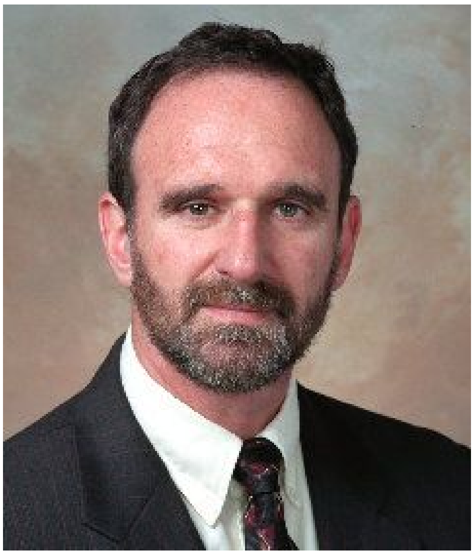
crittografia a chiave pubblica


Whitfield Diffie


Martin Hellman

New Directions in Cryptography

> *We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware . . . has brought the cost of high grade cryptographic devices down to where it can be used in . . . remote cash dispensers and computer terminals. At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science.*

Diffie e Helmann, IEEE IT **22** (1976)

- nel 1976, Diffie e Hellman pubblicano “New directions in cryptography”
- viene considerato l’inizio della crittografia a chiave pubblica (PKC)
- propongono uno schema di scambio della chiave basato sul logaritmo discreto
- introducono l’idea di un crittosistema a chiave pubblica
- la prima realizzazione di un crittosistema di questo tipo si ha nel 1978 con l’RSA

crittografia simmetrica

- Alice e Bob condividono la stessa chiave k – scelta e scambiata fra loro prima di cominciare a comunicare
- la chiave dà luogo a una funzione di cifratura e_k e una funzione di decifratura d_k
- è facile ricavare d_k da e_k (nel DES, l’unica cosa che cambia è l’ordine delle chiavi)
- se si sa cifrare, si sa anche decifrare

idea della crittografia a chiave pubblica

- sviluppare un crittosistema in cui data la funzione di cifratura e_k sia computazionalmente difficile determinare d_k
- Bob rende pubblica la sua funzione di cifratura e_k
- Alice (e chiunque altro) può scrivere a Bob, cifrando il messaggio con la e_k senza bisogno di accordi preliminari
- Bob è l'unico che può decifrare il messaggio
- analogia con un lucchetto, che chiunque può usare, ma di cui solo Bob ha la chiave

funzioni unidirezionali

- bisogna che la funzione di cifratura e sia una funzione unidirezionale (one-way function)
- informalmente, una funzione invertibile $f : \mathcal{P} \to \mathcal{C}$ si dice unidirezionale se
    - dato $x \in \mathcal{P}$, il calcolo di $f(x)$ è facile
    - per quasi tutti gli $y \in \mathcal{C}$ il calcolo di $f^{-1}(y)$ è difficile
    - dato $x \in \mathcal{P}$, il calcolo di $f(x)$ è realizzabile con una complessità polinomiale
    - per quasi tutti gli $y \in \mathcal{C}$ il calcolo di $f^{-1}(y)$ non è realizzabile con una complessità polinomiale (è **NP**-completo?)
- Esempio una funzione ritenuta unidirezionale: sia $n = pq$, p e q numeri primi "abbastanza grandi", b un intero coprimo con $\phi(n)$; sia $f : \mathbb{Z}_n \to \mathbb{Z}_n$ t.c.

$$f(x) = x^b \pmod{n}$$

trapdoor

- se la funzione è unidirezionale, anche per Bob è impossibile decifrare il messaggio
- una trapdoor (botola) one-way function è una funzione unidirezionale che diventa facile da invertire, se si conosce un'informazione supplementare
- l'informazione supplementare viene tenuta segreta da Bob, e usata per decifrare
  - ci sarà una chiave pubblica nota a tutti – che serve per cifrare
  - e una chiave privata nota solo a Bob – che serve per decifrare

- un crittosistema a chiave pubblica non sarà mai a segretezza perfetta
- il testo cifrato $y = e_k(x)$ ci dà informazioni su x
- se conosco e_k - posso provare tutti i plaintext fino a trovare quello che, cifrato, ci dà y
- questo deve essere computazionalmente infattibile

cenni sulla complessità computazionale

- come si misura la complessità computazionale di un algoritmo?
- si parla di algoritmi che operano su numeri interi
- la “grandezza” di un input è pari al numero di bit nella sua rappresentazione binaria ($\approx \log_2 n$)
- la complessità di un algoritmo si misura in termini di operazioni bit
- sono operazioni bit:
    - addizione fra due cifre binarie (es. $0 + 1$);
    - sottrazione fra due cifre binarie (es. $1 - 0$);
    - moltiplicazione fra due cifre binarie (es. $1 \cdot 1$);
    - divisione di un intero a *due* cifre binarie per *una* cifra binaria (es. 10 diviso 1);
    - traslazione a *sinistra* di un posto, cioè moltiplicazione per 2, ad esempio 11 diventa 110, e traslazione a *destra* di un posto, cioè divisione per 2.

complessità polinomiale e esponenziale

Definizione

La complessità computazionale di un algoritmo che opera sugli interi è data dal numero di operazioni bit occorrenti per eseguirlo.

è una funzione (della lunghezza dell'input)

Definizione

Un algoritmo $\mathcal{A}$ per eseguire un calcolo su numeri interi si dice polinomiale se esiste un intero positivo d tale che il numero di operazioni bit necessarie per eseguire l'algoritmo su interi di lunghezza binaria al più k è $\mathcal{O}(k^d)$.

Definizione

Un algoritmo si dice esponenziale se il numero di operazioni bit necessarie per eseguire l'algoritmo su interi di lunghezza binaria al più k è dello stesso ordine di 2^{ck}, per una costante $c > 0$

esempi

- siano n, m interi, $L(n) = k$ (lunghezza binaria), $L(m) = h$, $k \geq h$
- per calcolare la somma $n + m$ servono al più $2k$ operazioni bit quindi $\mathcal{O}(k)$ l'algoritmo che calcola la somma è polinomiale
- per calcolare il prodotto (usando l'algoritmo "tradizionale") servono al più $\mathcal{O}(k^2)$ operazioni bit
- per calcolare il MCD(n, m) usando l'algoritmo di Euclide servono al più $\mathcal{O}(k^2)$ operazioni bit
- per trovare la fattorizzazione di n usando il metodo delle divisioni successive servono $\mathcal{O}(2^{ck})$ operazioni bit

- polinomiale $\leftrightarrow$ (computazionalmente) facile
- esponenziale $\leftrightarrow$ (computazionalmente) difficile
- mostare che esiste un algoritmo polinomiale che risolve un dato problema è semplice - basta descrivere l'algortimo
- ma come si fa a mostare che non esiste un algoritmo polinomiale che risolve un dato problema?

- sia **P** la classe dei problemi P per cui esiste un algoritmo che risolve P in tempo polinomiale (esempio trovare il massimo comun divisore di due numeri interi)
- il problema P appartiene alla classe **NP** se esistono algoritmi (non necessariamente polinomiali) che risolvono il problema ma è possibile verificare, con un algoritmo polinomiale, se un dato assegnato è soluzione o meno del problema (esempio: fattorizzare un intero n)
- si ha $\mathbf{P} \subseteq \mathbf{NP}$
- il problema centrale della teoria della complessità è stabilire se se

$$\mathbf{P} \neq \mathbf{NP}$$

- un problema di **NP** si dice **NP**–*completo* se per ogni problema Q in **NP** esiste un algoritmo polinomiale che riduce la soluzione di Q a quella di P
- quindi se esiste un algoritmo polinomiale che risolve un problema **NP**–completo allora ogni problema di tipo **NP** sarebbe anche in **P**
- quindi si avrebbe $\mathbf{P} = \mathbf{NP}$
- i problemi **NP**–completi sono i problemi *computazionalmente più difficili* tra quelli di tipo **NP**.