Un' Introduzione ai problemi NP, NP completi e alle loro applicazioni alla crittografia

Antonio Corbo Esposito

Napoli, 18 Maggio 2018

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
- Protocolli crittografici
- Curve Ellittiche
- Tecniche di calcolo del logaritmo discreto

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
- Protocolli crittografici
- Curve Ellittiche
- Tecniche di calcolo del logaritmo discreto

  - Gruppo
  - Anello
  - Campo
  - Aritmetica modulare

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
  - Complessità degli algoritmi
  - Classi P ed NP
  - Problemi NP completi
  - Problemi NP di uso attuale
- Protocolli crittografici
- Curve Ellittiche
- Tecniche di calcolo del logaritmo discreto

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
- Protocolli crittografici
  - Generazione dei numeri primi
  - RSA
  - Diffie-Hellman
  - DLP
- Curve Ellittiche
- Tecniche di calcolo del logaritmo discreto

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
- Protocolli crittografici
- Curve Ellittiche
  - Legge di gruppo
  - Studio di una curva
  - DLP
- Tecniche di calcolo del logaritmo discreto

Introduzione

- Oggetti matematici fondamentali
- Teoria della complessità computazionale
- Protocolli crittografici
- Curve Ellittiche
- Tecniche di calcolo del logaritmo discreto
  - Baby-step, Giant-step
  - Pollard-ρ

Oggetti matematici fondamentali

Gruppo

Un gruppo è una struttura algebrica costituita da un insieme e un operazione binaria che soddisfa i seguenti assiomi:

L'operazione di gruppo è chiusa $\forall\, x, y \in G$ vale $x \circ \mathrm{y} = \mathrm{z}$ e $z \in G$

- $\circ$ è associativo $\forall\, x, y, z \in G$ vale $(x \circ \mathrm{y}) \circ z = x \circ (y \circ z)$
- $\circ$ ha un elemento neutro $\forall\, x \in G\ \exists\, e \in G : x \circ e = e \circ x = x$
- Esiste un inverso $\forall\, x \in G\ \exists\, x^{-1} \in G : x \circ x^{-1} = x^{-1} \circ x = e$ dove e è l'elemento neutro

Anello

Un anello è un insieme con due operazioni binarie, $(G, +, \cdot)$ che deve rispettare:

- $(G, +)$ è un gruppo abeliano con elemento neutro 0
- + è commutativa, cioè $\forall x, y \in G$ vale $x + y = y + x$
- $\cdot$ è associativa ed ha 1 come elemento neutro
- $\cdot$ è distributiva rispetto a $+$, $\forall x, y, z \in G$ vale $x \cdot (y + z) = x \cdot y + x \cdot z$

Campo

Un campo è un anello commutativo in cui ogni elemento diverso da 0 ha un inverso.

Quindi se G è un campo devono valere:

- Proprietà associativa sia per l'addizione che per la moltiplicazione:
$$a + (b + c) = (a + b) + c \text{ e } a \cdot (b \cdot c) = (a \cdot b) \cdot c$$
- Proprietà commutativa: $a + b = b + a$ e $a \cdot b = b \cdot a$
- Identità ovvero esistono due distinti elementi in G: 0 e 1 tale che $a + 0 = 0 + a = a$ e $a \cdot 1 = 1 \cdot a = a$
- Inverso moltiplicativo: $\forall a \neq 0 \in G, \exists a^{-1} : a \cdot a^{-1} = 1$
- Inverso additivo: $\forall a \in G, \exists (-a) : a + (-a) = 0$
- Proprietà distributiva: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

Campo

Definizione

Un campo $\mathbb{F}$ si dice finito se contiene un numero finito di elementi. La cardinalità di $\mathbb{F}$ è il suo numero di elementi.

Teorema esistenza ed unicità dei campi finiti

- Se $\mathbb{F}$ è un campo finito, allora contiene p^n elementi dove p è un numero primo ed $n \geq 1$
- Per ogni potenza di un numero primo p^n esiste un unico campo finito, a meno di isomorfismi, di ordine p^n e si indica con $\mathbb{GF}(p^n)$ o con $\mathbb{F}(p^n)$.

Aritmetica modulare

Una descrizione del campo con p elementi è $\mathbb{Z}_p$ dato dalle classi di congruenza modulo un numero primo p dove la somma e il prodotto vengono seguite dalla riduzione modulo p.

Una descrizione del campo con p^n elementi può essere data da $\mathbb{Z}_p[x]/q(x)$ dove $q(x)$ è un polinomio irriducibile di grado n in $\mathbb{Z}_p[x]$.

Ad es. il campo con quattro elementi può essere visto come $\mathbb{Z}_2[x]/(x^2+x+1)$ e cioè dagli elementi $0, 1, x, x+1$.

Qualche esempio di addizione in $\mathbb{Z}_2[x]/(x^2+x+1)$

$1+(x+1)=x;\ x+(x+1)=1;\ 1+1=0;$

Qualche esempio di moltiplicazione in $\mathbb{Z}_2[x]/(x^2+x+1)$

$x(x+1)=1; xx=x^2=x+1;\ (x+1)^2=x.$

Teoria della complessità computazionale

Complessità degli algoritmi

La complessità di un algoritmo è definita come la potenza di calcolo necessaria per eseguirlo. Solitamente viene misurata tenendo conto di due variabili:

- *T* per la *complessità temporale*
- *S* per la *complessità spaziale*

Solitamente viene espressa tramite la ***Big O notation*** che permette di approssimare l'andamento di un algoritmo ignorando le costanti moltiplicative tramite una funzione che asintoticamente si comporta come l'originale.

Complessità degli algoritmi

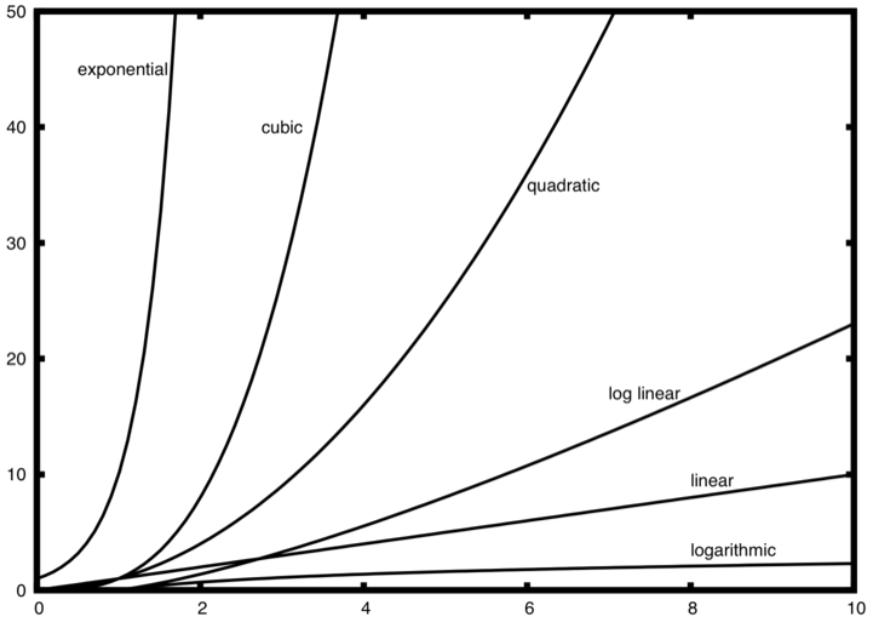
Le definizioni associate alla *Big O notation* in base alle funzioni usate per approssimare sono:

- $f(n) = \mathcal{O}(g(n))$ significa che $k \cdot g(n)$ è un limite superiore (funzione che approssima dall'alto) di $f(n)$. Esiste quindi una costante k tale che $f(n) \leq k \cdot g(n)$ per valori abbastanza grandi di n
- $f(n) = \Omega(g(n))$ significa che $k \cdot g(n)$ è un limite inferiore (funzione che approssima dal basso) di $f(n)$. Esiste quindi una costante k tale che $f(n) \geq k \cdot g(n)$, per valori abbastanza grandi di n
- $f(n) = \Theta(g(n))$ significa che $k_1 \cdot g(n)$ è un limite superiore di $f(n)$ e $k_2 \cdot g(n)$ è un limite inferiore per ogni $n \geq n_0$. Esistono delle costanti k_1 e k_2 per cui $f(n) \leq k_1 \cdot g(n)$ e $f(n) \geq k_2 \cdot g(n)$ con ciò possiamo affermare che $g(n)$ fornisce una buona approssimazione di $f(n)$.

Complessità degli algoritmi

Tramite la *Big O notation* possiamo raggruppare funzioni simili nella stessa classe di complessità; possiamo avere ad esempio:

- Funzioni *costanti* $f(n) = 1$
- Funzioni *logaritmiche* $f(n) = \log n$
- Funzioni *lineari* $f(n) = n$
- Funzioni *superlineari* $f(n) = n \log n$
- Funzioni *quadratiche* $f(n) = n^2$
- Funzioni *polinomiali* $f(n) = n^k$
- Funzioni *esponenziali* $f(n) = c^n$
- Funzioni *fattoriali* $f(n) = n!$



Complessità degli algoritmi

Nella tabella di seguito è possibile notare come all'aumentare della complessità, varia il tempo di esecuzione di un algoritmo.

Classi	Complessità	#operazioni per $n = 10^6$	Tempo
Costante	$\mathcal{O}(1)$	1	1 μsec.
Lineare	$\mathcal{O}(n)$	10^6	1 sec.
Quadratica	$\mathcal{O}(n^2)$	10^{12}	11.6 days
Cubica	$\mathcal{O}(n^3)$	10^{18}	32000 yrs.
Esponenziale	$\mathcal{O}(2^n)$	10^{301030}	10^{301006} yrs.

*I calcoli sono stati effettuati considerando un computer in grado di eseguire 1 operazione in 1μ secondo.

Classe di complessità computazionale P

- La classe dei problemi $\mathbf{P}$ è data dai problemi che possono essere risolti in un tempo dato da una funzione $f(n) = \mathcal{O}(n^k)$ polinomiale nella lunghezza n dell'input;
- Esempi di problemi nella classe $\mathbf{P}$ possono essere:

- Il calcolo del valore di un polinomio in un campo finito;
- Il calcolo del determinante di una matrice quadrata di ordine n
- La risoluzione di un sistema lineare di m equazioni in n incognite

- Questo tipo di problemi non hanno applicazioni di tipo crittografico, in quanto non danno a chi vuole crittare i dati un vantaggio significativo su chi vuole decrittare i dati stessi.

Classe di complessità computazionale NP

- La classe dei problemi **NP** è data dai problemi la cui **soluzione** può essere **verificata** in un tempo dato da una funzione $f(n) = \mathcal{O}(n^k)$ polinomiale nella lunghezza n dell'input;
- **NP** sta per Nondeterministic Polynomial Time
- Dalla definizione segue che $\mathbf{P} \subset \mathbf{NP}$
- Esistono molti problemi in **NP** per i quali non si conosce un algoritmo di risoluzione che funzioni in tempo polinomiale (e quindi per i quali non si sa se siano in **P**)
- Nessuno è però in grado di dimostrare se $\mathbf{P} = \mathbf{NP}$ o $\mathbf{P} \neq \mathbf{NP}$.
- Tutta la crittografia moderna è basata sulla ricerca di problemi che siano in **NP** (ma non in **P**) e dunque sulla fiducia che $\mathbf{P} \neq \mathbf{NP}$

Problemi NP completi

- Una nozione importante è quella dei problemi **NP-completi**,
- Un problema si dice **NP-completo** se: qualora esista un algoritmo che lo risolva in tempo polinomiale, da ciò si può dedurre che $\mathbf{P} = \mathbf{NP}$
- Si tratta, in senso informale, dei problemi **«più difficili»** presenti nella classe $\mathbf{NP}$
- Il prototipo dei problemi **NP-completi** è il problema della soddisfacibilità booleana o problema **SAT** (Teorema di Cook Levin, 1971)
- Un sistema quadratico di n equazioni in n incognite è **NP-completo** su qualunque campo finito, anche $\mathbb{Z}_2$ (problema **MQ**, Garey e Johnson 1979)
- Uno dei pochi metodi di crittazione per i quali esista una «dimostrazione» di difficoltà di decrittazione è il **QUAD** (Berbain, Gilbert, Patarin, 2006), è basato su un generatore di bit pseudocasuali crittograficamente sicuri costituito da un sistema di 320 equazioni in 160 incognite su $\mathbb{Z}_2$

Altri problemi NP usati in crittografia

- Il problema della fattorizzazione di un numero di n bits non è attualmente risolubile in tempo polinomiale (su un computer non quantistico)
- Il miglior algoritmo conosciuto per la sua risoluzione è il GFNS (General Field Number Sieve) che ha una complessità di $\mathcal{O}(\exp(\sqrt[3]{\frac{64(n(\ln(n))^2)}{9}}))$
- È noto però che tale problema è però risolubile in tempo polinomiale su un computer «quantistico» (algoritmo di Shor, 1997)
- Informalmente, la capacità di calcolo di un computer quantistico è data dal numero di «qubits» che può manipolare
- Il più grande numero fattorizzato da un computer quantistico è 291311=523x557
- Recentemente la Cina ha investito circa 10 mld di dollari per un centro di ricerca sui computer quantistici. Gli USA spendono circa 1 mld/anno, la UE 300 mln/anno.

Altri problemi NP usati in crittografia

- Un altro problema **NP** usato in crittografia è il problema del logaritmo discreto, che verrà ampiamente discusso in seguito
- Anche tale problema è vulnerabile da parte dell'algoritmo di Shor
- Per fare un esempio, attualmente la sicurezza della blockchain di bitcoin è basata sulla difficoltà di invertire una funzione che si chiama **SHA256.** Non si conoscono algoritmi di quantum computing in grado di invertire tale funzione.
- L'identità digitale di un wallet di un possessore di bitcoin è invece creata con un algoritmo che si chiama firma digitale **ECDSA** (che verrà anch'esso presentato in seguito). Tale algoritmo, basato sulle curve ellittiche, è anch'esso vulnerabile da parte dell'algoritmo di Shor.
- Quindi, in caso di avvento dei quantum computers, il sistema di transazioni basato su bitcoin crollerebbe (nella versione attuale)

Protocolli crittografici

Generazione di numeri primi

Algoritmo Rabin-Miller

Scegliamo un numero a caso p. Calcoliamo b che è il numero di volte che 2 divide $p-1$ e calcoliamo m in modo tale da avere $p = 1 + 2^b m$.

1. Si sceglie un numero a caso a minore di p
2. $j = 0$ e $z \equiv a^m \ (mod\ p)$
3. Se $z = 1$ o $z = p - 1$ allora p è probabilmente un numero primo.
4. Se $j > 0$ e $z = 1$ allora p non è primo
5. $j = j + 1$
6. Se $j < b$ e $z \neq p - 1$ allora $z = z^2 \ (mod\ p)$ e ricomincia dal punto 3. Se $z = p - 1$ allora p è probabilmente un numero primo.
7. Se $j = b$ e $z \neq p - 1$ allora p non è primo.

Diffie-Hellman

Algoritmo:

- A (Alice) sceglie un numero intero grande x e invia a B (Bob)

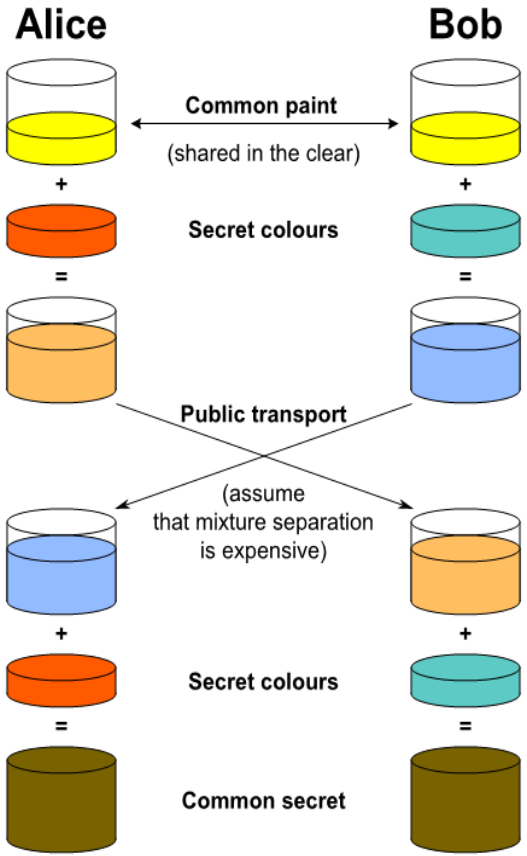
$$X = g^x \ (mod\ n)$$

- B sceglie un numero intero grande y e invia ad A

$$Y = g^y \ (mod\ n)$$

- A calcola $k_0 = Y^x \ (mod\ n)$
- B calcola $k_1 = X^y \ (mod\ n)$

Siccome $(g^x)^y = (g^y)^x = g^{xy}$ ho che $k_1 = k_2 = k$



DLP

Il **problema del logaritmo discreto** è definito come segue

Sia $(G,\circ)$ un gruppo ciclico di ordine n e sia g un generatore di G avendo

$$y = g^x = g \circ g \circ \cdots \circ g$$

trovare x.

Dobbiamo quindi calcolare $x = \log_g y$ detto appunto logaritmo discreto.

Una cosa da notare è che la soluzione al problema del logaritmo discreto non è unica. Se G è un gruppo ciclico con g come generatore allora ho che $g^x = g^z \iff x \equiv z \ (mod\ n)$

Schema chiave pubblica – chiave privata

- Analogia (semplificata) di come funziona la coppia chiave pubblica – chiave privata

Crittazione – decrittazione di un messaggio

Firma digitale – verifica della firma

RSA

Generazione della *chiave pubblica* e *privata*

- Vengono generati due grandi numeri primi p e q entrambi con la stessa lunghezza.
- Calcoliamo $n = pq$ detto modulo e $\phi = (p-1)(q-1)$
- Scegliamo ora un numero intero a caso e tale che $1 < e < \phi$ e $gdc(e, \phi) = 1$
- Tramite *l'algoritmo esteso di Euclide* calcoliamo d tale che:

$$1 < d < \phi \text{ e } ed \equiv 1 \ (mod\ )$$

- La *chiave pubblica* è data dalla coppia di valori (n, e), mentre la *chiave privata* è d

RSA

Crittazione dei dati

- Il soggetto interessato alla comunicazione ottiene la chiave pubblica del mittente (n, e)
- Il messaggio da inviare viene rappresentato come un intero $m \in [0, n-1]$
- Calcoliamo $c = m^e \ (mod\ n)$
- Il messaggio cifrato da inviare è c

Decrittazione dei dati

- Il destinatario che ha fornito la sua chiave pubblica e che riceve il messaggio criptato c per recuperare il messaggio originale usa la sua chiave privata d ed ottiene $m = c^d \ (mod\ n)$.

Curve ellittiche

Curve ellittiche

Una *curva ellittica* E su un campo $\mathbb{K}$ indicata con $E/\mathbb{K}$ è data dall'equazione di $Weierstraß$:

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

con $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$

Un condizione da rispettare affinche E sia ellitticha è quella della non singolarità. Un punto sulla curva è detto singolare se entrambe le derivate parziali sono nulle.

Siano:

$$b_2 = a_1^2 + 4a_2, \qquad \mathrm{b}_4 = a_1a_3 + 2a_4, \qquad b_6 = a_3^2 + 4a_6,$$
$$b_8 = a_1^2a_6 - a_1a_3a_4 + 3a_2a_6 + a_2a_3^2 + a_4^2$$

Utilizzando la trasformazione: $y \to y - \frac{(a_1x+a3)}{2}$ otteniamo la curva isomorfa

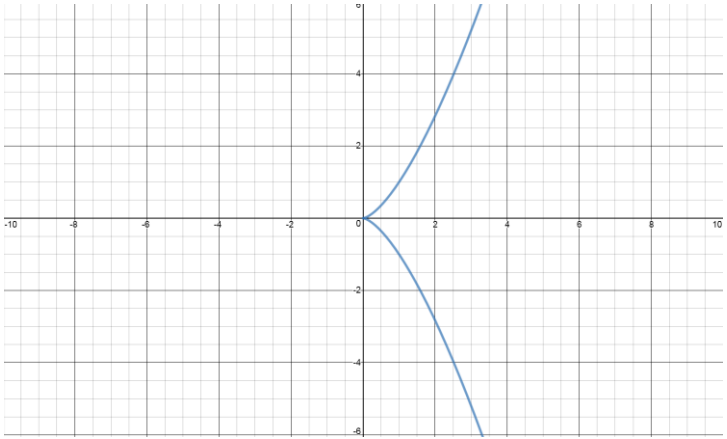
$$y^2 = x^3 + \frac{b_2}{4}x^2 + \frac{b_4}{2}x + \frac{b_6}{4}$$

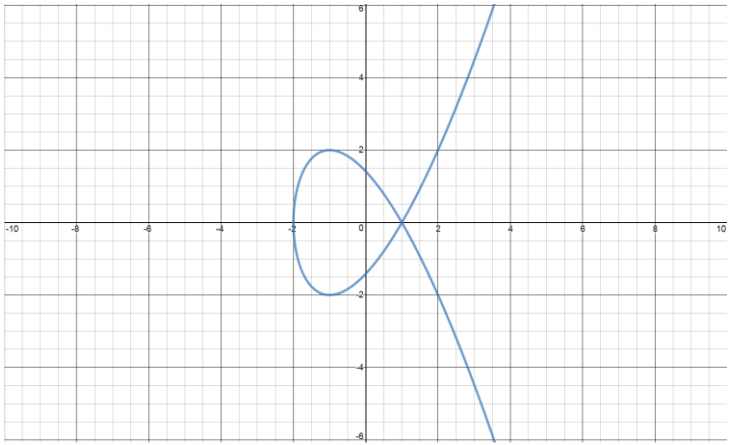
Curve ellittiche

Sia E una curva definita su $\mathbb{K}$. Il discriminante di E denotato da Δ:

$$\Delta = -b_2^2\, b_8 - 8b_4^3 - 27b_6^2 + 9b_2 b_4 b_6$$

La curva si dice *non singolare* e quindi *ellittica* se Δ è diverso da zero.





EC su campi finiti

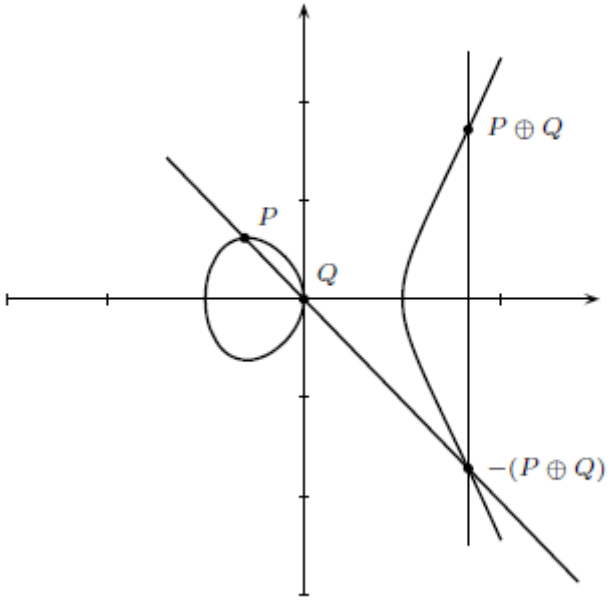
Una curva *ellittica* su $\mathbb{F}_p$ con $p > 3$ è l'insieme di tutti i punti $(x, y) \in \mathbb{F}_p$

$$y^2 = x^3 + ax + b \ (mod\ p)$$

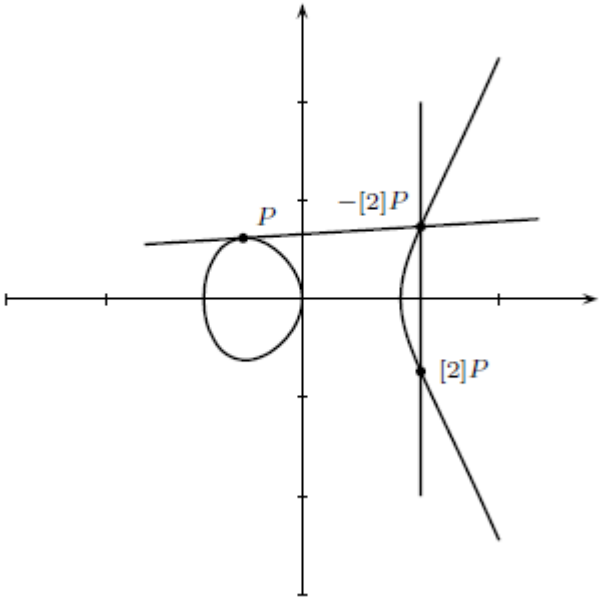
Insieme ad un punto immaginario all'infinito e con $a, b \in \mathbb{F}_p$ e $4a_4 + 27b_2 \neq 0 \ (mod\ p)$

Curve ellittiche

Legge di gruppo



Addizione



Raddoppio

Curve ellittiche

Legge di gruppo su un campo $\mathbb{K} = \mathbb{F}_p$ con $p > 3$

$$y^2 = x^3 + ax + b$$

- Identità: $P + \Theta = P$ per ogni P appartenente alla curva
- Opposto: Sia $P = (x, y)$ allora $(x, y) + (x, -y) = \Theta$. Il punto $(x, -y)$ viene indicato con $-P$
- Addizione: Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ punti sulla curva con $P \neq \pm Q$. Allora $R = P + Q = (x_3, y_3)$ con :

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \qquad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x1 - x_3) - y_1$$

- Raddoppio: Sia $P = (x_1, y_1)$ un punto sulla curva e $P \neq -P$. Allora $R = P + P = (x_3, y_3)$ con:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \qquad y3 = \left(\frac{3x_1 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

Cardinalità di un gruppo

Il numero di punti che possono trovarsi su un $\boldsymbol{EC}$ è in generale difficile da determinare, esiste però un teorema che ci permette di stimare la cardinalità del gruppo:

Teorema di Hasse-Weil

Sia E una curva ellittica definita su $\mathbb{F}_q$. Allora

$$|E(\mathbb{F}_q)| = q + 1 - t$$

con $|t| \leq 2\sqrt{q}$

DLP su curve ellittiche

Sia $(G, \bigoplus)$ un gruppo ciclico di ordine p, con p primo. Siamo $P, Q \in G$ e sia P ungeneratore di G. Il logaritmo discreto in G di Q rispetto a P è pari a

$n = \log_P Q$ quindi:

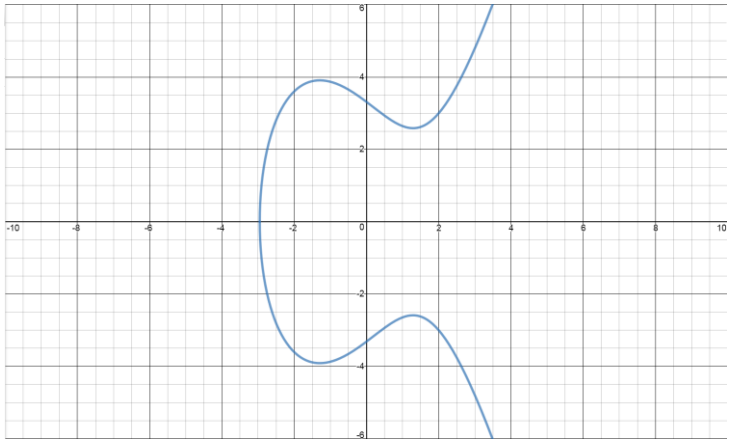
$$Q = n \cdot P = P \bigoplus \cdots \bigoplus P$$

Bisogna calcolare n, modulo p, conoscendo P e Q. La sicurezza di un sistema che usa un problema di logaritmo discreto su curve ellittiche dipende dalla scelta dell'operazione di gruppo $\bigoplus$ e dalla scelta di G. Nel nostro caso abbiamo che:

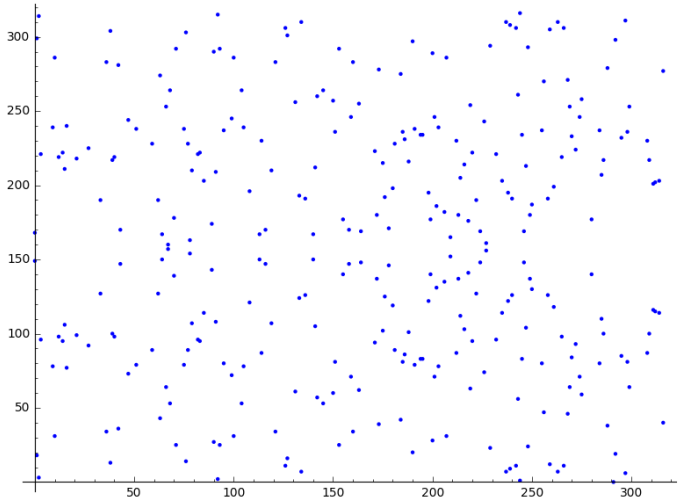
$$Q = P + \cdots + P = n \cdot P$$

Studio di una curva

Sia $p = 317$ e $E: y^2 = x^2 - 5x + 11$ una curva ellittica definita su $\mathbb{F}_{317}$



Curva su $\mathbb{R}$



Curva su $\mathbb{F}$

Studio di una curva

Campo reale:

$P = (3, 4.7958)$ $\quad Q = (-2.9054, 1)$

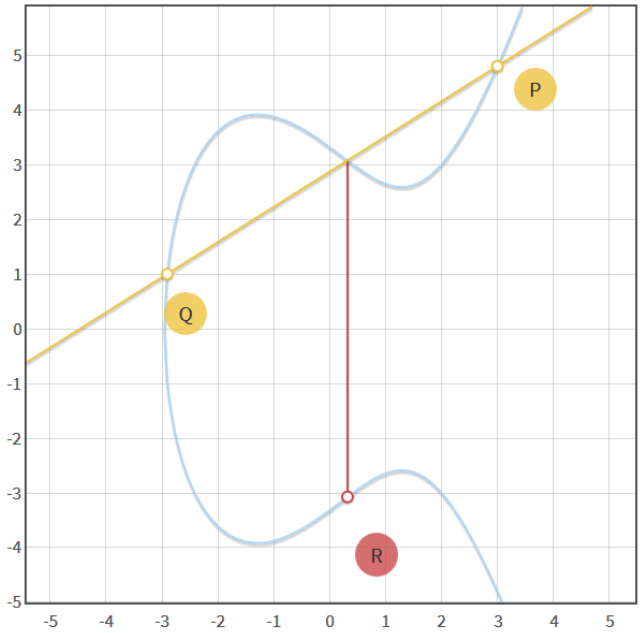
$R = P + Q = (x_3, y_3)$

$$x_3 = \left(\frac{4.7958 - 1}{2.9054 - 3}\right)^2 - 3 + 2.9054 = 0.3186$$

$$y_3 = \left(\frac{1 - 4.7958}{-2.9054 - 3}\right)(3 - 0.3186) - 4.7958$$

$$= -3.0723$$

$$R = (0.3186, -3.0723)$$



Studio di una curva

Campo finito:

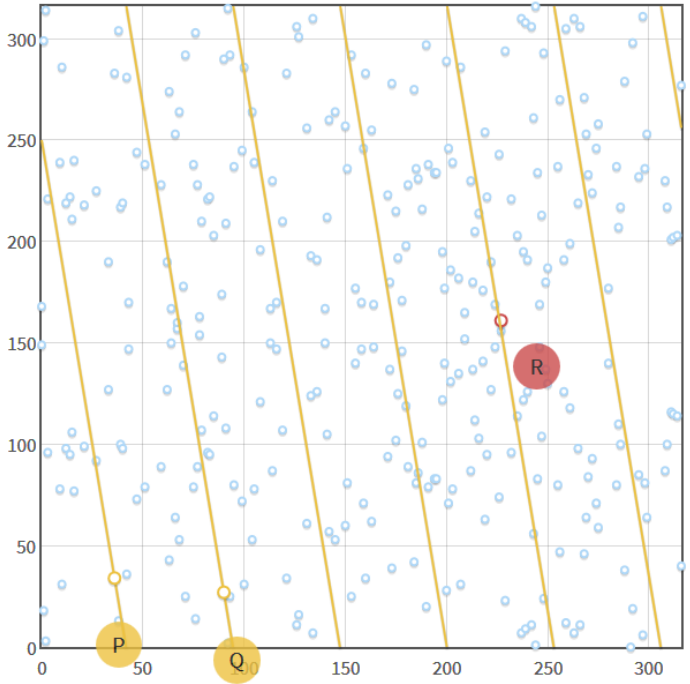
$P = (36,32)$ $\quad\quad$ $Q = (90,27)$

$R = P + Q = (x_3, y_3)$

$$x_3 = \left(\frac{34-27}{90-36}\right)^2 - 36 - 90 = 226 \; mod \; 317$$

$$y_3 = \left(\frac{27-34}{90-36}\right)(36-226) - 23 = 161 \; mod \; 317$$

$R = (227,161)$



Studio di una curva

Campo reale:

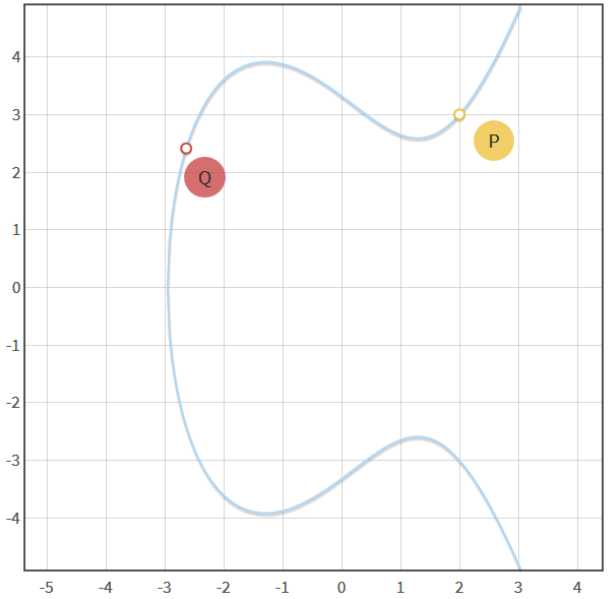
$P = (2,3)$

$R = 2 \cdot P = (x_3, y_3)$

$$x_3 = \left(\frac{3 \cdot 2^2 - 5}{2 \cdot 3}\right)^2 - 2 \cdot 2 = -2.6388$$

$$y_3 = \left(\frac{3 \cdot 2^2 - 5}{2 \cdot 3}\right)(2 + 2.6388) - 3 = 2.4120$$

$R = (-2.6388, 2.4120)$



Studio di una curva

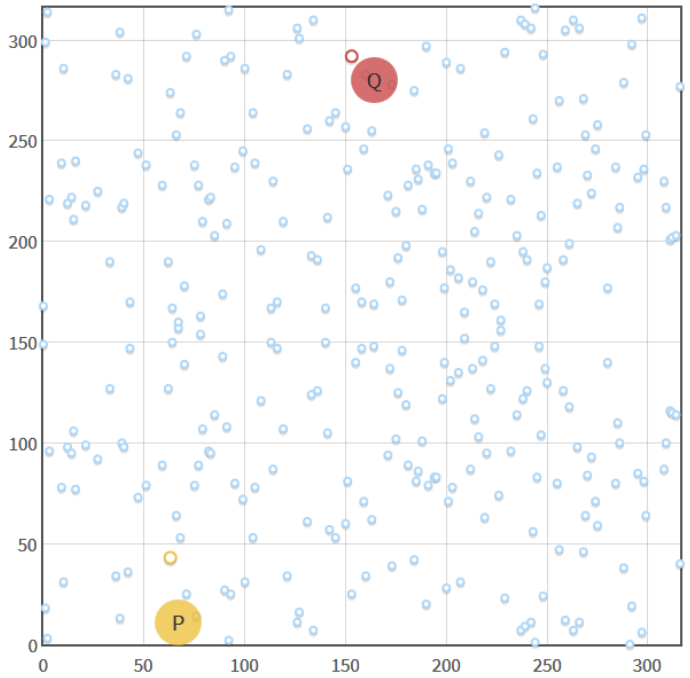
Campo finito:

$P = (63,43)$ $\qquad R = 2 \cdot P = (x_3, y_3)$

$$x_3 = \left(\frac{3 \cdot 63^2 - 5}{2 \cdot 43}\right)^2 - 2 \cdot 63 = 153 \; mod \; 317$$

$$y_3 = \left(\frac{3 \cdot 63^2 - 5}{2 \cdot 43}\right)(63 - 153) - 43 \; = 292 \; mod \; 317$$

$R \; = \; (153,292)$



Ordine del gruppo

Per stimare l'ordine del gruppo applichiamo il *teorema di Hasse-Weil* e otteniamo che:

$$282 \leq |E(\mathbb{F}_q)| \leq 353$$

Poichè la caratteristica della nostra curva è molto basso possiamo contare ogni singolo punto, ottenendo così 312.

Applicazioni di curve ellittiche

ECDSA

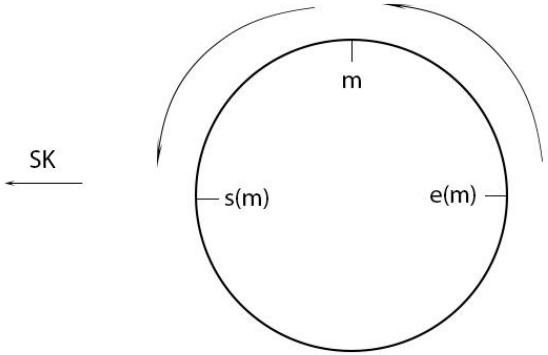
È un algoritmo che utilizza le curve ellittiche per firmare un messaggio tramite firma digitale.

Alice vuole inviare un messaggio a Bob e quest'ultimo vuole assicurarsi dell'identità del mittente. Concordano prima dei parametri che sono l'equazione della curva, il campo su cui e definita, il generatore della curva e l'ordine della curva

Alice genera una chiave private da e una chiave pubblica

Applicazioni di curve ellittiche

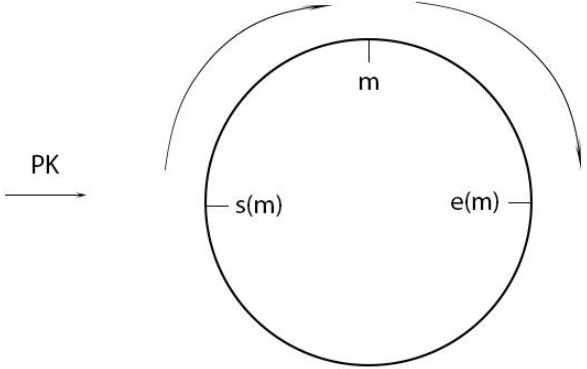
SecretKey(SK)



Firma del messaggio: $m \rightarrow s(m)$

Decrittazione del messaggio: $\mathrm{e}(m) \rightarrow m$

PublicKey(PK)



Verifica della firma: $s(m) \rightarrow m$

Crittazione del messaggio: $m \rightarrow e(m)$

Applicazioni di curve ellittiche

Firma

Alice firma il messaggio m eseguendo:

1. Calcola $e = HASH(m)$ tramite una funzione di hashing come SHA-256
2. Sia z l'insieme degli L_n bit più a sinistra di e dove L_n è la lunghezza in bit dell'ordine del gruppo n.
3. Seleziona in modo random un intero $k \in [1, n-1]$
4. Calcola il punto $(x_1, y_1) = k \cdot G$
5. Calcola $r = x_1 \ (mod\ n)$. Se $r = 0$ andare al punto 3
6. Calcola $s = k^{-1}\ (z\ +\ rd_A)\ (mod\ n)$. Se $s = 0$ andare al punto 3
7. La firma è data dalla coppia di valori (r, s)

Applicazioni di curve ellittiche

Verifica della firma

Per verificare la firma di Alice, Bob deve avere una copia del punto Q_A. Come prima cosa verifica che il punto si trovi sulla curva ed esegue i seguenti controlli:

- Controlla che Q_A non sia l'identità
- Controlla che Q_A si trovi sulla curva
- Controlla che $n \cdot Q_A = 0$

Successivamente si passa alla *verifica della firma digitale*.

1. Controlla che r ed s siano interi in $[1, n-1]$. Se non è cosi allora la firma non valida.
2. Calcola $e = HASH(m)$ dove la funzione di hash è la stessa usata per la firma
3. Calcola $w = s^{-1} \ (mod\ n)$
4. Calcola $u_1 = zw \ (mod\ n)$ e $u_2 = rw \ (mod\ n)$
5. Calcola il punto sulla curva $(x_1, y_1) = u_1 \cdot G + u_2 \cdot Q_A$. Se $(x_1, y_1) = \Theta$ allora la firma non è valida
6. La firma è valida se $r \equiv x_1 \ (mod\ n)$ altrimenti è invalida

Applicazioni di curve ellittiche

Verifica della correttezza dell'algoritmo

$$
\begin{aligned}
C &= u_1 \cdot G + u_2 \cdot G = (u_1 + u_2 d_A) \cdot G \\
&= (z s^{-1} + r d_A s^{-1}) \cdot G = (z + r d_A) s^{-1} \cdot G \\
&= (z + r d_A)(z + r d_A)^{-1} (k^{-1})^{-1} \cdot G \\
&= k \cdot G
\end{aligned}
$$

■

Calcolo del logaritmo discreto

Baby-step, Giant-step

Osservazione

Sia x un intero positivo. Possiamo scrivere x come:

$$x = am + b$$

con a, m, b interi. Possiamo quindi scrivere il problema del logaritmo discreto su EC come:

$$Q = xP$$

$$Q = (am + b)P$$

$$Q = amP + bP$$

$$Q - amP = bP$$

Baby-step, Giant-step

Algoritmo:

- Calcoliamo $m = \lceil \sqrt{n} \rceil$
- Per ogni b in $0, \dots, m$ calcoliamo bP e salviamo il tutto in una hash table
- per ogni a in $0, \dots, m$ calcoliamo:
  - amP
  - $Q - amP$
  - controlliamo nella hast table se esiste un punto tale che $Q - amP = bP$
  - se esiste allora abbiamo trovato $x = am + b$

Pollard-ρ

Sia G il gruppo nel quale si trovano i miei elementi, dividiamo G in tre sottoinsiemi S_1, S_2, S_3 di ordine più o meno uguale tra di loro. Supponiamo inoltre che $1 \notin S2$ e definiamo la sequenza $x_0, x_1, \dots x_n$ con $x_0 = 1$ e :

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i & se\ x_i \in S_1 \\ x_i^2 & se\ x_i \in S_2 \\ \alpha \cdot x_i & se\ x_i \in S_3 \end{cases}$$

con $i \geq 0$, $ord(G) = n$, α generatore di G e β contenuto in G. Questa sequenza a sua volta ne definisce altre due $a_0, a_1, \dots$ e $b_0, b_1, \dots$ che soddisfano $x_i = \alpha^{a_i}\beta^{b_i}$ per $i \geq 0$.

Pollard-ρ

Definiamo poi $a_0 = 0$, $b_0 = 0$ e per $i \geq 0$ abbiamo:

$$a_{i+1} = \begin{cases} a_i, & se\ x_i \in S_1 \\ 2a_i\ (mod\ n), & se\ x_i \in S_2 \\ a_i + 1\ (mod\ n), & se\ x_i \in S_3 \end{cases}$$

$$b_{i+1} = \begin{cases} b_i + 1\ (mod\ n), & se\ x_i \in S_1 \\ 2b_i\ (mod\ n), & se\ x_i \in S_2 \\ b_i & se\ x_i \in S_3 \end{cases}$$

Per trovare due elementi del gruppo x_i e x_{2i} tale che $x_i = x_{2i}$ possiamo utilizzare *l'algoritmo di Floyd*. Otteniamo cosi $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$ e quindi $\alpha^{a_{2i}-a_i} = \beta^{b_i-b_{2i}}$

$$\Rightarrow (b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i)\ (mod\ n)$$

Baby-step Giant-step vs Pollard-ρ vs Brute Force attack

Comparando gli algoritmi precedentemente elencati utilizzando come base una curva su un campo di ordine 10331 otteniamo i seguenti risultati:

```
Curve order: 10331
Using bruteforce
Computing all logarithms: 100.00% done
Took 2m 31s (5193 steps on average)
Using babygiantstep
Computing all logarithms: 100.00% done
Took 0m 6s (152 steps on average)
Using pollardsrho
Computing all logarithms: 100.00% done
Took 0m 21s (138 steps on average)
```

Baby-step Giant-step vs Pollard-ρ vs Brute Force attack

Utilizzando invece una curva su un campo di ordine 123779 otteniamo :

```
Curve order: 123779
Using bruteforce
Computing all logarithms: 100.00% done
Took 5h 51m 31s (61866 steps on average)
Using babygiantstep
Computing all logarithms: 100.00% done
Took 3m 56s (527 steps on average)
Using pollardsrho
Computing all logarithms: 100.00% done
Took 14m 11s (481 steps on average)
```

Grazie per l’attenzione