Crittografia

Federico Matteoni

A.A. 2020/21

Indice

Introduzione

Prof.ssa: Anna Bernasconi.
Vedremo i cifrari da un punto di vista prettamente algoritmico. Vedremo anche i cifrari storici, ormai non più utilizzabili, perché hanno "aperto la strada", per poi passare ai cifrari perfetti (soluzione ideale ma con costo elevato).
Poi esamineremo i cifrari simmetrici, a chiave pubblica, curve ellittiche, firma digitale, SSL. Protocolli zero knowledge, blockchain e crittografia quantistica.

Libro di testo: Bernasconi, Ferragina, Luccio - Elementi di Crittografia.

Esame Orali nel caso di esami a distanza, scritto nel caso di esami in presenza, closed-book.

Capitolo 1

Introduzione alla Crittografia

1.1 Introduzione

Crittografia Significa "*scrittura nascosta*", si intendono tecniche matematiche per mascherare i messaggi per non renderli leggibili a terzi (**crittografia**) o tentare di svelarli quando non si è il legittimo destinatario (**crittoanalisi**). Quindi tecniche di protezione e viceversa.
Esiste per i due mondi in contrapposizione: persone che vogliono scambiarsi privatamente informazioni e gli *impiccioni* che desiderano ascoltare o intromettersi nelle conversazioni altrui (per curiosità, investigazione o altri scopi).

Due gruppi di persone Chi vuole proteggersi userà **metodi di cifratura**, gli altri useranno **metodi di crittoanalisi**

Crittografia Metodi di Cifratura

Crittoanalisi Metodi di . . . **crittologia** studio comunicazione canali non sicuri e relativi problemi

1.1.1 Lo scenario

Alice vuole comunicare con Bob su un canale insicuro, quindi adottano un metodo di cifratura per spedire il messaggio in chiaro `m` sottoforma di crittogramma `c` (testo cifrato) che deve essere: incomprensibile al crittoanalista Eve (eavesdropper) in ascolto sul canale, ma facilmente decifrabile da Bob.

MSG Insieme dei messaggi in chiaro

CRITTO Insieme dei crittogrammi

$$C : \text{MSG} \to \text{CRITTO}$$
$$D : \text{CRITTO} \to \text{MSG}$$

Sono operazioni da poter fare in tempo polinomiale. C e D sono una l'inversa dell'altra, ma C **deve essere iniettiva**.

1.1.2 Antichi esempi

Erodoto Nelle *Storie*, V secolo a.C.
Messaggi tatuati sulla testa, coperti dai capelli e riscoperti rasando la testa.

Scitale Spartani. Asta cilindrica in due esemplari identici. Si avvolgeva una striscia di carta attorno al cilindro e scritta. La chiave del cifrario è il diametro dello scitale.

Enea Tattico Un libro qualsiasi con un insieme di lettere segnate, o sostituire le vocali con simboli grafici.

Cifrario di Cesare Il più antico cifrario di concezione moderna. L'idea di base è che il crittogramma è ottenuto dal messaggio in chiaro `m` sostituendo ogni lettera con quella di tre posizioni più avanti nell'alfabeto.
Es. A $\to$ D, Z $\to$ C. La segretezza dipende interamente dalla conoscenza del metodo, era destinato all'uso ristretto da un piccolo gruppo di persone.

1.2 Livello di segretezza

Classificazione in base al livello di segretezza

> Cifrari per **uso ristretto**
> Le tecniche con cui si calcola e decifra il crittogramma sono tenute segrete in ogni loro aspetto. Impiegati per comunicazioni classificate (diplomatiche o militari), non adatti per uso di massa.
>
> Cifrari per **uso generale**
> Ogni codice segreto non può essere mantenuto tale per troppo a lungo. La parte segreta si limita alla chiave, nota solamente agli utenti che stanno comunicando.
> Vengono studiati dalla comunità, coinvolgendo tantissime persone. Solo la chiave deve essere segreta.
> **Il nemico conosce il sistema**.
> Quindi C e D sono note, la chiave **segreta** `k` è usata come input sia in C che in D:
> `c = C(m, k)`, `m = D(c, k)`
> Se non si conosce `k`, anche conoscendo C e D non si possono estrarre informazioni dal crittogramma.
> Tenere segreta una sola chiave è più facile che segretare l'intero metodo. Tutti possono usare C e D pubbliche con chiavi diverse, e se un crittoanalista entra in possesso di una chiave posso generarne semplicemente una nuova.

1.2.1 Chiavi segreta

Se la segretezza dipende unicamente dalla chiave bisogna proteggersi dagli attacchi a forza bruta, quindi avere un gran numero di chiavi, così da essere immuni da chi le prova tutte.
Inoltre la chiave deve essere scelta in modo casuale e non prevedibile, sennò il crittoanalista può provare le chiavi ovvie.

Attacco esauriente Il crittoanalisa potrebbe sferrare un attacco a forza bruta verificando la significatività delle sequenze `D(c, k)` $\forall$ `k`.
Se $|\text{Key}| = 10^{20}$ e con un calcolatore che impiega 10^{-6} per calcolare `D(c, k)` servirebbe in media più di un milione di anni per scoprire il messaggi provando tutte le chiavi. Però la segretezza può essere violata con altre tecniche: esistono cifrari più sicuri di altri pur con uno spazio di chiavi più piccoli.
Un cifrario complicato non è necessariamente più sicuro e **mai sottovalutare la bravura del crittoanalista**.

1.2.2 Crittoanalista

Comportamento Il comportamento di un crittoanalista può essere:

Passivo, quando si **limita ad ascoltare** la comunicazione

Attivo, quando **agisce sul canale** disturbando la comunicazione o modificando il contenuto dei messaggi.

Attacchi a un sistema crittografico Hanno l'obiettivo di forzare un sistema. Il metodo e il livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista:

Cipher Text Attack: conosce una serie di crittogrammi

Known Plain-Text Attack: conosce una serie di coppie `(m, c)`

Chosen Plain-Text Attack: si procura coppie `(m, c)` relative a messaggi in chiaro da lui scelti.
Tutta la crittografia a chiave pubblica è soggetta a questo tipo di attacco (avendo la chiave pubblica, cifro dei messaggi che penso possano passare e ascolto finché non trovo nella comunicazione i crittogrammi in mio possesso).

Man in the Middle Il crittoanalista si installa sul canale di comunicazione:

Interrompe le comunicazioni dirette tra gli utenti Alice e Bob

le **sostituisce con messaggi propri**

e **convince** ciascun utente **che tali messaggi provengano leggitimamente dall'altro** utente.

Quindi il crittoanalista **Eve si finge Bob agli occhi di Alice e Alice agli occhi di Bob**.

Esiti

Successo pieno, si scopre completamente D o si ottiene la chiave

Successo limitato, si scopre solo qualche informazione ma sufficiente per comprendere il messaggio

1.2.3 Situazione attuale

Cifrari perfetti Inattaccabili, esistono ma richiedono operazioni complesse, **chiavi lunghe tanto quanto il messaggio e mai riutilizzabili**.
Shannon, 1945 (pubblicato nel 1949 per motivi di segretezza militare): `m` e `c` appaiono totalmente scorrelati, come se `c` fosse una stringa casuale di bit.
Nessuna informazione può filtrare dal crittogramma. Vedremo la teoria matematica.

One-Time Pad Anche detto blocco monouso, sicuro ma per essere usato bene richiede chiavi segrete totalmente casuali e lunghe quanto il messaggio. Come generarla e come scambiarla?

Cifrari attuali Nella crittografia di massa non si usano cifrari perfetti, ma **cifrari *dichiarati* sicuri**, inviolati dagli esperti e che usano algoritmi solo esponenziali per decrittare senza chiave. Il tempo per violare un cifrario è enorme e rende l'operazione insostenibile → impossibilità *pratica* di forzare il cifrario.

Dichiarati sicuri Non è noto se questi problemi matematici richiedano algoritmi *necessariamente* esponenziali o se sono dovuti all'incapacità nostra di trovare metodi più efficienti. Si riconduce a $P = NP$

1.2.4 Cifrari odierni

Advanced Encryption Standard AES, simmetrico a blocchi con chiavi di 128-256bit, pubblicamente noto e realizzabile su computer di ogni tipo. Il messaggio è diviso a blocchi lunghi quanto la chiave.

Le chiavi Sono stabilite dai mezzi elettronici (PC, smartphone, terminale...) e su Internet si scambia una chiave per ogni sessione.

Scambio delle chiavi La chiave va comunicata in sicurezza su un canale non ancora sicuro. Un'intercettazione nello scambio della chiave compromette il sistema.
Nel 1976 viene proposto un algoritmo per generare e scambiare una chiave segreta su un canale insicuro, senza necessità di scambiare informazioni o di incontrarsi in precedenza.
Si chiama **protocollo DH**, ancora largamente utilizzato nei protocolli crittografici su Internet.
Si scambiano pezzi di chiave tramite la rete e unendole a informazioni locali si costruisce la chiave.

Chiave pubblica Diffie ed Hellman hanno anche proposto la crittografia a chiave pubblica.

Cifrari simmetrici: stessa chiave per cifrare e decifrare, nota solo ai due utenti che comunicano. La scelgono di comune accordo e la tengono segreta.

Cifrari asimmetrici: chiavi pubbliche usate per cifrare e chiavi private per decifrare.
`c = C(m, k`$_{pub}$`)`
`m = D(m, k`$_{prv}$`)`
Si rende necessario che la C sia una one-way trapdoor: calcolare il crittogramma deve essere facile (polinomiale), ma decifrare `c` deve essere computazionalmente difficile (a meno di conoscere la trapdoor, la chiave privata).

RSA Rivest, Shamir, Adleman, 1977. Propongono un sistema a chiave pubblico facile da calcolare e difficile da invertire.

Vantaggi

Comunicazione molti a uno
Tutti possono inviare in modo sicuro allo stesso destinatario usando la sua chiave pubblica, ma solo lui può decifrarli. Un crittoanalista non può decifrare anche se conosce C, D e k_{pub}

Se n utenti vogliono comunicare servono solo $2n$ chiavi invece delle $n(n-1)/2$ necessarie nei cifrari simmetrici (una coppia per ogni coppia di utenti)

Non è richiesto nessun scambio

Svantaggi

Sono molto lenti rispetto ai cifrari simmetrici (polinomi di terzo grado)

Sono esposti ad attacchi di tipo chosen plain-text, perché conosco la chiave pubblica
Scelgo un numero qualsiasi di messaggi in chiaro, costruisce i crittogrammi relativi e ascolta sul canale confrontando i crittogrammi in transito e se trova un riscontro sa esattamente qual è il messaggio passato.

Come si usa Oggi si usa un cifrario a chiave segreta (AES) per le comunicazioni di massa, e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo senza incontri fisici tra gli utenti.
Diventa lento solo lo scambio delle chiavi. Siamo anche al sicuro da attacchi chosen plain-text perché se la chiave è scelta bene risulta imprevedibile dal crittoanalista.

1.3 Rappresentazione matematica di oggetti

Per rappresentare gli oggetti scegliamo dei **caratteri** da un **insieme finito** detto **alfabeto**.
Un **oggetto** è **rappresentato da una sequenza ordinata di caratteri dell'alfabeto**. L'ordine dei caratteri è importante: a **oggetti diversi corrispondono sequenze diverse** e **il numero di oggetti che si possono rappresentare non ha limiti**. Significa che fissando un numero n arbitrariamente grande possiamo sempre creare un numero di oggetti $> n$, con sequenze via via più grande.

Alfabeto Γ con $|\Gamma| = s$ e N oggetti da rappresentare.
$d(s, N)$: lunghezza della sequenza più lunga che rappresenta un oggetto dell'insieme. A noi interessa la rappresentazione che minimizza $d(s, N)$, cioè $d_{min}(s, N)$
Una rappresentazione è tanto più efficiente quanto $d(s, N)$ si avvicina a $d_{min}(s, N)$

Esempio $s = 1, \Gamma = \{0\}$ l'unica possibilità è variare la lunghezza $\Rightarrow d_{min}(1, N) = N$, estremamente sfavorevole.
$s = 2, \Gamma = \{0, 1\}$, $\forall\, k \geq 1$ ho 2^k sequenze di lunghezza k. Il numero totale di sequenze lunghe da 1 a k è $2^{k+1} - 2$ (si esclude anche la sequenza nulla). Con N oggetti da rappresentare $\Rightarrow k \geq \log_2(N + 2) - 1 \Rightarrow N$ sequenze diverse tutte di $\log_2(N)$ caratteri.

Efficiente Codifica efficiente quando c'è questa riduzione logaritmica, **efficiente** quando . Sequenze della stessa lunghezza è vantaggioso perché non servono caratteri separatori. Per questo è necessario che l'alfabeto contenga almeno due caratteri.
La **notazione posizionale** è una rappresentazione efficiente indipendentemente dalla base $s \geq 2$ scelta. Un intero N è rappresentato con un numero d di cifre $\mid \log_s(N) \leq d \leq \log_s(N) + 1$

1.4 Richiamo della teoria della calcolabilità

Problemi computazionali Formulati matematicamente di cui cerchiamo una soluzione algoritmica: **decidibili** (e **trattabili** o **non trattabili**), o **non decidibili**.
Calcolabilità $\rightarrow$ **Algoritmo** e **problema non decidibile**
Complessità $\rightarrow$ **Algoritmo efficiente** e **problema intrattabile**.

Numerabilità Due insiemi A e B hanno lo stesso numero di elementi $\Leftrightarrow$ si può stabilire una **corrispondenza biunivoca** tra i loro elementi.
Questo porta alla definizione di **numerabile**: un insieme è numerabile $\Leftrightarrow$ i suoi elementi possono essere messi in **corrispondenza biunivoca con i numeri naturali**.
Numerabile significa che **possiede un'infinità numerabile di elementi**. Esempi: l'insieme dei numeri naturali N, l'insieme degli interi Z (avendo n in corrispondenza biunivoca con $2n + 1$ per $n \geq 0$ e $n \leftrightarrow 2|n|$ per $n < 0$, dando la sequenza $0, -1, 1, -2, 2 \ldots$) o anche l'insieme dei naturali pari ($2n \leftrightarrow n$)

Enumerazione delle sequenze Si vuole elencare in uno ordine ragionevole le sequenze di lunghezza finita costruite su un alfabeto finito. Le sequenze non sono in numero finito, quindi non si potrà completare l'elenco.
Lo scopo è **raggiungere qualsiasi sequenza σ arbitrariamente scelta in un numero finito di passi**. σ deve dunque trovarsi a **distanza finita** dall'inizio dell'elenco. Non va bene l'ordine del dizionario perché non saprei la posizione della prima stringa che inizia con b perché le stringhe composte da tutte a sono infinite.
Si stabilisce un ordine tra i caratteri. Si ordinano prima in lunghezza crescente e, a pari lunghezza, in ordine alfabetico.

Esempio $\Gamma = \{a, b, \ldots, z\}$, avrei
$a, b, \ldots, z$,
$aa, ab, \ldots, az, ba, bb, \ldots, bz, \ldots, zz, \ldots$
Ad una sequenza arbitraria corrisponde un numero intero, e la sequenza s arbitraria si troverà tra quelle di lunghezza $|s|$ in posizione alfabetica. Quindi ad una sequenza arbitraria $\leftrightarrow n$ che indica la posizione nell'elenco, e ad un numero naturale $n \leftrightarrow$ la sequenza che occupa l'n-esima posizione nell'elenco.

La **numerazione delle sequenze è fattibile perché sono di lunghezza finita**, anche se illimitata. Cioè per qualunque intero d scelto a priori, esistono sequenze di lunghezza maggiore di d. Per sequenze di lunghezza infinita la numerazione non è possibile

Insiemi non numerabili Insiemi non equivalenti a N come R, $(0, 1)$, l'insieme di tutte le linee del piano, insieme delle funzioni in una o più variabili... $\Rightarrow$ l'**insieme dei problemi computazionali non è numerabile**. Perché un problema computazionale è sempre visualizzabile come una funziona matematica, che associa ad ogni insieme di dati espressi da k numeri interi il corrispondente risultato espresso da j numeri interi

$$f : N^k \rightarrow N^j$$

Quindi l'insieme di queste f **non è numerabile**.

Diagonalizzazione $F = \{$ funzioni $f \mid f : N \rightarrow \{0, 1\}\}$, ogni $f \in F$ è rappresentata da una sequenza infinita

x 0 1 2 3 ... n ...

$f(x)$ 0 1 0 1 ... 0 ...

ma se è possibile è rappresentabile con una regola (f 0 se x pari 1 se x dispari)
Per assurdo, ipotizzo F numerabile. Si può assegnare ad ogni funzione un numero progressivo nella numerazione e costruire una tabella infinita con tutte le funzioni.

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...

Definisco $g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases} \Rightarrow g$ non può corrispondere a nessuna delle f_i della tabella, perché differisce da tutte le funzioni almeno nella diagonale principale.
$g(x) \mid 0\,1\,1\,1\ldots$
Per assurdo $\exists\, j \,|\, g(x) = f_j(x) \Rightarrow g(j) = f_j(j)$ ma per la definizione $g(j)$ è il complemento di $f_j(j)$, quindi $g(j) \neq f_j(j)$ **contraddizione**.
Per qualunque numerazione scelta esiste sempre almeno una funzione esclusa, quindi F non è numerabile.

1.4.1 Algoritmi

Algoritmi la **formulazione di un algoritmo**, una sequenza finita di operazioni, completamente e univocamente determinate, **dipende dal modello di calcolo utilizzato**.
Qualunque modello si scelga, gli algoritmi devono essere descritti da sequenze finite di caratteri di un alfabeto finito $\Rightarrow$ sono **possibilmente infiniti ma numerabili**.

Problemi computazionali Sono **funzioni matematiche** che associano ad ogni insieme di dati il corrispondente risultato, e **non sono numerabili** come visto prima.

Problema della rappresentazione C'è una drastica perdita di potenza, perché gli algoritmi sono numerabili ma sono meno dei problemi computazionali

$$|\{Problemi\}| >> |\{Algoritmi\}|$$

$\Rightarrow$ **esistono problemi privi di un corrispondente algoritmo di calcolo**. Per esempio, il problema dell'arresto.

Lezione di Turing *Non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.*

1.4.2 Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo?
Oppure
la decidibilità è una proprietà del problema?

I linguaggi di programmazione esistenti sono tutti equivalenti?
Ce ne sono di alcuni più potenti/più semplici di altri?
Ci sono algoritmi descrivibili in un linguaggio ma non in un altro?
È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o altri calcolatori?
Le teorie della calcolabilità e della complessità dipendono dal modello di calcolo?

Tesi di Church-Turing Tutti i *ragionevoli* modelli di calcolo **risolvono esattamente la stessa classe di problemi**, quindi **si equivalgono nella possibilità di risolvere i problemi** pur operando con diversa efficienza.

Tesi C-H: la decidibilità è una proprietà del problema

Incrementi qualitativi sui calcolatori o sui linguaggi di programmazione servono **solo** ad abbassare i tempi di esecuzione o rendere più agevole la programmazione.

1.4.3 Decidibilità e trattabilità

Ci sono quindi problemi che non possono essere risolti da nessun calcolatore, indipendentemente dal tempo impiegato (**problemi indecidibili**).
Ci sono poi problemi decidibili che possono richiedere tempi di risoluzione esponenziali nella dimensione dell'istanza (**problemi intrattabili**).
Ci sono poi problemi che possono essere risolti con algoritmi di costo polinomiale nella dimensione dell'inpu (**problemi trattabili**).
Abbiamo poi una famiglia di problemi il cui stato non è noto: clique (cricca), cammino hamiltoniano... Sappiamo risolverli (decidibili) con algoritmi di costo esponenziale, ma non abbiamo limiti inferiori esponenziali. I migliori limiti inferiori sono polinomiali: c'è un gap fra il limite inferiore (polinomiale) e costo della migliore soluzione a disposizione (esponenziale) (**presumibilmente intrattabili**).

Notazione

Studiamo la dimensione dei dati trattabili in funzione dell'incremento della velocità del calcolatori.
Dati i calcolatori C_1, C_2 (k volte più veloce di C_1) e tempo di calcolo a disposizione t, avrò n_1 dati trattabili in tempo t su C_1 e n_2 trattabili in tempo t su C_2.

Si **osserva** che **usare C_2 per un tempo t equivale a usare C_1 per un tempo $k \cdot t$.**

Algoritmi polinomiali Un algoritmo polinomiale che risolve il problema in $c \cdot n^s$ secondi, con c ed s costanti.

C_1 $c \cdot n_1^s = t \Rightarrow n_1 = (t/c)^{1/s}$

C_2 $c \cdot n_2^s = t \Rightarrow n_2 = k^{1/s} \cdot (t/c)^{1/s}$

$\Rightarrow$ $n_2 = k^{1/s} \cdot n_1$, miglioramento di un fattore **moltiplicativo** $k^{1/s}$

Algoritmi esponenziali Un algoritmo polinomiale che risolve il problema in $c \cdot 2^n$ secondi, con c costante.

C_1 $c2^{n_1} = t \Rightarrow 2^{n_1} = t/c$

C_2 $c2^{n_2} = k \cdot t \Rightarrow 2^{n_1} = k \cdot t/c = k2^{n_1}$

$\Rightarrow$ $n_2 = n_1 + \log_2(k)$, miglioramento di un fattore **additivo** $\log_2(k)$

Di conseguenza **un algoritmo efficiente è di gran lunga più importante di un calcolatore più potente**.

1.4.4 Tipologie di problemi

Dato un problema Π su un insieme di istanze in ingresso I con un insieme di soluzioni S.

Problemi decisionali Richiedono una risposta binaria $S = \{0, 1\}$, quindi istanze positive $x \in I \,|\, \Pi(x) = 1$ o negative $x \in I \mid \Pi(x) = 0$. Esempio: verifica se un numero è primo, o se un grafo è connesso.
La teoria della complessità computazionale è definita principalmente in termini di problemi di decisione: risposta binaria, quindi il tempo per restituire la risposta è costante, e la complessità di un problema è già presente nella versione decisionale.

Problemi di ricerca Data un'istanza x, richiede di restituire una soluzione s.

Problemi di ottimizzazione Data un'istanza x, si vuole trovare la **migliore** soluzione s tra tutte quelle possibili. Esempio: clique di dimensione massima, cammino minimo...

1.4.5 Classi di complessità

Dato un problema **decisionale** Π ed un algoritmo A, diciamo che A **risolve** Π se, data un'istanza di input x, $A(x) = 1 \Leftrightarrow \Pi(x) = 1$
A risolve Π in **tempo** $t(n)$ **e spazio** $s(n)$.

Classi Time e Space

Time($f(n)$): insieme dei **problemi decisionali che possono essere risolti in tempo O($f(n)$)**

Space($f(n)$): insieme dei **problemi decisionali che possono essere risolti in spazio O($f(n)$)**

Classe P Classe dei problemi risolvibili in **tempo** polinomiale nella dimensione dell'istanza di input.
Algoritmo polinomiale nel tempo: $\exists c, n_0 > 0 \mid$ il numero di passi elementari è al più n^c per ogni input di dimensione $n > n_0$.

Classe PSPACE Classe dei problemi risolvibili in **spazio** polinomiale nella dimensione dell'istanza di input. Molto più grande di P.
Algoritmo polinomiale nello spazio: $\exists\, c, n_0 > 0 \mid$ il numero di celle di memoria è al più n^c per ogni input di dimensione $n > n_0$.

Classe EXPTIME Classe dei problemi risolvibili in tempo esponenziale nella dimensione dell'istanza di input.

$$P \subseteq PSPACE \subseteq EXPTIME$$

Non è noto se queste inclusioni siano note, ad oggi. L'unico risultato dimostrato finora riguarda P ed EXPTIME: esiste un problema che può essere risolto in tempo esponenziale ma per cui il tempo polinomiale non è sufficiente (es: torri di Hanoi).

1.4.6 Certificato

Per alcuni problemi, per le istanze accettabili (istanze in cui la risposta del problema decisionale è si), è possibile certificare che quell'istanza è accettabile con un certificato y che può convincerci dell'accettabilità.
Per clique, il certificato è il sottoinsieme di k vertici che forma la clique. Per il cammino hamiltoniano è la permutazione degli n vertici che formano il cammino. Per SAT, sono le assegnazioni che rendono vera la formula. Il certificato ha dimensione polinomiale (k, n) e la verifica del certificato è lineare.
Una volta che ho il certificato lo vado a verificare: attestato breve di esistenza di una soluzione con determinate proprietà. Si definisce solo per istanze accettabili, perché spesso la non accettabilità non è facile costruire un certificato.

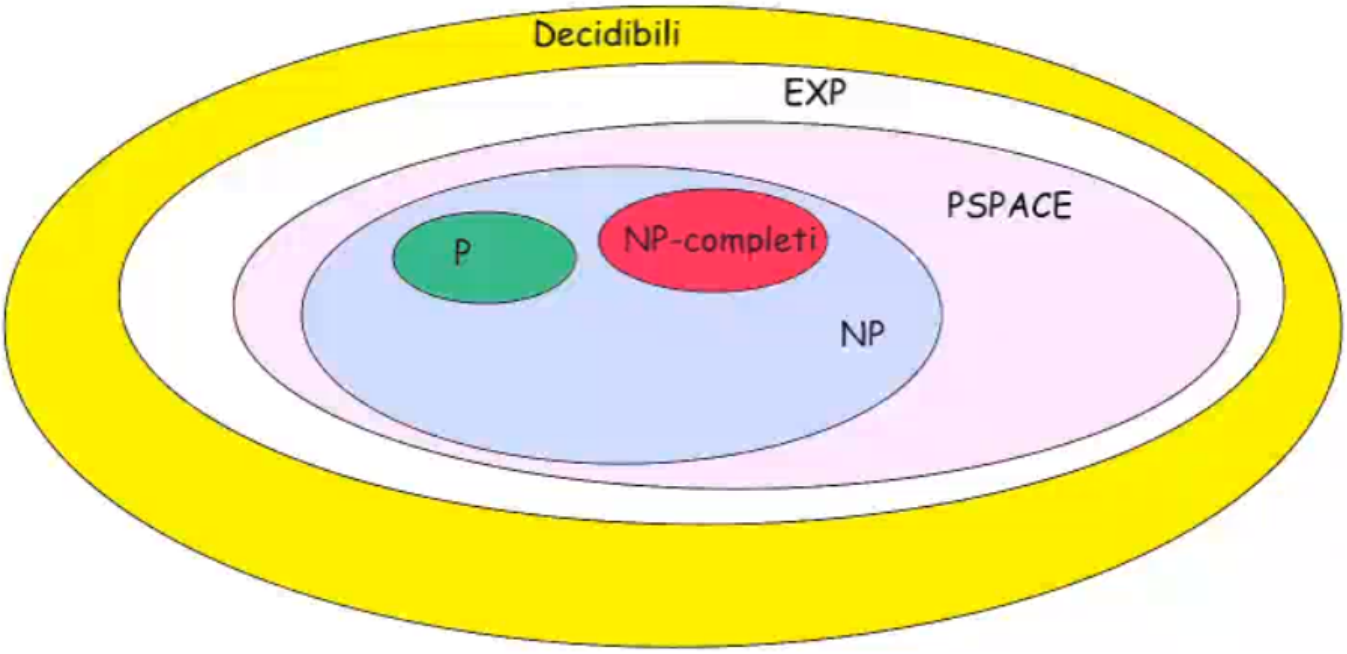
Idea Usare il costo della verifica di un certificato per un'istanza accettabile per **caratterizzare la complessità del problema** stesso.
Un problema è **verificabile in tempo polinomiale** se: tutte le istanze accettabili ammettono un certificato di lunghezza polinomiale ed esiste un algoritmo di verifica polinomiale in n.

Classe NP Classe dei problemi decisionali **verificabili in tempo polinomiale**. (NP = polinomiale su macchine non deterministiche)

P $\subset$ NP? Ovviamente si, ogni problema in P ammette un certificato verificabile in tempo polinomiale: eseguo l'algoritmo che risolve il problema per costruire il certificato.
Quello che non sappiamo è $P = NP$ oppure $P \neq NP$. Si pensa $P \neq NP$.
Si possono individuare i problemi più difficili in NP, ovvero quelli candidati ad appartenere ad NP se $P \neq NP$: sono i problemi NP-completi, quelli per cui se esiste un algoritmo polinomiale per risolvere un NP-completo allora tutti i problemi NP potrebbero essere risolti in tempo polinomiale e quindi P = NP.
Quindi tutti i problemi NP-completi sono risolvibili in tempo polinomiale oppure nessuno lo è.

Tutti i problemi NP-completi possono essere ridotti l'un l'altro, sono NP-equivalenti.

Gerarchia delle classi secondo le attuali congetture

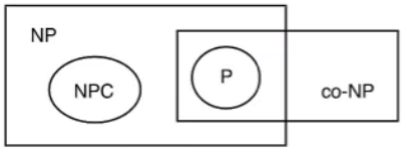


La fattorizzazione ad esempio $\in NP - (P \cup NPcompleti)$, infatti è risolto in tempo polinomiale su macchine quantistiche.

1.4.7 Classi co-P e co-NP

C'è molta differenza tra certificare l'esistenza e certificare la non esistenza di una soluzione. Dato un problema Π possiamo definire coΠ che accetta tutte e sole le istanze rifiutate da Π.
La classe coP è la classe per cui coΠ ∈ P. P = coP, i problemi complementari e i co-complementari (originali) si possono entrambi risolvere in tempo polinomiale: risolvo il problema complementare e complemento il risultato.

Questo non vale per coNP, la classe per cui coΠ ∈ NP. Si congettura che siano diverse, se la congettura è vera allora $P \neq NP$



Capitolo 2

Sequenze Casuali

2.1 Esempi di algoritmi numerici

Algoritmo di Euclide Algoritmo per il calcolo dell'MCD fra due interi.
Suppongo due interi a, b con $a \geq b$, $a > 0$ e $b \geq 0$
$\text{MCD}(a,b) = \begin{cases} a & b = 0 \\ MCD(a, a \bmod b) & else \end{cases}$

Valutazione complessità Data I istanza di input composta da a, b, vengono rappresentati ad esempio in base due. Quindi la dimensione n dell'istanza di input I $n = |I| = \Theta(\log a + \log b) = \Theta(\log a)$.
L'algoritmo è ricorsivo, quindi bisogna valutare il numero delle chiamate ricorsive, che dipenderanno dai dati. Ci saranno istanze in cui si termina subito (ad esempio se a multiplo di b, cioè $a \bmod b = 0$). In generale, **il numero di chiamate cresce con** $\log a$, perché $a \bmod b$ rimpiazza a.

Si osserva che $a \bmod b < \frac{a}{2}$
Questo perché $a = qb + (a \bmod b)$ e siccome per ipotesi $a \geq b \Rightarrow b \geq 1$ e lo è anche $q \Rightarrow a \geq b + (a \bmod b) > (a \bmod b) + (a \bmod b)$ perché $b > (a \bmod b)$.
$\Rightarrow 2(a \bmod b) < a \Rightarrow (a \bmod b) < \frac{a}{2}$.
Prima chiamata su a, b.
Seconda su $b, (a \bmod b)$.
Terza su $(a \bmod b), (b \bmod (a \bmod b))$.
Quindi ad ogni chiamata a si riduce almeno della metà, e lo possiamo fare al massimo $\log a$ volte
Quindi avrò $O(\log a)$ **ricorsioni**.

Il costo del calcolo del modulo è $O(\log a \cdot \log b) = O(\log^2 a)$
Complessivamente $\text{T}(n) = O(\log^3 a) = O(n^3)$ **polinomiale nella dimensione dell'istanza** $|I|$ (cioè nel numero di cifre), **polilogaritmico nel valore dei dati**

Test di primalità Versione inefficiente.
Primo(N): `for` $(i = 2, i \leq \sqrt{N}, i++)$
`if` $N\%i == 0$ `return false`
`else` *a fine ciclo* `return true`.
Uso la proprietà che se N non è primo allora ha almeno un divisore $\leq \sqrt{N}$.

Valutazione di complessità $I = N$, $|I| = \Theta(N) = n$
Ho $\sqrt{N}$ iterazioni, ciascuna di costo $\Theta(\log^2 N)$

$T(n) = O(\sqrt{N} \cdot \log^2 N) = O(2^{\frac{n}{2}} \cdot n^2)$ **pseudopolinomiale**, cioè **polinomiale nel valore di N ma esponenziale nella dimensione** $|I| = n$.

2.2 Casualità

Problema Data una sequenza binaria, vogliamo **capire se è una sequenza casuale** o meno. Le sequenze casuali sono importanti sia per la generazioni delle classi, sia perché **in crittografia spesso si ricorrono ad algoritmi randomizzati che usano sequenze casuali per funzionare**.

Significato algoritmico della casualità Vedendo la teoria di Kolmogorov. Prendiamo due sequenze

h: 1111...1 lunga n

h': 10110110101011010100101...0

La prima è molto facile da descrivere (*scrivi n "uni"*), mentre **descrivere la seconda è molto meno pratico**: l'intuizione è che **una sequenza casuale non si può descrivere in modo compatto**.
Ponendo $n = 20$, la probabilità di generare h è $P(h = (1/20)^{20}$, $P(h') = (1/20)^{20}$ (1/2 per generare 1, 1/2 per generare 0...).
A_h algoritmo che genera h. Formalizzabile semplicemente (*genera n uni*)
$|A_h| = \#$ bit di A_h codificato in binario $= \log n +$ const (la parte costante è la generazione e l'output, varia solo n) $\Rightarrow$ con $\log n$ bit ne abbiamo descritti n.
$A_{h'} =$ `print` h', $|A_{h'}| > |n| = |h'|$

L'intuizione è **una sequenza binaria è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta di** h. Se posso usare meno bit vuol dire che la sequenza ha una qualche regolarità.

Sistemi di calcolo Sono un infinità numerabile $S_1 \dots S_i \dots$
Prendiamo S_i, p programma che genera la sequenza h nel sistema S_i, cioè $S_i(p) = h$
Def: **la complessità di Kolmogorov di** h **nel sistema** S_i **è** $K_{S_i}(h) = min\{\ |p|\ \mid S_i(p) = h\}$ cioè la minima lunghezza del programma p che in S_i genera h stessa.

Se la sequenza h non segue alcuna legge semplice di regolarità, allora **il più breve programma in grado di generarla dovrà contenerla al suo interno**, cioè sarà almeno lungo quanto la sequenza stessa e la genererà trasferendola in output. Quindi $K_{S_i}(h) = |h| +$ `const`$_i$. La costante è la parte di programma che trasferisce in output, dipende da S_i ma non ha h.

Sistema di calcolo universale Tra tutti i sistemi di calcolo possibili ne esiste uno **universale in grado di simulare tutti gli altri**. Lo chiamiamo S_u e lo prendiamo in considerazione.
Supponiamo $p \mid S_i(p) = h$, allora $S_u(\langle i, p\rangle) = S_i(p) = h$. Ottengo $q = \langle i, p\rangle$ programma che genera h in S_u

$|q| = |\langle i, p\rangle| = |i| + |p| = \log_2 i + |p|$ quindi la lunghezza di q dipende da i ma non da h.
$\forall\, h\ \forall\, i\ \ K_{S_u}(h) \leq K_{S_i}(h) + C_i$
L'uguale vale per le sequenze generate per simulazione di S_i non essendoci per S_u algoritmi più "brevi".
Il minore vale per sequenze generabili con programmi più corti (ad esempio per simulazione su un altro sistema $S_j \neq S_i$).

Def La complessita di Kolmogorov di una sequenza h è $K(h) = K_{S_u}(h)$

2.2.1 Sequenze casuali

Sequenza casuale Una sequenza h è casuale se $K(h) \geq |h| - \lceil \log_2 h \rceil$
Non entra in gioco come genero la sequenza, la **casualità è una proprietà della sequenza**.

Conteggio delle sequenze $\forall\, n, \exists$ sequenze casuali (secondo Kolmogorov) di lunghezza n
Dim: n, $S = 2^n$ # sequenze binarie di lunghezza n
T = # sequenze di lunghezza n NON casuali. L'obiettivo è dimostrare che T < S.
Pongo $N = \#$ sequenze binarie di lunghezza $< n - \lceil \log_2 n \rceil = \sum_{i=0}^{n-\lceil \log n \rceil - 1} 2^i = s^{n-\lceil \log n \rceil} - 1$
Tra queste N sequenze ci sono anche i programmi che generano le T sequenze non casuali di lunghezza n.
$\Rightarrow T \leq N < S \Rightarrow T < S$

Quindi non solo esistono ma sono anche la **maggioranza**, essendo enormemente più numerose di quelle non casuali. Lo vediamo studiando il rapporto

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil \log n \rceil}}{2^n} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log n \rceil}} \quad \lim_{n \to +\infty} \frac{T}{S} = 0$$

Stabilire la casualità Data una sequenza arbitraria di lunghezza n, stabilire se è casuale secondo Kolmogorov è un problema **indecidibile**.

Dim: per assurdo suppongo esista un algoritmo Random | $\text{Random}(h) = \begin{cases} 1 & h \textit{ casuale} \\ 0 & \textit{altrimenti} \end{cases}$

Possiamo costruire l'algoritmo Paradosso che enumera tutte le possibili sequenze binarie in ordine crescente di lunghezza.

Paradosso:
`for` (binary h = 1 → infty) `do`
`if` ($|h| - \lceil \log |h| \rceil > |P|$ && Random(h) == 1) `return` h

P è una sequenza binaria che rappresenta la codifica del programma complessivo Paradosso + Random, quindi $|P|$ = |Paradosso| + |Random|, costante che non dipende da h, perché la sequenza h non compare in P ma solo come nome di variabile. Il valore rimane registrato fuori dal programma.
Paradosso quindi restituisce come risultato la prima sequenza casuale | $|h| - \lceil \log_2 |h| \rceil > |P|$
Quindi se ∃ sequenze casuali di qualsiasi lunghezza, quindi certamente ne esisterà una che soddisfa entrambe le condizioni dell'`if`, che viene generata.
Ma la prima condizione mi dice che il programma rappresentato da P è breve e genera h, quindi h non è casuale perché prodotta con un programma breve.
Quindi $K(h) \leq |P|$, cioè P genera h) ma $|P| < |h| l \lceil \log_2 |h| \rceil$ quindi h non è casuale.
Ma la seconda condizione dice h casuale, giungendo ad un **paradosso** dato dall'assumere l'esistenza di Random.

2.2.2 Sorgente binaria casuale

Generatore Genera una sequenza di bit con queste proprietà:

1. P(0) = P(1) = 1/2, cioè genera 1 o 0 a pari probabilità.
   Si può indebolire richiedendo P(0) > 0, P(1) > 0 immutabili nel tempo della generazione.

2. La generazione di un bit è indipendente dalla generazione degli altri.
   ⇒ non si può prevedere il valore di un bit osservando quelli già generati.

Perché possiamo indebolire la prima proprietà? Supponiamo di essere in un caso in cui P(0) > P(1), allora è **sempre possibile bilanciare la sequenza**.
Supponiamo di generare 001100111000010100 e si **dividono a coppie** 00 11 00 11 10 00 01 01 00 e si scartano le coppie uguali. Si associano le coppie miste, ad esempio 01 → 0 e 10 → 1. Si presentano in modo equiprobabile, quindi la sequenza si ribilancia ottenendo 100 (caso poco significativo perché sequenza corta).

Esistono queste sorgenti? Non si sa. Nella pratica non è possibile garantire la perfetta casualità o l'indipendenza. Quindi sfrutteremo le casualità presenti in processi fisici o processi software.

Generatore di sequenze brevi

Fenomeni casuali presenti in natura Ad esempio il rumore su un microfono o il tempo di decadimento di alcune particelle, sfruttabili come sorgenti di casualità.
Il problema di questo approccio è che bisogna non avere accesso fisico ai dispositivi usati (es: microfono manomesso), oltre alla difficoltà pratica di usare certe sorgenti.

Processi software Come la temperatura, la posizione della testina del disco fisico...

Pseudocasuale Si genera la casualità **mediante un algoritmo**, **cercandola all'interno di processi matematici**. **Generatore di numeri pseudo-casuali**: ad esempio `rand()` del C.
Perché pseudocasuali? Perché sono algoritmi deterministici e brevi, quindi non risultano casuali secondo Kolmogorov.

Come funzionano? Partono da un *seed* (seme), breve sequenza che viene amplificata per creare una sequenza più lunga. Quindi un generatore è un **amplificatore di casualità**.

Input: seme (sequenza o valore breve)

Output: flusso di bit arbitrariamente lungo e periodico.
Al suo interno contiene una sottosequenza che si **ripete**, quindi **un generatore è tanto migliore tanto più è lungo il suo periodo**

Avengo $s = \#$ bit nel seme e n lunghezza della sequenza ottenuta dal generatore, con $n >> s$, ho **una sequenza diversa per ogni seme**, con 2^s possibili semi.
$\#$ sequenze diverse $2^s << 2^n$ $\#$ sequenze possibili

Generatore lineare $x_i = (a \cdot x_{i-1} + b) \bmod n$ con a, b, n interi positivi.
Il seme è un valore intero iniziale casuale x_0, quindi quando $x_i = x_0$ la sequenza si ripete.
Dobbiamo avere $\text{MCD}(b, n) = 1$, $(a-1)$ divisibile per ogni fattore primo di n e $(a-1)$ dev'essere un multiplo di 4 se anche n lo è.
Servono per garantire che il generatore produca una permutazione degli interi da 0 a $m-1$

2.3 Test statistici

Per valutare le sequenze prodotte da un generatore pseudocasuale.
Si valuta se la sequenza presenta le proprietà tipiche di una sequenza casuale:

test di frequenza

poker test: se sottosequenze siano distribuite in modo equo

test di autocorrelazione: verifica che non ci siano regolarità nella sequenza ottenuta

run test: verifica se sottosequenze massimali di elementi tutti ripetuti abbiano una distribuzione esponenziale negativa, cioè più sono lunghe meno sono frequenti.

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit**, molto severo che implica tutti gli altri 4 test statistici. Intuitivamente, verifica che sia impossibile prevedere gli elementi della sequenza prima di generarli.

Test di prossimo bit Un generatore binario **supera** il test di prossimo bit **se non esiste un algoritmo polinomiale in grado di prevedere l'$i+1$-esimo bit della sequenza a partire dalla conoscenza degli i bit precedentemente generati con probabilità maggiore di 1/2**.
Quindi se si hanno a disposizione risorse polinomiale non si può prevedere il prossimo bit. I generatori che superano questo test sono detti **generatori crittograficamente sicuri**.

Generatore polinomiale Non è crittograficamente sicuro.
$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \ldots + a_t x_{i-1} + a_{t-1}) \bmod n$
$r_i = \frac{x_i}{n}$

2.4 Generatori crittograficamente sicuri

Come costruire generatori crittograficamente sicuri Si ricorre alle **funzioni one-way**: funzioni facili da calcolare ma difficili da invertire, cioè **computabili in tempo polinomiale ($x \mapsto f(x)$)**, ma **computazionalmente difficile invertire la funzione ($y \mapsto x = f^{-1}(y)$)**. Come costruire queste funzioni?

Idea Con f one-way, scelgo il seme x_0. Genero S: x $f(x)$ $f(x_1) = f(f(x))$ $\ldots$ $x_i = f(f(\ldots(x_{i-1})\ldots))$
Cioè si itera l'applicazione della funzione one-way un numero arbitrario di volte. L'**idea è restituire la sequenza al contrario**, perché se conosco x_{i+1} non riesco a calcolare facilmente x_i.
Ogni elemento della sequenza S si può calcolare efficientemente con l'elemento precedente, ma non dai valori successivi perché f è one-way. Si calcola S per un certo numero di passi senza svelare il risultato e si comunicano gli elementi in ordine inverso. Ogni elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati.

Generatori binari crittograficamente sicuri Si usano i **"predicati *hard core*" delle funzioni one-way**.
Un predicato hard core di una funzione one-way $f(x)$ è $b(x)$ se $b(x)$ è facile da calcolare quando x è noto, ma è difficile da prevedere se si conosce $f(x)$.

Un esempio di funzione one-way è l'elevamento a quadrato in modulo ($f(x) = x^2 \bmod n$ con n non primo). Un predicato hard core è "$b(x) = x$ è dispari"

Generatore BBS Crittograficamente **sicuro**.
$n = p \cdot q$, con p e q primi **grandi**.

$p \bmod 4 = 3$

$q \bmod 4 = 3$

$2\lfloor \frac{p}{4} \rfloor + 1$ e $2\lfloor \frac{q}{4} \rfloor + 1$ **primi fra loro**

$\Rightarrow$ y coprimo con n, si calcola $x_0 = y^2 \bmod n$ e **si usa come seme** per **calcolare una successione di** $m \leq n$ **interi**

$x_i = (x_{i-1})^2 \bmod n$
$b_i = 1 \Leftrightarrow x_{n-i}$ è dispari

$b_1 = 1 \Leftrightarrow x_{n-1}$ dispari
$b_0 = 1 \Leftrightarrow x_n$ dispari

Quindi $x_0 = b_n$, ottengo una sequenza del tipo

$x_0 \to x_1 \to \ldots \to x_{n-1} \to x_n$

$\Rightarrow$ $b_n \to b_{n-1} \to \ldots \to b_1 \to b_0$
La **sequenza viene restituita in ordine inverso** $b_0 b_1 \ldots$

2.4.1 Generatori di numeri pseudocasuali basati su cifrari simmetrici

Idea Prendere un cifrario simmetrico e la sua chiave. Anziché usarlo per costruire un crittogramma, si sostituisce il messaggio da cifrare con un **seme iniziale** legato al generatore. Si comincia a cifrare in questo modo: produce una sequenza imprevedibile per le proprietà del cifrario.
Di seguito, un esempio approvato dal FIPS:

Usa il DES

r = # bit delle parole che vengono prodotte
($r = 64$ nel DES)

s = seme casuale di r bit

m = # parole da produrre

k = chiave segreta del cifrario

```
Generatore(s, m){ //flusso di output di m*r bit
        d = <rapp. su r bit di data e ora>;
        y = C(d, k);
        z = s;
        for (i = 1; i <= n; i++) {
                xi = C(y xor z, k);
                z = C(y xor xi, k);
                output(xi);
        }
}
```

2.5 Algoritmi randomizzati

Si dividono in due classi fondamentali:

Algoritmi **Las Vegas**
Generano un **risultato sicuramente corretto** in un **tempo probabilmente breve**.
Caso tipico: quick sort. Qualche passo a caso per cercare di evitare i casi sfavorevoli.

Algoritmi **Montecarlo**
Generano un **risultato probabilmente corretto** in un **tempo sicuramente breve**.
Probabilità di errore deve essere **arbitrariamente piccola e matematicamente misurabile**.
Caso tipico: test di primalità.

2.5.1 Test di primalità (Miller, Rabin)

N intero (dispari) da testare di n bit
$\Rightarrow$ N-1 è pari, $N-1 = 2^w \cdot z$ con z dispari, w esponente della potenza di 2 più grande che divide $N-1$
Es: $N = 21, N-1 = 20 = 2^2 \cdot 5$ quindi $z = 5, w = 2$

Da N, calcolo $N-1 \to \frac{N-1}{2} \to \frac{N-1}{4} \to \ldots \to z = 1$ quindi # divisioni per $2 \leq \log N = n$ volte
Quindi trovo w e z in maniera efficiente, $O(n)$ passi

Sia quindi N primo e $2 \leq y \leq N-1$ arbitrario detto *testimone*, allora

P1 MCD(N,y) = 1, per la primalità

P2 $(y^z \bmod N = 1) \vee (\exists\, i \;\; 0 \leq i \leq w-1 \mid y^{2^i \cdot z} \bmod N = -1)$

Se una delle due proposizione è falsa allora N non è primo, ma ci sono numeri composti che verificano P1 e P2 ma non sono primi (sono pochi).

Lemma 1 (Miller, Rabin) N è composto, allora il numero di testimoni y che soddisfano i predicati è basso.
Cioè N composto $\Rightarrow$ il numero di interi $y \mid 2 \leq y \leq N-1$ che soddisfano entrambi i predicati P1 e P2 è minore di $N/4$
La probabilità di scegliere un testimone che rende veri P1 e P2 $< \frac{N/4}{N-2} < \frac{1}{4}$

N y scelto a caso in $[2, N-1]$:

se uno dei due predicati è **falso** $\to N$ è **certamente composto**

se sono entrambi veri $\Rightarrow N$ è composto con probabilità $< \frac{1}{4}$, dunque N è primo con probabilità $> \frac{3}{4}$

Iterando il test k volte, la probabilità di errore diventa $< \left(\frac{1}{4}\right)^k$, con $k = 30$ diventa inferiore al 10^{-18}.
Di seguito l'algoritmo.

```
Verifica(N,y) { //controlla la validita del certificato y (certifica che N sia composto)
        if (P1 == false or P2 == false) return 1; //N certamente composto
        else return 0; //N probabilmente primo (prob errore < 1/4)
}

TestMR(N,k) {
        for (i = 0; i < k; i++) {
                //sceglie a caso y in [2, N-1]
                if (Verifica(N, y) == 1) return 0; //N non primo
        }
        return 1; //N probabilmente primo (prob errore < (1/4)^k)
}
```

Valutazione di complessità TestMR costa k volte Verifica, quindi valuteremo quest'ultimo.
Il calcolo dell'MCD, quindi di P1, è facile. Quindi indaghiamo la valutazione di P2: bisognerà calcolare

$y^z \bmod N == 1$ e, in caso non sia verificato

$y^{2^i \cdot z} \bmod N$, con $0 \leq i \leq w-1$

Nella seconda parte di P2 và calcolato $y^z \bmod N$, poi $y^{2z} \bmod N$ come quadrato del precedente.
L'esponente massimo per y è $i = w-1$ (da $N-1 = 2^w \cdot z$), perché $2^{w-1} \cdot z = \frac{N-1}{2}$
Quindi $y^{\frac{N-1}{2}} \bmod N$, al massimo voglio eseguire $\log N$ moltiplicazioni. La moltiplicazione la sappiamo fare polinomiale, e l'elevamento a potenza possiamo eseguirlo polinomiale con l'**algoritmo delle quadrature successive** o esponenziazione veloce.
w elementi al quadrato con $w = O(\log N)$.
Quindi l'algoritmo MR dà un test efficiente per la primalità.

Algoritmo delle quadrature successive Vogliamo calcolare $x = y^z \bmod s$, con x, z, s stesso ordine di grandezza.

Si scompone z in una somma di potenze di 2
$z = \sum_{i=0}^{t} k_i \cdot 2^i$ con $k_i \in \{0, 1\}$
Esempio $z = 45 = 32 + 8 + 4 + 1$
Il massimo t come visto è $t = \lfloor \log_2 z \rfloor = \Theta(\log z)$

Si calcolano tutte le potenze $y^{2^i} \bmod s$ per $1 \leq i \leq t = \lfloor \log_2 z \rfloor$, ciascuna come il quadrato della precedente.
$y^{2^{i+1}} \bmod s = \left(y^{2^i}\right)^2 \bmod s$
Esempio: $x = 9^{45} \bmod 11$ e $45 = 32 + 8 + 4 + 1$, $t = 5$

$y^2 \bmod s = 9^2 \bmod 11 = 4$
$y^4 \bmod s = 4^2 \bmod 11 = 5$
$y^8 \bmod s = 5^2 \bmod 11 = 3$
$y^{16} \bmod s = 3^2 \bmod 11 = 9$
$y^{32} \bmod s = 9^2 \bmod 11 = 4$

Calcoliamo $x = y^z \bmod s = \Pi_{(i \mid k_i \neq 0)} \left(y^{2^i}\right) \bmod s$
Nell'esempio:
$y^z \bmod s = 9^{45} \bmod 11 = 9^{32+8+4+1} \bmod 11 =$
$((9^{32} \bmod 11) \cdot (9^8 \bmod 11) \cdot (9^4 \bmod 11) \cdot (9^1 \bmod 11)) \bmod 11 =$
$(4 \cdot 3 \cdot 5 \cdot 9) \bmod 11 = 1$

Costo: $t = \Theta(\log_2 z)$ quadrature e al più t moltiplicazioni $\Rightarrow \Theta(\log_2 z)$ quadrature e $O(\log z)$ moltiplicazioni. Ogni moltiplicazione ha un costo al più quadratico nel numero di cifre.
Quindi l'algoritmo è **polinomiale nella dimensione dei dati**.

2.5.2 Generazione di numeri primi

In pratica è una **generazione di un numero casuale seguita da un test di primalità**. Se il test fallisce, lo si incrementa di due iterando fino a trovare un numero *dichiarato* primo.
Sono pochi i numeri da testare grazie alla densità: il numero di interi primi e minori di N tendono a $\frac{N}{\log_e N}$ per $N \to \infty$.
Quindi per N sufficientemente grande, in un suo intorno di ampiezza $\log_e N$ cade mediamente un numero primo.

```
Primo(n): { //n: # bit del numero generato
        //genera un numero primo di almeno n bit
        //probabilita errore < (1/4)^k
        S = seq di n-2 bit prodotti da un generatore binario pseudocasuale
        N = (1 S 1) //N ha n bit ed e dispari
        while(TestMR(N,k) == 0) { N = N + 2 } //O(n) = O(logN) volte
        //TestMR costo polinomiale in n = logN -> O(n^3)
        return N;
}
```

L'algoritmo è polinomiale, circa $O(n^4)$

2.6 Classe RP

Random Polinomial Classe dei problemi decisionali verificabili in tempo polinomiale randomizzato.
Dato un problema Π, e x istanza di input allora y è un **certificato probabilistico** per l'istanza x **se ha una lunghezza** $|y|$ **al più polinomiale** in $|x|$ **e** y **è estratto perfettamente a caso da un insieme associato a** x

$A(x, y)$ in tempo polinomiale **attesta con certezza che** x ***non*** **possiede la proprietà esaminata da** Π, cioè $\Pi(x) = 0$, **oppure attesta che** x **possiede la proprietà** esaminata da Π **con probabilità** $> \frac{1}{2}$.

Capitolo 3

Cifrari storici

Scopo Consentire comunicazioni **sicure** tra poche persone, ma **i cifrari storici sono stati tutti forzati**.
La **cifratura e decifrazione** erano tutte **realizzate a carta e penna**, mentre i **messaggi** da cifrare erano **frasi di senso compiuto** in linguaggio naturale, quindi con l'alfabeto classico di 26 lettere.

3.1 Principi di Bacone

XIII Secolo

C e D devono essere **funzioni facili da calcolare**

Impossibile ricavare D se C non è nota

Il **crittogramma** $c = \mathrm{C}(m)$ deve **apparire innocente**

3.2 Antichi esempi

3.2.1 Scitale

Metodo più antico di cui si ha notizia, inventato dagli spartani nel V secolo a.C.
Asta cilindrica costruita in due esemplari identici posseduti dai due corrispondenti.

3.2.2 Erodoto, Storie

Si tatuava il messaggio sulla testa rasata di un messaggero, si aspettava che ricrescessero i capelli e si portava a destinazione, rivelando il messaggio a seguito di una seconda rasatura.

3.2.3 Enea Tattico

Opera militare del IV secolo a.C. con un capitolo dedicato ai messaggi segreti. Consigliava di inviare un libro qualsiasi sottolineandovi un sottoinsieme di lettere che costituiscono il messaggio, oppure di sostituire le vocali di un testo con altri simboli grafici.

3.2.4 Cifrario di Cesare

Il più antico cifrario di concezione moderna. L'idea è che il crittogramma è ottenuto dal messaggio in chiaro sostituendo ogni lettera con quella di 3 posizioni più avanti nell'alfabeto.
Non ha una chiave segreta, e la segretezza dipende dalla conoscenza del metodo: scoprire il metodo significa compromettere irrimediabilmente l'impiego. Il cifrario era quindi destinato all'uso ristretto di un gruppo di conoscenti.

Generalizzandolo a k posizioni più avanti, si rende più sicuro ($1 \leq k \leq 25$) e si ha k come chiave segreta.

Formulazione matematica

Con pos(X) indichiamo la **posizione nell'alfabeto della lettera X**: ad esempio pos(A) = 0, pos(Z) = 25...
La chiave k è quindi $1 \leq k \leq 25$, con $k = 26$ il cifrario lascerebbe invariato il messaggio.

Cifratura di X Lettera Y | pos(Y) = (pos(X) + k) mod 26

Decifrazione di Y Lettera X | pos(X) = (pos(Y) - k) mod 26

Un esempio con $k = 10$. Cifriamo R, cui pos(R) = 17. La sua cifratura è (17+10) mod 26 = 1 = pos(B)
Quindi R $\rightarrow$ B.

Crittoanalisi

Se si conosce la struttura del cifrario, in breve tempo **si possono applicare tutte le chiavi possibili**, che sono solo 25, ad un crittogramma: così viene **decifrato** e contemporaneamente si scopre la chiave segreta k. Come cifrario risulta quindi inutilizzabile a fini crittografici.

Gode della proprietà commutativa: data una sequenza di chiavi e di operazioni di cifratura e decifrazione, l'**ordine delle operazioni può essere permutato arbitrariamente** senza modificare il crittogramma finale.
Per esempio, date k_1 e k_2 chiavi e s sequenza,

$$\mathrm{C}(\mathrm{C}(s, k_2), k_1) = \mathrm{C}(s, k_1 + k_2)$$

$$\mathrm{D}(\mathrm{D}(s, k_2), k_1) = \mathrm{D}(s, k_1 + k_2)$$

Una sequenza di operazioni di cifratura e decifrazione può essere ridotta ad una sola operazione di cifratura o decifrazione.

Inoltre **comporre più chiavi non aumenta la sicurezza** del sistema.

3.3 Classificazione dei cifrari storici

3.3.1 Cifrari a sostituzione

Sostituiscono ogni lettera del messaggio in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata.

Sostituzione monoalfabetica Es: Cifrario di Cesare.
Alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel crittogramma.
Si possono impiegare funzioni di cifratura e decifrazione più complesse dell'addizione e della sottrazione in modulo, ottenendo uno spazio delle chiavi molto più ampio (ma sempre con una sicurezza molto modesta).

Per esempio, usare come $k = \langle a, b \rangle$, cifrare pos(Y)=(a·pos(X)+b) mod 26 e decifrare
pos(X) = $a^{-1} \cdot$ (pos(Y)$-b$) mod 26 con a^{-1} inverso di a mod 26 ($a \cdot a^{-1} = 1$ mod 26).
C'è quindi un **vincolo forte** sulla chiave: MCD(a, 26) $\neq 1 \Rightarrow$ la funzione di cifratura non è iniettiva e la decifrazione diventa impossibile. Ad esempio con $k = \langle 13, 0 \rangle$ tutte le lettere posizione pari sono trasformate in A, mentre tutte quelle dispari sono trasformate in N.
Le chiavi possibili sono quindi 12 (scelte per a) $\cdot$ 26 (scelte per b) = 312 chiavi che sono poche.

Se la segretezza dipende unicamente dalla chiave, allora **il numero di chiavi deve essere così grande da essere praticamente immune dal brute force** e deve essere **scelta in modo casuale**.

Cifrario completo prendendo una permutazione arbitraria dell'alfabeto come chiave:
lettera in chiaro in posizione i $\longrightarrow$ lettera di posizione i della permutazione.

La chiave è di 26 lettere, con uno spazio delle chiavi esteso a $26! - 1$, cioè circa $4 \cdot 10^{26}$ chiavi, il che lo rende molto vasto e inesplorabile.
Ma non è comunque sicuro: si può forzare senza ricorre a brute force, sfruttando la struttura logica dei messaggi in chiaro e l'occorrenza statistica delle lettere.

Sostituzione polialfabetica
Alla stessa lettera del messaggio corrisponde una lettera scelta in un insieme di lettere possibili, secondo una regola opportuna (a seconda della posizione o del contesto in cui appare la lettera nel messaggio). L'esempio più antico è l'archivio cifrato di Augusto.
I documenti in archivio erano scritti in numeri, invece che lettere. Augusto li scriveva in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade. Sostituiva ogni lettera del documento con il numero che indicava la distanza, nell'alfabeto greco, di tale lettera con quella in pari posizione nell'Iliade.
Per esempio:

> Lettera in posizione i nel documento: α
>
> Lettera in posizione i nell'Iliade: ϵ
>
> Carattere in posizione i nel crittogramma: 4 (distanza tra α e ϵ)

Se la chiave è lunghissima, il cifrario diventa difficile da forzare. Venne usato anche della Seconda Guerra Mondiale, prendendo come chiave una pagina prefissata di un libro e cambiandola di giorno in giorno. Lo **svantaggio** è **registrare per iscritto la chiave**.

Cifrario di Alberti con due dischi, dove si cambia chiave ogni volta che si incontra un carattere speciale. Inserendo spesso caratteri speciali (scartati nel messaggio ricostruito) diventa difficile da attaccare e il continuo cambio di chiave rende inutili gli attacchi basati sulla frequenza dei caratteri. La **Macchina Enigma** è un'**estensione elettromeccanica del cifrario di Alberti**.

Cifrario di Vigenère, con una parola segreta k come chiave. La cifratura di un messaggio m avviene disponendo m e k su due righe adiacenti, allineando le lettere in verticale (se k è più corta di m la si ricopia più volte). Ogni lettera X di m risulta allineata ad una lettera Y della chiave. La X è sostituita nel crittogramma con la lettera che si trova nella cella T all'incrocio tra la riga che inizia con X e la colonna che inizia con Y

A	B	C	...	X	Y	Z
B	C	D	...	Y	Z	A
C	D	E	...	Z	A	B
...	...	...	...	...	...	...
X	Y	Z	...	U	V	W
Y	Z	A	...	V	W	X
Z	A	B	...	W	X	Y

Quindi le lettere allineate con A non subiscono modifiche. Quelle allineate con B sono traslate di una posizione in avanti, quelle con R di 17 posizioni...
Una stessa lettera in chiaro è cifrata in modi diversi a seconda della lettera con cui è allineata, mentre per la decifrazione si esegue il processo inverso.
La sicurezza del metodo è influenzata dalla lunghezza della chiave: se contiene h caratteri, le apparizioni della stessa lettera distanti un multiplo di h nel messaggio si sovrappongono alla stessa lettera della chiave, quindi sono trasformate nella stessa lettera cifrata.
I cifrari polialfabetici non sono molto più potenti dei monoalfabetici se le chiavi non sono molto lunghe.
Se si estende Vigenèere impiegando una chiave lunga quanto il testo, casuale e non riutilizzabile, il cifrario diventa inattaccabile: **one-time pad**.

3.3.2 Cifrari a trasposizione

Permutano le lettere del messaggio in chiaro secondo una regola prefissata.
L'idea di base è eliminare qualsiasi struttura linguistica presente nel crittogramma: permutando le lettere del messaggio in chiaro e inserendone eventualmente altre ignorate nella decifrazione.

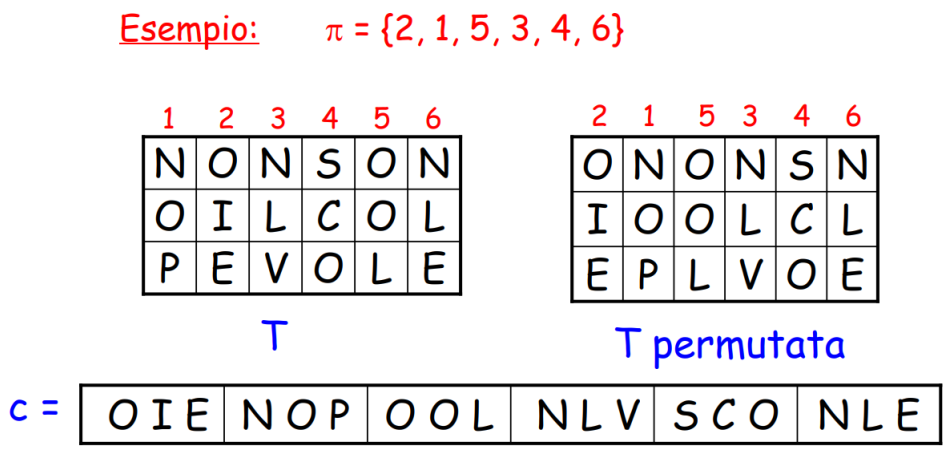
Semplice La chiave è un intero h e una permutazione π degli interi $\{1, 2, \ldots, h\}$
Nella cifratura si suddivide il messaggio in m blocchi da h lettere e si permutano le lettere di ciascun blocco secondo π.
Se la lunghezza di m non è divisibile per h, si aggiungono alla fine delle lettere qualsiasi (padding): partecipano alla trasposizione, ma sono ignorate perché la decifrazione le riporta alla fine del messaggio.
Ci sono $h! - 1$ chiavi ed h non è fissato a priori: tanto è più grande tanto è più difficile impostare un brute force. Al crescere di h però cresce anche la difficoltà di ricordare π.

Permutazione di colonne $k = \langle c, r, \pi \rangle$ con:

c ed r denotano il numero di colonne e righe di una tabella di lavoro T

π è una permutazione degli interi $\{1, 2, \ldots, c\}$

Il messaggio m è decomposto in blocchi $m_1, m_2 \ldots$ di $c \cdot r$ caratteri.
Nella cifratura i caratteri di ogni blocco sono distribuiti tra le celle di T in modo regolare, scrivendoli per righe dall'alto verso il basso. Poi le colonne vengono permutate secondo π e si prendono le colonne dalla prima leggendole dall'alto verso il basso e da sinistra verso destra, ottenendo il crittogramma.



Per cifrare il prossimo blocco, si azzera T e si ripete il procedimento.
Le chiavi sono teoricamente esponenziali nella lunghezza del messaggio, non essendoci vincoli su r e c.

Cifrario a griglia Antenato: cifrario di Richelieu. Il crittogramma può essere celato in un libro qualsiasi. La chiave è data da una scheda perforata e dall'indicazione di una pagina del libro: la decifrzione consiste nel sovrapporre la scheda alla pagina, e le lettere visibili attraverso i fori costituiscono il messaggio in chiaro.

3.4 Crittoanalisi Statistica

La sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi. Ci sono altri metodi di attacco: i cifrari storici sono stati violati con un attacco statistico di tipo cipher text.
Nella crittoanalisi statistica si fanno delle ipotesi. Ci sono delle **informazioni note al crittoanalista**: il metodo usare per la cifratura e la decifrazionem, il linguaggio naturale con cui è scritto il messaggio e si ammette che il messaggio sia sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma.

Attacco La frequenza con cui appaiono in media le varie lettere dell'alfabeto è ben studiata in ogni lingua. Dati simili sono noti per le frequenze di diagrammi (gruppi di due lettere consecutive), trigrammi (gruppi di tre lettere) e così via (**q-grammi**).

Decifrazione cifrari monoalfabetici Se un crittogramma è generato per sostituzione monoalfabetica allora la frequenza di Y nel crittogramma è circa la frequenza della corrispondente X del messaggio.
Nei **cifrari completi** si associano le lettere in base alle frequenze, provando le varie combinazioni. Con le combinazioni di prova, si studiano le frequenze dei possibili diagrammi, dei trigrammi e così via.

Decifrazione cifrari polialfabetici Più difficile. Per esempio in Vigenère ogni lettera Y del crittogramma dipende da una coppia di lettere $\langle$X, K$\rangle$ provenienti dal messaggio e dalla chiave.
Se la chiave è di h caratteri, il crittogramma è composto di h sottosequenze, ciascuna ottenuta per sostituzione monoalfabetica. Il problema è scoprire h per scomporre il crittogramma e continuare la decifrazione con il metodo monoalfabetico.
Quindi il messaggio contiene quasi sicuramente gruppi di lettere adiacenti ripetuti più volte. Le apparizioni della stessa sottosequenza allineate con la stessa porzione di chiave danno sottosequenze identiche. Si cercano nel crittogramma coppie di posizioni p_1, p_2 in cui indiziano sottosequenze identiche. La distanza $d = p_2 - p_1$ è *probabilmente* uguale ad h o ad un suo multiplo.

Decifrazione di cifrari a trasposizione Le lettere nel crittogramma sono le stesse del messaggio in chiaro, non ha senso un attacco statistico basato sulle frequenze: si studiano i q-grammi
Se si conosce la lunghezza h della chiave, si divide il crittogramma in porzioni di lunghezza h, in ciascuna si cercano gruppi di q lettere che formano i q-grammi più diffusi. se un gruppo deriva effettivamente da un q-gramma, si scopre parte della permutazione.

Conclusione La rilevazione delle frequenze delle singole lettere del crittogramma è un potente indizio per discernere tra i vari tipi di cifrario:

nei cifrari a trasposizione, l'istogramma delle frequenze coincide circa con quello proprio del linguaggio

nei cifrari a sostituzione monoalfabetica, i due istogrammi coincidono a meno di una permutazione delle lettere

nei cifrari a sostituzione polialfabetica, l'istogramma del crittogramma è assai più appiattito di quello del linguaggio

Capitolo 4

Cifrari perfetti

Shannon, 1949

Cifrario perfetto Sono cifrari che ci offrono una **sicurezza incondizionata**, proteggono le informazioni con certezza assoluta indipendentemente dalla potenza di calcolo. Si rende **essenziale avere la chiave**, un'attacco **brute force non permette di decifrare**.
Informalmente, un cifrario è perfetto se la **sicurezza è garantita qualunque sia l'informazione carpita dal canale di comunicazione**.
Abbiamo quindi due spazi:

MSG: lo **spazio dei messaggi**

CRITTO: lo **spazio dei crittogrammi**

e due variabili aleatorie:

M: descrive il **comportamento del mittente**, con valori in MSG

C: descrive il **processo di comunicazione**, con valori in CRITTO

Ho delle probabilità:

$P(M = m)$: probabilità che il mittente voglia spedire il messaggio m

$P(M = m \mid C = c)$: probabilità condizionata che il messaggio inviato sia m posto che sul canale stia transitando il crittogramma c

Con $\forall\ m \in$ MSG e $\forall\ c \in$ CRITTO.
Scenario: il crittoanalista conosce tutto del sistema tranne la chiave, quindi conosce:

la distribuzione di probabilità con cui il mittente invia i messaggi

cifrario utilizzato

lo spazio delle chiavi (KEY)

Definizione Un cifrario è perfetto se $\forall\ m \in$ MSG e $\forall\ c \in$ CRITTO, allora $P(M = m) = P(M = m \mid C = c)$, cioè la conoscenza di c non raffina la conoscenza. In un cifrario perfetto, la conoscenza del crittoanalista non cambia dopo che è stato osservato un crittogramma in transito. Quindi **il messaggio ed il crittogramma appaiono del tutto scorrelati, nessuna informazione sul messaggio filtra dal crittogramma**.

4.0.1 Teorema di Shannon

In un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale del numero dei messaggi possibili (cioè con probabilità non nulla di essere inviati)

Dim Per assurdo

$N_m = \#$ dei messaggi possibili, cioè $m \in$ MSG $\mid P(M = m) > 0$

$N_k = \#$ delle chiavi

Suppongo per assurdo che $N_k < N_m$. Suppongo $c \in$ CRITTO con $\mathrm{P}(\mathrm{C} = c) > 0$. Conto i messaggi che possono corrispondere al crittogramma $\Rightarrow s$ messaggi potenzialmente corrispondenti a c. Questi s messaggi sono i messaggi che posso ottenere decifrando c con ogni chiave possibile: con la chiave k_1 ottengo m_1, k_2 ottengo m_2, ma anche possibile che con k_3 ottenga di nuovo m_2 dato che non so come funziona il cifrario. Posso dire con certezza che $s \leq N_k$ e per ipotesi $N_k < N_m$
Ottengo $s < N_m \Rightarrow \exists\, m \in$ MSG con $\mathrm{P}(\mathrm{M} = m) > 0 \mid \mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c) = 0$, ottenendo una conoscenza maggiore (posso escludere m) quindi il cifrario non è perfetto, ottenendo una **contraddizione**.

4.0.2 One-Time Pad

1917, Mauborgne, Vernam
Usa l'alfabeto binario {0,1}, usato sia durante le guerre mondiali che durante la guerra fredda. L'idea è avere una chiave lunghissima consumata man mano e mai riutilizzata.
MSG, CRITTO e KEY sono spazi di sequenze binarie. La regola è pubblica e usa lo xor. Quindi sia C che D usano lo xor (somma in modulo 2) per trasformare il messaggio nel crittogramma e viceversa.
$m \in \{0,1\}^n$ e $k \in \{0,1\}^n$ con $n > 0$ fissato.
$c = C(m,k) = m \oplus k$ Con $n = 5$, $m = 10110$, $k = 01011$ allora $c = 11101$.
La decifrazione $m = D(c,k) = c \oplus k = (m \oplus k) \oplus k) = m \oplus (k \oplus k) = m$ perché $k \oplus k = 0 \ldots 0$
Fondamentale non riutilizzare la chiave. Questo perché avendo c', c'' ottenuti con la stessa k da m', m'', faccio $c' \oplus c'' = (m' \oplus k) \oplus (m'' \oplus k) = m' \oplus m'' \oplus (k \oplus k) = m' \oplus m''$, acquisendo conoscenza (lunghi tratti di 0 significa tratti uguali nei messaggi in chiaro).

Teorema Quando il One-Time Pad è perfetto. Ipotesi:

1. Tutti i messaggi hanno lunghezza n, con del padding o divisione in blocchi all'occorrenza

2. Tutte le sequenze di n bit sono messaggi possibili (per quelle prive di senso probabilità molto bassa ma comunque > 0)

3. Chiave scelta **perfettamente a caso per ogni messaggio**

Il **teorema** dice che sotto 1, 2 e 3 One-Time Pad è **un cifrario perfetto e minimale** (usa un numero minimo di chiavi)

Dim Dimostro che è perfetto, cioè che $\forall\, m \in$ MSG e $\forall\, c \in$ CRITTO, allora $\mathrm{P}(\mathrm{M} = m) = \mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c)$
$\mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c) = \frac{P(M=m \wedge C=c)}{P(C=c)} = \frac{P(M=m)\cdot P(C=c)}{P(C=c)} = \mathrm{P}(\mathrm{M} = m)$
Per le proprietà dello xor, fissato m chiavi diverse producono crittogrammi diversi $\Rightarrow \exists!$ chiave $k \mid m \oplus k = c\ \forall\, c \in$ CRITTO, $\mathrm{P}(\mathrm{C} = c) = \mathrm{P}($scegliere l'unica chiave $k \mid m \oplus k = c) = \frac{1}{2^n}$, questo perché $\{\mathrm{M} = m\}$ e $\{\mathrm{C} = c\}$ sono **eventi indipendenti**.
Per quanto riguarda la minimalità, cioè $N_k \geq N_m$, il One-Time Pad per ogni n ha $N_k = N_m = N_{CRITTO} = 2^n$, questo perché anche le chiavi sono sequenze di n bit come i messaggi e i crittogrammi.

Attacchi

Bruteforce: non ha senso, ogni chiave fa ricostruire un messaggio possibile quindi non aggiunge conoscenza

Rimane il problema di generare le sequenze di bit completamente casuali.

Osservazione Rimuoviamo la seconda ipotesi nella dimostrazione. Per esempio in inglese i messaggi significativi sono circa $\alpha^n << 2^n$ con $\alpha = 1.1$
Quindi $N_m = \alpha^n$, $N_k \geq N_m$ cioè $N_k \geq \alpha^n$, la chiave sarà una sequenza di t bit con $t \mid 2^t \geq \alpha^n$
Risolvo trovando $t \geq \log_w \alpha^n = n \log_2 \alpha = 0.12 \cdot n$

Per confondere l'avversario è opportuno che coppie diverse (m,k) producano lo stesso crittogramma $\Rightarrow$ # coppie $(m,k) >>$ # crittogrammi, cioè $\alpha^n 2^t >> 2^n$

4.1 Principi di Shannon

Principi per la resistenza agli attacchi di crittoanalisi statistica:

Diffusione: tutti i caratteri del testo in chiaro si devono spargere nel crittogramma.

Confusione: combinare testo in chiaro e chiave in modo complesso, per non permettere al crittoanalista di sapere le due sequenze analizzando il crittogramma.

Standard 1972 NBS (National Bureau of Standards) → 1973 NIST (National Institute for Security and Technology)

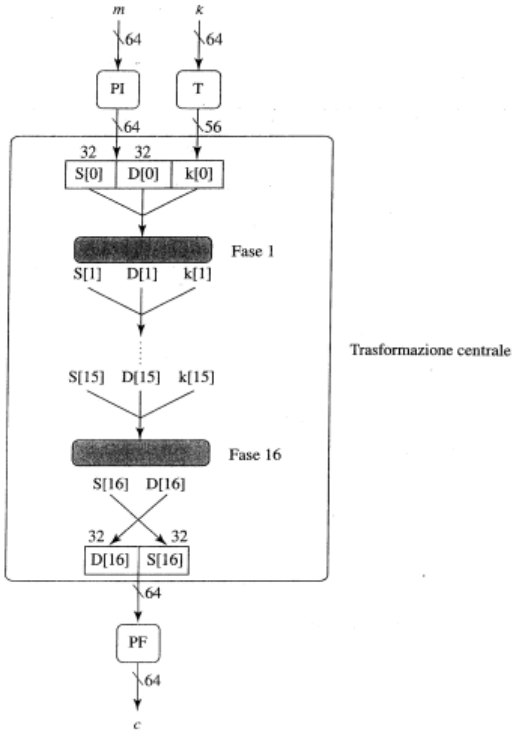
Sicurezza basata sulla segretezza della chiave e non sul processo di cifratura e decifrazione (che è **pubblico**)

Algoritmo efficiente in software e hardware

IBM propone **LUCIFER**, poi migliorato con l'NSA. Da 128 bit a 56 bit per la chiave: variazioni nella s-box.
Si arriva al 1977 con il **DES**, accettato e reso pubblicamente disponibile (licenza d'uso gratuita): **Data Encryption Stantard**. Rimane fino al 1999, quando si sconsiglia il DES in favore del 3DES. Dal 2005 anche il 3DES risulta superato, al suo posto **AES** (**Advanced Encryption Standard**)

4.2 Data Encryption Standard

Cifratura a blocchi di 64 bit e chiave segreta di 64 bit: 56 bit casuali e 8 bit di parità suddivisi in: 7 bit, bit parità, 7 bit, bit parità...
$r = 16$ fasi in cui si ripetono le stesse operazioni.



Struttura del DES

m: blocco del messaggio

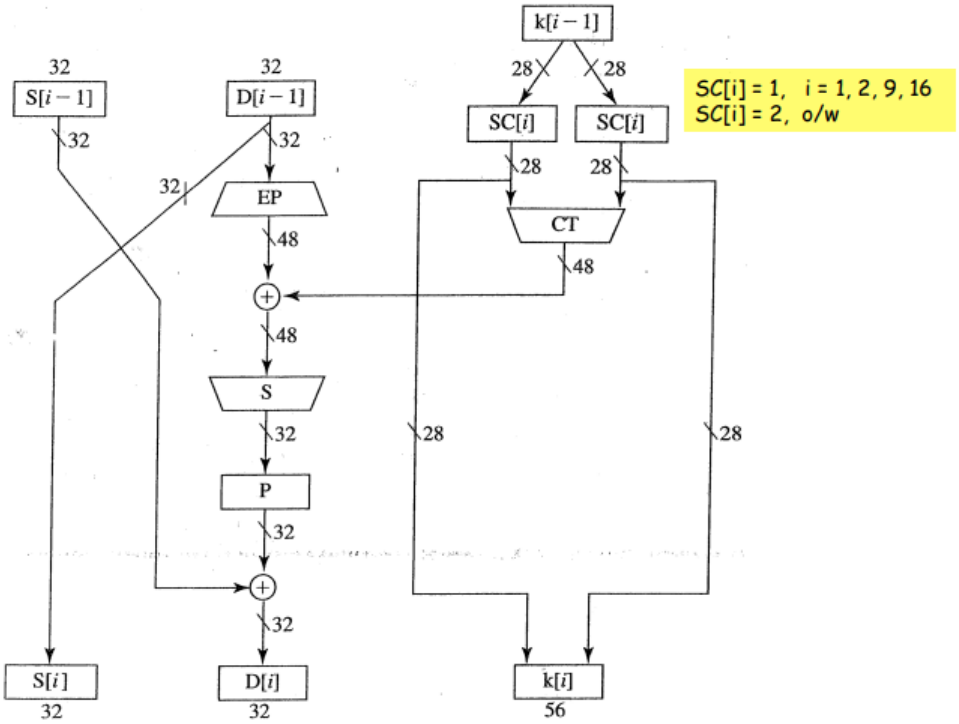
c: corrispondente blocco del crittogramma

k: chiave segreta con i bit di parità

$\forall\, i = 1, \ldots, 16$ ho

$S[i] = D[i-1]$
$D[i] = S[i-1] \oplus f(D[i-1], k[i-1])$
f: funzione **non** lineare

Fase i-esima del DES



Permutazioni

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Permutazione PI

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Permutazione PF

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	52	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Trasposizione T

Le tabelle vanno lette per righe.

Permutazione PI: riordina i bit del messaggio $m = m_1 m_2 \dots m_{64}$ come $m_{58} m_{50} \dots m_7$: ad esempio porta in posizione 40 (numero della cella) il bit in posizione 1 (contenuto della cella)

Permutazione PF: è l'inversa di PI, cioè nell'esempio riporta in posizione 1 il bit in posizione 40

Trasposizione T: provvede anche a scartare dalla chiave $k = k_1 k_2 \dots k_{64}$ i bit di controllo $k_8, k_1 6, \dots k_6 4$, generando una sequenza di 56 bit che costituisce la prima sottochiave $k[0]$

Funzioni CT e EP

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Funzione CT

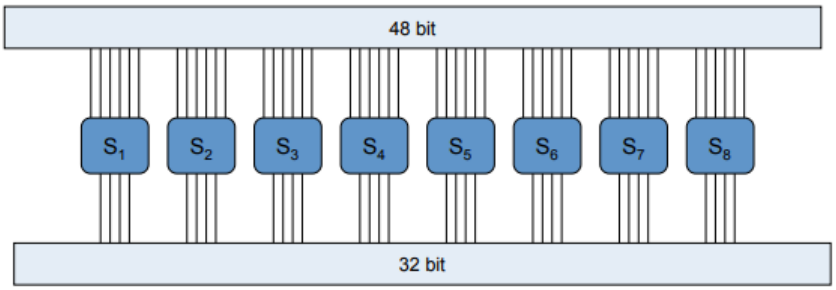
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Funzione EP

Funzione CT: 8 bit dell'ingresso (es: il bit 9) non sono presenti in uscita

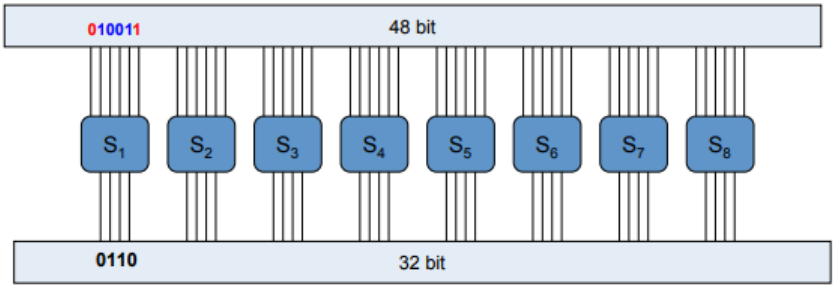
Funzione EP: 16 bit di ingresso sono duplicati (es: il bit 32 è copiato nelle posizioni 1 e 47 dell'uscita)

S-box



S_1	y=0	y=1	y=2	y=3	y=4	y=5	y=6	y=7	y=8	y=9	y=10	y=11	y=12	y=13	y=14	y=15
x=0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
x=1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
x=2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
x=3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Tabella che definisce la sottofunzione S_1. Le sottofunzioni $S_2, \ldots, S_8$ sono definite in modo simile.



S_1	y=0	y=1	y=2	y=3	y=4	y=5	y=6	y=7	y=8	**y=9**	y=10	y=11	y=12	y=13	y=14	y=15
x=0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
→ x=1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
x=2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
x=3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Permutazione P

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Permutazione di 32 bit che genera il blocco finale $D[i]$.

Attacchi Due metodi principali:

1. Architetture apposite progettate per attaccare il DES
2. Calcolo distribuito su più macchine

Attacchi esaurienti **Chosen Plain Text**: il crittoanalista si procura coppie $\langle m, c_1\rangle, \langle \overline{m}, c_2\rangle$

Da $C(m,k) = \begin{array}{l} c_1 \Rightarrow k \textit{ probabile chiave} \\ c_2 \Rightarrow \overline{k} \textit{ probabile chiave} \end{array}$

Crittoanalisi differenziale 2^{47} coppie $\langle m, c\rangle$ scelti dal crittoanalista. L'attacco costa complessivamente 2^{55} con $r = 16$.

Capitolo 5

Esercizi

5.1 Complessità e randomizzazione

Esercizio 1 Un sistema crittografico impiega chiavi private di 46 bit. Per decifrare un messaggio m data la chiave, un programma in assembler impiega un ciclo di 128 istruzioni ripetuto in media tante volte quanti sono i bit che costituiscono m. Impiegando un calcolatore che esegue un'operazione assembler in un tempo medio di 10ns, indicare in ordine di grandezza quanti anni sarebbero necessari in media per condurre un attacco esaustivo sulle chiavi per un messaggio m di 1000 bit.
Indicare i calcoli eseguiti.

Svolgimento

chiavi $= 2^{46}$

operazioni per chiave $= 128 \cdot |m|$

$|m| = 1000$ bit $= 10^3$ bit

1ns $= 10^{-9}$s

$\Rightarrow t = 2^{46} \cdot 128 \cdot 10^3 \cdot 10 \cdot 10^{-9}\text{s} = 2^{46} \cdot 2^7 \cdot 10^{-5}\text{s} = 2^{53} \cdot 10^{-5}\text{s} = 2^3 \cdot 2^{50} \cdot 10^{-5}\text{s}$ e dato che $2^{10} \simeq 10^3$
$\Rightarrow \simeq 2^8 \cdot (10^3)^5 \cdot 10^{-5}\text{s} = 8 \cdot 10^{10}\text{s} \simeq 2500$ anni

Esercizio 2 Si vuole generare una sequenza di bit pseudocasuali utilizzando il generatore BBS basato sulla legge:

$$x_i = x_{i-1}^2 \ mod\ n \quad b_i = 1 \Leftrightarrow x_{m-i} \text{ è } dispari$$

1. **Scegliere** $n = 11 \cdot 23$ e verificare che 11 e 23 soddisfano i requisiti del BBS
2. Sia M la propria matricola. Porre $y =$ M mod 100 e $x_0 = y^2$ mod n e **indicare** una sequenza di 10 bit generati, riportando i calcoli
3. **Discutere** se il generatore è crittograficamente sicuro

Svolgimento

1. Verificare che 11 mod 4 = 3 (ok) e 23 mod 4 = 3 (ok)
   Verificare che $2\lfloor\frac{11}{4}\rfloor + 1 = 2 \cdot +1 = 5$ e $2\lfloor\frac{23}{4}\rfloor + 1 = 11$ siano primi fra loro (sono primi entrambi, ok)
2. Usiamo M = 123456, M mod 100 = 56 e 11·23 = 253

$x_0 = 56^2 \bmod 253 = 100 \to 0$
$x_1 = 100^2 \bmod 253 = 133 \to 1$
$x_2 = 133^2 \bmod 253 = 232 \to 0$
$x_3 = 232^2 \bmod 253 = 188 \to 0$
$x_4 = 188^2 \bmod 253 = 177 \to 1$
$x_5 = 177^2 \bmod 253 = 210 \to 0$
$x_6 = 210^2 \bmod 253 = 78 \to 0$
$x_7 = 78^2 \bmod 253 = 12 \to 0$
$x_8 = 12^2 \bmod 253 = 144 \to 0$
$x_9 = 144^2 \bmod 253 = 243 \to 1$
$\longrightarrow 1000010010$

Esercizio 3 Sia C una sequenza ottenuta rappresentando in binario ciascuna delle due cifre centrali del numero di matricola del candidato, prendendo per ciascuna di esse i tre bit meno significativi, concatenando questi due gruppi di bit e aggiungendo 1 in testa.

1. Eseguire 37^C mod 100 per esponenziazioni successive **indicando** i calcoli eseguiti
2. **Spiegare** perché tale metodo di calcolo è considerato efficiente

Svolgimento Con M = 123456, 3 = 011 e 4 = 100
C = 1011100 = 64 + 16 + 8 + 4 = 92

1. $37^{92} \bmod 100 = 37^{64+16+8+4} \bmod 100$

   $37^4 \bmod 100 = 69^2 \bmod 100 = \underline{61}$
   $37^8 \bmod 100 = 61^2 \bmod 100 = \underline{21}$
   $37^{16} \bmod 100 = 21^2 \bmod 100 = \underline{41}$
   $37^{32} \bmod 100 = 41^2 \bmod 100 = 81$
   $37^{64} \bmod 100 = 81^2 \bmod 100 = \underline{61}$
   $\Rightarrow 37^{92} \bmod 100 = (61 \cdot 41 \cdot 21 \cdot 61) \bmod 100 = 81$

2. Perché esegue un numero di moltiplicazioni logaritmico nel valore dell'esponente (dunque lineare nella dimensione)

Esercizio 4 Applicando l'algoritmo Miller-Rabin, individuare un numero N primo di tre cifre decimali con probabilità di errore minore di $\frac{1}{50}$, spiegando il procedimento eseguito.

Svolgimento $N = 113$, ho $\frac{1}{4^k} < \frac{1}{50}$ per $k = 3$, quindi servono 3 testimoni y scelti a caso $\in [2, 112]$
Pongo $y = 3$

1. MCD(113, 3) = MCD(3, 113 mod 3) = MCD(2, 3 mod 2) = MCD(1, 0) = 1 ok
2. N = 113
   N-1 = 112 = $2^4 \cdot 7$, $w = 4$ e $z = 7$
   $3^7 \bmod 113 = 3^{1+2+4} \bmod 113 =$

   $3^2 \bmod 113 = 9$
   $3^4 \bmod 113 = 9^2 \bmod 113 = 81$

   $= (3 \cdot 9 \cdot 81) \bmod 113 = 40 \neq 1$, bisogna proseguire nella valutazione di P2
   $0 \leq i \leq w - 1 = 3$
   $i = 0$ $3^{2^0 \cdot 7} \bmod 113 = 3^7 \bmod 113 = 40 \neq -1$ (40 preso dal precedente risultato)
   $i = 1$ $3^{2^1 \cdot 7} \bmod 113 = (3^7)^2 \bmod 113 = 40^2 \bmod 113 = 18 \neq -1$ (40 preso dal precedente risultato)
   $i = 2$ $3^{2^2 \cdot 7} \bmod 113 = (18)^2 \bmod 113$ 98 $\neq -1$ (18 preso dal precedente risultato)
   $i = 3$ $3^{2^3 \cdot 7} \bmod 113 = (98)^2 \bmod 113$ -1 $= -1$ ok (98 preso dal precedente risultato)

Per gli altri 2 valori di y il procedimento è analogo

5.2 Cifrari storici

Esercizio 1 **Decifrare** i seguenti crittogrammi

1. YMNXJCJWHNXJNXJFXD (Cifrario Cesare, chiave $k \neq 3$)
2. REXETSIH ONSICESI UCIFTFID REHTLIET (Cifrario a permutazione semplice, $h = 8$)

Svolgimento

1. Con $k = 21$ diventa THISEXERCISEISEASY
2. Con $\pi = \{5, 8, 7, 6, 4, 3, 2, 1\}$ diventa THISEXER CISEISNO TDIFFICU LTEITHER

Esercizio 2 Dato un cifrario affine (mod 26), si fa un attacco di tipo chosen plain-text usando il testo *hahaha*. Il testo cifrato è *nonono*. Determinare la funzione di cifratura.

Svolgimento Un cifrario affine è un cifrario che associa a pos(X) = a·pos(Y) + b
Bisogna trovare $a, b \mid \text{pos}(n) = a\cdot\text{pos}(h) + b \bmod 26 \wedge \text{pos}(o) = a\cdot\text{pos}(a) + b \bmod 26$
con pos(n) = 13, pos(h) = 7, pos(o) = 14 e pos(a) = 0
$\begin{cases} 13 = a \cdot 7 + b \ mod\ 26 \Leftrightarrow a = 11 \\ 14 = a \cdot 0 + b \ mod\ 26 \Leftrightarrow b = 14 \end{cases}$ ottentendo i due parametri $a = 11$ e $b = 14$ che danno il cifrario
pos(Y) = (11·pos(X) + 14) mod 26

Esercizio 3 Se nei cifrari affini si lavorasse in modulo 27 invece che modulo 26, quante sarebbero le chiavi possibili? E in modulo 29?

Svolgimento Con un alfabeto di 27 caratteri, a deve essere primo con $27 = 3^3$
$\rightarrow$ a può assumere i valori da 1 a 26 non multipli di 3
$\longrightarrow \phi(27) = \phi(3^3) = 2 \cdot 3^2 = 18$ (tramite la **funzione di Eulero** $\phi(p^k) = (p-1)p^{k-1}$ con p primo)
$\rightarrow$ b può assumere tutti i valori $\in [0, 26]$
$\Rightarrow$ # chiavi = 18·27 - 1 = 485 (il -1 esclude la coppia $(a, b) = (1, 0)$ che lascia invariato il messaggio)

Con un alfabeto di 29 caratteri: 29 è primo, quindi a può assumere tutti i valori $\in [1, 28]$ ($\rightarrow \phi(29) = 28$) e b tutti i valori $\in [0, 28]$
$\Rightarrow$ # chiavi $= 28 \cdot 29 - 1 = 811$

Esercizio 4 Questo esercizio ha lo scopo di dimostrare che un cifrario affine iterato ha la stessa sicurezza di un cifrario singolo.
Si considerino due cifrari affini:

$$C_1(x) = (a_1 \cdot x + b_1) \bmod 26$$

$$C_2(x) = (a_2 \cdot x + b_2) \bmod 26$$

Dimostrare che $\exists$ un cifrario affine $C_3 \mid C_3(x) = C_2(C_1(x))$

Svolgimento $C_2(C_1(x)) = (a_2 \cdot C_1(x) + b_2) \bmod 26 = (a_2 \cdot (a_1 + b_1) + b_2) \bmod 26$
$= (a_1a_2x + a_2b_1 + b_2) \bmod 26 = (a_3 \cdot x + b_3) \bmod 26$
Con $a_3 = a_1a_2 \bmod 26$, $b_3 = a_2b_1 + b_2 \bmod 26$

Esercizio 5 Il crittogramma c = MBR OJFGA SWNTE CNK QJDIL NURW MBR XHMR è ottenuto cifrando m = THE QUICK BROWN FOX JUMPS OVER THE GATE con un cifrario a sostituzione monoalfabetica completo.

1. Quanta informazione relativa alla chiave si può determinare conoscendo la coppia m, c?
2. Quante chiavi differenti potrebbero essere state usate per cifrare il messaggio m?
3. **Decifrare** il crittogramma MBR TRHLRP WHE HTHV CWND PNEYNE ZNN, che è stato cifrato usando la stessa chiave usata per cifrare m

Svolgimento

1. Parte della permutazione che definisce il cifrario.
   Più precisamente, si trova l'immagine cifrata di 22 caratteri su 26.
2. 24 chiavi
   Infatti mancano le corrispondenze per 4 caratteri, e se ne potrebbero costruire 4! = 24 differenti
3. THE WEASEL RUN AWAY FROM LONDON ZOO

Esercizio 6 Usando il metodo di Vigenère, **cifrare** il messaggio CRITTOGRAFIA impiegando come chiave le prime 4 lettere del proprio cognome. **Spiegare** inoltre come tale cifrario possa essere attaccano.

Svolgimento Si incrocia la lettera del messaggio con la lettera del crittogramma sulla tabella di Vigenère per trovare la lettera cifrata. Ad esempio, alla colonna C e riga M corrisponde la lettera O.

```
  C R I T T O G R A F I A
  M A T T M A T T M A T T
→ O A B M F O Z K M F B T
```

Per la spiegazione, vedi appunti.

Esercizio 7 Si deve cifrare il messaggio APPELLODIFEBBRAIO impiegando come chiave una permutazione arbitraria e segreta delle 26 lettere dell'alfabeto.

1. **Mostrare** la permutazione scelta e il crittogramma ottenuto
2. **Calcolare** il numero di prove necessarie per condurre un attacco esauriente sulle chiavi
3. **Discutere** la possibilità di un attacco più efficiente confrontandolo con quello del punto 2

Svolgimento

1.
2. In generale 26! - 1
   Ma per decifrare il crittogramma ottenuto cifrando APPELLODIFEBBRAIO occorrono meno prove, perché il crittogramma contiene 10 lettere diverse tra loro da decifrare.
   $\Rightarrow$ # prove $= 26 \cdot 25 \cdot \ldots \cdot 17 = \frac{26!}{16!} \simeq 2 \cdot 10^{13}$
3. Crittoanalisi statistica