Corso di *Data Security & Privacy*[*]
Esercizi su crittografia a chiave condivisa

Michele Boreale
Università di Firenze
Dipartimento Statistica, Informatica, Applicazioni
michele.boreale@unifi.it

1 Esercizi di verifica

I seguenti esercizi vertono sugli argomenti esposti nei capitoli da 1 a 6 delle *Note*. Di molti di essi viene fornita la traccia, più o meno dettagliata, di un possibile svolgimento.

1. Si citi un'applicazione per cui può essere utile richiedere la *Data Origin Authentication*, e una per cui può essere utile la *Peer Entity Authentication*.

   Traccia. Pensare ai due più comuni servizi di Internet.

2. Si definisca precisamente il servizio di *Data Origin Authentication*. Quindi, si faccia l'esempio di un cifrario a chiave condivisa in cui la corretta decifrazione di un messaggio $C = E_k[M]$ da parte del ricevente assicura il ricevente stesso dell'autenticità e integrità del messaggio. Quale proprietà del cifrario è alla base della garanzia di autenticità e integrità?

   Traccia. Si faccia riferimento all'*Avalanche Criterion*, che dovrebbe essere soddisfatto da un buon cifrario a chiave condivisa. Nel caso della decryption, questo implica che una modifica di pochi bit del ciphertext si ripercuote a valanga e in modo casuale casuale nel plaintext. Dunque se un attaccante attivo modifica un ciphertext $C = E_K[M]$ in C', alterandone qualche bit, con alta probabilità il risultato di decifrare C' con K sarà una stringa senza senso (non un plaintext legittimo), che quindi verrà rigettata dal ricevente come non autentica.

   Un cifrario che gode di questa proprietà è, per esempio, DES.

3. Un attaccante intercetta il plaintext[1] $P =$`meetmeateight`, e il relativo ciphertext, C. E' noto che cifrario usato è quello di Hill, con blocchi di dimensione $m = 2$. Dire se l'attaccante può condurre con successo o no un attacco di tipo known-plaintext per recuperare la chiave (*Suggerimento*: 2 è un fattore di 26...).

   Traccia. Si noti prima di tutto che *non* viene richiesto di portare effettivamente a termine l'attacco, ma solo di determinare se esso è possibile. Ricordiamo che per portare a termine l'attacco è necessario che l'attaccante disponga di una matrice P^* di blocchi

[*] Laurea Magistrale in Informatica, Università di Firenze, A.A. 2022-2023, II semestre.

[1] La corrispondenza lettere-numeri è:

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

(colonne) di plaintext, che sia invertibile $\mod 26$. La condizione di invertibilità modulo 26 è, come si è visto, che $\mathrm{MCD}(\det P^*, 26) = 1$. Ora, consideriamo una matrice P^* ottenibile giustapponendo due generici blocchi del plaintext dato, visti come vettori colonna, $[a, b]^T$ e $[c, d]^T$:

$$P^* = \begin{bmatrix} a & c \\ b & d \end{bmatrix}.$$

Vale $\det P^* = ad - cd$. Poiché le posizioni di indice 1, 3, 5,... del plaintext sono tutte occupate da lettere *pari*, a e c sono pari. Ne segue che $\det P^*$ è pari (perché?). Da questo fatto segue facilmente la conclusione.

4. Il cifrario *Affine Cipher* è una variante di Hill, in cui fa parte della chiave anche un vettore, che viene sommato al prodotto chiave×plaintext. Si definisca l'Affine Cipher di dimensione m, precisando esattamente anche cosa sono $\mathcal{P}, \mathcal{K}$ e $\mathcal{C}$.

   Traccia. Poniamo $\mathcal{P} \subseteq \mathbb{Z}_{26}^m$, $\mathcal{C} = \mathbb{Z}_{26}^m$ e

   $$\mathcal{K} = \{(A, B) : A \text{ è una matrice } m \times m \text{ invertibile } \mod 26 \text{ e } B \in \mathbb{Z}_{26}^m\}.$$

   Definiamo (vedendo i blocchi di plaintext e ciphertext come vettori-colonna):

   $$\begin{aligned} E_K(P) &\stackrel{\text{def}}{=} A \times P + B \mod 26 \\ D_K(C) &\stackrel{\text{def}}{=} A^{-1} \times (C - B) \mod 26. \end{aligned}$$

5. Si descriva un attacco ciphertext-only all'Affine Cipher con blocchi di dimensione $m = 1$, che sfrutti il fatto che il plaintext è composto da caratteri generati secondo le frequenze caratteristiche della lingua Inglese, dove le due lettere più frequenti sono `e` $= 4$ e `t` $= 19$.

   Traccia. Supponiamo dapprima che l'attaccante possieda due coppie plaintext-ciphertext (m_i, c_i) $(i = 1, 2)$. Ciascuna di esse corrisponde ad una equazione soddisfatta dalla chiave $k = (a, b)$ (sconosciuta):

   $$c_1 = a \times m_1 + b \mod 26 \quad (1)$$
   $$c_2 = a \times m_2 + b \mod 26 \quad (2)$$

   L'attaccante quindi prova a risolvere le due equazioni di cui sopra, nelle incognite a e b. Sottraendo membro a membro, otteniamo un'equazione nella sola incognita a:

   $$c_1 - c_2 = a \times (m_1 - m_2) \mod 26$$

   Possiamo risolvere questa equazione a patto che esista $(m_1 - m_2)^{-1} \mod 26$, cioè che $\mathrm{MCD}(m_1 - m_2, 26) = 1$, e trovare

   $$a = (c_1 - c_2) \times (m_1 - m_2)^{-1} \mod 26.$$

   Una volta trovato a, possiamo ricavare b dalla (1) o (2). Dunque la condizione sufficiente per l'attacco è di avere due coppie tali che $m_1 - m_2$ sia invertibile modulo 26.

   Nel caso della lingua Inglese, queste le due coppie in questione possono essere recuperate assumendo che i due ciphertext più frequenti nel ciphertext, c_1 e c_2, corrispondano rispettivamente alle due lettere più frequenti in Inglese, ovvero $m_1 =$ `e` $= 4$ e $m_2 =$ `t` $= 19$. Si noti che $m_1 - m_2 = 4 - 19 = -15 = 11 \mod 26$ è invertibile modulo 26.

6. Dire qual è il valore atteso di $I_{\mathbf{c}}(\mathbf{y})$ quando $\mathbf{y}$ è una sequenza casuale di lettere di A.

   Traccia. Sia $\mathbf{y} = (y_1, ..., y_n)$ una sequenza di lettere dell'alfabeto A, generate casualmente e indipendentemente l'una dall'altra (ripetizioni ammesse). Allora, per n abbastanza grande rispetto alla dimensione di A, abbiamo:

$$I_{\mathbf{c}}(\mathbf{y}) = \sum_{i \in A} \frac{f_i}{n} \cdot \frac{f_i - 1}{n - 1} \approx \sum_{i \in A} \frac{1}{|A|^2} = \frac{1}{|A|} .$$

   Per esempio, se $A = \mathbb{Z}_{26}$, $I_{\mathbf{c}}(\mathbf{y}) \approx 1/26$.

7. Si dia un algoritmo per generare una permutazione casuale delle lettere dell'alfabeto, vista come sequenza (array) di 26 elementi senza ripetizioni, assumendo di avere a disposizione una primitiva per la generazione di numeri casuali `randint(0,j)`, che quando invocata restituisce un numero intero casuale tra 0 e j compresi. In particolare

   (a) Dare prima un algoritmo elementare, che riempie le posizioni di un vettore di 26 elementi una alla volta, chiamando ripetutamente `randint(0,25)`, ripetendo l'estrazione quando viene estratto un elemento già presente. Stimare quante chiamate a `randint(0,25)`, in media, sono necessarie.

   (b) Formalizzare il seguente algoritmo alternativo (Fisher-Yates) che procede per cancellazioni successive da un elenco. Programmarlo in Python. Discuterne la complessità computazionale.

      1. Scrivi in ordine da sinistra a destra gli elementi da 0 a 25;
      2. Scegli un numero casuale k compreso tra 0 e il numero di elementi non ancora cancellati (escluso);
      3. Contando da sinistra, cancella l'elemento k-mo non ancora cancellato, e scrivilo alla fine di un elenco separato;
      4. Ripetere dal passaggio 2 fino a quando tutti gli elementi non siano stati cancellati.

      L'elenco separato di numeri costruito via via al passo 3 è ora una permutazione casuale dei numeri da 0 a 25.

   (c) Al passo 3 sopra del metodo di Fisher-Yates, si spreca del tempo a contare gli elementi rimanenti. In una variante dell'algoritmo, detto algoritmo di Durstenfeld (o Knuth shuffle), ad ogni iterazione l'elemento cancellato al passo 2 viene spostato alla fine dell'elenco tramite un opportuno scambio. Ciò riduce la complessità temporale dell'algoritmo a $O(n)$. Formalizzare questo algoritmo e programmarlo in Python.

   Per ulteriori dettagli su questo algoritmo, si consulti la pagina Wikipedia `https://en.wikipedia.org/wiki/Fisher-Yates_shuffle`.

8. Si discuta, ferma restando la segretezza della chiave, se e come in One-Time-Pad sono possibili attacchi attivi all'autenticazione. (*Sugg.* Provare la proprietà dello XOR: $\overline{x \oplus y} = \overline{x} \oplus y$).

   Traccia. Ci si rifaccia all'Esercizio 1 e ci si interroghi su questo: One-Time-Pad possiede la proprietà espressa dall'*Avalanche Criterion*? Si sfrutti poi la proprietà dello XOR enunciata nel suggerimento per far vedere come l'attaccante possa, invertendo bit in posizioni del ciphertext da lui selezionate, invertire i corrispondenti bit del plaintext.

Tale modifica può passare inosservata da parte del ricevente, se il risultato della modifica del plaintext genuino è anch'esso un plaintext.

9. Si consideri il cifrario definito da $\mathcal{P} = \{a, b\}$, $\mathcal{K} = \{k_1, k_2, k_3\}$ e $\mathcal{C} = \{w, x, y, z\}$, e dalla funzione di encryption definita nella seguente tabella (accanto ad ogni plaintext e chiave è scritta la corrispondente probabilità):

	a $(\frac{3}{4})$	b $(\frac{1}{4})$
k_1 $(\frac{1}{4})$	w	x
k_2 $(\frac{1}{2})$	x	y
k_3 $(\frac{1}{4})$	y	z

(a) Si calcoli $\Pr(M = m|C = x)$, per ciascun plaintext $m = a, b$.
(b) Si dica se il cifrario è o meno perfetto.

Traccia. (a) Usando le solite abbreviazioni, sappiamo che in generale, per un qualsiasi cifrario, vale (limitandosi a m e c di probabilità non nulla):

$$p(m|c) = \frac{p(c|m)p(m)}{p(c)} \tag{3}$$

dove

$$p(c|m) = \sum_{k:E_k[m]=c} p(k)$$

e

$$p(c) = \sum_{(k,m):E_k[m]=c} p(k)p(m)\,.$$

Ci siamo quindi ricondotti alle probabilità note, specificate nella definizione del cifrario. Usando queste formule per il caso $c = x$ otteniamo:

$$\begin{aligned} p(x|a) &= p(k_2) = \frac{1}{2} \\ p(x|b) &= p(k_1) = \frac{1}{4} \end{aligned}$$

e

$$p(x) = p(k_2)p(a) + p(k_1)p(b) = \frac{7}{16}.$$

E infine, sostituendo quanto sopra trovato in (3):

$$\begin{aligned} p(a|x) &= \frac{\frac{1}{2}\cdot\frac{3}{4}}{\frac{7}{16}} = \frac{6}{7} \\ p(b|x) &= \frac{\frac{1}{4}\cdot\frac{1}{4}}{\frac{7}{16}} = \frac{1}{7}\,. \end{aligned}$$

(b) Si può vedere ispezionando la tabella, e senza fare alcun calcolo, che cifrario non è perfetto: per esempio, quanto vale $Pr(M = a|C = z)$?

10. Si dimostri, o si confuti con un controesempio, la seguente affermazione: in ogni cifrario perfetto, per ogni coppia ciphertext-chiave (c, k) esiste al più un plaintext m tale che $E_k[m] = c$.

Traccia. Riflettere sull'iniettività della funzione di encryption $E_k[\cdot]$.

11. Si provino o confutino tramite controesempi le seguenti affermazioni, dove m, k e c variano su elementi di probabilità non nulla.

    (a) In un cifrario perfetto, per ogni c e k esiste un m tale che $E_k[m] = c$.

    (b) In un cifrario perfetto, per ogni c e m esiste un k tale che $E_k[m] = c$.

    Traccia. La prima affermazione è falsa: possono esistere delle chiavi per cui un certo ciphertext non è generabile, qualunque sia il plaintext scelto. Per esempio, in un OTP su blocchi di 4 bit, si scelga $\mathcal{P} = \{0000, 1111\}$, dove i due plaintext hanno ciascuno una probabilità non nulla. Preso $c = 1100$ e $k = 0100$, si vede c non è generabile a partire da k, per nessuno dei plaintext.

    La seconda affermazione è vera: se fosse falsa, verrebbe meno la condizione di cifrario perfetto (perché?).

12. Si dimostri che in ogni cifrario perfetto con $|\mathcal{C}| = |\mathcal{K}|$ (indicando qui con $|\cdot|$ is numero di elementi aventi probabilità non nulla) vale quanto segue: per ogni coppia plaintext-ciphertext (m, c), esiste esattamente una chiave k tale che $E_k[m] = c$.

    Traccia. Consideriamo qui solo m e c di probabilità non nulla. L'affermazione da dimostrare vuol dire che da ogni m esiste un solo modo - una sola chiave - per andare a c.

    Intanto, è facile vedere, usando Bayes, che $p(c) = p(c|m) > 0$. Ora, $p(c|m) > 0$ implica che, fissato m, deve esistere *almeno una* chiave che porta da m a c. D'altra parte, poiché questo fatto vale per ogni c possibile, vuol dire che tutti i ciphertext sono raggiungibili da m, usando opportune chiavi. Dunque

    $$|\mathcal{C}| = |\{E_k[m] : k \in \mathcal{K}\}|.$$

    Ora, per definizione $|\{E_k[m] : k \in \mathcal{K}\}| \leq |\mathcal{K}|$. Inoltre l'uguaglianza vale solo a patto che chiavi distinte danno chiphertext distinti a partire da m. Ovvero, per ogni m e c c'è una sola chiave che porta a c. Se questo non fosse vero, dunque, avremmo:

    $$|\mathcal{C}| = |\{E_k[m] : k \in \mathcal{K}\}| < |\mathcal{K}| = |\mathcal{C}|.$$

    che è naturalmente una contraddizione. Dunque l'affermazione che per ogni m e c c'è una sola chiave che porta a c deve essere vera.

13. Si formalizzi in maniera rigorosa il cifrario Shift Encryption visto a lezione, precisando tra l'altro cosa sono $\mathcal{P}$, $\mathcal{C}$ e $\mathcal{K}$, come viene generato il plaintext e come viene scelta la chiave. In dipendenza dalla formalizzazione adottata, si provi poi o si dia un controesempio della seguente affermazione: Shift Encryption è un cifrario perfetto.

    Traccia. Ci sono due modi naturali di operare tramite Shift Encryption, che corrispondono a due formalizzazioni diverse. Nella prima, $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. Stabiliamo che la chiave viene scelta nuova ad ogni messaggio, con con probabilità uniforme. E' facile vedere che questo cifrario è la versione di OTP sull'alfabeto di 26 lettere. Dunque esso è un cifrario perfetto, qualunque sia la distribuzione di probabilità sul plaintext.

    Nella seconda formalizzazione, poniamo $\mathcal{P} \subseteq \mathbb{Z}_{26}^*$; per concretezza, poniamo che $\mathcal{P}$ sia un linguaggio naturale, ad esempio l'Italiano. Poniamo poi $\mathcal{C} = \mathbb{Z}_{26}^*$ e infine $\mathcal{K} = \mathbb{Z}_{26}$. Assumiamo infine che la chiave sia scelta con probabilità uniforme. La funzione di encryption non fa altro che sommare ad un plaintext la chiave, modulo 26, lettera per

lettera. Questo cifrario non può essere perfetto, dato che $|\mathcal{K}| < |\mathcal{P}|$. Per esempio, si consideri il ciphertext $c =$`jp` (ottenibile con probabilità non nulla, per esempio a partire dal plaintext `io` e dalla chiave $k = 1$). Prendiamo ora il plaintext $m =$`tu`. E' chiaro che $0 < p(c) \neq p(m|c) = 0$: infatti l'insieme dei ciphertext ottenibili a partire da $m =$`tu`, usando una delle 26 chiavi, è {`tu,uv,vw,...,st`}, ed esso non contiene c.

14. Si discuta la seguente affermazione: in un cifrario di Feistel, per ogni blocco di bit B e chiave K, vale che $E_K[D_K[B]] = B$.

    Traccia. Decifrare B con K equivale a cifrare B usando l'ordinamento delle sottochiavi $K_n, ..., K_1$.

15. Si vedano l'ingresso e l'uscita di una S-Box come due vettori di variabili aleatorie, diciamo $X = (X_1, ..., X_m)$ e $Y = (Y_1, ..., Y_n)$, dove X_i e Y_j rappresentano bit casuali. In altre parole, $Y = \mathrm{S-Box}(X)$. Denotatiamo con $X^{(i)}$ il vettore aleatorio ottenuto da X invertendo solo l'i-mo bit (cioè $X^{(i)} = (X_1, ..., \overline{X}_i, ..., X_m)$) e con $Y^{(i)}$ l'uscita corrispondente della S-box. Infine denotiamo la probabilià che una certa variabile aleatoria Z assume il valore z come $\Pr(Z = z)$. Servendosi di queste notazioni, enunciare in maniera rigorosa i criteri di *Strict Avalanche* e *Bit Independence*.

    Traccia. *SAC*. Per ogni ingresso i e uscita j: $\Pr(Y_j^{(i)} \neq Y_j) = \cdots$. *BIC*. Per ogni ingresso i e uscite distinte j e k: $\Pr(Y_j^{(i)} \neq Y_j | Y_k^{(i)} \neq Y_k) = \cdots$.

16. Si consideri uno schema di Feistel in cui la funzione $F(K, R)$ è stata scelta **lineare**. In altre parole, si supponga che esista una matrice binaria M rettangolare di dimensioni opportune, tale che $F(K, R) = M \cdot [K, R]^T$, dove $[K, R]$ denota il vettore-riga ottenuto da K e R. Si intende che le operazioni del prodotto matrice-vettore vengono svolte in $\mathbb{Z}_2$: il prodotto di elementi corrisponde all'AND e la somma allo XOR.

    (a) Dire come una permutazione P che agisce su blocchi di m bit può essere descritta come una funzione lineare $P : \mathbb{Z}_2^m \to \mathbb{Z}_2^m$.

    (b) Descrivere un round di Feistel $\mathrm{Round}(K, L, R)$ come una sequenza di operazioni lineari. In particolare, descrivere la forma di una matrice T tale che $\mathrm{Round}(K, L, R) = T \cdot [K, L, R]^T$.

    (b) Dire come il singolo round può essere attaccato con un attacco known-plaintext. Generalizzare a n round.

2 Esercizi di approfondimento

2.1 Cifrari a blocchi e ridondanza

Nei cifrari a blocchi vengono cifrati blocchi di lettere plaintext di dimensione fissata $m \geq 1$, anziché singole lettere. Abbiamo detto che la cifratura a blocchi rende più difficili le tecniche di analisi delle frequenze. Infatti man mano che la dimensione q cresce, i blocchi *tipici* di plaintext, e dunque di ciphertext, tendono *rapidamente*

(a) a diventare equiprobabili;

(b) ad assumere probabilità piccole.

Entrambi questi fatti contribuiscono a rendere l'analisi delle frequenze più difficile. Infatti, ciascun blocco tende ora ad assomigliare a ciascun altro (istogramma piatto, fatto (a)), rendendone difficile l'identificazione in base alla frequenza; e ad occorrere piuttosto di rado (fatto (b)), rendendo così necessario raccogliere una grande quantità di ciphertext per poterne catalogare un certo numero. Nel seguito, cercheremo di giustificare queste osservazioni in maniera più quantitativa.

Supporremo, per semplicità, che la sequenza di lettere del plaintext venga generata mediante estrazioni i.i.d. di lettere, da un alfabeto finito A, secondo una certa distribuzione di probabilità nota $\mathbf{p}$ (si veda il punto (f) per un modello più accurato). Fisato un intero $m \geq 1$, diremo che un blocco di m lettere, $\sigma \in A^m$, è *tipico* se ogni lettera a occorre in σ un numero di volte $f_a \approx p_a q$, dove p_a è la probabilità di a come specificato da $\mathbf{p}$; ovvero se $p_a \approx f_a/m$. Ovvero, nei blocchi tipici, ogni lettera a occorre all'incirca una frazione p_a del totale di m lettere (NB: questa definizione si può rendere completamente rigorosa, ma soprassediamo su questo per il momento).

(a) Dimostrare la seguente formula per la probabilità di un blocco tipico, basata solo su m e sui p_a:

$$p(\sigma) \approx \prod_{a \in A} p_a^{m p_a}$$

Si noti che questa formula non dipende dalla disposizione particolare delle lettere all'interno del blocco, ma solo dalla loro abbondanza relativa, come espressa dalle frequenze f_a e dunque dalle p_a. Dunque i blocchi tipici sono (approssimativamente) equiprobabili.

(b) Passando ai logaritmi in base 2 nella formula di cui sopra, dimostrare che

$$p(\sigma) \approx 2^{-mH(\mathbf{p})} \tag{4}$$

dove

$$H(\mathbf{p}) \stackrel{\text{def}}{=} -\sum_{a \in A} p_a \log p_a$$

è l'*entropia di Shannon* della distribuzione $\mathbf{p}$ (NB: qui si conviene che $0 \cdot \log 0 = 0$). Poiché $H(\mathbf{p}) \geq 0$ (con $=$ solo nel caso degenere in cui una singola lettera abbia probabilità 1), questa formula dimostra che la probabilità di un blocco tipico decresce esponenzialmente, e dunque in modo molto veloce, al crescere della dimensione m del blocco. La legge esponenziale di decrescita (4) è governata unicamente dall'entropia della distribuzione del plaintext: tanto più alta è l'entropia tanto più veloce sarà la decrescita.

L'entropia di Shannon è una misura della ridondanza del linguaggio sottostante. Entropia bassa vuol dire linguaggio ridondante (comprimibile), entropia alta linguaggio poco ridondante (non comprimibile). I linguaggi naturali tendono a essere strutturati, quindi ad avere bassa entropia (alta ridondanza). Viceversa, un linguaggio completamente casuale ha entropia massima (bassa ridondanza), pari a $\log |A|$. Convenzionalmente, l'entropia viene misurata in bit.

(c) Dimostrare che, per osservare una occorrenza di un certo fissato blocco tipico σ_0, occorrerà raccogliere *in media* un numero di blocchi di ciphertext pari a

$$\approx 2^{mH(\mathbf{p})}$$

ovvero $\approx m2^{mH(\mathbf{p})}$ lettere di ciphertext individuali. Stimare numericamente quest'ultima quantità per blocchi di $m = 13$ lettere alfabetiche (codificabili in blocchi di 64 bit), estratte secondo la distribuzione tipica della lingua Inglese, v.
`http://www.oxfordmathcenter.com/drupal7/node/353`
(NB: considerare blocchi *non sovrapposti*, ovvero solo quelli che iniziano alle posizioni $1, m+1, 2m+1, ...$).

(d) Si provi che, per un qualsiasi blocco tipico o meno, la probabilità esatta di σ è

$$p(\sigma) \quad = \quad 2^{-m\left(H(\mathbf{q})+D(\mathbf{q}||\mathbf{p})\right)} \tag{5}$$

dove: posta $k = |A|$ la cardinalità dell'alfabeto, definiamo $\mathbf{q} = (q_0, ..., q_k) \stackrel{\text{def}}{=} (f_0/m, ..., f_k/m)$ la *distribuzione empirica* delle lettere nel blocco σ; e $D(\mathbf{q}||\mathbf{p}) \stackrel{\text{def}}{=} \sum_a q_a \log_2(\frac{q_a}{p_a})$ la *divergenza di Kullback-Leibler (KL)* fra $\mathbf{p}$ e $\mathbf{q}$ (NB: porre $q_a \log(q_a/p_a) = 0$ se $q_a = 0$ e $q_a \log(q_a/p_a) = +\infty$ se $q_a > 0$ e $p_a = 0$).

La divergenza di KL è una quantità, sempre non negativa, che rappresenta una sorta di "distanza" tra le distribuzioni $\mathbf{q}$ e $\mathbf{p}$. Essa vale 0 se e solo se $\mathbf{q} = \mathbf{p}$. L'equazione (5) mostra che la probabilità del blocco σ diminuisce velocemente al crescere dell'entropia della distribuzione empirica $\mathbf{q}$, e al suo allontanarsi da $\mathbf{p}$.

(e) (Scimmia di Eddington) Una scimmia batte dei tasti su una macchina da scrivere, producendo una sequenza di caratteri. Stimare il n. *medio* di battute affinché nella sequenza appaia l'opera omnia di Shakespeare, vista come un particolare blocco σ_0 di $m \approx 10^6$ caratteri. Considerare due situazioni distinte: (1) la scimmia batte i tasti a casaccio; (2) la scimmia batte i tasti secondo la distribuzione tipica della lingua inglese. Come nel punto (c), considerare blocchi non sovrapposti. Nel punto (1), porre particolare attenzione a cosa siano le distribuzioni $\mathbf{p}$ e $\mathbf{q}$ da usare nella formula (5).

(f) Un modello più accurato della generazione del testo tiene in considerazione le correlazioni tra le lettere di blocco, a differenza del modello i.i.d. non è adatto. La *Proprietà di Equiripartizione Asintotica* afferma che (sotto certe condizioni) i blocchi tipici σ hanno tutti all'incirca la stessa probabilità, $p(\sigma) \approx 2^{-mH}$, dove H è una quantità nota come *entropy per letter* e definita da $H := \lim_{N\to+\infty} \frac{1}{N} H(\mathbf{p}^N)$, dove $\mathbf{p}^N$ denota la distribuzione di probabilità sui blocchi di N lettere del linguaggio. H può dunque essere stimata come $H \approx \frac{1}{N} H(\mathbf{p}^N)$ per N grande. Una stima per la lingua inglese è $H = 1.5$ bit per lettera. Usando questa stima, si ricalcoli la quantità richiesta al punto (c).

2.2 Indici di coincidenza

In questo esercizio vengono approfonditi alcuni aspetti matematici degli indici di coincidenza. Per una sequenza di n caratteri alfabetici $\mathbf{x} = (x_1, ..., x_n)$, poniamo per ogni carattere $i \in \mathbb{Z}_{26}$

$$f_i = \text{ numero di occorrenze del carattere } i \text{ in } \mathbf{x}.$$

Possiamo vedere sia $\mathbf{x}$ che f_i come variabili aleatorie. Assumiamo in particolare che tutte le variabili x_j $(1 \le j \le n)$ siano identicamente distribuite secondo una distribuzione di probabilità $\mathbf{p} = (p_1, ..., p_{26})$ sull'alfabeto (per esempio, potrebbe essere $\mathbf{p} = \mathbf{p}_E$, la distribuzione tipica della lingua Inglese).

(a) Sia $i \in \mathbb{Z}_{26}$ qualsiasi fissato. Notare che $f_i = \sum_{j=1}^{n} Y_j$, dove Y_j è la variabile Bernoulli che assume il valore 1 se $x_j = i$, 0 altrimenti. Sfruttando la linearità del valore atteso, dimostrare che $E[f_i] = np_i$.

(b) Dimostrare quindi che $E[f_i^2] = \text{var}(f_i) + E[f_i]^2$, dove $\text{var}(f_i) = E[f_i^2] - E[f_i]^2$ è la varianza di f_i.

(c) Assumere ora che i caratteri x_j siano estratti in maniera i.i.d. Notare che f_i è allora una distribuzione binomiale (somma delle n Bernoulli $Y_1, ..., Y_n$). Sfruttando la formula per la varianza della distribuzione binomiale e il punto precedente, dimostrare che $E[f_i^2] = np_i - np_i^2 + n^2p_i^2$. Concludere che

$$E[f_i(f_i - 1)] = E[f_i^2] - E[f_i] = n(n-1)p_i^2$$

(d) Ricordare che $I_c(\mathbf{x}) = \sum_{i \in \mathbb{Z}_{26}} \frac{f_i}{n} \frac{f_i - 1}{n-1}$. Sfruttare il risultato del punto precedente per dare una formula esatta di $E[I_c(\mathbf{x})]$. Provare cioè che

$$E[I_c(\mathbf{x})] = \sum_{i=0}^{25} p_i^2 \,.$$

(e) Si consideri la seconda fase dell'attacco a Vigenère, volta a determinare la chiave lettera per lettera. Questa fase implica, dato un certo vettore di probabilità empiriche, $\mathbf{q}_0 = (f_0/n, ..., f_{25}/n)$, trovare il suo shift circolare di k posizioni ($0 \leq k \leq 25$), diciamo $\mathbf{q}$, che meglio approssima il vettore delle probabilità caratteristico della lingua Inglese, $\mathbf{p}$. In altre parole, cerchiamo quel particolare shift circolare $\mathbf{q}$ di $\mathbf{q}_0$ tale che vale

$$\mathbf{p} \approx \mathbf{q}.$$

Trattando per semplicità l'approssimazione di cui sopra come una vera uguaglianza, dimostrare che, tra tutti i 26 shift possibili, questo è il vettore $\mathbf{q}$ che massimizza il prodotto scalare $< \mathbf{p}, \mathbf{q} >$. Allo scopo, servirsi della disuguaglianza di Cauchy-Schwarz, che afferma che, dati due qualsiasi vettori $\mathbf{v}$ e $\mathbf{u}$

$$| < \mathbf{v}, \mathbf{u} > | \leq ||\mathbf{u}||_2 \cdot ||\mathbf{v}||_2 \,.$$

2.3 Un crittogramma Vigenère

Si decifri il seguente testo cifrato, ottenuto tramite un cifrario di Vigenère a partire da un plaintext in lingua Inglese.

```
OKZARVGLNSLFOQRVVBPHHZAMOMEVHLBAITLZOWSXCSZFEQFICOOVDXCIISOOVXEIYWNHHLVQHSOWD
BRPTTZZOWJIYPJSAWQYNOYRDKBQKZPHHTLIHDEMICGYMSEVHKVXTQPBWMEWAZZKHLJMOVEVHJYSJR
ZTUMCVDGLZVBUIWOCPDZVEIGSOGZRGGOTAHLCSRSCXXAGPDYPSYMECRVPFHMYWZCYHKMCPVBPHYIF
WDZTGVIZEMONVYQYMCOOKDVQIMSOKLBUEBFZISWSTVFEWVIAWACCGHDRVVZOOBANRYHSSQBUIMSDW
VBNRXSSOGLVWKSCGHLNRYHSSQLVIYCFHVWJMOVEKRKBQMOOSVQAHDGLGWMEOMCYOXMEEIRTZBCFLZ
BVCVPRQVBLUHLGSBSEOUWHRYHSSEIEVDSCGHZRGOSOPBBUIQWNHRZFEIRPBWMEXCSPASBLXZFCWWW
EMZGLDDBUIOWNTHVPICOOTRZOMYRPBKMEIIHCOQKRWCSNFRAFIYWEKLBUSPHEVHAYMBVESVBGVZAZ
FVPRAJIWRQMIIMUZPDKXXJHSSRBUIMGTRHBUIMSHCXTQFZBZFHBHVIOYRWPRXCFPSRNGLZAVBHEVX
OVPMZMEIAIWZBIJEMSEVDBGLZMHSUMGVVWWWQOGLZCCPLARWYSNZLVRXCOEHKMLAZFPGLVXMIUHWW
PVXDBECWPRJDBLZQQTLOALFHBUIKOEVZWHPYPPRLNSMXIWHWPNXOCZHKMLOISH
```

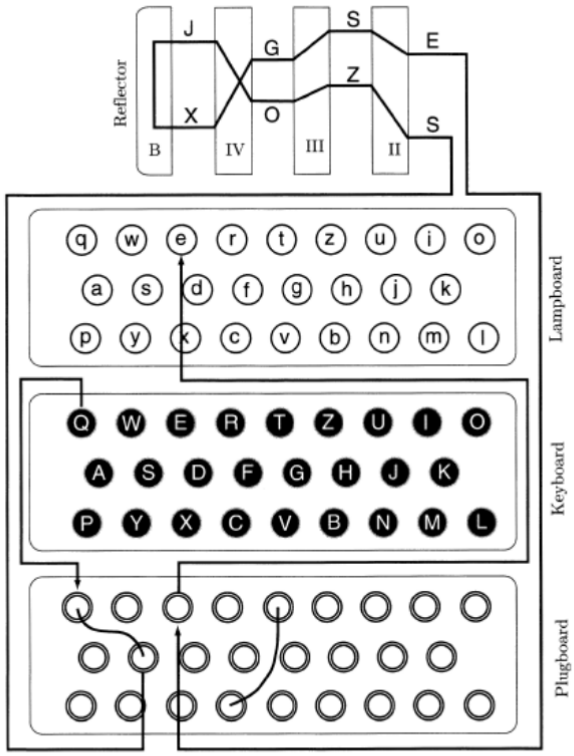
Si impieghi il metodo degli indici di coincidenza, eventualmente accoppiato con il metodo di Kasiski. Illustrare i vari passaggi, in particolare:

1. le ripetizioni nel testo, le loro distanze e i valori di m (lunghezza della chiave) da esse suggeriti;
2. i valori degli indici di coincidenza che si ottengono per il valore corretto di m;
3. per ciascuna delle m lettere della chiave, come è stato individuato lo shift che dà il valore della chiave.

Si può far uso di strumenti disponibili online per il computo delle statistiche rilevanti del testo (ripetizioni, frequenze, etc.), come per esempio quelli offerti dalla pagina *Black Chamber* di Simon Singh:
`http://www.simonsingh.net/The_Black_Chamber/vigenere_cracking_tool.html`.

2.4 Enigma

Lo schema della macchina Enigma in uso all'esercito tedesco durante la II guerra mondiale è illustrato nella figura seguente, tratta da [1]. Gli elementi elettromeccanici essenziali sono i seguenti.



- una *tastiera (keyboard)* alfabetica, attraverso la quale viene immesso l'input (plaintext da cifrare, o ciphertext da decifrare), una lettera alla volta;
- una *plugboard*, dove un insieme di connessioni realizzano lo scambio tra alcune (tipicamente 10) coppie di lettere. Ad esempio, `Q` $\mapsto$ `S` e `S` $\mapsto$ `Q`; in tal caso si dice che `Q` è *steckered to* `S`, e viceversa;
- tre *rotori*, connessi a cascata. Ciascun rotore implementa una fissata permutazione, tratta da un set noto di possibili permutazioni. Un rotore può essere fatto avanzare, rispetto a una posizione 0 convenzionale, di un certo numero i di posizioni, $0 \leq i \leq 25$. Il rotore spostato di i posizioni corrisponde alla permutazione ottenuta spostando in avanti circolarmente di i posizioni ogni lettera prima di farla passare attraverso la permutazione relativa alla posizione 0;
- un *riflettore*, che realizza uno scambio fissato tra 13 coppie di lettere (quindi ogni lettera subisce uno scambio quando passa dal riflettore);
- una *lampboard*, sulla quale viene visualizzato l'output (ciphertext o plaintext ottenuti), una lettera alla volta.

Quando una lettera è immessa mediante la tastiera, essa attraversa i vari elementi secondo una sequenza predeterminata, per arrivare alla lampboard (le permutazioni corrispondenti sono indicate nel seguito con lettere greche):

- plugboard (σ), rotore 1 (α), rotore 2 (β), rotore 3 (γ)
- riflettore (π)
- rotore 3 all'indietro (γ^{-1}), rotore 2 all'indietro (β^{-1}), rotore 1 all'indietro (α^{-1}), plugboard all'indietro (σ^{-1}).

Matematicamente, le permutazioni σ (plugboard) e π (riflettore) sono definite in modo da essere delle *involuzioni*: una doppia applicazione ne annulla l'effetto. Per cui, ad es. $\sigma(\sigma(x)) = x$ per ogni lettera x; dunque $\sigma^{-1} = \sigma$. Inoltre σ (plugboard) ha esattamente *sei punti fissi* (cioè lettere x tali che $\sigma(x) = x$), mentre π non ha *nessun* punto fisso. Dunque, ad ogni istante, la permutazione ρ realizzata dalla macchina può essere descritta come

$$\rho \quad = \quad \sigma \circ \tau \circ \sigma \qquad \text{dove } \tau = \alpha^{-1} \circ \beta^{-1} \circ \gamma^{-1} \circ \pi \circ \gamma \circ \beta \circ \alpha \,.$$

Si noti che τ corrisponde alla trasformazione realizzata da una plugboard vuota. Ad ogni immissione di una lettera, la disposizione dei rotori cambia, secondo la seguente regola: il rotore 1 avanza di una posizione ad ogni lettera, il rotore due avanza di una posizione ogni 26 lettere, il rotore 3 avanza di una posizione ogni 26^2 lettere[2]. Ad ogni istante, la disposizione dei tre rotori (loro identità e avanzamento di ciascuno rispetto alla posizione 0 di riferimento) costituisce lo *stato* della macchina. Poiché la permutazione realizzata cambia ad ogni lettera, Enigma è un cifrario polialfabetico. In particolare, a partire da un dato stato iniziale, man mano che vengono immesse delle lettere in input, Enigma realizzerà una sequenza di permutazioni

$$\rho_1, \rho_2, \rho_3, ...$$

dove

$$\rho_j = \sigma \circ \tau_j \circ \sigma \,.$$

Si noti che σ rimane fissa per tutta la sequenza, mentre τ_j è la permutazione realizzata nello stato j-mo dei rotori con plugboard *vuota* (nessuna lettera steckered).

Sia i collegamenti interni dei rotori che quello del riflettore, che la struttura della macchina erano noti ai criptanalisti, i quali anzi avevano fabbricato numerose repliche della stessa. Pertanto una *chiave* Enigma consiste dei seguenti elementi

1. una tripla (R_1, R_2, R_3), con $R_1, R_2, R_3 \in S$, dove S è un insieme prefissato di 5 rotori, ciascuno dei quali realizza una permutazione *nota*;
2. una tripla $(i, j, k) \in \mathbb{Z}_{26}^3$, che specifica la posizione iniziale di ciascuno dei tre rotori, con 26 posizioni possibili per ciascun rotore;
3. una permutazione σ dell'alfabeto che specifica le connessioni della plugboard, che sia un'involuzione con sei punti fissi.

I primi due elementi costituiscono lo *stato iniziale* della macchina. La chiave è dunque costituita da due elementi:

[2]Questa è una descrizione leggermente semplificata dell'avanzamento dei rotori. Tale semplificazione non ha comunque alcuna influenza sulla tecnica di attacco che esporremo.

- lo **stato iniziale** dei rotori, dato da (R_1, R_2, R_3) insieme a (i, j, k);
- le **connessioni della plugboard** σ.

Il numero complessivo di chiavi è tale da rendere impraticabile un semplice attacco forza bruta (si veda il punto (a) più sotto). Tuttavia, vedremo che la criptanalisi può prendere di mira e determinare *separatamente* lo stato iniziale dei rotori e la plugboard. Questa possibilità rappresenta la principale debolezza di Enigma. Nel seguito esporremo le idee matematiche essenziali dell'attacco, se non i dettagli.

(a) Dare una formula per il numero di possibili chiavi Enigma. Dare poi un'ordine di grandezza, come multiplo di una potenza di 10, di tale numero e dire a quanti bit corrisponde. Calcolare il numero esatto di possibili stati iniziali.

(b) Dimostrare che, detta ρ la permutazione realizzata dalla macchina in uno stato qualsiasi, ρ è una involuzione e non ha punti fissi. Dire come può essere effettuata la decryption.

L'attacco a Enigma condotto dai criptanalisti britannici a Bletchley Park era di tipo known plaintext. Una parola la cui occorrenza era ritenuta molto probabile all'interno di un messaggio di plaintext era detta *crib*. Dato che il linguaggio militare era altamente stereotipato, l'esistenza di crib in certi messaggi non era difficile da congetturare. Per esempio, i bollettini meteo diramati ogni giorno alle ore 6.05 contenevano invariabilmente la parola `WETTER` (*tempo*). Il criptanalista doveva però stabilire in quale posizione, all'interno del ciphertext intercettato, si collocava esattamente il crib. In questo compito egli era aiutato dal fatto che, in qualsiasi stato, la permutazione ρ non ha punti fissi. Cioè, per qualsiasi lettera x del testo in chiaro, la lettera di ciphertext corrispondente è diversa: $\rho(x) \neq x$. Questo gli permetteva di scartare diverse posizioni che non risultavano compatibili.

(c) Un crib usato per decifrare i messaggi del D-Day è il frammento di plaintext `WETTERVORHERSAGEBISKAYA` (*previsioni del tempo Biscaglia*). Si confronti questo con il frammento di ciphertext intercettato

`QFZWRWIVTYRESXBFOGKUHQBAISEZ`

e si determini una possibile coppia (plainetxt, ciphertext).

Una volta individuato una coppia (plainetxt, ciphertext), si possono stabilire una serie di relazioni accoppiando singole lettere di plaintext e ciphertext, in stati successivi numerati convenzionalmente 1,2,..., come segue:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
W E T T E R V O R H  E  R  S  A  G  E  B  I  S  K  A  Y  A
R W I V T Y R E S X  B  F  O  G  K  U  H  Q  B  A  I  S  E
```

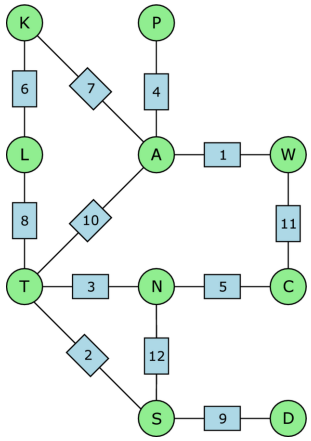
Questa tabella va letta così:

- nello stato 1 (iniziale), `W` viene transformata in `R`;
- nello stato 2, `E` viene transformata in `W`;
- ...

Queste relazioni possono essere rappresentate da un particolare grafo detto *menu*: i nodi sono lettere ed esiste un arco etichettato $x \overset{j}{\longleftrightarrow} y$ solo se $\rho_j(x) = y$ è una relazione deducibile dalla colonna j della tabella; qui ρ_j è la permutazione corrispondente allo stato j-mo della macchina. Si noti che il gli archi sono bidirezionali, perché le permutazioni Enigma sono involuzioni ($\rho(x) = y$ sse $\rho(y) = x$). Per esempio, con la seguente coppia plaintext-ciphertext

1	2	3	4	5	6	7	8	9	10	11	12
A	T	T	A	C	K	A	T	D	A	W	N
W	S	N	P	N	L	K	L	S	T	C	S

si può formare il seguente menu:



(d) Si disegni il menu relativo alla coppia crib-ciphertext per il crib `WETTERVORHERSAGEBISKAYA` dato prima. Se ne individuino tutti i cicli.

Dato un qualsiasi stato iniziale, definiamo il *grafo di Turing* relativo a tale stato iniziale come il grafo in cui esiste un arco $x \overset{j}{\longleftrightarrow} y$ se e solo se $\tau_j(x) = y$, per $j = 1, ..., N$, con N fissato. In pratica, N è preso uguale alla lunghezza del crib. Qui τ_j denota la permutazione relativa allo stato j-mo a partire dallo stato iniziale 1 dato, impiegando una plugboard *vuota*.

(e) Dimostrare che il menu del crib è contenuto come sottografo nel grafo di Turing relativo allo stato iniziale corretto della chiave, a patto di ridenominare i nodi del secondo grafo con σ. Ovvero

se $x \overset{j}{\longleftrightarrow} y$ nel menu allora $\sigma(x) \overset{j}{\longleftrightarrow} \sigma(y)$ nel grafo di Turing.

In altre parole, σ stabilisce un isomorfismo tra il menu ed un sottografo del grafo di Turing relativo allo stato iniziale della chiave.

In particolare, se il menu contiene un ciclo, diciamo $x \overset{15}{\longleftrightarrow} y \overset{20}{\longleftrightarrow} z \overset{14}{\longleftrightarrow} x$, un ciclo isomorfo a questo dovrà esistere anche nel grafo di Turing relativo allo stato iniziale della chiave. Se abbiamo un candidato allo stato iniziale, e il ciclo non è presente nel relativo grafo, quello stato candidato può essere scartato. Questo vale per tutti i cicli presenti nel menu. Dunque il menu pone delle condizioni ai possibili candidati stati iniziali della chiave, tanto più stringenti quanto maggiore è il numero dei cicli e delle lettere in essi coinvolti. Anche *senza conoscere la configurazione della plugboard* σ, il criptanalista può dunque determinare lo stato iniziale della chiave (o almeno, un insieme molto ristretto di candidati): è sufficiente

che passi in rassegna tutti i possibili stati dei rotori, e selezioni quei pochi che soddisfano il menu, nel senso precisato sopra. Quelli, cioè, che soddisfano l'esistenza di cicli isomorfi a quelli del menu. Questo era essenzialmente il compito svolto, in maniera automatica, dalle famose *bombe* di Turing. Ciascuna bomba impiegava numerose repliche di Enigma, collegate in serie e in parallelo. Una bomba passava in rassegna un insieme di possibili stati iniziali, e controllava, per ciascuno, il soddisfacimento simultaneo di tutte le condizioni (cicli) imposti dal menu. Diverse bombe potevano lavorare in parallelo per esplorare parti diverse dello spazio degli stati iniziali. Si noti che il numero di stati iniziali, per quanto alto in termini assoluti, è di molti ordini di grandezza inferiore al numero di possibili chiavi e alla portata di un metodo di ricerca esaustivo automatizzato (v. punto (a)).

(f) Si descriva un modo di connettere opportunamente tre macchine Enigma in cascata per controllare l'esistenza di un ciclo del tipo $x_1 \overset{15}{\longleftrightarrow} x_2 \overset{20}{\longleftrightarrow} x_3 \overset{14}{\longleftrightarrow} x_1$ in un grafo di Turing relativo ad un qualsiasi stato iniziale.

Una volta che lo stato iniziale della chiave è stato individuato, una decifrazione parziale del messaggio è già possibile impiegando una plugboard vuota. In particolare, la decifrazione sarà corretta in corrispondenza di quelle posizioni del ciphertext dove appaiono lettere non *steckered*. Le connessioni della plugboard si possono ottenere a questo punto lavorando manualmente per tentativi ed errori e servendosi di varie euristiche, e/o impiegando semplici tecniche di analisi delle frequenze. Preziose informazioni sono anche fornite dal menu del crib. Infatti, è possibile *ipotizzare* uno steckering tra due lettere e verificare se l'ipotesi è compatibile con il menu e lo stato iniziale trovato, oppure può essere scartata.

(g) Supponiamo che per una certa lettera x valga $\sigma(x) = y$. Dimostrare che per ogni ciclo del menu contenente x, diciamo

$$x = x_0 \overset{j_1}{\longleftrightarrow} x_1 \overset{j_2}{\longleftrightarrow} \cdots \overset{j_k}{\longleftrightarrow} x_k = x$$

deve valere

$$y \quad = \quad \tau_{j_1} \circ \cdots \circ \tau_{j_k}(y)$$

e dunque nel grafo di Turing relativo allo stato iniziale corretto con plugboard libera deve esistere un ciclo che coinvolge y del tipo

$$y \overset{j_1}{\longleftrightarrow} \cdots \overset{j_k}{\longleftrightarrow} y\,.$$

Spiegare come l'esistenza di questo percorso può essere controllata collegando in cascata opportunamente k macchine Enigma.

2.5 Enigma, Banburismus e indici di coincidenza

Una fase dell'attacco all'Enigma navale (che utilizzava quattro rotori invece che tre) consisteva nell'individuare l'identità dei due rotori più esterni in uso in un dato giorno. Tale informazione, che è un elemento della chiave, infatti contribuiva a ridurre il numero configurazioni iniziali da testare tramite le bombe. Allo scopo veniva applicata una tecnica denominata *Banburismus*, sviluppata da A. M. Turing a Bletchley Park. Senza entrare nei dettagli, diremo solo che tale tecnica presupponeva di confrontare due sequenze di ciphertext intercettati, per stabilire se essi fossero o meno stati prodotti da due macchine Enigma con la stessa configurazione della

plugboard e rotori *allineati*, cioè nella stessa configurazione iniziale[3]. Chiameremo la prima ipotesi (macchine allineate) H_1 e la seconda (macchine non allineate) H_0.

Chiamiamo $\mathbf{x}, \mathbf{y}$ i due ciphertext, che supporremo della stessa lunghezza N. Il confronto prevede di scrivere i due ciphertext allineati su due righe, e contare il numero R di posizioni in cui in $\mathbf{x}$ e in $\mathbf{y}$ mostrano lo stesso carattere. Tali coincidenze sono dette *repeats*. Per esempio, nel seguente caso

```
VWBDJLKWIPEHVYGQZWDTHRQXIKEESQSSPZXARIXEABQIRUCKHGWUEBPF
YNSCFCCPVIPEMSGIZWFLHESCIYSPVRXMCFQAXVXDVUQILBJUABNLKMKD
              - --  -   -          -  -   --
```

abbiamo $R = 9$ repeats su $N = 57$ caratteri. L'idea di base per rilevare l'allineamento è che il numero di repeats dovrebbe essere più alto in caso di allineamento che in caso di non allineamento. Idealizzando un po' il funzionamento di Enigma, supporremo che se vale H_0, allora le sequenze $\mathbf{x}, \mathbf{y}$ non sono correlate, e a tutti gli effetti possono essere viste come due sequenze indipendenti di caratteri casuali estratti ciascuno con prob. $1/26$. Viceversa, se vale H_1 le sequenze $\mathbf{x}, \mathbf{y}$ saranno correlate: in particolare, se nei plaintext relativi alle due sequenze appaiono due caratteri uguali nella stessa posizione, essi risulteranno cifrati con lo stesso carattere nelle due sequenze di ciphertext, dando origine ad un repeat. Infine, considereremo la generazione del plaintext come una estrazione di caratteri i.i.d., secondo la distribuzione di probabilità $\mathbf{p}$ tipica della lingua data (es. Tedesco).

(a) Dimostrare che il valore atteso di R è Nm, dove $m = m_0 = 1/26$ sotto H_0, mentre sotto H_1 $m = m_1 = \sum_{i=0}^{25} p_i^2$ è il valore atteso dell'indice di coincidenza nella lingua del plaintext; m viene detto *matching probability*.

(b) Dare una formula, che dipenda da m, per la probabilità che ci siano esattamente k repeats, per $0 \leq k \leq N$. Cioè una formula per

$$\Pr(R = k \,|\, H_i) \quad \text{per } i = 0, 1.$$

(c) Una volta contati i repeats, diciamo k, le ipotesi H_0 e H_1 venivano confrontate calcolando la *log-evidence in favor of* H_1:

$$L(k) \stackrel{\text{def}}{=} \log_{10} \frac{\Pr(R = k \,|\, H_1)}{\Pr(R = k \,|\, H_0)}\,.$$

In sostanza, ignorando il ruolo del log che viene introdotto per convenienza numerica (evitare di dover lavorare con numeri molto piccoli), $L(k)$ dice quante volte il valore di repeats osservato è più probabile sotto l'ipotesi H_1 (macchine allineate) invece che sotto l'ipotesi H_0 (macchine non allineate). Se $L(k)$ è positiva, essa fornisce evidenza a favore di H_1, se negativa a favore di H_0, se nulla, nessuna evidenza. L'unità di misura della log-evidence veniva detta *ban*. Spesso venivano usati i suoi sottomultipli, come il *deciban* (un decimo di ban). Dare una formula esplicita per $L(k)$, a partire dalla formula del punto (b).

(d) Dare una formula per il valore atteso di $L(R)$, sotto H_0 e sotto H_1, sfruttando il punto (a).

[3] Più in generale, con configurazioni iniziali ad una distanza prefissata l'una dall'altra, in termini di scatti

(e) Sfruttando il punto precedente, e assumendo i valori $m_0 = 1/26$ e $m_1 = 1/13$ (valore approssimato per l'indice di coincidenza del Tedesco tipico dell'Enigma navale), stimare numericamente quanti caratteri N di ciphertext occorrono *in media* per ottenere una log-evidence di ± 20 deciban in favore di H_1 (NB: una $L(k)$ di 20 deciban corrisponde ad una probabilità di 100 contro 1 in favore di H_1).

Per ulteriori dettagli su Banburismus e dintorni, si può consultare il libro di J.C. MacKay *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 4/E, 2005.

3 Esercizi di programmazione

Per gli esercizi di seguito proposti, si intende che la soluzione deve essere programmata nel linguaggio Python.
NB: gli esercizi di programmazione svolti e consegnati ai fini dell'esonero DEVONO consistere di DUE parti: (1) una parte scritta, che risponde in maniera sintetica ma esauriente a tutte le domande poste nel testo dell'esercizio e illustra le tecniche e gli algoritmi usati; (2) il codice vero e proprio, corredato di istruzioni per l'uso e commenti, che va inviato via email al docente.

3.1 Analisi delle frequenze di un testo

Scrivere una funzione che calcoli le seguenti caratteristiche di un testo (sequenza di caratteri). Si faccia riferimento all'Esercizio 2.1 per la definizione dei concetti teorici.

1. Istogramma della frequenza delle 26 lettere;
2. Dato $m \geq 1$ in input, distribuzione empirica degli m-grammi (blocchi di m lettere);
3. Dato $m \geq 1$, indice di coincidenza ed entropia della distribuzione degli m-grammi.

Illustrare i risultati ottenuti impiegando la funzione sul primo capitolo di *Moby Dick* (H. Melville, 1851), per $m = 1, 2, 3, 4$.

3.2 Cifrario di Hill

Programmare una funzione che permetta all'utente di cifrare e decifrare con il cifrario di Hill, e di forzare un ciphertext tramite l'attacco known plaintext.

3.3 SAC e BIC per le S-Box

Sia data una S-Box, vista come funzione $S : \{0,1\}^m \to \{0,1\}^n$. Considerare lo Strict Avalanche Criterion (SAC) per le S-Box: riferendosi alla notazione introdotta nell'Esercizio 14 della Sezione 1, si denoti X l'input della S-Box, con $Y = S(X)$ l'output, con $Y^{(i)} = S(X^{(i)})$ l'output quando l'i-mo bit di input di X è invertito, e con $a_{ij} \stackrel{\text{def}}{=} \Pr(Y_j^{(i)} \neq Y_j)$ la probabilità che invertendo il bit i-mo dell'ingresso, il bit j-mo dell'uscita sia anche lui invertito.

1. Porre $\Delta_{ij} \stackrel{\text{def}}{=} \{x \in \{0,1\}^m : S(x^{(i)})_j \neq S(x)_j\}$. Dimostrare che $a_{ij} = \frac{|\Delta_{ij}|}{2^m}$.
2. Programmare una funzione Python che calcoli a_{ij} per una S-Box di DES, per $1 \leq i \leq 6$ e $1 \leq j \leq 4$. Qui si intende che 1 è l'indice del bit più a sinistra (più significativo) di un blocco di bit.

3. Tabulare tutti i valori a_{ij} per per la S-Box S_1 di DES. Individuare per quali valori di i, j lo scostamento $|\frac{1}{2} - a_{ij}|$ è massimo.

4. Ripetere gli stessi passaggi per la proprietà *Bit Independence Criterion (BIC)*, che considera la probabilità b_{ijk} che, invertendo l'input i-mo, si invertano gli output j-mo e k-mo ($j \neq k$). In questo caso, bisognerà considerare anche insiemi $\Gamma_{ijk} \stackrel{\text{def}}{=} \{x \in \{0,1\}^m : S(x^{(i)})_j \neq S(x)_j \text{ e } S(x^{(i)})_k \neq S(x)_k\}$. Stimare gli scostamenti $|a_{ij} \cdot a_{ik} - b_{ijk}|$ per ogni i, j, k.

Riferimenti

[1] David Salomon. *Data Privacy and Security*, Springer, 2003.