversi:

$$c_A = m^3 \pmod{n_A}$$

$$c_B = m^3 \pmod{n_B}$$

$$c_C = m^3 \pmod{n_C}$$

Se n_A, n_B e n_C sono coprimi allora Eve può usare il teorema dei Cinesi per calcolare un valore m la cui radice cubica intera ritorna m .

Rompere RSA equivale a fattorizzare n .

Randomized RSA

Un modo robusto di utilizzare RSA è la seguente: supponiamo di avere una sequenza di singoli bit da cifrare. Allora prendiamo la chiave pubblica di RSA (n, e) , allora se cifriamo 0 rimane 0, e così anche 1:

$$0^e \equiv 0 \pmod{n}$$

$$1^e \equiv 1 \pmod{n}$$

quindi il testo cifrato è uguale al testo in chiaro! Si potrebbe risolvere questa cosa creando pacchetti di messaggi, ma magari non è possibile perchè devo mandarli immediatamente. Qualcuno ha però dimostrato che il bit meno significativo del testo in chiaro è un hard-core bit per l'esponenziazione modulare $m^e \pmod{n}$ (quindi è difficile calcolare il bit meno significativo del testo in chiaro senza trapdoor, ovvero d aka one-way function) L'idea è quella di mettere il bit da cifrare come bit meno significativo, e gli altri bit vengono messi a random!

Esempio

Alice vuole mandare a Bob un bit: prende la chiave pubblica di Bob (n_B, e_B) . Sceglie poi un intero random $x < \frac{n_B}{2}$ (quindi $2x < n_B$). Poi manda a Bob $y = (2x + b)^{e_B} \pmod{n_B}$.

Bob quando riceve y computa $y^{d_B} \pmod{n_B} = 2x + b$ e prende il bit meno significativo del risultato.

Ovviamente questo è fattibile per cifrare qualche bit ogni tanto, e non per cifrare un bit alla volta un film.

Lezione 21

Firme digitali e funzioni di hash

Una firma digitale è una quintupla (P, A, K, S, V) dove:

- P è l'insieme di tutti i possibili messaggi che vorremmo firmare
- A è l'insieme di tutte le possibili firme
- K è l'insieme di tutte le possibili chiavi
- $S = \{\text{sig} : P \times K \rightarrow A\}$ è l'insieme delle funzioni di firma
- $V = \{\text{ver} : P \times A \times K \rightarrow \{T, F\}\}$ è l'insieme delle funzioni di verifica

Come nelle funzioni di cifratura e decifratura, scegliendo una chiave $k \in K$ essa diventa un parametro:

$$\begin{aligned}\text{sig}_k &: P \rightarrow A \\ \text{ver}_k &: P \times A \rightarrow \{T, F\}\end{aligned}$$

La funzione di verifica mi dice sempre T o F.

La coppia (x, y) è chiamata un messaggio firmato.

Diversamente da un autografo (che è fatto a mano), le firme:

- la firma y è separata dal documento x
- la firma non è sempre la stessa: dipende dal documento, quindi ce ne sono diverse per diversi documenti

La firma digitale serve per autenticare l'originatore del messaggio: solo chi conosce una certa informazione segreta può produrre la firma del messaggio.

Per firmare serve una chiave segreta, per verificare si usa una chiave pubblica; quindi il meccanismo è simile a quello dei crittosistemi a chiave pubblica ma al contrario.

Attenzione: autentica il generatore del messaggio ma non chi lo ha spedito!

Esempio

Alice vuole firmare un messaggio m :

- Lei sceglie una coppia $(\text{sig}_k, \text{ver}_k)$ di algoritmi
- Mantiene segreta sig_k e rende ver_k pubblica
- Calcola $\sigma = \text{sig}_k(m)$

A questo punto Bob vuole verificare la firma prodotta da Alice:

- Considera la coppia (m, σ)
- Prende l'algoritmo ver_k e accetta la valida se e solo se $\text{ver}_k(m, \sigma) = T$.

Eve vorrebbe:

- Rompere totalmente il sistema: in qualche modo riesce a determinare la chiave segreta k e quindi firmare documenti al suo posto
- Existential forgery: dopo aver osservato qualche coppia $(x_1, y_1), \dots, (x_i, y_i)$ di messaggi con le rispettive firme, quando un nuovo messaggio arriva Eve può produrre una valida firma (valida solo per quel messaggio o per una classe di messaggi che hanno qualche similarità).

Quando Alice vuole mandare un messaggio firmato m (non cifrato, solo firmato) a Bob, lei ha una funzione di cifratura E_A e una funzione di decifratura D_A . Mantiene segreta D_A mentre E_A è pubblica.

A questo punto se vuole firmare m usa $D_A(m)$ e la manda a Bob. Egli per verificare la firma prende l'algoritmo pubblico E_A e controlla che

$$E_A(D_A(m)) = m$$

E accetta la firma se e solo se la funzione di verifica ritorna vero.

Di solito però si firma un documento cifrato, quindi Alice calcola $D_A(E_B(m))$ utilizzando E_B come algoritmo di cifratura pubblico di Bob.

Bob infine per trovare m calcolerà:

$$D_B(E_A(D_A(E_B(m)))) = m$$

Quindi controlla la firma e poi decifra il messaggio.

Così però non va bene: se Eve intercettasse potrebbe prendere la firma e impersonificare Alice mandando $D_E(E_B(m))$. Quindi è meglio cifrare anche la firma, così solo Bob può leggere la firma.

Schema El Gamal

La base su cui è stato prodotto DSA (Digital Signature Algorithm) che è algoritmo usato ora.

Lo schema di El-Gamal non è deterministico: ogni messaggio (anche uguale) ha sempre una firma diversa. Come il suo crittosistema, la sicurezza si basa sui logaritmi discreti.

- Si parte da un numero primo p
- Si trova un generatore g del generatore del gruppo ciclico \mathbb{Z}_p^*
- I messaggi da cifrare sono elementi di \mathbb{Z}_p^*
- Le firme sono coppie (γ, δ) con $\gamma \in \mathbb{Z}_p^*$ e $\delta \in \mathbb{Z}_{p-1}$ dove \mathbb{Z}_{p-1} non sarà un campo ($p - 1$ è pari) ma un anello o boh

Il problema, come nel crittosistema, la cifra digitale del messaggio è grande circa il doppio del messaggio cifrato.

La chiave segreta sarà a , con $0 < a < p - 1$, Alice calcola $\beta = g^a \pmod{p}$. La chiave pubblica è la tripla (p, g, β) (gruppo ciclico, generatore e versione nascosta della chiave segreta). Per firmare m sceglie una $k \in \mathbb{Z}_{p-1}^*$, calcola $\gamma = g^k \pmod{p}$ e $\delta = (m - a\gamma)k^{-1} \pmod{p-1}$. La coppia (γ, δ) è la firma di m .

Per ottenere m , mi basta fare le sostituzioni:

$$m = a\gamma + k\delta \pmod{p-1}$$

Bob accetta la firma se e solo se:

$$\beta^\gamma \gamma^\delta \equiv g^m \pmod{p}$$

Infatti se la firma è stata effettuata correttamente allora

$$\beta^\gamma \gamma^\delta \equiv g^{a\gamma} \cdot g^{k\delta} \equiv g^m \pmod{p}$$

Mettiamo che Eve voglia produrre una firma per un messaggio m senza conoscere il valore di a . Se Eve sceglie γ e vuole calcolare il valore di δ in modo tale da rendere vera $\beta^\gamma \gamma^\delta \equiv g^m \pmod{p}$. Il problema è che per farlo deve calcolarsi il seguente logaritmo discreto:

$$\log_\gamma(g^m \beta^{-\gamma})$$

Invece se sceglie δ e vuole computare γ , deve risolvere l'equazione $\beta^\gamma \cdot \gamma^\delta \equiv g^m \pmod{p}$ rispetto a γ ma è infattibile anche questo problema.

Infine se sceglie sia γ che δ e vuole calcolarsi un valore per m , deve calcolare il logaritmo discreto $\log_g \beta^\gamma \cdot \gamma^\delta$.

Il problema è che Eve può calcolare γ, δ e m contemporaneamente! (procedimento sbatti)

Però è possibile calcolare un messaggio m e verificare che (γ, δ) sia una firma valida. Quindi questo schema non è completamente rotto nel senso di aver trovato la chiave segreta, però così Eve può calcolare messaggi e firme.

Il problema è che la firma diventa troppo grande e in più El Gamal ha questa falla.

Sarebbe bello se avessimo una funzione che prende in input un messaggio di lunghezza arbitraria e ritorni un piccolo output in one-way. Soluzione: funzioni Hash! A questo punto anziché firmare il messaggio fermo l'impronta hash del messaggio e non ho più il problema della dimensione visto che viene il doppio non più del messaggio.

Questo previene anche l'attacco al El Gamal visto che posso calcolare i valori di γ, δ e m , ma non so qual è il messaggio che produce quell'impronta (m)!

Funzioni di Hash

Esse calcolano un'impronta (o message digest) per un dato in input. Questa è piccola e di solito ha una lunghezza fissa (128 bit per MD5, 160 bit per SHA-1, 256 bit per SHA-256). Possono essere usate per verificare l'integrità di un messaggio trasmesso.

Definizione: una famiglia di funzioni di hash è una quadrupla (X, Y, K, H) dove:

- X è l'insieme dei possibili messaggi
- Y è l'insieme delle possibili impronte
- K è l'insieme delle possibili chiavi (noi non usiamo)
- $H = \{h_k : X \rightarrow Y | k \in K\}$ è l'insieme delle funzioni hash

L'insieme X dei messaggi può essere finito o infinito (molto molto grande), mentre l'insieme delle impronte Y è sempre finito. Possiamo concludere che $|X| \geq |Y|$, anzi si assume che $|X| \geq 2|Y|$. In pratica è una funzione di compressione (da non confondere con archivi zip ecc).

I tre problemi difficili da risolvere quando si parla di funzioni di hash:

1. Preimage:

Input: funzione di hash $h : X \rightarrow Y$ e $y \in Y$
Output $x \in X$ tale che $h(x) = y$

In pratica ci viene chiesto di invertire la funzione di hash, h dev'essere quindi one-way (data l'impronta dev'essere difficile calcolare un input che la generi)

2. Second preimage:

Input: funzione di hash $h : X \rightarrow Y$ e $x \in X$
Output: $x' \in X$ tale che $x \neq x'$ e $h(x') = h(x)$

In pratica dato un messaggio x , e la sua impronta calcolata tramite una funzione h dev'essere difficile trovare un messaggio x' che abbia lo stesso valore in output della funzione

3. Collision:

Input: funzione di hash $h : X \rightarrow Y$
Output: $x, x' \in X$ tali che $x' \neq x$ e che $h(x') = h(x)$

Dev'essere difficile trovare due x che abbiano la stessa impronta (esempio ti mando un file x e poi ti dico guarda che in realtà ti ho mandato x').

Esso è più semplice da attaccare rispetto a second preimage, posso sempre provare ad attaccarlo.

Algoritmo FindCollision

Osservazione: collision è più facile da attaccare rispetto a second preimage.
L'algoritmo assume che possiamo valutare la funzione h per q volte

```
Scegli a caso  $X_0 \subseteq X$ , con  $|X_0| = q$ 
for all  $x \in X_0$  do
    Calcola  $y_x = h(x)$ 
    if  $y_x = y_{x'}$  per qualche  $x \neq x'$  then return  $(x, x')$ 
    else return "Fail"
```

Da notare che l'algoritmo non è completamente deterministico (a causa della scelta casuale su X_0 , se sono fortunato mi ritorna una coppia altrimenti un errore).

Lezione 22

Vogliamo capire qual è la probabilità di successo di FindCollision, proviamo a riscriverlo in maniera più facile da analizzare:

```

Scegli a caso  $X_0 \subseteq X$ , con  $|X_0| = q$ 
fingerprints = []
for all  $x \in X_0$  do
    Calcola  $y_x = h(x)$ 
    if  $y_x \in \text{fingerprints}$  then return  $(x, x')$ 
    else add  $y_x$  to fingerprints
return "Fail"

```

Teorema: diciamo che $M = |Y|$ (insieme delle possibili impronte). La probabilità di successo ϵ di FindCollision è

$$\epsilon = 1 - \left(\frac{M-1}{M} \right) \left(\frac{M-2}{M} \right) \dots \left(\frac{M-q}{M} \right)$$

Dimostrazione: dato $X_0 = \{x_1, \dots, x_q\}$, per ogni i compresa in $0 < i \leq q$ si definisce E_i l'evento $h(x_i) \notin \{h(x_1), \dots, h(x_{i-1})\}$, allora $\text{Prob}[E_1] = 1$, e le altre:

$$\text{Prob}[E_1 \wedge E_2 \wedge \dots \wedge E_q] = \left(\frac{M-1}{M} \right) \left(\frac{M-2}{M} \right) \dots \left(\frac{M-q+q}{M} \right)$$

Questa è la probabilità che fallisca, quindi la ϵ sopra è quella che abbia successo.

Il paradosso del compleanno

Riscrivo la probabilità di non avere collisioni come:

$$(1 - \frac{1}{M})(1 - \frac{2}{M}) \dots (1 - \frac{q-1}{M}) = \prod_{i=1}^{q-1} \left(1 - \frac{i}{M}\right)$$

Ora approssimiamo $1 - x \approx e^{-x}$, quindi

$$\text{Prob[NoCollisions]} \approx e^{-\frac{1}{M}} \approx e^{-\frac{1}{M} \sum_{i=1}^{q-1} i} = e^{-\frac{q(q-1)}{2M}}$$

quindi la probabilità di trovare (almeno) una collisione è

$$\text{Prob[AtLeastOneCollision]} \approx 1 - e^{-\frac{q(q-1)}{2M}} = \epsilon$$

Questa relazione lega: probabilità di avere una collisioni (ϵ), numero di possibili impronte(M) e lo "sforzo" computazionale (numero di query) che siamo disposti a fare (q) nel cercare una collisione.

Ora proviamo ad estrarre lo sforzo computazionale:

$$e^{-\frac{q(q-1)}{2M}} = 1 - \epsilon$$

$$-\frac{q(q-1)}{2M} = \ln(1-e)$$

$$q^2 - q = 2M\ln\left(\frac{1}{1-\epsilon}\right)$$

$$q \approx \sqrt{2M\ln\left(\frac{1}{1-\epsilon}\right)}$$

Quindi questo valore rappresenta (circa) il numero di query che devo fare per trovare almeno una collisione con probabilità ϵ che ha M possibili impronte.

Proviamo $\epsilon = \frac{1}{2}$:

$$q \approx \sqrt{2M\ln 2}$$

$$q \approx 1.17\sqrt{M}$$

Quindi le hash di circa \sqrt{M} elementi abbiamo una collisione con probabilità $\frac{1}{2}$.

Con 40 bit di impronta abbiamo $M = 2^{40} \rightarrow \sqrt{M} = 2^{20} \approx 10^6$ (Non sicuro! 1 milione li provo in tra).

Con 128 bit di impronta abbiamo $M = 2^{128} \rightarrow \sqrt{M} = 2^{64} \approx 10^6$ (Sicuro per poco tempo).

Di solito si utilizzano *almeno* 160 bit.

Questa è la stessa cosa di chiedersi quante persone servono per avere la probabilità $\frac{1}{2}$ che due di loro siano nate nello stesso giorno:

$$\sqrt{365} \cdot 1.17 \approx 23 \text{ lol}$$

Costruzione di funzioni di hash iterate

Data una funzione di compressione $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ con $t \geq 1$

Le tecnica è composta da tre fasi:

- **Pre-computazione**

Data una stringa in input x che ha lunghezza almeno $m+t+1$, creiamo una stringa y tale per cui $|y| \equiv 0 \pmod{t}$ (quindi che abbia lunghezza multiplo di t).

Di solito si usa una funzione di padding $pad(x)$ che aggiunge bit alla fine uguali a zero, poi dividiamo y in r blocchi, ognuno di t bit:

$$y = y_1 || y_2 || \dots || y_r$$

Questa fase di pre-computazione dovrebbe assicurare che la funzione $f(x) = y$ computata sia la più vicina possibile ad una iniettiva (input diversi hanno output diversi) se no è più facile trovare collisioni. Questa condizione non è difficile da soddisfare in quanto $|y| = rt \geq |x|$.

- **Computazione:**

Definiamo un vettore di inizializzazione $z_0 = IV$ (IV = Initialization Vector) che di solito è noto.

$$z_0 = IV$$

$$z_1 = \text{compress}(z_0 || y_1)$$

$$z_2 = \text{compress}(z_1 || y_2)$$

...

$$z_r = \text{compress}(z_{r-1} || y_r)$$

- **Trasformazione output:**

Si trasforma l'output (opzionale)

Questa costruzione generale è stata specializzata da Merkle e Damgard in due modi.

Con questa costruzione possiamo dimostrare che se la funzione *comprimi* è resistente alle collisioni allora anche la funzione *hash* iterata è resistente alle collisioni.

Si prende un input x di lunghezza n , con $|x| = n \geq m + t + 1$ per $t \geq 2$. Si divide x in k blocchi, ognuno di $t - 1$ bits:

$$x = x_1 || x_2 || \dots || x_k$$

con $k = [\frac{n}{t-1}]$ e $|x_k| = t - 1 - d$ con $0 \leq d \leq t - 2$ (d sarebbe la lunghezza del pezzo che manca per creare padding, si aggiungono tanti 0).

Si aggiunge un blocco in $k + 1$ che contiene il valore del numero d (così so quanti 0 ho aggiunto per padding).

mhhh

Alcune funzioni di Hash

La prima funzione di hash è stata MD4 nel 1990, MD5 è una modifica di MD4 che è più resistente agli attacchi. Però queste versioni sono suscettibili all'attacco del compleanno.

Viene introdotto SHA come standard, che poi diventa SHA-1 dopo aver corretto una falla. Anche in SHA-1 però è stata trovata una collisione, quindi ora bisogna usare SHA-256.

SHA-1 non chiesto all'orale come funziona

Digital Signature Algorithm

È stato preso El Gamal e sono state effettuate alcune modifiche.

"i dettagli non ve li chiedo all'orale"

=)

min2s