## 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
- la funzione di cifratura $e$ è 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

## 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 $c > 0$*

- 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?

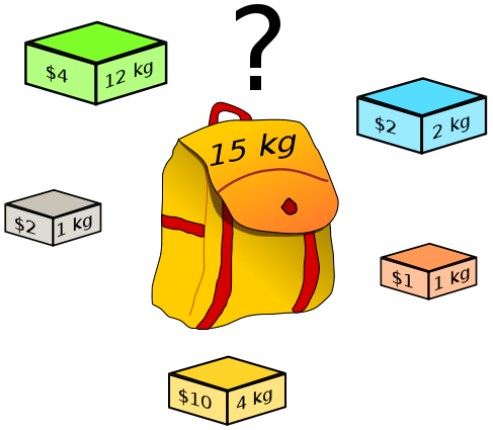
- sia **P** la classe dei problemi $P$ per cui esiste un algoritmo che risolve $P$ in tempo polinomiale
- il problema $P$ appartiene alla classe **NP** se esistono algoritmi (non necessariamente polinomiali) che risolvono il problema ma è possibile verificare, con un algoritmo polinomiale, se un dato 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 $P$ 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**.

## problema dello zaino (knapsack problem)

- un problema ritenuto difficile è il problema dello zaino
- dato uno zaino che può contenere $b$ unità
- e dati $k$ oggetti di volume $a_1, \ldots, a_k$
- il problema è riempire lo zaino usando alcuni oggetti della lista



# problema dello zaino (knapsack problem)

- dobbiamo trovare una $k$-pla $(e_1, \ldots, e_k)$, $e_i = 0, 1$
- tale che $b = \sum_i e_i a_i$
- se possibile – oppure mostare che una tale $k$-pla non esiste
- trovare una soluzione è difficile – procedendo per tentativi ho $2^k$ possibilità da controllare
- ci sono algoritmi migliori, ma nessuno è polinomiale
- ma data una $k$-pla, è facile verificare se è una soluzione
- si può mostrare che è un problema NP-completo

## esempio

- data la lista $(323, 412, 33, 389, 544, 297, 360, 486)$
- e dato $b = 1228$
- un metodo possibile è un procedimento greedy (avido, goloso)
- a ogni passo, prendo l'oggetto più grande che entra ancora nello zaino
- in questo esempio, ho
  $1228 = 544 + 684 = 544 + 486 + 198 = 544 + 486 + 33 + 165$
- e qui ci fermiamo
- invece si ha che $1228 = 412 + 33 + 297 + 486$
- se avessimo una lista di 100 numeri di 50 cifre ognuno il problema diventa inavvicinabile

## sequenze supercrescenti

- ci sono particolare $k$-ple per cui il problema dello zaino è facile
- la sequenza $(a_1, \ldots, a_k)$ si dice supercrescente se ogni termine è maggiore della somma dei termini precedenti
- $\sum_{j=1}^{i-1} a_j < a_i$
- se ho una lista $(a_1, \ldots, a_k)$ supercrescente
- è facile vedere che il procedimento greedy visto prima risolve il problema dello zaino
- e che - se esiste - la soluzione è unica
- per esempio la successione $1, 2, 4, 8, 16, \ldots$ delle potenze di due è supercrescente
- il procedimento greedy è quello che si usa per scrivere un intero in base 2

## sequenze supercrescenti-esempio

- data la lista supercrescente $(1, 3, 7, 15, 31, 63, 127, 255)$
- dato $b = 328$
- $328 = 255 + 73 = 255 + 63 + 10 = 255 + 63 + 7 + 3$

## un crittosistema dal problema dello zaino

- la chiave pubblica di Bob è una lista $(a_1, \ldots, a_k)$ di interi positivi
- per cifrare, il messaggio dev'essere una stringa binaria di lunghezza $k$
- $x = (e_1, \ldots, e_k)$, $e_i = 0, 1$ – Alice calcola $\sum_i e_i a_i = b$
- e trasmette $b$ a Bob
- nessun attaccante può decifrare – bisogna risolvere un problema dello zaino!
- in questo momento, non può decifrare neanche Bob...

## il crittosistema di Merkle–Hellman

- non possiamo usare una sequenza supercrescente come chiave pubblica!
- bisogna "mascherare" questa proprietà
- Bob sceglie una lista $(a_1, \ldots, a_k)$ supercrescente
- sceglie poi un intero $n > \sum a_i$ e un intero $u$ con $(u, n) = 1$
- costruisce una nuova lista $(a_1^*, \ldots, a_k^*)$
- dove $a_i^* = ua_i \pmod{n}$
- $(a_1^*, \ldots, a_k^*)$ è la chiave pubblica
- $(a_1, \ldots, a_k)$, $n$ e $u$ sono la chiave privata

- la cifratura funziona come già detto: Alice deve cifrare $x = (e_1, \ldots, e_k)$ – calcola $\sum_i e_i a_i^* = b^*$ e trasmette $b^*$ a Bob
- per decifrare, Bob calcola $v$, l'inverso di $u$ modulo $n$
- poi calcola $vb^* \equiv b \bmod n$
- $b \equiv vb^* = v \sum e_i a_i^* \equiv v \sum e_i u a_i$ modulo $n$
- $v \sum e_i u a_i = uv \sum e_i a_i \equiv \sum e_i a_i$ modulo $n$
- ora sia $b$ che $\sum e_i a_i$ sono minori di $n = \sum a_i$
- quindi $b = \sum e_i a_i$
- Bob deve risolvere un caso facile del problema dello zaino per decifrare

## esempio

- data la lista supercrescente $(1, 3, 7, 15, 31, 63, 127, 255)$
- Bob sceglie $n = 557$ maggiore della somma dei termini della lista e $u = 323$ coprimo con 557
- moltiplicando per 323 e riducendo mod 557 ottiene $(323, 412, 33, 389, 544, 297, 360, 486)$
- la stringa binaria 01100101 viene codificata da Alice come $412 + 33 + 297 + 486 = 1228$
- Bob riceve 1228 – sa che l'inverso di 323 mod 557 è 169
- calcola $1228 \cdot 169 \bmod 557$ e ottiene 328
- $328 = 255 + 73 = 255 + 63 + 10 = 255 + 63 + 7 + 3$
- riottiene quindi la stringa 01100101

## esempio/testo

- vogliamo cifrare il messaggio `ciao`: serve una tabella di conversione lettere $\to$ stringhe

**Tabella 7.5.** Equivalenti numerici a due cifre e binari

| | | |
|---|---|---|
| a $\longrightarrow$ 00 = 00000 | j $\longrightarrow$ 09 = 01001 | s $\longrightarrow$ 18 = 10010 |
| b $\longrightarrow$ 01 = 00001 | k $\longrightarrow$ 10 = 01010 | t $\longrightarrow$ 19 = 10011 |
| c $\longrightarrow$ 02 = 00010 | l $\longrightarrow$ 11 = 01011 | u $\longrightarrow$ 20 = 10100 |
| d $\longrightarrow$ 03 = 00011 | m $\longrightarrow$ 12 = 01100 | v $\longrightarrow$ 21 = 10101 |
| e $\longrightarrow$ 04 = 00100 | n $\longrightarrow$ 13 = 01101 | w $\longrightarrow$ 22 = 10110 |
| f $\longrightarrow$ 05 = 00101 | o $\longrightarrow$ 14 = 01110 | x $\longrightarrow$ 23 = 10111 |
| g $\longrightarrow$ 06 = 00110 | p $\longrightarrow$ 15 = 01111 | y $\longrightarrow$ 24 = 11000 |
| h $\longrightarrow$ 07 = 00111 | q $\longrightarrow$ 16 = 10000 | z $\longrightarrow$ 25 = 11001 |
| i $\longrightarrow$ 08 = 01000 | r $\longrightarrow$ 17 = 10001 | |

- ciao $\to$ 00010 01000 00000 01110
- i nostri testi in chiaro sono blocchi di lunghezza 8: 00010010 00000000 11101000
- chiave pubblica $= (323, 412, 33, 389, 544, 297, 360, 486)$; la cifratura è la terna $(749, 0, 1312)$

## il crittosistema di Merkle–Hellman

- il crittosistema ora descritto è dovuto a Merkle e a Hellman (1978)
- è elegante – e molto più semplice dell'RSA
- sfortunatamente non è molto sicuro
- A. Shamir, A polynomial-time algorithm for breaking the basic Merkle - Hellman cryptosystem, IEEE Transactions on Information Theory (1984)
- il "mascheramento" non funziona molto bene
- un variante "più sicura" è il crittosistema di Chor-Rivest (1988) che usa il logaritmo discreto