Università degli Studi dell'Aquila
Facoltà di Ingegneria

Tesi di Laurea in Ingegneria Informatica e Automatica

Crittografia Ellittica

Relatore:
Prof. Gabriele Di Stefano

Laureando:
Marco Carolla

Anno Accademico 2016-2017

Indice

Capitolo 1

Introduzione alla Sicurezza Informatica

Con la parola composta da "*scritto*" e "*grafia*" intendiamo quella tematica della sicurezza in generale che mira ad una scrittura segreta, tale che non possa essere letta senza conoscere l'artificio usato nella corrispondenza. Parliamo di scrittura cifrata quando il risultato della lettura di un testo è privo di significato apparente. La cifratura di un testo può essere fatta per:

- "*trasposizione*" mediante la quale la traduzione del linguaggio chiaro in linguaggio segreto ha luogo con spostamento o inversione degli elementi in chiaro
- "*sostituzione*" degli elementi del linguaggio con segni convenzionali che pur potendo essere di qualsiasi genere, sono in pratica quelli della normale scrittura, lettere o cifre
- "*sistema misto*" che combina le precedenti interponendole una dopo l'altra

Un esempio di crittografia lo troviamo nell'Antica Grecia per opera di Polibio. Una sua idea di crittografia consisteva nel disporre le lettere dell'alfabeto in una matrice avente righe e colonne numerate, questo sistema prende il nome di *Scacchiera di Polibio*. L'impostazione base vede rappresentate le 26 lettere dell'alfabeto in questa scacchiera di 5 righe e 5 colonne, con la premura di porre due lettere (diciamo J e K) all'interno della stessa casella. Il carattere cifrato corrisponde alla coppia riga-colonna determinata dalla lettera in chiaro: oggi viene adottato un sistema analogo per far riferimento agli elementi di una matrice. L'utilizzo di questo metodo fa corrispondere al testo in chiaro "*Polibio*" il testo cifrato "*35 34 31 24 12 24 34*".

Pochi anni più tardi, a Roma, l'imperatore Cesare ideò un sistema trasposizione oggi noto come *cifrario di Cesare* facente uso di un singolo alfabeto per crittografare il testo, esso è quindi un sistema *Monoalfabetico*. Utilizzare questo cifrario con una trasposizione di 3 caratteri significa che il testo in chiaro "*Cesare*" corrisponderà al testo cifrato "*Fhzdvh*".
Intorno al 16° secolo venne introdotto un sistema *Polialfabetico* facente uso di una variante del cifrario di Cesare nel quale la trasposizione non è più rappresentata da un alfabeto "*ordinato*", bensì abbiamo ora una serie di simboli presi da molteplici alfabeti. La chiave di questo sistema consiste nel conoscere l'ordine della sequenza di tali simboli.

Oggi, gli scambi di messaggi sulla rete internet avvengono grazie ai protocolli della pila ISO/OSI ed arrivano al destinatario passando per mezzi di comunicazione non sicuri. Ciò costituisce un chiaro problema sulla riservatezza della comunicazione. Un altro problema è il riconoscimento dell'interlocutore: se esiste la possibilità di intercettare ed inviare messaggi, allora chiunque può mascherarsi da interlocutore ed inviare messaggi sotto falso nome. Si parla di algoritmi crittografici per trasformare reversibilmente un testo, rendendolo decifrabile solo a chi dispone di opportune informazioni.
Per crittografare un messaggio m mediante algoritmi di crittografia definiamo le funzioni "$E(m, K)$" per la cifratura e "$D(c, K)$" per la decifratura. Quando viene applicata $E(\cdot)$ al messaggio m si ottiene il messaggio cifrato "c" per mezzo di una chiave K. La reversibilità di tali funzioni si basa sulla stessa K (od una diversa per algoritmi più complicati descritti in seguito). In definitiva abbiamo le due relazioni

$$\begin{cases} c = E(m, K) & \text{Cifratura di } m \text{ in } c \\ m = D(c, K) & \text{Decifratura di } c \text{ in } m \end{cases}$$

Le chiavi sono sostanzialmente numeri pseudo-casuali molto grandi, la cui lunghezza si misura in bit: maggiore è la lunghezza della chiave, più saranno onerose le operazioni di cifratura e decifratura. Essendo ciascun protocollo di crittografia aperto e **noto** a tutti, la chiave assume un ruolo critico nella sicurezza: chiunque conosca quella di decifratura può decrittare e leggere il messaggio.
In definitiva classifichiamo gli algoritmi di cifratura in *Reversibili* ed *Irreversibili*. I

primi si suddividono a loro volta in due classi di algoritmi: quelli simmetrici (che necessitano di una singola chiave crittografica) e quelli asimmetrici (per i quali abbiamo bisogno di due chiavi). Gli algoritmi irreversibili vengono dati dall'applicazione di funzioni di hash.

1.1 Crittografia Simmetrica

Il messaggio in chiaro viene crittografato mediante l'algoritmo S_c e chiave simmetrica K^S, viene poi decrittato mediante l'algoritmo S_d e la **stessa chiave** K^S usata al passaggio precedente. Due utenti della rete, Alice e Bob, che vogliano usare questo sistema di crittografia dovranno agire come segue:

1. Concordare i due algoritmi S_c e S_d complementari, ossia S_d permetterà di decrittare il messaggio c dopo la cifratura mediante S_c
2. Concordare la chiave K^S
3. Alice, per mandare il suo messaggio m_A a Bob, procede nell'applicare l'algoritmo di crittografia con la chiave simmetrica, ovvero calcola $C_A = S_c(m_A, K^S)$ e lo invia sul canale non sicuro
4. Bob riceve il messaggio crittato C_A, applica l'algoritmo di decrittazione con la chiave simmetrica ed ottiene il messaggio in chiaro di Alice:

$$S_d(C_A,\, K^S) = S_d[S_c(m_A,\, K^S),\, K^S] = m_A$$

La chiave K^S, unica per entrambi gli interlocutori, determina la sicurezza dell'algoritmo e la difficoltà nel decifrarlo qualora non si disponga di tale informazione. Dimensione ridotta della chiave, velocità elevate nella cifratura e decifratura, rendono gli algoritmi simmetrici adatti a trattare messaggi molto lunghi. D'altronde non è stato chiarito **come** i due interlocutori siano giunti a concordare la medesima chiave: se per scambiare messaggi si ricorre ad una chiave crittografica, come si scambia una chiave crittografica?

1.2 Crittografia Asimmetrica

Il problema dello scambio della chiave simmetrica K^S viene risolto dalla crittografia asimmetrica. In questo ambito gli algoritmi prevedono che un utente sia in possesso di una *coppia di chiavi complementari*, una chiave è detta **pubblica** K^+ e l'altra è detta **privata** K^-, generate al momento della comunicazione. La chiave pubblica K^+ viene scambiata dai due interlocutori, mentre quella privata K^- deve essere mantenuta segreta. Il concetto di crittografia a chiave pubblica nasce nel 1976 per merito di Whitfield Diffie e Martin Hellman. Due anni dopo nacque l'algoritmo RSA[I], il più diffuso e utilizzato attualmente. Altri algoritmi asimmetrici sono il Diffie-Hellman, e il DSS (Digital Signature Standard) ma usato solo per le firme digitali.
La matematica alla base della crittografia asimmetrica determina che, data la coppia di chiavi (K^-, K^+), il testo cifrato C tramite K^+ possa essere decifrato esclusivamente dalla corrispondente K^-. La conoscenza della chiave pubblica non deve consentire il calcolo di quella privata: dati Alice e Bob, due utenti della rete che vogliono scambiarsi informazioni in modo riservato, si procede nel seguente modo:

1. Alice genera coppia di chiavi $({K_A}^-, {K_A}^+)$, Bob genera la coppia $({K_B}^-, {K_B}^+)$
2. Si procede allo scambio delle chiavi pubbliche. Alice è ora in possesso di $({K_A}^-, {K_A}^+, {K_B}^+)$ mentre Bob ha $({K_B}^-, {K_B}^+, {K_A}^+)$
3. Alice può ora mandare il suo messaggio m_A cifrandolo con la chiave **pubblica** di Bob: $C_A = {K_B}^+(m_A)$
4. Bob riceve il messaggio C_A e può decifrarlo in quanto conosce la chiave privata corrispondente:

$$ {K_B}^-(C_A) = {K_B}^-({K_B}^+(m_A)) = m_A $$

La complementarietà delle chiavi permette l'integrità dei messaggi scambiati. L'introduzione di una seconda chiave comporta un aumento della complessità computazionale e conseguenti tempi elevati di calcolo non solo per le operazioni di generazione

[I] Il nome è dato dalle iniziali dei tre ricercatori del MIT: Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman

delle chiavi ma soprattutto per quelle di cifratura e decifratura.
Gli algoritmi asimmetrici costituiscono quindi un'alternativa più sicura rispetto quelli simmetrici ma risultano essere fortemente inefficienti per trattare documenti di grossa dimensione, sono significativamente più lenti di qualsiasi algoritmo simmetrico.

1.3 Crittografia Ibrida

Si possono combinare i vantaggi dei due sistemi di crittografia visti ottenendo la "*Crittografia Ibrida*". I due interlocutori Alice e Bob agiranno come segue:

1. Alice genera la coppia di chiavi $({K_A}^-, {K_A}^+)$, Bob genera la coppia $({K_B}^-, {K_B}^+)$

2. Si procede allo scambio delle chiavi pubbliche. Alice è ora in possesso di $({K_A}^-, {K_A}^+, {K_B}^+)$ mentre Bob ha $({K_B}^-, {K_B}^+, {K_A}^+)$

3. Chi dei due voglia iniziare la comunicazione genera la chiave simmetrica K^S. Poniamo il caso che Alice sia a generarla, le chiavi in suo possesso sono quindi $({K_A}^-, {K_A}^+, K^S, {K_B}^+)$

4. Ella procede nel cifrare il suo messaggio m_A con la chiave **simmetrica** K^S ottenendo $C_A = K^S(m_A)$. Stiamo ora parlando di crittografia simmetrica, rendendo possibile lo scambio di messaggi molto grandi

5. Inoltre dovrà anche cifrare la propria chiave simmetrica con quella pubblica di Bob ottenendo $k_A = {K_B}^+(K^S)$ in modo che Bob possa decifrare il messaggio. Ciò che Alice invierà è la coppia (C_A, k_A)

6. Bob riceve la coppia (C_A, k_A) e procede nel decifrare la chiave simmetrica di Alice applicando la propria chiave privata

$$ {K_B}^-(k_A) = {K_B}^-[{K_B}^+(K^S)] = K^S $$

La correttezza di quanto calcolato è il fondamento della crittografia asimmetrica.

Egli si trova ora in possesso della chiave simmetrica di Alice con la quale può decifrare il messaggio C_A applicando il fondamento della crittografia simmetrica:

$$K^S(C_A) = K^S(K^S(m_A)) = m_A$$

L'approccio appena visto usa la crittografia simmetrica per crittografare e decrittare una chiave simmetrica, condivisa tra i due interlocutori. La chiave K^S, molto più piccola di quelle usate pubblica e privata, consente minori tempi di calcolo per le funzioni di crittografia e la sua inversa. La crittografia ibrida riesce ad unire i vantaggi dei due algoritmi visti fornendo un metodo sicuro di comunicazione.

Un importante fattore che si è tralasciato fino a questo punto è la verifica di autenticità dell'interlocutore. Tutti gli scambi di chiave effettuati sono tra Alice e Bob, due persone distanti tra loro che non possono accertarsi "di persona" sul loro interlocutore. Come può procedere Bob qualora volesse verificare che il suo interlocutore sia effettivamente Alice? Il metodo adottato per verificare l'autenticità delle parti in gioco è detto Firma Digitale e si basa sui principi della crittografia asimmetrica.

1.4 Firma Digitale

L'utente Bob richiede che Alice dimostri la sua autenticità prima di procedere allo scambio di messaggi ed ogni volta che egli lo riterrà opportuno. La firma digitale di Alice di compone dei seguenti passaggi:

1. Alice genera coppia di chiavi $({K_A}^-, {K_A}^+)$;
2. Cifra il suo messaggio m_A con la chiave propria chiave **privata** ottenendo

   $$C_A = {K_A}^-(m_A)$$

   La sua firma digitale sarà quindi la terna $F_A = (C_A, H(m_A), {K_A}^+)$
3. Invia quindi la firma digitale, sul canale non sicuro, a Bob

In questo momento tutti possono dire che se, applicando la chiave *pubblica* di Alice, vale l'uguaglianza ${K_A}^+(C_A) = m_A$ allora Alice è effettivamente chi dice di essere.
Il problema di fondo è ancora l'inefficienza degli algoritmi asimmetrici per gestire il messaggio m_A. Si considera quindi una soluzione alternativa per gestire le firme digitali: l'utilizzo delle funzioni di hash, particolari funzioni matematiche unidirezionali (*one-way*), che prendono in input il messaggio in chiaro e restituiscono un numero la cui lunghezza in bit è determinata dall'algoritmo di hash utilizzato (comunemente 128 o 160 bit). Il valore di output è detto *impronta* o **hash** del messaggio e costituisce una vera e propria impronta digitale poichè l'hash non è computazionalmente invertibile, ossia data l'impronta è molto difficile risalire al testo originale. Tuttavia, per quanto difficile sia, l'hash è soggetto a collisioni, debolezza data dal restituire un output a lunghezza fissa nonostante input di lunghezza variabile, solitamente molto più lunghi dell'output. Si parla quindi di "collisione" qualora due messaggi diversi portino al medesimo hash. Le principali funzioni di hash: SHA-1[II], SHA-256, MD5.

Il paradigma della Firma Digitale diventa, pertanto, il seguente:

1. Alice genera la coppia di chiavi $({K_A}^-, {K_A}^+)$;
2. Calcola l'hash del suo messaggio m_A tramite l'algoritmo di hash H ottenendo $C_A = H(m_A)$. Tale hash, a lunghezza fissa, ha il vantaggio di essere più corto del messaggio m_A favorendo quindi le operazioni di crittografia e decrittazione;
3. Critta tale hash mediante la sua chiave privata ${K_A}^-$ ottenendo infine

   $$s = {K_A}^-(C_A) = {K_A}^-[H(m_A)]$$

   La sua firma digitale sarà quindi la terna $F_A = (s, H(m_A), {K_A}^+)$;
4. Invia la firma digitale, sul canale non sicuro, a Bob

In questo momento, essendo vero che l'algoritmo di hash è noto a tutti, Bob è in grado di calcolare $C_A = H(m_A)$. Sulla base di questo messaggio cifrato, andiamo a decifrare s mediante la chiave pubblica di Alice:

[II] Deprecato dopo il 2011 dal NIST, **N**ational **I**nstitute of **S**tandards and **T**echnology, e dopo la confermata collisione tramite l'esperimento **SHAttered** della Google [19]

$$K_A{}^+(s) = K_A{}^+[K_A{}^-(C_A)] = C_A$$

Il confronto tra i due messaggi cifrati appena computati determina se Alice sia effettivamente chi dice di essere. Infine il messaggio in chiaro viene ora mascherato dall'hash e non più decifrabile da terzi.

Capitolo 2

Nozioni fondamentali di Algebra

Una **curva** nel linguaggio matematico è sinonimo di linea. Si parla non di curva in senso assoluto, ma di tipi di curve (come curva continua, curva piana, curva liscia), ciascuno dei quali è rigorosamente definito [1].
Lo studio di una curva si effettua su di un Campo K: insieme di elementi, non vuoto, sul quale vengono definite le operazioni binarie della somma e del prodotto e le loro operazioni inverse, sottrazione e divisione. I campi più noti sono quelli dei numeri naturali $\mathbb{N}$, dei reali $\mathbb{R}$, dei complessi $\mathbb{C}$. Campi con un numero finito di elementi sono detti "*Campi finiti*" o anche "*Campi di Galois*", vengono rappresentati dalle notazioni, equivalenti tra loro: $\mathbb{F}_{n^m} = \mathbb{Z}/n^m\mathbb{Z} = \{0, 1, \ldots, n-1\}$; con n un numero primo, m un numero naturale maggiore o uguale ad 1. I campi della forma $\mathbb{F}_{2^m}$ vengono detti "*campi binari*", quelli invece definiti come $\mathbb{F}_n$ vengono detti "*campi primi*". Il numero di elementi del campo è pari ad n, detto **ordine**.
Ciascun campo ha una **caratteristica** definita secondo la seguente formula:

$$n \in \mathbb{N},\ n \neq 0,\ char(\mathbb{F}_{n^m}) = \begin{cases} \min(n) \mid \underbrace{1+1+\cdots+1}_{\text{n volte}} = 0 & \text{se } \exists n \\ 0 & \text{altrimenti} \end{cases} \tag{2.1}$$

la caratteristica è pari al minimo numero di volte che l'elemento identità della somma (**1**) deve essere sommato a se stesso per ottenere l'elemento identità della motiplicazione (**0**). Se non fosse possibile definire tale n si assume la caratteristica pari a *zero*.
Per un campo finito, l'ordine e la caratteristica coincidono. Si può dimostrare che

il numero n debba essere *primo* procedendo per assurdo: se esso non fosse primo permetterebbe una scomposizione del tipo $n = n_1 \cdot n_2$, dove $n_1 > 1$ ed $n_2 > 1$ allora:

$$0 = \underbrace{1 + 1 + \cdots + 1}_{\text{n}} = \underbrace{(1 + 1 + \cdots + 1)}_{n_1} \cdot \underbrace{(1 + 1 + \cdots + 1)}_{n_2}$$

Dato che il termine centrale è pari a 0 per definizione di caratteristica, almeno uno dei due termini a destra deve corrispondere a 0 ma ciò è assurdo in quanto n è già il numero minimo per soddisfare tale relazione. I due fattori scelti n_1 ed n_2 verificherebbero l'uguaglianza solo nei casi $n_1 = 1$, $n_2 = n$ o viceversa $n_1 = n$, $n_2 = 1$ ma il valore "1" è stato escluso nell'ipotesi iniziale. Non esistendo quindi tali due numeri n_1 ed n_2 tali che il loro prodotto sia n si deduce la primalità di quest'ultimo.
I campi $\mathbb{Q}$, $\mathbb{R}$ e $\mathbb{C}$ hanno caratteristica 0.

2.1 Gruppi

Dati un insieme G ed una funzione "$\bullet$" definita come operazione binaria per la quale valga la proprietà della *chiusura*

$$(a, b) \to a \bullet b : G \times G \to G$$

allora la coppia $(G, \bullet)$ prende il nome di **gruppo** qualora vengano rispettati gli *assiomi di gruppo*:

1. *Associatività*: $(a \bullet b) \bullet c = a \bullet (b \bullet c)$, $\forall (a, b, c) \in G$;
2. *Esistenza dell'identità*: $\exists! e \in G \mid e \bullet a = a \bullet e = a$;
3. *Esistenza dell'inverso*: $\exists b \in G \mid a \bullet b = b \bullet a = e$.

La funzione "$\bullet$" definita prende il nome di **Legge del Gruppo G** ma viene solitamente omessa nel far riferimento ad un gruppo. Il tipo di Legge influenza come gli elementi interni vengono combinati tra loro: un gruppo è *additivo* se la Legge è definita come "somma" e l'elemento identità è lo 0, alternativamente un gruppo è detto *moltiplicativo* se tale Legge è una "moltiplicazione" e l'elemento identità è 1.
Agli assiomi di gruppo possiamo aggiungere la proprietà commutativa: se è vero che

$$a \bullet b = b \bullet a,\ \forall a, b \in G$$

allora il gruppo G è detto *Gruppo Abeliano*.

Parliamo di **sottogruppo** S quando ci riferiamo ad un **sottoinsieme non vuoto** del gruppo G per il quale valgano le condizioni

$$S \subset G \iff \begin{cases} a \bullet b \in S & \forall a, b \in S \\ a^{-1} \in S & \forall a \in S \end{cases}$$

dove con il termine "a^{-1}" si intende l'elemento inverso di a [2].
Parliamo di **ordine** di un gruppo quando facciamo riferimento alla sua cardinalità (il suo numero di elementi), le notazioni usate sono $\mid G \mid$ oppure $\#G$. I sottogruppi di G con ordine $\mid S \mid < \mid G \mid$ sono detti "sottogruppi propri".
Una proprietà molto importante per i sottogruppi discende dal *Teorema di Lagrange* il quale afferma che "la cardinalità di un sottogruppo H del gruppo G è un **divisore intero** della cardinalità di G". In termini meno formali, se il gruppo G ha cardinalità N ed uno dei suoi sottogruppi ha cardinalità n allora il rapporto $m = N/n$ è un numero intero $m \in \mathbb{Z}$ detto **cofattore**.

Un gruppo G viene definito **ciclico** qualora sia generato a partire da un singolo elemento $g \in G$ per il quale si rispetti la relazione

$$\begin{aligned} &G = \{0, g, 2g, \ldots, (n-1)g\} && \text{con } ng = 0 && \text{notazione } \textit{Additiva} \\ &G = \{1, g, g^2, \ldots, g^{n-1}\} && \text{con } g^n = 1 && \text{notazione } \textit{Moltiplicativa} \end{aligned}$$

ovvero ogni altro elemento del gruppo può essere ottenuto applicando ripetitivamente la Legge di Gruppo o la sua inversa all'elemento g.
Il punto g viene quindi detto **generatore** del gruppo G, ha un **ordine** indicato con la notazione $\langle g \rangle$ e corrisponde al più piccolo numero m tale che $g^m = 1$ (in notazione Moltiplicativa) oppure tale che $mg = 0$ (in notazione Additiva) se tale m esiste, altrimenti è infinito (a tal proposito vedasi la formula (2.2) seguente):

$$m \in \mathbb{N},\ m \neq 0,\ \langle g \rangle = \begin{cases} \min(m) \mid mg = 0 & \text{se } \exists m, \text{ notazione } \textit{Additiva} \\ \min(m) \mid g^m = 1 & \text{se } \exists m, \text{ notazione } \textit{Moltiplicativa} \\ \infty & \text{altrimenti} \end{cases} \tag{2.2}$$

Per gruppi ciclici l'ordine del generatore g e l'ordine del suo gruppo G coincidono [5].
Il campo dei numeri interi $\mathbb{Z}$ forma un Gruppo con legge "+", elemento identità 0, e proprietà commutativa. Tale campo è quindi un *Gruppo Additivo Abeliano*. È facile dimostrare che tale campo non possa essere un Gruppo Moltiplicativo poichè dato un qualsiasi elemento $z \in \mathbb{Z}$ il suo inverso sarà un numero razionale $q = 1/z$, $q \in \mathbb{Q}$; al più abbiamo $q = 1$ se scegliamo $z = 1$ ma ciò non soddisferebbe tutti gli altri punti del campo.

Definiamo ora la tripla $(R, +, \cdot)$ con il termine "anello" [4]: la sua coppia $(R, +)$ rappresenta un gruppo additivo abeliano, mentre l'operazione "$\cdot$" segue le proprietà associativa, distributiva e commutativa, il suo elemento neutro $e = 1$ viene definito come segue

$$e \cdot a = a \cdot e = a, \, \forall a \in R.$$

La caratteristica un anello R è il più piccolo n tale che valga la (2.1) per ogni suo elemento, può esser quindi definita come il *minimo comune multiplo* delle caratteristiche di tutti i suoi elementi. Se tale minimo n non esiste allora si assume che la caratteristica sia "0" per definizione [5]. Tutti gli anelli con caratteristica $char(R) = 0$ sono *infiniti*. L'unico anello con caratteristica $char(R) = 1$ è l'anello banale, costituito da un solo elemento. Tutti gli anelli con caratteristica maggiore di 1 sono costituiti da un numero finito di elementi. Suddivisioni dei numeri interi del tipo "$\mathbb{Z}/n\mathbb{Z}$" forniscono anelli modulari con elementi in numero finito e caratteristica $char(R) = n$.
Esempio di un sottogruppo generato da un anello modulare: detto n un numero naturale diverso da zero, tutti i sottogruppi della forma $\mathbb{Z}/n\mathbb{Z}$ sono di ordine finito n e contengono gli elementi

$$H = \mathbb{Z}/n\mathbb{Z} = \{n\mathbb{Z}, \, 1 + n\mathbb{Z}, \, \ldots, \, (n-1) + n\mathbb{Z}\}$$

preso, ad esempio $n = 5$, il gruppo $H = \mathbb{Z}/5\mathbb{Z}$ comprenderà, ciclicamente, gli elementi $H = \{0, 1, 2, 3, 4\}$.

2.2 Spazio Proiettivo

"Definito con V uno spazio vettoriale, $\mathbb{P}$ è lo *spazio proiettivo* costituito dai vettori monodimensionali dei sottospazi vettoriali di V. Se lo spazio vettoriale V ha dimensione $n+1$, lo spazio proiettivo da esso generato avrà dimensione n" [6].
Possiamo pensare ad uno spazio proiettivo come ad un'estensione dello spazio Euclideo mediante l'aggiunta dei *punti all'infinito*. A seconda dello spazio euclideo considerato, ad esempio una retta o un piano, si vengono a definire, nello spazio proiettivo, una retta proiettiva o un piano proiettivo.
Estendere il piano Euclideo [8] a quello proiettivo significa:

1. per ogni classe di rette parallele, aggiungere un singolo nuovo punto. Tale punto è considerato come punto di incontro per ogni retta della stessa classe. Classi di diverse rette parallele avranno diversi punti. Questi punti sono chiamati *punti all'infinito*;

2. aggiungere una nuova retta incidente su tutti e soli i punti all'infinito. Questa retta è chiamata **la** *retta all'infinito*.

Ne consegue l'esclusione dell'esistenza di rette proiettive parallele.
Dato il punto $E = (x, y)$ nel piano Euclideo, la scrittura $E = [xZ : yZ : Z]$, con $Z \in \mathbb{R}\backslash\{0\}$ un numero reale diverso da zero è data in termini di *coordinate omogenee* o "*coordinate proiettive*".
Riguardo la notazione: per distinguere le coordinate proiettive da quelle cartesiane, spesso vengono usate notazioni differenti da (x, y). A volte viene sostituita la virgola "**,**" con il simbolo "**:**", altre volte le parentesi tonde "**()**" vengono sostituite da quadre "[]" ed altre volte vengono usate entrambe le notazioni assieme; infine si è soliti preferire le lettere maiuscole per punti proiettivi. Esempio: coordinate cartesiane $(x, y) = (1, 4)$, omogenee $[X : Y : Z] = [1 : 4 : Z]$. Per mantenere un evidente cambio di coordinate verrà usata quest'ultima notazione.

Diversamente da quanto accade per lo spazio Euclideo, due terne di coordinate omogenee rappresentano lo stesso punto se e solo se questo è ottenuto moltiplicando le coordinate per una costante diversa da 0. Ciò significa che, preso il punto sul piano cartesiano $(1, 2)$, questo può esser rappresentato dai punti omogenei $[1 : 2 : 1]$,

$[2:4:2]$ o più in generale $[Z:2Z:Z]$.
Dal punto omogeneo $[X:Y:Z]$ otteniamo il punto cartesiano $(X/Z,\ Y/Z)$. Un caso degenere è per $Z=0$ per cui il punto rappresentato nello spazio proiettivo è il punto all'infinito $\mathcal{O}=[0:1:0]$. L'insieme di tutti i punti $[X:Y:0]$, ovvero di tutti i punti all'infinito, è la retta all'infinito espressa con la notazione $[X:1:0]$ dove X appartiene allo spazio proiettivo considerato.

Definito con K un generico campo avente caratteristica diversa da 2, possiamo rappresentare il generico polinomio di grado d, nelle incognite (x,y) e coefficienti $a_i \in K$ come:

$$y = f(x) = a_d x^d + a_{d-1}x^{d-1} + \cdots + a_i x^i + \cdots + a_0$$

ed in coordinate proiettive [10]:

$$Y = F(X,Z) = a_d X^d + a_{d-1}X^{d-1}Z + \cdots + a_i X^i Z^{d-i} + \cdots + a_0 Z^d$$

Polinomi di gradi diversi generano curve diverse: si parla di retta per un polinomio di grado 1, di conica per un polinomio di grado 2, di cubica se il grado è 3. Una proprietà delle curve piane in $\mathbb{P}^2$ è il "genere", concetto direttamente legato al grado d di una curva tramite la formula "*genere-grado*" secondo la quale possiamo definire il genere g di una curva come $g=\frac{1}{2}(d-1)(d-2)$.
La curva cubica $y^2=x^3-x$ ha associata l'equazione polinomiale omogenea cubica $Y^2Z=X^3-XZ^2$ che definisce una curva proiettiva $C\in\mathbb{P}^2$ come *Curva Ellittica*. Tale curva presenta grado $d=3$ e quindi genere $g=1$.
Infine, detta $C=f(X,Y)$ una curva in un campo K, definiamo un suo punto $P(X,Y)$ come **K-razionale** se entrambe le sue coordinate sono razionali ed appartengono al campo K

$$X,Y = \textit{K-Razionali} \iff X,Y \in \mathbb{Q}\cap K$$

Un particolare punto K-razionale è il punto all'infinito $\mathcal{O}$.

Capitolo 3

Curve Ellittiche

Una curva ellittica su di un campo K è una curva cubica, di grado 3 e genere 1, in due variabili, avente un punto K-razionale. Il punto K-razionale può essere il punto all'infinito $\mathcal{O}$ ed il campo K è solitamente il campo dei numeri reali $\mathbb{C}$, dei reali $\mathbb{R}$, dei razionali $\mathbb{Q}$ o un campo finito. Se la caratteristica $char(K)$ del campo K è diversa da 2 e da 3 possiamo scrivere la formula nella forma proiettiva estesa di Tate-Weierstrass [14]:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

impostando $Z = 1$ otteniamo la forma estesa:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \tag{3.1}$$

Tramite due cambi di variabili otteniamo la forma breve della curva: lavoriamo dapprima sul membro di sinistra ed applichiamo la trasformazione $Y = y - \frac{a_1X + a_3}{2}$

$$\begin{aligned}
&Y^2 + a_1XY + a_3Y \\
&= \left(y - \frac{a_1}{2}X - \frac{a_3}{2}\right)^2 + a_1X\left(y - \frac{a_1}{2}X - \frac{a_3}{2}\right) + a_3\left(y - \frac{a_1}{2}X - \frac{a_3}{2}\right) \\
&= y^2 + \frac{a_1^2}{2}X^2 - \frac{a_1a_3}{2}X + \frac{a_3^2}{2}
\end{aligned}$$

Portiamo quindi i termini in X al membro destro dell'equazione (3.1) e semplifichia-

mo i termini simili

$$\begin{aligned} y^2 &= -\left(\frac{{a_1}^2}{2}X^2 - \frac{a_1 a_3}{2}X + \frac{{a_3}^2}{2}\right) + X^3 + a_2X^2 + a_4X + a_6 \\ &= X^3 + AX^2 + BX + C \end{aligned}$$

dove abbiamo $A = a_2 - \frac{{a_1}^2}{2}$, $B = a_4 + \frac{a_1 a_3}{2}$ e $C = a_6 - \frac{{a_3}^2}{2}$. Applichiamo quindi il secondo cambio di variabile $X = x - \frac{A}{3}$, il secondo membro diventa:

$$\begin{aligned} & X^3 + AX^2 + BX + C \\ &= \left(x - \frac{A}{3}\right)^3 + A\left(x - \frac{A}{3}\right)^2 + B\left(x - \frac{A}{3}\right)^3 + C \\ &= x^3 - Ax^2 + \frac{A^2}{3}x - \frac{A^3}{27} + Ax^2 + -\frac{A^3}{9} - \frac{2}{3}A^2x + Bx - \frac{AB}{3} + C \\ &= x^3 + (B - A^2)x + \frac{2A^3 - 9AB + 27C}{27} \\ &= x^3 + ax + b \end{aligned}$$

Abbiamo quindi ottenuto la forma breve

$$y^2 = x^3 + ax + b \tag{3.2}$$

Tuttavia per renderla una vera curva ellittica dobbiamo imporre che sia liscia, ovvero non singolare, per cui non devono esistere radici multiple. Possiamo aggiungere questa condizione dicendo che il determinante dell'espressione $x^3 + ax + b$ deve essere diverso da zero, $\Delta = -4a^3 - 27b^2 \neq 0$. L'aggiunta di questa condizione ci porta alla

$$\textit{Equazione di Weierstrass}: \begin{cases} y^2 = x^3 + ax + b \\ 4a^3 \neq 27b^2 \end{cases} \bigcup \{\mathcal{O}\} \tag{3.3}$$

in unione al punto all'infinito $\mathcal{O}$.

Dimostrazione della condizione $4a^3 \neq 27b^2$: La condizione da imporre per una curva liscia è la derivabilità in ogni punto della curva, va quindi rispettata la relazione

$\frac{d}{dx}C(x) \neq 0 \ \forall x \in C$ dove C è l'equazione della nostra curva. Partiamo dall'equazione (3.2) della curva:

$$y^2 = x^3 + ax + b \qquad \text{Applichiamo la derivata in } x$$
$$\frac{d}{dx}y^2 = \frac{d}{dx}(x^3 + ax + b)$$
$$0 = 3x^2 + a \qquad \text{(Eq. derivata)}$$
$$x^2 = -\frac{a}{3}$$

Riprendiamo l'equazione iniziale

$$y^2 = x^3 + ax + b \qquad \text{Consideriamo } y = 0$$
$$x^3 + ax + b = 0 \qquad \text{Moltiplichiamo per } x$$
$$x^4 + ax^2 + bx = 0 \qquad \text{Applichiamo la } x^2 \text{ trovata prima}$$
$$\left(-\frac{a}{3}\right)^2 + a\left(-\frac{a}{3}\right) + bx = 0 \qquad \text{Semplifichiamo}$$
$$\left(\frac{a^2}{9}\right) - \frac{a^2}{3} + bx = 0 \qquad \text{Troviamo quindi la } x$$
$$x = \frac{2a^2}{9b}$$

Torniamo alla (Eq. derivata)

$$0 = 3x^2 + a \qquad \text{ed applichiamovi la } x \text{ trovata}$$
$$0 = 3\left(\frac{2a^2}{9b}\right)^2 + a \qquad \text{Svolgiamo il quadrato}$$
$$0 = \frac{4a^4}{27b^2} + a \qquad \text{Dividiamo per } a$$
$$0 = \frac{4a^3}{27b^2} + 1 \qquad \text{Otteniamo dunque}$$
$$4a^3 + 27b^2 = 0 \qquad \text{c.v.d.}$$

La derivata si annulla per $4a^3 + 27b^2 = 0$ dimostrando la tesi iniziale.

È possibile affermare che le curve ellittiche formano un Gruppo date le seguenti affermazioni:

1. L'operazione binaria "+" lega due punti della curva ad un terzo ancora appartenente alla curva. Definita in tal modo, essa gode della proprietà della "*chiusura*" e risulta valida per esser definita Legge di Gruppo. La formula che esprime questo concetto è la seguente:

$$A + B = -C$$

dove A, B e C sono tre punti della curva

2. L'elemento identità è il punto all'infinito $\mathcal{O} = \infty$ o $\mathcal{O} = [0 : 1 : 0]$ in coordinate proiettive. Tale punto appartiene alla curva e lo si può dimostrare [11] per ogni curva ellittica:

$$\begin{aligned} y^2 &= x^3 + ax + b && \text{Rendiamola in forma proiettiva} \\ Y^2Z &= X^3 + aXZ^2 + bZ^3 && \text{Calcoliamo } \mathcal{O} \text{ imponendo } Z = 0 \\ 0 &= X^3 && \text{Da cui } X = 0\text{, calcoliamo la } Y \\ Y^2Z &= bZ^3 \end{aligned}$$

restando vero che $Z = 0$, la Y può assumere qualsiasi valore ma, trovandoci in uno spazio proiettivo tutte le terne $[0, Y_i, 0]$ sono equivalenti in quanto multipli scalari secondo Y_i. Si dimostra quindi che il punto trovato si comporta come elemento identità, si nota che il punto $\mathcal{O} = [0 : 1 : 0]$ da noi cercato ha molteplicità 3 (infatti abbiamo $X^3 = 0$) ed appartiene a qualsiasi curva ellittica

3. L'elemento inverso del punto P è "$-P$", il suo simmetrico rispetto all'asse x, ancora appartenente alla curva. La dimostrazione di quest'affermazione è riportata come Caso Particolare più avanti nella trattazione della legge di gruppo

4. Sapendo che è possibile applicare la legge di gruppo a due punti qualsiasi della curva e confermata l'esistenza dell'elemento inverso, è possibile rispettare l'assioma dell'associatività. Detti A e B due punti della curva tali che la loro somma tramite Legge di Gruppo sia il punto -C, possiamo dire che vale:

$$(A + B) + C = (A + C) + B$$

L'associatività viene dunque dimostrata considerando valido il risultato ottenuto per ogni terna di punti della curva.

Avendo rispettato gli assiomi di gruppo, abbiamo provato che le curve ellittiche costituiscano effettivamente un Gruppo.
Infine, generalizzando l'associatività del gruppo, possiamo concludere che la seguente formula

$$A + (B + C) = B + (A + C) = C + (A + B) = C + (B + A) = \cdots = \mathcal{O}$$

sia sempre valida per ogni curva ellittica. Quello così definito corrisponde ad un *Gruppo Additivo Abeliano*.

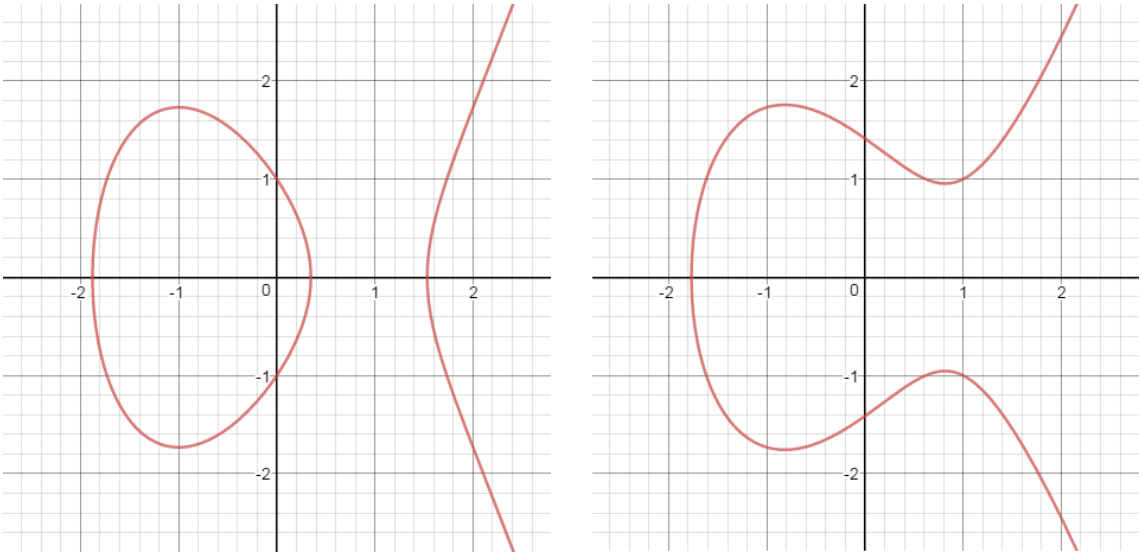


Figura 3.1: Tipiche rappresentazioni di curve ellittiche

La figura (3.1) mostra la curva $y^2 = x^3 + ax + b$ in due momenti diversi:
il grafico di *sinistra* mostra la curva **divisa** in due parti, con $a = -3$, $b = 1$;
il grafico di *destra* mostra la curva **intera**, data dai parametri $a = -2$, $b = 2$.

3.1 La Legge di Gruppo

Usiamo il termine "*Point Addition*" per far riferimento alla somma "+" di due punti sulla curva, questa operazione costituisce la Legge di Gruppo per le curve ellittiche. Definiamo rigorosamente la Point Addition come "*Operazione binaria tra due punti A e B, K-razionali, di una curva ellittica E tale per cui $A + B = -C$. Il risultato $-C$ è anch'esso un punto K-razionale appartenente alla curva E*".
In alcuni casi, si fa riferimento alla Point Addition non come operatore binario ma come ternario "$A + B + C = \mathcal{O}$" evidenziando il legame dei tre punti con l'elemento identità $\mathcal{O}$.

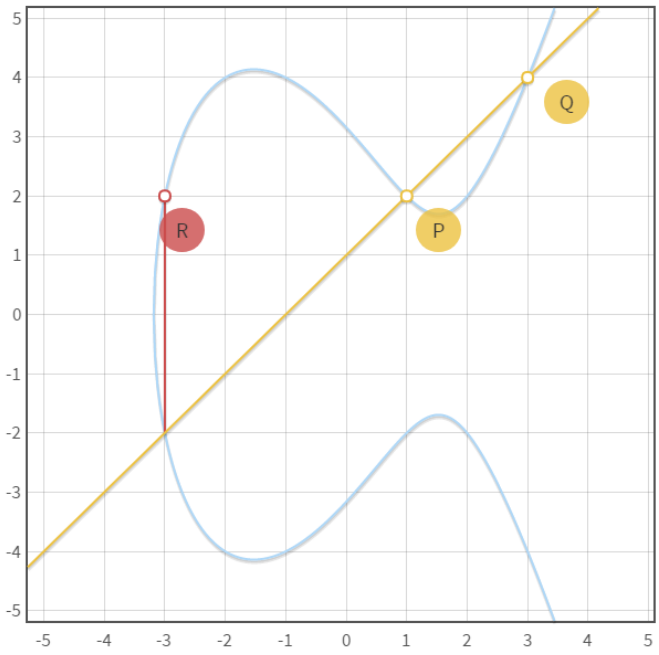


Figura 3.2: Point Addition sulla curva $y^2 = x^3 - 7x + 10$

$$P = (1, 2),\ Q = (3, 4),\ R = -(P + Q) = (-3, 2)$$

Detta (x_C, y_C) la coppia di coordinate che rappresenta il punto C, intersezione della retta passante per $A = (x_A, y_A)$ e $B = (x_B, y_B)$ con la curva E, il suo simmetrico $-C = (x_C, -y_C)$ costituisce il risultato della legge di gruppo. Per semplice

costruzione geometrica, la retta r_{AB} passante per i due punti A e B presenterà un coefficiente angolare m ed una quota q. Le formule per determinare direttamente la coppia (x_C, y_C) sono:

$$\begin{cases} m = \frac{y_A - y_B}{x_A - x_B} \\ x_C = m^2 - (x_A + x_B) \\ y_C = m(x_A - x_C) - y_A \end{cases}$$

Dimostrazione: definiamo la retta come $y = mx + q$ e la curva ellittica come $y^2 = x^3 + ax + b$. L'intersezione della retta con la curva dà luogo ai tre punti A, B, C per cui possiamo scrivere

$$\begin{cases} y = mx + q \\ y^2 = x^3 + ax + b \end{cases} \iff (mx + q)^2 = x^3 + ax + b$$

Portiamo tutto nello stesso membro dell'equazione ed uguagliamo il risultato a zero. Svolgendo i conti otteniamo un'equazione di terzo grado nell'incognita x:

$$\begin{aligned} x^3 + ax + b - (mx + q)^2 &= 0 \\ x^3 + ax + b - (m^2x^2 + q^2 + 2mqx) &= 0 \\ x^3 - m^2x^2 + (a - 2mq)x + (b - q^2) &= 0 \end{aligned}$$

Eguagliamo infine tale equazione al prodotto delle radici x_A, x_B ed x_C:

$$\begin{aligned} 0 &= (x - x_A)(x - x_B)(x - x_C) \\ x^3 - m^2x^2 + (a - 2mq)x + (b - q^2) &= (x - x_A)(x - x_B)(x - x_C) \end{aligned}$$

Svolgendo il termine destro ed uguagliando i coefficienti legati alle incognite x dello stesso grado otteniamo le uguaglianze:

$$\begin{cases} x^3 = x^3 \\ -m^2\boldsymbol{x^2} = -(x_A + x_B + x_C)\boldsymbol{x^2} \\ (a - 2mq)x = (x_Ax_B + x_Ax_C + x_Bx_C)x \\ b - q^2 = x_Ax_Bx_C \end{cases}$$

Dall'uguaglianza in $\boldsymbol{x^2}$ troviamo

$$x_C = m^2 - x_A - x_B = m^2 - (x_A + x_B)$$

come anticipato inizialmente. Possiamo ora calcolare l'ordinata y_C a partire dall'equazione della retta e facendo riferimento ad uno dei due punti A o B per determinare la quota della retta:

$$\begin{cases} y_C = m(x_A - x_C) - y_A & \text{Punto di riferimento: } A \\ y_C = m(x_B - x_C) - y_B & \text{Punto di riferimento: } B \end{cases}$$

Abbiamo quindi verificato che le formule date sono valide.

Mediante queste formule è possibile calcolare anche la differenza tra due punti: se la somma viene espressa come $A + B = C$ allora possiamo esprimere la sottrazione come $B = C - A$. Si è già detto che alla notazione "$-A$" facciamo corrispondere il punto simmetrico di A, quindi la sottrazione può esser considerata un caso particolare della Point Addition in cui $B = C - A = C + (-A)$.

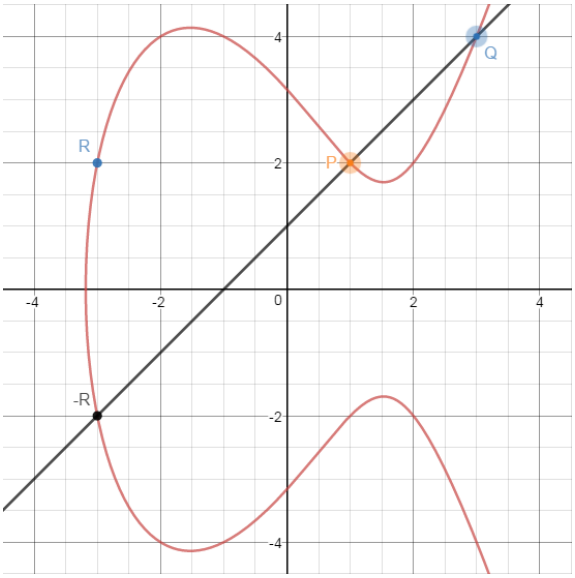


Figura 3.3: P+Q=R

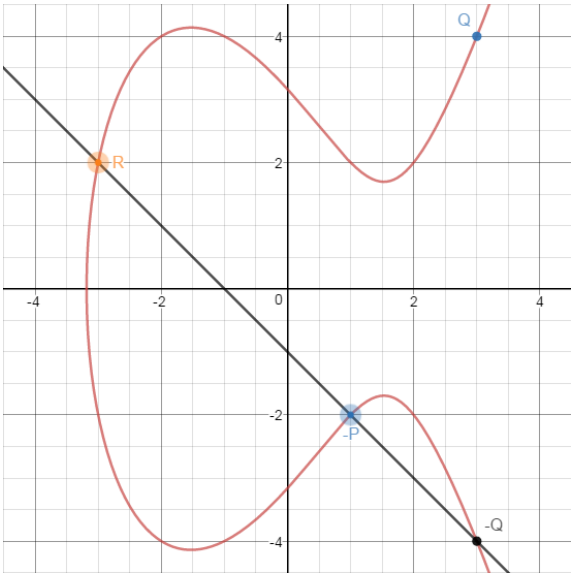


Figura 3.4: R+(-P)=Q

Illustrate le formule che governano la legge di gruppo, è bene discutere riguardo i casi particolari di questa operazione:

1. Somma un punto P ed il suo simmetrico $Q = -P$.
   Tramite il metodo della Point Addition si costruisce una retta per i due punti che risulta parallela all'asse y e che andrà ad intersecare la curva nel punto all'infinito $\mathcal{O}$. Definiamo quindi $P + Q = P + (-P) = \mathcal{O}$ (fig. 3.5)

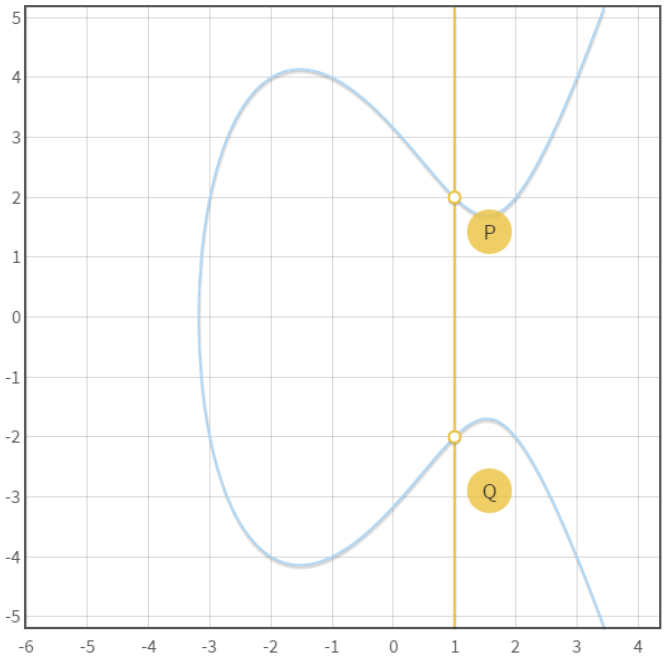


Figura 3.5: Curva $y^2 = x^3 - 7x + 10$, $P = (1, 2)$, $Q = -P = (1, -2)$

2. Somma di un punto P ed il punto all'infinito $\mathcal{O}$.
   La retta che unisce il punto P ed il punto $\mathcal{O}$ interseca la curva in un punto $Q = -P$. Simmetrizzando e calcolando il punto $-Q$ otteniamo il risultato "$-(-Q) = P$". Ricordando che il punto $\mathcal{O}$ viene trattato come elemento identità del gruppo possiamo infine scrivere $P + \mathcal{O} = \mathcal{O} + P = P$. Si consideri, a tal proposito, la stessa figura (3.5) del caso precedente

3. Somma di un punto P e se stesso.
   Non avendo due punti distinti non è possibile tracciare la retta per i due punti. Fissato il punto P, immaginiamo di prendere un secondo punto ed avvicinarlo

a P: la retta tra i due punti diventa tangente alla curva nel punto P. La tangente intercetta quindi il punto $-R$ sulla curva, il punto simmetrico è il risultato cercato: $P + P = -(-R) = R$,

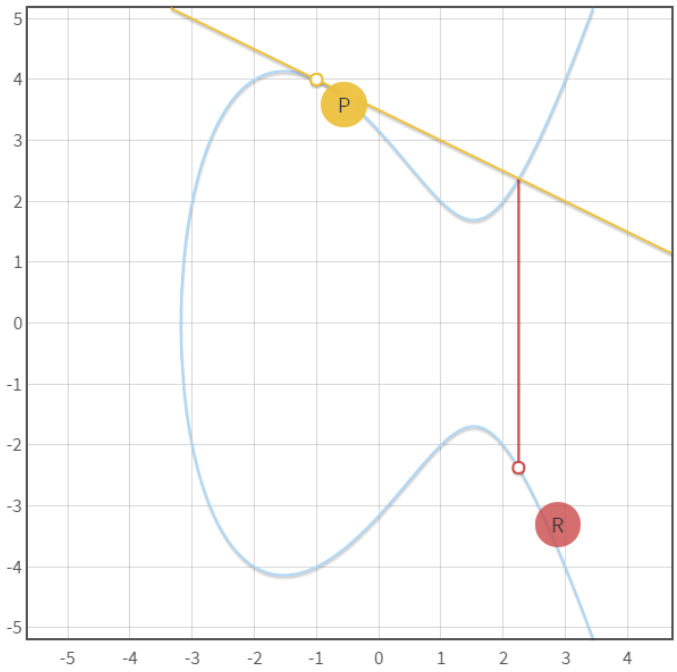


Figura 3.6: Curva $y^2 = x^3 - 7x + 10$, $P = Q = (-1, 4)$

Definiamo questo caso con la dicitura "*Point Doubling*". Le equazioni viste per la legge di gruppo tuttavia non permettono di calcolare il punto $2P$: bisogna trovare delle formule che tengano conto della **tangente** alla curva E nel punto P. La retta tangente $y = mx + q$ presenterà un coefficiente angolare m dato

dal gradiente dell'equazione (3.2):

$$
\begin{aligned}
E : y^2 &= x^3 + ax + b && \text{Dividiamo tutto per } y^2 \\
1 &= \frac{x^3 + ax + b}{y^2} && \text{Applichiamo il gradiente} \\
m &= \frac{dy}{dx}E && \text{Imponiamo l'equazione di } E \\
m &= \frac{dy(x^3 + ax + b)}{dx(y^2)} && \text{Che possiamo vedere come} \\
m &= \frac{\dfrac{d}{dx}(x^3 + ax + b)}{\dfrac{d}{dy}(y^2)} && \text{Otteniamo dunque} \\
m &= \frac{3x^2 + a}{2y}
\end{aligned}
$$

Si noti come il coefficiente angolare m dipenda non solo dalle coordinate del punto P ma anche dal coefficiente "a" presente nell'equazione della curva. Definiti quindi i punti $A = (x_A, y_A)$ e $C = (x_C, y_C) = -2A$ della curva, le operazioni per effettuare una Point Doubling sono:

$$
\begin{cases}
m = \dfrac{3{x_A}^2 + a}{2y_A} \\
x_C = m^2 - 2x_A \\
y_C = m(x_A - x_C) - y_A
\end{cases}
\tag{3.4}
$$

Le coordinate di C sono calcolate mediante formule analoghe alla Point Addition: si considera l'intersezione della tangente con la curva E, si eguaglia il risultato al prodotto delle radici x_A e x_C ricordando che la prima ha molteplicità 2. Infine si considerano i monomi dello stesso grado, esattamente come fatto per la Legge di Gruppo, trovando le formule qui sopra descritte

4. Somma di $P + P$ quando P è un flesso per la curva.
   In un punto di flesso la concavità della curva cambia , la tangente nel punto P **non** interseca nuovamente la curva, di conseguenza non intercetterà neanche il punto all'infinito. Il punto P ha un **ordine** n (formula 2.2), per il quale vale l'uguaglianza $nP = \mathcal{O}$. Un punto che presenti un ordine **finito** viene detto

"*Punto di Torsione*" [15]. Essendo un punto di flesso, P assume ordine 3 ed è quindi un punto di Torsione.
Imponiamo $n = 3$ per il nostro punto, la relazione $P + P + P = \mathcal{O}$ è la diretta conseguenza dell'ordine di un punto di Torsione e ci permette di scrivere:

$$P + P = -P \iff P = flesso$$

3.2 Moltiplicazione Scalare

Calcolare nP, con n un numero intero, su una curva ellittica costituisce l'operazione detta "*moltiplicazione scalare*" o Point Multiplication, operazione molto comune nelle applicazioni crittografiche. Allo scopo, supponiamo di voler calcolare il punto $100P$ ed illustriamo alcune tecniche utilizzate in pratica per ottenere un risultato nel minor tempo possibile.
La tecnica più intuitiva consiste nell'applicare ripetutamente la Legge di Gruppo per computare i successivi $n - 1$ punti ma la scelta diventa impraticabile qualora n sia un numero grande (possiamo definire tale anche un $n = 100$ per motivi che verranno mostrati nei prossimi algoritmi).

3.2.1 Double & Add

Per introdurre questo algoritmo, ragioniamo inizialmente all'interno dei numeri naturali $\mathbb{N}$. Il nostro scopo è partire dal numero $m_0 = 1$ ed arrivare ad $m_i = 100$ tramite i passi e le sole operazioni di somma unitaria (detta "*Add*") e moltiplicazione con fattore 2 (detta "*Double*"):

Passo i	m_{i-1}	Operazione	m_i
1	1	Double	2
2	2	Add	3
3	3	Double	6
4	6	Double	12
5	12	Double	24
6	24	Add	25
7	25	Double	50
8	50	Double	100

Abbiamo raggiunto il nostro scopo in un totale di 8 passi, un risultato nettamente migliore degli $n - 1 = 99$ passi visti precedentemente. Applicare questo algoritmo su di una curva ellittica significa nel computare una Point Addition per ogni *Add* ed una Point Doubling per ogni *Double*.
Parliamo allora di una possibile implementazione del metodo appena visto. Consideriamo il nostro $n = 100$ ed esprimiamolo in notazione binaria:

$$n_{10} = 100_{10} \iff 1100100_2 = d_6d_5d_4d_3d_2d_1d_0 = n_2$$

Il procedimento da utilizzare consiste nel calcolare n_2 tramite l'operazione del Point Doubling per ogni d_i, procedendo dalla seconda cifra più significativa (in questo caso d_5) fino a d_0. Nel caso in cui troviamo "$d_i = 1$" facciamo seguire una Point Addition.

d_i	Operazione	n_2	Commenti
$d_6 = 1$	/	1	Il primo d_i sarà sempre 1;
$\boldsymbol{d_5 = 1}$	Double	10	Avendo $d_5 = 1$ bisogna usare una ADD;
	Add	1**1**	d_5 è sistemata;
$d_4 = 0$	Double	110	
$d_3 = 0$	Double	1100	
$\boldsymbol{d_2 = 1}$	Double	11000	Come prima: segue una ADD;
	Add	1100**1**	ed anche d_2 è sistemata
$d_1 = 0$	Double	110010	
$d_0 = 0$	Double	1100100	

Questo metodo, detto *Double&Add*, permette il calcolo di nP in $\lfloor log_2(n) \rfloor + \theta$ operazioni. Le point doubling sono $\lfloor log_2(n) \rfloor$ mentre le point addition sono θ, un numero

corrispondente al totale di "1" nella notazione binaria, per cui $\theta \leq \lfloor log_2(n) \rfloor$. Il costo computazionale è nettamente ridotto rispetto alle $n-1$ operazioni del primo metodo.

3.2.2 Montgomery Ladder

Esiste un metodo simile alla Double&Add, e viene detto **Montgomery Ladder**. Detti P il punto da moltiplicare ed n il moltiplicatore la cui rappresentazione binaria n_2 è ancora $d_j d_{j-1} \dots d_2 d_1 d_0$, definiamo due parametri come $P_1 = P$ e $P_2 = 2P$. La Montgomery Ladder procede come segue:
da $i = j - 1$, fino a $i = 0$

$$\text{se } d_i = 1 \rightarrow \begin{cases} P_1 & = P_1 + P_2 \\ P_2 & = 2P_2 \end{cases}$$

$$\text{altrimenti } \rightarrow \begin{cases} P_1 & = 2P_1 \\ P_2 & = P_1 + P_2 \end{cases}$$

decrementa i di 1

A fine ciclo, il punto cercato nP è il valore di P_1.
Una differenza con l'algoritmo precedente è il calcolo di entrambe le operazioni, una Double ed una Add, per ogni cifra d_j. La moltiplicazione scalare mediante la Montgomery Ladder richiede dunque $2\lfloor log_2(n) \rfloor$ operazioni. Il numero trovato corrisponde alle operazioni necessarie in una Double&Add qualora $n = 2^x - 1$, $x \in \mathbb{N}$, la cui notazione binaria è data soli "1".
Una riduzione del tempo computazionale è possibile mediante la parallelizzazione della Montgomery Ladder su due processori: uno si occuperà della Point Addition e l'altro della Point Doubling riuscendo a dimezzare il tempo di calcolo ed arrivare a $O(\lfloor log_2(n) \rfloor)$ tempo di operazioni. Lo stesso tempo computazionale viene impiegato dalla Double&Add solo nel caso in cui la notazione binaria sia del tipo $n = 2^x$, $x \in \mathbb{N}$. Si conclude che la Montgomery Ladder parallelizzata è generalmente più veloce della Double&Add.

3.3 Campi Finiti

Prima di iniziare a parlare di algoritmi fondati sulle curve ellittiche, è bene precisare che la quantità *finita* di memoria dei calcolatori non è sufficiente per gestire punti di *infinite* cifre. I campi modulari $\mathbb{F}_p$ restringono le coordinate dei punti nell'intervallo $[0, p-1]$ e favoriscono l'implementazione dei concetti studiati fino ad ora. In ambito crittografico, vanno esaminate tre proprietà in particolare: la cardinalità (o ordine) di gruppi e sottogruppi; il punto generatore del sottogruppo; il cofattore del sottogruppo.
Un campo finito sul quale si intende definire una curva ellittica deve rispondere a due proprietà principali: il suo ordine p deve essere un numero primo; la caratteristica $char(\mathbb{F}_p)$ deve essere diversa da 2 e da 3 per garantire la scrittura nella forma estesa di Tate-Weierstrass (3.1)

$$\mathbb{F}_p,\ p \neq \{2, 3\}$$
$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \text{ mod(p)}$$

L'equazione breve di Weierstrass (3.3) che governa la curva assume di conseguenza un carattere modulare:

$$E(\mathbb{F}_p) = \begin{cases} y^2 = x^3 + ax + b \bmod(p) \\ 4a^3 \neq 27b^2 \bmod(p) \end{cases} \bigcup \{\mathcal{O}\} \tag{3.5}$$

3.3.1 Cardinalità

Per una curva ellittica modulare non possiamo dire che la sua cardinalità corrisponda all'ordine p: non tutti i punti del campo soddisfano l'equazione (3.5), a maggior ragione se consideriamo la condizione $4a^3 \neq 27b^2 \bmod(p)$ notiamo come vengano esclusi alcuni valori. Per avere un intervallo di riferimento entro il quale possa variare la cardinalità $\#E$ si è soliti ricorrere al "*Teorema di Hasse*". Il teorema definisce K_q come un campo finito avente ordine q (primo, diverso da 2 e da 3), E una curva ellittica definita su tale K. Detta $\#(E_K)$ la cardinalità della curva, il valore che questa assume è da considerarsi nella disequazione 3.6:

$$\#(E_K) \leq (1 \pm \sqrt{q}\,)^2 \tag{3.6}$$

e quindi all'interno dell'intervallo

$$\lceil q + 1 - 2\sqrt{q} \rceil \leq \#(E_K) \leq \lfloor q + 1 + 2\sqrt{q} \rfloor$$

Una curva E che presenti cardinalità pari ad un numero primo è detta "*curva prima*". Se la curva presenta cardinalità $\#(E_K) = q$, allora viene detta "**anomala**". Infine, la curva viene detta "**supersingolare**" se $\#(E_K) = q + 1$ [32]; alternativamente se, data la cardinalità $\#(E_K) = q + 1 + t$, vale $t = 0 \bmod(q)$ [33].
Prendiamo dei numeri q per i quali vengono rispettate le ipotesi del teorema: definita con "min" la cardinalità minima per la curva e "max" il suo valore massimo, applicando il teorema di Hasse troviamo

q	min	max
5	2	10
19	12	28
67	52	84
313	278	349
971	910	1034

È importante notare che il teorema usato restituisce la cardinalità della curva includendo il punto all'infinito.
Un algoritmo per determinare univocamente la cardinalità $\#E$ è quello di Schoof [48]. Chiamiamo $N \geq 4\sqrt{q}$ un numero la cui scomposizione in fattori primi corrisponde a $N = \prod_{i=1}^{j} p_i$. Il calcolo di $\#E$ viene reso possibile dal Teorema Cinese del Resto (descritto nel capitolo 5.1.4) applicato al sistema seguente:

$$\begin{cases} \#E \bmod(p_1) \\ \#E \bmod(p_2) \\ \dots \\ \#E \bmod(p_j) \end{cases}$$

3.3.2 Punti Generatori

Un punto P che generi un gruppo ciclico viene detto Generatore del gruppo. Tutti i punti appartenenti a questo gruppo possono esser calcolati applicando ripetitivamente la legge di gruppo al punto P.
Prendiamo la curva $E : y^2 = x^3 + 2x + 2 \bmod(17)$ ed il punto $P = (5, 1)$. La

curva ed il sottogruppo generato da A contengono entrambi 19 elementi in totale. L'elemento successivo possiamo dirlo $Q = 2P$ e si calcola con le stesse formule della Point Doubling (formule 3.4) espresse in modulo 17 per il caso in esame.

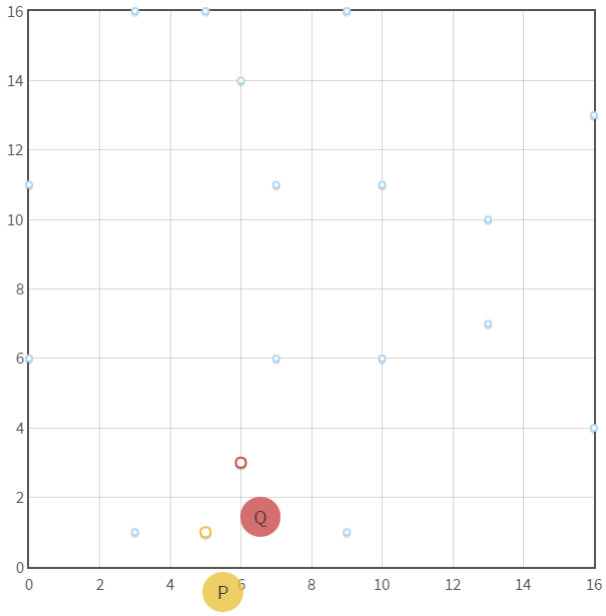


Figura 3.7: Curva $y^2 = x^3 + 2x + 2 \bmod(17)$, punto $P = (5, 1)$, $Q = 2P = (6, 3)$

Si noti che ora **l'asse di simmetria** non risulta più essere $y = 0$ ma **diventa** $\boldsymbol{y = p/2}$ eccezion fatta per i punti che giacciono sull'asse x. Questi punti, descritti con le notazioni equivalenti tra loro $(x, y) = (x, 0) \bmod(y)$, risultano simmetrici a loro stessi essendo vero che $y = 0 \bmod(y)$.
È possibile fare un'ulteriore verifica del risultato contando i punti tramite una serie di Point Addition: partiamo dal punto $A = (5, 1)$ identifichiamo le coordinate di nA con (nA_x, nA_y)

n	nA_x	nA_y	n	nA_x	nA_y
1	5	1	10	7	11
2	6	3	11	13	10
3	10	6	12	0	11
4	3	1	13	16	4
5	9	16	14	9	1
6	16	13	15	3	16
7	0	6	16	10	11
8	13	7	17	6	14
9	7	6	18	5	16
			19	/	/

Il punto $19A$ rappresenta il punto all'infinito e verifica che la cardinalità della curva E sul campo $\mathbb{F}_{17}$ è 19. Trovandoci in un campo modulare, il punto $20A$ esiste e corrisponde alla Point Addition di $19A + A = \mathcal{O} + A = A$. In generale possiamo scrivere ogni punto kP basandoci sulla seguente espressione:

$$[k \bmod(\#E_K)] \cdot P \tag{3.7}$$

Come anticipato nel "Caso Particolare (1)" abbiamo $A + 18A = 19A = \mathcal{O}$ poichè si applica la Legge di Gruppo a due punti distinti di eguale ascissa ($x_A = x_{18A} = 5$) ma ordinata diversa ($y_A = 1$, $y_{18A} = 16$). Sebbene tale caso presupponesse che i due punti sommati dovessero essere simmetrici tra loro, ora la simmetria sembra venir meno. Ricordando che l'asse di simmetria è $y = p/2 = 17/2 = 8.5$ possiamo facilmente verificare che i due punti rispettino effettivamente la simmetria. Possiamo, ad esempio, sommare le ordinate dei due punti e dividere per due, ottenendo il punto medio tra A e $18A$:

$$\frac{y_A + y_{18A}}{2} = \frac{1 + 16}{2} = \frac{17}{2} = 8.5$$

confermando che il punto medio tra A e $18A$ si trova sull'asse di simmetria e quindi verifica quanto affermato prima.
Restando vero che per ogni punto della curva esiste il suo simmetrico, possiamo evidenziare un risultato importante:

$$2A + 17A = 3A + 16A = 4A + 15A = \ldots = 19A = \mathcal{O}$$

Ognuna di queste coppie di punti presenta ascissa uguale ed ordinata diversa, sono tutte coppie di punti simmetrici tra loro. Si arriva allora alle seguenti *Considerazioni Personali*:

1. Il calcolo dei 19 punti ha mostrato una ricorsività nei valori delle **ascisse**: i primi 9 punti hanno ascissa nella sequenza $5, 6, 10, 3, 9, 16, 0, 13, 7$ mentre i successivi 9 hanno ascissa in sequenza invertita. Inoltre solo due punti *successivi tra loro* presentano la stessa ascissa: $9A$ e $10A$, punti simmetrici, la cui somma è $19A = \mathcal{O}$. Si è cercato quindi di ideare un metodo efficace per determinare la cardinalità delle curve in campi modulari arrivando ad affermare quanto segue:
   "Dati due punti nG e $(n+1)G$ di una curva E definita su un campo finito K_q, detto H il sottogruppo generato dal punto G, se vale $x_n = x_{n+1}$ allora le seguenti affermazioni

$$\begin{cases} nG + (n+1)G = \mathcal{O} \\ \#H = n + (n+1) = 2n + 1 \end{cases} \tag{3.8}$$

   sono vere e la cardinalità è sempre **dispari**. È possibile generalizzare la (3.8) per ogni coppia di punti $(n-k)G$ e $(n+1+k)G$ aventi ascisse uguali $x_{n-k} = x_{n+k+1}$. La distanza, in termini di successive Point Addition tra i due punti, è pari a $2k+1$. Alcuni esempi di quanto affermato sono dati dalle seguenti relazioni: $(2A + 17A) = (3A + 16A) = (4A + 15A) = \mathcal{O}$.

   Se invece si trova che, dati due punti $(n-1)G$ ed $(n+1)G$, le loro ascisse coincidano $x_{n-1} = x_{n+1}$, allora la (3.8) diventa

$$\begin{cases} 2nG = \mathcal{O} \\ \#H = 2n \end{cases} \tag{3.9}$$

   in questo caso abbiamo sempre cardinalità **pari**. Anche in questo caso evidenziamo una generalizzazione: la (3.9) vale per ogni coppia di punti $(n-k)G$ e $(n+k)G$ di medesima coordinata $x_{n-k} = x_{n+k}$. La distanza tra i due punti corrisponde a calcolare $2k$ Point Addition".

Possiamo verificare che quanto espresso valga solo all'interno di sottogruppi del campo K_p servendoci di un esempio: data la curva $E : y^2 = x^3 + 4x + 5$ nel campo $\mathbb{Z}/31\mathbb{Z}$ e punto generatore $P = (18, 9)$. Dal teorema di Hasse la curva ha cardinalità compresa nell'intervallo $[27, 37]$ ed il conto esatto mostra che $\#E_K = 28$. Il punto P considerato genera un sottogruppo H i cui punti sono $P = (18, 9), 2P = (0, 6),\ 3P = (7, 29),\ 4P = (7, 2),\ 5P = (0, 25),\ 6P = (18, 22),\ 7P = \mathcal{O}$. La ricorsività delle ascisse vista precedentemente si mantiene valida in H, deduciamo che la cardinalità di questo sottogruppo è 7 e resta valida la relazione

$$P + 6P = 2P + 5P = 3P + 4P = \mathcal{O} \Rightarrow \#H = 7$$

in quanto $3P$ e $4P$ sono punti successivi tra loro ed ascissa $x_{3P} = x_{4P} = 7$. Infine considerando che $\#H \neq \#E_K$ si dimostra che l'affermazione precedente resta valida all'interno di sottogruppi $H \subset K_p$.

2. In un sottogruppo, possiamo calcolare il punto simmetrico di $Q = (x_Q, y_Q)$ considerando "$-Q = (x_Q, \#H - y_Q)$", dove $\#H$ è cardinalità del sottogruppo $H \subset \mathbb{F}_{31}$. Utilizzando i punti Q e $-Q$, cerchiamo di determinare $\#H$.
   La Legge di Gruppo ci permette di dire che $Q - Q = \mathcal{O} = \#H \cdot G$ in quanto simmetrici ed appartenenti ad un campo modulare. Dette n e m le loro molteplicità tali per cui $Q = nG$ e $-Q = mG$, la considerazione precedente ci porta ad affermare che $n + m = \#H$. La conclusione alla quale arriviamo è che, per due numeri naturali n ed m

$$n, m \in \{0, 1, \ldots, q - 1\} \mid Q = nG, -Q = mG \Rightarrow \#H = n + m \tag{3.10}$$

   Siamo quindi riusciti a determinare la cardinalità di H a partire da due punti simmetrici tra loro.

3.3.3 Cofattore

Riprendiamo il concetto di cofattore di un sottogruppo. Come detto nel capitolo (2.1), sulla base del teorema di Lagrange possiamo affermare che dato un gruppo G con cardinalità $\#G$, il suo sottogruppo H avrà cardinalità detta $\#H$ tale che valga la seguente formula:

$$h = \frac{\#G}{\#H},\ h \in \mathbb{Z}$$

Dove con h si esprime il cofattore di H.
Possiamo far uso del cofattore per determinare la cardinalità di un sottogruppo partendo dalla sola $\#G$. Prendiamo in esempio la curva $E : y^2 = x^3 + 4x + 5$ nel campo $\mathbb{F}_{31}$. La curva presenta cardinalità $\#G = 28$ i cui divisori interi sono $T = \{1, 2, 4, 7, 14, 28\}$. A questo punto dobbiamo introdurre un generatore: preso il punto $P = (18, 9)$ cerchiamo di ottenere il punto $\mathcal{O}$ mediante successive Point Multiplication tP dove $t \in T$ è il generico elemento di T e corrisponde ai divisori interi di $\#G$. Troviamo che $7P = 14P = 28P = \mathcal{O}$ e la relazione (3.7) non è più valida nel sottogruppo poichè viene verificata per più di un valore. Possiamo esporre tale formula in modo più generale dicendo che

$$[k \bmod(\#H)] \cdot P \tag{3.11}$$

In questa ottica affermiamo che $\#H = 7$ per la curva in considerazione per via del fatto che $t_5 = 14$, $t_6 = 28$ sono multipli interi di $\#H$: per la ciclicità dell'algebra modulare questi valori corrispondono allo stesso elemento in modulo 7. In conclusione solo il più grande divisore primo t rappresenta la cardinalità del sottogruppo, in questo caso $\#H = 7$ e rispettivo cofattore $h = \#G/\#H = 28/7 = 4$.
Una curva come $E : y^2 = x^3 + 2x + 4 \bmod(13)$ presenta cardinalità $N = 17$ e, per ogni suo punto, viene identificato un sottogruppo H di ancora 17 elementi. Infatti i divisori di N sono solo 1 e se stesso determinando che, preso un qualsiasi punto della curva che non sia $\mathcal{O}$, esisterà un sottogruppo H di cardinalità $\#H = 17$. Per la conclusione proposta si è dovuto escludere il divisore "1" per il seguente motivo: preso un punto P, generatore del sottogruppo H, sappiamo che bisogna rispettare la relazione $\#H \cdot P = \mathcal{O}$; se avessimo assunto $\#H = 1$ avremmo ottenuto

$$\#H \cdot P = 1P = P = \mathcal{O}$$

si sarebbe identificato quindi un sottogruppo avente il solo elemento neutro della Legge di Gruppo e nessun altro punto.
Sulla base di quanto detto finora possiamo concludere che i sottogruppi H presentano cofattore $h = 1$ qualora tale sottogruppo comprenda tutti (e soli) i punti della curva.

Capitolo 4

Algoritmi per la Sicurezza Informatica

Consideriamo in questo capitolo alcuni algoritmi della sicurezza informatica ed i loro corrispondenti algoritmi basati sugli studi effettuati. Tratteremo di: crittografia simmetrica, asimmetrica e firma digitale.
Riguardo la crittografia verrà mostrato il paradigma ElGamal per cifrare in modo sicuro un messaggio, seguito poi dalla sua variante per mezzo delle curve ellittiche, la *Crittografia Ellittica*. Proseguiremo con la crittografia asimmetrica illustrando il funzionamento del protocollo RSA evidenziandone (qui e nei capitoli successivi) i motivi che spingono ad approcciarsi verso nuovi algoritmi utilizzando la teoria delle curve ellittiche. Infine tratteremo della firma digitale implementata negli algoritmi DSA e ECDSA.

4.1 ElGamal

La crittografia di un messaggio tramite l'algoritmo di ElGamal si basa sulla generazione di parametri pubblici e privati [17]. Identifichiamo i parametri pubblici con la notazione $t = (p, q, g, K^+)$, la scelta di questi avviene nel seguente modo:

1. **p** deve essere un numero primo. Si preferiscono numeri molto grandi in modo da garantire la sicurezza dell'algoritmo, tuttavia ciò comporta un maggior impiego di risorse per i calcoli;

2. **q** deve essere un divisore primo di "$p - 1$";

3. **g** deve corrispondere ad un numero scelto nell'intervallo $[1, p-1]$ avente ordine pari a q.

Possiamo quindi definire la chiave privata K^- come un numero intero, casuale, scelto nell'intervallo $[1, q-1]$. Complementare ad essa, calcoliamo la chiave pubblica come $K^+ = g^{K^-} \mod(p)$.
Riuscire a trovare K^- a partire dai parametri pubblici costituisce il "*Problema del Logaritmo Discreto*". Verrà fatto riferimento al problema appena descritto mediante la notazione DLP, dall'inglese "**D**iscrete **L**ogarithm **P**roblem".

Ci interessiamo ora ad esporre come funzioni la crittografia di un messaggio m. L'utente Alice, intenzionata all'invio sicuro del suo m_A, si comporta come segue:

1. Sceglie un numero casuale k appartenente all'intervallo $[1, q-1]$;
2. Calcola $c_1 = g^k \mod(p)$;
3. Calcola $c_2 = m \cdot (K^+)^k \mod(p)$;
4. Invia la coppia (c_1, c_2) al suo interlocutore Bob.

Bob è capace di decifrare il messaggio procedendo al calcolo:

$$\begin{aligned} c_2 \cdot c_1^{-K^-} \mod(p) = & \\ &= m \cdot (K^+)^k \cdot g^{k(-K^-)} \mod(p) \\ &= m \cdot g^{k(K^-)} \cdot g^{-k(K^-)} \mod(p) \\ &= m \end{aligned}$$

In questo modo si è recuperato il messaggio originale, tuttavia sia per Alice che per Bob è necessario conoscere la chiave privata K^- implicando una precedente comunicazione nella quale è avvenuto lo scambio della chiave simmetrica, il segreto condiviso.

4.2 Crittografia Ellittica

I parametri pubblici di cui ci si serve per l'implementazione della crittografia ellittica sono:

- i numeri **a**, **b** che serviranno da coefficienti nell'equazione (3.5) per descrivere la curva ellittica;
- un numero primo **p** atto a specificare la cardinalità del campo $\mathbb{F}p$;
- il punto **G**, generatore del sottogruppo H;
- il numero primo $\mathbf{n} = \#H$, rappresentante la cardinalità del sottogruppo;
- il numero **h**, ovvero il cofattore.

Per semplicità andiamo ad identificare tutti questi valori sotto il nome di t; la sua espressione è quindi:

$$t = (a, b, p, G, n, h)$$

4.2.1 Codificare il Messaggio

Codificare, inviare e decodificare un messaggio di testo in termini di curve ellittiche è un problema leggermente più complesso rispetto ad altri algoritmi. ElGamal, e similmente qualsiasi altro algoritmo che preveda la cifratura di un messaggio di testo, divide il messaggio in blocchi a lunghezza fissa di B bit. Eventualmente una parte dei bit finali viene posta a 0 in modo da ottenere un completamento a B bit favorendo una corretta codifica per messaggi di lunghezza arbitraria. Ogni parola di B bit viene quindi criptata secondo le specifiche dell'algoritmo, inviata al destinatario ed infine decriptata. Il messaggio originale è dato dalla concatenazione di tutte queste parole decriptate.
Quanto avviene per le curve ellittiche è differente: abbiamo bisogno di rappresentare un messaggio di testo in punti sulla curva. Dobbiamo assicurarci che ogni simbolo abbia una corretta rappresentazione, individuata da un punto K-razionale. Aggiungiamo il fatto che cambiare i coefficienti a, b della curva oppure la cardinalità p del campo $\mathbb{F}_p$ identifica una nuova, diversa, curva e di conseguenza una differente

rappresentazione per ciascun carattere del messaggio. *Ogni curva porta a diverse rappresentazioni dello stesso messaggio.*
Assumiamo di voler codificare il nostro messaggio velocemente e semplicemente facendo uso di un algoritmo probabilistico [18]. La prima cosa di cui dobbiamo assicurarci è che la probabilità P di fallire nella codifica sia molto bassa; a tale scopo definiamo un numero k come $2^{-k} < P$. Valori concreti per questo parametro possono esser scelti nell'intervallo $[30, 50]$.
Scegliamo di codificare i caratteri del testo in chiaro (lettere, numeri e/o simboli) nella forma di numeri interi. Possiamo, ad esempio, codificare un testo in cui vengono permessi solo i caratteri $A, B, \ldots, Z$ mappandoli sui numeri interi $1, 2, \ldots, 26$, per un totale di T caratteri codificati. Inoltre dobbiamo permettere che ogni carattere possa esser rappresentato come un punto distinto dagli altri sulla curva: la cardinalità p del campo diviene di fondamentale importanza per gestire messaggi con molti caratteri diversi. Per permettere un tale requisito dobbiamo assicurarci quindi che valga la relazione $p > Tk$. Gli ultimi due parametri da definire sono: c ovvero il valore numerico che rappresenta il nostro carattere; j un numero compreso nell'intervallo $j \in [1, k]$. L'algoritmo di codifica consiste quindi nel considerare dapprima $j = 1$ e calcolare

$$\begin{cases} x = ck + j \bmod(p) \\ y^2 = x^3 + ax + b \bmod(p) \end{cases}$$

In caso la y trovata non corrispondesse ad un numero intero, si procede ad incrementare la j di 1 e si calcolano di nuovo le coordinate x, y. Il fallimento dell'algoritmo consiste nel non poter trovare un valore di y accettabile per nessun valore di j.
Se la codifica è andata a buon fine allora $c \iff P = (x, y)$ con le coordinate appena calcolate. Iterare l'algoritmo per ogni carattere del messaggio permette di codificarne l'intero contenuto in una serie di punti.

Assunto che siamo riusciti a codificare correttamente il carattere c, passiamo a trattare della sua corretta decodifica. Otteniamo il carattere iniziale applicando la seguente formula:

$$c = \left\lfloor \frac{x-1}{k} \right\rfloor$$

Qui intendiamo il carattere c come la sola parte intera del risultato, per questo si è evidenziata l'operazione di arrotondamento per difetto $\lfloor \cdot \rfloor$.

Codifica proposta

Codificare e decodificare un messaggio m devono essere due funzioni facilmente computabili ed eseguibili in modo deterministico per entrambi Alice e Bob. L'idea che si propone per soddisfare tali richieste si basa sulle seguenti considerazioni:

1. Ogni carattere ammesso C nella codifica viene reso nel suo corrispondente valore decimale c in ASCII secondo quanto riportato in [26];

2. Ipotizzando di accettare x caratteri diversi tra loro, la curva E dovrà presentare una cardinalità minima di $n = x + y$. Il numero y corrisponde alla somma dei punti minimi necessari per determinare ogni punto sulla curva in modo unico. Punti di cui tener conto sono: il punto all'infinito; K^S che serve da chiave simmetrica qualora la codifica avvenga tramite crittografia simmetrica; K^+ che serve da chiave pubblica qualora si utilizzi una crittografia asimmetrica; S detto Seed, un casuale per mezzo del quale possiamo aggiungere entropia negli algoritmi ellittici, introdotto più avanti;

3. Dai parametri pubblici t prendiamo G, il punto generatore. La codifica consiste nel far corrispondere ogni carattere al suo corrispondente punto cG della curva

La decodifica consistere nell'avere una tabella di due colonne strutturata similmente alla tabella (4.2.1)

Carattere $\rightarrow$ c	cG
@ $\rightarrow$ 64	x_{64}
A $\rightarrow$ 65	x_{65}
a $\rightarrow$ 97	x_{97}
7 $\rightarrow$ 55	x_{55}
. $\rightarrow$ 46	x_{46}

La colonna "cG" tiene conto solo della coordinata x_c del punto $cG = (x_c, y_c)$ in modo da diminuire la memoria necessaria al mantenimento della tabella. Possiamo

dimezzare i record da memorizzare se ad ogni coordinata x_c facciamo corrispondere due caratteri consecutivi della tabella ASCII. Indichiamo i due caratteri "@" e "A" con la stessa ascissa x_c. Al carattere con codice c pari facciamo corrispondere l'ordinata y_c positiva, al secondo facciamo corrispondere la y_c negativa. Procedendo in modo simile per le altre lettere arriviamo ad ottenere una tabella simile:

Caratteri → c	cG
(@, A) → 64	x_{64}
(b, c) → 98	x_{98}
(0, 1) → 48	x_{48}
(6, 7) → 54	x_{54}
(., /) → 46	x_{46}

Esempio: Alice sta codificando un messaggio da mandare a Bob e si trova a codificare il carattere "1". Il suo valore ASCII è 49: decrementa quindi il codice a $c = 48$ e computa $cG = 48G$. Il carattere da inviare deve corrispondere all'ordinata $y_{48} < 0$ quindi Alice simmetrizza il punto *se necessario* ed infine invia il punto $cG = (x_{48}, -y_{48})$ a Bob. Quest'ultimo cercherà la coordinata x_{48} nella tabella trovando il valore $c = 48$. Guardando il segno di y_{48} Bob deduce che il carattere effettivo non è 48 ma il successivo ovvero $c = 49$ dal quale ottiene correttamente "1".

Pseudocodici per l'implementazione.
Input codifica: carattere C, punto G della curva (preso dai parametri t),

Algorithm 1 Codifica

```
c ← ASCII2DEC(C)                          ▷ Converte C in intero
is_odd ← c%2
if is_odd then
    c ← c − 1
end if
Q ← cG
if ((y_Q > 0 and is_odd) or (y_Q < 0 and !is_odd)) then return -Q
end if
return Q
```

Input decodifica: il punto $Q = (x_Q, y_Q)$ inviato da Alice

Algorithm 2 Decodifica

```
d ← Decode_Table(x_Q)                    ▷ Prende il valore ASCII di x_Q
if y_Q < 0 then
    d ← d + 1
end if
return DEC2ASCII(d)                      ▷ Ritorna il carattere ASCII
```

Infine è possibile aggiungere casualità agli algoritmi visti facendo uso di un segreto condiviso tra le due parti ma oscuro a tutti gli altri utenti della rete: la chiave simmetrica $K^S = (x_S, y_S)$. Essendo anch'essa un punto della curva, definiamo la funzione "$shuffle(x_S, y_S, n)$" che ne usa le coordinate per dare in output un numero s modulo n. Questo sarà ignoto a tutti meno che ad Alice e Bob i quali possono computare $(s+c)G$ per codificare in modo unico i caratteri ASCII e generare la loro tabella $(s + c) \to (s + c)G$. Lo pseudocodice per la codifica cambia leggermente: si dovrà ora calcolare il punto $Q \leftarrow (c + s)G$ invece che $Q \leftarrow cG$.

4.2.2 Inviare il Messaggio

Si evidenzia adesso come avviene l'invio in modo sicuro di un messaggio da parte di Alice. Assumiamo che, detto m il messaggio, m_i sia il suo carattere i-esimo da codificare.

1. Il primo passo che compie Alice è prendere m_i e rappresentarlo come punto M della curva mediante la codifica proposta;

2. Scegliamo la chiave privata d come un numero intero appartenente all'intervallo $[1, n - 1]$;

3. Calcoliamo la chiave pubblica come $P = dG$;

4. Calcoliamo infine il messaggio cifrato $C = M + dP$;

5. Si invia all'interlocutore Bob la coppia (P, C).

Bob è in grado di decifrare il messaggio trovando dapprima $M = C - dP$ ed infine estraendo il messaggio m dal punto M.
Verifichiamo il corretto funzionamento dell'algoritmo: sappiamo che il messaggio crittografato è $C = M + dP$ e che $P = dG$; il coefficiente d è analogo alla chiave simmetrica di ElGamal. Essendo vero che tale chiave simmetrica è conosciuta da entrambi gli interlocutori e loro solamente, Bob (come anche Alice) sarà in grado di calcolare il punto dP usando l'algoritmo della Double&Add o la Montgomery Ladder. L'ultimo passo da compiere è applicare la Legge di Gruppo tra il punto C ed il punto $S = -dP$.

Per questo algoritmo, parliamo di ECDLP riferendoci al problema di calcolare la chiave privata d a partire dal punto P. La sigla ECDLP, intesa come **E**lliptic **C**urve **D**iscrete **L**ogarithm **P**roblem, è data per analogia al problema visto con ElGamal. Inoltre, sebbene le curve ellittiche non presentino calcoli di esponenziali o di logaritmi, la sigla mira ad evidenziare la difficoltà computazionale del problema. Risolvere l'ECDLP si ritiene essere un compito ben più oneroso del DLP come verrà mostrato nel capitolo 5.

4.3 RSA

Il sistema a chiave asimmetrica (o pubblica) più noto e diffuso è RSA, utilizzato per autenticare utenti sulla rete e per garantire integrità dell'informazione. L'algoritmo si propone di garantire la protezione della comunicazione tra due interlocutori Alice (A) e Bob (B) facendo uso di due chiavi, una privata K^- ed una pubblica K^+. La matematica alla base di RSA è data dal *Teorema di Eulero* secondo il quale "se due numeri a e n sono coprimi allora $a^{\theta(n)} = 1 \bmod(n)$". La funzione $\theta(n)$ di Eulero corrisponde al totale di numeri interi, positivi, minori di z e coprimi (senza alcun fattore in comune) ad esso. Due esempi: $\theta(10) = 4$, ovvero $\{1, 3, 7, 9\}$; preso invece un numero primo $\theta(11) = 10$. Per ogni numero primo z risulta chiaro che la sua funzione θ corrisponda a $z - 1$.
Una proprietà importante di questo teorema [25] è che, dati due numeri primi p e q, il loro prodotto è $n = pq$ per il quale possiamo scrivere:

$$\theta(n) = \theta(pq) = \theta(p) \cdot \theta(q) = (p-1)(q-1)$$

Mostriamo di seguito come vengono generate le chiavi crittografiche in RSA:

1. Si scelgono due numeri casuali p e q. Per un corretto funzionamento dell'algoritmo bisogna assicurarsi che i due numeri siano entrambi **primi** e di **ordine simile** ma *lunghezza in bit differente* in modo da rendere la fattorizzazione più difficile [16];

2. Si calcola $n = p \cdot q$. Tale numero verrà detto **modulo** per le chiavi privata e pubblica, inoltre la lunghezza in bit di n corrisponde alla *lunghezza della chiave* usata per l'algoritmo;

3. Si calcola $\theta(n) = (p-1) \cdot (q-1)$, questo valore va mantenuto segreto;

4. Va ora scelto un numero e_A tale che $1 < e_A < n$, coprimo a $\theta(n)$;

5. Infine va scelto un numero d_A tale che "$e_A d_A - 1$" sia interamente divisibile per $\theta(n)$, ovvero $e_A d_A = 1 \bmod(\theta(n))$;

6. Si ottengono quindi le chiavi $K_A^+ = (n, e_A)$, $K_A^- = (n, d_A)$.

I passaggi illustrati vanno eseguiti da entrambi gli interlocutori A e B. Successivamente segue lo scambio delle chiavi pubbliche in modo che Alice venga in possesso della terna (K_A^-, K_A^+, K_B^+) mentre Bob avrà (K_B^-, K_B^+, K_A^+). Immaginiamo che sia Alice a voler inviare un messaggio m_A a Bob. Tale azione è possibile finchè tale messaggio abbia lunghezza in bit inferiore ad n. Lo scambio del messaggio avviene previa crittografia:

$$c_A = (m_A)^{e_B} \bmod(n)$$

Alice ha quindi crittografato il suo messaggio con la chiave pubblica di Bob e procede all'invio del messaggio c_A. Questo procedimento implica che solo l'utente Bob sarà in grado di decifrare il messaggio tramite la sua chiave privata, infatti egli procederà come segue:

$$m_A = (c_A)^{d_B} \bmod(n)$$

È chiaro che Bob ha ora ottenuto il messaggio originale di Alice.
Ciò che dobbiamo dimostrare è quindi:

$m_A = (c_A)^{d_B} \bmod(n)$	Sostituiamo $c_A = (m_A)^{e_B} \bmod(n)$
$m_A = [(m_A)^{e_B}]^{d_B} \bmod(n)$	Raccogliamo l'esponente di m_A
$m_A = m_A^{e_B \cdot d_B} \bmod(n)$	Applichiamo: $e_B d_B \bmod(\theta(n)) = 1 \Rightarrow e_B d_B = 1 + h\theta(n)$
$m_A = m_A^{1+h\theta(n)} \bmod(n)$	Semplifichiamo l'esponente
$m_A = m_A(m_A^{\theta(n)})^h \bmod(n)$	Applichiamo il teorema di Eulero
$m_A = m_A(1)^h \bmod(n)$	Dato che $1^h = 1$ scriviamo
$m_A = m_A \bmod(n)$	c.v.d.

Il numero h usato è un qualsiasi numero intero, la sua utilità è esprimere il numero $e_B d_B - 1$ come multiplo intero di $\theta(n)$. Si nota nell'ultimo passaggio come non sia necessario fornire ulteriori dettagli sui valori di h in quanto figura come esponente al numero "1". Un ultima cosa da notare è che il risultato finale $m_A = m_A \bmod(n)$ determina la limitazione nell'inviare solo messaggi m_A con lunghezza in bit inferiore ad n.
Aver analizzato l'algoritmo RSA ci permette di fare osservazioni critiche sul suo funzionamento:

- Il messaggio m_A deve avere lunghezza in bit inferiore al modulo n, lo notiamo dal risultato finale in cui $m_A = m_A \bmod(n)$. Messaggi molto lunghi necessitano di un modulo grande rendendo la computazione dei due numeri primi p e q un compito difficile e dispendioso in termini di tempo di calcolo

- l'intera sicurezza dell'algoritmo si basa sulle divisioni in modulo n per cui fattorizzare tale numero e trovare i suoi fattori primi permette di computare il segreto d a partire dalla chiave pubblica (n, e). Si arriva quindi a decifrare il messaggio crittato c tramite la procedura standard

- la difficoltà nel "*rompere*" RSA consiste nel riuscire a trovare il messaggio m partendo dal testo cifrato $c = m^e \bmod(n)$.

4.4 Crittografia Ellittica Asimmetrica

Quanto dimostrato da Rivest, Shamir e Adleman viene ora applicato, analogamente, alle curve ellittiche. Ricordiamo che i parametri pubblici da considerare sono dati dalla sestupla $t = (a, b, p, G, n, h)$. Come precedentemente definito nel capitolo 4.2.2 la chiave privata è il numero intero, casuale, $d \in [1, n-1]$, la chiave pubblica è il punto della curva $P = dG$.
Gli interlocutori Alice e Bob prima di poter comunicare devono procedere allo scambio delle chiavi pubbliche ottenendo una terna di valori ciascuno: Alice possiede (d_A, P_A, P_B), Bob invece possiede (d_B, P_B, P_A).
I calcoli che venivano fatti in RSA sono ora molto accelerati: non abbiamo bisogno di calcolare numeri primi; non siamo costretti a scegliere numeri abbastanza grandi tali che il loro prodotto sia, in bit, maggiore del messaggio da inviare.

Alice, per comunicare con Bob, deve provvedere alla rappresentazione del suo messaggio m_A in punti sulla curva. Detto allora m_i l'i-esimo carattere del messaggio:

1. Rappresenta il carattere m_i come punto M della curva;
2. Calcola il messaggio cifrato $C = M + d_A P_B$;
3. Invia a Bob il messaggio C.

Avendo utilizzato la chiave P_B per ottenere C, solo Bob sarà in grado di decifrare il messaggio arrivatogli. Egli calcola semplicemente $M = C - d_B P_A$ ed ottiene il punto M. Dovrà ora prendere il messaggio m_i che viene rappresentato da quel punto tramite l'algoritmo di decodifica. L'invio del messaggio completo m_A è possibile mediante iterazioni successive dell'algoritmo per ogni carattere i. Infine, assemblare ordinatamente tutti gli m_i significa ritrovare il messaggio m_A originale.
Dimostriamo che il messaggio trovato da Bob è effettivamente quello criptato da

Alice:

$$
\begin{aligned}
C &= M + d_A P_B && \text{Vediamolo in termini di } M \\
M &= C - d_A P_B && \text{Ricordiamo che } P_B = d_B G \\
M &= C - d_A d_B G && \text{Valendo la proprietà commutativa} \\
M &= C - d_B d_A G && \text{Per Bob vale inoltre che } d_A G = P_A \\
M &= C - d_B P_A
\end{aligned}
$$

La formula ottenuta si basa su parametri conosciuti da Bob per questo gli è possibile calcolare M. Inoltre abbiamo dimostrato la veridicità e la validità dell'algoritmo.
I calcoli compiuti da Alice sono: x passaggi per calcolare dP; y passaggi per rappresentare il singolo carattere m_i in un punto della curva; un passaggio che consiste nella Point Addition per computare C. Abbiamo quindi che gli $y + 1$ passaggi per arrivare a C vengono svolti T volte in modo da rappresentare tutti i caratteri del messaggio m_A. Il totale di operazioni svolte è:

$$\text{Operazioni} = x + T(y + 1)$$

4.4.1 ECDH

L'algoritmo ECDH, acronimo di **E**lliptic **C**urve **D**iffie-**H**ellman, si basa sull'accordo di una chiave. Dette Alice e Bob le parti in gioco, l'algoritmo si basa sui seguenti passi:

1. Entrambi A e B generano le loro chiavi private e pubbliche. Alice avrà chiave privata un numero casuale compreso tra 1 ed n-1, $d_A = random(1, n-1)$, e pubblica $P_A = d_A G$; Bob avrà invece la chiave privata $d_B = random(1, n-1)$, e pubblica $P_B = d_B G$

2. Si passa allo scambio di chiavi pubbliche P_A e P_B. Alice si trova ora in possesso della terna (d_A, P_A, P_B), similmente Bob conosce (d_B, P_B, P_A). Questi due passi sono gli stessi iniziali visti precedentemente

3. Entrambi A e B passano dunque a calcolare il Segreto condiviso $K^S = (x_S, y_S)$, Alice computa $K_A^S = d_A P_B$ e Bob, similmente, $K_B^S = d_b P_A$.

I segreti condivisi K_A^S e K_B^S si dimostrano essere uguali: avendo detto che le curve ellittiche formano un gruppo abeliano, è vero che vale la proprietà commutativa per cui è valida la seguente:

$$K_A^S = d_A \cdot P_B = d_A \cdot (d_B \cdot G) = d_B \cdot (d_A \cdot G) = d_B \cdot P_A = K_B^S$$

Alice e Bob possono ora comunicare in modo sicuro usando una crittografia simmetrica a partire dal segreto K^S tramite funzioni che ne utilizzano le coordinate per ottenere un risultato r mediante il quale crittografare i dati successivi. Un esempio può essere $C = rG + M$ intendendo con C il messaggio crittografato.

4.4.2 Un esempio reale

L'istituto NIST[I] ha proceduto alla standardizzazione di diverse curve ellittiche con il risultato di aver definito un diverso "protocollo" sulla base della curva considerata. Parliamo in questo esempio del protocollo **secp256k1**; le notazioni usate sono in base esadecimale:

- **p** = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F = $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 2^0$;
- **a** = 0;
- **b** = 7;
- **G** = (x_G, y_G)
  x_G = 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798,
  y_G = 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8;
- **n** = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141;
- **h** = 1

[I] National Institute of Standards and Technology

Usando la curva $E : y^2 = x^3 + ax + b$ descritta, l'algoritmo ECDH applicato al protocollo secp256k1 permette ad Alice di calcolare:

- d_A = random(1, n-1) = CA26D640 64C04556 8BC087B7 73137390 D2D412D8 E85E3BBA 11E7A87A F02A460C
- $P_A = (x_A, y_A) = d_A G$
- x_A = 50F1D077 DE0807EB 17E45781 BAD7AC8A 85EFBE48 5DB3988F 546CF300 E50F4750
- y_A = C5C0324A F29FE9CF 4FC0E305 EE406E4D CF1D6166 9877CA62 B16ADE01 2466C0C7

Bob invece calcola:

- d_B = random(1, n-1) = 8840D8F4 08C63786 693A58C3 431489FE 6452C91E 2BD85B5A A982EC04 82B7D965,
- $P_B = (x_B, y_B) = d_B G$
- x_B = D9852BD0 8856BF07 1FF719E4 2B8D4CB1 E82D66F5 B0CB00CF C6B62892 E1A8FFCD
- y_B = C28B6FCA 34E247C8 32EE7C4F 55EC1DD5 150E6800 66BDFD86 1A8A26A7 724CE7E9

Il segreto condiviso sarà il punto di coordinate: $S = (x_S, y_S)$,

- x_S = 11F9892D 57C6EAAD B9DE6475 BBCA2204 01FEAA84 3B7A1DB1 3C44C348 C0FB92F1
- y_S = 89FC9ED0 FF4B1D91 3E4978EF DF23BAD8 947528E0 C4DDD00D C0119149 DD5C08ED

4.5 Firma Digitale

L'idea della firma digitale si basa sul seguente concetto: Alice vuole firmare un messaggio m con la sua chiave privata d_A per dimostrare di essere effettivamente Alice e non un impostore. A questo punto segue Bob, il quale vorrà accertarsi di star parlando effettivamente con Alice, il quale cercherà di validare la firma tramite la chiave pubblica di Alice stessa P_A. La forza della firma digitale è che solamente Alice è in grado di *generare* una firma valida F_A grazie all'uso della sua chiave **privata** e, allo stesso tempo, tutti gli utenti della rete sono in grado di verificarne l'autenticità grazie all'uso della chiave **pubblica** P_A di Alice.
Gli algoritmi che derivano dall'idea della firma digitale si basano sull'hash del messaggio $H = hash(m)$ piuttosto che sul messaggio in chiaro m in modo che il messaggio m resti segreto nonostante esso venga calcolato per la verifica della firma.

4.5.1 Scelta dell'hash

L'algoritmo per produrre l'hash deve esser nell'ottica di una funzione crittograficamente sicura. In data 23 Febbraio 2017, Google ha completato l'esperimento "SHAttered" [19] con il quale si mirava a trovare una collisione sull'algoritmo SHA-1. L'attacco portato avanti da Google è riuscito nell'impresa mediante $9.2 \cdot 10^{18}$ calcoli di SHA-1.
Sebbene ormai deprecato dal NIST nel 2011, molte applicazioni web e browser fanno ancora uso di SHA-1, ad esempio Internet Explorer e GIT.
GIT è un'applicazione che permette a gruppi di persone di lavorare assieme, da remoto, sullo stesso progetto; lo spazio di lavoro è detto *Repository*. Il salvataggio dei dati si basa su "*commit*" generati da SHA-1 rendendo essenzialmente possibile creare due spazi di lavoro con lo stesso commit ma contenuti differenti. La vulnerabilità dell'applicazione risiede nella concreta possibilità che un utente malintenzionato C, a conoscenza di un commit c_A, possa creare un Repository con del codice malevolo (ad esempio una *Backdoor*) ed esser in grado di ottenere lo stesso commit c_A della vittima. A questo punto, scambiare i due repository significa poter attaccare la vittima senza che questa possa rendersene conto.
Famiglie di algoritmi più sicuri per produrre l'hash di un testo sono SHA-2 o SHA-3.

4.5.2 DSA

DSA, acronimo di **D**igital **S**ignature **A**lgorithm, è un algoritmo per la generazione di firme digitali [20] basato sulla scelta di parametri iniziali dell'algoritmo ed il calcolo di due chiavi, una pubblica ed una privata. I parametri in gioco sono:

- **p** numero primo detto *modulo*. Il suo valore è scelto in modo che rientri nell'intervallo $2^{L-1} < p < 2^L$,
  - Il numero L definisce la lunghezza in bit del modulo. La scelta di L non dipende dall'utente, bensì deve esser specificata da enti governativi. I valori accettati sono 1024, 2048 e 3072;
- **q** numero primo, divisore di "$p-1$" e compreso nell'intervallo $2^{N-1} < q < 2^N$,
  - Il numero N definisce la lunghezza in bit di q. Come già affermato per L, il valore di N deve esser specificato da un ente governativo. I valori accettati sono 160, 224 e 256;
- **g** elemento definito nell'intervallo $1 < g < p$ generatore di un sottogruppo $H \subset \mathbb{F}_p$,
  - la cardinalità del sottogruppo deve essere pari a q;
  - il gruppo $\mathbb{F}_p$ è un gruppo moltiplicativo;
- **x** la chiave privata, la scelta di x deve ricadere nell'intervallo $[1, q-1]$;
- **y** la chiave pubblica, viene calcolata come $y = g^x \bmod(p)$;
- **k** un numero segreto, da mantenersi unico per ogni messaggio. La scelta di k deve ricadere nell'intervallo $[1, q-1]$;
- **z** stringa corrispondente ai primi μ bit più a sinistra di H; tuttavia deve esser convertita in un numero intero prima di poter essere utilizzata (come espresso in [20], pagine 19, 68 e 69),
  - $\boldsymbol{\mu} = min(N, h)$ è il valore minimo tra N ed h,

– dato il messaggio M, l'impronta $\boldsymbol{H} = hash(M)$ ha una lunghezza di $\boldsymbol{h}$ bit.

La conoscenza di questi parametri permette di proseguire lo studio dell'algoritmo. La firma F che si ottiene è la coppia:

$$F = (r, s) \iff \begin{cases} r = [g^k \bmod(p)] \bmod(q) \\ s = [k^{-1} \cdot (z + xr)] \bmod(q) \end{cases}$$

Nonostante sia estremamente raro, può succedere che uno dei due parametri di F siano pari a 0. In questi casi si procede ad un secondo calcolo di k e quindi della firma.

Verifica

Ipotizzato che Alice abbia generato correttamente la sua firma F, i parametri che vengono mandati a Bob sono $A = (p, q, g, H, F = (r, s), y)$. Il primo passo per la verifica consiste nel considerare i parametri r ed s: se entrambi si trovano nell'intervallo aperto $(0, q)$ allora Bob ha ricevuto dei parametri corretti e può proseguire nel calcolare i seguenti valori

1. N, lunghezza in bit di q
2. $\mu = min(N, h)$ dove h è la lunghezza dell'impronta $H = hash(M)$
3. $w = s^{-1} \bmod(q)$
4. z corrisponde ai primi μ bit della stringa $H = hash(M)$; va quindi convertito in interger
5. $a = zw \bmod(q)$
6. $b = rw \bmod(q)$
7. $R = [(g^a y^b) \bmod(p)] \bmod(q)$

Se si trova che $R = r$ allora si conclude che la verifica si è conclusa con successo. In caso contrario il messaggio M o la firma F potrebbero esser stati generati erroneamente oppure l'utente malintenzionato C può aver provato a riprodurre F; per tali motivi la firma è da considerarsi non valida.

4.5.3 ECDSA

L'ultimo algoritmo che andiamo ad analizzare è la trasposizione della Firma Digitale nell'ambito delle curve ellittiche. Applicare una firma sulle chiavi è compito dell'ECDSA - **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm.
Alice vuole firmare l'hash $H = hash(m)$ per mezzo della propria chiave privata d. I parametri da utilizzare sono $t = (a, b, p, G, n, h)$. Scelto un algoritmo di hashing sicuro, l'impronta $H = hash(m)$ dovrà essere troncata ad n bit. Tale stringa va convertita in numero intero, ottenendo T. La firma viene quindi generata tramite i seguenti calcoli:

1. $d = random(1, n-1)$ e $P = dG$, le chiavi privata e pubblica di Alice
2. $k = random(1, n-1)$
3. $Q = kG = (x_Q, y_Q)$, un punto della curva
4. $f = k^{-1}(T + x_Q d) \bmod(n)$

Se entrambi i valori di x_Q ed f risultano diversi da zero allora la coppia $F = (x_Q, f)$ costituisce la **firma**. In caso almeno uno dei due parametri sia pari a zero bisogna svolgere nuovamente l'intero algoritmo.
Una volta generata una firma valida si procede al suo invio insieme alla chiave pubblica P tramite la quale viene reso possibile validare la firma F.

Bob, una volta in possesso della firma $F = (x_Q, f)$ di Alice, vuole verificarne l'autenticità. Tramite i parametri pubblici t, la firma F ricevuta, l'hash H, Bob procede come segue:

1. Genera T come la sottostringa di n caratteri presa dall'impronta H
   Calcola quindi
2. $u = f^{-1}T \bmod(n)$;
3. $v = f^{-1}r \bmod(n)$;
4. $R = uG + vP = (x_R, y_R)$.

La firma viene quindi ritenuta valida se l'uguaglianza $x_Q = x_R$ viene rispettata. L'uguaglianza cercata ci permette di dire che $Q = \pm R$. Come verrà dimostrato di seguito, i due punti coincidono.
Dimostriamo la correttezza dell'algoritmo partendo dal punto R e procedendo come segue:

$$
\begin{aligned}
R &= uG + vP, && \text{sostituiamo } P_A = dG \\
&= uG + vdG, && \text{raggruppiamo secondo G} \\
&= (u + vd)G, && \text{sostituiamo } u = f^{-1}T \bmod(n),\ v = f^{-1}x_Q \bmod(n) \\
&= (f^{-1}T + f^{-1}x_Q \cdot d)G \bmod(n), && \text{raggruppiamo secondo } f^{-1} \\
&= f^{-1}(T + x_Q \cdot d)G \bmod(n)
\end{aligned}
$$

Per concludere la dimostrazione dobbiamo considerare $f = k^{-1}(T + x_Q d)$ ed evidenziare $k = f^{-1}(T + x_Q d)$ da sostituire nell'equazione di sopra. Il risultato finale è $R = kG = Q$, valore calcolato in fase di generazione della firma e che quindi verifica la correttezza dell'algoritmo.
Si noti che il segreto k viene usato per calcolare il punto P; quest'ultimo serve poi per generare x_Q. L'unico modo che l'utente malintenzionato C ha di duplicare la firma di Alice è venire a conoscenza di k a partire da x_Q ed f. Risulta evidente quanto sia importante mantenere un alto livello di sicurezza per impedire a C di risolvere facilmente l'ECDLP.

Come suggerito dal NIST nello standard FIPS PUB 186-4 [20] la scelta del parametro k va ripetuta ad ogni firma. Tale numero segreto va per tanto protetto da utilizzi e modifiche da parte di terzi. Non attenersi allo standard costituisce un indebolimento dell'algoritmo, rendendo l'ECDLP un problema ben più semplice da risolvere di quanto dovrebbe essere. Per trattare un esempio concreto possiamo citare il caso della PlayStation3 della Sony [21]: il segreto k scelto non è mai stato rigenerato per ogni gioco ma reso *statico*. Possedere un singolo gioco non permette il calcolo della chiave privata, tuttavia è possibile se si posseggono almeno due giochi originali. Il team a capo di questa scoperta ha evidenziato i passaggi necessari al calcolo della chiave privata.
Potendo rappresentare ciascun gioco con la tripla (T_1, x_Q, f_1) e (T_2, x_Q, f_2). Sapen-

do che $Q_1 = k_1 G$ e $Q_2 = k_2 G$ avere un k statico induce a dire $k_1 = k_2$ e quindi $Q_1 = Q_2 = (x_Q, y_Q)$; per questo motivo si è posto lo stesso numero x_Q in entrambe le triple. Passiamo ad f:

$f = k^{-1}(T + x_Q d) \bmod(n)$ — Scriviamo la differenza tra le due f

$f_1 - f_2 = k^{-1}[T_1 + x_Q d - (T_2 + x_Q d)] \bmod(n)$ — Semplifichiamo $x_Q d$ e $-x_Q d$

$f_1 - f_2 = k^{-1}(T_1 - T_2) \bmod(n)$ — Evidenziamo quindi k

$k = (T_1 - T_2)(f_1 - f_2)^{-1} \bmod(n)$

Siamo quindi giunti a calcolare il valore del segreto k a partire dai due hash troncati T_i e dai loro valori firmati f_i. Tutti i parametri nell'equazione di f sono noti, eccezion fatta per la chiave d. Il calcolo si basa sull'evidenziarla nell'equazione della firma ottenendo

$$d = x_Q^{-1}(fk - T) \bmod(n)$$

Il paradigma di firma utilizzato necessita quindi di una buona entropia dalla quale generare i parametri più sensibili, primo fra tutti è la chiave privata.

Capitolo 5

Sicurezza dell'ECDLP

Nella sicurezza informatica dire che un problema è difficile da risolvere significa che algoritmi e protocolli basati su di esso risultano crittograficamente più sicuri di altri che fanno uso di procedure semplici o prevedibili. Gli algoritmi (ElGamal, RSA, DSA) analizzati nel capitolo 4 sono costruiti sulla base di problemi del logaritmo discreto (DLP), ognuno con lo scopo di proteggere dati sensibili da utenti malintenzionati.
Sappiamo che il DLP viene descritto a partire dalla generica equazione

$$y = a^x \bmod(n)$$

con l'intento di calcolare x dati per noti gli altri valori y, a, n. L'algoritmo più veloce [25, 43] per la risoluzione di questo problema presenta complessità temporale pari a

$$e^{\sqrt[3]{ln(n)\cdot ln(ln(n))^2}} \tag{5.1}$$

Nel 2010 si è difatti riusciti a fattorizzare [44] una chiave RSA da 768 bit, per una lunghezza totale di 232 cifre. Progresso tecnologico e nuove teorie matematiche contribuiscono ad abbassare la difficoltà nel risolvere il DLP facendo emergere il bisogno di nuovi algoritmi che possano garantire l'integrità e la riservatezza delle informazioni in modo efficace. A questo proposito si evidenzierà, nel corso di questo capitolo, come le curve ellittiche offrano quanto richiesto.

5.1 Attacchi all'ECDLP

Questa sezione viene incentrata su differenti algoritmi che permettono di risolvere il problema del logaritmo discreto per le curve ellittiche avendo la conoscenza dei soli parametri pubblici t descritti fino ad ora.
ECDH, ECDSA e tutti gli algoritmi basati su curve ellittiche fondano la loro sicurezza crittografica sull'ECDLP, ovvero: *dati due punti P e Q di una curva ellittica definiti come $Q = kP$, qual è la difficoltà computazionale nel calcolare k?*

5.1.1 Baby Step, Giant Step

L'algoritmo di Shank, il "*Baby Step, Giant Step*" divide il problema del logaritmo discreto in due problemi più semplici. La prima parte dell'algoritmo consiste nel calcolo di $\sqrt{n}$ punti della curva. La seconda parte mira a trovare una combinazione lineare di punti tale da ottenere una corrispondenza tra i punti calcolati nella prima parte. Partiamo dal considerare una generica combinazione lineare secondo la quale il nostro segreto k venga visto come $rs + t$, definendo ogni parametro come numeri interi $r, s, t \in \mathbb{Z}$. In definitiva, il problema ECDLP espresso come $Q = kP$ può esser visto nella forma $Q = (rs + t)P$ e quindi scomposto in

$$Q - rsP = tP \tag{5.2}$$

La forma (5.2) è quella che andiamo ad analizzare.
Imponiamo il valore di s fisso e consideriamo r, t variabili secondo quanto riportato di seguito:

$$\begin{cases} s = \lceil \sqrt{n} \rceil \\ r \in [0, s) \\ t \in [0, s] \end{cases}$$

Come anticipato inizialmente, il primo problema è il Baby Step: calcolare i punti tP al variare di t ovvero $\{\mathcal{O}, P, 2P, \ldots, (s-1)P, sP\}$. Non possiamo dunque pensare di ricorrere ad una Double&Add o alla Montgomery Ladder: bisogna usare t volte la Legge di Gruppo. In aggiunta, bisogna tener traccia dei punti trovati rendendo ne-

cessario salvarli in una "*hash table*"[I] evidenziando entrambe le coordinate del punto ottenuto ed il valore di t utilizzato.
Adesso dobbiamo occuparci del Giant Step: calcolare i punti $R_i = Q - rsP$ al variare di r. L'algoritmo termina quando un R_i corrisponde ad un punto salvato nella hash table. Recuperare il valore di t e di r determina la soluzione dell'ECDLP portandoci a calcolare $rs + t \to k$.

Quali sono i motivi dietro la scelta di $s = \lceil \sqrt{n} \rceil$? La spiegazione si trova osservando l'equazione $Q = rsP + tP$

$$\begin{cases} r = 0 \to Q = rsP + tP = 0P + tP & = tP \\ r = 1 \to Q = sP + tP & = (s+t)P \\ r = 2 \to Q = 2sP + tP & = (2s+t)P \\ \dots & \\ r = s-1 \to Q = (s-1)sP + tP & = (s^2 - s + t)P \end{cases}$$

Facendo ora variare t nell'intervallo $[0, s]$ significa calcolare i punti

$$\begin{cases} r = 0 \to \{\mathcal{O}, P, 2P, \dots, sP\} \\ r = 1 \to \{sP, (s+1)P, \dots, 2sP\} \\ \dots \\ r = s-1 \to \{(s^2 - s)P, (s^2 - s + 1)P, \dots, (s^2 - s + s)P\} \end{cases}$$

L'ultimo punto calcolato è $(s^2 - s + s)P = (s^2)P = nP$ per definizione di s.
Le operazione che hanno reso possibile questo risultato sono esattamente s moltiplicazioni scalari per trovare tutti i tP ed un massimo di altre s operazioni per il calcolo di $Q - rsP$ fino ad avvenuta collisione con tP.
Dover salvare s punti nella hash table che comporta una complessità spaziale di $O(s) = O(\sqrt{n})$. Possiamo evitare di salvare il punto all'infinito ed il punto noto P determinando che ora la tabella comprenderà $\lceil \sqrt{n} \rceil - 2$ record, un risultato insignificante per grandi valori di n. Esiste un'altra possibilità per ridurre i dati da salvare: sfruttare la simmetria della curva. Per ogni punto $tP = (x, y)$ andiamo a

[I] Matrice data da array associativi. Nel nostro caso le chiavi di tali array sono i valori computati di t; i valori degli array sono le coordinate di ciascun punto ottenuto.

salvarne solo la coordinata x, in tal modo dimezziamo lo spazio complessivo utilizzato. Al tempo stesso ciascun record della tabella evidenzierà non solo l'ascissa dei punti $+tP$ ma anche quella dei loro simmetrici, permettendo di trascurare un'altra metà dei punti ottenibili. Trovare la molteplicità di un punto simmetrico è possibile basandosi su quanto esposto nel capitolo 3.3.1 nelle Considerazioni Personali 2. La complessità spaziale così definita risulta

$$O\Big(\Big\lceil \frac{s}{4} \Big\rceil\Big) = O\Big(\Big\lceil \frac{\lceil \sqrt{n} \rceil}{4} \Big\rceil\Big) \tag{5.3}$$

Esempio: la curva P-192 (standardizzata dal NIST in [22]) presenta cardinalità, espressa in notazione esadecimale, $n = FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ 99DEF836\ 146BC9B1\ B4D22831$; il numero di operazioni per calcolare tutti i punti tP sarà $s = \sqrt{n} \approx 7.9 \cdot 10^{28}$. Ciascuna coordinata è costituita da 48 bytes portando ciascun punto a pesare 96 bytes di memoria per esser salvato. L'intera tabella di hash richiederà $96 \cdot 7.9 \cdot 10^{28} = 7.5 \cdot 10^{30}$ bytes di memoria per l'algoritmo base (5.2) e circa $1.8 \cdot 10^{30}$ bytes mediante la complessità (5.3). In entrambi i casi abbiamo valori troppo grandi da memorizzare.

5.1.2 La ρ di Pollard

Un altro algoritmo per la risoluzione dell'ECDLP è la ρ di Pollard. Quella che verrà mostrata di seguito è una variante del classico algoritmo di Pollard introdotto nel 1975 (nel quale si parlava di fattorizzare un numero primo), ora applicata alla teoria delle curve ellittiche come riportato in [29]. L'approccio che stiamo per mostrare, riprende l'idea della combinazione lineare di punti vista con il BSGS ma necessita di una complessità spaziale nettamente inferiore.
Partendo dall'equazione $Q = kP$ determiniamo $Q = aP + bQ$, $kP = AP + BQ$ dove a, b, A, B sono numeri interi, in modulo n. Sull'esempio della curva $y^2 = x^3 + 2x + 2$ mod(17) riportato nel capitolo (3.3.1), supponiamo essere $Q = 3P = (10, 6)$. Una combinazione lineare di punti per ottenere Q può essere $a = 6$, $b = -1$ secondo la quale otteniamo $Q = aP + bQ = 6P - Q$ dove $6P = (16, 13)$ e $-Q = -3P = (10, 11)$. Computare la legge di gruppo tra questi due punti conduce al valore cercato $Q = (10, 6)$ corrispondente a $3P$. Essendo vero che $k = 3$ abbiamo correttamente risolto il problema. In formule abbiamo che:

$$\text{Dati i parametri } a,\ b,\ A,\ B \in s$$
$$\exists (a,b) \neq (A,B) \mid aP + bQ = AP + BQ.$$

dove con s intendiamo una sequenza finita di parametri (verrà meglio definita a breve).
L'algoritmo lo possiamo dunque esporre partendo dal calcolo di k nell'equazione del logaritmo discreto per le curve ellittiche:

$Q = kP$	Applichiamo la combinazione lineare
$aP + bQ = AP + BQ$	Sostituiamo il punto $Q = kP$
$aP + bkP = AP + BkP$	Portiamo a sinistra AP ed a destra bkP
$aP - AP = BkP - bkP$	Raccogliamo il punto P
$(a - A)P = (Bk - bk)P$	Raccogliamo la k a destra
$(a - A)P = (B - b)kP$	Semplifichiamo P ed aggiungiamo il mod(n)
$a - A = (B - b)k \bmod(n)$	Evidenziamo la k
$k = (a - A)(B - b)^{-1} \bmod(n)$	

Si è dovuto aggiungere il modulo n in modo da ottenere valori accettabili di k all'interno del sottogruppo in cui è definita la curva.
Per procedere con l'algoritmo abbiamo bisogno di generare una **sequenza finita** di j valori. La sequenza detta s viene determinata da coppie dei due parametri a, b generati in modo pseudo-casuale. La struttura di s sarà quindi

$$s = \{(a_1, b_1), (a_2, b_2), \ldots, (a_j, b_j)\}$$

Per ogni coppia di s calcoliamo il corrispondente punto $R_i = a_iP + b_iQ$ e determiniamo una seconda sequenza, che chiameremo S, data dai punti R_i:

$$S = \{R_1, R_2, \ldots, R_j\}$$

Essendo i punti P e Q due elementi di un gruppo ciclico, la sequenza S delle loro combinazioni lineari R_i risulta essere ciclica. La ρ di Pollard si basa sul trovare due coppie diverse di s che determinano due punti R_i uguali in S.
La ρ di Pollard prosegue mediante algoritmi di ricerca dei cicli. Uno di questi è l'algoritmo di Floyd, "*la Lepre e la Tartaruga*", che data la sequenza S permette di

leggere ogni suo elemento facendo uso di due velocità differenti. Chiamiamo T la tartaruga, questa ha il compito di leggere ogni singolo punto R_i; chiamiamo invece L la lepre, questa leggerà i punti R_i alternati. Confrontando i punti trovati da T ed L ad ogni loro passo arriviamo a trovare due punti R_T e R_L con le stesse coordinate ma generati da diverse coppie di s. Terminiamo l'algoritmo con il calcolo del segreto k per mezzo della $k = (a - A)(B - b)^{-1} \bmod(n)$.
Come riportato da Certicom in [24] la ρ di Pollard è l'algoritmo più veloce nel risolvere l'ECDLP mediante una complessità temporale

$$O\Big(\sqrt{\frac{\pi n}{2}}\Big) \tag{5.4}$$

Una sua versione parallelizzata su 2600 computer è stata utilizzata da Chris Monico nel 2004 [23] per una sfida lanciata dalla Certicom. La sfida consisteva nel riuscire a trovare il segreto k di lunghezza pari a 109 bit su una determinata curva ellittica. L'ordine di tale curva corrisponde a $n = 2^{109}$ ed il tempo richiesto per risolvere l'ECDLP è stato pari a 17 mesi di computazioni.
Mostriamo, in proporzione, quanto tempo sarebbe stato necessario al professor Monico ed il suo team per risolvere lo stesso problema ma su differenti curve ellittiche. Detto t il coefficiente di tempo impiegato, ossia 17 mesi, detta B la lunghezza in bit della curva considerata; il tempo complessivo T lo assumiamo corrispondere a:

$$T = t \cdot \frac{\sqrt{\frac{\pi 2^B}{2}}}{\sqrt{\frac{\pi n}{2}}} = t \cdot \frac{\sqrt{2^B}}{\sqrt{2^{109}}} = t \cdot \sqrt{2^{B-109}}$$

Vediamo qualche esempio con alcune lunghezze B standard per le curve ellittiche:

B	T (mesi)
160	$8 \cdot 10^8$
224	$3.4 \cdot 10^{18}$
256	$2.2 \cdot 10^{23}$
384	$4.1 \cdot 10^{42}$
512	$4.5 \cdot 10^{60}$

Sebbene sia una semplice proporzione i tempi di riuscita sono troppo elevati.
Eseguire la ρ di Pollard su M processori è un compito che può essere svolto in due

modi differenti, apportando diversi tempi di risoluzione. Un primo approccio consiste nel calcolare M sequenze S_m ed assegnarne una ad ogni processore. Ognuno di questi dovrà eseguire l'algoritmo di Pollard riducendo il costo computazionale finale di sole $\sqrt{M}$ operazioni. Infatti, ogni processore avrà la medesima probabilità degli altri $M-1$ di trovare una collisione [49, 50], il miglioramento introdotto è sublineare. Per diminuire linearmente il costo computazionale bisogna assegnare ad ogni processore la medesima sequenza S ma ognuno dovrà partire da punti iniziali P_m diversi.
Descriviamo infine le complessità computazionali che abbiamo ottenuto con il metodo base e le due differenti versioni parallelizzate:

ρ	Standard	Versione 1	Versione 2
	$O\left(\sqrt{\frac{\pi n}{2}}\right)$	$O\left(\sqrt{\frac{\pi n}{2M}}\right)$	$O\left(\frac{1}{M}\sqrt{\frac{\pi n}{2}}\right)$

5.1.3 La λ di Pollard

Un secondo algoritmo ideato da Pollard è il cosidetto λ o "*metodo del canguro*" [30]. La sua applicazione è valida qualora sia possibile restringere in un intervallo $[min, max]$ i valori in cui si trova il segreto k.
Si parte dal considerare due punti: uno noto, ad esempio il punto generatore G, ed uno casuale calcolato al momento. Saltare da un punto ad un altro è un compito dato a due canguri. Il comportamento di ciascun canguro viene descritto da una funzione di iterazione $x_{i+1} = f(x_i, d_i)$ espressa come

$$\begin{aligned} f(x_i, d_i) &= x_i \cdot c^{d_i \cdot x_i \bmod(M)} \bmod(n) \\ c \mid x_0 &= c^b \bmod(n) \\ d &= \text{ distanza di salto} \end{aligned}$$

La distanza di salto d viene generalmente identificata [31] da un elemento dell'insieme finito S delle potenze di 2, per cui $S = \{2^0, 2^1, \ldots, 2^M\}$. La scelta di un elemento d_i di S è casuale ad ogni salto (iterazione).
Il canguro addestrato T inizia la sua serie di salti tramite la funzione $f(t_i, d_i)$,

partendo dal punto iniziale t_0. Concludiamo le iterazioni dopo j salti ottenendo:

$$t_j = c^{b+D_T(j-1)} \bmod(n)$$
$$D_T(j) = \sum_{i=0}^{j} d_i \cdot t_i \bmod(M)$$

Parte ora il secondo canguro, quello selvatico W, da un punto iniziale, casuale $w_0 = c^k$. Applichiamo la medesima funzione di iterazione per calcolare i suoi salti successivi e memorizziamo le coppie (w_i, d_i). Per ogni salto di W controlliamo se sia arrivato su un punto già trovato da T. Indichiamo con α il numero di salti necessari a T per arrivare al punto di collisione; con β il numero di salti di W per raggiungere lo stesso punto. Il punto di collisione identifica $w_\beta = t_\alpha$ dai quali, espressi nelle loro forme esponenziali, otteniamo

$$k = b + D_T(\alpha - 1) - D_W(\beta - 1) \tag{5.5}$$

ovvero il coefficiente che risolve l'ECDLP [29].

Anche questo algoritmo può venir parallelizzato su molteplici processori. Presi M processori, ognuno ricopre il ruolo del canguro addestrati. Applichiamo l'algoritmo appena visto con delle piccole modifiche: invece di un punto iniziale G ci serviranno M punti distinti per i canguri addestrati T; similmente avremo bisogno di M punti casuali dai quali far partire i canguri selvatici W. L'algoritmo termina quando due canguri di natura diversa saltano sullo stesso punto. Il calcolo del coefficiente k segue la stessa formulazione di (5.5).
L'algoritmo della λ di Pollard è probabilistico, riuscire a trovare k dipende dalla scelta per il punto iniziale del canguro selvatico. Se l'algoritmo dovesse fallire, lo si può ripetere cambiando il punto iniziale w_0. Per concludere analizziamo i costi computazionali delle due versioni:

Versione Base	Versione Parallelizzata
$O\left(\sqrt{\frac{\pi n}{2}}\right)$	$O\left(\frac{1}{M}\sqrt{\frac{\pi n}{2}}\right)$

Entrambi rispecchiano l'analoga versione della ρ di Pollard, tuttavia bisogna conoscere un intervallo molto ristretto di valori nel quale cercare k, un requisito difficil-

mente noto. Per questo motivo l'algoritmo della ρ viene ritenuto più efficiente della λ.

5.1.4 Poligh-Hellman

L'algoritmo di Pohlig-Hellman [33] si adatta molto bene a curve di ordine n non primo, portando alla risoluzione dell'ECDLP in breve tempo. Per trovare il coefficiente k in $Q = kP$, scomponiamo n nei suoi fattori primi:

$$n = \prod_{i=1}^{r} p_i^{e_i}$$

Dobbiamo definire r coefficienti k_i a partire dalla rappresentazione polinomiale di grado $e_i - 1$, a coefficienti naturali $0 \leq a_i \leq p_i - 1$:

$$k_i = a_0 + a_1 p_i + a_2 p_i^2 + \ldots + a_{e_i-1} p_i^{e_i-1} \tag{5.6}$$

In aggiunta al polinomio, vanno calcolati i seguenti parametri, validi per ciascun k_i: $P_0 = \frac{n}{p_i} P$; $Q_0 = \frac{n}{p_i} Q$; l'insieme di punti $S = \{P_0, 2P_0, \ldots, (p_i - 1)P_0\}$. Tramite i punti di S possiamo determinare il coefficiente a_0 considerando che $Q_0 = a_0 P_0$ e cercandolo all'interno dell'insieme.
Infatti, dato che per il punto P_0 vale $p_i P_0 = p_i \cdot \frac{n}{p_i} P_0 = nP = \mathcal{O}$, definiamo:

$$\begin{aligned}
Q_0 &= \frac{n}{p_i} Q \\
&= \frac{n}{p_i} (k_i P) \\
&= \frac{n}{p_i} (a_0 + a_1 p_i + a_2 p_i^2 + \ldots + a_{e_i-1} p_i^{e_i-1}) P \\
&= \frac{n}{p_i} (a_0) P + (a_1 + a_2 p_i + \ldots + a_{e_i-1} p_i^{e_i-2}) nP \\
&= a_0 P_0 + (a_1 + a_2 p_i + \ldots + a_{e_i-1} p_i^{e_i-2}) \mathcal{O} \\
&= a_0 P_0
\end{aligned}$$

I restanti coefficienti a_j vengono calcolati similmente p_i^{j+1} e $Q_k = (n/p_i^{j+1})Q$. Una volta calcolati tutti i coefficienti risolviamo il polinomio (5.6) per trovare il parametro k_i. Bisogna allora ripetere il procedimento illustrato r volte, fino a computare tutti

i k_i e quindi arrivare a scrivere il sistema:

$$\begin{cases} k = k_1 \bmod(p_1^{e_1}) \\ k = k_2 \bmod(p_2^{e_2}) \\ \dots \\ k = k_r \bmod(p_r^{e_r}) \end{cases} \tag{5.7}$$

Risolvere questo sistema significa risolvere l'ECDLP. Il Teorema Cinese del Resto assicura che la soluzione del sistema sia unica. Il teorema è il seguente: dato un sistema del tipo (5.7) in cui i moduli $p_i^{e_i}$ siano tutti coprimi tra di loro ed i parametri k_i siano numeri interi; la soluzione $n = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$ è unica. Chiamato $N_i = n/p_i^{e_i}$, affermiamo che il Massimo Comun Divisore $MCD(N_i, p_i^{e_i}) = 1$ per ogni i. Essendo allora vero che $a_i N_i = 1 \bmod(p_i^{e_i})$, la soluzione sarà la stessa per ogni i, quindi unica, e vale

$$k = \sum_{i=1}^{r} a_i N_i k_i$$

L'algoritmo di Pohlig-Hellman è efficiente quando si considerano fattori primi "piccoli". Per una curva prima (ovvero la sua cardinalità è un numero primo) l'algoritmo avrebbe un singolo $k = p_i = n$, con scomposizione in $k = a_0$ ed inducendo a calcolare un insieme S di tutti gli n punti. Il risultato sarebbe lo stesso di una ricerca completa di tutti i punti della curva, rendendo l'algoritmo inefficiente quanto una ricerca esaustiva. Similmente accade quando la cardinalità n presenta scomposizione in numeri primi molto grandi: dover computare l'insieme S rende inefficiente o computazionalmente impraticabile l'algoritmo.

Infine, qualora le condizioni per l'utilizzo dell'algoritmo siano favorevoli, il costo complessivo di operazioni si determina essere [34]:

$$O\Big(\sum_{i=1}^{r} e_i(\log(n) + \sqrt{p_i})\Big)$$

5.1.5 Side Channel

Gli attacchi detti "*Side Channel*" non sono intrusivi, non necessitano di raccogliere i dati scambiati tra due interlocutori per determinare la chiave privata K^- in gio-

co. Questi attacchi mirano a calcolare K^- in base a fattori fisici quali: tempo di elaborazione per l'algoritmo e consumo di potenza del processore. La vulnerabilità maggiore nella crittografia ellittica risiede in come viene calcolata la chiave pubblica, ossia quante volte venga usata la Legge di Gruppo, quante volte la Point Doubling ed "*ascoltando*" in che ordine queste vengano elaborate. Come visto nel capitolo 3.2.1 la Double&Add usa una determinata sequenza di operazioni per giungere al risultato finale. Detta k la nostra chiave privata, il punto $Q = kG$ rappresenta la chiave pubblica. Un attacco Side Channel si applica proprio nel momento in cui si calcola Q, misurando la potenza assorbita ad ogni operazione fatta. Applicare un attacco di questo tipo è possibile avendo a disposizione un calcolatore munito di oscilloscopio. Unendo i dati di durata e potenza assorbita raccolti, si ricostruisce l'andamento della moltiplicazione scalare. Far corrispondere ad ogni variazione la relativa operazione (Point Addition o Point Doubling) significa trovare la sequenza di operazioni che portano a calcolare Q e, di conseguenza, il nostro segreto k.
Il metodo più semplice per proteggersi da questi attacchi passivi consiste nel ricorrere alla Montgomery Ladder. Come visto nella sua implementazione, questo metodo computa sempre una addizione ed una moltiplicazione per ogni cifra binaria di k, permettendo tempi ed assorbimenti di potenza costanti. Senza poter distinguere quale operazione viene adoperata ad ogni passo, si conclude che un attacco Side Channel non sia applicabile a questo algoritmo.

5.1.6 Considerazioni finali

Come visto finora, esistono diversi attacchi che mirano a risolvere l'ECDLP. Ogni algoritmo si fonda su principi matematici differenti o su curve ellittiche aventi specifiche proprietà. Il BSGS non presenta facile applicazione per via del suo costo spaziale; l'algoritmo di Poligh-Hellman richiede curve supersingolari, non trovando alcuna applicazione tra le curve standardizzate dal NIST: tutte le curve promosse sono curve prime, il cui ordine minimo ha lunghezza pari 192 bit. Test di primalità si trovano nello standard [20] a dimostrazione del fatto che le curve utilizzate sono assolutamente sicure contro questo tipo di attacco.
Una seconda nota da evidenziare è che tali curve hanno cardinalità $\#E$ diversa dalla caratteristica di campo p; pertanto non possono esser definite come Curve Anoma-

le. Esistono algoritmi per la risoluzione dell'ECDLP su curve anomale ma, come affermato dallo stesso ideatore Nigel Smart in [36], si tratta di "*metodi di interesse teorico piuttosto che pratico*" e non troverebbero applicazione nelle curve del NIST. Gli algoritmi di Pollard restano i più veloci a nostra disposizione sebbene non presentino un tempo di calcolo sub-esponenziale, rendendoli inefficaci contro le curve di grandi ordini n. In aggiunta, la λ di Pollard richiede un intervallo ristretto sul quale operare: non conoscere l'intervallo esclude questo algoritmo da quelli applicabili contro l'ECDLP.
È possibile velocizzare gli algoritmi visti, o aumentarne le probabilità di riuscita, facendo affidamento alle formule (3.8) e (3.10) mediante le quali risolviamo l'ECDLP anche quando arriviamo a calcolare $-Q$, il punto simmetrico alla nostra chiave pubblica. Gli algoritmi di Pollard avrebbero due punti sui quali fermarsi raddoppiando le probabilità di riuscita. L'algoritmo di Poligh-Hellman non viene influenzato da queste formule, infatti non viene modificato l'ordine n sul quale esso si basa. Per lo stesso motivo l'attacco di Smart resta un esercizio teorico e l'hash table del BSGS non subisce miglioramenti sensibili.
In ultimo, attacchi passivi come i Side Channel trovano scarsa applicazione se le moltiplicazioni scalari vengono effettuate ricorrendo alla Montgomery Ladder e non vengono influenzati dalle formule sulle quali si è appena discusso.

5.2 Livelli di Sicurezza

Dall'analisi condotta precedentemente in questo capitolo si è arrivati a mostrare come gli algoritmi più veloci per risolvere l'ECDLP siano gli algoritmi di Pollard. Volendo risolvere il problema dell'ECDLP, se l'algoritmo più veloce a noi noto richiede 2^k operazioni, diciamo che il problema presenta un **livello di sicurezza k**.
Confrontiamo la (5.1) per risolvere il DLP e la (5.4) per l'ECDLP, il metodo più oneroso è il secondo (figura 5.1):

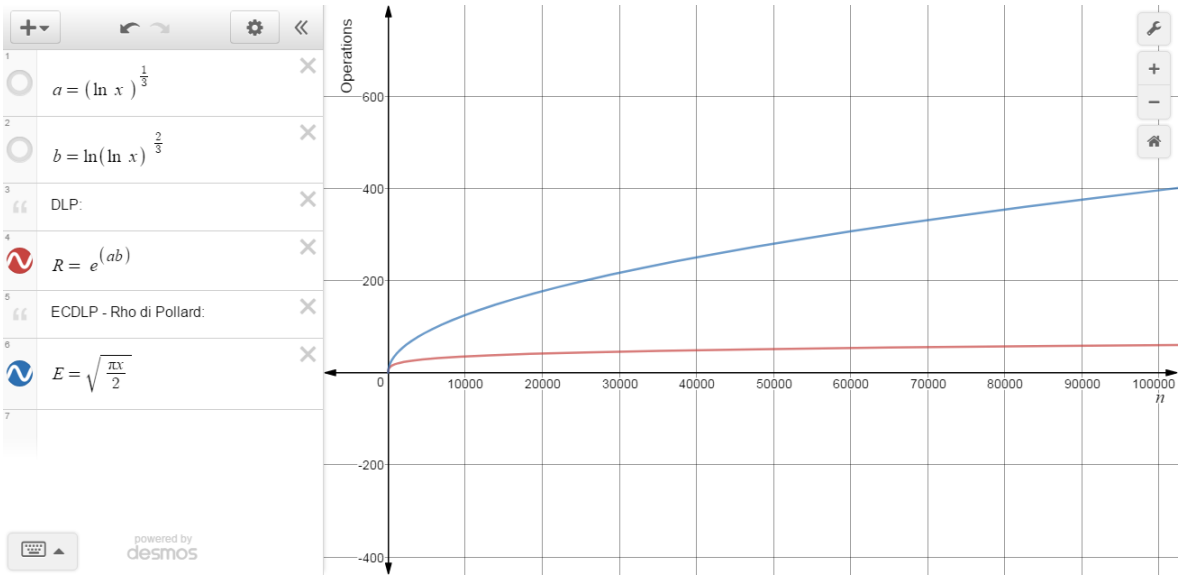


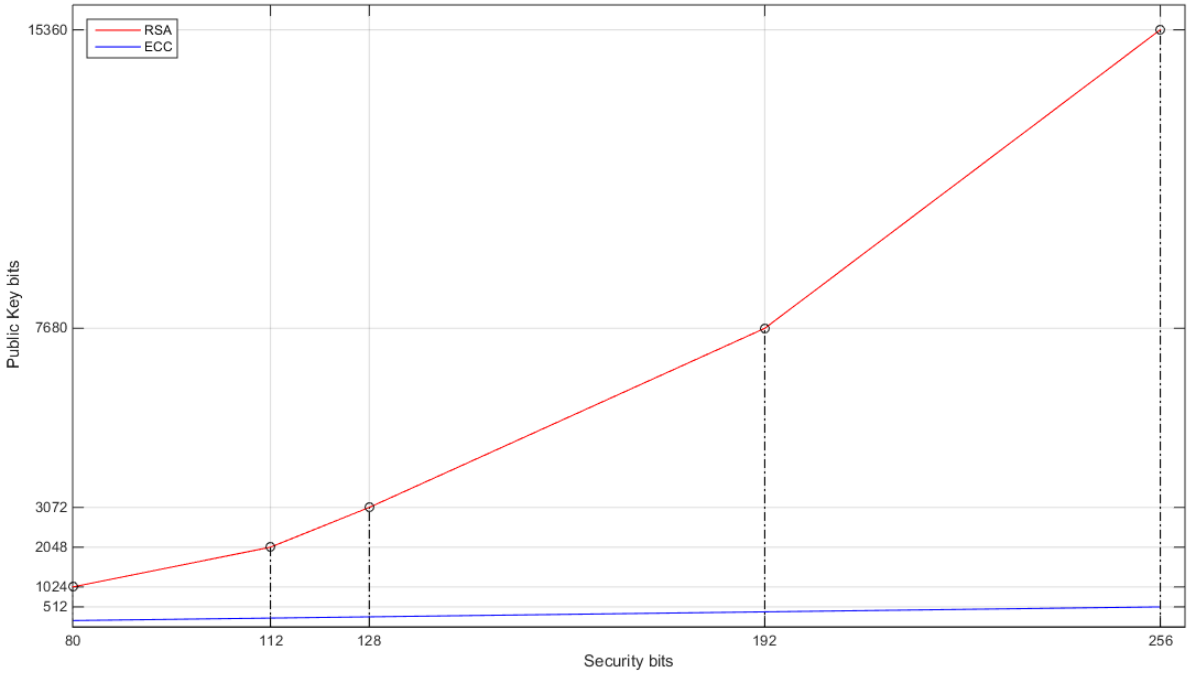
Figura 5.1: Confronto delle complessità computazionali per DLP e ECDLP

La figura mostra quanto più complicato sia risolvere il logaritmo discreto su una curva ellittica anche con piccoli valori di n, confermando quanto affermato da Certicom in [24]. Per avere un'idea di un ordine n accettabile si fa riferimento alla pubblicazione da parte del NIST in [27] in cui si evidenzia come la lunghezza di n debba essere almeno di 2048 bit nei protocolli di accordo e scambio di chiave che facciano uso di RSA. Di conseguenza, tutte le chiavi RSA con lunghezza inferiore ai 2048 bit non sono permesse. Tuttavia una chiave simile presenta $\lceil log_{10}(2^{2048}) \rceil = 617$ cifre, di gran lunga superiore alle 6 cifre mostrate in figura (5.1).

Mettiamo a confronto la sicurezza di algoritmi basati su RSA, curve ellittiche e crittografia simmetrica come fatto in [28]. Nella tabella (5.2) si evidenzia il livello di sicurezza di ciascun algoritmo, ossia la lunghezza di ciascuna chiave:

Livello di Sicurezza	DSA K^-, K^+	RSA	ECC	ECC/RSA
80	160, 1024	1024	160	15%
112	224, 2048	2048	224	10%
128	256, 3072	3072	256	8%
192	384, 7680	7680	384	5%
256	512, 15360	15360	512	3%

Gli algoritmi della crittografia ellittica (ECC) mostrano identico livello di sicurezza della chiave *privata* usata in DSA. Questi, inoltre, richiedono esattamente il doppio dei bit per eguagliare una gli algoritmi della crittografia simmetrica. Il motivo di questo rapporto deriva dal fatto che ECC utilizza due chiavi crittografiche mentre la crittografia simmetrica solo una. L'ultima colonna "ECC/RSA" rappresenta il rapporto in percentuale delle chiavi usate nei due metodi. Maggiore è la sicurezza offerta, minore è la chiave ellittica in rapporto ad RSA.
Sui dati della tabella (5.2) grafichiamo le chiavi per RSA ed ECC al fine di mantenere uno stesso livello di sicurezza:



Vogliamo dunque mostrare un ultimo confronto tra RSA ed ECC: a parità di

livello di sicurezza, mostriamo il numero di operazioni necessarie a risolvere il DLP e l'ECDLP mediante gli attacchi più efficaci considerati. Chiamiamo "L" il relativo livello di sicurezza al quale facciamo riferimento. Per RSA il miglior algoritmo risolutivo necessita di $e^{\sqrt[3]{ln(n)\cdot ln(ln(n))^2}}$ operazioni; per ECC abbiamo la ρ di Pollard con $\sqrt{\frac{\pi n}{2}}$ operazioni. Il valore detto n rappresenta il numero 2^b ovvero il numero che otteniamo mediante chiavi da b bit. Nella colonna ECC/RSA si mostra un rapporto tra le operazioni necessarie ad entrambi gli algoritmi per risolvere il problema del logaritmo discreto.

Prendiamo in considerazione anche il tempo di elaborazione dell'algoritmo: un anno MIPS corrisponde al numero di operazioni che vengono svolte in 365 giorni di attività da un singolo processore che lavora alla frequenza di 1 milione di operazioni al secondo. Tale numero corrisponde a $MIPS = 3.1 \cdot 10^{13}$ operazioni totali. Gli anni MIPS necessari per il calcolo x operazioni sono calcolati come il rapporto $x/MIPS$, vedasi le colonne *MIPS RSA* e *MIPS ECC*. La colonna *MIPS ECC/RSA* non viene mostrata: passare da un valore della colonna *ECC* al corrispettivo *MIPS ECC* significa dividere secondo il coefficiente *MIPS*, lo stesso concetto si applica per *RSA* concludendo che il rapporto *ECC/RSA* corrisponda esattamente a *MIPS ECC/RSA*.

	Operazioni			Anni MIPS	
L	RSA	ECC	ECC/RSA	RSA	ECC
80	$3.8 \cdot 10^{13}$	$1.5 \cdot 10^{24}$	$3.9 \cdot 10^{10}$	1.2	$4.7 \cdot 10^{10}$
112	$1.9 \cdot 10^{18}$	$6.5 \cdot 10^{33}$	$3.4 \cdot 10^{15}$	$6.0 \cdot 10^{4}$	$2.0 \cdot 10^{20}$
128	$5.2 \cdot 10^{21}$	$4.3 \cdot 10^{38}$	$8.2 \cdot 10^{16}$	$1.6 \cdot 10^{8}$	$1.3 \cdot 10^{25}$
192	$6.0 \cdot 10^{31}$	$7.9 \cdot 10^{57}$	$1.3 \cdot 10^{26}$	$1.9 \cdot 10^{18}$	$2.5 \cdot 10^{44}$
256	$1.4 \cdot 10^{42}$	$1.5 \cdot 10^{77}$	$1.0 \cdot 10^{35}$	$4.4 \cdot 10^{28}$	$4.7 \cdot 10^{63}$

Aver scelto l'algoritmo base della ρ di Pollard non cambia molto la situazione rispetto la sua versione parallelizzata: colmare la differenza di operazioni tra i due problemi richiede tanti processori quanto il rapporto ECC/RSA relativo alla colonna Operazioni. Solo per il livello minimo, 80 bit, si dovrebbero considerare circa 39 miliardi di processori attivi simultaneamente.

Quanto mostrato nella tabella costituisce un ulteriore incentivo verso la scelta della crittografia ellittica. Il livello medio di sicurezza, 128 bit, comporta dapprima un impiego di memoria nettamente inferiore in ECC rispetto RSA (solo l'8% della lun-

ghezza in bit) e, al tempo stesso, le operazioni da computare per risolvere l'ECDLP risultano $8.2 \cdot 10^{16}$ volte maggiori in confronto con il DLP. Il tempo di calcolo per rompere i due logaritmi discreti è notevole in entrambi i casi ma non possiamo fare affidamento su un tempo di $1.6 \cdot 10^8$ (RSA, 128 bit) per determinare tale algoritmo come *crittograficamente sicuro* per gli anni futuri. Compatibilità e retrocompatibilità con dispositivi in uso in tutto il mondo sono fattori fondamentali nella scelta di un buon algoritmo di crittografia. L'importanza di mantenere chiavi "piccole" è fondamentale per mantenere RSA efficiente: trovare e moltiplicare due numeri primi molto grandi può causare una latenza eccessiva sui dispositivi meno performanti. Il risultato è la deprecazione di RSA [27] per il 31 Dicembre 2017.
A confronto, chiavi più piccole per ECC consentono di velocizzare notevolmente i calcoli di generazione per la chiave privata: si fa infatti uso di un numero casuale d scelto in un ampissimo intervallo. Ricordando che la chiave privata d per algoritmi ellittici va presa nell'intervallo $[1, n-1]$ diamo un'idea di quanto ampia sia la scelta. Nello standard [20] la NIST raccomanda l'utilizzo di cinque curve diverse, ognuna per un differente livello di sicurezza. Tali curve sono denominate P_x dove x determina la più grande potenza di 2 utilizzata nell'esprimere la cardinalità del campo primo $\mathbb{F}_p$:

$$
\begin{aligned}
P_{192} &\rightarrow p = 2^{192} - 2^{64} - 1 \\
P_{224} &\rightarrow p = 2^{224} - 2^{96} + 1 \\
P_{256} &\rightarrow p = 2^{256} - 2^{224} + 2^{96} - 1 \\
P_{384} &\rightarrow p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
P_{521} &\rightarrow p = 2^{521} - 1
\end{aligned}
$$

La curva standardizzata P_{192} ha equazione $y^2 = x^3 - 3x + b$ [22] con cardinalità

$$n = 6277101735386680763835789423176059013767194773182842284081$$

Ne consegue che scelta della chiave privata, per la curva più "*debole*" crittograficamente parlando, ricade nell'ampissimo intervallo $[1, 6.2 \cdot 10^{58}]$.

Appurato ciò, nuovi algoritmi fanno uso di un numero detto **Seed** S in base al quale vengono generati i coefficienti a e b per la curva. I seed sono generalmente

identificati come "*nothing up my sleeve*": le prime cifre decimali della funzione seno applicata a numeri interi; le cifre iniziali del π; il numero di Nepero e; il rapporto aureo e così via. Ciò che li rende numeri "*crittograficamente sicuri*" è potervi dare una giustificazione matematica.
Applicazioni del Seed si trovano nelle stesse curve standardizzate dal NIST; sebbene la presenza del coefficiente fisso $a = -3$, il parametro b viene calcolato come segue:

- si sceglie un seed s di lunghezza pari a 160 bit
- si calcola c =SHA1(s) basato ovvero sull'output SHA1 del seed
- si calcola $b = \sqrt{-\frac{27}{c}} \bmod(p)$, detto p l'ordine del campo $\mathbb{F}_p$

Le curve così costruite prendono la denominazione **Pseudo-Casuali**.

5.3 Costruzione di Curve Ellittiche sicure

I parametri che abbiamo utilizzato per descrivere gli algoritmi di crittografia e firma digitale mediante curve ellittiche fanno uso dei parametri pubblici

$$t = (a, b, p, G, n, h)$$

con i quali si intendono i parametri (a, b) della curva definita su di un campo primo $\mathbb{F}_p$. Inoltre si è detto come, scelto un punto generatore G, si possa identificare un sottogruppo di cardinalità n e cofattore h tali per cui valga $\#E = nh$.
Vogliamo ora determinare la scelta di tali parametri in accordo con i vari livelli di sicurezza $L \in \{80, 112, 128, 192, 256\}$.

1. Per soddisfare la condizione di curva liscia dobbiamo imporre che $4a^3+27b^2 \neq 0 \bmod(p)$;
2. per tali coefficienti la curva deve avere cardinalità $\#E \neq p$ per evitare che sia detta Anomala e suscettibile all'attacco di Smart (capitolo 5.1.6);
3. l'ordine n e la cardinalità p devono essere tali che, per ogni intero B nell'intervallo $[1, 100)$, valga la relazione $p^B \neq 1 \bmod(n)$;

4. per essere resitente contro l'algoritmo di Pohlig-Hellman (capitolo 5.1.4), l'ordine n deve essere un grande numero primo. Mantenere vera tale affermazione implica che la relazione $\#E = nh$ sia verificata per un grande n ed un, quanto possibile, piccolo cofattore h. Per questo motivo imponiamo la condizione $h \leq 2^{L/8}$, dove L rappresenta il livello di sicurezza per cui stiamo generando i parametri. Quanto più elevato il livello di sicurezza, tanto più piccolo dovrà essere il cofattore in rapporto ad n;

5. infine terminiamo imponendo una condizione sull'ordine n: i più grandi fattori primi di $n-1$ ed $n+1$ detti v, w, devono verificare le condizioni $log_n(v) > 19/20$ e $log_n(w) > 19/20$

Le condizioni esposte sono state prese in accordo alla ricerca della Certicom [37]. D'altronde può verificarsi che nonostante Alice abbia generato in modo sicuro la curva, Bob voglia accertarsi dei parametri scelti. La validazione avviene come descritto di seguito:

1. la cardinalità p del campo deve rispettare la sicurezza crittografica attribuita alla curva, percui bisogna avere

$$\lceil log_2(p) \rceil = \begin{cases} 192 & \text{se } L = 80, \\ 2L & \text{se } 80 < L < 256, \\ 521 & \text{se } L = 256, \end{cases}$$

   inoltre, come definito durante la generazione dei parametri, deve valere

$$p^B \neq 1 \bmod(n),\ \forall B \in [1, 100)$$

2. i parametri (a, b) devono appartenere al campo $\mathbb{F}_p$ e verificare che la curva sia liscia, ossia $4a^3 + 27b^2 \neq 0 \bmod(p)$;

3. la cardinalità n deve corrispondere ad un numero primo, diverso da p;

4. le coordinate del generatore $G = (x_G, y_G)$ devono appartenere sia al campo $\mathbb{F}_p$ e sia alla curva verificando che valga $y_G^2 = x_G^3 + ax_G + b \bmod(p)$. Un ulteriore controllo mira a determinare se l'ordine di G sia effettivamente n cercando di verificare l'uguaglianza $nG = \mathcal{O}$;

5. per il cofattore deve valere la condizione imposta durante la generazione dei parametri $h < 2^{L/8}$;

6. Infine bisogna verificare che la cardinalità verifichi il teorema di Hasse per cui

$$\lceil\, p + 1 - 2\sqrt{p}\,\rceil \leq \#E \leq \lfloor\, p + 1 + 2\sqrt{p}\,\rfloor$$

oppure verificare il valore corretto di $\#E$ applicando l'algoritmo di Schoof.

Capitolo 6

Conclusioni

In questa tesi sono state evidenziate le curve ellittiche su campi primi e le loro applicazioni crittografiche. Dal confronto con i protocolli ElGamal, RSA e DSA, l'uso delle curve ellittiche permette grandi miglioramenti sia in termini di risorse utilizzate sia per la sicurezza che queste offrono.

Sono stati analizzati alcuni algoritmi di risoluzione del problema del logaritmo discreto per le curve ellittiche riportando come questi siano possibili solo per determinate curve (anomale, supersingolari) o adatti ad ogni tipo di curva ma con alcuni grandi svantaggi (una hash table troppo grande da redigere per il BSGS, algoritmi probabilistici come quelli di Pollard).
Numerosi enti quali NIST, Certicom, Brainpool, NSA hanno curato lo studio di questi attacchi e si sono occupati di redigere standard sulla scelta di curve ellittiche. NIST e Certicom risultano i due enti più attivi nelle problematiche attuali di performance e nel definire e proporre le curve oggi utilizzate in ECDH ed ECDSA.

Concludiamo il discorso mostrando una curva ideata dal professor Daniel Julius Bernstein dell'università di Chicago: la **Curve25519**.
La curva è stata progettata per diminuire le operazioni svolte nella moltiplicazione scalare ed al tempo stesso mantenere un alto livello di sicurezza. Sebbene sia una curva ellittica per definizione, la forma in cui troviamo la curva viene detta *forma di Montgomery*. Il passaggio [40] da una curva ellittica $E : y^2 = x^3 + ax + b$ descritta su

di un campo primo $\mathbb{F}_{p^m}$ alla sua forma di Montgomery $E^M : BY^2 = X^3 + AX^2 + X$ è possibile qualora vengano soddisfatte delle ipotesi:

- la caratteristica del campo $\mathbb{F}_{p^m}$ deve essere diversa da 2, stiamo ancora una volta escludendo i campi binari
- l'equazione $x^3 + ax + b = 0$ deve avere almeno una soluzione all'interno del campo F_{p^m}
- il numero $3y^4 + a$ deve avere soluzioni nel campo F_{p^m}
- bisogna rispettare la relazione $B(A^2 - 4) \neq 0$ all'interno del campo $\mathbb{F}_{p^m}$.

L'equazione della Curve25519 è mostrata di seguito:

$$Y^2 = X^3 + 486662X^2 + X \bmod(2^{255} - 19)$$

Due particolarità della curva sono una sicurezza di 128 bit e l'impiego di chiavi crittografiche da 32 byte. La chiave pubblica non è un punto della curva ma la sua sola coordinata x: ciò permette di dimezzare le risorse necessarie all'invio ed alla conservazione in memoria della stessa.
Test effettuati dallo stesso Bernstein, riportati in [45, 46], mettono a confronto la Curve25519 con numerose altre curve, tra le quali quelle standardizzate dal NIST. Si evidenziano quindi una sensibile riduzione nel numero di operazioni e nel tempo di calcolo necessari alla generazione delle chiavi crittografiche.
La base matematica ed i coefficienti scelti per la costruzione della Curve25519 identificano una curva non anomala, nè supersingolare risultando immune all'attacco di Smart; il grande numero primo scelto per la cardinalità della curva offre grande protezione contro l'attacco di Pohlig-Hellman. Infine, come affermato dallo stesso Bernstein in [45] l'implementazione della curva [47] è immune ad attacchi Side-Channel per via di un'efficiente implementazione della scala di Montgomery.
Si conclude dicendo che questa curva rappresenta un'ottima scelta per l'implementazione di protocolli crittografici basati sulla crittografia ellittica.

Ringraziamenti

Il tempo trascorso in questi anni sembra corrispondere ad un attimo. Anni di studio, amicizia, difficoltà hanno contribuito a rendermi una persona più matura e capace di affrontare i problemi a testa alta. La scelta dell'ateneo è seguita da un consiglio di un mio grande amico, Luca, che non ha mai smesso di essere un grande punto di riferimento e consigli. Voglio ringraziarlo ancora una volta per l'amicizia, la pazienza e la costanza che ha messo in ogni cosa rendendola speciale. Un grazie allo stesso modo speciale lo devo a Chiara, per l'infinita gentilezza e attenzione che pone in ogni parola, per la calma con cui affronta le difficoltà e tutta l'amicizia e l'affetto che ha dimostrato in questi anni. Tra i tanti colleghi di università e di corso, Gianpaolo è stato un Amico, un compagno di banco e di studio; di avventure, risate e delle tante frustrazioni per gli esami. Sempre propenso ed impegnato nel risolvere le situazioni più difficili, senza mai far pesare i suoi sforzi, mi ha dato il coraggio e la forza di andare avanti e proseguire tutte le volte che stavo per cedere. Gabriele, compagno di corso anche lui, mi ha offerto spunti di riflessione e pensieri personali che mi hanno portato ad osservare la realtà di tutti i giorni sotto un'ottica nuova, influenzando positivamente e con ottimismo il mio modo di pensare. Devo un grazie ad Alessandro per avermi dato la fiducia e la determinazione a cambiare e completare gli studi; ed al suo infinito ottimismo. Ringrazio ancora una volta Benedetta per l'amicizia e per avermi aiutato in numerose occasioni, soprattutto per la disponibilità quando non potevo rivolgermi ad altri.
In particolare voglio ringraziare Simone, Daniele e tutta la loro famiglia per l'affetto e la disponibilità dimostrati da sempre. Un secondo grazie a Simone per esser stato

inseparabile nonostante la distanza e l'aver condiviso tantissime esperienze ed idee. Non posso dimenticare di ringraziare Leonardo per la più grande amicizia che abbia mai avuto. Lo ringrazio per avermi aiutato e dato consigli ogni giorno; per condividere con me così tante esperienze personali da rendere la nostra amicizia unica in assoluto; per aver trascorso anni in casa insieme ed avermi sempre sopportato con il sorriso. Soprattutto sento di doverlo ringraziare per avermi insegnato a farmi carico dei miei problemi ed a restare calmo e sereno di fronte le difficoltà che non sapevo affrontare.
Arrivo quindi a ringraziare tutti i professori dell'ateneo, rivolgendo un sincero grazie al professor Di Stefano per avermi indicato l'argomento di tesi, per la pazienza portata ed il suo prezioso aiuto che mi ha permesso di completare questo studio con mia grande soddisfazione.

Ringrazio i miei genitori e le mie sorelle per ogni giorno e per l'opportunità di studio datami. In particolare voglio ringraziare mio padre per aver avuto fiducia in me e la pazienza di aspettare fino a questo giorno. Infine vorrei ringraziare mia madre per avermi dato la possibilità di giungere dove sono arrivato e dirle di aver mantenuto la promessa fatta tanti anni fa.

Bibliografia

[1] *Enciclopedia Treccani*, Istituto della Enciclopedia Italiana, 1970.

[2] James S. Milne, *Group Theory*, versione 3.14, 2017.

[3] Mark Reeder, *Notes on Group Theory*, Addison Wesley, Massachusetts, 2015.

[4] Karl-Heinz Fieseler, *Groups, Rings and Fields*, 2010.

[5] Robert B. Ash, *Abstract Algebra: The Basic Graduate Year*, Dipartimento di Matematica, Università di Illinois, revisione Febbraio 2011.

[6] Nigel Hitchin, *Projective Geometry*, Professore di Geometria, Università di Oxford, 2003.

[7] Ulf Persson, *Projective Geometry*, Dipartimento di Matematica, Chalmers Università di Tecnologie, 1989.

[8] Sonja Gorjanc, *Extended Euclidean Plane* - `http://www.grad.hr/geomteh3d/skripta/uvod_eng.html`, traduzione in inglese a cura di Helena Halas e Iva Kodrnja, Università di Zagreb.

[9] Franz Lemmermeyer, *Lecture Notes* - `http://www.fen.bilkent.edu.tr/~franz/ta/ta01.pdf`, Professore di Matematica, 2004.

[10] James W. Kiselik, *Conic and Cubic Plane Curves*, Dipartimento di Matematica, Università di Chicago, 2015.

[11] `WolframAlpha.com` per la consultazione delle seguenti definizioni: Curva ellittica, Punto K-Razionale.

[12] Geir Ellingsrud, *Elliptic curves - Basics*, Dipartimento di Matematica, Università di Oslo, 2014.

[13] Christine Croll, *Torsion Points of Elliptic Curves Over Number Fields*, Tesi di laurea breve in Matematica, Università del Massachusetts, 2006.

[14] John Tate, *The Arithmetic of Elliptic Curves*, Dipartimento di Matematica, Università di Harvard, 1973.

[15] Evan Dummit, *Mathematical Cryptography part 5: Elliptic Curves in Cryptography*, Dipartimento di Matematica, Università di Rochester, 2016.

[16] R.L. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.*

[17] Darrel Hankerson, Alfred Menezes, Scott Vanstone, *Guide to Elliptic Curve Cryptography*, Casa Editrice "Springer-Verlag", 2004.

[18] Neal Koblitz, *A Course in Number Theory and Cryptography*, Dipartimento di Matematica, Università di Washington Seconda Edizione, 1994.

[19] Google, `http://shattered.io/`, esperimento condotto per verificare la possibilità di collisione in SHA-1, 23 Febbraio 2017.

[20] NIST, *FIPS PUB 186-4 - Digital Signature Standard (DSS)*, Categoria: Computer Security, Sottocategoria: Crittografia, Luglio 2013.

[21] BBC, `http://www.bbc.com/news/technology-12116051`, *"iPhone hacker publishes secret Sony PlayStation 3 key"*, 06 Gennaio 2011.

[22] NIST, *Recommended Elliptic Curves For Federal Government Use*, 1999.

[23] Certicom, articolo: *"Certicom Announces Elliptic Curve Cryptography Callenge Winner"*, 2004.

[24] Certicom, *The Elliptic Curve Cryptosystem* - Remarks on the Security of the Elliptic Curve Cryptosystem, 2000.

[25] William Stallings, *Cryptography and Network Security: Principles and Practice*, Quinta edizione, Pearson Editore.

[26] Tabella ASCII - American Standard Code for Information Interchange, `http://www.asciitable.com/`.

[27] NIST, *"Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths"*, Special Publication 800-131A, Revision 1, `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf`, Novembre 2015.

[28] NIST, *"Recommendation for Key Management"*, NIST Special Publication 800-57, Part 1, Revision 4, `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf`, Gennaio 2016.

[29] Matthew Musson, *"Attacking the Elliptic Curve Discrete Logarithm Problem"*, Tesi di Laurea Magistrale delle Scienze Matematiche e Statistiche, Acadia University, 2006.

[30] J. M. Pollard, *"Monte Carlo methods for index computation mod p"*, Vol 32, Numero 143.

[31] Paul C. van Oorschot, Michael J. Wiener, *"Parallel Collision Search with Cryptanalytic Applications"*, 1996.

[32] Andrew Sutherland, *"E18.783 - Elliptic Curves"*, Dipartimento di Matematica, MIT, 2017.

[33] Mandy Zandra Seet, *"ELLIPTIC CURVE CRYPTOGRAPHY - Improving the Pollard-Rho Algorithm"*, Università di New South Wales, 02 Novembre 2007.

[34] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *"Handbook of Applied Cryptography"*, 1996.

[35] Peter Novorney, *"Weak Curves in Elliptic Curve Cryptography"*, 2010.

[36] N. P. Smart, *"The Discrete Logarithm Problem on Elliptic Curves of Trace One"*.

[37] Daniel R. L. Brown, *"Standards for Efficient Cryptography 1 (SEC 1)"*, versione 2.0, Certicom, 21 Maggio 2009.

[38] Daniel Hein, *"Elliptic Curve Cryptography ASIC for Radio Frequency Authentication"*, Gratz University of Technology, 2008.

[39] Daniel Julius Bernstein, ‘‘`https://cr.yp.to/ecdh.html"`, Professore di Matematica, Eindhoven University of Technology 2005.

[40] Katsuyuki Okeya, Hiroyuki Kurumatani, Kouichi Sakurai, *"Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications"*, Professore di Matematica, Università di Illinois, Chicago 2005.

[41] Daniel J. Bernstein, Tanja Lange, *"SafeCurves: choosing safe curves for elliptic-curve cryptography"*, `https://safecurves.cr.yp.to`, 22 Gennaio 2017.

[42] Manfred Lochter, *"ECC Brainpool Standard Curves and Curve Generation"*, versione 1.0, 19 Ottobre 2005.

[43] Jonathan Katz, Yehuda Lindell, *"Introduction to Modern Cryptography"*, seconda edizione, 2014.

[44] Andrew Odlyzko, *"Discrete logarithms over finite fields"*, Università del Minnesota, 2012.

[45] Daniel J. Bernstein, *"Curve25519: new Diffie-Hellman speed records"*, Università di Chicago, 2006.

[46] Daniel J. Bernstein, ‘‘`https://cr.yp.to/ecdh/reports.html"` - *"Speed reports for elliptic-curve cryptography"*, Università di Chicago.

[47] Daniel J. Bernstein, ‘‘`https://cr.yp.to/ecdh.html"` - *"A state-of-the-art Diffie-Hellman function"*, Università di Chicago.

[48] Gregg Musiker, *"Schoof's Algorithm for Counting Points on $E(\mathbb{F}_q)$"*, Università del Minnesota, Professore associato di Matematica, 7 Dicembre 2005.

[49] Paul C. van Oorschot, Michael J. Wiener, *"Parallel Collision Search with Cryptanalytic Applications"*, 1996.

[50] Silje Christensen, Simen Johnsrud, *"Speeding up the Pollards Rho algorithm"*, 1996.

[51] Immagini generate grazie a

- `https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/reals-add.html`
- `https://www.desmos.com/`
- Matlab