Università degli Studi di Roma "La Sapienza"

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Matematica "Guido Castelnuovo"

Tesi di Laurea Triennale in Matematica

Modelli computazionali per il calcolo del logaritmo discreto

Candidato: Andrea Reni

Relatore: Prof.ssa Dina Ghinelli
Correlatore: Dott. Daniele A. Gewurz

Anno Accademico 2008 - 2009

Indice

Introduzione

Dato un elemento primitivo γ di un campo finito $GF(\mathrm{q})$, il logaritmo discreto di un elemento non nullo $\omega \in GF(\mathrm{q})$ è il più piccolo intero x tale che

$$\omega = \gamma^x.$$

Oggi calcolare il logaritmo discreto di un elemento appartenente ad un campo è ritenuto uno dei problemi di difficile trattabilità in matematica, insieme, per esempio, alla fattorizzazione in numeri primi di un intero. La difficoltà di questo problema, oltre a destare interesse negli studiosi di algebra, ha implicazioni anche in altri campi, come quello dell'informatica o della crittografia. Difatti, molti sistemi crittografici diverrebbero insicuri se un algoritmo efficiente per trovare il logaritmo discreto venisse scoperto.

Il primo capitolo di questa tesi tratterà le proprietà algebriche di tale problema, arrivando a dare una dimostrazione di come sia possibile calcolare il logaritmo discreto avendo trovato un'esplicitazione polinomiale, la quale però risulterà non essere efficiente in termini di calcolo.

Nel secondo capitolo, invece, verranno analizzati i principali algoritmi ad oggi utilizzati per calcolare il logaritmo discreto quando si lavora in campi di differente ordine. Cominceremo analizzando un algoritmo efficiente solo per campi piccoli, l'algoritmo di *ricerca esaustiva*, e termineremo descrivendo "*il metodo del calcolo dell'indice*", il quale riesce a calcolare il logaritmo discreto con successo probabilistico e con tempo di esecuzione sub-esponenziale, il migliore ad oggi conosciuto. Ovviamente algoritmi di maggiore efficienza potrebbero essere stati scoperti, dal momento che le agenzie governative, come l'americana National Security Agency, trattano tali scoperte come informazioni riservate.

Infine, vedremo perché il problema del logaritmo discreto abbia una così grande importanza in crittografia e come la difficoltà di questo problema abbia ispirato due ricercatori nella realizzazione del primo scambio di chiavi

tramite linee di comunicazioni insicure, così dando vita ad una vera e propria rivoluzione in crittografia.

Capitolo 1

Il problema del logaritmo discreto

"Sie ahnen nicht, wieviel Poesie in der Berechnung einer Logarithmentafel enthalten ist"

"Non avete idea di quanta poesia ci sia nel calcolare una tavola dei logaritmi"

Carl Friedrich Gauss

La ricerca del logaritmo discreto può essere eseguita in qualunque gruppo ciclico finito. Preferiamo, però, da ora in poi, trattare il problema del logaritmo discreto nel gruppo moltiplicativo di un campo finito, ossia nell'insieme degli elementi non nulli del campo. Prima però dobbiamo mostrare che il gruppo moltiplicativo di un campo è ciclico.

1.1 Prime definizioni

Proposizione 1.1 *Sia F un campo e sia H un sottogruppo finito di ordine* n *del gruppo moltiplicativo F^*. Allora H è un gruppo ciclico, ed è costituito da tutte le radici* n*-esime dell'unità di F.*

Dimostrazione: Se H ha ordine n, l'ordine di ogni elemento α di H divide n, sicché α è una radice n-esima dell'unità, ossia una radice del polinomio $X^n - 1$. Tale polinomio ha al più n radici, e pertanto non vi sono altre radici in F. Ne segue che H è l'insieme di tutte le radici n-esime dell'unità in F. È più difficile dimostrare che H è ciclico. A questo scopo, utilizziamo il teorema di struttura per i gruppi abeliani finitamente generati, il quale ci

dice che H è isomorfo a un prodotto diretto di gruppi ciclici:

$$H \approx \mathbb{Z}/(d_1) \times ... \times \mathbb{Z}/(d_k),$$

dove $d_1 \mid d_2 \cdots \mid d_k$ e $n = d_1 d_2 \ldots d_k$. L'ordine di qualunque elemento di questo prodotto divide d_k, poichè d_k è un multiplo comune di tutti gli interi d_i. Pertanto ogni elemento di H è una radice del polinomio $x^{d_k} - 1$ che ha al più d_k radici in F. Ma H contiene n elementi e inoltre $n = d_1 \cdots d_k$. Allora l'unica possibilità è che $n = d_k$, con $k = 1$, e quindi H è ciclico. □

Teorema 1.2 *Sia K un campo di ordine q, con $q = p^r$, p primo e $r \geq 1$. Allora il gruppo moltiplicativo K^* degli elementi non nulli di K è un gruppo ciclico di ordine $q - 1$.*

Dimostrazione: La radice n-esima dell'unità in un campo F è un elemento ζ tale che $\zeta^n = 1$. Pertanto ζ è una radice n-esima dell'unità se e soltanto se ζ è una radice del polinomio

$$X^n - 1,$$

ossia se e soltanto se il suo ordine, come elemento del gruppo moltiplicaitivo F^*, divide n. Gli elementi non nulli di un campo finito con q elementi sono le $(q-1)$-esime radici dell'unità. In un campo questi elementi formano un gruppo ciclico.

Il gruppo moltiplicativo G di ogni campo finito $GF(q)$, q potenza di un primo, è quindi ciclico. Ciò vuol dire che esiste un elemento non nullo $\gamma \in GF(q)$ che genera il sottogruppo G, chiamato elemento primitivo. □

Diamo ora la definizione del *problema del logaritmo discreto*, tema centrale di questa tesi.

Definizione 1.3 ***(il problema del logaritmo discreto)*** *Dato un elemento primitivo di $GF(q)$ e un qualunque elemento $\omega \in GF(q)^* = GF(q)\backslash\{0\}$, il logaritmo discreto di ω di base γ è il più piccolo intero x per il quale*

$$\omega = \gamma^x.$$

Il problema del logaritmo discreto è il problema di trovare un tale esponente x che soddisfi l'equazione. Scriveremo $x = \log_\gamma \omega$.

Osservazione Si vede subito che se esiste una soluzione per il problema del logaritmo discreto, ne esistono infinite altre. Infatti tramite il Piccolo Teorema di Fermat, sia p l'ordine del campo, vediamo che $\gamma^{p-1} \equiv 1 \pmod{p}$. Quindi se x è una soluzione di $\gamma^x = \omega$, allora $x + k(p-1)$ è una soluzione per ogni valore di k intero, dato che

$$\gamma^{x+k(p-1)} = \gamma^x \cdot (\gamma^{p-1})^k \equiv \omega \cdot 1^k \equiv \omega \pmod{p}$$

Definiamo quindi il $\log_\gamma \omega$, modulo $p-1$, di modo che la soluzione del problema sia unica.

Si verifica che $\log_\gamma$ induce una funzione ben definita

$$\log_\gamma : \mathbb{F}_p^* \longrightarrow \mathbb{Z}/(p-1)\mathbb{Z}$$

e gode della proprietà

$$\log_\gamma ab = \log_\gamma a + \log_\gamma b \qquad \forall a, b \in \mathbb{F}^*{}_p. \tag{1.1}$$

Quindi chiamarlo logaritmo è più che ragionevole, dato che converte la moltiplicazione in addizione nello stesso modo della funzione logaritmo usuale.

Bisogna però puntualizzare che il logaritmo discreto a differenza della sua controparte continua nei reali e nei complessi, ha un comportamento assai irregolare. La terminologia è comunque ragionevole, dato che in entrambi i casi è l'inverso dell'elevazione a potenza e gode della proprietà (1.1).

Osservazione Un vecchio termine per il logaritmo discreto è *indice*, denotato da $ind_\gamma \omega$. La terminologia indice è ancora comunemente usata in teoria dei numeri. È inoltre conveniente se esiste una confusione tra logaritmi ordinari e logaritmi discreti, dato che, per esempio, la quantità $\log_2$ frequentemente occorre in entrambe i contesti.

1.2 Logaritmo di Zech

Se si rappresentano gli elementi non nulli di $GF(q^n)$ come potenze di un elemento primitivo, la moltiplicazione nel gruppo ciclico G degli elementi non nulli di $GF(q^n)$ è banale, mentre l'addizione diventa difficile.

Descriveremo brevemente uno schema chiamato *Logaritmo di Zech*, il quale ci aiuterà ad ovviare a tale problema in alcuni casi particolari.

Sia γ un elemento primitivo fissato appartenente a $F = GF(q)$, q potenza di un primo. Per ogni elemento ω di F^*, definiamo il logaritmo di ω l'unico intero c con $0 \leq c \leq q-2$ che soddisfi $\gamma^c = \omega$ e scriviamo $c = \log_\gamma \omega$. Poniamo inoltre $\log_\gamma 0 := \infty$. Se adesso identifichiamo gli elementi di F con il loro logaritmo discreto, moltiplicare due elementi di F sarà banale dato che in questo modo, ovviamente, ci si riduce all 'addizione del corrispondente logaritmo discreto, come già visto sopra

$$\log_\gamma \alpha\beta = \log_\gamma \alpha + \log_\gamma \beta \tag{1.2}$$

Ovviamente, l'addizione (1.2) è eseguita modulo $q-1$, con la convenzione $\infty + c = \infty$. Se vogliamo eseguire addizioni con queste rappresentazioni, avremo bisogno di determinare il logaritmo discreto di $\alpha + \beta$ per $\alpha, \beta \in F$. Senza ledere generalità, possiamo assumere $\alpha, \beta \neq 0$.

Si noti che $\gamma^c + \gamma^d = \gamma^c(1+\gamma^{d-c})$ quindi si possono sommare due elementi di F^* rappresentati dai loro logaritmi discreti c e d sommando c al logaritmo discreto di $1 + \gamma^{d-c}$.

Perciò è sufficiente essere capaci di determinare il logaritmo discreto per somme un cui addendo è 1. Questo motiva la *definizione del logaritmo di Zech*:

$$Z(e) \text{ è il logaritmo discreto di } 1 + \gamma^e. \text{ Quindi } 1 + \gamma^e = \gamma^{Z(e)}.$$

Usare il logaritmo discreto insieme al logaritmo di Zech è utile in applicazioni dove sono richieste ripetute computazioni su un campo finito di ordine relativamente piccolo, dato che poi i logaritmi di Zech possono essere calcolati precedentemente e, quando servono per calcolare l'addizione, ricercati in una semplice tabella.

Esempio: Si consideri il campo $\mathbb{F}_{32}$ definito dal polinomio irriducibile $f(x) = 1 + x^2 + x^5$ in $\mathbb{F}_2[x]$ e sia α una radice del polinomio. La tavola di logaritmi di Zech per il campo $\mathbb{F}_{32}$ è mostrata nella Tabella 1.1. Usando tale tebella l'addizione nel campo diventa banale. Per esempio

$$\alpha^7 + \alpha^{12} = \alpha^7(1 + \alpha^5) = \alpha^{7+Z(5)} = \alpha^9$$

□

I logaritmi di Zech posso essere utilizzati, inoltre, per semplificare la soluzione di equazioni quadratiche e cubiche. Si noti che ognuna di tali equazioni può essere messa nella rispettiva forma standard $x^2 + x + c = 0$ o $x^3 + x + c = 0$ tramite sostituzioni opportune. È chiaro che una tale tabella di logaritmi di Zech diventa non pratica per campi finiti grandi. Perciò la possibilità di usare questo tipo di rappresentazioni per campi grandi dipende dalla possibilità di calcolare dei logaritmi discreti, che in generale è ritenuto un problema assai arduo.

In molti casi dove $q = 2^m$, trovare radici di polinomi quadratici e cubici su $\mathbb{F}_q$ è relativamente semplice se abbiamo una tabella di logaritmi di Zech per il campo.

Ogni equazione quadratica

$$ax^2 + bx + c = 0 \qquad a, b, c \in \mathbb{F}_q, \qquad a \neq 0, b \neq 0,$$

può essere trasformata in un'equazione quadratica della forma

$$y^2 + y + d = 0 \tag{1.3}$$

tramite la sostituzione $x = \frac{b}{a}y$. Se $y = \alpha^i$ è una radice di (1.3) allora

$$\alpha^{2i} + \alpha^i + d = 0 \qquad \text{cioè} \qquad \alpha^{i+Z(i)} = d$$

i	$Z(i)$	i	$Z(i)$	i	$Z(i)$
0	∞	11	19	22	7
1	18	12	23	23	12
2	5	13	14	24	15
3	29	14	13	25	21
4	10	15	24	26	28
5	2	16	9	27	6
6	27	17	30	28	26
7	22	18	1	29	3
8	20	19	11	30	17
9	16	20	8	∞	0
10	4	21	25		

Tabella 1.1: Tabella di logaritmi di Zech del campo $\mathbb{F}_{32}$

Se $d = \alpha^k$ allora $i + Z(i) \equiv k \pmod{q-1}$ e le radici di (1.3) possono essere trovate risolvendo tale congruenza. Similmente qualunque equazione cubica

$$ax^3 + bx^2 + cx + d = 0 \qquad a, b, c, d \in \mathbb{F}_q, \qquad a \neq 0, \qquad b^2 + ac \neq 0$$

può essere messa nella forma

$$y^3 + y + e = 0 \tag{1.4}$$

tramite la sostituzione $x = \frac{b}{a} + \frac{(b^2+ac)^{1/2}}{a} y$. Se $y = \alpha^i$ è una radice dell'equazione (1.4) e $e = \alpha^k$ allora

$$\alpha^{3i} + \alpha^i = \alpha^k \qquad \text{cioè} \qquad \alpha^i \alpha^{Z(2i)} = \alpha^k.$$

Cerchiamo un tale i tale che

$$i + Z(2i) \equiv k \qquad (\text{mod } q-1)$$

o, equivalentemente (dato che $Z(2i) = 2Z(i)$)

$$i + 2Z(i) \equiv k \qquad (\text{mod } q-1).$$

Per campi grandi questi metodi sono ovviamente impraticabili.

1.3 Esplicitare il Logaritmo Discreto

Un risultato interessante, ma non molto pratico, per calcolare logaritmi in campi finiti è quello di *rappresentare polinomialmente la funzione logaritmo*. Mullen e White [10] ne diedero una formulazione esplicita, la dimostrazione che presentiamo qui appartiene a Niederreiter [11].

Iniziamo questa discussione dimostrando che ogni funzione f da $\mathbb{F}_q$ in $\mathbb{F}_q$ può essere rappresentata da un polinomio in $\mathbb{F}_q$. Sia γ un generatore per $\mathbb{F}_q$, si costruiscano equazioni della forma

$$\begin{aligned} f(0) &= a_0 \\ f(\gamma^k) &= \sum_{i=0}^{q-1} a_i (\gamma^k)^i \end{aligned} \tag{1.5}$$

con $0 \leq k \leq q-2$.

Abbiamo così un sistema in q equazioni e q incognite (i coefficienti a_i) con un'unica soluzione. Questo prova l'asserzione.

Tali a_i saranno i coefficienti del polinomio cercato

$$f(x) = \sum_{i=0}^{q-1} a_i x^i.$$

Ora, sia $q = p^r$ e per ogni j, $0 \leq j \leq q-2$, sia

$$j = \sum_{i=0}^{r-1} \lambda_i^{(j)} p^i, \qquad \text{con } \lambda_i^{(j)} \in \{0, 1, 2, \ldots, p-1\}.$$

Semplicemente abbiamo espresso ogni j in base p. Vedendo $\mathbb{F}_q$ come uno spazio vettoriale su $\mathbb{F}_p$, possiamo rappresentare gli elementi di $\mathbb{F}_q$ come r-uple su $\mathbb{F}_p$. Se $l(\gamma^j) = j$ con $0 \leq j \leq q-2$, è la funzione logaritmo allora definiamo

$$f(\gamma^j) = (\lambda_0^{(j)}, \ldots, \lambda_{r-1}^{(j)}) \in \mathbb{F}_q$$

$$f(0) = (p-1, \ldots, p-1) \in \mathbb{F}_q$$

Dato che f può essere rappresentato da una funzione polinomiale, ne segue che la funzione logaritmo può essere rappresentata da una funzione polinomiale. Come sopra sia γ un generatore per $\mathbb{F}_q$, $q = p^r$, e sia $y = \log_\gamma a$ per qualche $a \in \mathbb{F}_q$, $a \neq 0$. Allora possiamo scrivere

$$y = \sum_{i=0}^{r-1} y_i p^i \qquad \text{con } y_i \in \{0, 1, 2, \ldots, p-1\}.$$

Se possiamo trovare y_0, allora, dato che $y = y_0 + pt$, riduciamo il problema alla ricerca di t come

$$t = \log_\gamma b$$

avendo definito b come

$$b = (\gamma^{-y_0} a)^{1/p} = (\gamma^{pt})^{1/p} = \gamma^t.$$

Ripetendo la procedura si determinano $y_1, y_2, y_3, \ldots, y_{r-1}$. Quindi, è sufficiente mostrare che possiamo trovare una rappresentazione polinomiale per y modulo p. Abbiamo bisogno dei seguenti risultati per la dimostrazione di Niederreiter. (Si supponga $0^0 = 1$)

Lemma. 1.4 *Per interi $j \geq 0$ abbiamo*

$$\sum_{\zeta \in \mathbb{F}_q} \zeta^j = \begin{cases} 0 & , \text{ se } j = 0 \text{ o } j \neq 0 \pmod{q-1} \\ -1 & , \text{ altrimenti} \end{cases}$$

Lemma. 1.5 *Se $q \geq 3$ e k è un intero con $0 \leq k \leq q-1$, allora*

$$\sum_{\zeta \in \mathbb{F}_q, \zeta \neq 1} \frac{\zeta^k}{1-\zeta} \quad = \quad k \quad \in \mathbb{F}_p.$$

Dimostrazione: Per $k = 0$ il risultato è banale. Sia

$$S_k = \sum_{\zeta \in \mathbb{F}_q, \zeta \neq 1} \frac{\zeta^k}{1-\zeta} \qquad k = 0, 1, \ldots, q-1.$$

Per $1 \leq k \leq q-1$ abbiamo

$$\begin{aligned} S_k = \sum_{i=1}^{k} (S_i - S_{i-1}) &= \sum_{i=1}^{k} \sum_{\zeta \in \mathbb{F}_q, \zeta \neq 1} \frac{\zeta^{i-1}(\zeta - 1)}{1-\zeta} \\ &= -\sum_{i=1}^{k} \sum_{\zeta \in \mathbb{F}_q, \zeta \neq 1} \zeta^{i-1} \\ &= -\sum_{i=1}^{k} \Big(\big(\sum_{\zeta \in \mathbb{F}_q} \zeta^{i-1} \big) - 1 \Big) \\ &= -\sum_{i=1}^{k} (-1). \end{aligned}$$

L'ultima equazione segue dal Lemma 1.4. Infine $S_k = -\sum_{i=1}^{k}(-1) = k$, e dato che k è un intero abbiamo che $k \in \mathbb{F}_p$.

□

Diamo ora la dimostrazione di Niederreiter del risultato di Mullen e White.

Teorema 1.6 *Sia γ un generatore per $\mathbb{F}_q$. Per ogni $a = \gamma^y \in \mathbb{F}_q^*$, $q \geq 3$, abbiamo*

$$y \equiv -1 + \sum_{i=1}^{q-2} \frac{a^i}{\gamma^{-i} - 1} \pmod{p}.$$

Dimostrazione: Se $a = \gamma^y$ e $\zeta = \gamma^i$ allora $\zeta^y = \gamma^{iy} = a^i$. Preso $k = 1 + y$ nel Lemma 1.5 abbiamo

$$\sum_{\zeta \in \mathbb{F}_q,\ \zeta \neq 0,1} \frac{\zeta^{1+y}}{1-\zeta} \equiv 1 + y \pmod{p}$$

quindi

$$\begin{aligned} y &\equiv -1 + \sum_{\zeta \in \mathbb{F}_q,\ \zeta \neq 0,1} \frac{\zeta^{1+y}}{1-\zeta} \\ y &\equiv -1 + \sum_{\zeta \in \mathbb{F}_q,\ \zeta \neq 0,1} \frac{\zeta^{y}}{\zeta^{-1}-1} \\ y &\equiv -1 + \sum_{i=1}^{q-2} \frac{a^i}{\gamma^{-i}-1} \pmod{p} \end{aligned}$$

□

1.4 Osservazioni generali

Due gruppi ciclici di ordine n sono sempre isomorfi, ciò non vuol dire, però, che un efficiente algoritmo per calcolare il logaritmo discreto in uno di essi implica un efficiente algoritmo per l'altro. L'affermazione è ovvia quando si considera che ogni gruppo ciclico di ordine n è isomorfo al gruppo additivo $\mathbb{Z}_n$ e calcolare logaritmi in $\mathbb{Z}_n$ è abbastanza facile se si utilizza l'algoritmo euclideo. Quindi, il problema del logaritmo discreto può essere riformulato in questo modo:

Determinare un algoritmo computazionalmente efficiente per calcolare un isomorfismo da un gruppo ciclico di ordine n al gruppo ciclico additivo $\mathbb{Z}_n$.

Ci sono molti modi per rappresentare un campo finito con q^n elementi, tutti isomorfi.

Siano F_1 e F_2 campi finiti di ordine q^n generati dai polinomi irriducibili $f(x)$ e $g(x)$ rispettivamente. Si considerino F_1 e F_2 come spazi vettoriali sopra $GF(q)$. Sia $f(\alpha) = 0$ e $g(\beta) = 0$, e si supponga che gli elementi in F_1 siano rappresentati mediante la base $\{1, \alpha, \ldots, \alpha^{n-1}\}$ e F_2 da $\{1, \beta, \ldots, \beta^{n-1}\}$. Supposto che esista un algoritmo efficiente per calcolare i logaritmi in F_2

rispetto alla base β possiamo ridurre il problema di trovare logaritmi in F_1 a quello di trovare logaritmi in F_2. Per fare ciò, abbiamo bisogno di trovare una radice del polinomio $f(x)$ nel campo F_2. Se $r = \sum_{i=0}^{n-1} b_i \beta^i$ e $f(r) = 0$, allora poniamo $T(\alpha) = r$ e questo può essere usato per definire una trasformazione lineare T da F_1 a F_2, che manda potenze di α in potenze di r. Dato che

$$\frac{\log_\beta T(w)}{\log_\beta T(\alpha)} = \log_{T(\alpha)} T(w) = \log_\alpha w,$$

ne segue che se il $\log_\beta T(\alpha) = x$, allora $\log_\alpha w = (\log_\beta T(w))/x$.

Capitolo 2

Algoritmi per il calcolo del logaritmo discreto

"Je n'ai pas le temps..."

Evariste Galois

In questo capitolo tratteremo alcuni algoritmi utilizzati per calcolare, tramite elaboratori, il logaritmo discreto di un elemento di un gruppo.

Sia γ un generatore per $GF(q)^*$, vogliamo calcolare il logaritmo di ω, sempre appartenente a $GF(q)^*$, di base γ.

Gli algoritmi verranno esaminati in relazione al loro tempo di esecuzione, quindi per la loro efficienza.

2.1 Notazioni

Si supponga di stare cercando di risolvere un problema matematico, il cui input ha grandezza variabile. Come esempio, si consideri il problema della fattorizzazione in numeri primi di un intero, il cui input è un numero N mentre l'output è un fattore primo di N. Vogliamo sapere quanto tempo si impiega per risolvere il problema in relazione alla grandezza dell'input.

Definizione 2.1 *Sia $A > 0$ una costante, indipendente dalla grandezza dell'input, tale che se l'input è lungo $O(k)$ bit, allora ci vogliono $O(k^A)$ passi per risolvere il problema. Allora si dice che il problema è risolvibile in* tempo polinomiale.

Se prendiamo $A = 1$, allora si dice che il problema è risolvibile in tempo lineare*, e se prendiamo $A = 2$, allora si dice che il problema è risolvibile in* tempo quadratico*. Gli algoritmi il cui tempo è polinomiale sono considerati algoritmi veloci.*

D'altra parte, se esiste una costante $c \geq 0$ tale che per un input di grandezza $O(k)$, esiste un algoritmo che risolve il problema in $O(e^{ck})$ passi, allora si dice che il problema è risolubile in tempo esponenziale*. Gli algoritmi a tempo esponenziale sono considerati lenti.*

Tra gli algoritmi a tempo esponenziali e quelli a tempo polinomiali, esistono, inoltre, gli algoritmi a tempo sub-esponenziale*. Questi hanno la proprietà che, per ogni $\varepsilon \geq 0$, risolvono il problema in $O(e^{\varepsilon k})$ passi.*

Per calcolare i tempi di esecuzione degli algoritmi useremo i seguenti risultati. Per un intero a, definiamo la sua lunghezza in bit *length*, ossia il numero di bit nella rappresentazione binaria di a, denotata con $len(a)$, più precisamente

$$len(a) := \begin{cases} \lfloor \log_2 a \rfloor + 1 & \text{se } a \neq 0 \\ 1 & \text{se } a = 0 \end{cases}$$

Si noti che se $len(a) := l$ allora $\log_2 a < l \leq \log_2 a + 1$, o equivalentemente, $2^{l-1} \leq a < 2^l$.

Teorema 2.2 *Siano a e b interi arbitrari.*

(i) Possiamo calcolare $a \pm b$ in $O(len(a) + len(b))$

(ii) Possiamo calcolare $a \cdot b$ in $O(len(a) \cdot len(b))$

(iii) Se $b \neq 0$, possiamo calcolare il quoziente $q := \lfloor a/b \rfloor$ e il resto $r := a \bmod b$ in $O(len(b)len(q))$.

La parte (iii) del teorema è dimostrata in Shoup [4].

Vediamo ora, brevemente, come si calcolano i tempi di esecuzione degli algoritmi per i campi finiti. Sia n un intero positivo. Per ogni $\alpha \in \mathbb{Z}_n$, esiste un unico intero $a \in \{0, 1, \ldots, n-1\}$ tale che $\alpha = [a]_n$; chiamiamo questo intero a la *rappresentazione canonica* di α. Rappresenteremo gli elementi di $\mathbb{Z}_n$ tramite la loro rappresentazione canonica $[\alpha]$.

L'addizione e la sottrazione in $\mathbb{Z}_n$ può essere fatta in $O(len(n))$: dati $\alpha, \beta \in \mathbb{Z}_n$, per calcolare $[\alpha + \beta]$, dobbiamo prima calcolare la somma in $\mathbb{Z}$ $[\alpha] + [\beta]$, poi ridurla modulo n se il risultato è uguale o maggiore di

n; similmente si calcola $[\alpha - \beta]$, sempre tramite riduzione modulo n. La moltiplicazione in $\mathbb{Z}_n$ può essere calcolata in $O(len(n)^2)$, calcoliamo $[\alpha \cdot \beta]$ come $[\alpha] \cdot [\beta] \bmod n$, usando la moltilpicazione in $\mathbb{Z}$ e la divisione con il resto.

Un altro problema interessante è l'elevamento a potenza in $\mathbb{Z}_n$: siano $\alpha \in \mathbb{Z}_n$ e e un intero non negativo, calcolare $\alpha^e \in \mathbb{Z}_n$. Il metodo più ovvio sarebbe moltiplicare iterativamente α e volte, il cui tempo richiesto sarebbe $O(e \cdot len(n)^2)$. Per valori piccoli di e, questo procedimento va bene, ma esiste un algoritmo, l'algoritmo *repeated-squaring* o *square-and-multiply*, che calcola α^e con $O(len(e))$ moltiplicazioni in $\mathbb{Z}_n$, quindi impiega $O(len(e) \cdot len(n)^2)$. Tale algoritmo calcola velocemente potenze di numeri interi, rappresentando l'esponente in forma binaria. Vediamo un

Esempio: Si voglia calcolare 3^{13}. La rappresentazione binaria di 13 è $(1101)_2$, quindi $8 + 4 + 1 = 13$

$$3^{13} = 3^1 \cdot 3^4 \cdot 3^8 = 1594323$$

□

Per primi tratteremo gli algoritmi i cui tempi di esecuzione sono esponenziali. Algoritmi di questo tipo sono inefficienti per campi di ordine molto grande. Comunque, per certi casi particolari (campi di ordine relativamente piccolo) non sono così malvagi.

2.2 Algoritmi di collisione

Di seguito saranno analizzati alcuni *algoritmi di collisione*, ovvero algoritmi dove vengono confrontati valori appartenenti a delle tabelle. Tali algoritmi terminano quando avviene una *collisione*, appunto, un'uguaglianza tra il valore cercato e il valore calcolato.

2.2.1 Il metodo della tabella

Il primo metodo che descriviamo per calcolare logaritmi discreti nel gruppo moltiplicativo del campo $GF(q)$ è anche il più intuitivo: consiste nel costruire una tabella contenente i logaritmi di tutti gli elementi del gruppo. Dopo di che diventa una banalità trovare il logaritmo cercato.

Esempio: Sia $G = \mathbb{F}_7^*$, il gruppo moltiplicativo degli interi modulo 7, e sia $\omega = 3$ (ω è un generatore del gruppo), allora si costruisce facilmente la seguente lista:

$$\begin{array}{lll} \log_3 1 = 0, & \log_3 2 = 2, & \log_3 3 = 1, \\ \log_3 4 = 4, & \log_3 5 = 5, & \log_3 6 = 3. \end{array}$$

□

2.2.2 Algoritmo di ricerca esaustiva

Il secondo metodo è la *ricerca esaustiva*, molto simile al precedente, differisce solamente per il fatto che ogni logaritmo calcolato k_i viene confrontato con il logaritmo cercato ω.

Si supponga che $\gamma \in \mathbb{Z}_p^*$ generi un sottogruppo G di $\mathbb{Z}_p^*$ di ordine $m \geq 1$, e siano γ e $\omega \in G$. Si vuole calcolare $\log_\gamma \omega$.

Algorithm 2.2.1: RICERCAESAUSTIVA(γ, ω)

$b \leftarrow 1$
$i \leftarrow 0$
while $b \neq \omega$ **do**
    $b \leftarrow b \cdot \gamma$
    $i \leftarrow i + 1$
output (i)

Questo algoritmo ovviamente è corretto e il loop principale si fermerà sempre dopo al più m iterazioni. Quindi il tempo totale di impiego è $O(m \cdot len(p)^2)$.

Proposizione 2.3 *Sia G il gruppo moltiplicativo di $GF(q)$ e sia $\gamma \in G$ un elemento di ordine M. (Si ricordi che ciò vuol dire che $\gamma^M = e$ e che nessun'altra potenza minore di γ è uguale ad e). Allora il problema del logaritmo discreto*

$$\gamma^x = \omega$$

può essere risolto in $O(len(M) \cdot len(q)^2)$

Dimostrazione: Basta fare una lista di valori γ^x per $x = 1, 2, \ldots, M - 1$. Si noti che ogni successivo valore può essere ottenuto moltiplicando il precedente per γ. Se una soluzione di $\gamma^x = \omega$ esiste, allora ω apparirà nella lista.

□

2.2.3 Algoritmo baby step giant step

Di seguito analizzeremo un algoritmo che viene utilizzato quando si lavora in campi di ordine relativamente piccolo. Come il nome ci suggerisce, tale algoritmo è diviso in due parti (*step*). In entrambe le parti vengono calcolati dei valori che verrano poi implementati in delle liste.

Proposizione 2.4 *Sia $\gamma \in \mathbb{Z}_p^*$ un generatore del sottogruppo G di $\mathbb{Z}_p^*$. Sia $N \geq 2$ l'ordine del gruppo G. Il seguente algoritmo, l'algoritmo* baby step giant step*, risolve il problema del logaritmo discreto*

$$\gamma^x = \omega$$

in $O(\sqrt{N} \cdot len(p)^2)$.

1. *Sia $n = 1 + \lfloor \sqrt{N} \rfloor$, quindi in particolare, $n \geq \sqrt{N}$.*
2. *Si creino due liste:*
   - *Lista 1: $e, \gamma, \gamma^2, \ldots, \gamma^n$*
   - *Lista 2: $\omega, \omega \cdot \gamma^{-n}, \omega \cdot \gamma^{-2n}, \ldots, \omega \cdot \gamma^{-n^2}$.*
3. *Esiste sempre un abbinamento tra le due liste, diciamo $\gamma^i = \omega \cdot \gamma^{-jn}$.*
4. *Allora $x = i + jn$ è una soluzione di $\gamma^x = \omega$.*

Dimostrazione: Quando creiamo la *Lista 2*, cominciamo calcolando la quantità $u = \gamma^{-n}$ e poi compiliamo la *Lista 2* computando $\omega, \omega \cdot u, \omega \cdot u^2, \ldots$. Quindi creare le due liste impiega circa $2n$ moltiplicazioni.

$$O(2n \cdot len(p)^2) = O(n \cdot len(p)^2) = O(\sqrt{N} \cdot len(p)^2).$$

Per provare che l'algoritmo funziona, occorre mostrare che la *Liste 1* e la *Lista 2* abbiamo sempre una collisione. Per vedere ciò, sia x l'incognita di $\gamma^x = \omega$ e scriviamo x come

$$x = nq + r \qquad \text{con } 0 \leq r < n$$

Sappiamo che $1 \leq x < N$, quindi

$$q = \frac{x - r}{n} < \frac{N}{n} < n \qquad \text{dato che } n > \sqrt{N}.$$

Quindi possiamo riscrivere l'equazione $\gamma^x = \omega$ come

$$\gamma^r = \omega \cdot \gamma^{-qn} \qquad \text{con } 0 \leq r < n \text{ e } 0 \leq q < n.$$

Quindi γ^r è nella *Lista 1* e $\omega\gamma^{-qn}$ è nella *Lista 2*, il chè mostra che la *Lista* 1 e la *Lista* 2 hanno un elemento in comune.

□

Vediamo come implementare l'algoritmo. L'idea è di calcolare tutti i valori γ^i per $i = 0, 1, 2, 3, \ldots, n-1$ (*Baby Step*) e costruire un array T che associ γ^i al valore i. Per $b \in \mathbb{Z}_p^*$, possiamo scrivere $T[b]$ per denotare il valore associato a b, scrivendo $T[b] = \perp$ se non c'è nessun valore. Possiamo assumere che T sia implementato così da accedere a $T[b]$ velocemente. Usando una struttura dati appropriata T può essere implementata di modo che l'accesso degli elementi individuali impieghi $O(len(p))$. Per esempio una tale struttura dati potrebbe essere un albero di ricerca. Possiamo costruire l'array associato T usando il seguente algoritmo:

Algorithm 2.2.2: BABYSTEP(T, p, γ)

INITIALIZE $T//T[b] = \perp$ per ogni $b \in \mathbb{Z}_p^*$
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $T[b] \leftarrow i$
    $b \leftarrow b \cdot \gamma$

Chiaramente tale algoritmo impiega $O(N^{1/2} len(p)^2)$. Dopo aver costruito la tabella, facciamo eseguire la seguente procedura (*Giant Step*)

Algorithm 2.2.3: GIANTSTEP(T, ω)

$\gamma' \leftarrow \gamma^{-n}$
$b \leftarrow \omega, \quad j \leftarrow 0, \quad i \leftarrow T[b]$
while $i = \perp$ **do**
    $b \leftarrow b \cdot \gamma'$
    $j \leftarrow j + 1$
    $i \leftarrow T[b]$
$x \leftarrow j \cdot n + i$
output (x)

Anche se questo algoritmo è molto più veloce della *ricerca esaustiva* ha lo svantaggio che richiede uno spazio per circa $N^{1/2}$ elementi di $\mathbb{Z}_p$. Ovviamente c'è un compromesso "spazio/tempo": scegliendo n più piccolo, abbiamo una tavola di grandezza $O(n)$, ma il tempo di impiego sarà proporzionale a $O(N/n)$.

2.3 Gruppi di ordine q^e

Si supponga che $\gamma \in \mathbb{Z}_p^*$ generi un sottogruppo G di $\mathbb{Z}_p^*$ di ordine q^e dove $q > 1$ e $e \geq 1$ e siano dati $p, q, e, \gamma,$ e ω, con $\gamma,$ e $\omega \in G$, vogliamo calcolare $\log_\gamma \omega$. Esiste un semplice algoritmo che permette di ridurre il problema al problema di calcolare il logaritmo discreto nel sottogruppo di $\mathbb{Z}_p^*$ di ordine q. Forse è più facile descrivere l'algoritmo in modo ricorsivo.

Il caso base si ha quando $e = 1$; in tal caso usiamo un algoritmo per il sottogruppo di $\mathbb{Z}_p^*$ di ordine q. Perciò, potremmo impiegare l'algoritmo *baby/giant step* o, se q è molto piccolo, l'algoritmo di *ricerca esaustiva*. Si supponga ora che $e > 1$. Scegliamo un intero f, con $0 < f < e$. Una differente strategia per scegliere f produce un differente algoritmo - discuteremo di questo di seguito. Si supponga $\omega = \gamma^x$, dove $0 \leq x < q^e$. Allora possiamo scrivere $x = q^f \cdot v + u$, dove u e v sono interi con $0 \leq u < q^f$ e $0 \leq v < q^{e-f}$. Quindi,

$$\omega^{q^{e-f}} = \gamma^{q^{e-f}u}.$$

Si noti che $\gamma^{q^{e-f}}$ ha ordine moltiplicativo q^f e quindi se calcoliamo ricorsivamente il logaritmo discreto di $\omega^{q^{e-f}}$ di base $\gamma^{q^{e-f}}$, otteniamo u. Ottenuto u, si osservi che

$$\frac{\omega}{\gamma^u} = \gamma^{q^f v}.$$

Si noti che γ^{q^f} ha ordine moltiplicativo q^{e-f} e quindi se calcoliamo ricorsivamente il logaritmo discreto di $\frac{\omega}{\gamma^u}$ in base γ^{q^f}, otteniamo v, da cui poi calcoliamo $x = q^f v + u$.

Mettiamo le idee precedenti in una procedura ricorsiva, l'algoritmo RDL (Recursive Discrete Logarithm):

Algorithm 2.3.1: AlgoritmoRDL$(p, q, e, \gamma, \omega)$

---

if $e = 1$
  return $(\log_\gamma \omega)$ // caso base: utilizzare un altro algoritmo
else
  select $f \in \{1, \ldots, e-1\}$
  $u \leftarrow \mathbf{RDL}(p, q, f, \gamma^{q^{e-f}}, \omega^{q^{e-f}})$ // $0 \le u < q^f$
  $v \leftarrow \mathbf{RDL}(p, q, e-f, \gamma^{q^f}, \omega/\gamma^u)$ // $0 \le v < q^{e-f}$
return $(q^f v + u)$

Per analizzare il tempo di esecuzione dell'algoritmo ricorsivo, si noti che il tempo di esecuzione del corpo di una chiamata ricorsiva è $O(e \cdot len(q) len(p)^2)$. Per calcolare il tempo totale di esecuzione, dobbiamo sommare i tempi di tutte le chiamate ricorsive più il tempo di tutti i casi base.

Ignorando la strategia scelta per scegliere f, il numero totale dei casi base è e. Si noti che tutti i casi base calcolano il logaritmo discreto di base $\gamma^{q^{e-1}}$. Assumendo che i casi base vengano implementati usando l'algoritmo Baby/Giant Step, il tempo totale per tutti i casi base è $O(e \cdot q^{1/2} len(p)^2)$. Il tempo totale per la ricorsione (non includendo il calcolo dei casi base) dipende dalla strategia usata per scegliere come dividere f. È utile rappresentare il comportamento dell'algoritmo usando un albero ricorsivo. È un albero binario, dove ogni nodo rappresenta una chiamata ricorsiva dell'algoritmo; la radice dell'albero rappresenta la chiamata iniziale dell'algoritmo; per ogni nodo N nell'albero, se N rappresenta la chiamata ricorsiva i, allora il figlio di N (se esiste) rappresenta una chiamata ricorsiva fatta da i. Possiamo

naturalmente organizzare i nodi dell'albero ricorsivo tramite livelli: la radice dell'albero ricorsivo è al livello 0, i suoi figli al livello 1, i suo nipoti al livello 2, e così via. La profondità dell'albero ricorsivo è definita dal massimo livello di un nodo.

Consideriamo due differenti strategie per dividere f.

- se scegliamo sempre $f = 1$ o $f = e - 1$, allora la profondità della ricorsione dell'albero è $O(e)$. Il tempo con cui contribuisce per ogni livello dell'albero di ricorsione è $O(e \cdot len(q)len(p)^2)$, e perciò il tempo totale di ricorsione è $O(e^2 \cdot len(q)len(p)^2)$. Si noti che se $f = 1$, allora l'algoritmo è essenzialmente ricorsivo di coda, e quindi può essere facilmente convertito in un algoritmo iterativo.

- se usiamo una strategia bilanciata *divide et impera*, scegliendo $f \approx e/2$, allora la profondità della ricorsione dell'albero è $O(len(e))$, dove il tempo d'impiego che contribuisce ad ogni livello dell'albero ricorsivo è ancora $O(e \cdot len(q)len(p)^2)$. Quindi il tempo totale d 'impiego per la ricorsione è $O(e \cdot len(e)len(q)len(p)^2)$.

Assumendo di usare la strategia bilanciata ricorsiva più veloce, e che usiamo l'algoritmo *baby/giant step* per i casi base, il tempo totale dell'algoritmo RDL è :

$$O((e \cdot q^{1/2} + e \cdot len(e)len(q)) \cdot len(p)^2).$$

2.4 Logaritmo discreto in $\mathbb{Z}_p^*$

Si supponga che siano dati un primo p, insieme alla fattorizzazione in primi di

$$p - 1 = \prod_{i=1}^{r} q_i^{e_i},$$

un generatore γ per $\mathbb{Z}_p^*$ e $\omega \in \mathbb{Z}_p^*$. Vogliamo calcolare $\log_\gamma \omega$.

Si supponga che $\omega = \gamma^x$, dove $0 < x < p-1$. Allora per $i = 0, 1, 2, \ldots, r$ abbiamo

$$\omega^{(p-1)/q_i^{e_i}} = \left(\gamma^{(p-1)/q_i^{e_i}}\right)^x.$$

Si noti che $\gamma^{(p-1)/q_i^{e_i}}$ ha ordine moltiplicativo $q_i^{e_i}$, e se x_i è il logaritmo discreto di $\omega^{(p-q)/q_i^{e_i}}$, allora abbiamo $0 \leq x_i < q_i^{e_i}$ e $x \equiv x_i \pmod{q_i^{e_i}}$. Quindi se calcoliamo i valori $x_1, \ldots, x_r$ usando l'algoritmo RDL nel paragrafo 2.3 possiamo ottenere x usando l'algoritmo del Teorema Cinese del Resto. Se

definiamo $q := max(q_1, \ldots, q_r)$, allora il tempo d'impiego di tale algoritmo sarà limitato da $q^{1/2} \cdot len(p)^{O(1)}$. Quindi possiamo concludere dicendo che:

la difficoltà di calcolare il logaritmo discreto in $\mathbb{Z}_p^*$ è determinata dalla grandezza del più grande primo divisore di $p-1$.

2.5 Algoritmi sub-esponenziali

Questa sezione presenta algoritmi a tempo sub-esponenziale per calcolare il logaritmo discreto. Ad oggi sono considerati gli algoritmi più efficienti conosciuti.

2.5.1 Algoritmo SEDL

Presentiamo ora un algoritmo a successo probabilistico con tempo di esecuzione sub-esponenziale per calcolare logaritmi discreti. L'input dell'algoritmo è p, q, γ, ω, dove p e q sono primi, con q che divide $p-1$, γ un elemento di $\mathbb{Z}_p^*$ che genera un sottogruppo G di $\mathbb{Z}_p^*$ di ordine q, e $\omega \in G$. Assumiamo per semplicità che q^2 non divida $p-1$. Il che è equivalente a dire che q non divide $m := (p-1)/q$. Benché non strettamente necessario, tale assunzione semplifica la progettazione e l'analisi dell'algoritmo, e inoltre, per applicazioni crittografiche, tale ipotesi è quasi sempre soddisfatta. Per sommi capi, lo scopo principale del nostro algoritmo del logaritmo discreto è quello di trovare una rappresentazione di 1 rispetto a γ e ω, questo ci permette di calcolare il $\log_\gamma \omega$ (con alta probabilità). Più precisamente, il nostro scopo principale è quello di calcolare interi r e s in modo probabilistico, tale che $\gamma^r \cdot \omega^s = 1$. Detto ciò, con probabilità $1 - 1/q$, avremo $s \neq 0 \pmod q$, il che ci permette di calcolare $\log_\gamma \omega$ come $-rs^{-1} \pmod q$.

Sia H un sottogruppo di $\mathbb{Z}_p^*$ di ordine m. La nostra ipotesi che q non divida m implica che $G \cap H = \{1\}$, dato che l'ordine moltiplicativo di ogni elemento nell'intersezione deve dividere sia q che m, e quindi l'unica possibilità è che l'ordine moltiplicativo sia 1. Inoltre l'applicazione $\rho : G \times H \to \mathbb{Z}_p^*$ che manda (β, δ) in $\beta\delta$ è iniettiva, e dato che $\mathbb{Z}_p^* = qm$, deve anche essere surriettiva. Dato che H è l'immagine dell'omomorfismo che manda ogni valore α di $\mathbb{F}_p^*$ in α^q, possiamo generare a caso $\delta \in H$ semplicemente scegliendo $d \in \mathbb{Z}_p^*$ a caso, e ponendo $\delta := d^q$.

Siano $p_1, \ldots, p_k$ tutti i primi minori di un certo $y < p$. Siano $\pi_i := [p_1]_p \in \mathbb{Z}_p^*$ per $i = 1, \ldots, k$.

L'algoritmo ha due livelli. Nel primo livello, troviamo le relazioni della forma

$$\gamma^{r_i}\omega^{s_i}\delta_i = \pi_1^{e_{i1}} \cdots \pi_k^{e_{ik}}, \tag{2.1}$$

per $i = 1, \ldots, k+1$, dove $r_i, s_i, e_{i1}, \ldots, e_{ik} \in \mathbb{Z}$ e $\delta_i \in H$ $\forall i$. Otteniamo ognuna delle relazioni da una ricerca a caso, come segue: scegliamo r_i e $s_i \in \{0, 1, \ldots, q-1\}$ a caso, e allo stesso modo $d_i \in \mathbb{Z}_p^*$; dopo di che calcoliamo $\delta_i := d_i^q, \beta_i := \gamma^{r_i} \cdot \omega^{s_i}$, e $m_i := [\beta_i \cdot \delta_i]_p$. Ora il valore β_i è uniformemente distribuito su G, quando δ_i è uniformemente distribuito su H; perciò, il prodotto $\beta_i\delta_i$ è uniformemente distribuito su $\mathbb{Z}_p^*$, e quindi m_i è uniformemente distribuito su $\{1, \ldots, p-1\}$. Di seguito proveremo semplicemente a fattorizzare m_i con divisioni banali, provando tutti i primi $p_1, \ldots, p_k$ fino a y. Trovati tali primi, fattorizziamo completamente m_i, ottenendo una fattorizzazione del tipo

$$m_i = p_1^{e_{i1}} \cdots p_k^{e_{ik}}.$$

Per qualche esponente $e_{i1}, \ldots, e_{ik}$ e abbiamo la relazione (2.1). Per $i = 1, \ldots, k+1$, siano $v_i := (e_{i1}, \ldots, e_{ik}) \in \mathbb{Z}^{\times k}$, e i w_i denotino l'immagine dei v_i in $\mathbb{Z}_q^{\times k}$ (i.e. $w_i := ([e_{i1}]_q, \ldots, [e_{ik}]_q)$) dato che $\mathbb{Z}_q^{\times k}$ è uno spazio vettoriale sopra il campo $\mathbb{Z}_q$ di dimensione k, la famiglia di vettori $w_1, \ldots, w_{k+1}$ deve essere linearmente dipendente.

Il secondo livello dell'algoritmo usa l'eliminazione di Gauss su $\mathbb{Z}_q$ per trovare una dipendenza lienare sui vettori $w_1, \ldots, w_{k+1}$, il che vuol dire, trovare $c_1, \ldots, c_{k+1} \in \{0, \ldots, q-1\}$, non tutti nulli, tali che,

$$(e_1, \ldots, e_k) := c_1 v_1 + \cdots + c_{k+1} v_{k+1} \in q\mathbb{Z}_q^{\times k}.$$

Elevando ogni equazione (2.1) alla corrispondente potenza c_i, e moltiplicandole insieme otteniamo

$$\gamma^r \omega^s \delta = \pi_1^{e_1} \cdots \pi_k^{e_k},$$

dove

$$r := \sum_{i=1}^{k+1} c_i r_i, \qquad s := \sum_{i=1}^{k+1} c_i s_i, \qquad \text{e } \delta := \prod_{i=1}^{k+1} d_i^{c_i}.$$

Ora, $\delta \in H$, e dato che e_j è un multiplo di q, abbiamo inoltre $\pi_j^{e_j} \in H$ per $j = 1, \ldots, k$. Ne segue che $\gamma^r \cdot \omega^s \in H$. Ma dato che si ha anche $\gamma^r \cdot \omega^s \in G$, e $G \cap H = \{1\}$, ne segue che $\gamma^r \cdot \omega^s = 1$. Se abbiamo $s \neq 0 \pmod q$ possiamo calcolare $\bar{s} := s^{-1} \pmod q$, ottenendo

$$\omega = \gamma^{-r\bar{s}},$$

e quindi $-r\bar{s} \pmod{q}$ è il logaritmo discreto di ω di base γ. Se risulta $s \equiv 0 \pmod{q}$, l'algoritmo semplicemente riporterà *"failure"*. Se l'algoritmo, invece, non dà come output *"failure"*, allora l'output è sicuramente il logaritmo discreto di ω di base γ.

2.5.2 Il metodo del calcolo dell'indice

Il metodo più potente conosciuto per calcolare logaritmi in un gruppo è il *metodo del calcolo dell'indice*. La tecnica non si applica sempre ad un dato gruppo, ma quando lo fa dà sempre un algoritmo con tempo di esecuzione sub-esponenziale per calcolare logaritmi discreti.

Ricordiamo che *indice* è un vecchio termine del logaritmo discreto. Il primo calcolo dell'indice apparve nel lavoro di Western e Miller del 1968, anticipando di alcuni anni l'invenzione dei crittosistemi a chiave pubblica.

Sia G un gruppo ciclico finito di ordine n generato da γ nel quale vogliamo risolvere il problema del logaritmo discreto

$$\gamma^x \quad = \quad \omega. \tag{2.2}$$

Si supponga che $S = \{p_1, p_2, \ldots, p_t\}$ sia un qualche sottoinsieme di G con la proprietà che una parte significativa di tutti gli elementi in G possono essere scritti come prodotto di elementi di S. L'insieme S è usualmente chiamato *base di fattori* per il metodo del calcolo dell'indice. Piuttosto che risolvere la (2.2) direttamente, scegliamo di risolvere il problema per i valori contenuti in S.

Nel passo 1 del metodo del calcolo dell'indice proviamo, quindi, a trovare i logaritmi di tutti gli elementi di S come segue. Prendiamo un intero arbitrario a e tentiamo di scrivere γ^a come prodotto di elementi di S:

$$\gamma^a = \prod p_i^{\lambda_i} \qquad \text{per } i = 0, \ldots, t. \tag{2.3}$$

Se abbiamo successo, allora la (2.3) produce una congruenza lineare

$$a \equiv \sum_{i=1}^{t} \lambda_i \log_\gamma p_i \qquad (\text{mod } n). \tag{2.4}$$

Dopo aver trovato un numero sufficiente (maggiore di t) di relazioni della forma (2.4), il corrispondente sistema di equazioni avrà un'unica soluzione nelle indeterminate $\log_\gamma p_i$ per $1 \leq i \leq t$.

Nel passo 2 dell'algoritmo calcoliamo logaritmi individuali in G. Dato $\delta \in G$ vogliamo trovare un intero x tale che $\gamma^x = \delta$. Ripetutamente prendiamo interi arbitrari s fintanto che $\gamma^s \delta$ possa essere scritto come prodotto di elementi in S:

$$\gamma^s \delta = \prod_{i=1}^{t} p_i^{b_i}. \tag{2.5}$$

Abbiamo così

$$\log_\gamma \delta \equiv \sum_{i=1}^{t} b_i \log_\gamma p_i - s \qquad (\text{mod } n).$$

□

Osservazione Un problema secondario che abbiamo ignorato è il fatto che le equazioni lineari (2.4) sono congruenze modulo $p-1$. I metodi standard per l'algebra lineare come l'eliminazione di Gauss non funzionano molto bene quando si lavora componendo moduli. Il Teorema Cinese del Resto risolve tale problema. Prima si risolvono le congruenze (2.5) modulo q per ogni primo q che divide $p-1$. Poi, se q appare nella fattorizzazione di $p-1$ come una potenza q^e, portiamo la soluzione da $\mathbb{Z}/q\mathbb{Z}$ a $\mathbb{Z}/q^e\mathbb{Z}$. Infine usiamo il Teorema Cinese del Resto per combinare le soluzioni modulo potenze dei primi per ottenere una soluzione modulo $p-1$. Nelle applicazioni crittografiche si dovrebbe scegliere p tale che $p-1$ sia divisibile per un primo grande.

Capitolo 3

Il logaritmo discreto in crittografia

"We stand today on the brink of a revolution in cryptography"

Whitfield Diffie e Martin Hellman,
"New Directions in Cryptography"

Nel 1976 Whitfield Diffie e Martin Hellman publicarono il loro lavoro, ad oggi famoso, intitolato *"New Directions in Cryptography"*. Nel loro lavoro formularono il concetto di sistema di cifratura a chiave pubblica, forse la più grande scoperta crittografica di tutti i tempi. Con il loro scritto risolsero il più grande problema crittografico, rimasto insoluto per millenni. Scoprirono un metodo per comunicare in sicurezza utilizzando linee non sicure, quindi senza il bisogno di incontrarsi per scambiarsi delle chiavi da utilizzare per le cifrature. La pubblicazione di Diffie e Hellman fu un evento di estrema importanza, che mise le basi e definì gli scopi di un nuovo campo della matematica e dell'informatica, un campo la cui esistenza dipendeva sull'emergente tecnologia dei computer digitali.

Prima della pubblicazione di "New Directions in Criptography", la ricerca in crittoanalisi era di dominio della National Security Agency, e tutte le scoperte e gli studi in questo campo erano riservate. Infatti fino alla metà degli anni '90, il governo americano considerava gli algoritmi crittografici come munizioni, il che significava che esportarli era perseguito come un'azione proditoria. Il governo, inoltre, vietava discussioni libere e aperte che vertevano su algoritmi astratti utilizzabili in crittografia ed era ancora indecisa la

possibilità di usare tali algoritmi all'interno della nazione. Tutto questo per evitare che sofisticati sistemi crittografici andassero nelle mani di potenziali nemici degli Stati Uniti. Questione opinabile, dal momento che sì, erano algoritmi utilizzati per le comunicazioni, ma erano anche questioni di ricerca scientifica, per cui non v'era motivo che la comunità scientifica europea o degli altri paesi non potesse discuterne.

L'impossibilità, inoltre, dell'utilizzo domestico, per la privacy dei cittadini, scatenò una decisa protesta a questa politica del governo. La protesta assunse toni plateali, tanto che fu scritta una versione in tre righe dell'algoritmo RSA in un linguaggio di programmazione, chiamato *Perl*, e fu stampato su magliette e lattine di cola.

In quegli anni, indossare una maglietta RSA su un volo da New York in Europa, faceva rischiare chi la indossava una multa salata e 10 anni di prigione. Se invece la persona addirittura si tatuava sul corpo tale codice, non avrebbe potuto prendere voli per uscire dagli Stati Uniti essendo considerato esso stesso un'arma.

Sebbene tali proteste ebbero degli effetti sulla politica del governo, fu una semplice realtà a cambiare il corso delle cose. Gli algoritmi a chiave pubblica sono molto semplici tuttavia è richiesta una certa competenza per implementarli in maniera sicura. Il mondo è pieno di eccellenti matematici, scienziati e ingegneri, quindi la restrizione del governo sull'esportazione di un sistema resistente semplicemente incoraggiò la creazione di industrie crittografiche in altre parti del mondo.

Il più grande contributo di Diffie e Hellman fu la definizione di crittosistema a chiave pubblica (Public Key Cryptosystem, PKC) e delle sue componenti associate - *funzioni one-way* o *funzioni trappola*. Una funzione one-way è una funzione invertibile, facile da calcolare, ma la cui inversa è difficile da calcolare. Ma cosa vuol dire "difficile da calcolare"? Intuitivamente una funzione è difficile da calcolare se qualsiasi algoritmo che cerca di calcolare l'inversa in una quantità di tempo "ragionevole", per esempio meno dell'età dell'universo, quasi sicuramente fallisce, dove "quasi sicuramente" si intende probabilisticamente. I crittosistemi a chiave pubblica sono costruiti usando funzioni one-way che utilizzano una *botola*. La botola è un'informazione ausiliaria che ci permette di calcolare l'inversa facilmente.

Teoricamente, non è poi così difficile immaginarsi tali funzioni, quando si pensa astrattamente; ovviamente, però, trovarne una in pratica è stato un

problema portato avanti per centinaia di anni.

Diffie e Hellman fecero notevoli proposte per delle funzioni trappola, ma non produssero un esempio di PKC. Descrissero, però, un metodo a chiave pubblica con il quale un certo materiale poteva essere condiviso in un qualche canale insicuro. Il loro metodo chiamato *"scambio di chiavi Diffie-Hellman"* (Diffie-Hellman key exchange), è basato sull'assunzione che il problema del logaritmo discreto (PLD) è difficile da risolvere. Nel loro scritto Diffie e Hellman inoltre definirono una varietà di attacchi crittografici e introdussero il concetto importante della firma digitale. Con la pubblicazione del 1976, la sfida era quella di inventare un crittosistema a chiave pubblica efficiente. Il problema fu risolto in seguito da Rivest, Shamir e Adleman con il loro sistema a chiave pubblica RSA, utilizzato ancora oggi.

3.1 Scambio di chiave Diffie-Hellman

L'algoritmo di Diffie-Hellman per lo scambio di chiavi risolve il seguente problema. Alice e Bob vogliono condividere una chiave segreta per utilizzarla in una *cifratura simmetrica*[1], ma i loro unici mezzi di comunicazione sono insicuri. Ogni informazione che si scambiano è osservata dalla loro avversaria Eve. Come è possibile per Alice e Bob condividere una chiave senza renderla disponibile anche per Eve? A prima vista pare che Alice e Bob siano in un problema irrisolvibile. Ci fu un'idea brillante in Diffie e Hellman, ossia che la difficoltà del problema del logaritmo discreto per $\mathbb{F}_p^*$ dia una possibile soluzione. Il primo passo per Alice e Bob è di mettersi d'accordo su di un grande primo p e un intero non nullo γ modulo p. Alice e Bob rendono pubblici i valori p e γ; per esempio li possono mettere sul loro sito web, di modo che anche Eve possa conoscerli. Per varie ragioni che discuteremo più avanti, è meglio che scelgano γ tale che il suo ordine in $\mathbb{F}_p^*$ sia un primo grande. Il passo successivo è per Alice di scegliere un intero segreto a che non deve rilevare a nessuno, allo stesso tempo Bob sceglie un intero b che manterrà segreto. Bob e Alice usano i loro interi segreti per calcolare

$$A \equiv \gamma^a \pmod{p} \qquad \text{e} \qquad B \equiv \gamma^b \pmod{p}$$

Di seguito si scambiano i valori appena calcolati, Alice invia A a Bob e Bob invia B a Alice. Si noti che Eve può vedere i valori A e B, dato che sono

[1] Si parla di cifratura simmetrica quando si usa la stessa chiave privata per cifrare e decifrare il messaggio.

inviati in linee di comunicazioni insicure. Infine Bob e Alice di nuovo usano i loro interi segreti per calcolare

$$\bar{A} \equiv B^a \pmod{p} \qquad \text{e} \qquad \bar{B} \equiv A^b \pmod{p}$$

I valori che hanno calcolato, $\bar{A}$ e $\bar{B}$ rispettivamente, sono in realtà lo stesso, dato che

$$\bar{A} \equiv B^a \equiv (\gamma^b)^a \equiv \gamma^{ab} \equiv (\gamma^a)^b \equiv A^b \equiv \bar{B} \pmod{p}$$

Questo valore comune è la loro chiave. In generale il problema di Eve è questo. Conosce i valori A e B, quindi conosce i valori γ^a e γ^b. Conosce inoltre i valori γ e p, quindi se sa risolvere il *problema del logaritmo discreto* (PLD), allora può trovare i valori a e b, a partire dai quali le sarà facile calcolare la chiave segreta di Bob e Alice γ^{ab}. Si vede, quindi, che Alice e Bob sono sicuri fin tanto che Eve non sappia risolvere il PLD, ma questo non è propriamente corretto. È vero che un metodo per trovare il valore in comune di Alice e Bob è quello di risolvere il PLD, ma non è il preciso problema che Eve deve risolvere. La sicurezza della chiave comune di Alice e Bob si basa sulla difficoltà del seguente, potenzialmente più semplice, problema:

Definizione 3.1 *Sia p un numero primo e γ un intero. Il* problema di Diffie-Hellman *(PDH) è il problema di calcolare il valore $\gamma^{ab} \pmod{p}$ dai valori $\gamma^a \pmod{p}$ e $\gamma^b \pmod{p}$.*

È chiaro che il PDH non sia più difficile del PLD. Se Eve sa risolvere il PLD, allora può calcolare gli esponenti segreti a e b dall'intercettazione dei valori $A = \gamma^a$ e $B = \gamma^b$, e poi sarà facile per lei calcolare la loro chiave segreta γ^{ab}. Ma l'inverso è meno semplice. Si supponga che Eve abbia un algoritmo che efficientemente risolva il PDH. Può usarlo per risolvere efficientemente il PLD? Non vi è ancora una risposta.

3.2 Il crittosistema a chiave pubblica di ElGamal

Benchè l'algoritmo di scambio di chiavi di Diffie-Hellman produca un metodo per condividere una chiave segreta, non risponde al pieno scopo di essere un sistema crittografico a chiave pubblica, dato che un crittosistema permette di condividere informazioni specifiche, e non solo stringhe di bit prese a caso. Il primo crittosistema a chiave pubblica fu l'RSA di Rivest, Shamir, Adleman che pubblicarono nel 1978. Comunque benchè l'RSA

fosse il primo in ordine cronologico, quello che seguì in maniera più diretta il lavoro di Diffie Hellman è un sistema descritto da Taher ElGamal nel 1985. L'algoritmo di cifratura a chiave pubblica di ElGamal è basato sul problema del logaritmo discreto ed è imparentato con lo scambio di chiavi di Diffie-Hellman. In questa sezione descriviamo la versione del PKC ElGamal basato sul problema del logaritmo discreto in $\mathbb{F}_p^*$, ma la costruzione funziona praticamente su qualsiasi gruppo.

In questo caso Alice comincia pubblicando informazioni: la chiave pubblica e un algoritmo. La chiave pubblica è semplicemente un numero, e l'algoritmo è il metodo con cui Bob dovrà cifrare il suo messaggio utilizzando la chiave pubblica di Alice. Alice non pubblicherà la sua chiave privata, che sarà un altro numero. La chiave privata permette ad Alice, e solo ad Alice, di decifrare i messaggi che sono stati cifrati con la sua chiave pubblica. Tutto questo è alquanto vago e può valere per ogni crittosistema a chiave pubblica. Per il PKC di ElGamal, Alice ha bisogno di un numero primo grande p per il quale il problema del logaritmo discreto in $\mathbb{F}_p^*$, è difficile, e ha bisogno di un elemento γ modulo p di ordine (primo) grande. Può scegliere p e γ da sè, o possono essere stati preselezionati da qualcuno di fidato come un'industria o un'agenzia governativa. Alice sceglie un numero segreto a come chiave privata, e calcola la quantità

$$A \equiv \gamma^a \pmod{p}$$

Si noti la similitudine con il protocollo di Diffie-Hellman. Alice pubblica la sua chiave pubblica A e tiene la privata per sé. Ora si supponga che Bob voglia cifrare un messaggio utilizzando la chiave pubblica di Alice A. Assumiamo che il messaggio di Bob m sia un intero compreso tra 2 e p. Per cifrare m, Bob prima sceglie un numero casuale k modulo p . Poi usa k per cifrare uno e un solo messaggio, e poi lo getta via. Bob prende il suo testo originale m, sceglie un k casuale e la chiave pubblica A di Alice e li usa per calcolare le quantità

$$c_1 \equiv \gamma^k \pmod{p} \qquad \text{e} \qquad c_2 \equiv mA^k \pmod{p}$$

(si ricordi che γ e p sono parametri pubblici, quindi anche Bob li conosce). Il testo cifrato di Bob, cioè la cifratura di m, è la coppia di numeri (c_1, c_2), che manderà ad Alice. Come farà Alice a decifrare il testo cifrato di Bob? Dato che Alice conosce a, può calcolare la quantità

$$x \equiv c_1^a \pmod{p}$$

E quindi inoltre $x^{-1} \pmod p$. Alice poi moltiplica c con x^{-1} e il risultato è il testo originale m. Per vedere perchè, espandiamo il valore $x^{-1} \cdot c_2$ e scopriamo che

$$\begin{aligned}
x^{-1} \cdot c_2 &\equiv (c_1^a)^{-1} \cdot c_2 \pmod p && \text{, dato che } x \equiv c_1^a \pmod p \\
&\equiv (\gamma^{ak})^{-1} \cdot (mA^k) \pmod p && \text{, dato che } c_1 \equiv \gamma^k, c_2 \equiv mA^k \pmod p \\
&\equiv (\gamma^{ak})^{-1} \cdot (m(\gamma^a)^k) \pmod p && \text{, dato che } A \equiv \gamma^a \pmod p \\
&\equiv m \pmod p && \text{, dato che i termini } \gamma^{ak} \\
& && \quad \text{vengono cancellati.}
\end{aligned}$$

Che cosa serve ad Eve per decifrare il messaggio? Eve conosce i parametri pubblici p e γ, e inoltre conosce il valore $A \equiv \gamma^a \pmod p$, dato che la chiave pubblica di Alice A è di dominio pubblico. Se Eve sa risolvere il problema del logaritmo discreto, può trovare a e decifrare il messaggio.

Sia lo schema di Diffie-Hellman che quello di ElGamal sono largamente usati in pratica. Tipicamente il gruppo scelto è uno tra $\mathbb{F}_{2^m}^*$, $\mathbb{F}_p^*$ (p primo), o un gruppo dei punti di una curva ellittica su di un campo finito.

3.3 Il problema del logaritmo discreto nelle curve ellittiche

3.3.1 Curve ellittiche, prime definizioni

L'argomento delle curve ellittiche comprende un vasto campo della matematica. Qui di seguito daremo solo poche definizioni che ci serviranno per affrontare meglio il *problema del logaritmo discreto nelle curve ellittiche*.

Una *curva ellittica* è l'insieme delle soluzioni di un'equazione della forma

$$Y^2 = X^3 + AX + B.$$

Equazioni di questo tipo sono chiamate *equazioni di Weierstrass* dal matematico che le studiò in modo approfondito nel XIX secolo. Un'affascinante proprietà delle curve ellittiche è che se si prendono due punti in una curva ellittica e si "sommano", se ne produce un terzo sempre appartenente alla curva. Abbiamo messo le virgolette per la parola *sommano* perché in realtà l'operazione che sia fa tra due punti di una curva ellittica non è proprio la somma ordinaria, benché abbia le sue proprietà (commutativa e associativa).

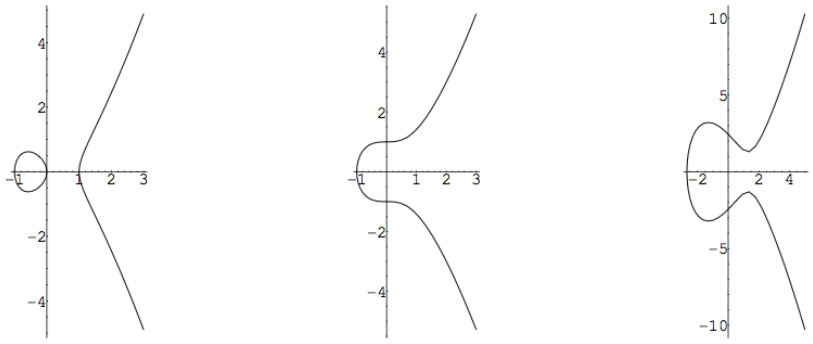

Figura 3.1: Grafici delle curve ellittiche sul campo $\mathbb{R}$, date dalle equazioni:

$$y^2 = x^3 - x, \quad y^2 = x^3 + 1, \quad y^2 = x^3 - 5x + 6.$$

Definizione 3.2 *Una* curva ellittica E *è l'insieme delle soluzioni di* un'equazione di Weierstrass

$$E : Y^2 = X^3 + AX + B,$$

insieme ad un punto extra $\mathcal{O}$*, dove le costanti* A *e* B *devono soddisfare*[2]

$$4A^3 + 27B^2 \neq 0.$$

La legge additiva su E è definita come segue. Siano P e Q due punti su E. Sia L la linea che congiunge P e Q, o la tangente a E in P, se $P = Q$. Allora l'intersezione di E e L è composta da tre punti P, Q e R', contati con le appropriate molteplicità e sottointendendo che $\mathcal{O}$ giace su ogni linea verticale. Scrivendo $R' = (a, b)$, la somma di P e Q è definita essere la riflessione $R = (a, -b)$ di R' rispetto l'asse delle ascisse. Inoltre se $P = (a, b)$ allora $-P = (a, -b)$.

Teorema 3.3 *Sia* E *una curva ellittica. Allora la legge additiva su* E *ha le seguenti proprietà:*

(a)	$P + \mathcal{O} = \mathcal{O} + P = P$	*per ogni* $P \in E$.
(b)	$P + (-P) = \mathcal{O}$	*per ogni* $P \in E$
(c)	$(P + Q) + R = P + (Q + R)$	*per ogni* $P, Q, R \in E$
(d)	$P + Q = Q + P$	*per ogni* $P, Q \in E$

[2]Questa condizione ci assicura che ogni curva ellittica possa essere messa nella forma di Weierstrass.

In altre parole, la legge additiva dà ai punti di E una struttura di gruppo abeliano.

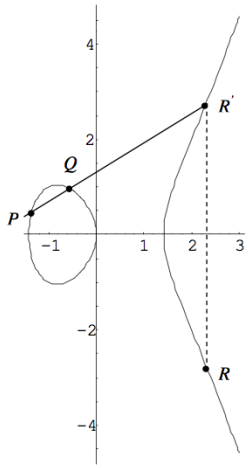

Figura 3.2: Sommare nel gruppo $\mathbb{E}(\mathbb{F}_p)$

Ora vediamo come definire le curve ellittiche su campi finiti. Semplicemenete si definisca una curva ellittica su $\mathbb{F}_p$ con un'equazione della forma

$$E : Y^2 = X^3 + AX + B \qquad \text{con } A, B \in \mathbb{F}_p \text{ che soddisfino } 4A^3 + 27B^2 \neq 0,$$

e vediamo i punti su E con coordinate in $\mathbb{F}_p$, che denotiamo con

$$\mathbb{E}(\mathbb{F}_p) = \{(x, y) : x, y \in \mathbb{F}_p \text{ che soddisfino } y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}$$

Teorema 3.4 *Sia E una curva ellittica su $\mathbb{F}_p$ e siano P e Q punti in $\mathbb{E}(\mathbb{F}_p)$.*

a) L' addizione applicata a P e Q produce un punto R in $\mathbb{E}(\mathbb{F}_p)$.

b) Tale addizione su $\mathbb{E}(\mathbb{F}_p)$ soddisfa tutte le proprietà della lista del Teorema 3.3. In altre parole, tale legge additiva rende $\mathbb{E}(\mathbb{F}_p)$ un gruppo finito.

3.3.2 Il problema del logaritmo discreto nelle curve ellitiche

Vediamo ora come il problema del logaritmo discreto nel gruppo $\mathbb{F}_p^*$, possa essere riformulato anche nel gruppo $\mathbb{E}(\mathbb{F}_p)$ dei punti di una curva ellittica. È

chiaro che Alice possa impostare il problema con i punti di una curva ellittica piuttosto che gli elementi di un gruppo. Sceglie due punti P e Q di $\mathbb{E}(\mathbb{F}_p)$ e li pubblica, la sua chiave segreta sarà l'intero n tale che:

$$Q = \underbrace{P + P + \cdots + P}_{n \text{ volte}} = nP$$

Allora Eve ha bisogno di scoprire quanto volte P debba essere sommato a se stesso di modo da ottenere Q. Si tenga in mente che benché la legge additiva sulle curve ellittiche è convenzionalmente scritta con il segno $+$, l'addizione in $\mathbb{E}$ è in realtà un'operazione molto complicata, perciò questo analogo ellittico del problema del logaritmo discreto potrà risultare abbastanza difficile da risolvere.

Definizione 3.5 *Sia $\mathbb{E}$ una curva ellittica su un campo finito $\mathbb{F}_p$ e siano P e Q punti in $\mathbb{E}(\mathbb{F}_p)$.* Il problema del logaritmo discreto nelle curve ellittiche $(PLDCE)$ *è il problema di trovare un intero n tale che $Q = nP$. Analogamente al problema del logaritmo discreto per $\mathbb{F}_p^*$, denotiamo tale intero n con*

$$n = \log_P(Q)$$

e chiamiamo n il logaritmo discreto ellittico di Q rispetto P.

Osservazione Bisogna precisare alcuni aspetti della definizione di $\log_P(Q)$. La prima difficoltà è il fatto che ci possono essere punti $P, Q \in \mathbb{E}(\mathbb{F}_p)$ tali che Q non sia un multiplo di P. In tal caso, $\log_P(Q)$ non è definito. La seconda difficoltà è che se esiste un valore n che soddisfi $Q = nP$, allora ne esistono molti altri. Per vedere ciò, facciamo prima vedere che esiste un intero s tale che $sP = \mathcal{O}$. Dato che $\mathbb{E}(\mathbb{F}_p)$ è finito, i punti nella lista $P, 2P, 3P, \ldots$ non possono essere tutti distinti. Quindi esiste un intero $k > j$ tale che $kP = jP$, e possiamo prendere $s = k - j$. Il più piccolo $s \geq 1$ è chiamato *ordine* di P (che divide l'ordine di $\mathbb{E}(\mathbb{F}_p)$). Quindi se s è l'odrine di P e se n_0 è un intero tale che $Q = n_0 P$, allora le soluzioni di $Q = nP$ sono gli interi $n = n_0 + is$ con $i \in \mathbb{Z}$. Questo vuol dire che il valore di $\log_P(Q)$ è veramente un elemento di $\mathbb{Z}/s\mathbb{Z}$, cioè, $\log_p(Q)$ è un intero modulo s, dove s è l'ordine di P. Impostiamo $\log_P(Q) = n_0$. Il vantaggio di definire i valori in $\mathbb{Z}/s\mathbb{Z}$ è che il logaritmo discreto ellittico così soddisfa

$$\log_P(Q_1 + Q_2) = \log_P(Q_1) + \log_P(Q_2) \qquad \text{per ogni } Q_1, Q_2 \in \mathbb{E}(\mathbb{F}_p). \quad (3.1)$$

Si noti l'analogia con il logaritmo ordinario e il logaritmo discreto in $\mathbb{F}_p^*$. Il fatto che il logaritmo discreto per $\mathbb{E}(\mathbb{F}_p)$ soddisfi (3.1) significa che rispetta la legge additiva.

3.4 Algoritmo Double-and-Add

Appare abbastanza difficile scoprire il valore n dai due punti P e $Q = nP$ in $\mathbb{E}(\mathbb{F}_p)$, è quindi difficile risolvere il *problema del logaritmo discreto nelle curve ellittiche (PLDCE)*. Vediamo come calcolare efficientemente il valore nP, noti n e P. Se n è grande, sicuramente non vogliamo calcolare nP calcolando $P, 2P, 3P, \ldots$. Il modo più efficiente per calcolare nP è un algoritmo chiamato *Double-and-Add*. Il metodo è molto simile all'algoritmo *square-and-multiply* descritto nella sezione 2.1. L'idea principale è analoga: scriviamo prima n in forma binaria come

$$n = n_0 + n_1 \cdot 2 + n_2 \cdot 4 + n_3 \cdot 8 + \cdots + n_r \cdot 2^r \quad \text{con } n_0, n_1, \ldots, n_r \in \{0, 1\}.$$

Assumiamo inoltre che $n_r = 1$. Poi calcoliamo le seguenti quantità:

$$Q_0 = P, \quad Q_1 = 2Q_0, \quad Q_2 = 2Q_1, \quad \ldots \quad , Q_r = 2Q_{r-1}.$$

Si noti che Q_i è semplicemente il doppio del precedente Q_{i-1}, quindi

$$Q_i = 2^i P.$$

Tali punti, quindi, sono il prodotto di P per potenze di 2, e per calcolarli bisogna fare r raddoppiamenti. Infine, calcoliamo nP usando al massimo r addizioni,

$$nP = n_0 Q_0 + n_1 Q_1 + n_2 Q_2 + \cdots + n_r Q_r.$$

Sia l'addizione di due punti in $\mathbb{E}(\mathbb{F}_p)$ un'unica operazione. Quindi il tempo totale per calcolare nP impiega al massimo $2r$ operazioni in $\mathbb{E}(\mathbb{F}_p)$. Questo procedimento rende possibile calcolare nP per valori di n anche molto grandi.

Osservazione Esiste una tecnica addizionale che può essere usata per ridurre di molto il tempo richiesto per calcolare nP. L'idea è di scrivere n usando somme e differenze di potenze di 2. La ragione per la quale diventa vantaggioso è perché generalmente ci sono molti meno termini, quindi molte meno addizioni sono necessarie per calcolare nP. È importante osservare che sottrarre due punti in una curva ellittica è facile come sommarli, dato che $-(x, y) = (x, -y)$.

3.5 Quanto è arduo il problema del logaritmo discreto nelle curve ellittiche?

Gli algoritmi di collisione descritti nella Sezione 2.2 sono facilmente adattabili ad ogni gruppo, per esempio al gruppo dei punti $\mathbb{E}(\mathbb{F}_p)$ di una curva ellittica. Per risolvere $Q = nP$, Eve sceglie a caso interi $j_1, \ldots, j_r$ e $k_1, \ldots, k_r$ tra 1 e p e crea due liste di punti:

- Lista A: $j_1P, j_2P, j_3P, \ldots, j_rP$
- Lista B: $k_1P + Q, k_2P + Q, k_3P + Q, \ldots, k_rP + Q$.

Appena trova un *match* (una collisione) tra le due liste, l'algoritmo conclude la sua ricerca, dato che se trova $j_uP = k_vP + Q$, allora $Q = (j_u - k_v)P$, quindi la soluzione.

Si noti la similitudine con l'algoritmo *baby/giant step* in 2.2.2, anche qui si creano due liste solo che in forma additiva.

Se r è maggiore di $\sqrt{p}$, diciamo $r \approx 3\sqrt{p}$, allora esiste una buona chance che ci sia una collisione. Abbiamo visto che esistono molti modi più veloci per risolvere il problema del logaritmo discreto per $\mathbb{F}_p^*$. In particolare, il metodo del calcolo dell'indice nella sezione (2.5.2), ha tempo di esecuzione sub-esponenziale. La ragione principale per cui le curve ellittiche sono usate in crittografia è il fatto che non esiste un algoritmo del calcolo dell'indice conosciuto per $PLDCE$. In altre parole, nonostante la complessa natura strutturale del gruppo $\mathbb{E}(\mathbb{F}_p)$, l'algoritmo più veloce ad oggi conosciuto per risolvere il $PLDCE$ non è migliore del generico algoritmo che funziona egualmente bene per risolvere il problema logaritmo discreto in un qualunque gruppo. Quindi l'$PLDCE$ appare molto più difficile del PLD.

Conclusioni

Abbiamo visto come la difficile trattabilità del *problema del logaritmo discreto* sia generatrice di svariate applicazioni in algebra, informatica e crittografia. L'utilizzo più vasto che ne è stato tratto è sicuramente quello crittografico.

Ogni giorno, indirettamente e al più delle volte senza rendersene conto, milioni di persone usano crittosistemi basati sul problema del logaritmo discreto quando alzano la cornetta del telefono o immettono il numero della loro carta di credito su internet. Come abbiamo visto questo è possibile grazie alla sicurezza che tali sistemi hanno nei confronti di un possibile attacco. Una sicurezza garantita dall'assenza di un algoritmo realmente efficiente. Sicuramente negli ultimi anni si sono fatti passi da gigante sia per quanto riguarda le capacità di calcolo degli elaboratori sia per quanto riguarda gli studi e le nuove scoperte in crittoanalisi.

Quanto possiamo concludere da questa tesi è che per ora il miglior crittosistema è quello basato sulle curve ellittiche, dato che, per questo caso, ancora non si è scoperto un algoritmo con tempo di esecuzione inferiore ad un tempo esponenziale.

Sicuramente in un futuro abbastanza prossimo, bisognerà trovare altri metodi per tutelare le comunicazioni, dato che ormai, già da tempo, si è ipotizzata la realizzazione di computer con delle capacità tali da infrangere qualunque crittosistema.

Un tale computer, chiamato *computer quantistico*, venne ideato nel 1985 da David Deutsch. Questo calcolatore non si basa più su le leggi della fisica classica, ma bensì sulle leggi della fisica quantistica. Una tale ipotetica macchina basa la sua progettazione sul *principio di sovrapposizione di stati*, potendo così elaborare più problemi contemporaneamente.

Tale calcolatore può essere concepito in due modi diversi, a seconda dell'interpretazione quantistica che si preferisce. Alcuni fisici vedono il computer quantistico come un'entità singola che esegue contemporaneamente i calcoli; altri come entità distinte, ognuna delle quali esiste in un universo separato ed esegue calcoli a partire da un solo numero per volta.

In un calcolatore classico il bit è l'unità fondamentale di elaborazione e può assumere i valori 0 oppure 1. Un computer quantistico, rispondendo alle leggi della meccanica quantistica, opera su bit in stato di sovrapposizione tra 0 ed 1, i cosiddetti *qubit*. Un qubit si trova in una sovrapposizione di stati finchè non ne viene chiesto il valore: a quel punto restituisce un valore. Prendendo N qubit, possiamo immaginare che rappresentino tutti i numeri interi compresi tra 0 e 2^{N-1}; è proprio questa la potenza di un computer quantistico, in quanto è in grado di accettare come input non un solo valore, ma una sovrapposizione di molti valori differenti ed in seguito eseguire una computazione su tutti i numeri simultaneamente.

Un risultato eccezionale ottenuto da Peter Shor nel 1994 è stato quello di trovare un algoritmo efficiente per la fattorizzazione di numeri interi e per il calcolo del logaritmo discreto tramite l'uso di un calcolatore quantistico. Tali algoritmi risolverebbero i due problemi in tempo polinomiale, quindi tutte le tecniche finora discusse verrebbero a perdere la loro sicurezza per uno scambio di informazioni.

Benchè la realizzazione di un computer quantistico sia molto lontana dalla realtà, alcuni esperimenti hanno ottenuto dei risultati promettenti; un gruppo di ricercatori è riuscito a fattorizzare il numero 15 facendo uso di 3 qubit.

Di fatto nessuno degli attuali sistemi a chiave pubblica sopravvivereb-be ad un calcolatore quantistico e nuovi metodi dovranno essere ideati al palesarsi di una di queste macchine che sembrano sfidare anche il buon senso.

Bibliografia

[1] M. Artin, *Algebra*, Bollati Boringhieri, 1991.

[2] I. F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, T. Yaghoobian *Application of Finite Fields*, Springer, 1993.

[3] J. Hoffstein, J. Pipher, Joseph H. Silverman, *An Introduction to Mathematical Cryptography*, Springer, 2008.

[4] V. Shuop, *A Computational Introduction to Number Theory and Algebra*, Victor Shoup, 2008.

[5] J. von zur Gathen, José Luis Imana *Arithmetic of finite fields*, Springer, 2008.

[6] A.M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance", in *Advances in cryptology* (Paris, 1984), 224–314, Lecture Notes in Comput. Sci., 209, Springer, 1985.

[7] D. Jungnickel, *Finite fields: structure and arithmetics*, B.I. Wissenschaftsverlag, 1993.

[8] Igor E. Shparlinski, *Computational and algorithmic problems in finite fields*, Springer, 1993.

[9] J. Buchmann, *Introduction to cryptography* 2a ed., Springer, 2004.

[10] G. Mullen, D. White, "A polynomial representation for logarithms in $GF(q)$", Acta Arith. 47 (1986), 255–261.

[11] H. Niederreiter, "A short proof for explicit formulas for discrete logarithms in finite fields", Appl. Algebra Engrg. Comm. Comput. 1 (1990), 55–57.