Complessità computazionale

— ○ —

Complessità e Crittografia

Piero A. Bonatti

Università di Napoli Federico II

Laurea Magistrale in Informatica

Tema della lezione I

- Le funzioni crittografiche per cifrare e decifrare messaggi comunemente utilizzate oggi *non* sono sicure in assoluto
- In teoria è possibile violare la sicurezza pur non conoscendo la chiave privata
- Le garanzie di sicurezza sono basate sulla complessità computazionale:
  - non sono disponibili algoritmi efficienti per "rompere" il cifrario in tempo utile per trarne vantaggio
  - "rompere" il cifrario appartiene a una classe di complessità che si pensa essere strettamente più grande di **P**
- Nel seguito vedremo di quali classi si tratta

Tema della lezione II

- Cifrare un messaggio e decifrarlo *non* sono problemi di decisione
- Richiedono di calcolare una stringa (il testo cifrato o il testo "in chiaro")
- Pertanto prima di discutere la complessità delle funzioni di cifratura e decifratura introduciamo le classi di complessità per il calcolo di funzioni
- Lo facciamo solo per gli analoghi di **P** e **NP** perchè come vedremo le funzioni crittografiche non possono avere *qualunque* complessità
- Le condizioni di esistenza per le funzioni crittografiche, inoltre, dipendono non dall'ipotesi $\mathbf{P} \neq \mathbf{NP}$, ma dall'ipotesi $\mathbf{P} \neq \mathbf{UP}$, dove **UP** è una sottoclasse di **NP**

Tema della lezione III

- Nozioni preliminari
  - Function problems e le classi **FP** e **FNP**
- Funzioni one-way per la crittografia
- Classi di complessità delle funzioni one-way
- Condizioni di esistenza per le funzioni one-way
  - MdT non ambigue e la classe **UP**

Function problems: le classi **FP** e **FNP**

Formalizzare i function problems

- Ricordiamo che un problema di decisione L è in **NP** sse esiste una R_L polynomially balanced e decidibile in tempo polinomiale tale che

$$L = \{x \mid \text{esiste } y \text{ tale che } (x, y) \in R_L\}$$

- Ricordiamo anche che i certificati y, in un modo o nell'altro, rappresentano una soluzione per l'istanza x
  - un ciclo hamiltoniano
  - una colorazione per un grafo
  - un tour il cui costo non supera il budget
  - ...
- Ricordiamo anche che i certificati/soluzioni per un x dato *non sono necessariamente unici*
  - $\Rightarrow$ risolvere un *function problem* significa produrre *una delle soluzioni*
  - non importa quale: una vale l'altra

Le classi **FP** e **FNP**

Definizione: Il *function problem* FL associato a $L \in$ NP

Data una istanza x di L, trovare una stringa y tale che $(x, y) \in R_L$.
Se non esiste, restituire "no".

Definizione di FNP

La classe dei function problem FL tali che $L \in$ **NP**

Definizione di FP

La classe dei function problem FL tali che $L \in$ **P**

- Chiaramente **FP** $\subseteq$ **FNP**
- Mediante oracoli, si definisce un analogo funzionale di **PH**

Riduzioni per i problemi funzionali

- Per ridurre A a B
  - non basta tradurre ogni istanza x di A in una istanza $R(x)$ di B tale che $R(x)$ ha una soluzione sse x ha una soluzione
  - dobbiamo produrre una *soluzione*, per x quindi
  - dobbiamo trasformare la soluzione di $R(x)$ in una per x

Definizione di riduzione per function problem

A è riducibile a B sse esistono due funzioni R e S tali che

1. R e S sono calcolabili in spazio logaritmico
2. se x è una istanza di A allora $R(x)$ è una istanza di B
3. la risposta a x è "no" sse la risposta a $R(x)$ è "no"
4. se y è una soluzione per $R(x)$ allora $S(y)$ è una soluzione per x

Completezza per le classi funzionali

Perfettamente analoga al caso decisionale

Definizione

Un problema A è completo per una classe di problemi funzionali $\mathrm{F}\mathcal{C}$ sse

1. $A \in \mathrm{F}\mathcal{C}$
2. tutti i problemi in $\mathrm{F}\mathcal{C}$ sono riducibili ad A

Completezza per le classi funzionali

Esempio

Definizione: FSAT

Data una espressione booleana ϕ, restituire un truth assignment che la soddisfa, oppure "no" se ϕ è insoddisfacibile

- Si può mostrare che FSAT è **FNP**-completo

Self-reducibility per SAT

Formalizziamo la vecchia corrispondenza tra problemi decisionali e funzionali

Proposizione

FSAT si può risolvere in tempo polinomiale deterministico sse lo stesso vale per SAT

- **Prova (solo se)**: data una MdT deterministica M che risolve FSAT in tempo polinomiale se ne deriva una $(M; M')$ per FSAT:
  - se M termina in h allora M' si sposta in "yes" e termina; altrimenti (M termina in "no") M' termina con "no"

(segue)

Self-reducibility per SAT – II

Formalizziamo la vecchia corrispondenza tra problemi decisionali e funzionali

- **Prova (se)**: dato un algoritmo deterministico A che risolve SAT in tempo polinomiale se ne deriva uno A' per FSAT così:

  Input: $\phi(x_1, ..., x_n)$, $i \in [1, n]$, $T[\,] : bool$

  /* valore iniziale di $i = 0$ */

  1. if $i > n$ return T
  2. if $A(\phi(T[1], .., T[i], true, x_{i+2}, ..., x_n)) =$ "yes"
     - $T[i+1] = true$; return $A'(\phi(x_1, ..., x_n), i+1, T[\,])$
  3. if $A(\phi(T[1], .., T[i], false, x_{i+2}, ..., x_n)) =$ "yes"
     - $T[i+1] = false$; return $A'(\phi(x_1, ..., x_n), i+1, T[\,])$
  4. return "no"

QED

Conseguenze della self-reducibility

Teorema 10.2

FP $=$ **FNP** sse **P** $=$ **NP**

- **Prova (se):** Se **P** $=$ **NP** allora i function problems associati a **P** ed **NP** sono gli stessi, e per definizione **FP** $=$ **FNP**
- **Prova (solo se):** Se **FP** $=$ **FNP** allora FSAT si può risolvere deterministicamente in tempo polinomiale
- Per la Proposizione, anche SAT si può risolvere in tempo polinomiale deterministico
- Siccome un problema **NP**-completo è in **P**, segue che **P** $=$ **NP**

QED

Crittografia e funzioni one-way

Funzioni crittografiche – utilizzo classico

- Ingredienti chiave per la crittografia
  - una funzione di *codifica* (*encoding*) E
  - una funzione di *decodifica* (*dencoding*) D
  - una coppia di *chiavi* d, e usate rispettivamente per la codifica e la decodifica
  - vincolo di correttezza: $D(d, E(e, x)) = x$
- Crittografia *simmetrica* (o a *chiave privata*): $d = e$
- Crittografia *asimmetrica* (o a *chiave pubblica*): $d \neq e$
  - e è pubblica, tutti possono cifrare messaggi
  - d è privata: solo il proprietario può decifrare i messaggi

Vincoli di complessità su E e D

- Cifrare e decifrare messaggi *con le chiavi* deve essere facile
  - E e D sono in **FP**
- "Rompere" il cifrario deve essere difficile

Definizione di $\textsc{attack}(E, e)$

Dati E, e e un messaggio cifrato $y = E(e, x)$ trovare x

- Domanda: quanto possiamo rendere difficile l'attacco?
  - **FNP**? $\mathbf{FNP}^{\mathbf{NP}}$? **FEXP**? ...

Analisi di complessità

Conseguenze dei vincoli su E e D

- Quanto è grande $E(e, x)$?
- $|E(e, x)| \leq |x|^k$
  - perchè in tempo polinomiale E può solo produrre stringhe polinomiali
- C'è un altro vincolo: E non deve perdere informazione (altrimenti non si può ricostruire x)
  - assumendo che x non sia "troppo ridondante", la massima compressione operata da E può essere solo polinomiale
  - quindi $|x| \leq |E(e, x)|^\ell$, ovvero $|x|^{1/\ell} \leq |E(e, x)|$
- Complessivamente $|x|^{1/\ell} \leq |E(e, x)| \leq |x|^k$

Analisi di complessità

Limite superiore alla complessità intrinseca degli attacchi

Proposizione

Se per ogni x, $|E(e,x)| \geq |x|^{1/\ell}$, allora $\text{ATTACK}(E,e)$ è in **FNP**

- **Prova**: Per trovare x dati E, e e $y = E(e,x)$:
  1. generare nondeterministicamente x tale che $|x| \leq |y|^{\ell}$ (tempo polinomiale)
  2. verificare (deterministicamente) se $E(e,x) = y$ (tempo polinomiale)

  QED

- Quindi la complessità massima degli attachi è limitata superiormente !
  - a meno che E non sia magicamente in grado di comprimere gli input più che polinomialmente

Funzioni one-way

Formalizzazione di $E_e(x) = E(e, x)$

Definizione

Una funzione f su stringhe è *one-way* sse

1. f è iniettiva (quindi invertibile) e $|x|^{1/k} \leq |f(x)| \leq |x|^k$
2. f è in **FP**
3. f^{-1} non è in **FP**

- Le funzioni one-way *esistono?*
- Nota: è un requisito necessario ma insufficiente in pratica:
  - si deve calcolare efficientemente f^{-1} con informazione aggiuntiva $(D(d, f(x)) = f^{-1}(y))$
  - l'inversione dev'essere difficile non solo nel caso peggiore, ma nella maggior parte dei casi
  - ...

La classe **UP**

- Vedremo che le funzioni one-way esistono sse $\mathbf{P} \neq \mathbf{UP}$

Definizione di MdT non ambigue e **UP**

Una MdT M è non-ambigua (*unambiguous*) sse ciascun input è accettato *al massimo* da una computazione.

UP è la classe di linguaggi accettati da MdT non ambigue nondeterministiche in tempo polinomiale

- Chiaramente le MdT deterministiche sono non-ambigue $\Rightarrow \mathbf{P} \subseteq \mathbf{UP}$
- Chiaramente $\mathbf{UP} \subseteq \mathbf{NP}$

Sull'esistenza di one-way functions

Teorema 12.1

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **Prova (solo se)**: Supponiamo che f sia una funzione one-way. Dobbiamo mostrare che **UP** $\neq$ **P**.
- Lo faremo mostrando che il linguaggio

$$L_f = \{(x, y) \mid \text{per qualche } z \leq x,\ f(z) = y\}$$

  appartiene a **UP** $\setminus$ **P**
- $\leq$ è un ordinamento in cui le stringhe più corte precedono le più lunghe. A parità di lunghezza, usare ordinamento lessicografico. Ad es. su alfabeto binario

$$\epsilon < 0 < 1 < 00 < 01 < 10 < 11 < 000 < \ldots.$$

(segue)

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **segue prova (se)**: Definiamo una MdT U che decide L_f. Dati x, y:
  1. U genera nondeterministicamente z t.c. $|z| \leq |y|^k$
     - limite dato dalla def. di one-way function, punto 1
  2. controlla se $f(z) = y$ in tempo deterministico polinomiale
     - per la def. di one-way function, punto 2
  3. se si, controlla se $z \leq x$ (tempo polinomiale)
     - succede al massimo per 1 stringa z (f è iniettiva)
     - U è non-ambigua
- U prova che $L_f \in$ **UP**. Resta da dimostrare che $L_f \notin$ **P**

(segue)

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **segue prova (se)**: Supponiamo per assurdo che esista un algoritmo deterministico polinomiale per L_f.
- Mostriamo come invertire f in tempo polinomiale con ricerca binaria
  - questo contraddice il punto 3 della def. di one-way function e conclude la prova

(segue)

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **segue prova (se)**: Calcolo di $f^{-1}(y)$ (es. su alfabeto binario)
  - Controllare se $(1^{|y|^k}, y) \in L_f$. Se no, non esistono z di lunghezza $\leq |y|^k$ t.c. $f(z) = y$. Se si:
  - Ripetere il controllo per $(1^{|y|^k-1}, y)$, $(1^{|y|^k-2}, y)$, ... fino a trovare la lunghezza ℓ di $f^{-1}(y)$
  - Cercare la stringa x t.c. $f(x) = y$ con ricerca binaria.
    - prima controllare se $(01^{\ell-1}, y) \in L_f$.
    - notare che $01^{\ell-1}$ è la stringa $\leq$-massima che inizia per 0: quindi la risposta è si sse il primo bit di x è 0
    - se la risposta è si, provare ricorsivamente se $(001^{\ell-2}, y) \in L_f$.
    - se la risposta è no, provare ricorsivamente se $(101^{\ell-2}, y) \in L_f$.
- Numero di chiamate all'ipotetico algoritmo per L_f: $|y|^k + \ell = O(2n^k)$. Quindi $f^{-1} \in$ **FP** (assurdo) (fine parte "se")

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **Prova (solo se)**: Supponiamo che **UP** $\neq$ **P**, dunque esiste $L \in \mathbf{UP} \setminus \mathbf{P}$. Dobbiamo mostrare che esiste una one-way function
- Sia U la MdT non ambigua nondeterministica che accetta L in tempo polinomiale. Definiamo f_U come segue:

$$f_U(x) = \begin{cases} 1y & \text{se } x \text{ è una computazione di } U \text{ che accetta } y \\ 0x & \text{altrimenti} \end{cases}$$

(y può essere estratta dalla 1^a configurazione in x)

(segue)

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **Prova (solo se)**: Ora basta mostrare che f_U è una one-way function. Cominciamo dal punto 1
- f_U è iniettiva perchè U è non-ambigua
  - non esistono 2 computazioni x e x' che accettano la stessa y
- $|x| \leq |y|^k$ (perchè U usa tempo polinomiale) e $|x| \leq |x|^k$, quindi anche $|x| \leq |f_U(x)|^k$ ovvero

$$|x|^{1/k} \leq |f_U(x)|$$

- $|1y| \leq |x|$ (y è una sottostringa di x) e $|0x| = |x| + 1$, quindi per qualche c

$$|f_U(x)| \leq |x|^c$$

(segue)

Sull'esistenza di one-way functions

Esiste una one-way function se e solo se **UP** $\neq$ **P**

- **segue prova (solo se)**: Punto 2: f_U è calcolabile in tempo polinomiale
  - verificare se x è una computazione di U si può fare in tempo polinomiale ispezionando il programma di U (tempo costante) per ogni passo di computazione (lineari in x)
  - estrarre y da x ha costo lineare
- Punto 3: f_U^{-1} non è calcolabile in tempo polinomiale.
  - Se lo fosse, sarebbe $L \in \mathbf{P}$ (assurdo):
  - $f_U^{-1}(1y)$ è definita $\Leftrightarrow$ U accetta y $\Leftrightarrow$ $y \in L$

QED

Discussione

- Quindi la crittografia si fonda sull'ipotesi che $\mathbf{P} \neq \mathbf{UP}$
  - non $\mathbf{P} \neq \mathbf{NP}$, perchè potrebbe essere $\mathbf{P} = \mathbf{UP} \subset \mathbf{NP}$
- Se potessimo *dimostrare* che una funzione f è one-way, avremmo immediatamente che $\mathbf{P} \neq \mathbf{NP}$
  - da $\mathbf{P} \subset \mathbf{UP} \subseteq \mathbf{NP}$
- Ad oggi, le funzioni che *sembrano* essere one-way functions comprendono

$$f_{MUL}(p, C_p, q, C_q) = \begin{cases} \underline{p \cdot q} & \text{se } C_p, C_q \text{ certificano che } p, q \text{ primi} \\ p, C_p, q, C_q & \text{altrimenti} \end{cases}$$

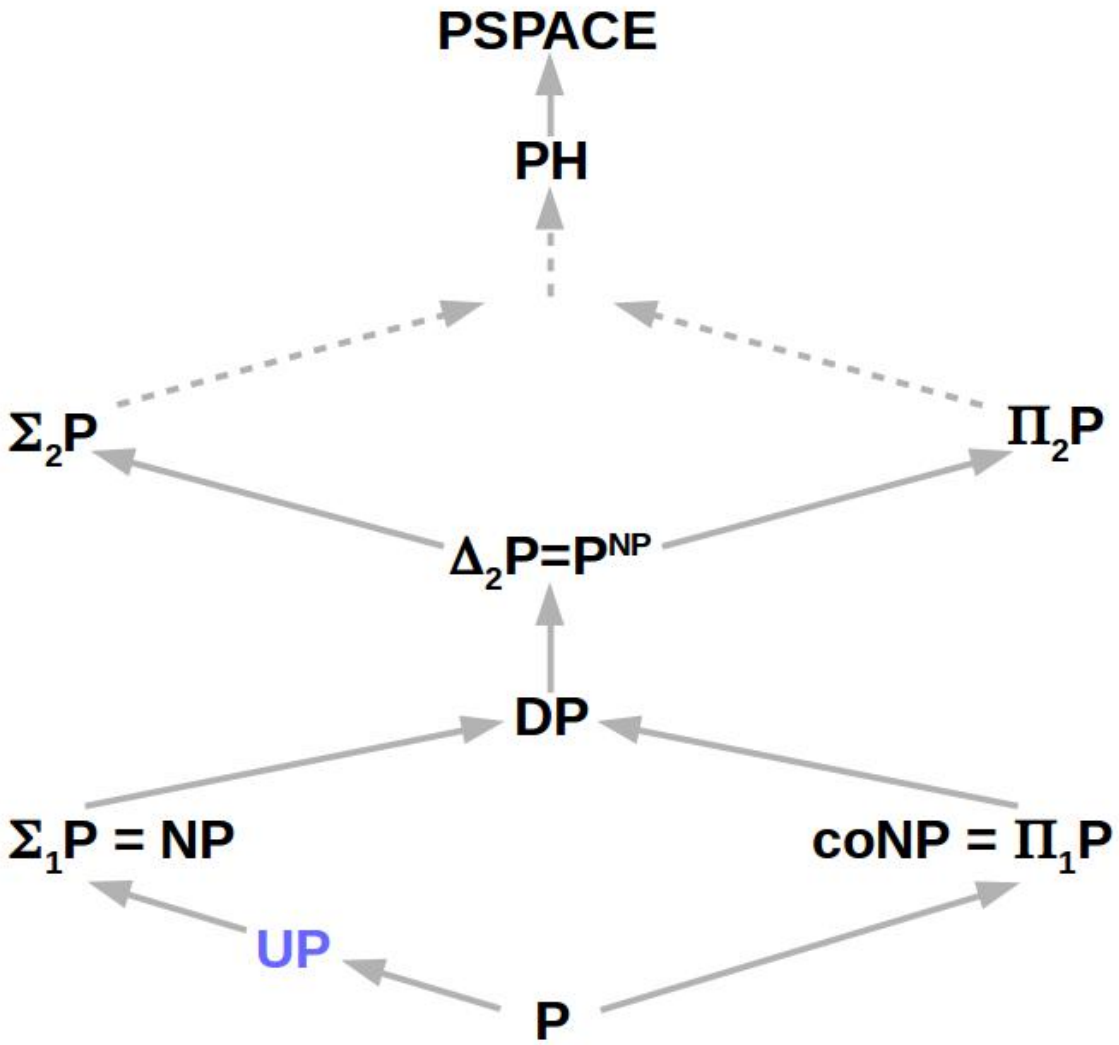
$$f_{EXP}(p, C_p, r, x) = p, C_p, \underline{r^x \bmod p}$$

- Opportunamente combinate definiscono RSA

Materiale di riferimento

- Papadimitriou: Parte III
  - Capitolo 10, parte del paragrafo 3
  - Capitolo 12, parte del paragrafo 1

Riassunto


PSPACE
PH
$\Sigma_2 P$
$\Pi_2 P$
$\Delta_2 P = P^{NP}$
DP
$\Sigma_1 P = NP$
coNP = $\Pi_1 P$
UP
P

Esercizi

1. Supponendo che f_{EXP} sia one-way, dire se STRATEGIC COMPANIES si può ridurre a 2-SAT
2. Se trovassimo un algoritmo deterministico polinomiale per VALIDITY, f_{MUL} potrebbe essere una one-way function?
3. Se scoprissimo che $\mathbf{P} \neq \mathbf{NP}$, sapremmo dire se esistono one-way functions?