Crittografia su curve ellittiche

Gaspare Ferraro

24 Giugno 2017

UNIVERSITÀ DI PISA

Indice

1 Il problema della sicurezza

Con l'avanzare del progresso tecnologico, la diffusione della banda larga e un intenso utilizzo dell'informatica nella vita quotidiana, i temi della privacy e sicurezza acquistano maggiore importanza.

Per questo motivo necessitiamo di sistemi crittografici sempre più avanzati per garantire affidabilità nelle comunicazioni, un esempio è la crittografia su curve ellittiche, descritta in questa relazione, che rappresenta un'ottima alternativa al più noto metodo RSA [1] sotto molti aspetti.

Le curve ellittiche, particolari curve algebriche con le quali è possibile definire un sistema crittografico, sono ampiamente usate in famosi contesti come ad esempio il browser Google Chrome [2] e la criptovaluta Bitcoin.

I primi capitoli saranno dedicati ad una presentazione teorica delle curve ellittiche in matematica, verranno poi illustrati gli utilizzi pratici delle curve ellittiche in crittografia, mentre nell'ultimo capitolo si analizzeranno gli algoritmi implementati e confrontati con il metodo RSA dai punti di vista delle prestazioni e della sicurezza.

In appendice è presente la libreria scritta in linguaggio Java che implementa, a scopo didattico, gli algoritmi che verranno successivamente presentati.

[1] Algoritmo di crittografia asimmetrica, dalle iniziali dei suoi inventori (Rivest-Shamir-Adleman), basato sul problema della fattorizzazione.

[2] ECDSA for WebRTC: Better Security, Better Privacy and Better Performance: `https://developers.google.com/web/updates/2016/06/webrtc-ecdsa`

2 Curve Ellittiche

2.1 Definizioni preliminari

Per meglio comprendere la definizione di curva ellittica è necessario prima introdurre altre due definizioni.

Definizione 1 (Campo)**.** *Un campo è una struttura algebrica composta da un insieme non vuoto* $\mathbb{K}$ *e da due operazioni binarie interne, chiamate* $addizione$ *e* $moltiplicazione$*, per le quali valgano le proprietà associativa e commutativa e l'esistenza dell'elemento neutro e dell'inverso di ciascun elemento dell'insieme (tranne al più l'elemento neutro della moltiplicazione).*

Definizione 2 (Caratteristica di un campo)**.** *La caratteristica di un campo* $\mathbb{K}$ *è il più piccolo numero naturale* n *tale che, sommando* n *volte l'elemento neutro della moltiplicazione (indicato con il numero* 1*), si ottenga l'elemento neutro dell'addizione (indicato con il numero* 0*).*

$$\underbrace{1+1+...+1+1}_{n \text{ volte}} = 0 \tag{1}$$

Se tale numero n *non esiste, la caratteristica del campo è* 0 *per definizione.*

Diamo ora la definizione di curva ellittica:

Definizione 3 (Curva ellittica)**.** *Una curva ellittica* E *su un campo* $\mathbb{K}$ *è l'insieme dei punti* $(x, y) \in \mathbb{K}^2$ *che soddisfano l'equazione:*

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{2}$$

con $a, b, c, d, e \in \mathbb{K}$*.*
A questo insieme aggiugiamo un punto O *chiamato "punto all'infinito".*

Se la caratteristica del campo $\mathbb{K}$ è diversa da 2 e da 3 è possibile riscrivere l'equazione che definisce la curva ellittica in una forma più compatta (detta forma normale di Weierstrass):

$$y^2 = x^3 + ax + b \tag{3}$$

La cosa che rende interessanti le curve ellittiche in crittografia è la possibilità di definire una struttura algebrica di gruppo abeliano additivo. Tale struttura è caratterizzata dalla legge di gruppo, ovvero un'operazione binaria interna chiamata somma (tra punti di una curva ellittica), che gode delle proprietà di associatività e commutatività, con elemento neutro ed esistenza dell'inverso per ogni punto.

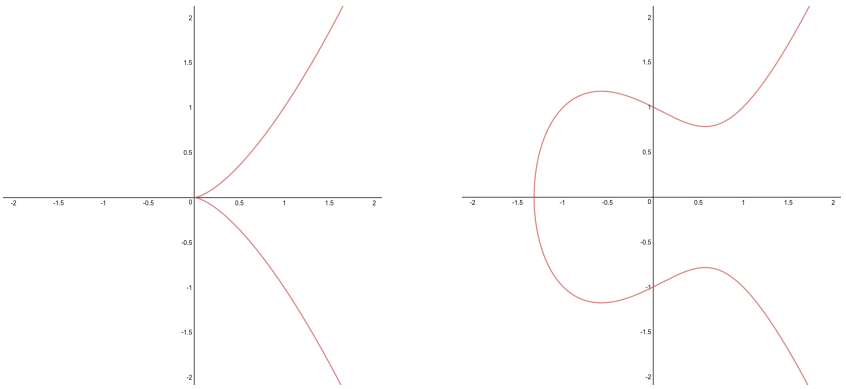
2.2 Curve ellittiche su $\mathbb{R}$

Analizziamo inizialmente le curve ellittiche definite sul campo $\mathbb{R}$, di più facile comprensione, ricordando dalla definizione 1 che il campo $\mathbb{R}$ ha caratteristica 0.

I punti di una curva ellittica su $\mathbb{R}$, di parametri $a, b \in \mathbb{R}$, sono i punti dell'insieme $E(a, b)$ definito come:

$$E(a,b) = \{\, (x,y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b \,\} \tag{4}$$

Con l'assunzione che $4a^3 + 27b^2 \neq 0$, condizione necessaria per la non singolarità della curva, ovvero l'assenza di punti non singolari dove non è definita una tangente.



(a) Curva ellittica singolare $y^2 = x^3$ (b) Curva ellittica non singolare $y^2 = x^3 - x + 1$

Possiamo ora dare una spiegazione intuitiva della somma in modo geometrico: considerati due punti $P, Q \in E(a, b)$ con $Q \neq P$ e $Q \neq O$, la somma $P + Q = R$ si ottiene tracciando una retta passante per i punti P e Q, che interseca la curva in un punto R'. Il punto R cercato è quindi il punto simmetrico a R' rispetto all'asse delle ascisse, un esempio di tale procedimento:

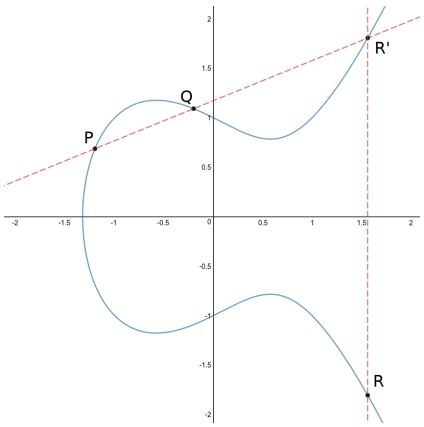


Figura 2: Somma dei punti P e Q

Nel caso $Q = P$, la somma $P + P = 2P$ (raddoppio del punto P) è, invece, ottenuta tracciando la retta tangente passante per P (che esiste sempre, poichè la curva è non singolare per assunzione), analogalmente alla somma generale tale retta interseca la curva in un punto R' del quale si prende il punto simmetrico R rispetto all'asse delle ascisse. Un esempio in figura:

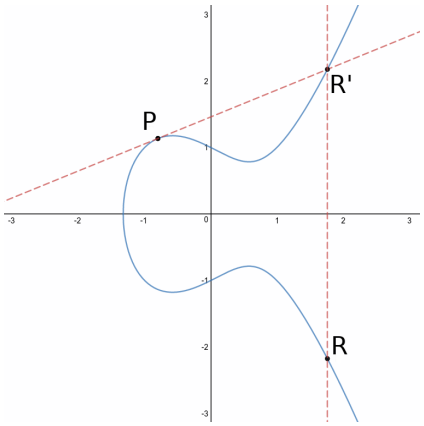


Figura 3: Raddoppio del punto P

Infine, nel caso in cui $Q = O$ (somma del punto P con il punto O), la retta tracciata è la verticale passante per P' che è il simmetrico P rispetto alle ascisse, il quale punto simmetrico è nuovamente P stesso.

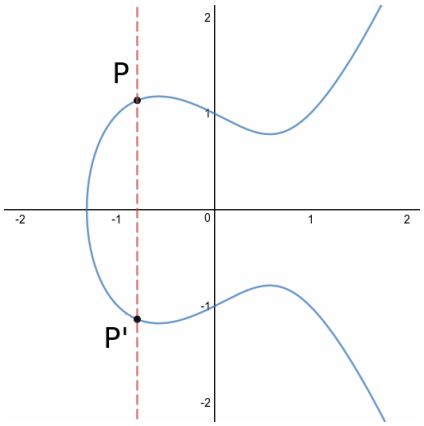


Figura 4: Raddoppio del punto P

Più formalmente per la somma valgono le seguenti proprietà:

1. **Chiusura**: $\forall P, Q \in E(a, b)$, $P + Q \in E(a, b)$

2. **Identità**: $\forall P \in E(a,b),\ P + O = P$

3. **Opposto**: Sia $P = (x,y) \in E(a,b)$ il suo opposto $-P \in E(a,b)$ è il punto di coordinate $(x,-y)$

4. **Somma**: Siano $P = (x_1,y_1), Q = (x_2,y_2) \in E(a,b)$ con $P \neq Q$
   allora $P + Q = R$ con $R = (x_3,y_3) \in E(a,b)$ dove:

$$\begin{aligned} x_3 &= \Big(\frac{y_2 - y_1}{x_2 - x_1}\Big)^2 - x_1 - x_2 \\ y_3 &= \Big(\frac{y_2 - y_1}{x_2 - x_1}\Big)(x_1 - x_3) - y_1 \end{aligned} \tag{5}$$

5. **Raddoppio**: Sia $P = (x_1,y_1) \in E(a,b)$ con $P \neq -P$

   allora $P + P = R$ con $R = (x_2,y_2) \in E(a,b)$ dove:

$$\begin{aligned} x_2 &= \Big(\frac{3x_1^2 + a}{2y_1}\Big)^2 - 2x_1 \\ y_2 &= \Big(\frac{3x_1^2 + a}{2y_1}\Big)(x_1 - x_2) - y_1 \end{aligned} \tag{6}$$

6. **Associatività**: $\forall P,Q,R \in E(a,b), P + (Q + R) = (P + Q) + R$

7. **Commutatività**: $\forall P,Q \in E(a,b), P + Q = Q + P$

2.3 Curve ellittiche su campi finiti

Più importanti in ambito crittografico ma meno intuitive, sono le curve ellittiche definite sui campi finiti.

Definizione 4 (Campo finito)**.** *Un campo finito, detto anche campo di Galois, è un campo con un numero finito di elementi.*

In particolare ogni campo finito ha p^n elementi, con p primo e n naturale e, a meno di isomorfismo, esiste unico campo finito con p^n elementi, che indicheremo con $\mathbb{F}_{p^n}$ o $GF(p^n)$ (dall'inglese Galois Field).

Un campo finito $\mathbb{F}_{p^n}$ ha caratteristica uguale a p, senza perdita di generalità consideriamo solo il caso in cui $p > 3$ che ci permette di scrivere l'equazione generale nella forma normale di Weierstrass.

Una curva ellittica su un campo finito è quindi l'insieme dei punti definito come:

$$E_{p^n}(a,b) = \{\,(x,y) \in \mathbb{F}_{p^n}^2 \mid y^2 = x^3 + ax + b\,\} \cup \{\,O\,\} \tag{7}$$

Per semplicità di operazioni consideriamo le curve con $n = 1$, chiamate curve prime, in quanto esiste un isomorfismo $\mathbb{F}_p \cong \mathbb{Z}_p$. In questo campo le operazioni sono quelle dell'algebra modulare, le curva prime sono definite quindi come:

$$E_p(a,b) = \{\,(x,y) \in \mathbb{Z}_p^2 \mid y^2 = x^3 + ax + b \mod p\,\} \cup \{\,O\,\} \tag{8}$$

Possiamo notare come le formule definite precedentemente per il campo reale valgano anche per le curve sui campi finiti, ovviamente considerando le operazioni sui relativi campi.

I grafici di tali insiemi non sono più delle curve continue, come visto precedentemente, ma un insieme finito di punti che soddisfano la definizione, un esempio pratico:

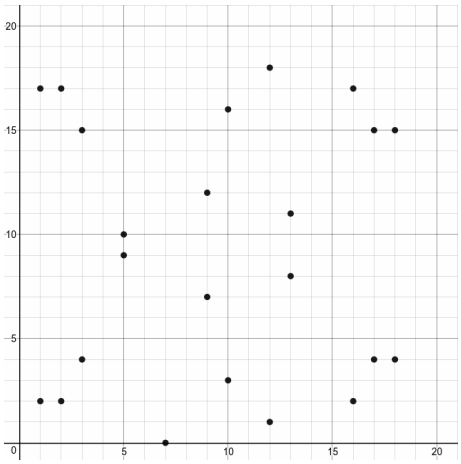


Figura 5: Curva $E_{19}(-7, 10)$

Oltre alle curve prime, nella crittografia su curve ellittiche, si utilizzano inoltre le cosiddette curve binarie, definite sul campo $\mathbb{F}_{2^m}$, i cui elementi possono essere visti come tutti gli interi a m cifre binarie. Avendo $\mathbb{F}_{2^m}$ caratteristica 2, non è possibile esprimerla con la forma normale di Weierstrass; la curva è quindi definita come:

$$E_{2^m}(a,b) = \{\,(x,y) \in \mathbb{F}_{2^m}^2 \mid y^2 + xy = x^3 + ax^2 + b\,\} \tag{9}$$

Le curve binarie si prestano ad una migliore implementazione su sistemi hardware, in quanto le operazioni binarie sono facilmente rappresentabili con l'utilizzo di porte logiche mentre al contrario, le curve prime, si prestano ad una migliore implementazione su sistemi software. Dal punto di vista teorico tutte le operazioni e algoritmi successivamente descritti si applicano indistintamente ad entrambi i tipi di curve.

Dalla teoria dell'algebra modulare sappiamo che non sempre esiste la radice quadrata di un numero, in generale in $\mathbb{Z}_p$ esattamente $(p-1)/2$ elementi sono residui quadratici. Ricordando che le curve sono definite per y^2 e che non tutti i valori di x danno luogo ad un residuo quadratico oppure che valori distinti di x possono generare lo stesso residuo quadratico, notiamo che alcuni punti delle curve non possono essere definiti.

Un parametro importante, basato su questi fatti, che useremo in seguito è l'ordine di una curva.

Definizione 5 (Ordine di una curva ellittica)**.** *L'ordine N di una curva ellittica $E_{p^n}(a,b)$ è la cardinalità dell'insieme, ovvero il numero dei punti che la compongono.*

Una curva prima $E_p(a,b)$ può evidentemente avere un massimo di $2p+1$ punti: tutte le p coppie di punti $(x, \pm y)$ che soddisfano $y^2 = x^3 + ax + b$ e il punto all'infinito.

Una stima più accurata sul limite superiore dell'ordine di una curva è data dal seguente teorema:

Teorema 1 (Hasse)**.** *L'ordine N di una curva ellittica $E_{p^n}(a,b)$ verifica la disuguaglianza $|N - (p^n+1)| \leq 2\sqrt{p^n}$*

Mentre non è nota nessuna limitazione inferiore non banale.

3 Il problema del logaritmo discreto

Per definire un sistema crittografico sulle curve ellittiche, è necessario prima di tutto trovare una funzione one-way trap-door che ne garantisca la sicurezza, ricordiamo che una funzione one-way trap-door è una funzione facile da calcolare ma molto difficile da invertire se non si è a conoscenza di particolari informazioni.

Nel metodo RSA la funzione one-way trap-door è la fattorizzazione: dati due numeri primi p e q, calcolare il loro prodotto $n = pq$ è facile, mentre trovare i due fattori primi conoscendo solo n è un'operazione molto complessa per la quale non si conoscono algoritmi efficienti, conoscendo invece uno dei due fattori ricavare il secondo è banale.

In modo analogo all'algebra modulare è possibile dare una definizione di logaritmo discreto per le curve ellittiche considerando l'operazione di prodotto scalare:

Definizione 6 (Logaritmo discreto per curve ellittiche)**.** *Dati due punti di una curva ellittica P e Q chiamiamo, dove è definito, l'intero k per cui $Q = kP$, logaritmo in base P del punto Q.*

Notiamo che è possibile calcolare il prodotto kP in modo efficiente utilizzando, invece che $\Theta(k)$ somme, solamente $\Theta(log_2 k)$ operazioni di raddoppio e $O(log_2 k)$ somme grazie all'algoritmo dei raddoppi successivi:

Algoritmo 1 Calcolo efficiente del prodotto $Q = kP$

Input: intero k, punto P
Output: punto Q

```
1: function fastMult(k, P)
2:     Q ← P
3:     k ← k − 1
4:     while k > 0 do
5:         if k mod 2 = 1 then
6:             Q ← Q + P
7:         k ← k/2
8:         P ← 2P
9:     return Q
```

Contrariamente al prodotto, l'operazione inversa che consiste nel trovare l'intero k conoscendo i punti P e Q, conosciuta come problema del logaritmo discreto per curve ellittiche (ECDLP), non ha algoritmi efficienti noti per risolverlo.

Definiamo infine un altro parametro importante in crittografia: l'ordine n di punto P (da non confondere con l'ordine N della curva):

Definizione 7. *Ordine di un punti L'ordine di punto P è il più piccolo intero positivo n tale che $nP = O$ (ovviamente $n \leq N - 1$, con N l'ordine della curva).*

3.1 Attacchi al problema

I possibili attacchi noti al problema si possono dividere in due categorie: gli algoritmi general-purpose e quelli special-purpose che sfruttano particolari assunzioni per attaccare determinate classi di curve.

Il metodo più famoso è quello chiamato metodo ρ di Pollard, general-purpose, che sfrutta il paradosso dei compleanni per trovare, in $O(\sqrt{N})$ con una probabilità del 50%, quattro interi n_1, n_2, n_3, n_4 tali che $n_1P + n_2Q = n_3P + n_4Q$.
In questo modo sia ha che $k = (n_1 - n_3)/(n_2 - n_4) \mod n$, con n ordine del punto P.

Altri metodi più complessi special-purpose sono applicabili alle cosiddette curve anomale, con lo scopo di trovare particolari isomorfismi che rendono più facili le operazioni.

3.2 Scelta della curva

Gli attacchi noti e altri teoremi matematici sulle curve ellittiche ci forniscono fortunatamente alcuni degli accorgimenti [2] nella scelta di curve $E_q(a, b)$ e dell'utilizzo di particolari punti P di ordine N, come ad esempio:

- q possibilmente numero primo di Merseinne (ovvero esprimibile nella forma $2^p - 1$ per qualche primo p).
- Uso del coefficiente $a = -3$, per ragioni di efficienza, secondo lo standard IEEE P1363 [3].
- Il fattore più grande di N deve essere diverso da q.
- Il numero N deve avere un fattore r tale che un numero di operazioni pari a $\sqrt{r}$ risulti intrattabile.

Per facilitare la scelta, l'agenzia governativa statunitense NIST [3], ha pubblicato nel 1999 un documento [4] contenente una lista di curve e punti standard da usare, privi di difetti dagli attacchi noti.

[3] National Institute of Standards and Technology

4 Crittografia su curve ellittiche

Andiamo ora ad illustrare i più famosi algoritmi di crittografia basati sulle curve ellittiche, assumendo che i messaggi da criptare siano dei numeri interi che rappresentano, in qualche codifica, le informazioni da trasmettere.

4.1 Codifica dei messaggi

Siccome tutte le operazioni viste finora, sulle curve ellittiche, coinvolgono solo punti appartenenti alle curve e non numeri interi, occorre prima trasformare il messaggio m in un punto P della curva prima o binaria. Purtroppo non sono noti algoritmi deterministici polinomiali, tuttavia esistono degli algoritmi randomizzati molto efficienti che hanno una probabilità bassa di non trovare punti sulla curva. Uno dei più famosi che illustriamo è l'algoritmo di Koblitz.

Algoritmo 2 Codifica di un messaggio m in un punto P della curva $E_p(a, b)$

Input: m messaggio; p, a, b parametri della curva prima; h intero positivo fissato tale che $(m + 1)h < p$
Output: punto $P_m = (x, y)$ o Fallimento

```
1:  function KOBLITZ(m, h, p, a, b)
2:      x ← mh
3:      i ← 0
4:      while i < h do
5:          z ← x^3 + ax + b  mod p
6:          if z residuo quadrato modulo p then
7:              y ← radice quadrata di z modulo p
8:              P ← (x, y)
9:              return P
10:         x ← x + 1
11:         i ← i + 1
12:     return Fallimento
```

Siccome la probabilità che un numero sia residuo quadrato modulo p è circa $1/2$, la probabilità di successo dell'algoritmo è circa $1 - 2^{-h}$.
La decodifica del punto $P_m = (x, y)$ è molto semplice: $m = \lfloor x/h \rfloor$.

L'algoritmo per semplicità usa a parole le due funzioni per la verifica del residuo quadrato e per l'estrazione della radice, tali funzioni sono facilmente implementabili se si lavora modulo un numero primo [11].

Per esempio se $p = 3 \mod 4$ si può sfruttare il Criterio di Eulero: z è un residuo quadrato se $z^{(p-1)/2} = 1 \mod p$ e la sua radice quadrata è $y = z^{(p+1)/4} \mod p$. Simili funzioni esistono per i primi p che non soddisfano la precedente condizione.

4.2 Scambio di chiavi

I cifrari simmetrici e asimmetrici hanno vantaggi e svantaggi che li caratterizzano, i primi sono più veloci ma anche più vulnerabili in quanto è presente solo una chiave privata, i secondi invece permettono di scambiare chiavi private in maniera sicura ma sono più onerosi come risorse di calcolo.

Un terzo tipo di cifrario, detto ibrido, unisce i vantaggi di entrambi i cifrari: lo scambio delle chiavi avviene mediante cifrario asimmetrico e quello dei messaggi mediante cifrario simmetrico.

Per lo scambio sicuro delle chiavi si utilizza una variante dell'algoritmo DH (dai nomi dei creatori Diffie-Hellman) che sfrutta le curve ellittiche, chiamato ECDH (Elliptic Curve Diffie-Hellman).

L'ECDH richiede che due utenti X e Y si accordino pubblicamente su un campo finito e una curva ellittica definita su tale campo, nonchè un punto B di ordine n molto grande. In alternativa è consigliato usare, come precedentemente scritto, una delle curve ellittiche raccomandate dal NIST[4].

L'algoritmo genera la chiave S in due passaggi svolti in contemporanea dai due utenti:

- X e Y scelgono casualmente due interi positivi $n_X, n_Y < n$ che rappresentano le rispettive chiave private. Le corrispondenti chiavi pubbliche $P_X = n_X B$ e $P_Y = n_Y B$ vengono reciprocamente scambiate.
- X e Y, che ricevono rispettivamente i punti P_Y e P_X, calcolano entrambi il punto S come $n_X P_Y = S$ e $n_Y P_X = S$.
- Infine sia X che Y condividono lo stesso punto $S = (x_S, y_S)$, ottenuto tramite scelte casuali di entrambi, possono trasformarlo in una chiave segreta k per la cifratura simmetrica. Ad esempio ponendo $k = x_S \mod 2^{256}$.

Tale protocollo di scambio chiavi è al sicuro da attacchi di tipo man-in-the-middle passivi: intercettando le chiavi pubbliche P_X e P_Y durante le comunicazioni per ottenere le chiavi private n_X e n_Y sarebbe necessario risolvere il problema del logaritmo discreto dei punti P_X e P_Y modulo B.

4.3 Scambio di messaggi

Per il classico scambio di messaggi cifrati, si può utilizzare una variante del cifrario a chiave pubblica di ElGamal che sfrutta le curve ellittiche.

Anche in questo caso i due utenti, che vogliono comunicare in maniera sicura, devono accordarsi su una curva ellittica e un punto B della curva con ordine n elevato, inoltre. se si utilizza la codifica di Koblitz per la trasformazione dei messaggi, sull'intero h scelto.

Gli utenti X e Y generano le chiavi private, in modo analogo a quanto visto prima, e si scambiano le chiavi pubbliche.

Le due operazioni sono quindi definite come:

Algoritmo 3 Cifratura del messaggio m nella coppia di punti $< V, W >$

Input: m messaggio; p, a, b parametri della curva prima; B punto base; h per la codifica del messaggio; P_D chiave pubblica del destinatario
Output: $< V, W >$ Coppia di punti

1: **function** CIFRA(m, p, a, b, B, h, P_D)
2: $P_m \leftarrow Koblitz(m, h, p, a, b)$
3: $r \leftarrow$ randomInt()
4: $V \leftarrow rB$
5: $W \leftarrow P_m + rP_D$
6: **return** $< V, W >$

Algoritmo 4 Decifratura della coppia di punti $< V, W >$ nel messaggio m

Input: $< V, W >$ coppia di punti; p, a, b parametri della curva prima; B punto base; h per la decodifica del messaggio; n_D chiave privata del destinatario
Output: m messaggio decifrato

1: **function** DECIFRA(V, W, p, a, b, B, h, n_D)
2: $P_m \leftarrow W - n_D V$
3: $m \leftarrow \lfloor x_{P_m}/h \rfloor$
4: **return** m

L'operazione di decodifica del punto P_m è corretta in quanto:
$W - n_D V = P_m + rP_D - n_D(rB) = P_m + r(n_D B) - n_D(rB) = P_m$.
Inoltre anche intercettando la coppia di punti $< V = rB, W = P_m + rP_D >$ per ricavare P_m si dovrebbe calcolare r da V, ovvero risolvere il problema del logaritmo discreto.

4.4 Firma Digitale

Infine, una funzionalità molto importante consiste nella firma digitale di un documento elettronico, che sostituisce la più classica, ma meno sicura, firma manuale. Tale operazione deve in soddisfare quattro requisiti:

- La firma non deve essere falsificabile.
- La firma non può essere riutilizzata ma legata strettamente al documento firmato.
- Il documento non è alterabile, ovvero bisogna avere la sicurezza che la firma sia riferita al documento nella sua forma originale.
- La firma non può essere ripudiata, ovvero rappresenta una prova legale di appartenenza dell'autore.

Le operazioni che un algoritmo di firma digitale deve fornire sono due:

- Firma. Dato un documento m genera una firma f.
- Verifica. Dato un documnet m e una firma f ne controlla l'autenticità.

Poichè spesso i documenti sono di dimensioni importanti, risulta poco efficiente firmare direttamente il documento, per questo si preferisce utilizzare una funzione hash sicura che generi un'impronta di dimensioni ridotte rispetto al documento originale.

Esempi famosi di funzioni hash ampiamente usate sono l'MD5 e lo SHA-1, anche se entrambe le funzioni non sono più ritenute sicure, la prima da parecchi anni mentre la seconda recentemente.

Esistono diversi metodi per la firma digitale che sfruttano sia cifrari simmetrici che asimmetrici, illustriamo una variante del DSA (Digital Signature Algorithm) che sfrutta le curve ellittiche chiamata ECDSA[4] (Elliptic Curve Digital Signature Algorithm).

[4]Tale algoritmo è diventato standard ISO (14888) nel 1998, standard ANSI (X9.62) nel 1999 e standard IEEE (P1363 2) nel 2000 ed è ormai ampiamente utilizzato in tutti gli ambiti: dai Browser desktop e mobile alla criptovaluta Bitcoin.

Le due funzioni di firma e verifica sono le seguenti:

Algoritmo 5 Firma digitale di un messaggio m

Input: m messaggio da verificare; p, a, b parametri della curva prima; B punto base; n ordine del punto B; $hash$ funzione di hash; n_D chiave privata
Output: (r, s) firma del messaggio

1: **function** FIRMA($m, p, a, b, B, n, hash, n_D$)
2: $k \leftarrow randomInt(1, n-1)$
3: $Q \leftarrow kB$
4: $r \leftarrow x_Q \mod n$
5: **if** $r = 0$ **then** ritorna al punto 2
6: $e \leftarrow hash(m)$
7: $s \leftarrow k^{-1}(e + rn_D) \mod n$
8: **if** $s = 0$ **then** ritorna al punto 2
9: **return** (r, s)

Algoritmo 6 Verifica di una firma digitale (r, s) per un messaggio m

Input: m messaggio da verificare; (r, s) firma digitale; p, a, b parametri della curva prima; B punto base; n ordine del punto B, $hash(m)$ funzione di hash, P_D chiave pubblica
Output: Accetta o Rifiuta

1: **function** VERIFICA($m, p, a, b, B, n, hash, P_D$)
2: **if** $!(1 \leq r, s \leq n-1)$ **then**
3: **return** Rifiuta
4: $e \leftarrow hash(m)$
5: $w \leftarrow s^{-1} \mod n$
6: $u_1 \leftarrow ew \mod n$
7: $u_2 \leftarrow rw \mod n$
8: $Q \leftarrow u_1 B + u_2 P_D$
9: **if** $Q = 0$ **then**
10: **return** Rifiuta
11: $v \leftarrow x_Q \mod n$
12: **if** $v \neq r$ **then**
13: **return** Rifiuta
14: **return** Accetta

5 Testing

Passiamo ora ad una analisi pratica della libreria Java scritta e presente in appendice.

5.1 Scelte implementative

Per lavorare nei campi finiti è stata creata una classe FinitePrimeField che implementa le operazioni necessarie dell'algebra modulare; in particolare l'operazione più complicata, il calcolo della radice modulo primo, viene implementata distinguendo tre diversi casi in base al numero primo e sfruttando il Criterio di Eulero e l'algoritmo di Tonelli-Shanks.

Le curve prime vengono modellate dalla classe PrimeCurve che necessita del campo finito sul quale si vuole lavorare, dei parametri a e b dell'equazione e del valore h per la codifica di Koblitz. La classe implementa inoltre le operazioni di somma tra due punti, prodotto scalare, opposto di punto e ricerca di un punto data la ascissa.

Per rappresentare un generico punto su una curva ellittica è stata inoltre definita la classe Point che contiene le coordinate x e y del punto oltre al tipo di punto che si sta rappresentando (differenziando tra punto valido, non valido e punto all'infinito).

I diversi algoritmi presentati sono implementati nella classe ECC, in particolare per l'ECDH è stata creata una classe ECDHKey per modellare la coppia chiave privata, chiave pubblica.

La classe Main definisce i parametri delle curve e contiene le batterie di test e benchmark che vedremo più avanti.

5.2 Correttezza

Per verificare la correttezza dei diversi algoritmi, sono stati effettuati dei test 5 curve ellittiche distinte: 3 curve con primo piccolo ($E_{19}(-7, 10)$, $E_{29}(-7, 10)$ e $E_{97}(-7, 10)$), per coprire i tre diversi casi nel calcolo delle radici quadrate e 2 curve raccomandate dal NIST ($P - 192$ e $P - 521$).

Per ogni funzione e per ogni curva vengono effettuati un numero elevato test usando interi o punti casuali al fine di coprire ogni possibile caso, in particolare ogni funzione viene testata in modo diverso:

- Il campo finito viene testato controllando la correttezza di tutte le operazioni implementate.
- La curva ellittica viene testata effettuando diverse operazioni tra punti della curva, coprendo tutti i casi della somma tra due punti.
- La codifica di Koblitz viene testata codificando e decodificando il messaggio e controllando se l'informazione è corretta.
- Lo scambio di messaggi con ElGamal viene testato codificando e codificando il messaggio e controllando se l'informazione è corretta.
- Lo scambio di chiavi con l'ECDH tra due utenti viene testato simulando le due fasi con lo scambio delle chiavi pubbliche e confrontando se al termine delle operazioni entrambi gli utenti hanno in possesso lo stesso punto.
- La firma digitale con l'ECDSA viene testata firmando un messaggio e verificando l'integrità prima con il messaggio originale e poi con un messaggio alterato.

Dai test pratici si evince che tutte le funzioni implementate sono corrette.

5.3 Prestazioni

Per valutare le prestazioni degli algoritmi implementati, sono stati effettuati numerosi test, sulle 5 curve usate, per stimare il numero medio di operazioni al secondo su una macchina di uso quotidiano. I risultati pratici sono:

Algoritmo	$E_{19}(-7, 10)$	$E_{29}(-7, 10)$	$E_{97}(-7, 10)$	$P - 192$	$P - 521$
Koblitz	288515	253811	252046	5850	583
ElGamal	8412	3724	3654	354	88
ECDH	108973	60506	41847	1263	341
ECDSA	1	1	1	1	1

Tabella 1: Confronto delle prestazioni, espresse in operazioni/secondi, degli algoritmi implementati lavorando con le 5 curve.

Da come si può notare dalla tabella, le operazioni più veloci sono quelle della codifica di Koblitz e per lo scambio delle chiavi con l'ECDH, mentre lo scambio di messaggi con ElGamal e la firma digitale con l'ECDSA risultano operazioni più complesse.

5.4 Confronto con RSA

Un primo confronto, tra i due sistemi crittografici, può essere fatto confrontando a parità di livello di sicurezza, definito come il costo computazionale per forzare un sistema, la dimensione in bit delle chiavi.

Il NIST, in uno dei suoi studi[5] riguardo alla sicurezza dei diversi sistemi, ha pubblicato una tabella che confronta, a parità di livello di sicurezza, i bit necessari per il modulo nell'RSA con i bit necessari nell'ordine del punto base nell'ECC:

RSA e DH (bit del modulo)	ECC (bit dell'ordine)
1024	160
2048	224
3072	256
7680	384
15360	512

Tabella 2: Confronto, a parità di sicurezza, del numero di bit tra i due sistemi

Ovviamente chiavi di dimensione più piccola assicurano vantaggi dal punto di vista delle prestazioni, hardware che necessita di meno risorse ed elaborazioni che avvengono in tempi brevi e spazi ridotti, caratteristiche che rendono quindi l'ECC adatto a dispositivi a ridotte performance.

Confrontiamo ora le prestazioni pratiche, testando su una macchina di uso quotidiano [5], quante operazioni di firma e verifica al secondo possono svolgere i due sistemi con chiavi di diverse dimensione.

A tale scopo, per ridurre i possibili errori legati a diverse implementazioni, usiamo le funzioni di benchmark messe a disposizione da OpenSSL [6], standard de facto per la sicurezza nella comunicazioni.

Sistema usato	Firme al secondo	Verifiche al secondo
RSA-512	88888	1333333
RSA-1024	30429	500000
RSA-2048	4449	146825
RSA-4096	619	39802
ECC-160	80000	22009
ECC-192	68975	18330
ECC-224	51781	13126
ECC-256	103913	47058
ECC-384	23088	5470
ECC-521	10929	2497

Tabella 3: Confronto delle prestazioni di firma e verifica tra i due sistemi utilizzando parametri di diversa dimensione.

Notiamo come per l'RSA il numero di operazioni al secondo decresca molto velocemente a confronto con l'ECC, tenendo anche conto il precedente confronto a parità di sicurezza.

Infine notiamo anche come per l'ECC-256 abbiamo un maggiore numero di operazioni al secondo, non a caso, la criptovaluta Bitcoin, utilizza questo tipo di curva [7] che presenta particolari proprietà che permettono una implementazione molto efficiente.

[5] Un portatile con processore intel i7-7700HQ e 16GB di memoria RAM

[6] Libreria open-source che implementa i protocolli SSL e TLS oltre a svariate funzioni crittografiche (`https://www.openssl.org`)

[7] In particolare l'insieme dei parametri usati è noto come Secp256k1 (`https://en.bitcoin.it/wiki/Secp256k1`)

Riferimenti bibliografici

[1] Anna Bernasconi, Paolo Ferragina, Fabrizio Luccio, *Elementi di crittografia*, Università di Pisa, Ottobre 2014.

[2] Daniel R. L. Brown, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Certicom Research, 2010.

[3] P1363 Working Group, *IEEE P1636.3™/D1 Draft Standard for Identity-based Public-key Cryptography Using Pairings*, New York, IEEE, 2008.

[4] NIST, *Recommended elliptic curves for federal government use*, 1999.

[5] NIST, *Recommendation for Key Management – Part 1: General (Revision 3)*, 2012.

[6] Emanuele Bellini, *La crittografia a curve elittiche e applicazioni*, Università degli studi di Trento, 2011.

[7] Battaglia Ilenia, *Le curve elittiche in crittografia*, Università degli studi di Palermo, 2006.

[8] Gerardo Pelosi, *Note di Crittografia su Curve Ellittiche*, Politecnico di Milano, 2003.

[9] Margherita Lazzarini, *Crittografia basata su curve ellittiche e implementazione di funzioni di libreria per Cryptokit*, Università di Bologna, 2012.

[10] Kenneth H. Rosen, *Elliptic Curves: Number theory and cryptography*, University of Maryland, 2008

[11] Laila Akif, *Algoritmi per il calcolo di radici quadrate nei campi finiti su risorse di calcolo massivamente parallele*, Università di Roma Tre, 2014

A Libreria Java

Una semplice implementazione in Java degli algoritmi precedentemente illustrati è pubblicamente disponibile su GitHub all'indirizzo `https://github.com/GaspareG/EccLib`

A.1 Campo finito

`https://github.com/GaspareG/EccLib/blob/master/src/FinitePrimeField.java`

A.2 Curva prima

`https://github.com/GaspareG/EccLib/blob/master/src/PrimeCurve.java`

A.3 Punto di una curva

`https://github.com/GaspareG/EccLib/blob/master/src/Point.java`

A.4 Codifica di Koblitz

Encrypt: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L7`

Decrypt: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L22`

A.5 ECDH

Prima fase: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L50`

Seconda fase: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L58`

A.6 ElGamal

Encrypt: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L30`

Decrypt: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L40`

A.7 ECDSA

Firma: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L65`

Verifica: `https://github.com/GaspareG/EccLib/blob/master/src/ECC.java#L83`

A.8 Testing

https://github.com/GaspareG/EccLib/blob/master/src/Main.java