

Crittografia

AlpT (@freaknet.org)

<http://freaknet.org/alpt>

May 6, 2009

Abstract

Questo testo e' una rielaborazione personale degli appunti presi durante il corso di Teoria dei Codici e Crittografia, tenuto dal Prof. Re Riccardo presso il dipartimento di Matematica, Catania, A.A. 2008/2009.

Saro' ben lieto di correggere ogni eventuale errore che mi comunicherai.

Buon lettura.

~ ~

Copyright ©2008 Andrea Lo Pumo aka AlpT <alpt@freaknet.org>. All rights reserved.

This document is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this document; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Contents

1	Varie	1
2	Tempistica	1
3	*	5

1 Varie

Definition 1.1. Con $||[n]_b|$ indichiamo il numero di cifre di n in base b

Proposition 1.2. Sia $n \in \mathbb{N}$, allora

$$|[n]_b| = \lfloor \log_b n \rfloor + 1$$

e' il numero di cifre di n scritto in base b

Proof:

$$n = n_0 + n_1b + n_2b^2 + \dots + n_kb^k$$

$$\text{dove } k = \max k \in \mathbb{N} \mid b^k \leq n \Rightarrow b^k \leq n < b^{k+1} \Rightarrow k \leq \log_b n < k+1 \Rightarrow \lfloor \log_b n \rfloor = k = |[n]_b| - 1 \quad \square$$

Proposition 1.3. Diffie-Hellmann.

Sia F_q un campo finito e $F_q^* = \langle g \rangle$,

1. A sceglie un intero $0 \leq a \leq q-1 = |F_q^*|$
2. B sceglie un intero $0 \leq b \leq q-1$
3. A trasmette a B g^a
4. B trasmette g^b
5. La chiave comune che useranno A e B sara' g^{ab}

A e B sono in grado di calcolare g^{ab} perche' conoscono rispettivamente (a, g^b) e (b, g^a) .
La supposizione di Diffie-Hellmann dice: se C conosce solo g^a e g^b , allora per calcolare g^{ab} , dovra' prima calcolare a e b risolvendo il logaritmo discreto:

$$\log g^b, \log g^a$$

Il logaritmo discreto e' un problema non polinomiale.

Il protocollo Diffie-Hellmann non puo' essere usato senza firma digitale, perche' altrimenti C puo' fare dello spoofing:

$$\begin{aligned} A : g^a &\longrightarrow C \\ C : g^c &\longrightarrow A \\ B : g^b &\longrightarrow C \\ C : g^c &\longrightarrow B \\ \text{chiave tra } A, C : g^{ac}, &\text{ chiave tra } B, C : g^{bc} \end{aligned}$$

2 Tempistica

Definition 2.1. Date due successioni $\{x_n\}, \{y_n\} \subseteq \mathbb{C}$, poniamo

$$x_n = O(y_n) \stackrel{\text{def}}{\iff} \exists C \in \mathbb{R} \exists N \in \mathbb{N} : |x_n| \leq C|y_n| \quad \forall n > N$$

Questa definizione si puo' estendere su funzioni di tipo $\mathbb{N}^m \longrightarrow \mathbb{C}$.
In pratica $f(n) = O(g(n))$ vuol dire che $f(n)$ e' limitata da g , a meno di una costante. Quindi

cresce meno rapidamente di quanto cresce g .
Usiamo la seguente convenzione:

$$a = b + O(y_n) \stackrel{\text{def}}{\Leftrightarrow} a - b = O(y_n)$$

Proposition 2.2.

1. $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k \Rightarrow f(n) = O(g)$
2. $O(kg(n)) = O(g(n))$
3. $O(\log_b n) = O(\log_2 n)$
4. $f(x) \in \mathbb{R}[x] \Rightarrow O(f(n)) = O(n^{\deg f})$

Proof:

(1) Dim 3.

$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

e quindi con la 2. si ha la tesi. □

Definition 2.3. Sia f un algoritmo. Indichiamo con $\text{Time}(f(n))$ il numero di operazione elementari che f impiega per calcolare $f(n)$. Come operazioni elementari consideriamo quelle logiche tra due bit.

Proposition 2.4. Siano $a, b \in \mathbb{N}$

1. $\text{Time}(a + b) = O(\max(\lfloor \log_2 a \rfloor, \lfloor \log_2 b \rfloor))$
2. $\text{Time}(ab) = O(\lfloor \log_2 a \rfloor \lfloor \log_2 b \rfloor)$
3. $\text{Time}(a/b) = O(\max(\lfloor \log_2 a \rfloor, \lfloor \log_2 b \rfloor)^2)$
4. $\text{Time}(\text{MCD}(a, b)) = O(\lfloor \log_2 \rfloor^3)$
5. $\text{Time}(\text{Bezout}(a, b)) = O(\lfloor \log_2 \rfloor^3)$

dove $\text{Bezout}(a, b) = (\lambda, \mu)$ tali che $\text{MCD}(a, b) = \lambda a + \mu b$.

Proof:

(1)

Si adopera il classico algoritmo di addizione: si scrivono le cifre di a e b in due righe e si fanno le addizioni delle cifre (bit) con riporto. Se $a > b$, allora $\text{Time}(a + b) = O(k)$ dove k e' il numero di cifre (bit) di a .

(2)

Si adopera la classica moltiplicazione. Si conta quante addizioni si fanno. Se $a \geq b$ si ottiene $\text{Time}(ab) = h\text{Time}(a + b) = hO(k)O(hk)$. Dove h e' il numero di bit di b .

(3)

Si utilizza l'algoritmo di divisione classico e si conta quante sottrazioni si fanno.

(4)

Si utilizza l'algoritmo euclideo per il MCD e si conta quanti passi si adoperano:

$$\begin{aligned}
a &= bq_1 + r_1 \\
b &= r_1q_2 + r_2 \\
r_1 &= r_2q_3 + r_3 \\
r_2 &= r_3q_4 + r_4 \\
&\vdots \\
r_j &= r_{j+1}q_{j+2} + r_{j+2} \\
&\vdots \\
r_{n-3} &= r_{n-2}q_{n-1} + r_{n-1} \\
r_{n-2} &= r_{n-1}q_n + r_n \\
r_{n-1} &= r_nq_{n+1} + 0 \\
d &= r_n
\end{aligned}$$

(4.1) $r_{j+1} < \frac{1}{2}r_{j-1}$, ovvero r_{j+2} e' piu' che dimezzato rispetto a r_{j-1}

CASE: $r_j \leq \frac{1}{2}r_{j-1}$

$$r_{j-1} = r_jq_{j+1} + r_{j+1} \quad \underbrace{\Rightarrow}_{\text{resto della divisione}} \quad r_{j+1} < r_j \leq \frac{1}{2}r_{j-1}$$

CASE: $r_j > \frac{1}{2}r_{j-1}$

r_{j+1} e' il resto della divisione r_{j-1}/r_j . Ma r_j e' piu' della meta' di r_{j-1} , quindi il quoziente e' $q_{j+1} = 1$ e

$$r_{j-1} = r_jq_{j+1} + r_{j+1} = r_j + r_{j+1} \Rightarrow r_{j+1} = r_{j-1} - r_j < r_{j-1} - \frac{1}{2}r_{j-1} = \frac{1}{2}r_{j-1}$$

Passiamo quindi affermare che ogni due passi r_j e' piu' che dimezzato, quindi il numero di passi t e' $< \log_2 r_1$,

$$r_1 < b \leq a \Rightarrow t < \lfloor \log_2 a \rfloor + 1 = k$$

Ad ogni passo eseguiamo una divisione e una addizione:

$$\begin{aligned}
\text{Time}(MCD(a, b)) &= (\text{Time}(a + b) + \text{Time}(a/b))t < (\text{Time}(a + b) + \text{Time}(a/b))k = (O(k) + O(k^2))k \Rightarrow \\
&\Rightarrow \text{Time}(MCD(a, b)) = O(k^3)
\end{aligned}$$

(5)

L'algoritmo per Bezout consiste nel calcolare $MCD(a, b)$ con Euclide facendo delle sostituzioni ad ogni passo (vedi AlgebraI). Ad ogni sostituzione si effettua una moltiplicazione e una addizione, quindi

$$\text{Time}(\text{Bezout}) = \text{Time}(MCD(a, b)) + (\text{Time}(a + b) + \text{Time}(a/b))t = 2O(k^3) = O(k^3)$$

□

Proposition 2.5. In $\mathbb{Z}_n$, si ha:

1. $\text{Time}(a + b) = O(\log_2 n)$
2. $\text{Time}(ab) = O(\log_2^2 n)$
3. $\text{Time}(a^{-1}) = O(\log_2^3 n)$

Proof:

(1) 1. e 2.

In $\mathbb{Z}_n$ le rappresentazioni minime di a, b sono tali che $|[a]_2|, |[b]_2| \leq |[n]_2|$. Quindi applicando la [2.4,pg.2] si ha la tesi.

(2)

Per calcolare a^{-1} si deve usare l'algoritmo di euclide:

$$\text{sia } b = a^{-1}$$

$$ab = 1 \mathbb{Z}_n \Leftrightarrow ab - \lambda n = 1 \Leftrightarrow ba - \lambda n = 1 \Rightarrow (b, \lambda) = \text{Bezout}(a, n)$$

Quindi per [2.4,pg.2] si ha la tesi.

□

Proposition 2.6. *Algoritmo della quadrature successive.*

Sia $m \in \mathbb{N}$. Per calcolare $b^m \mathbb{Z}_n$ scriviamo m in base due: $m = m_0 + m_1 2 + \dots + m_k 2^k$, $m_i \in \{0, 1\}$, allora

$$b^m = \prod_{i=0}^k b^{m_i 2^i}$$

poiche' $m_i = 0$ oppure $m_i = 1$, possiamo cosi' scrivere il prodotto:

$$a_0 = b, \quad b_0 = \begin{cases} b & m_0 = 1 \\ 1 & m_0 = 0 \end{cases}$$
$$a_{i+1} = a_i^2, \quad b_{i+1} = \begin{cases} b_i a_{i+1} & m_{i+1} = 1 \\ b_i & m_{i+1} = 0 \end{cases}$$

Tutte le moltiplicazioni si intendono in $\mathbb{Z}_n$.

Al massimo effettuiamo $k = \lfloor \log_2 m \rfloor$ moltiplicazioni, quindi per la [2.5,pg.3] si ha:

$$\text{Time} = O(\lfloor \log_2 m \rfloor \lfloor \log_2 n \rfloor^2)$$

Nota:

$$(b, n) = 1 \Rightarrow b^{\varphi(n)} = 1 \mathbb{Z}_n$$
$$m \underbrace{=}_{\text{divisione}} \varphi(n)q + m', \quad m' < \varphi(n)$$
$$\Rightarrow b^m = b^{m'} \mathbb{Z}_n$$

3 *

Index

*, 5

Tempistica, 1

Varie, 1