UNIVERSITÀ CA' FOSCARI – VENEZIA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica (Triennale)

Tesi di Laurea

Crittografia basata su Curve Ellittiche

Laureando: Gianluca Salvalaggio

Relatore: Chiar.mo prof. Riccardo Focardi

Anno Accademico 2004-2005

a Federica per l'infinita pazienza,

a Riccardo per i suoi "in bocca al lupo",

a tutti e due … per esserci.

Indice

Acronimi

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
DES	Data Encryption Standard
DHP	Diffie Hellman Problem
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
EC	Elliptic Curves
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDHP	Elliptic Curve Diffie-Hellman Problem
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECMQV	Elliptic Curve Menezes Qu Vanstone
FIPS	Federal Information Processing Standards
GF	Galois Field
GMR	Goldwasser, Micali, Rivest
IEEE	Institute of Electrical and Electronics Engineers
IFP	Integer Factorization Problem
MAC	Message Authentication Code
MIME	Multipurpose Internet Mail Extensions
MIPS	Million of Instructions Per Second
MOV	Menezes, Okamoto, Vanstone
NFS	Number Field Sieve
NIST	National Institute of Standards and Technology
PKI	Public Key Infrastructure
QFS	Quadratic Field Sieve
RFC	Request for Comments
RSA	Rivest, Shamir, Adleman
SSL	Secure Socket Layer
TLS	Transport Layer Security

Simboli Matematici

$b \mid a$	b divide a
$\lfloor x \rfloor$	più grande intero minore o uguale a x
$R \cong S$	isomorfismo fra gli anelli R ed S.
$R \oplus S$	somma diretta fra gli anelli R ed S.
$E[m]$	gruppo m-torsione della curva E
$\langle g \rangle$	gruppo ciclico generato da g
$\varphi(n)$	funzione di Eulero, esprime il n° di interi di Z_n coprimi a n.
$\overline{K}$	chiusura algebrica del campo K
Δ	discriminante
char(K)	caratteristica del campo K
$\deg_{MOV}(n)$	grado MOV relativo all'intero n
gcd(a,b)	massimo comune divisore fra a e b
Im(f)	immagine della funzione f
$\log n$	logaritmo in base 2 di n
$\ln n$	logaritmo naturale di n
$L_n[\alpha, c]$	$O(e^{(c+o(1))\ (\log n)^{\alpha}\ (\log\log n)^{1-\alpha}})$
R^*	insieme degli elementi invertibili dell'anello R
Z_n	insieme degli interi modulo n: 0, 1, 2, …, n –1

Prefazione

La sempre maggiore diffusione di dispositivi wireless e il crescente bisogno di *sicurezza* esigono l'utilizzo di sistemi crittografici che possano soddisfare le stringenti limitazioni imposte, in termini di consumo energetico, utilizzo di banda e capacità elaborativa. L'introduzione della Crittografia basata su Curve Ellittiche (**ECC**, *Elliptic Curve Cryptography*) è relativamente recente ma negli ultimi dieci anni tale tecnica si è rapidamente imposta come alternativa ai sistemi crittografici a chiave pubblica già largamente utilizzati come RSA e DSA. La principale attrattiva dell' ECC è che al momento non esistono algoritmi sufficientemente *veloci* che risolvano il problema matematico sul quale si basa. Ciò significa che l'ECC offre lo stesso livello di sicurezza dei sistemi tradizionali (RSA, DSA, Diffie-Hellman) utilizzando chiavi di dimensione inferiore, richiedendo così elaborazioni più veloci, consumi energetici contenuti e un risparmio di utilizzo della banda. Tali caratteristiche, quindi, rendono l'ECC particolarmente adatta all'utilizzo in ambienti con limitata disponibilità di risorse: palmari, telefoni cellulari, smart cards.

Obbiettivi e Organizzazione della Tesi. Verrà presentata la Crittografia basata su Curve Ellittiche, analizzando sia gli aspetti teorici che i protocolli utilizzati e soprattutto confrontando questa tecnica con le attuali soluzioni a chiave pubblica. Considerata la natura dell'argomento, l'esposizione sarà caratterizzata da un taglio necessariamente matematico e formale. Il primo capitolo riprende i concetti base della Crittografia. Il Capitolo 2 presenta i principali sistemi asimmetrici e introduce brevemente l'ECC mentre il terzo descrive le Curve Ellittiche e l'aritmetica ad esse abbinata. Il Capitolo 4 affronta lo studio del problema matematico sul quale si basa l' ECC, ossia l'Elliptic Curve Discrete Logarithm Problem (ECDLP). Nel Capitolo 5 si analizzano i principali protocolli crittografici basati sulle curve ellittiche. L'ultimo capitolo infine confronta l'ECC con gli altri sistemi a chiave pubblica. In Appendice A sono raccolte le nozioni matematiche fondamentali per la comprensione dell'argomento, l'Appendice B contiene le formule matematiche relative alla regola di addizione su Curve Ellittiche.

Al fine di evitare traduzioni dall'inglese all'italiano troppo "forzate", ho mantenuto molti termini nella lingua originale, ritenendo che questo non potesse compromettere la leggibilità dei contenuti.

Capitolo 1

Introduzione alla Crittografia

1.1 Concetti di base

Il termine Crittografia deriva dalla lingua greca (*kryptòs*, nascosto e *gràphein*, scrivere) ed è la scienza, in parte arte, delle scritture segrete. La crittografia, comunque, non mira a nascondere un particolare messaggio, obiettivo della steganografia (*steganòs*, coperto), bensì a modificarne il significato. Per rendere incomprensibile un testo lo si altera per mezzo di un procedimento concordato dal mittente e dal destinatario. Questi può quindi invertire il procedimento, e ricavare il testo originale. Il vantaggio della crittografia è che se il nemico intercetta il messaggio, esso risulta irriconoscibile. Infatti l'avversario, non conoscendo il procedimento di alterazione, troverà arduo, se non impossibile, ricostruire il significato originale. L'utilizzo del termine *nemico* e/o *avversario* trova giustificazione nel fatto che storicamente la crittografia è stata impiegata per scopi militari, in ambienti diplomatici e dai servizi segreti, oltre che per motivi passionali. Un avvincente storia della crittografia è raccontata in [30].

Precisamente la crittografia studia le tecniche matematiche che consentono di modificare un messaggio in modo da renderlo incomprensibile ad un avversario malintenzionato, ma leggibile al destinatario. La modifica del testo in chiaro viene detta *cifratura*, e produce il *testo cifrato*. Il procedimento inverso che dal messaggio cifrato ricostruisce il messaggio in chiaro è chiamato *decifratura*. Come detto precedentemente, il mittente ed il legittimo destinatario devono condividere a priori una conoscenza che consenta la cifratura del messaggio in chiaro e la successiva decifratura. Tale conoscenza però non è il processo di modifica stesso ma la cosiddetta *chiave* ossia una stringa alfanumerica che costituisce un parametro della funzione di cifratura e della funzione di decifratura. Pertanto il metodo di alterazione è noto a chiunque, e quindi anche al nemico, ma ogni volta viene parametrizzato da una chiave segreta condivisa fra mittente e destinatario. Questo concetto, universalmente condiviso, è passato alla storia come *Principio di Kerckhoff*, dal nome del linguista-crittografo fiammingo Auguste Kerckhoff che nel 1883 postulò tale idea in un articolo intitolato "*La cryptographie militare*":

"tutti gli algoritmi devono essere pubblici, solo le chiavi sono segrete"

Nel tempo infatti i vari sistemi di crittografia che basavano la propria sicurezza sulla segretezza del metodo (nel gergo anglosassone, *security by obscurity*) hanno fallito.

In Figura 1.1 è illustrato un generico modello crittografico, dove *P* rappresenta il testo in chiaro e *C* è il messaggio cifrato.

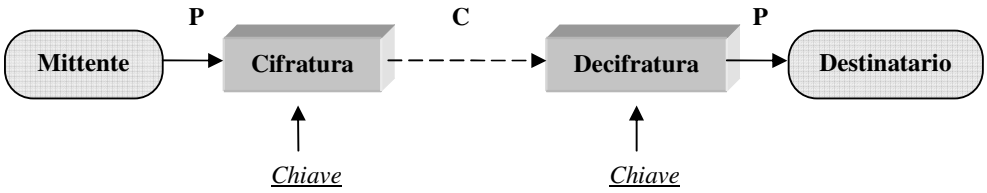

Figura 1.1 Generico schema crittografico

Se la crittografia si occupa dei possibili metodi di cifratura, la Crittoanalisi studia le tecniche che consentono di decifrare, leggi sconfiggere, tali metodi ossia di ricavare il testo in chiaro dal messaggio cifrato senza conoscere la chiave segreta ma sfruttando debolezze dell'algoritmo impiegato. Più precisamente la crittoanalisi mira a scoprire il valore della chiave utilizzata. Le aree della crittografia e della crittoanalisi formano insieme ciò che viene chiamata Crittologia.

Vediamo ora come le tecniche crittografiche vengono utilizzate per garantire comunicazioni sicure fra due controparti.

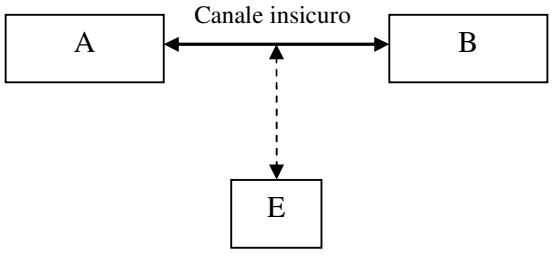

Figura 1.2 Modello di comunicazione

Nella Figura 1.2 sono rappresentate le entità *A* e *B* che comunicano attraverso un canale insicuro sul quale è in ascolto un entità nemica *E*. Ad esempio *A* e *B* possono essere due persone che dialogano attraverso telefoni cellulari ed *E* vuole ascoltare la loro conversazione. Oppure *A* può rappresentare il browser web di una persona α la quale desidera fare acquisti sul sito di shopping-online β rappresentato dal sito web *B*. In tale scenario il canale è Internet e l'entità *E* vuole leggere il traffico fra *A* e *B* per, ad esempio, ottenere il numero della carta di credito di α. Un terzo esempio vede *A* e *B* che si scambiano e-mail su Internet e l'avversario *E* che desidera leggere la loro corrispondenza oppure inviare a sua volta messaggi a *B*, facendo finta di essere *A*.
Gli scenari appena descritti ci aiutano a delineare quelli che sono i principali obiettivi di una comunicazione sicura:

Confidenziality: mantenere un informazione segreta a tutti tranne a chi deve leggerla. Quindi un messaggio inviato da *A* a *B* non deve essere leggibile da *E*.

Data Integrity: assicurare che i dati non vengano alterati senza autorizzazione. Quindi *B* deve poter rilevare se i messaggi inviati da *A* sono stati modificati da *E*.

Data Origin Authentication: verificare la fonte dei dati. *B* può verificare se i dati con mittente pari ad *A* sono stati effettivamente inviati da *A*.

Entity Authentication: verificare l'identità di un'entità. *B* deve essere convinto dell'identità della controparte con cui sta comunicando.

Non-repudiaton: impedire che un entità possa negare di aver commesso una certa azione. Quando *B* riceve un messaggio da *A*, non solo è convinto che tale messaggio è stato proprio generato da *A*, ma può anche convincere una terza parte neutrale di questo; pertanto *A* non può negare di avere inviato il messaggio a *B*.

Gli algoritmi crittografici possono essere divisi in due classi principali: gli algoritmi a chiave simmetrica e quelli a chiave pubblica (o asimmetrica). Nei paragrafi successivi entrambe le famiglie verranno presentate.

1.2 Crittografia a chiave simmetrica

Gli algoritmi di crittografia a chiave simmetrica usano la stessa chiave sia per la cifratura che per la decifratura, da cui il nome. Storicamente nati per primi, i cifrari simmetrici fanno uso, in modo più o meno evoluto, di tecniche di sostituzione e di trasposizione. Nella Figura 1.3 *E*( ) e *D*( ) rappresentano la funzione di cifratura e di decifratura mentre *K* è la chiave condivisa fra mittente e destinatario. Tale chiave dovrà essere comunicata attraverso un *canale sicuro* che, nella realtà dei fatti, è difficile da realizzare[1].

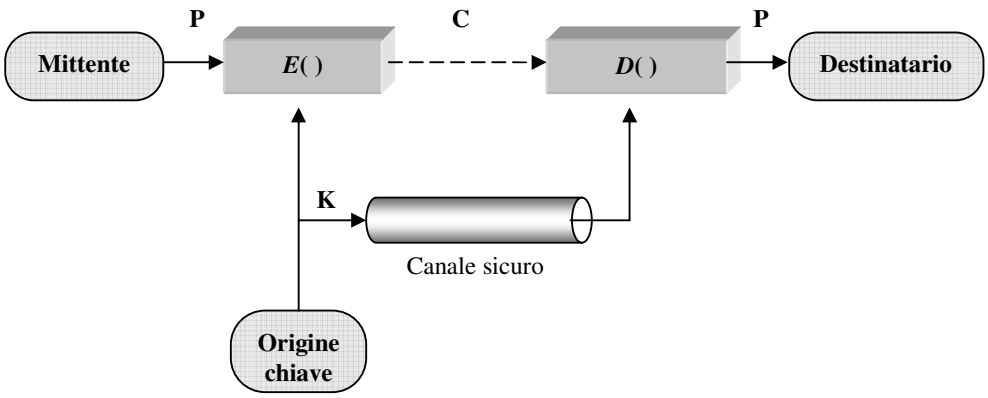

Figura 1.3 Sistema crittografico simmetrico

La caratteristica più apprezzata dei cifrari simmetrici è che sono particolarmente veloci e facili da implementare, sia in hardware che in software. L'aspetto negativo invece, riguarda il cosiddetto *Problema della Distribuzione delle Chiavi*: se ho *N* entità che devono comunicare in modo sicuro attraverso un cifrario di tipo simmetrico, ogni entità dovrà possedere $N - 1$ chiavi differenti, una per ognuna delle altre entità; questo significa che in totale dovranno essere generate, e

[1] D'altra parte questo canale sicuro, potrebbe essere utilizzato anche per trasmettere il messaggio stesso.

conservate, $\frac{N(N-1)}{2}$ chiavi diverse. Si capisce bene che già con valori modesti di N la cosa diventa complicata da gestire.

Vediamo ora una breve rassegna dei principali algoritmi simmetrici. Per approfondimenti si veda [22, 31, 32].

DES/3DES. Nel maggio del 1973 il National Bureau of Standards degli Stati Uniti sollecitò lo sviluppo del Data Encryption Standard (DES) ossia di un crittosistema di riferimento. Il DES venne progettato dall'IBM come modifica di un cifrario preesistente chiamato LUCIFER e venne pubblicato nel Federal Register il 17 marzo 1975. Nel gennaio del 1977 il governo degli Stati Uniti adottò il DES come standard per le informazioni "*unclassified*", ossia non riservate. Tralasciando i dettagli dell'algoritmo è utile evidenziare la lunghezza della chiave utilizzata: 56 bit. Per le potenze di calcolo disponibili negli anni '70, '80 tale dimensione era considerata sufficientemente sicura. Ma già nel 1998 un elaboratore progettato dall'Electronic Frontier Foundation e dalla Cryptography Research, denominato Deep Crack, riuscì a violare l'algoritmo DES con attacco a *forza bruta* in meno di 56 ore, con costi di realizzo alla portata di enti governativi e istituti di ricerca. Pertanto oggi il DES non è più considerato sicuro. Il cosiddetto *triplo DES*, 3DES, costituisce una evoluzione del DES originale, attraverso un uso ripetuto, in cascata, di tre cifrari DES; in tal modo è possibile portare la dimensione della chiave a 112 bit, ottenendo così un crittosistema molto robusto che viene largamente utilizzato in tutte le odierne soluzioni crittografiche commerciali.

AES. Verso la fine degli anni '90 il NIST (National Institute of Standards and Technology) decise che il governo statunitense aveva bisogno di un nuovo standard per le comunicazioni non riservate. A tal fine nel 1997 organizzò una gara internazionale, alla quale parteciparono crittografi da tutto il mondo, per individuare il nuovo standard che avrebbe preso il nome di AES (Advanced Encryption Standard). Nell'ottobre 2000 il NIST annunciò che il vincitore del concorso era l'algoritmo Rijndael, sviluppato da due studiosi belgi Joan Daemen e Vincent Rijmen e nel novembre 2001 AES divenne uno standard del governo degli Stati Uniti. Tale algoritmo supporta chiavi con dimensioni da 128, 192 e 256 bit, quindi è impensabile violarlo con attacco a forza bruta.

1.3 Crittografia a chiave pubblica

Come già detto nel paragrafo precedente, la distribuzione delle chiavi è sempre stato l'anello debole dei cifrari simmetrici: la chiave di cifratura coincide con quella di decifratura, pertanto deve essere protetta e al tempo stesso distribuita agli utenti legittimi del sistema.

Nel 1976 i ricercatori Whitfield Diffie e Martin Hellman proposero nel famoso articolo "*New Directions in Cryptography*" [7] il concetto di crittografia a chiave pubblica, apportando così una svolta radicale nella storia della crittografia. In sostanza un crittosistema a chiave pubblica prevede che la chiave di cifratura sia diversa da quella di decifratura e quest'ultima non facilmente derivabile dalla prima. Nella proposta di Diffie e Hellman l'algoritmo di cifratura $E(\ )$ e quello di decifratura $D(\ )$ dovevano sottostare e tre requisiti:

1. $D(E(P)) = P$
2. risulta estremamente difficile dedurre $D(\ )$ da $E(\ )$
3. $E(\ )$ non può essere forzato da un attacco di tipo *chosen plaintext*

Il metodo funziona così: Alice vuole ricevere messaggi segreti quindi trova due algoritmi $E(\ )$ e $D(\ )$ con le proprietà sopraelencate, parametrizzati rispettivamente da una chiave di cifratura e una chiave di decifratura. La chiave di cifratura viene resa pubblica mentre quella di decifratura viene custodita segretamente da Alice. Indichiamo con $E_{Pub(A)}$ l'algoritmo di cifratura parametrizzato con la chiave pubblica di Alice e con $D_{Priv(A)}(\ )$ l'algoritmo di decifratura parametrizzato dalla chiave privata di Alice. Se Bob vuole spedire in sicurezza un messaggio P ad Alice, si procura la chiave pubblica di quest'ultima, calcola $C = E_{Pub(A)}(P)$ ed invia C ad Alice. Alice decifrerà il messaggio usando la sua chiave privata, ossia calcolerà $D_{Priv(A)}(E_{Pub(A)}(P)\ ) = P$.

La crittografia a chiave pubblica, quindi, richiede che ogni utente abbia due chiavi: una *pubblica* usata da chiunque per cifrare i messaggi a lui destinati, e una *privata* da utilizzare per decifrare i messaggi ricevuti. L'aspetto da sottolineare è che queste due chiavi devono essere in qualche modo correlate, ma deve essere particolarmente difficile poter calcolare la chiave privata a partire da quella pubblica. Questo concetto di *difficoltà* si basa su problemi matematici che, sino ad ora, si sono rivelati particolarmente ardui da risolvere. Attualmente i problemi

matematici che si sono dimostrati maggiormente idonei allo sviluppo di sistemi crittografici a chiave pubblica sono:

> **Integer Factorization Problem** (IFP): dato un numero composto *n*, che è il prodotto di due grandi numeri primi *p* e *q*, trovare *p* e *q*. Su tale problema si basa il sistema RSA.
>
> **Discrete Logarithm Problem** (DLP): calcolare il logaritmo di un numero intero, all'interno di un campo finito. Su tale problema si basa l'algoritmo di firma digitale DSA, il sistema di cifratura di ElGamal e il sistema di scambio chiavi Diffie-Hellman.
>
> **Elliptic Curve Discrete Logarithm Problem** (ECDLP): è una forma generalizzata del DLP, all'interno dei punti di una Curva Ellittica. Su tale problema si basano gli analoghi su Curva Ellittica di DSA, ElGamal e Diffie-Hellman.

I cifrari basati su tali problemi verranno presentati nel secondo Capitolo, mentre i Capitoli 4 e 5 analizzeranno più a fondo i fondamenti teorici e i protocolli dei crittosistemi basati sulle Curve Ellittiche.
Caratteristica negativa degli schemi a chiave pubblica è che sono molto più lenti dei sistemi a chiave simmetrica, ma hanno il grande vantaggio di risolvere elegantemente il Problema della Distribuzione delle Chiavi. Nella pratica quindi, si utilizzano crittosistemi ibridi: gli schemi a chiave pubblica vengono usati preliminarmente per lo scambio delle chiavi (simmetriche) e la crittografia simmetrica viene impiegata per cifrare i dati veri e propri.

1.4 Firma digitale

Una delle applicazioni più importanti della crittografia a chiave pubblica, anche per i risvolti di carattere legale, è la cosiddetta *firma digitale* o *firma elettronica*. Essa consiste in un procedimento matematico che permette di associare, in modo inoppugnabile, un documento elettronico al suo legittimo proprietario e che produce una determinata sequenza binaria, chiamata appunto *firma*. Si capisce bene che, per garantire la veridicità del documento firmato, il processo di firma deve utilizzare un informazione nota solo al proprietario, cioè la sua chiave privata. Il destinatario poi, userà la chiave pubblica del mittente per verificare

l'autenticità della firma. In sostanza se Alice vuole inviare a Bob un messaggio P con la propria firma, calcola $S = Sign_{Priv(A)}(P)$ ed invia la coppia (P,S); in ricezione Bob applica l'algoritmo $Ver_{Pub(A)}(P,S)$ per verificare che S è proprio la firma di Alice del messaggio P. Come si può vedere, l'algoritmo di firma $Sign(P)$ utilizza la chiave privata di Alice mentre quello di verifica $Ver(P,S)$ impiega la chiave pubblica di Alice.

Va precisato che il processo di firma non assicura confidenzialità del messaggio, essa serve solo a garantire l'integrità e l'autenticità del messaggio stesso.

Nella pratica la firma digitale non viene applicata all'intero documento ma ad una sua versione compressa, chiamata *message digest*, ottenuta con l'impiego di funzioni *hash*. Una funzione hash è definita come segue.

Definizione Una *funzione hash* è una funzione $h(\ )$ con le seguenti proprietà:

1. compressione: h mappa un input x di lunghezza arbitraria nel valore $h(x)$ di lunghezza fissa n.
2. computazionalità: dato x e $h(\ )$, è facile calcolare $h(x)$.
3. one-way: per un qualunque valore hash z, è computazionalmente intrattabile trovare un input x tale che $z = h(x)$.
4. weak-collision resistant: dato x è computazionalmente intrattabile trovare un $x' \neq x$ tale che $h(x) = h(x')$.
5. strong-collision resistant: è computazionalmente intrattabile trovare due input distinti $x' \neq x$ tali che $h(x) = h(x')$.

Una trattazione più precisa dell'argomento è esposta in [31, 32].

1.5 Valutare un sistema crittografico

Ci sono più criteri da considerare quando si seleziona un particolare crittosistema. I principali sono:

Funzionalità: il sistema fornisce le funzionalità desiderate?

Sicurezza: quali sono le garanzie di sicurezza del sistema?

Prestazioni: per un dato livello di sicurezza richiesto, il sistema è performante?

Altri fattori che potrebbero influire sulla scelta sono l'esistenza di *best-practices* e standards emessi da organismi internazionali accreditati, la disponibilità di prodotti commerciali, la presenza di eventuali brevetti e la diffusione di pubblicazioni scientifiche a sostegno della validità del sistema.

Scopo di questa tesi è presentare la Crittografia basata su Curve Ellittiche (**ECC**) e considerato che essa è fondamentalmente a chiave pubblica, verrà confrontata con gli schemi asimmetrici attualmente più utilizzati: **RSA** e la famiglia **DL** (*Discrete Logarithm*) che racchiude le principali tecniche basate sul DLP.

Per quanto riguarda il primo criterio, tutte le tre famiglie RSA, DL ed ECC forniscono le funzionalità base richieste ad uno schema a chiave pubblica: cifratura, firma digitale e scambio chiavi.

In questi anni ricercatori e crittoanalisti hanno sviluppato diverse tecniche per testare la *sicurezza* dei sistemi RSA, DL ed ECC. Tale sicurezza è riconducibile alla difficoltà (*hardness*) dei problemi matematici sui quali si basano: rispettivamente IFP, DLP e ECDLP. Va precisato che nessuno di questi problemi è stato dimostrato essere intrattabile (difficile da risolvere in modo efficiente); piuttosto si ritiene che essi siano intrattabili poiché i numerosi sforzi di matematici e ricercatori non sono riusciti a produrre algoritmi efficienti per risolverli. In ogni caso, la difficoltà percepita di tali problemi influisce direttamente sulle *prestazioni* in quanto impone la dimensione delle chiavi utilizzate e questo a sua volta determina le prestazioni delle operazioni aritmetiche coinvolte. Nei Capitoli 2 e 4 viene presentato lo *stato dell'arte* degli algoritmi per risolvere i problemi IFP, DLP e ECDLP mentre nel Capitolo 6 i crittosistemi RSA, DL ed ECC vengono confrontati sulla base della dimensione della chiave.

Capitolo 2

Crittografia a Chiave Pubblica

2.1 Introduzione

Prima di analizzare i principali sistemi asimmetrici, definiamo una famiglia di funzioni che svolgono un ruolo significativo nella crittografia a chiave pubblica. Esse vengono dette funzioni *one-way* poiché sono facili da calcolare direttamente ma molto difficili da invertire. Più precisamente

Definizione 2.1 Una funzione $f : X \to Y$ è detta *one-way* se $\forall\ x \in X$ è "*facile*" calcolare $f(x)$ mentre $\forall\ y \in \mathrm{Im}(f)$ è "*computazionalmente intrattabile*" determinare un valore $x \in X$ tale che $y = f(x)$.

Un esempio di funzione one-way è la seguente: sia $X = Y = Z_{16}$ e $f(x) = 3^x \bmod 17$; preso un $x \in Z_{16}$ è relativamente facile calcolare $f(x)$, invece dato un numero, ad esempio 9, è difficile trovare il valore x tale che $3^x \bmod 17 = 9$.

E' possibile che un' informazione aggiuntiva possa semplificare il calcolo della funzione inversa; tale conoscenza viene detta *trapdoor information* (dall'inglese *botola*) [7]. Più precisamente

Definizione 2.2 Una funzione *trapdoor-one-way* è una funzione *one-way* $f : X \to Y$ tale che la conoscenza di un'informazione *trapdoor* rende agevole calcolare, $\forall\ y \in \mathrm{Im}(f)$, un valore $x \in X$ che soddisfa $y = f(x)$.

Va sottolineato che l'esistenza di funzioni *one-way* e *trapdoor-one-way* è strettamente legata ad una definizione precisa e rigorosa di "*facile*" e "*computazionalmente intrattabile*". In ogni caso l'esistenza di tali funzioni non è stata dimostrata in modo definitivo, si può solo dire che ci sono delle buone candidate. E' proprio su tali funzioni che si basano i principali sistemi crittografici asimmetrici.

Riprendiamo ora i concetti introdotti nel Par. 1.3. Si abbia un crittosistema a Chiave Pubblica per mezzo del quale un entità , chiamiamola Bob, vuole ricevere in modo sicuro il messaggio m :

Cifratura: $c = e(m)$ [dal messaggio m si ottiene il testo cifrato c]
Decifratura: $m = d(c)$ [a partire da c ricostruisco il messaggio m]

La funzione di cifratura $e(\ )$ e quella di decifratura $d(\ )$ utilizzeranno due chiavi differenti, rispettivamente la Chiave Pubblica, disponibile a chiunque, e la Chiave Privata, nota solo a Bob. E' importante, quindi, che dalla conoscenza di $e(\ )$ non si possa ricostruire $d(\ )$. In altre parole, la funzione di cifratura $e(\ )$ dovrà essere una funzione *one-way*, anzi più precisamente $e(\ )$ dovrà essere una funzione *trapdoor-one-way* in quanto Bob dovrà riuscire a decifrare il messaggio e quindi, sulla base di informazioni contenute nella Chiave Privata potrà invertire $e(\ )$.
Nei paragrafi che seguono verranno presentati gli schemi di cifratura e firma elettronica RSA (Par 2.2), quelli basati sul DLP (Par. 2.3) e una versione su curva ellittica di El Gamal (Par. 2.4). Tali descrizioni richiamano concetti di Teoria dei Numeri e di Algebra Astratta facilmente consultabili in Appendice A.

2.2 RSA

Il crittosistema RSA deve il nome ai suoi inventori R.Rivest, A. Shamir, L. Adleman, venne introdotto nel 1977 ed è il sistema a chiave pubblica maggiormente utilizzato. Esso viene impiegato per cifrare e per fornire firma elettronica. Prima di descrivere i meccanismi di RSA, però, diamo la seguente

Definizione 2.3 Il *problema della fattorizzazione intera* (Integer Factorization Problem, IFP) è il seguente: dato un intero positivo n, trovare la sua fattorizzazione prima, ossia trovare i primi $p_1, p_2, \ldots, p_k$ tali che

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

dove $e_i \geq 1$.

Generazione delle Chiavi RSA

La generazione della coppia di chiavi avviene secondo l'Algoritmo 2.1. La chiave pubblica consiste nella coppia di interi (n,e) dove n viene detto *modulus* ed è il prodotto di due numeri primi casuali e segreti p e q della stessa *bitlength*. L'*esponente di cifratura* è un intero e tale che $\gcd(e,\varphi(n))=1$, dove $\varphi(n)=(p-1)(q-1)$ è la funzione di Eulero (vedi Par A.1). La chiave privata è l'intero d, chiamato *esponente di decifratura*, tale che $ed \equiv 1 \bmod \varphi(n)$.

Algoritmo 2.1 Generazione delle chiavi RSA

1. sia l il valore che rappresenta la *bitlength* richiesta per il modulus
2. seleziono due primi p e q della stessa *bitlength* $l/2$.
3. calcolo $n = pq$ e $\varphi(n) = (p-1)(q-1)$
4. seleziono un intero e tale che $1 < e < \varphi(n)$ e $\gcd(e,\varphi(n)) = 1$
5. calcolo l'intero d tale che $1 < d < \varphi(n)$ e $ed \equiv 1 \bmod \varphi(n)$
6. Chiave Pubblica = (n,e), Chiave Privata = d

Cifratura/Decifratura RSA

Si immagini che Bob voglia cifrare un messaggio, da inviare ad Alice: l'algoritmo 3.2 descrive il processo di cifratura/decifratura.

Algoritmo 2.2 Cifratura/Decifratura RSA

1. Cifratura da parte di Bob: $m \rightarrow c$
   a) ottiene la chiave pubblica di Alice (n,e)
   b) rappresenta il messaggio come un intero $m \in [0, n-1]$
   c) calcola $c = m^e \bmod n$ e lo invia ad Alice
2. Decifratura da parte di Alice: $c \rightarrow m$
   a) usa la chiave privata d per ricostruire il messaggio: $m = c^d \bmod n$

Il meccanismo di RSA si basa sull'uguaglianza $m^{ed} \equiv m \pmod n$; ciò è conseguenza del fatto che, per costruzione, vale $ed = k\varphi(n)+1$, per un qualche k intero e quindi, applicando il Teorema di Eulero [31]:

$$m^{k\varphi(n)+1} \equiv \left[\left(m^{\varphi(n)}\right)^k m\right] \bmod n \equiv \left[(1)^k m\right] \bmod n \equiv m \bmod n$$

In questo caso la funzione *trapdoor-one-way* è $f(x) = x^e \bmod n$ e la *trapdoor information* consiste nell'intero d tale che $ed \equiv 1 \pmod{\varphi(n)}$.

Firma Digitale con RSA

Ipotizziamo che Alice voglia firmare un messaggio m da inviare a Bob; lo schema di Firma e successiva Verifica è illustrato nell'algoritmo 2.3. La funzione $H(\ )$ è una funzione hash con la quale Alice genera un *message digest* di m. La chiave pubblica e privata di Alice sono rappresentate rispettivamente da (n,e) e d.

Algoritmo 2.3 Firma e Verifica RSA

1. Alice genera la firma s del messaggio m
   a) calcola $h = H(m)$
   b) calcola $s = h^d \bmod n$
   c) s è la firma di Alice del messaggio m
2. Bob riceve la coppia (m,s) e verifica la firma di Alice
   a) calcola $h = H(m)$
   b) calcola $h' = s^e \bmod n$
   c) se $(h = h')$ allora la *firma è accettata*
      altrimenti la *firma è rifiutata*

Analogamente a prima, il funzionamento della verifica si basa sul fatto che $h^{ed} \equiv h \pmod{n}$.

2.1.1 Algoritmi per risolvere l'IFP

Appare evidente che la fattorizzazione di $n = pq$ comporta la rottura di RSA, poiché consente di ottenere l'inverso di e modulo $\varphi(n)$. Per quanto la cosa non sia dimostrata, è opinione diffusa che valga anche l'inverso, ossia che l'unico modo per invertire la funzione di cifratura RSA implichi la fattorizzazione del modulus n. Alcune pubblicazioni, però, suggeriscono che tale convinzione potrebbe essere falsa, ossia che l'IFP potrebbe essere più difficile del problema di invertire la funzione $m \rightarrow m^e \bmod n$ [3].

La fattorizzazione degli interi è uno dei problemi più antichi della matematica, basti pensare che molte tecniche utilizzate nei moderni algoritmi di fattorizzazione risalgono agli antichi Greci (come ad esempio il Crivello di Erastotene o l'algoritmo di Euclide per calcolare il gcd). Anche i grandi matematici del '700 e del '800 (Fermat, Eulero, Legendre, Gauss, Jacobi) hanno dato il loro contributo. E' comunque negli ultimi trent' anni, dopo l'introduzione della crittografia a chiave pubblica, che si sono avuti i progressi maggiori. Il fatto poi che la sicurezza del sistema RSA sia strettamente legata al problema della fattorizzazione spiega come mai la società *RSA Security* finanzi il concorso *RSA Factoring Challenge* che premia chi riesce a fattorizzare numeri sempre più grandi, i cosiddetti *RSA Challenge numbers*. I più recenti progressi nella fattorizzazione intera sono legati al *RSA Factoring Challenge.*

Vi sono due tipi di algoritmi per risolvere l' IFP, gli algoritmi *special-purpose* e quelli *general-purpose*. I primi sfruttano particolari caratteristiche del numero *n* da fattorizzare, mentre i secondi hanno un *running time* che dipende unicamente dalle dimensioni di *n*.

Algoritmi special-purpose. Un esempio di algoritmo *special-purpose* è costituito dal metodo di fattorizzazione basato sulle curve ellittiche (ECM), inventato da Hendrik Lenstra nel 1985. Il running time di tale algoritmo dipende dalle dimensioni dei fattori primi di *n*, e quindi tende ad individuare per primi fattori piccoli. Il progetto ECMNET, iniziato nel 1998 per individuare grandi fattori primi con ECM è riuscito a trovare fattori con più di 50 cifre (166 bits). Nel 1999 venne pubblicata la scoperta di un fattore di 54 cifre (180 bit) di un numero di 127 cifre (422 bit). Ad oggi l'ultimo record dell' ECM risale al 6 aprile 2005, giorno in cui Bruce Dodson ha individuato un fattore primo di 66 cifre del numero $3^{466} + 1$.

Algoritmi general-purpose. L'idea di base dei principali algoritmi di fattorizzazione *general-purpose* è detta *"differenza dei quadrati"* e consiste nel trovare due interi *x* e *y* tali che

$$(1)\ x^2 \equiv y^2 \bmod n \qquad \text{e} \qquad (2)\ x \neq \pm y \bmod n$$

dove *n* è il numero da fattorizzare .

La (1) può essere riscritta come $(x - y)(x + y) \equiv 0 \pmod{n}$ (ossia $n|(x - y)(x + y)$) e la (2) implica che $(x + y)$ e $(x - y)$ non sono divisibili per n; ciò significa che $\gcd(x - y, n)$ e $\gcd(x + y, n)$ sono due fattori non banali di n.

Gli algoritmi di fattorizzazione *general-purpose* più veloci sono il Quadratic Sieve (QS) e il Number Field Sieve (NFS) e il loro funzionamento si basa sull'idea di trovare una base di fattori primi per generare un sistema lineare di equazioni la cui soluzione porti ad equazioni del tipo (1), verificando la condizione (2).

Il QS venne sviluppato nel 1984 da Carl Pomerance ed inizialmente fu utilizzato per fattorizzare numeri con circa 70 cifre (223 bit). Nel 1994 un gruppo di ricercatori, guidato da Arjen Lenstra, riuscì a fattorizzare con il QS l'*RSA Challenge Number* (RSA-129) di 129 cifre (429 bit). Tale fattorizzazione venne portata a termine in 8 mesi, per un *running time* complessivo di 5000 anni-MIPS.

Il *running time* del QS è pari a $L_n\left[\frac{1}{2},1\right]$[2].

Inizialmente il Number Field Sieve venne sviluppato, nel 1989, come algoritmo *special-purpose* per fattorizzare interi nella forma $n = r^e - s$, con r e $|s|$ piccoli. Successivamente venne introdotta una sua versione generalizzata, talvolta indicata come *General Number Field Sieve* (GNFS), che consentiva di fattorizzare qualsiasi tipo di intero. Nel 1996 l'utilizzo del NFS ha consentito di fattorizzare un intero di 130 cifre (432 bit), richiedendo solo il 15% dei 5000 anni-MIPS che sono stati necessari per fattorizzare con il QS il RSA-129. Il 2 novembre 2005 è stato fattorizzato con NFS un intero di 640 bit (RSA-640).

L'algoritmo NFS ha un *running time* pari a $L_n\left[\frac{1}{3},1.923\right]$ ed è attualmente il più veloce algoritmo di fattorizzazione di interi. Lo studioso Richard Brent, dell'Oxford University Computing Laboratory, ha proposto una formula empirica [27] che predice l'anno Y nel quale si potrà fattorizzare un intero di D cifre decimali:

$$Y = 13.24\, D^{1/3} + 1928.6$$

Con questa formula Brent ha tenuto in considerazione i record di fattorizzazione conosciuti, la complessità del NFS e la legge di Moore (la potenza di calcolo

[2] La funzione $L_n[a,c]$ è descritta nel Paragrafo A.4.

raddoppia ogni 18 mesi). Secondo questa stima, nel 2010 e nel 2018 sarà possibile fattorizzare un numero rispettivamente di 768 bit (D = 231) e di 1024 bit (D = 309).

2.3 Sistemi basati sul Logaritmo Discreto

La sicurezza di molte tecniche crittografiche si basa sull'intrattabilità del Problema del Logaritmo Discreto, come ad esempio il protocollo di *key-agreement* Diffie-Hellman , il crittosistema El Gamal e l'algoritmo di firma elettronica DSA. Prima di tutto diamo la seguente definizione:

Definizione 2.4 Sia G un gruppo ciclico finito di ordine n. Sia α un generatore di G e β un elemento di G. Il *Logaritmo Discreto* (DL) di β in base α, indicato con $\log_\alpha \beta$, è l'unico intero x, $0 \le x \le n-1$, tale che $\alpha^x = \beta$.

I gruppi di particolare interesse in crittografia sono i gruppi moltiplicativi Z_p^* degli interi modulo p, con p primo, quindi si definisce

Definizione 2.5 Il *Problema del Logaritmo Discreto* (DLP) è il seguente: dato il numero primo p, il generatore α di Z_p^* e $\beta \in Z_p^*$, trovare l'intero x, $0 \le x \le p-2$, tale che $\alpha^x \equiv \beta \bmod p$.

L'utilizzo del DLP in crittografia trova motivo nel fatto che calcolare il logaritmo discreto è difficile, mentre l'operazione inversa di esponenziazione può essere calcolata in modo efficiente. In altre parole l'esponenziazione modulo p è una funzione *one-way*, per valori opportunamente grandi di p.

2.2.1 Protocollo Diffie-Hellman

Il protocollo Diffie-Hellman venne presentato, nello stesso articolo che introdusse la Crittografia a Chiave Pubblica, da Whitfield Diffie e Martin E. Hellman nel 1976 [7]. Esso è un protocollo di *key-agreement*, che consente a due entità A e B di stabilire una *chiave segreta condivisa K* attraverso un canale insicuro, senza richiedere, a priori, alcuna conoscenza in comune. Vediamo come funziona:

Protocollo 2.1 Key-agreement Diffie-Hellman

1. Si considerano un primo p e un generatore α di Z_p^*
2. A seleziona un intero casuale $x \in [1, p-2]$ e calcola $X = \alpha^x \bmod p$
3. A invia X a B
4. B seleziona un intero casuale $y \in [1, p-2]$ e calcola $Y = \alpha^y \bmod p$
5. B invia Y ad A
6. A riceve Y e calcola $K_x = Y^x \bmod p = (\alpha^y)^x \bmod p$
7. B riceve X e calcola $K_y = X^y \bmod p = (\alpha^x)^y \bmod p$
8. A e B condividono la chiave $K = K_x = K_y$

In questa situazione, per ottenere la chiave K, il nemico non dovrà risolvere direttamente il DLP ma il seguente problema:

Definizione 2.6 Si definisce *Problema Diffie-Hellman* (DHP) il seguente: dato un primo p, un generatore α di Z_p^* e i valori $\alpha^x \bmod p$ e $\alpha^y \bmod p$ trovare $\alpha^{xy} \bmod p$.

Sebbene sia facile dimostrare che la risoluzione del DLP in Z_p^* implichi la risoluzione del DHP, l'inverso non è ancora stato provato. In altre parole non è stata dimostrata l'equivalenza fra il DLP e il DHP. In [22] vengono elencate alcune condizioni per le quali si verifica tale equivalenza .

2.2.2 Sistema El Gamal

Questo algoritmo, presentato da El Gamal nel 1984, costituisce il primo crittosistema basato sul DLP. Lo schema base e le sue varianti vengono impiegati anche per implementare firma elettronica.

Cifratura/Decifratura El Gamal

Gli algoritmi 2.4 e 2.5 descrivono rispettivamente la generazione delle chiavi e lo schema di Cifratura/Decifratura del sistema El Gamal.

Algoritmo 2.4 Generazione delle Chiavi per El Gamal

1. genera un grande primo p e seleziona un generatore α di Z_p^*
2. seleziona un intero casuale $x \in [1, p-2]$ e calcola $\alpha^x \bmod p$
3. Chiave Pubblica = (p, α, α^x) , Chiave Privata = x

Algoritmo 2.5 Cifratura/Decifratura El Gamal

1. Cifratura da parte di Bob: $m \rightarrow c$
   a) ottiene la chiave pubblica di Alice (p, α, α^x)
   b) rappresenta il messaggio come un intero $m \in [0, p-1]$
   c) seleziona un intero casuale $k \in [1, p-2]$
   d) calcola $\gamma = \alpha^k \bmod p$ e $\delta = m \cdot (\alpha^x)^k \bmod p$
   e) invia $c = (\gamma, \delta)$ ad Alice
2. Decifratura da parte di Alice: $c \rightarrow m$
   a) usa la chiave privata x per calcolare $\gamma^{-x} \bmod p$
   b) ricostruisce m calcolando $(\gamma^{-x}) \cdot \delta \bmod p$.

Il calcolo di decifratura dell'algoritmo 3.5 consente di recuperare il messaggio m in quanto si ha:

$$(\gamma^{-x}) \cdot \delta \equiv \alpha^{-xk} m \alpha^{xk} \equiv m \pmod p.$$

Il sistema di cifratura El Gamal è non-deterministico in quanto il testo cifrato c dipende oltre che dal testo in chiaro m anche dal valore casuale k, scelto da Bob. Questo significa che uno stesso messaggio verrà cifrato in modi differenti, a seconda del valore k.

Firma Digitale con El Gamal

Anche lo schema di firma elettronica El Gamal è, in un certo senso, aleatorio. Questo significa che per un dato messaggio m vi sono numerose firme valide. L'algoritmo 2.6 descrive il processo in cui Alice usa la propria chiave privata x per firmare il messaggio m e Bob verifica tale firma utilizzando la chiave pubblica di Alice (p, α, α^x). Entrambi usano la funzione hash $H(\ )$.

Algoritmo 2.6 Firma e Verifica El Gamal

1. Alice genera la firma (r,s) del messaggio m
   a) seleziona un intero casuale $k \in [1, p-2]$ con $\gcd(k, p-1) = 1$
   b) calcola $r = \alpha^k \bmod p$
   c) calcola $s = k^{-1} [H(m) - xr] \bmod (p-1)$
   d) la coppia (r,s) è la firma di Alice del messaggio m
2. Bob verifica la firma di Alice (r,s) sul messaggio m.
   a) verifica che $r \in [1, p-1]$, altrimenti rifiuta la firma
   b) calcola $v_1 = \alpha^{xr} r^s \bmod p$
   c) calcola $v_2 = \alpha^{H(m)} \bmod p$
   d) accetta la firma $\Leftrightarrow$ $v_1 = v_2$

La verifica della firma si basa sulle seguenti considerazioni:
se s è correttamente generato da Alice allora $ks \equiv (H(m) - xr) \bmod (p-1)$, che equivale a scrivere $H(m) \equiv xr + ks \pmod{(p-1)}$. Ciò significa che $\alpha^{H(m)} \equiv \alpha^{xr + ks} \equiv (\alpha^x)^r r^s \pmod p$ e quindi $v_1 = v_2$, come richiesto.

2.2.3 Algoritmo DSA

Nel 1991 il NIST (U.S. National Institute of Standards and Technology) propose un nuovo schema di firma elettronica, il Digital Signature Algorithm (DSA). Esso divenne un Federal Information Processing Standard (FIPS 186), con il nome di *Digital Signature Standard*. Il DSA costituisce una variante dello schema di firma El Gamal. Gli algoritmi 2.7 e 2.8 illustrano rispettivamente la generazione della coppia di chiavi e lo schema di firma/verifica DSA.

Algoritmo 2.7 Generazione delle Chiavi per DSA

1. Seleziona un numero primo q a 160 bit, tale che $2^{159} < q < 2^{160}$.
2. Seleziona un primo p a 512 bit, tale che $q | (p-1)$.
3. Seleziona un elemento $\alpha \in Z_p^*$ di ordine q
4. Seleziona un intero casuale $x \in [1, q-1]$
5. Calcola $y = \alpha^x \bmod p$
6. Chiave Pubblica = (p, q, α, y), Chiave Privata = x

Nell'algoritmo seguente si ipotizza che Alice firmi, con la propria chiave privata, un messaggio *m* di lunghezza arbitraria e Bob successivamente verifichi tale firma, usando la chiave pubblica di Alice. Entrambi utilizzano la funzione hash *H*( ).

Algoritmo 2.8 Firma e Verifica DSA

1. Alice genera la firma (*r,s*) del messaggio *m*
   a) Seleziona un intero casuale $k \in [1, q-1]$
   b) Calcola $r = (\alpha^k \bmod p) \bmod q$
   c) Calcola $k^{-1} \bmod q$
   d) Calcola $s = k^{-1}[H(m) + xr] \bmod q$
   e) La coppia (*r,s*) è la firma di Alice del messaggio *m*
2. Bob verifica la firma di Alice (*r,s*) sul messaggio *m*
   a) Verifica che $r \in [1, q-1]$, altrimenti rifiuta la firma
   b) Calcola $w = s^{-1} \bmod q$
   c) Calcola $u_1 = w\,H(m) \bmod q$ e $u_2 = rw \bmod q$
   d) Calcola $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$
   e) Accetta la firma $\Leftrightarrow v = r$

I passaggi seguenti dimostrano come il meccanismo di firma/verifica DSA funzioni correttamente :
se (*r,s*) è la firma legittima di Alice sul messaggio *m* allora vale $H(m) \equiv -xr + ks \pmod q$ e quindi, moltiplicando per *w*, $wH(m) + xrw \equiv k \pmod q$. L'ultima congruenza può essere riscritta come $u_1 + xu_2 \equiv k \pmod q$, pertanto se si eleva α ad entrambi i membri si ottiene $(\alpha^{u_1} y^{u_2} \bmod p) \bmod q = (\alpha^k \bmod p)$, cioè $v = r$, come richiesto.

2.2.4 Algoritmi per risolvere il DLP

A differenza dell' IFP, dove la *dimensione del problema* è legata all'intero *n* da fattorizzare, nel DLP la dimensione dell'*input* coincide con il numero di punti *N* del gruppo *G* considerato. Se ad esempio G = Z_p^* , con *p* primo, si ha che $N = p - 1$.

Come per l'IFP, anche per il DLP vi sono due tipi di algoritmi di risoluzione. Gli algoritmi *special-purpose* sfruttano particolari caratteristiche del numero *N*, mentre il *running time* degli algoritmi *general-purpose* dipende solo dalle dimensioni di *N*.

I più veloci algoritmi *general-purpose* per risolvere il DLP si basano sul metodo *index-calculus*. Esso è un metodo probabilistico che si applica solo su campi finiti. Esempi di campi finiti comunemente usati nella pratica sono $GF(p)$ e $GF(2^m)$. Il metodo *index-calculus* è attualmente l'unico algoritmo che risolve il DLP in tempo *sub-esponenziale* e lavora in modo molto simile a NFS e QS. Richiede l'individuazione della cosiddetta *base di fattori S* che contiene numeri primi tali che una parte significativa degli elementi di *G* sia esprimibile come prodotto di elementi di *S*. In seguito costruisce un database di relazioni che si utilizza ogni volta viene richiesto il logaritmo di un elemento. Per approfondimenti si veda [14].

Attualmente il più veloce algoritmo per risolvere il DLP in $GF(2^m)$ è una variazione dell'*index-calculus* nota con il nome di *Algoritmo di Coppersmith* ed ha un *running time* pari a $L_{2^m}\left[\frac{1}{3}, c\right]$ con $c < 1.587$. Invece il più veloce algoritmo per calcolare logaritmi in Z_p^* è il Number Field Sieve (NFS), sempre una variante dell'*index-calculus*, con un *running time* uguale a $L_p\left[\frac{1}{3}, 1.923\right]$. Entrambi gli algoritmi hanno un *running time* sub-esponenziale. L'attuale record (2001, Joux-Lercier) di calcolo di logaritmo in Z_p^* è stato realizzato in un campo con un modulus *p* di 120 cifre.

2.4 Crittografia su Curva Ellittica

Ricordiamo dalla definizione 2.4 che il Logaritmo Discreto è definibile su un qualsiasi gruppo ciclico. I sistemi visti nel paragrafo precedente si appoggiano sul DLP, cioè sulla risoluzione del Logaritmo Discreto in gruppi moltiplicativi Z_p^*, con p primo. La crittografia basata su Curve Ellittiche (ECC), invece si basa sulla risoluzione Logaritmo Discreto nel gruppo di punti di una curva ellittica definita su un campo finito.

Ma, prima di tutto, cos'è una curva ellittica? La definizione rigorosa verrà data nel capitolo successivo, per ora possiamo dire che una curva ellittica E definita sul campo finito F_p , con p primo, è data dall'equazione

$$y^2 = x^3 + ax + b \qquad (2.1)$$

dove i coefficienti $a,b \in F_p$ verificano $4a^3 + 27b^2 \neq 0$. La coppia (x,y), con $x, y \in F_p$ che soddisfano la (2.1), è un punto della curva E. L'insieme di tutti i punti di E si indica con $E(F_p)$.

Come si vedrà meglio nel capitolo seguente, è possibile definire l'operazione di addizione (+) fra due punti P e Q di una curva ellittica, utilizzando lo speciale *punto all'infinito* Θ come elemento identità. Si dimostra altresì che $(E(F_p),+)$ forma un gruppo (additivo) abeliano. Ciò significa che è possibile definire il Logaritmo Discreto su sottogruppi ciclici di $E(F_p)$.

Vediamo ora di illustrare l'idea di base dell'ECC, realizzando un analogo del sistema El Gamal su curva ellittica.

2.4.1 El Gamal su curva ellittica

Sia E una curva ellittica definita su F_p e sia P un punto in $E(F_p)$ con ordine primo n. Consideriamo il sottogruppo ciclico generato da P

$$\langle P \rangle = \{\Theta, P, 2P, 3P, \ldots, (n-1)P\}$$

L'equazione di E, il punto P ed i primi p ed n sono conosciuti da tutti. La coppia di chiavi Pubblica e Privata è così generata:

- la Chiave Privata è un intero d scelto a caso nell'intervallo $[1, n-1]$

- la Chiave Pubblica è il punto $Q = dP$ appartenente a $\langle P \rangle$.

Il problema di determinare d conoscendo P e Q è proprio *l'Elliptic Curve Discrete Logarithm Problem* (ECDLP).

L'analogo EC dell'algoritmo El Gamal venne proposto da Koblitz nel 1986. A differenza del sistema originale, il messaggio non viene rappresentato come un intero ma come un punto M in $E(F_p)$. Il funzionamento è molto intuitivo: il punto M viene cifrato sommando ad esso il punto kQ, dove k è un intero casuale e Q è la chiave pubblica del destinatario. Il mittente quindi spedisce i punti kP e $M + kQ$. Il destinatario usa la propria chiave privata d per recuperare da essi il messaggio M. Infatti: $d(kP) = k(dP) = kQ$ e quindi $M + kQ - d(kP) = M + kQ - kQ = M$

Il procedimento è riassunto negli algoritmi 2.9 e 2.10

Algoritmo 2.9 Cifratura El Gamal su EC

INPUT: Chiave pubblica Q, messaggio m

OUTPUT: testo cifrato (C_1,C_2)

1. Rappresenta il messaggio m come un punto M in $E(F_p)$
2. Seleziona a caso $k \in [1, n-1]$
3. Calcola $C_1 = kP$
4. Calcola $C_2 = M + kQ$
5. Return(C_1,C_2)

Algoritmo 2.10 Decifratura El Gamal su EC

INPUT: Chiave privata d, testo cifrato (C_1,C_2)

OUTPUT: messaggio m

1. Calcola $M = C_2 - dC_1$ ed estrai m dal punto M
2. Return(m)

Si capisce che i meccanismi base dell'ECC sono abbastanza semplici. La novità, infatti, non risiede nello sfruttare nuovi problemi matematici ma nell'applicare un problema noto, il Logaritmo Discreto, in un particolare tipo di gruppo, quello dei punti di una curva ellittica. Come si vedrà più avanti, questo comporta notevoli vantaggi dal punto di vista della sicurezza.

Dopo questa breve introduzione all'ECC, approfondiamo l'argomento. Il capitolo che segue fornisce una descrizione formale delle Curve Ellittiche, evidenziando le caratteristiche di interesse crittografico. Nel Cap. 4 viene analizzato l'ECDLP, in particolare vengono presentati i metodi di risoluzione dello stesso. Infine il Cap. 5 descrive i sistemi crittografici basati su ECC utilizzati nella pratica.

Capitolo 3

Curve Ellittiche

Sebbene la Crittografia basata su Curve Ellittiche (ECC) sia stata introdotta appena vent' anni fà, lo studio delle Curve Ellittiche, in particolare nel campo della Teoria dei Numeri, dell'Algebra e della Geometria, risale alla metà del XIX secolo. Tali affascinanti oggetti matematici sono stati impiegati per risolvere problemi di varia natura, come il *problema dei numeri congruenti*[3] e la fattorizzazione di numeri interi. Recentemente , negli anni 1993-1994, L'ultimo Teorema di Fermat è stato dimostrato dal matematico Andrew Wiles utilizzando una avanzata teoria delle Curve Ellittiche [33].

Obiettivo di questo Capitolo è dare una descrizione matematica delle Curve Ellittiche. Verrà studiata l'aritmetica basata su curve ellittiche (legge di gruppo, moltiplicazione scalare) e verranno presentati i concetti di ordine di curva e ordine di punto, fondamentali per comprendere meglio l'ECC e l'ECDLP in particolare. Infine si studieranno famiglie di Curve Ellittiche considerate crittograficamente insicure, quindi da evitare nella pratica.

3.1 Introduzione

Consideriamo un generico campo K: potrebbe essere il campo finito $GF(q)$, con q potenza di un primo, l'insieme dei reali $\mathbb{R}$, il campo dei numeri razionali $\mathbb{Q}$ oppure l'insieme dei numeri complessi $\mathbb{C}$. Quindi diamo la seguente

Definizione 3.1 Una *curva ellittica E/K*, sul campo K è una curva *smooth* definita dall'Equazione Generalizzata di Weierstrass:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad (3.1)$$

dove $a_1, a_3, a_2, a_4, a_6 \in K$.

[3] numeri interi che corrispondono all'area di un triangolo rettangolo i cui lati hanno lunghezza intera

Indichiamo con $E(K)$ tutti i punti di K^2 che soddisfano l'equazione (3.1) insieme al cosiddetto "*punto all'infinito*" Θ, il cui significato verrà spiegato più avanti. Quindi si scrive:

$$E(K) = \left\{(x, y) \in K \times K \,\middle|\, y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\right\} \cup \{\Theta\}$$

Il termine *smooth* (dall'inglese, *liscio*) significa che in $E(K)$ non ci sono punti *singolari*, ossia punti nei quali entrambe le derivate parziali si annullano. Pertanto una curva *smooth* è anche detta *non-singolare*. A tale proposito si definisce *discriminante* Δ della curva E definita dall'equazione (3.1) la quantità:

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6$$

dove

$$d_2 = a_1^2 + 4a_2$$

$$d_4 = 2a_4 + a_1a_3$$

$$d_6 = a_3^2 + 4a_6$$

$$d_8 = a_1^2 a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

Si dimostra peraltro che una curva ellittica E è *non-singolare* se e solo se il suo discriminante Δ è diverso da zero [29].

Nella definizione di curva ellittica si è citato il cosiddetto *punto all'infinito* Θ. In modo un po' forzato, può essere visto come il punto di coordinate (∞, ∞) posizionato nella parte superiore dell'asse delle *y*. Una tale definizione è stata introdotta per avere consistenza formale. Ad esempio una retta che passa per il punto P e il punto all'infinito è la retta verticale passante per P. Il punto Θ, inoltre, non è posizionato solo nella parte superiore dell'asse delle *y* ma anche in quella inferiore. Azzardando un po' con l'immaginazione si può pensare che le due estremità dell'asse delle *y* si congiungano all'infinito proprio nel punto Θ.

La definizione di curva ellittica appena data si applica ad ogni tipo di campo, ma in crittografia si è interessati unicamente ai campi finiti. In particolare al *campo finito primo* $GF(p)$, con p primo molto grande, per la sua struttura e al *campo finito binario* $GF(2^m)$, per la facilità di implementazione hardware.

Vediamo ora come si può semplificare l'equazione (3.1).

Definizione 3.2 Due curve ellittiche E_1 e E_2 definite sul campo K e descritte da

$$E_1: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

$$E_2: \quad y^2 + \overline{a}_1xy + \overline{a}_3y = x^3 + \overline{a}_2x^2 + \overline{a}_4x + \overline{a}_6$$

sono *isomorfiche su K* se esistono $u,r,s,t \in K$, con $u \neq 0$, tali che il cambio di variabili

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + t)$$

trasforma l'equazione E_1 in E_2. Tale trasformazione è chiamata *trasformazione ammissibile di variabili*.

Procediamo quindi con la semplificazione della (3.1), distinguendo a seconda della caratteristica del campo K.

Char(K) $\neq 2,3$

Tale condizione è soddisfatta dall'insieme dei numeri reali $\mathbb{R}$ e soprattutto dai campi finiti primi $GF(p)$. Se la caratteristica di K è diversa da 2 e 3, la seguente trasformazione ammissibile di variabili [20]

$$(x, y) \rightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24} \right)$$

trasforma l'equazione (3.1) in

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

In questo caso la condizione di *non singolarità* della curva ha un significato ben preciso. Infatti se considero l'equazione implicita $F(x, y) = y^2 - f(x)$, dove $f(x) = x^3 + ax + b$, la non singolarità impone che non esista alcun punto di E in cui le due derivate parziali

$$\frac{\partial F}{\partial x} = -f'(x), \quad \frac{\partial F}{\partial y} = 2y$$

si annullino contemporaneamente. Considerando l'equazione $y^2 = f(x)$ si può calcolare la pendenza $\frac{dy}{dx}$ della retta tangente a *E* nel punto (*x,y*). Facendo uso della derivazione implicita si ottiene $2y\frac{dy}{dx} = f'(x)$ e quindi

$$\frac{dy}{dx} = \frac{f'(x)}{2y} = -\frac{\partial F / \partial x}{\partial F / \partial y}$$

L'ultima equazione ha senso se il denominatore è diverso da 0 e può essere interpretata come la pendenza di una linea verticale se il numeratore è diverso da 0 e il denominatore è nullo. Se però entrambe le quantità si annullano, la pendenza della retta tangente in quel punto perde significato. Dal punto di vista algebrico, se entrambe le derivate parziali si annullano nel punto $P = (x_0,y_0)$ significa che $f'(x_0) = 2y_0 = 0$, e quindi $y_0 = 0$. Inoltre, considerato che $y^2 = f(x)$, si ha anche $f(x_0) = 0$. E' peraltro facile dimostrare che un polinomio $f(x)$ ammette una radice multipla α se e solo se α è radice di $f(x)$ e di $f'(x)$ [1]. Quindi l'annullamento delle derivate parziali in $P = (x_0,y_0)$ implica che x_0 sia una radice multipla di $f(x)$ e quindi che il discriminante Δ di $f(x)$ sia nullo. Dal Par. A.3 ricordiamo che il discriminante del polinomio $f(x) = x^3 + ax + b$ vale $-\left(4a^3 + 27b^2\right)$. Diamo quindi la seguente

Definizione 3.3 Una curva ellittica *E* sul campo *K*, avente caratteristica diversa da 2 e 3, è definita dall'Equazione di Weierstrass

$$y^2 = x^3 + ax + b \qquad (3.2)$$

dove $a,b \in K$ soddisfano la disuguaglianza $-\left(4a^3 + 27b^2\right) \neq 0$.

A titolo di esempio consideriamo il campo $K = \mathbb{R}$. La Figura 3.1 visualizza la curva ellittica $y^2 = x^3 - 3x + 3$ mentre la Figura 3.2 rappresenta la curva $y^2 = x^3 - 3x + 2 = (x-1)^2(x+2)$ la quale presenta una singolarità nel punto (1,0); tale tipo di singolarità è chiamata *nodo*. In Figura 3.3, invece, è

rappresentata la curva $y^2 = x^3$, che nel punto (0,0) presenta una singolarità di tipo *cuspide*.

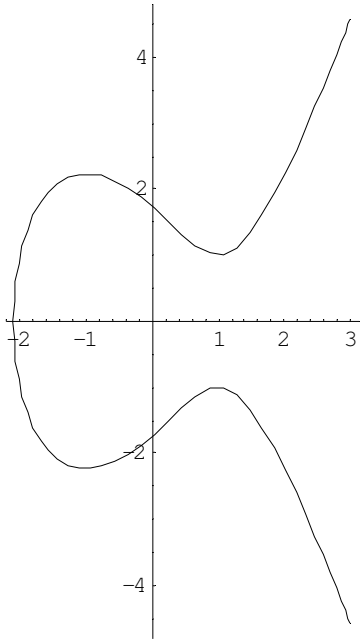

Figura 3.1: Curva ellittica $y^2 = x^3 - 3x + 3$

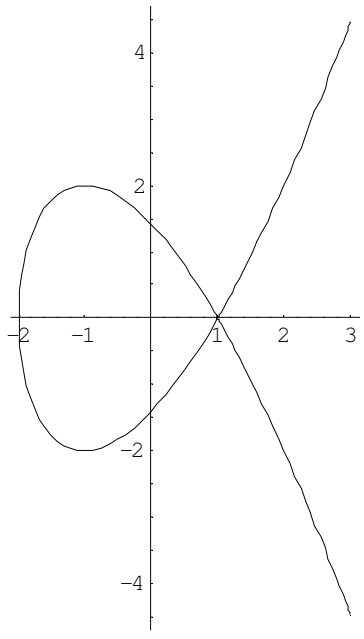

Figura 3.2: Curva $y^2 = x^3 - 3x + 2$

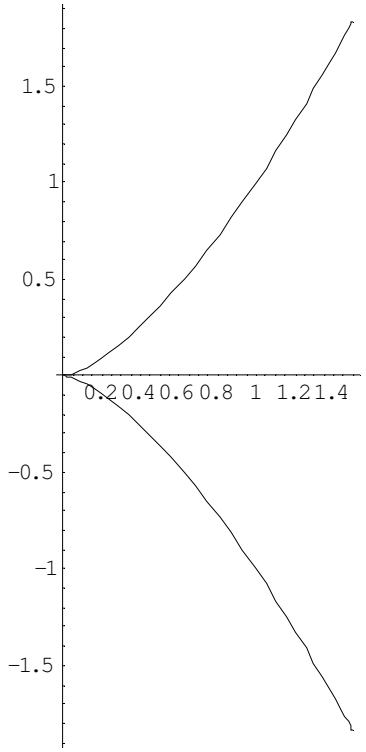

Figura 3.3: Curva $y^2 = x^3$

Char(K) = 2

Se la caratteristica del campo K è uguale a 2 l'equazione di Weierstrass (3.2) non è applicabile in quanto è singolare. Se infatti definisco $F(x,y) = y^2 - x^3 - ax - b$ si ha che $\frac{\partial F}{\partial y} = 2y = 0$ poiché in un campo con caratteristica uguale a 2 vale 2 = 0. Sia inoltre x_0 una radice di $\frac{\partial F}{\partial x} = -3x^2 - a$ e sia y_0 la radice quadrata di $x_0^3 + ax_0 + b$, allora il punto (x_0, y_0) appartiene alla curva ed in esso le due derivate parziali si annullano, da cui la singolarità.

Riconsideriamo quindi l'Equazione Generalizzata di Weierstrass (3.1) e distinguiamo due casi.

$a_1 \neq 0$

In tal caso la trasformazione ammissibile di variabili

$$(x, y) \rightarrow (a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3})$$

trasforma la (3.1) in:

$$y^2 + xy = x^3 + ax^2 + b \tag{3.4}$$

dove $a,b \in K$.

Tale curva è detta *non-supersingolare* (vedi Par. 3.6) e ha discriminante $\Delta = b$.

$a_1 = 0$

La seguente trasformazione ammissibile di variabili

$$(x, y) \rightarrow (x + a_2, y)$$

trasforma la curva E in

$$y^2 + cy = x^3 + ax + b \tag{3.5}$$

dove $a,b,c \in K$.

Tale curva è detta *supersingolare* e ha discriminante $\Delta = c^4$.

3.2 La legge di Gruppo

Una caratteristica molto importante dell'insieme dei punti di una curva ellittica $E(K)$, è legata alla possibilità di definire una struttura algebrica su tale insieme. In particolare si può definire una operazione di *addizione* "+" sui punti di $E(K)$ in modo che $(E(K), +)$ costituisca un gruppo (additivo) abeliano. Proprio tale gruppo viene utilizzato per costruire i crittosistemi basati su curve ellittiche.

La regola di addizione è detta *regola tangente-corda* ed ha una spiegazione geometrica molto intuitiva. Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ due punti distinti della curva E. Allora la somma $R = P + Q$ è così definita. Traccio una retta che contenga P e Q. Tale retta interseca la curva in un terzo punto R'. Individuo il punto simmetrico a R' rispetto all'asse delle x (ossia cambio il segno della coordinata y) e ottengo così R. Il procedimento è illustrato in Figura 3.4, nel caso $K = \mathbb{R}$.

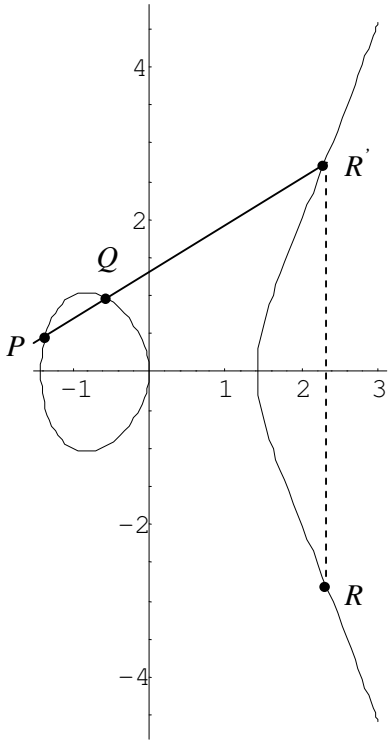

Figura 3.4: Addizione: $P + Q = R$

La regola di raddoppio (sommo il punto P a se stesso), invece, è come segue. Traccio la retta tangente alla curva in P. Tale retta interseca la curva in un secondo punto R'. Individuo il punto simmetrico a R' rispetto all'asse delle x e ottengo così $R = P + P$. La Figura 3.5 rappresenta il raddoppio del punto P.

Le formule algebriche relative alla cosiddetta *legge di Gruppo* sono consultabili in Appendice B. L'aspetto rilevante comunque è che i punti di una curva ellittica E, insieme all'operazione di addizione appena definita, formano un gruppo (additivo) abeliano, con Θ elemento identità.

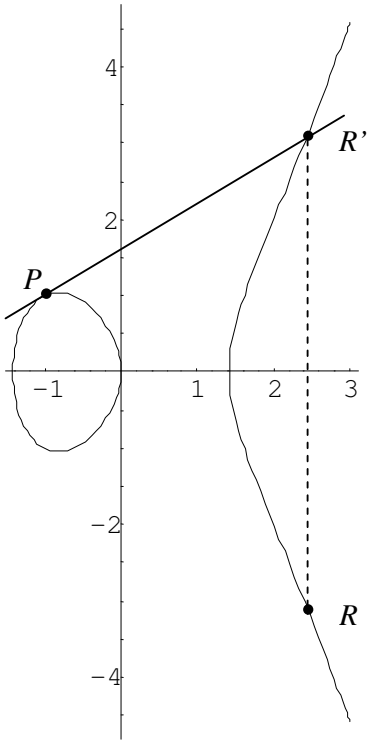

Figura 3.5: Raddoppio: $P + P = R$

3.3 Rappresentazione dei Punti

Adesso che abbiamo definito la necessaria struttura algebrica su $E(K)$, facciamo una considerazione di carattere implementativo. Come si può vedere in Appendice B, le formule di addizione presentate nel paragrafo precedente richiedono operazioni di inversione e moltiplicazione. Il calcolo dell'inverso, in modo particolare, è operazione molto costosa e quindi sarebbe meglio non farne utilizzo. A tale proposito è conveniente sostituire le coordinate normali (affini) con le *coordinate proiettive*, poiché queste ultime non richiedono il calcolo dell'inverso. Introduciamo ora i concetti basilari di tali coordinate.

3.3.1 Coordinate Proiettive

Si abbia il campo K, che potrà essere $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ oppure Z_p.

Definizione 3.4 Si definisce *spazio affine* $A^2(K)$ su K l'insieme:

$$A^2(K) = \{(x, y) | x, y \in K\}$$

Ad esempio il piano Euclideo $\mathbb{R}^2$ viene anche chiamato "spazio affine bi-dimensionale su $\mathbb{R}$", $A^2(\mathbb{R})$, e la coppia (x,y) è detta *punto affine*.
Consideriamo ora l'insieme L definito da :

$$L = \{(X,Y,Z) \mid X,Y,Z \in K \wedge X,Y,Z \text{ non tutti nulli}\}$$

e definiamo una relazione di equivalenza $\sim$ su L tale che:

$$(X,Y,Z) \sim (X',Y',Z') \Leftrightarrow (X,Y,Z) = \lambda(X',Y',Z') \text{ con } \lambda \in K^*$$

La classe di equivalenza della terna (X,Y,Z) viene chiamata *punto proiettivo* e visto che dipende dai rapporti fra X, Y e Z si indica con $(X:Y:Z)$, anche per evitare confusioni con la rappresentazione dei punti affini. Ad esempio le terne (4,2,6) e (12,6,18) sono equivalenti (con $\lambda = 3$) e appartengono alla classe di equivalenza (2:1:3).

Definizione 3.5 Si definisce *piano proiettivo* $P^2(K)$ su K l'insieme $L / \sim$ delle classi di equivalenza rispetto $\sim$, o più brevemente l'insieme di tutti i punti proiettivi. Pertanto si scrive:

$$P^2(K) = \{(X:Y:Z) \mid X,Y,Z \in K \wedge X,Y,Z \text{ non tutti nulli}\}$$

Analogamente si definisce *linea proiettiva* $P^1(K)$ su K l'insieme delle classi di equivalenza di coppie $(X,Y) \neq (0,0)$ con $X,Y \in K$ tali che:

$$(X,Y) \sim (X',Y') \Leftrightarrow (X,Y) = \lambda(X',Y) \text{con } \lambda \in K^*$$

Consideriamo una generica classe di equivalenza di $P^2(K)$ e distinguiamo a seconda del valore Z. Se $Z \neq 0$ il generico punto (X,Y,Z) è equivalente (basta prendere $\lambda = 1/Z$) a $(X/Z, Y/Z, 1)$. Se quindi imposto $X'=X/Z$ e $Y'=Y/Z$ la corrispondente classe di equivalenza è rappresentata da $(X',Y',1)$; tali classi vengono dette *punti finiti* di $P^2(K)$ ed il loro insieme si identifica con lo spazio affine $A^2(K)$. Se infatti considero il punto dello spazio affine (X',Y') esso può essere *collocato* nel piano proiettivo semplicemente impostando la coordinata Z uguale a 1. Le generiche terne $(X,Y,1)$ perciò sono in corrispondenza biunivoca con i punti dello spazio affine $A^2(K)$. Se invece $Z = 0$ le classi di equivalenza che si ottengono sono del tipo $(X:Y:0)$ con X e Y non entrambi nulli: il loro insieme si identifica con $P^1(K)$ e viene chiamato *linea all'infinito*.

In conclusione si ha $P^2(K) = A^2(K) \cup P^1(K)$.

Quelle viste sin qui sono chiamate *coordinate proiettive standard*. Nell'ambito dell' ECC viene considerata una forma generalizzata di coordinata proiettiva, ossia la *coordinata proiettiva pesata*. Essa può essere così definita:

Considerato l'insieme L, definito sopra, e gli interi positivi c e d, si costruisce la relazione di equivalenza $\sim$ su L

$$(X,Y,Z) \sim (X',Y',Z') \Leftrightarrow X = \lambda^c X',\ Y = \lambda^d Y'\ ,Z = \lambda,Z' \text{ con } \lambda \in K^*$$

3.3.2 Legge di Gruppo con Coordinate Proiettive

Vediamo ora come rappresentare una curva ellittica per mezzo di coordinate proiettive. Consideriamo l'Equazione di Weierstrass $y^2 = x^3 + ax + b$ e ne deriviamo la sua versione proiettiva. Per fare questo sostituiamo le coordinate affini (x,y) con quelle proiettive(X,Y,Z):

$$x = X/Z, \quad y = Y/Z$$

Ricordiamo infatti che vi è corrispondenza biunivoca fra i punti affini e i punti proiettivi con $Z \neq 0$. Si ottiene quindi

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

Tale equazione è soddisfatta da tutti i punti proiettivi con $Z \neq 0$ tali che i corrispondenti punti affini soddisfano la (3.2). Ponendo $Z = 0$, l'equazione proiettiva diventa $X^3 = 0$ e quindi $X = 0$. L'unica classe di equivalenza con X e Z entrambi nulli è (0:1:0). Essa è proprio il *punto all'infinito* Θ e rappresenta l'intersezione fra l'asse delle y e la *linea all'infinito*.

Nella pratica comunque si utilizzano le coordinate proiettive pesate, in quanto più efficienti, e in particolare le *coordinate Jacobiane* ($c = 2$ e $d = 3$). A titolo di esempio si consideri la regola del raddoppio a partire dall'equazione di Weierstrass (3.2). Se quindi $P = (X_1: Y_1: Z_1)$ è un punto appartenente alla curva E, si può scrivere $P = (X_1/Z_1^2 : Y_1/Z_1^3 : 1)$ e semplici calcoli consentono di ottenere le coordinate proiettive del punto $(X_3: Y_3: Z_3) = P + P = 2P$:

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned}$$

Nella Tabella 3.1 viene riportato il conteggio delle operazioni elementari per eseguire addizione e raddoppio, relativamente all'equazione $y^2 = x^3 - 3x + b$. La notazione $C_1 + C_2 \to C_3$ significa che i punti da sommare sono espressi nelle coordinate C_1 e C_2 e il risultato è espresso nelle coordinate C_3. In particolare le lettere A, P e J indicano rispettivamente coordinate affini, proiettive standard e Jacobiane. Le lettere I, M, S invece indicano il tipo di operazione elementare da eseguire: I = *inversione*, M=*moltiplicazione*, S = *elevazione al quadrato*.

Raddoppio $2P$		Addizione $P + Q$	
$2A \to A$	$1I, 2M, 2S$	$A + A \to A$	$1I, 2M, 1S$
$2P \to P$	$7M, 3S$	$P + P \to P$	$12\ M, 2\ S$
$2J \to J$	$4M, 4S$	$J + J \to J$	$12\ M, 4\ S$

Tabella 3.1: Conteggio delle operazioni per $y^2 = x^3 - 3x + b$

Come si può vedere dalla Tabella 3.1, le coordinate Jacobiane forniscono il metodo più veloce per calcolare il raddoppio di un punto P (4 moltiplicazioni e 4 elevazioni al quadrato). L'implementazione più veloce dell'addizione $P + Q$, invece, si ottiene con un sistema ibrido di coordinate affini-jacobiane [20].

3.4 Moltiplicazione Scalare

L'operazione di addizione può essere ripetuta più volte. In particolare se P è un punto di una curva ellittica e k è un intero positivo, si indica con kP la somma

$$Q = kP = \underbrace{P + P + \cdots + P}_{k \text{ volte}}$$

e viene chiamata *moltiplicazione scalare*. Se invece k è negativo, il simbolo kP rappresenta la somma $-k(-P)$. Se $k = 0$ si scrive $0\ P = \Theta$. Per valori grandi di k risulta inefficiente sommare ripetutamente P a se stesso, è molto più veloce effettuare i cosiddetti *raddoppi successivi* (*successive doubling*). Ad esempio per ottenere $19P$, si calcola:

$2P$, $4P = 2P + 2P$, $8P = 4P + 4P$, $16P = 8P + 8P$, $\mathbf{19}\boldsymbol{P} = 16P + 2P + P$

L'algoritmo che si ottiene è il seguente:

Algoritmo 3.1 Moltiplicazione scalare per raddoppi successivi

Sia k un intero positivo e P un punto su una curva ellittica. I seguenti passaggi consentono di calcolare kP.

1. Imposta $a = k$, $B = \Theta$, $C = P$
2. Se a è pari, imposta $a = a/2$ e $B = B$, $C = 2C$
3. Se a è dispari, imposta $a = a - 1$, $B = B + C$, $C = C$
4. Se $a \neq 0$ torna allo step 2.
5. Output: $B = kP$

Se t è la *bitlength* di k, tale algoritmo effettua t raddoppi e tante addizioni quanti sono gli 1 nella rappresentazione binaria di k (al più t).

Oltre a questo comunque, vi sono altri algoritmi per la moltiplicazione scalare, i quali sfruttano particolari proprietà delle curve ellittiche. A riguardo si veda [20].

I protocolli dell' ECC si basano principalmente sulla moltiplicazione scalare, e soprattutto sulla difficoltà di calcolare k, conoscendo P e kP. Questo è infatti il problema del Logaritmo Discreto su Curva Ellittica (ECDLP) e verrà presentato nel capitolo successivo.

3.5 Ordine di Gruppo

Dal punto di vista crittografico, un parametro molto importante è il numero di punti che appartengono ad una generica curva ellittica costruita sul campo finito $GF(q)$. Tale grandezza si indica con $\#E(GF(q))$ o semplicemente $\#E$, ed è detta *ordine della curva* $E/GF(q)$.

Consideriamo ad esempio la curva $E/GF(5)$ descritta da $y^2 = x^3 + x + 1$. Il conteggio dei punti è presto fatto: elenco tutti i possibili valori che può assumere x, calcolo i valori $x^3 + x + 1 \pmod 5$ e successivamente ricavo, se esiste, la radice quadrata.

x	$x^3 + x + 1 (\mathrm{mod}\, 5)$	y	*Punti*
0	1	± 1	(0,1), (0,4)
1	3		
2	1	± 1	(2,1), (2,4)
3	1	± 1	(3,1), (3,4)
4	4	± 2	(4,2), (4,,3)

Considerando anche Θ, si ottiene # $E(GF(5)) = 9$.

In generale comunque, per valori molto grandi di q, questo approccio non è praticabile. Una stima molto approssimativa ci consente di affermare che in $E(GF(q))$ vi sono al più $2q + 1$ punti. Infatti nell'equazione della curva si può assegnare ad x ogni elemento di $GF(q)$ e per ognuno di essi vi sono al più due y corrispondenti. Se aggiungiamo il punto all'infinito Θ, si ottiene proprio $2q + 1$.

Il seguente Teorema, comunque, fornisce dei limiti più precisi per #E.

Teorema 3.2 (Hasse)

Sia E una ellittica definita sul campo finito $GF(q)$. Allora l'ordine di E soddisfa:

$$q + 1 - 2\sqrt{q} \leq \#E(GF(q)) \leq q + 1 + 2\sqrt{q}$$

L'intervallo $[q + 1 - 2\sqrt{q},\, q + 1 + 2\sqrt{q}]$ è detto *intervallo di Hasse*.

Una formulazione alternativa del Teorema afferma che l'ordine di E soddisfa:

$$\#E(GF(q)) = q + 1 - t$$

dove $|t| \leq 2\sqrt{q}$; t è detta *traccia* di E.

Sebbene il Teorema di Hasse fornisca un intervallo di valori possibili per #E, i crittosistemi basati su Curva Ellittica devono selezionare opportunamente la curva ellittica, e questo richiede la conoscenza esatta dell'ordine di E.

3.5.1 Determinare l'ordine di una Curva

Vediamo alcuni algoritmi per effettuare il calcolo di #*E*.

Metodo *ingenuo* (*naive*)

Sia data una curva ellittica $y^2 = x^3 + ax + b$ definita su F_p; per x che assume tutti i possibili valori in F_p, l'equazione $y^2 = x^3 + ax + b$ avrà rispettivamente:

i) 2 soluzioni se $x^3 + ax + b$ è un residuo quadratico in F_p

ii) 1 soluzione se $x^3 + ax + b$ è divisibile per p

iii) 0 soluzioni se $x^3 + ax + b$ non è un residuo quadratico in F_p

Tali osservazioni possono essere formalizzate con il simbolo di Legendre e portano alla seguente formula:

$$\#E(F_p) = 1 + \sum_{x \in F_p} \left(1 + \left(\frac{x^2 + ax + b}{p} \right) \right) = 1 + q + \sum_{x \in F_p} \left(\frac{x^2 + ax + b}{p} \right)$$

Questo metodo, talvolta chiamato di Lang-Trotter, ha un *runtime* $O(p^{1+\epsilon})$ che è esponenziale in $\log p$. Infatti funziona bene per campi piccoli, ma è inutilizzabile per grandi valori di p. A puro titolo di esempio, in Tabella 3.2 sono riportati alcuni valori di $\#E(F_p)$ ottenuti usando la formula appena vista. L'ultima colonna consente di verificare che $\#E(F_p)$ è sempre interno all'intervallo di Hasse. Per tali calcoli si è usato il software *Mathematica*.

Curva Ellittica E	p	$\#E(F_p)$	Intervallo di Hasse
$y^2 = x^3 + x + 1$	991	952	[930, 1054]
	9973	9924	[9776, 10172]
	104729	105078	[104084, 105376]
$y^2 = x^3 + 7x + 11$	200117	200655	[199224, 201012]
	351811	351378	[350626, 352998]
	437149	437778	[435828, 438472]
$y^2 = x^3 + 2x + 3$	41947	41680	[41540, 42356]
	135899	135960	[135164, 136636]
	200929	201064	[200034, 201826]

Tabella 3.2 Calcolo di $\#E(F_p)$ per valori primi di p

Algoritmo di Schoof

Nel 1985 Schoof presentò il primo algoritmo *polynomial-time* per calcolare l'ordine $\#E(F_q)$ di una qualsiasi curva ellittica *E*. Il suo *running-time* è pari a $O((\log p)^8)$. Essendo comunque inefficiente per valori di interesse pratico di *q*, l'algoritmo venne successivamente ottimizzato da Atkin e Elkies, ottenendo il cosiddetto *algoritmo Schoof-Elkies-Atkin* (SEA). Ad oggi SEA è il migliore algoritmo per contare i punti di un'arbitraria curva ellittica su *campi primi*.

A titolo indicativo, vediamo l'approccio seguito da Schoof. Sia E/F_q la curva ellittica data da $y^2 = x^3 + ax + b$. Dal teorema di Hasse si ha che:

$$\#E(F_q) = q + 1 - t \ , \qquad \text{con} \quad |t| \leq 2\sqrt{q}$$

Sia $S = \{2,3,5,\ldots,L\}$ un insieme di numeri primi tali che

$$\prod_{s \in S} s \geq 4\sqrt{q}$$

Calcoliamo $t \bmod s$ per ogni $s \in S$, quindi dal Teorema del Resto Cinese determiniamo $t \bmod \prod s$. Poiché $|t| \leq 2\sqrt{q}$ questo ci consente di determinare univocamente *t*. Una descrizione più dettagliata dell'algoritmo è fornita in [33].

Algoritmo di Satoh

Nel 1999 Satoh propose un metodo completamente nuovo per calcolare l'ordine di una curva ellittica definita su un campo finito con caratteristica piccola. Alcune varianti del metodo di Satoh, inclusi gli algoritmi *Satoh-Skiernaa-Taguchi* (STT) e *Arithmetic Geometric Mean* (AGM), sono estremamente veloci nel caso di campi binari e in pochi secondi possono trovare curve ellittiche di interesse crittografico definite su $GF(2^{163})$. L'algoritmo di Satoh viene trattato in modo approfondito in [2].

3.5.2 Ordine dei Punti

Altra grandezza rilevante nell'ambito dell'ECC è *l'ordine di un punto*. Data una curva ellittica *E/K* e il punto $P \in E(K)$, si definisce *ordine di P* il più piccolo intero positivo *m*, se esiste, tale che $mP = \Theta$. Se tale intero non esiste si dice che il punto *P* ha *ordine infinito*. I *punti torsione* sono punti aventi ordine finito. Più precisamente:

Definizione 3.4 Data una curva *E/K* e l'intero positivo *m*, il punto $P \in E(\overline{K})$ è un *punto m-torsione* se $mP = \Theta$. Da notare che *m* non deve essere necessariamente l'ordine di *P*, può essere anche un suo multiplo. Infatti se *P* ha ordine *n* allora vale $knP = \Theta$ per tutti gli interi *k*.

Si ricordi che il simbolo $\overline{K}$ indica la chiusura algebrica del campo *K*. L'insieme dei punti *m-torsione* ci consente di dare la seguente definizione:

Definizione 3.5 Data una curva *E/K* e l'intero positivo *m*, l'insieme

$$E[m] = \left\{P \in E(\overline{K}) \mid mP = \Theta\right\}$$

forma un gruppo chiamato *gruppo m-torsione*.

E' da sottolineare che *E*[*m*] contiene punti con coordinate in $\overline{K}$, non solamente in *K*. Dal Teorema di Lagrange (vedi Par. A.2.1) ricordiamo che l'ordine di un punto divide sempre l'ordine del gruppo. Quindi se la curva *E* è definita in un campo finito F_q allora tutti i punti di *E* sono *punti torsione*.
Un importante risultato che descrive la struttura dei *gruppi m-torsione* viene dal seguente teorema.

Teorema 3.4 Sia *p* la caratteristica del campo *K*, *n* un intero positivo e *E*/*K* una curva ellittica definita su *K*. Se *p* non divide *n* oppure vale 0, allora[4]:

$$E[n] \cong Z_n \oplus Z_n$$

Se invece $p \mid n$, si scrive $n = p^r l$ con *l* non divisibile da *p*, allora:

$$E[n] \cong Z_n \oplus Z_l \quad \text{oppure} \quad E[n] = Z_l \oplus Z_l$$

3.6 Curve speciali

Come si vedrà nel capitolo successivo, vi sono famiglie di curve ellittiche che si rivelano particolarmente vulnerabili dal punto di vista crittografico. E' opportuno quindi capire la natura di queste curve.

[4] Per una descrizione del simbolo di *somma diretta* $\oplus$ si veda il Par. A.2

3.6.1 Curve Supersingolari

Vediamo come il concetto di *gruppo torsione* ci consente introdurre la nozione di curva *supersingolare*. Dal teorema 3.4 notiamo che se $n = p$ allora $E[p]$ può essere isomorfico a Z_p oppure può degenerare nell'insieme banale.

Definizione 3.6 Sia p la caratteristica del campo F_q. La curva ellittica E/F_q è detta *ordinaria* se $E[p] \cong Z_p$ altrimenti *supersingolare* se $E[p] \cong 0$. In altre parole la curva è *supersingolare* se $E[p] = \{\Theta\}$.

Va notato che i termini *singolare* e *supersingolare*, quando applicati ad una curva ellittica, sono del tutto incorrelati.
E' possibile caratterizzare la supersingolarità di una curva ellittica E in relazione all'ordine $\#E$ della curva stessa. Vale infatti il seguente teorema

Teorema 3.5 Sia E una curva ellittica definita su F_q, dove q è una potenza del numero primo p. Dal teorema di Hasse, si può scrivere $t = q + 1 - \#E(F_q)$. Allora:

$$E \text{ è } supersingolare \Leftrightarrow t \equiv 0 \pmod{p}$$

che equivale ad affermare:

$$E \text{ è } supersingolare \Leftrightarrow \#E(F_q) \equiv 1 \pmod{p}$$

In altro modo si può dire che la curva E/F_q è *supersingolare* se la traccia t è divisibile per la caratteristica di F_q.
Le curve *supersingolari* sono caratterizzate dal fatto che in esse i calcoli sono più veloci, pertanto dal punto di vista crittografico esse sono considerate "*deboli*". Ciò nonostante una scelta opportuna di curve *supersingolari* consente di realizzare sistemi crittografici molto efficienti, basati sul *Weil pairing* [33].

3.6.2 Curve anomale

Per completezza citiamo altre due famiglie di curve.

Definizione 3.7 Una curva ellittica E definita sul campo primo F_p è detta *curva anomala* (*su campo primo*) se $\#E(F_p) = p$, quindi se la traccia t è uguale a 1.

Come si capirà nel capitolo seguente, tali curve sono considerate molto vulnerabili e quindi rigorosamente vietate in tutti gli standards ECC.

Definizione 3.8 Si definisce *curva di Koblitz*, o *curva binaria anomala*, una curva definita su $GF(2^m)$ la cui equazione ha coefficienti in $GF(2) = \{0,1\}$. Le curve di Koblitz sono due:

$$y^2 + xy = x^3 + 1$$
$$y^2 + xy = x^3 + x^2 + 1$$

Caratteristica principale di tali curve è che su di esse la moltiplicazione scalare non richiede i raddoppi di punto. Conseguentemente, studi di Gallant, Lambert,Vanstone e Wiener, Zuccherato, hanno mostrato che la risoluzione dell' ECDLP in curve di Koblitz su $GF(2^m)$ può essere velocizzato di un fattore $\sqrt{m}$. In ogni caso, anche con valori grandi di m ($m > 160$), questo miglioramento rimane insignificante rispetto al numero totale di calcoli necessari, pertanto non pregiudica la sicurezza di queste curve. Le curve di Koblitz sono state standardizzate nel FIPS 186-2 del NIST.

3.7 Weil *pairing*

Il Weil *pairing* (dall'inglese, *accoppiamento*) è uno strumento molto importante nello studio delle curve ellittiche. Nell'ambito dell' ECC è stato utilizzato sia per dimostrare importanti risultati che per attaccare il problema ECDLP.

Sia E una curva ellittica definita sul campo K e sia n un intero positivo non divisibile da char(K). Dal Teorema 3.4 si ha che $E[n] \cong Z_n \oplus Z_n$. Definisco

$$\mu_n = \{x \in \overline{K} \mid x^n = 1\}$$

come l'insieme delle radici n-esime dell'unità in $\overline{K}$. Poiché char(K) non divide n, l'equazione $x^n = 1$ non ha radici multiple, quindi ha n radici in $\overline{K}$. Quindi μ_n è ciclico di ordine n. In queste condizioni vale il seguente Teorema.

Teorema 3.6 (Weil *pairing*)

Sia E una curva ellittica definita sul campo K e sia n un intero positivo non divisibile da char(K). Si definisce il seguente accoppiamento (*pairing*):

$$e_n : E[n] \times E[n] \to \mu_n$$

detto *Weil pairing*, che soddisfa la seguenti proprietà:

1. e_n è bilineare in ogni variabile:
$$e_n(P + P', Q) = e_n(P, Q) e_n(P', Q)$$
$$e_n(P, Q + Q') = e_n(P, Q) e_n(P, Q')$$
2. e_n è non degenerativo: se per qualche $P \in E[n]$ si ha $e_n(P, Q) = 1$ per ogni $Q \in E[n]$ allora $P = \Theta$.
3. $e_n(P, P) = 1$ per tutti i $P \in E[n]$
4. $e_n(P, Q) = 1 / e_n(Q, P)$ per tutti i $P, Q \in E[n]$

Il Weil *pairing* quindi fornisce una mappatura fra coppie di punti della curva ellittica E/K e la chiusura algebrica $\overline{K}$. Più precisamente stabilisce un isomorfismo fra coppie di punti *n-torsione* e l'insieme delle radici *n*-esime dell'unità in $\overline{K}$.
Conseguenza importante è il seguente Corollario

Corollario 3.1 Se $E[n] \subseteq E(K)$ allora $\mu_n \subseteq K$

In [33] si può trovare un approfondimento rigoroso dell'argomento.
Diamo ora la seguente definizione:

Definizione 3.7 Sia E una curva ellittica definita su F_q e sia n un intero positivo. Il più piccolo intero k tale che $E[n] \subseteq E(F_{q^k})$ è detto *grado MOV*, relativamente ad n, e si indica con $\deg_{MOV}(n)$.

Il grado MOV $\deg_{MOV}(n)$, quindi, esprime la dimensione del più piccolo campo estensione di F_q che contiene tutti i punti *n*-torsione. Ricordando la definizione di chiusura algebrica per campi finiti (Par. A.2), si capisce che tale valore esiste

sempre. Come apparirà evidente nel capitolo successivo, il grado MOV rappresenta un parametro di sicurezza della curva ellittica. La sigla MOV deriva da Menezes, Okamoto e Vanstone, i tre studiosi che hanno sfruttato il Weil *pairing* per ideare un nuovo attacco all'ECDLP (vedi Par. 4.3.1).

Capitolo 4

Logaritmo Discreto su Curve Ellittiche

4.1 Introduzione

Come già anticipato nel Par. 2.4, la sicurezza dell'ECC si basa sulla difficoltà del problema del Logaritmo Discreto su curva ellittica (ECDLP). Diamo quindi la seguente

Definizione 4.1 Il *problema del logaritmo discreto su EC* (ECDLP) è: data una curva ellittica E definita sul campo finito F_q, un punto $P \in E(F_q)$ di ordine n e un punto $Q \in \langle P \rangle$, trovare l'intero $k \in [0,\ldots, n-1]$ tale che $Q = kP$. L'intero k è chiamato *logaritmo discreto di Q in base P* e si indica con $k = \log_P Q$.

I parametri delle curve ellittiche per crittosistemi, quindi, devono essere scelti accuratamente in modo da resistere ai noti *attacchi* al ECDLP. L'attacco più ovvio ed ingenuo (*naive*) al ECDLP consiste nella *ricerca esaustiva*, con la quale viene calcolata la sequenza P, $2P$, $3P$,… fino a che non si ottiene Q. Il *running time* è n passi nel caso peggiore ed $n/2$ passi nel caso medio. Si capisce bene che per contrastare la ricerca esaustiva è sufficiente considerare valori di n molto grandi (per esempio $n \geq 2^{80}$).

Data l'attuale potenza elaborativa dei calcolatori il problema ECDLP è considerato *intrattabile*, previa una selezione opportuna della curva ellittica. Ribadiamo comunque, che manca una prova matematica della sua *intrattabilità*, infatti nessuno ha dimostrato che non vi può essere un algoritmo efficiente che possa risolvere l'ECDLP. Ciononostante l'intrattabilità dell'ECDLP trova sostegno nei numerosi anni in cui ricercatori e matematici hanno cercato, senza riuscirvi, un algoritmo *general-purpose* di tipo *sub-esponenziale*.

Anche per risolvere l'ECDLP vi sono due tipi di algoritmi: i *special-purpose* che si utilizzano quando la curva ellittica presenta certe caratteristiche ed i *general-purpose* che hanno tempi di esecuzione dipendenti solo dalla dimensione della curva stessa.

Di seguito vengono presentati i principali attacchi *general-purpose*, e due tipi di attacchi *special-purpose*. In alcuni casi, oltre ad illustrare le caratteristiche del singolo attacco, verrà riportata anche una descrizione più dettagliata della parte algoritmica. Essa verrà indicata con il titolo Analisi dell'Algoritmo e terminerà con il simbolo ◙. Ai fini della comprensione generale dell'argomento la lettura di tale descrizione non è strettamente necessaria, per quanto consigliata.

4.2 Attacchi generali

4.2.1 Metodo Pohlig-Hellman

L'algoritmo Pohlig-Hellman riduce il calcolo di $k = \log_P Q$ nel calcolo del logaritmo discreto nei sottogruppi di $\langle P \rangle$ di ordine pari ai fattori primi di n dove, lo ricordiamo, n è l'ordine di P. Sia quindi $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$ la fattorizzazione prima di n. La strategia seguita dall'algoritmo consiste nel calcolare $k_i = k \bmod p_i^{e_i}$ per ogni $i \in [1, r]$ e risolvere il sistema di congruenze

$$\begin{cases} k \equiv k_1 \pmod{p_1^{e_1}} \\ k \equiv k_2 \pmod{p_2^{e_2}} \\ \quad\vdots \\ k \equiv k_r \pmod{p_r^{e_r}} \end{cases}$$

Il Teorema del Resto Cinese garantisce l'unicità della soluzione $k \bmod n$.

Analisi dell'Algoritmo

Vediamo ora come il calcolo di ogni k_i si riduce al calcolo di e_i logaritmi discreti nel sottogruppo di $\langle P \rangle$ di ordine p_i. Per semplificare la notazione scriviamo e ed p ad indicare il generico e_i e p_i. Scriviamo ora k in base p:

$k = z_0 + z_1 p + z_2 p^2 + \cdots + z_{e-1} p^{e-1} + z_e p^e + \cdots$ che può essere scritto come

$$k = z_0 + z_1 p + \cdots + z_{e-1} p^{e-1} + p^e (z_e + \cdots)$$

dove gli $z_i \in [0, p-1]$. Quindi $k \bmod p^e = z_0 + z_1 p + \cdots + z_{e-1} p^{e-1}$. Vediamo ora come calcolare i coefficienti z_0, z_1, …, z_{e-1}.

Definisco $P_0 = (n/p)P$ e $Q_0 = (n/p)Q$, quindi l'ordine di P_0 è p. Esplicitiamo $Q = kP$ e ricordando che l'ordine di P è n (quindi $nP = \Theta$) otteniamo:

$$Q_0 = \frac{n}{p}Q = \frac{n}{p}\left[(z_0 + z_1 p + \cdots)P\right] = \frac{n}{p} z_o P + (z_1 + \cdots)nP = z_0 P_0$$

Pertanto z_0 si ottiene risolvendo un istanza ECDLP in $\langle P_0 \rangle$.

Successivamente si calcola $Q_1 = (n/p^2)(Q - z_0 P)$. Esplicitando ancora Q, si ha:

$$Q_1 = \frac{n}{p^2}(kP - z_0 P) = (k - z_0)\left(\frac{n}{p^2}P\right) = \left[z_1 + p(z_2 + \cdots)\right]\left(\frac{n}{p}P\right)$$
$$= z_1 \frac{n}{p}P + (z_2 + \cdots)nP = z_1 P_0$$

Anche z_1, quindi, si ottiene risolvendo un istanza ECDLP in $\langle P_0 \rangle$.

Il procedimento continua in questo modo per i rimanenti coefficienti, pertanto il generico z_t si ottiene calcolando $Q_t = \frac{n}{p^{t+1}}\left(Q - z_0 P - z_1 pP - \cdots - z_{t-1} p^{t-1} P\right)$ e risolvendo il logaritmo $Q_t = z_t P_0$. ◙

Si capisce che per resistere all'*attacco Pohlig-Hellman* i sottogruppi generati dai fattori primi di n devono essere sufficientemente grandi e quindi l'ordine n di P deve essere divisibile per un grande primo. Per il resto del Par. 4.2 si suppone che n sia primo.

4.2.2 Metodo Baby-Step Giant Step

Tale metodo, sviluppato da Shanks nel 1973, ha un *running time* pari a $O(\sqrt{n})$ e richiede un occupazione di memoria $O(\sqrt{n})$ dove n è l'ordine del punto P.

Analisi dell'Algoritmo

L'algoritmo funziona così:

1. Si individua un intero $m \geq \sqrt{n}$ e si calcola mP.
2. Si memorizza un array di m valori iP, con $i \in [0, m-1]$

3. Si calcolano i punti $Q - jmP$, con $j \in [0, m-1]$, finché non si trova una corrispondenza (*matching*) con un elemento dell'array creato al punto 2.
4. Se $iP = Q - jmP$ allora si ha $Q = kP$ con $k \equiv i + jm \pmod{n}$

La procedura appena descritta si basa sulle seguenti osservazioni. Poiché $m^2 \geq n$ il valore k che stiamo cercando soddisfa $0 \leq k < m^2$. Pertanto si può scrivere $k = k_0 + mk_1$ con $k_0 = k \bmod m$ e $k_1 = (k - k_0)/m$. Sia k_0 che k_1 sono interi minori di *m*. Quando, nel punto 3, vale $i = k_0$ e $j = k_1$ si ottiene:

$$Q - k_1 mP = kP - k_1 mP = (k - mk_1)P = k_0 P$$

e quindi si ha il *matching*.

Il punto *iP* viene calcolato aggiungendo *P* (il *baby step*) a (*i* – 1)*P*, mentre il punto *Q* – *jmP* viene calcolato sommando –*mP* (il *giant step*) a *Q* – (*j* – 1)*mP*. ◙

A causa dell'enorme richiesta di memoria (per *n* molto grandi), tale algoritmo viene raramente utilizzato.

4.2.3 Metodo Pollard ρ

Questo algoritmo venne introdotto nel 1978. Ha un *running time* confrontabile a quello del metodo *Baby-step Giant-step*, ma in compenso richiede pochissima memoria. L'idea di base è la stessa ma mentre il *Baby-step Giant-step* è deterministico il *Pollard* ρ è di tipo aleatorio. Il suo funzionamento infatti si basa sul cosiddetto "*Paradosso del Compleanno*": il numero minimo di persone che devo considerare affinché due di esse compiano gli anni nello stesso giorno, con probabilità del 50%, è 24 (valore approssimabile a $\sqrt{\pi 365/2}$). Più in generale, ogniqualvolta si selezionano a caso degli elementi da un insieme di dimensione *n*, è sufficiente selezionarne $O(\sqrt{n})$ per avere una probabilità del 50% di selezionare lo stesso elemento due volte (di avere cioè *collisione*).

Analisi dell'Algoritmo

Si abbia sempre da calcolare il logaritmo $k = \log_P Q$, con *n* ordine di *P*.

L'algoritmo considera una *funzione iterativa* $f : \langle P \rangle \to \langle P \rangle$ dal comportamento *pseudo-casuale*. Inizia con un punto casuale P_0 e calcola le iterazioni

$P_{i+1} = f(P_i)$. Poiché $\langle P \rangle$ è un insieme finito, ad un certo punto ci sarà *collisione*, ossia si avranno due indici $t < j$ tali che $P_t = P_j$. Per come è costruita la sequenza, inoltre, si ha $P_{t+l} = P_{j+l}$ per ogni $l > 0$. Ciò significa che la sequenza P_i, dopo la collisione, diventa periodica di periodo $s = j - t$, ossia

$$P_i = P_{i-s} \qquad \text{per tutti gli } i \geq t + s \qquad (4.1)$$

La rappresentazione grafica di tale processo ricorda la lettera greca ρ (da cui il nome di tale algoritmo). Il valore t è chiamato *lunghezza della coda*, mentre s è detto *lunghezza del ciclo*. Se la funzione f è sufficientemente casuale, si avrà collisione in un tempo $O(\sqrt{n})$. Un implementazione banale memorizzerebbe tutti i punti P_i fino ad ottenere *collisione*, ma questo richiederebbe un occupazione $O(\sqrt{n})$, proprio come il *Baby-step Giant-step*. L'algoritmo di Pollard invece utilizza il Metodo di *cycle-finding* di Floyd: si evita lo spreco di memoria, al prezzo di qualche calcolo in più. L'idea di Floyd consiste nel calcolare le coppie (P_i,P_{2i}) con $i = 1,2, \ldots$ fino a quando si ottiene il *matching* $P_i = P_{2i}$. Dopo aver calcolato una nuova coppia la precedente viene eliminata: in tal modo l'utilizzo di memoria è minimo. Il numero i di coppie da calcolare prima di avere *matching* è tale che $t \leq i \leq t + s$. In tale intervallo infatti esiste sicuramente un i che soddisfa la (4.1) ed è multiplo di s, pertanto i e $2i$ differiscono per un multiplo di s e quindi $P_i = P_{2i}$. Di conseguenza il numero di passi per trovare un *matching* è al più un multiplo costante di $\sqrt{n}$.

Vediamo ora come scegliere la *funzione iterativa* f. Essa, oltre a generare una sequenza pseudo-casuale, deve fornire informazioni utili per trovare k in caso di *matching*. Un modo per ottenere questo consiste nel partizionare $\langle P \rangle$ in L sottoinsiemi disgiunti dalla dimensione confrontabile $\{S_1, S_2, \ldots, S_L\}$ (valori tipici di L sono 16 e 32). Si scelgono $2L$ interi casuali a_i e b_i mod n e si definisce

$$M_i = a_i P + b_i Q$$

Infine si definisce la funzione f:

$$f(G) = G + M_i \quad \text{se} \quad G_i \in S_i$$

Intuitivamente si può pensare ad f come ad una passeggiata casuale (*random walk*) in $\langle P \rangle$ con passi rappresentati dai valori M_i. Per iniziare la *random walk* si

scelgono due interi casuali a_0 e b_0 e si definisce il punto iniziale $P_0 = a_0 P + b_0 Q$. Per come è costruita la sequenza, quindi, il generico punto P_i sarà sempre esprimibile nella forma $P_i = u_i P + v_i Q$. La presenza di un *matching* $P_i = P_j$ significa $u_i P + v_i Q = u_j P + v_j Q$ e quindi $(u_i - u_j)P = (v_j - v_i)Q = (v_j - v_i)kP$. Pertanto si ha $(u_i - u_j) \equiv (v_j - v_i)k \pmod n$ e il valore k è presto calcolato:

$$k = (u_i - u_j)(v_j - v_i)^{-1} \bmod n \qquad \blacksquare$$

L'algoritmo *Pollard* ρ ha un *running time* atteso di circa $\sqrt{\pi n/2}$ *steps* (dove uno *step* corrisponde ad una somma EC) e ad oggi è considerato il metodo più veloce per risolvere l'ECDLP. Nel 1999 Van Oorschot e Wiener hanno proposto una versione parallelizzata su r processori, ottenendo così un *running time* pari a $\left(\sqrt{\pi n/2}\right)/r$ [20]. L'esecuzione in parallelo, quindi, ha aumentato la velocità di un fattore r, ma la complessità è rimasta di tipo *fully-exponential*. Va ricordato infatti (vedi Par.A.4), che la complessità è legata al numero di bits $O(\log n)$ che esprimono l'input n, pertanto $\sqrt{n} = n^{1/2} = 2^{(\log n)/2}$ è esponenziale in $\log n$.

4.2.4 Metodo index-calculus

Nel Par.2.2.4 si è visto che gli algoritmi di tipo *index-calculus* per risolvere il DLP nel gruppo moltiplicativo F_q^* di un campo finito F_q richiedono un tempo sub-esponenziale. D'altro canto, nel paragrafo precedente, si è affermato che l'algoritmo più veloce per risolvere l'ECDLP ha *running time fully-exponential*. E' legittimo chiedersi, quindi, se esistono algoritmi di tipo *index-calculus* che risolvano l'ECDLP. Tale interrogativo se lo pose lo stesso Miller nel suo articolo del 1985 con il quale introdusse l'ECC. Egli osservò che a differenza dei campi finiti F_p o F_{2^m} dove è facile costruire la *base di fattori S*, nell'insieme $E(F_p)$ tale operazione risulta meno immediata. La soluzione più naturale consiste nell'*innalzare* (*lift*) la curva E/F_p nella curva $\tilde{E}$ definita sull'insieme dei razionali $\mathbb{Q}$ e nel costruire la *base di fattori S* con i punti P_i di E/F_p tali che i corrispondenti punti *lifted* $\tilde{P}_i$ siano linearmente indipendenti in $\tilde{E}(\mathbb{Q})$ e abbiano un peso (*height*) piccolo. Il *peso* di un punto $\tilde{P} \in \tilde{E}(\mathbb{Q})$ è legato al numero di bit

necessari per rappresentare le coordinate di $\tilde{P}$. Sia Miller che in seguito Silverman e Suzuki hanno mostrato che tale approccio non può funzionare. I motivi sono principalmente due. Il primo è che non si conosce un metodo efficiente per *innalzare* (*lift*) i punti di $E(F_p)$ in $\tilde{E}(\mathbb{Q})$. La seconda ragione è che il numero di punti di $\tilde{E}(\mathbb{Q})$ con basso *peso* è molto piccolo, pertanto solo un piccolissimo sottoinsieme di $E(F_p)$ può formare la *base di fattori S*. Una trattazione rigorosa di tali considerazioni va oltre lo scopo di questa Tesi, e può essere affrontata in [20, 33].

Una nuova linea di attacco all'ECDLP è stata recentemente proposta da J. Silverman, con l'algoritmo "*xedni calculus*". Esso costituisce una variante dell'*index-calculus* e più precisamente inverte l'ordine di esecuzione delle varie fasi dell'*index-calculus*. Non a caso il nome *xedni* corrisponde all'inversione della parola *index*. Per quanto questo nuovo approccio sia molto elegante, successive analisi hanno mostrato che lo *xedni* è di scarso utilizzo pratico.

4.3 Attacchi speciali

4.3.1 Attacco MOV

L'attacco MOV utilizza il Weil *pairing* per convertire un'istanza di ECDLP in $E(GF(q))$ in un istanza di DLP in $GF(q^m)$, per un qualche valore di m. Poiché il DLP in campo finito può essere *attaccato* dai metodi *index-calculus*, esso può essere risolto in modo più veloce, fintantoché $GF(q^m)$ non è molto più grande di $GF(q)$, ossia finché m è relativamente piccolo. Il valore m coincide con la grandezza $\deg_{MOV}(n)$ vista nel Par. 3.7.

Tale metodo venne proposto nel 1993 da Menezes, Okamoto e Vanstone (M.O.V).

Analisi dell'Algoritmo

Sia E una curva ellittica su $GF(q)$ e siano $P,Q \in E(GF(q))$. Sia n l'ordine di P tale che $\gcd(n,q) = 1$. Si vuole determinare k tale $Q = kP$. Anzitutto bisogna verificare che tale k esiste, utilizzando il seguente risultato:

Teorema 4.1 Esiste *k* tale che $Q = kP \Leftrightarrow nQ = \Theta \wedge$ *il Weil pairing* $e_n(P,Q) = 1$

Vediamo ora come procede l'attacco MOV. Anzitutto si individua il $\deg_{MOV}(n)$ cioè il più piccolo *m* tale che $E[n] \subseteq E[GF(q^m)]$. Il Corollario 3.1 ci garantisce che $\mu_n \subseteq GF(q^m)$. L'algoritmo è il seguente:

1. Scelgo un punto casuale $T \in E(GF(q^m))$
2. Calcolo l'ordine *r* di *T*.
3. Sia $d = \gcd(r,n)$ e $T_1 = (r/d)T$. Quindi T_1 ha ordine *d* che divide *n*, perciò $T_1 \in E[n]$.
4. Calcolo $\xi_1 = e_n(P,T_1)$ e $\xi_2 = e_n(Q,T_1) = e_n(kP,T_1)$. Sia ξ_1 che ξ_2 appartengono a $\mu_d \subseteq GF^*(q^m)$(*). Essendo il *pairing* $e_n(\cdot,\cdot)$ bilineare si ha $e_n(kP,T_1) = e_n(P+P+\cdots P,T_1) = [e_n(P,T_1)]^k$
5. Risolvo il logaritmo discreto $\xi_2 = \xi_1^k$ in $GF^*(q^m)$. Ottengo così $k \pmod d$
6. Ripeto con altri punti *T* casuali fino a quando il *minimo comune multiplo* dei valori *d* ottenuti è pari a *n*. Questo, grazie al Teorema del Resto Cinese, fornisce $k \pmod n$.
7.

(*)*In step 4 si ha* $\xi_1 \in \mu_d$ *poiché* $\xi_1^d = e_n(P,dT_1) = e_n(P,\Theta) = 1$, *analogo per* ξ_2.

■

Potenzialmente il grado MOV *m* può essere grande, in tal caso il DLP in $GF^*(q^m)$ non è più semplice del corrispondente ECDLP in $E(GF(q))$ e quindi l'attacco MOV perde la sua efficacia. Conoscere il valore esatto di $\deg_{MOV}(n)$, perciò, è molto importante ma non sempre facile. A tale proposito il seguente teorema fornisce un utile criterio di scelta del punto *P*, o meglio del suo ordine *n* .

Teorema 4.2 (Balasubramanian – Koblitz)
Sia E è una curva definita su $GF(q)$ ed n è un divisore primo di $\#E$ che non divide $q-1$. Allora

$$E(GF(q^k)) \text{ contiene } n^2 \text{ punti di ordine } n \Leftrightarrow n \mid q^k - 1$$

Quindi per evitare che l'algoritmo di riduzione sia praticabile bisogna accertarsi che l'ordine del punto P non divida $q^k - 1$ per valori di k per i quali il DLP in $GF(q^k)$ è trattabile[5].

In ogni caso vi sono curve particolarmente vulnerabili all'attacco MOV? Si, le curve *supersingolari*, che ricordiamo hanno traccia t divisibile per char(E(GF(q))), sono facilmente attaccabili con questo metodo. Vale infatti il seguente

Teorema 4.3 Sia E una curva definita su $GF(q)$ con traccia $t = q + 1 - \#E = 0$. Sia n un intero positivo. Se esiste un punto $P \in E(GF(q))$ di ordine n allora:

$$E[n] \subseteq E(GF(q^2))$$

Pertanto l'ECDLP definito su curve *supersingolari* $E(GF(q))$ con $t = 0$ può essere convenientemente, o pericolosamente a seconda del punto di vista, ridotto ad un DLP in $GF(q^2)$! Se invece la curva è *supersingolare* ma $t \neq 0$, il grado MOV rimane comunque molto piccolo. Infatti si dimostra che se E è una curva *supersingolare* ed n è un divisore primo di $\#E(GF(q))$ allora $\deg_{MOV}(n)$ è minore o uguale a 6.

E' evidente, quindi, che per ridurre l'efficacia dell'attacco MOV non bisogna assolutamente utilizzare curve *supersingolari*.
Un altro tipo di *attacco di riduzione* è l'attacco Frey-Rück che sfrutta il cosiddetto Tate *pairing*. Per approfondimenti si veda [2].

[5] Ricordiamo, dal Par. 3.5, che $E[n]$ è isomorfico a $Z_n \oplus Z_n$ quindi ha n^2 punti.

4.3.2 Attacco su curve anomale

Data una curva *anomala* E/F_p, studi indipendenti di Araki e Satoh, Semaev, Smart hanno mostrato come sia possibile stabilire un isomorfismo fra $E(F_p)$ ed il gruppo additivo F_p. Grazie a questo risulta possibile risolvere l'ECDLP in $E(F_p)$ in tempo polinomiale! D'altra parte si è visto che tale approccio non è praticabile su altri tipi di curve. Quindi per garantirsi l'immunità dall'attacco Araki-Satoh-Semaev-Smart è sufficiente verificare che la curva non sia *anomala*.

4.4 Riassunto

I principali attacchi conosciuti all'ECDLP che abbiamo presentato in questo capitolo sono riassunti in Tabella 4.1. Le considerazioni riportate si riferiscono ad una curva ellittica E definita sul campo finito F_q e ad un punto $P \in E(F_q)$ di ordine *n*.

Attacco	**Contromisura**
Pohlig-Hellman	Selezionare *n* primo
Baby Step-Giant Step	Già con valori non troppo elevati di *n* , richiede troppa memoria. Impraticabile
Pollard ρ	Selezionare *n* in modo che $\sqrt{n}$ rappresenti una quantità eccessiva di calcoli. Il valore minimo da considerare per *n* è 2^{160}
MOV	Verificare che *n* non divida $q^k - 1$ per tutti i valori $1 \le k \le 20$. Questo esclude le curve supersingolari
Araki-Satoh-Semaev-Smart	Verificare che $\#E(F_q) \neq q$

Tabella 4.1 Principali attacchi all'ECDLP e relative contromisure

Al momento il metodo Pollard ρ è considerato il più veloce attacco *general-purpose* al problema del Logaritmo Discreto su EC. Avendo un *running time*, lo ricordiamo, pari a $O\left(\sqrt{\frac{n\pi}{2}}\right)$ tale attacco è *fully-exponential*.

D'altra parte nel Cap. 2 si è evidenziato che i problemi IFP e DLP, ai quali corrispondono rispettivamente i crittosistemi RSA e DSA/ElGamal, sono risolvibili con algoritmi di tipo *sub-exponential*.

Risulta chiaro, adesso, perché la crittografia basata su Curve Ellittiche ha suscitato in questi anni tanto interesse. Il motivo è semplice: essa si basa su un problema matematico molto più difficile dei tradizionali IFP e DLP.
Consapevoli di questa intrinseca sicurezza dell'ECDLP vediamo, nel capitolo che segue, come esso viene sfruttato per scopi crittografici, ossia andiamo ad analizzare i principali algoritmi dell'ECC.

Capitolo 5

Crittografia basata su Curve Ellittiche

5.1 Introduzione

L'impiego delle Curve Ellittiche (EC) nel campo della Crittografia venne proposto in modo indipendente da Victor Miller (*IBM Watson Research Lab*) e Neil Koblitz (*University of Washington*) rispettivamente nel 1985 e 1986. Di fatto essi non inventarono nuovi algoritmi crittografici, piuttosto furono i primi ad implementare gli esistenti crittosistemi a chiave pubblica usando le curve ellittiche. Più precisamente Miller propose un analogo del protocollo Diffie-Hellman mentre Koblitz presentò un analogo del crittosistema El Gamal. Entrambi quindi utilizzarono il gruppo di punti di una curva ellittica definita su un campo finito per realizzare crittosistemi basati sul Logaritmo Discreto. Abbiamo già compreso che il vantaggio principale di questo approccio risiede nella mancanza di algoritmi con running time *sub-esponenziale* che possano calcolare logaritmi discreti in questi gruppi. Di conseguenza è possibile usare un gruppo basato sui punti di una curva ellittica con dimensioni più piccole, mantenendo lo stesso livello di sicurezza. Questo si traduce in chiavi crittografiche più piccole, risparmio di banda e implementazioni più veloci.

Nel 1991 venne proposto il primo analogo EC dell'RSA da Koyama, Maurer, Okamoto e Vanstone. Tale sistema si basa sui punti di una curva ellittica definita sull'anello Z_n (con n intero composito) e la *trapdoor information* è costituita dall'ordine della curva. Lavori successivi però mostrarono che l'analogo EC non introduceva significativi miglioramenti rispetto all'RSA originale.

Nel seguito, quindi, la Crittografia basata su Curve Ellittiche (ECC) verrà intesa come sola applicazione del Logaritmo Discreto sull'insieme dei punti di una curva ellittica (ECDLP). I primi tre paragrafi introducono alcuni concetti preliminari dell'ECC: i parametri di dominio, la generazione della coppia di chiavi e la rappresentazione di un messaggio. Nei paragrafi 5.5, 5.6 e 5.7 vengono descritti

degli algoritmi per produrre rispettivamente Firma Digitale, Cifratura e Scambio Chiavi. Il Par. 5.8 analizza il livello di standardizzazione raggiunto dall'ECC.

5.2 Parametri di Dominio

Un crittosistema basato su Curve Ellittiche viene definito da un insieme di parametri che descrivono la curva ellittica E/F_q, il campo finito F_q, un punto $P \in F_q$ e l'ordine n di P. Tali parametri devono essere selezionati sulla base di scelte implementative ed in modo che l'ECDLP su $E(F_q)$ resista agli attacchi conosciuti. Nel loro insieme, vengono chiamati *parametri di dominio*.

Definizione 5.1 I *parametri di dominio* di un crittosistema basato su Curve Ellittiche è l'insieme $D = (q, FR, S, a, b, P, n, h)$, dove:

- q è l'ordine del campo finito F_q
- FR (*field representation*) indica la rappresentazione usata per gli elementi di F_q.
- S è il *random seed* (dall'inglese, *seme casuale*) che viene utilizzato nel caso la curva ellittica venga generata in modo casuale. Esempi di generazione casuale di curve ellittiche si trovano in [20].
- $a, b \in F_q$ sono i coefficienti dell'equazione che descrive la curva E ($y^2 = x^3 + ax + b$ nel caso di un *campo primo*, $y^2 + xy = x^3 + ax^2 + b$ nel caso di *campo binario*).
- $P = (x_P, y_P)$ è un punto di $E(F_q)$, con ordine primo, detto *punto base*.
- n è l'ordine di P.
- $h = \#E(F_q)/n$ è il *cofactor*

5.2.1 Criteri di scelta

I *parametri di dominio* appena descritti devono essere scelti in modo da contrastare gli attacchi visti nel capitolo precedente. Pertanto devono essere soddisfatte le seguenti condizioni:

i) per resistere agli attacchi Pohlig-Hellman e Pollard ρ $\#E(F_q)$ deve essere divisibile per un primo n sufficientemente grande ($>2^{160}$).

ii) per resistere all'attacco MOV il suddetto n non deve dividere $q^k - 1$ per tutti i $k \in [1, C]$, dove C è grande abbastanza da rendere impraticabile la risoluzione del DLP in $GF(q^C)$ (nella pratica $C = 20$ è sufficiente).

iii) per resistere all'attacco Araki-Satoh-Semaev-Smart deve valere $\#E(F_q) \neq q$.

E' importante quindi che i parametri di dominio siano generati e successivamente validati in modo da soddisfare i requisiti appena elencati.

5.3 Generazione delle Chiavi

Ricordiamo che un crittosistema basato su Curve Ellittiche è a *chiave pubblica*, quindi utilizzerà una coppia di chiavi. Tale coppia sarà associata ad un particolare insieme di *parametri di dominio* $D = (q, FR, S, a, b, P, n, h)$. La chiave pubblica corrisponde ad un punto Q scelto a caso nel gruppo $\langle P \rangle$ generato da P. La corrispondente chiave privata è $d = \log_P Q$. Appare evidente che il calcolo della chiave privata d, a partire dalla chiave pubblica Q, coincide con l'ECDLP. Quindi è fondamentale che i *parametri di dominio* siano stati selezionati in modo da rendere l'ECDLP intrattabile.

Algoritmo 5.1 Generazione delle Chiavi

INPUT: Parametri di dominio $D = (q, FR, S, a, b, P, n, h)$

OUTPUT: Chiave pubblica Q, Chiave privata d

1. Seleziona a caso $d \in [1, n-1]$
2. Calcola $Q = dP$.
3. Return(Q,d)

5.3.1 Validazione delle Chiavi

Scopo della *validazione della chiave pubblica* è di verificare che una chiave pubblica possieda certe proprietà. Una validazione con esito positivo, comunque, dimostra che una chiave privata associata esiste logicamente, ma non garantisce che tale chiave privata sia stata effettivamente calcolata e ancora meno che il

sedicente proprietario la possieda. La validazione di chiave pubblica è particolarmente importante nel protocollo di *key-agreement* Diffie-Hellman dove un'entità *A* calcola una chiave segreta *K* combinando la propria chiave privata con la chiave pubblica di un'altra entità *B*. Se quest'ultima fosse disonesta, potrebbe utilizzare una chiave pubblica invalida in modo che la conoscenza di *K* possa rivelare informazioni sulla chiave privata di *A*.

Algoritmo 5.2 Validazione della chiave pubblica

INPUT: Parametri $D = (q, FR, S, a, b, P, n, h)$, chiave pubblica $Q = (x_Q, y_Q)$

OUTPUT: Accettazione o rifiuto della chiave pubblica *Q*

1. Verifica che $Q \neq \Theta$
2. Verifica che x_Q e y_Q siano elementi di F_q
3. Verifica che *Q* soddisfi l'equazione della curva *E*, definita da *a* e *b*
4. Verifica che $nQ = \Theta$
5. Se almeno un controllo è fallito Return("rifiutata")
   altrimenti Return("accettata")

Va precisato che le curve ellittiche su campi primi usate nella pratica hanno spesso *cofactor h* = 1, quindi vale *n* = #*E*. In tali casi le verifiche degli step 1, 2, 3 implicano automaticamente $nQ = \Theta$. Un tipo di attacco che si rivela efficace nel caso in cui non ci sia stata validazione della chiave pubblica, in particolare non sia stato verificato che $nQ = \Theta$, è l'attacco *small subgroup* [20]. Infatti se il *cofactor* è sufficientemente grande, questo attacco consente di calcolare la chiave privata della controparte.

5.4 Rappresentazione del messaggio

Per rappresentare il messaggio all'interno della curva ellittica vi sono tecniche diverse. Descriviamo il metodo di *embedding* proposto da Koblitz, attraverso il quale ogni messaggio viene rappresentato come un punto della curva.

Si supponga, senza perdere di generalità, che *E* sia una curva ellittica $y^2 = x^3 + ax + b$ definita sul campo primo F_p. Sia *p* un primo tale che:

$$p \equiv 3 \pmod 4 \qquad (5.1)$$

Sia m il messaggio, espresso come un intero che soddisfa $0 \le m < p/100 - 1$. Definiamo $x_i = 100m + i$, con $i = 0,1,\cdots,99$. Tali valori appartengono a F_p in quanto rispettano $i \le x_i < p + (i - 100) < p$.

Calcoliamo in sequenza , per $i = 0,1,\cdots,99$, i valori $s_i = x_i^3 + ax_i + b$ fino a quando non si ottiene un quadrato modulo *p*. Ricordando il Criterio di Eulero (Par. A.1), osserviamo che se $s_i^{(p-1)/2} \equiv 1 (\text{mod } p)$ allora s_i è un residuo quadratico modulo *p*. Grazie alla (5.1) si dimostra che la corrispondente radice quadrata è facilmente ricavabile: $y_i \equiv s_i^{(p+1)/4} (\text{mod } p)$. In tal modo si ottiene il punto $P_m = (x_i, y_i)$ sulla curva *E*. Per recuperare il messaggio *m* da P_m è sufficiente calcolare $\lfloor x_i / 100 \rfloor$ ossia il più grande intero minore o uguale di $x_i/100$.

Poiché s_i è un elemento casuale di F_p^*, che è ciclico di ordine dispari, la probabilità che s_i sia un quadrato è pari circa a 1/2 [13]. Di conseguenza la probabilità di non riuscire ad individuare un quadrato per tutti i valori $i = 0,1,\cdots,99$ è pari a $(1/2)^{100} = 2^{-100}$, quindi trascurabile.

5.5 Firma Digitale

5.5.1 Introduzione

Nel primo capitolo è stato presentato il concetto di Firma Digitale. Vediamo una definizione più formale.

Definizione 5.2 Uno schema di firma digitale Σ è costituito da quattro algoritmi:

1. Un algoritmo per generare l'insieme di *parametri di dominio D*
2. Un algoritmo che ha in input *D* e genera la coppia di chiavi (*Q*, *d*)
3. Un algoritmo di *firma* che riceve in input *D*, la chiave privata *d*, un messaggio *m* e produce una firma *S*.
4. Un algoritmo di verifica che riceve in input *D*, la chiave pubblica *Q*, il messaggio *m* e la presunta firma *S* . Tale firma viene accettata o rifiutata.

Introduciamo ora un criterio di sicurezza per schemi di firma, dovuto a Goldwasser, Micali e Rivest (GMR).

Definizione 5.3 Uno schema di firma Σ è detto sicuro (o GMR-*sicuro*) se non può essere contraffatto da un avversario con potenza di calcolo limitata e che può eseguire attacchi di tipo *chosen-message*. In altre parole un avversario che può ottenere firme legittime di ogni messaggio di sua scelta non riesce produrre una firma valida di un qualsiasi messaggio nuovo.

Tale definizione di sicurezza è molto forte, infatti concede all'avversario ampi spazi di manovra. Nella realtà però, spesso l'avversario non può ottenere tutte le firme che desidera. D'altra parte, nel progettare nuovi schemi di firma, è prudente ispirarsi al criterio di sicurezza più forte. A tale riguardo, il concetto di *sicurezza*-GMR è largamente accettato come riferimento.
Vediamo adesso l'algoritmo di firma elettronica basato su EC che ha raggiunto la maggiore diffusione.

5.5.2 ECDSA

L'algoritmo ECDSA (Elliptic Curve Digital Segnature Algorithm) è l'analogo EC del DSA (Par. 2.2.3). Gli algoritmi 5.3 e 5.4 illustrano rispettivamente il processo di firma e di verifica. In entrambi $H(\ )$ indica una funzione *hash*.

Algoritmo 5.3 Generazione della firma ECDSA

INPUT: Parametri $D = (q, FR, S, a, b, P, n, h)$, chiave privata d, messaggio m

OUTPUT: Firma (r,s)

1. Seleziona a caso $k \in [1, n-1]$
2. Calcola $kP = (x_1, y_1)$ e $r = x_1 \bmod n$
3. Se $r = 0$ torna allo step 1
4. Calcola $e = H(m)$
5. Calcola $s = k^{-1}(e + dr) \bmod n$. Se $s = 0$ torna allo step 1
6. Return (r,s)

Algoritmo 5.4 Verifica della firma ECDSA

INPUT: Parametri $D = (q, FR, S, a, b, P, n, h)$, chiave pubblica Q, messaggio m, firma (r,s).

OUTPUT: Accettazione o rifiuto della firma

1. Verifica che r e s siano interi nell'intervallo [1, n – 1] altrimenti Return("rifiutata")
2. Calcola $e = H(m)$
3. Calcola $w = s^{-1} \bmod n$
4. Calcola $u_1 = ew \bmod n$ e $u_2 = rw \bmod n$
5. Calcola $X = (x_1, y_1) = u_1 P + u_2 Q$
6. Se $X = O$ allora Return ("rifiutata")
7. Calcola $v = x_1 \bmod n$
8. Se $v = r$ Return ("accettata") altrimenti Return ("rifiutata")

Dimostrazione che la Verifica è corretta

Se (r,s) è la firma legittima del messaggio m allora $s \equiv k^{-1}(e + dr) (\bmod n)$.

Perciò $k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d \pmod n$.

Infine $X = u_1 P + u_2 Q = (u_1 + u_2 d)P = kP$ e quindi $v = r$ come richiesto.

Affinché ECDSA sia GMR-*sicuro*, è necessario che l'ECDLP in $\langle P \rangle$ sia intrattabile e che la funzione *hash* $H(\ )$ sia crittograficamente sicura (*one-way* e *strong collision resistant*). Non è stato provato comunque che tali condizioni sono anche sufficienti. D'altra parte l'ECDSA nel *modello di gruppo ideale*, al posto del gruppo $\langle P \rangle$, e con funzione *hash* sicura è stato dimostrato essere GMR-*sicuro* [2]. Tale risultato, sebbene non implichi logicamente la sicurezza nei gruppi reali, ispira una ragionevole confidenza nella sicurezza dell' ECDSA.

Il controllo nello step 1 dell'algoritmo di Verifica (che r ed s siano nell'intervallo [1, n – 1]) è una misura di sicurezza che previene determinati attacchi. Si consideri ad esempio il caso in cui venga ignorato il controllo $r \neq 0$ e, per ridurre le dimensioni dei *parametri di dominio*, venga selezionato il punto base P con

coordinata x nulla, ossia $P = (0, \sqrt{b})$. In tali condizioni un avversario può contraffare la firma dell'entità A su un qualsiasi messaggio m. Infatti è facile verificare che ($r = 0$, $s = e$) , con $e = H(m)$, è una firma valida di m:
nello step 4, infatti, si ottiene $u_1 = 1$ e $u_2 = 0$, quindi $X = P$ e infine $v = r = 0$ come richiesto.

Molto importante per la sicurezza dell'ECDSA è il *per-message secret k* (step 1 dell'algoritmo di Generazione). Se infatti un avversario ottenesse il valore k usato da A per generare la firma (r,s) sul messaggio m, potrebbe ricavare la chiave privata di A, poiché:

$$d = r^{-1}(ks - e) \bmod n \qquad (5.2)$$

dove $e = H(m)$. In aggiunta, studi di Howgrave-Graham e Smart hanno mostrato come la conoscenza di pochi (ad esempio 5) bit consecutivi dei valori k usati per firmare un centinaio di messaggi permetta di calcolare la chiave privata. Tali osservazioni confermano che il *per-message secret k* deve essere generato casualmente ogni volta, utilizzato in modo sicuro ed eliminato subito dopo l'utilizzo.

5.6 Cifratura a Chiave Pubblica

Gli algoritmi di cifratura a Chiave Pubblica vengono usati anche per soddisfare il requisito di *Confidentiality* (vedi Par.1.1), ossia per trasmettere in modo sicuro un messaggio fra due entità A e B. Come già detto, essi risultano più lenti degli algoritmi simmetrici, pertanto sono utilizzati per cifrare brevi messaggi come PIN, numeri di carta di credito oppure per trasportare chiavi di sessione da utilizzare con sistemi simmetrici.
Nel paragrafo 2.4.1 l'analogo EC del sistema El Gamal ci ha permesso di introdurre la crittografia basata su Curve Ellittiche. La versione base, così come l'abbiamo presentata, non viene utilizzata nelle applicazioni pratiche, piuttosto si impiegano algoritmi che ad essa si ispirano. Come il sistema ECIES.

5.6.1 ECIES

L'Elliptic Curve Integrated Encryption System (ECIES) fu proposto da Bellare e Rogaway e costituisce una variante dell'algoritmo El Gamal. E' chiamato *integrato* poiché combina un sistema a chiave pubblica con uno simmetrico. In

particolare uno schema Diffie-Hellman *one-pass* viene usato per ottenere due chiavi k_1 e k_2. Il termine *one-pass* significa che le chiavi condivise, *R* e *Z*, vengono decise da una sola entità. La chiave k_1 è usata per la cifratura simmetrica del messaggio *m*, k_2 invece fornisce l'autenticità del testo cifrato ottenuto. Gli algoritmi 5.7 e 5.8 illustrano le fasi di cifratura e decifratura.

Algoritmo 5.7 Cifratura ECIES

INPUT: Parametri $D = (q, FR, S, a, b, P, n, h)$, chiave pubblica *Q*, messaggio *m*

OUTPUT: Testo cifrato (*R*, *C*, *t*)

1. Seleziona a caso $k \in [1, n-1]$
2. Calcola $R = kP$ e $Z = (x_z, y_z) = hkQ$. Se $Z = \Theta$ torna allo step 1
3. $(k_1, k_2) \leftarrow \text{KDF}(x_z, R)$
4. Calcola $C = Enc_{k_1}(m)$ e $t = MAC_{k_2}(C)$
5. Return(*R*, *C*, *t*)

Algoritmo 5.8 Decifratura ECIES

INPUT: Parametri di dominio $D = (q, FR, S, a, b, P, n, h)$, chiave privata *d*, testo cifrato (*R*, *C*, *t*).

OUTPUT: messaggio *m* o rifiuto del testo cifrato

1. Esegui una validazione della chiave *R*. Se la validazione fallisce Return("testo cifrato rifiutato").
2. Calcola $Z = (x_z, y_z) = hdR$. Se $Z = \Theta$ allora Return("testo cifrato rifiutato")
3. $(k_1, k_2) \leftarrow \text{KDF}(x_z, R)$
4. Calcola $t' = MAC_{k_2}(C)$. Se $t \neq t'$ allora Return("testo cifrato rifiutato")
5. Calcola $m = Dec_{k_1}(C)$
6. Return(*m*)

Dimostrazione che la Decifrazione è corretta

Se (*R*, *C*, *t*) è il legittimo testo cifrato del messaggio *m*, allora:

$$hdR = hd(kP) = hk(dP) = hkQ$$

Il decifratore può generare la stessa coppia di chiavi (k_1, k_2) calcolate dal mittente e quindi può correttamente recuperare il messaggio *m*.

Lo schema ECIES utilizza le seguenti primitive crittografiche:

1. KDF( ) è la *key derivation function* che, agendo come una funzione *hash*, genera la coppia di chiavi (k_1, k_2).
2. Enc_k( ) rappresenta una funzione di cifratura di un sistema simmetrico (come ad esempio AES). Dec_k( ) è la corrispondente funzione di decifratura.
3. $MAC_k(D)$ è un algoritmo di *message authentication code* (come ad esempio HMAC). Tali algoritmi utilizzano una chiave k, condivisa fra mittente e destinatario, per generare un *checksum crittografico* (o codice MAC) relativo all'informazione D. Garantiscono autenticazione di messaggio e sono funzioni non invertibili [31].

Osservazione sulla KDF( )

Il punto $Z = hkQ = hdR$, detto *Diffie-Hellman shared secret*, è un punto condiviso in modo segreto fra mittente e destinatario. Il punto $R = kP$ invece è una sorta di *chiave pubblica one-time*. A tale proposito è utile osservare che le due chiavi simmetriche k_1 e k_2 dipendono, attraverso la KDF( ), sia da Z che da R. La dipendenza da R è importante perché se non ci fosse un avversario potrebbe sostituire R nel testo cifrato (R, C, t) ottenendo un altro testo cifrato valido (R', C, t) relativo allo stesso messaggio in chiaro m.

5.7 Scambio Chiavi

I protocolli di Scambio Chiave (in letteratura, *key establishment*) consentono a due entità di utilizzare una rete insicura per condividere segretamente una chiave, da utilizzare eventualmente in seguito con sistemi di cifratura simmetrica. La chiave condivisa è detta *chiave di sessione*. In particolare, con i protocolli di *key transport* una entità genera la chiave e la invia alla seconda entità. Nei protocolli di *key agreement* invece entrambe le entità forniscono informazioni che verranno usate per calcolare la chiave condivisa.

In questo paragrafo vedremo due protocolli di *key agreement* basati su EC: lo schema base Diffie-Hellman (ECDH) e l'ECMQV.

5.7.1 ECDH

L'analogia con il protocollo visto nel Par. 2.2.1 è evidente. Vi sono due entità, *A* e *B*, che inviandosi reciprocamente un messaggio riescono a condividere una chiave segreta. Il funzionamento è schematizzato nel Protocollo 5.1

Protocollo 5.1 ECDH

1. *A* e *B* condividono i Parametri di Dominio $D = (q, FR, S, a, b, P, n, h)$
2. *A* seleziona un intero casuale $x \in [1, n-1]$ e calcola $X = xP$
3. *A* invia X a B
4. *B* seleziona un intero casuale $y \in [1, n-1]$ e calcola $Y = yP$
5. *B* invia Y a A
6. *A* riceve Y e calcola $K_A = xY = xyP$
7. *B* riceve X e calcola $K_B = yX = yxP$
8. *A* e B condividono la chiave segreta $K = K_A = K_B$

I messaggi trasmessi (xP e yP) vengono detti *ephemeral public keys* (dall'inglese, *effimere*) in quanto sono nella forma di chiavi pubbliche basate sul Logaritmo Discreto ma durano per un breve periodo di tempo.

Similmente a quanto visto nel protocollo Diffie-Hellman originale, il compito di ricavare xyP conoscendo xP e yP è detto *Elliptic Curve Diffie-Hellman Problem* (ECDHP). Chiaramente se si riesce a risolvere l'ECDLP si può anche risolvere l'ECDHP, mentre l'implicazione inversa non è stata dimostrata.

Il sistema ECDH è molto versatile poiché non richiede alcuna condivisione di informazioni a priori fra le due entità A e B. Tale aspetto, però, è al tempo stesso una debolezza del protocollo, come lo dimostra il seguente esempio.

Lo Figura 5.1 illustra un attacco di tipo *man-in-the-middle* al protocollo ECDH. In questo attacco Alice concorda la chiave $K_A = xzP$ con Eva, pensando di averla concordata con Bob. Analogamente Bob concorda la chiave $K_B = ywP$ con Eva, convinto di essersi accordato con Alice. A questo punto Eva può esaminare la comunicazione fra Alice e Bob inoltrando di volta in volta i messaggi alle due vittime

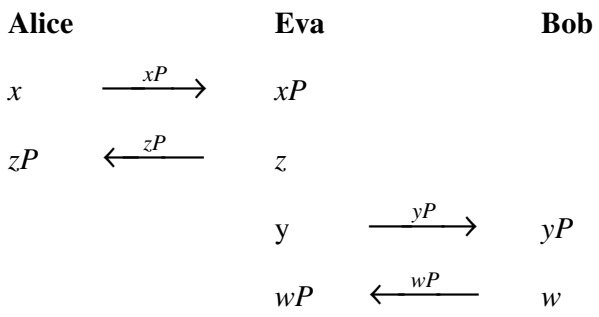

Figura 5.1 Attacco *man-in-the-middle* a ECDH

Il problema, evidenziato da questo tipo di attacco, è che il protocollo ECDH non fornisce *data-origin authentication*, ossia Alice e Bob non conoscono l'esatta provenienza della *ephemeral public key* appena ricevuta.

Un metodo per ottenere *data-origin authentication* è quello di firmare le *ephemeral keys* con un opportuno algoritmo di firma elettronica. In questo modo Alice non spedirà solo il valore xP ma il messaggio

$$(xP,(r,s))$$

dove (r,s) è la firma ECDSA di xP. L'inconveniente di questa versione *firmata* del protocollo ECDH è l'aumento della banda utilizzata: infatti per generare una chiave di sessione condivisa bisogna inviare le due *ephemeral key* insieme alle rispettive firme. Una valida alternativa è data dal protocollo ECMQV.

5.7.2 ECMQV

Il protocollo di *key-agreement* ECMQV (Elliptic Curve Menezes Qu Vanstone) costituisce un'evoluzione dell'ECDH. La differenza sostanziale è che ECMQV prevede che ognuna delle due entità possieda la chiave pubblica, non *ephemeral*, della controparte. Tornando all'esempio del paragrafo precedente, questo significa che:

- (Q_A, d_A) è la coppia di chiavi Pubblica/Privata di Alice
- $(Q_B, d_B,)$ è la coppia di chiavi Pubblica/Privata di Bob
- Alice ottiene in modo fidato la chiave pubblica di Bob Q_B
- Bob ottiene in modo fidato la chiave pubblica di Alice Q_A

Fatta questa premessa, il Protocollo 5.2 descrive il funzionamento di ECMQV.

Protocollo 5.2 ECMQV

1. Alice e Bob condividono i Parametri $D = (q, FR, S, a, b, P, n, h)$
2. Alice seleziona un intero casuale $x \in [1, n-1]$ e calcola $X = xP$
3. Alice invia X a Bob
4. Bob seleziona un intero casuale $y \in [1, n-1]$ e calcola $Y = yP$
5. Bob invia Y ad Alice
6. Alice calcola $s_A = (x + \overline{X}\, d_A) \bmod n$
7. Bob calcola $s_B = (y + \overline{Y}\, d_B) \bmod n$
8. Alice calcola $K_A = hs_A(Y + \overline{Y}Q_B)$
9. Bob calcola $K_B = hs_B(X + \overline{X}Q_A)$
10. Alice e Bob condividono la chiave $K = K_A = K_B$

Anzitutto osserviamo che il simbolo $\overline{X}$ rappresenta i primi $f/2$ bit della componente x del punto X, dove f è la *bitlength* di n ($f = \lfloor \log n \rfloor + 1$); analogamente per $\overline{Y}$. Pertanto $\overline{X}$ e $\overline{Y}$ sono valori scalari. La chiave condivisa K invece è un punto della curva ellittica.

Dimostrazione che il *key-agreement* è corretto

E' sufficiente esplicitare i valori K_A e K_B, per verificare che coincidono:

$$K_A = hs_A(Y + \overline{Y}Q_B) = hs_A(y + \overline{Y}d_B)P = hs_As_BP$$
$$K_B = hs_B(X + \overline{X}Q_A) = hs_B(x + \overline{X}d_A)P = hs_Bs_AP$$

Il valore $s_A = (x + \overline{X}\, d_A) \bmod n$ è detto *firma implicita* di Alice per la *ephemeral key X*. Di fatto è una firma poiché, vista la presenza di d_A, solo Alice può calcolarla. E' *implicita* nel senso che viene indirettamente verificata da Bob quando questi si avvale dell'uguaglianza:

$$s_AP = X + \overline{X}Q_A$$

Come si può vedere dagli step 1-5 l'ECMQV utilizza gli stessi messaggi, quindi stessa banda, dell'ECDH base. In questo caso però il *key-agreement* è autenticato, quindi sia Bob che Alice sono certi di condividere la chiave di sessione con l'entità desiderata. In aggiunta tentativi di attacco *man-in-the-middle* fallirebbero

perché la chiave privata dell'avversario (nell'esempio Eva) non consentirebbe di ottenere una chiave di sessione condivisa con la vittima.

5.8 Standards dell'ECC

Gli algoritmi qui presentati costituiscono una valida alternativa ai sistemi attualmente in uso (RSA, DSA, Diffie-Hellman). Questo soprattutto in ragione delle considerazioni di sicurezza fatte nel Par. 4.4. L'utilizzo diffuso di un crittosistema però è strettamente legato ad una completa interoperabilità fra implementazioni diverse e ad un riconoscimento a livello internazionale. Decisivo, quindi, è il processo di standardizzazione al quale deve essere sottoposto un sistema crittografico. Prima di tutto è importante che siano standardizzate le famiglie di curve, con relativi parametri di dominio, da utilizzare per scopi crittografici. A tale proposito il NIST ha pubblicato un elenco di *curve raccomandate* che costituisce un riferimento per tutte le implementazioni ECC [25]. Riportiamo adesso gli standards relativi all'ECC promossi dai principali organismi internazionali.

ANSI L'American National Standards Institute è un organizzazione privata la cui missione è promuovere la diffusione di standards. Il comitato X9 è parte dell'ANSI e sviluppa standards per i servizi bancari e finanziari. In particolare il sottocomitato X9F, che si occupa delle problematiche di sicurezza informatica, ha sviluppato due standards relativi all'ECC:

- X9.62 (1999): standardizza l'ECDSA
- X9.63 (2001): standardizza vari protocolli di *key agreement* e *key transport*, tra i quali ECDH, ECMQV e ECIES.

NIST Il National Institute of Standards and Technology è un'agenzia federale del Dipartimento del Commercio degli Stati Uniti. Tra i suoi compiti vi è la pubblicazione dei FIPS (Federal Information Processing Standards) che regolamentano problematiche di sicurezza per ambienti governativi. Il NIST ha accreditato l'algoritmo ECDSA, insieme a DSA e RSA, come algoritmo di Firma Elettronica nel FIPS 186-2, pubblicato il 27 gennaio 2000 [24]. Un altro standard molto importante è il FIPS 140-2 il quale specifica i requisiti che il governo degli Stati Uniti richiede a tutti i prodotti hardware e software che trattano informazioni

riservate (escluse quelle di carattere militare). In esso gli unici algoritmi asimmetrici riconosciuti sono RSA, DSA e ECDSA.

IEEE L'Institute of Electrical and Electronics Engineers è un organizzazione privata dedicata a promuovere pubblicazioni, convegni e standards. Il gruppo di lavoro IEEE P1363 si occupa della standardizzazione della crittografia a chiave pubblica e lo standard 1363-2000 (Standards Specification for Public-Key Cryptography) contiene, tra gli altri, gli algoritmi ECDSA, ECDH, ECMQV. La bozza 1363a invece è un complemento al 1363-2000 ed include anche ECIES.

IETF L'Internet Enginnering Task Force si occupa di sviluppare standards relativi ai protocolli utilizzati su Internet. A livello crittografico l'IETF è stata decisiva per la diffusione degli standards IPSec, TLS (Transport Layer Security), S/MIME. Vediamo come tali standards contemplano l'uso dell'ECC:

- IPSec: Il protocollo IKE (Internet Key Exchange), spiegato nell'RFC 2409, prevede l'impiego di ECDH per *key agreement*. Inoltre l'Internet-Draft "*ECP Groups for IKE and IKEv2*" [12] descrive come utilizzare certe curve raccomandate dal NIST in IKE. Su tale draft diversi produttori come Cisco e Nortel basano le proprie implementazioni di ECDH in IKE.
- SSL/TLS: Attualmente vi è solo un Internet-Draft, "*ECC Cipher Suites for TLS*" [11], che descrive come integrare gli algoritmi di scambio chiavi basati su ECC in TLS. In particolare specifica l'uso di ECDH nella fase di *handshake* e ECDSA come meccanismo di autenticazione. Considerato che è ancora in stato di *draft* (bozza) non esistono soluzioni commerciali che utilizzano l'ECC in TLS.
- S/MIME: L'utilizzo di S/MIME (RFC 3369) consente di proteggere la posta elettronica, basandosi sulla Cryptographic Message Syntax (CMS). L'RFC 3278 "*Use of ECC Algorithms in Cryptographic Message Syntax* " standardizza l'impiego degli algoritmi ECDH, ECDSA, ECMQV in CMS.

ISO/IEC L'International Standards Organization e l'International Electrotechnical Commision hanno congiuntamente sviluppato standards in ambito crittografico. Lo standard ISO/IEC 15946 descrive diversi algoritmi di

firma elettronica e di *key establishment* basati su EC, come ECDSA, ECDH, ECMMQV.
La Tabella 5.1 riassume i principali standards ECC e gli algoritmi ai quali si riferiscono.

Standard	Algoritmo	Status
ANSI X9.62	ECDSA	approvato
ANSI X9.62	ECDSA, ECDH, ECMQV	bozza
FIPS 186-2	ECDSA	approvato
IEEE P1363	ECDSA, ECDH, ECMQV	approvato
IEEE P1363a	ECIES	bozza
ISO 15946	ECDSA, ECDH, ECMQV	bozza

Tabella 5.1 Standards ECC

Per quanto riguarda il supporto commerciale dell'ECC un'azienda in particolare, la Certicom, va menzionata. Essa infatti è particolarmente impegnata, anche a livello di brevetti, nel commercializzare prodotti crittografici basati su EC. Analogamente a quanto fatto dalla concorrente RSA Security anche la Certicom ha istituito nel 1997 un concorso, il *Certicom ECC Challenge*, per risolvere l'ECDLP con chiavi di dimensione sempre più grande. Ad oggi la più difficile istanza ECDLP proposta dalla Certicom che è stata risolta è ECCp-109, dove 109 rappresenta la *bitlength* di n (l'ordine del punto base P). Tale risultato è stato ottenuto nel novembre 2002 grazie all'impiego di 10000 PC per 549 giorni.

Capitolo 6

Considerazioni conclusive

6.1 Confronti

Ora che abbiamo approfondito sia gli aspetti teorici che implementativi dell'ECC è opportuno commisurare la sicurezza garantita da questa tipo di crittografia con quella assicurata dai problemi IFP e DLP. Considerato che vengono utilizzati algoritmi differenti il confronto avverrà basandosi sulle dimensioni delle chiavi utilizzate, a parità di sicurezza garantita.

Prima di questo però è necessario fare la seguente considerazione. Molte applicazioni con particolari esigenze di "sicurezza" possono impiegare algoritmi crittografici diversi. A loro volta ognuno di tali algoritmi può essere implementato con chiavi di dimensioni variabili. A seconda delle circostanze una chiave eccessivamente grande rispetto alle necessità potrà influire negativamente sulle prestazioni. D'altra parte l'uso di chiavi troppo piccole potrebbe non garantire un livello adeguato di sicurezza. Pertanto sarebbe opportuno individuare il livello minimo della dimensione della chiave che garantisca un ragionevole grado di sicurezza. A tale esigenza ha cercato di rispondere il NIST con il documento SP 800-57 "*Reccomendation for Key Management"* [25]. In questa pubblicazione sono state indicate le dimensioni minime delle chiavi che forniscono un adeguato livello di sicurezza, per i prossimi 30 anni. Tali risultati fanno riferimento alla cifratura simmetrica (AES, 3DES, etc.) e sono riassunti in Tabella 6.1

Protection lifetime	Minimum key *bitlength*
fino al 2010	80
dal 2011 al 2030	112
dal 2030	128

fonte: NIST SP 800-57

Tabella 6.1 Key *bitlength* raccomandate dal NIST

Dalla Tabella 6.1 si osserva, ad esempio, che fino al 2010 gli algoritmi simmetrici devono utilizzare chiavi di almeno 80 bit per essere considerati sicuri. Si capisce quindi perché il DES (chiave a 56 bit) non è più considerato affidabile. Dopo il 2030 invece le chiavi utilizzate dovranno essere lunghe almeno 128 bit.
Torniamo adesso al confronto fra ECC e le altre cifrature asimmetriche. A tale proposito ricordiamo il *running-time* dei più veloci algoritmi *general-purpose* che risolvono i problemi matematici sottostanti: IFP, DLP, ECDLP. Essi sono riportati in Tabella 6.2.

Problema matematico	Algoritmi Crittografici	Complessità	
		Runnig-time	Tipo
IFP	RSA	$L_n\left[\frac{1}{3}, 1.923\right]$	Sub exponential
DLP	El Gamal, DSA, DH	$L_p\left[\frac{1}{3}, 1.923\right]$	Sub exponential
ECDLP	ECDSA, ECMQV, ECIES, ECDH	$O\left(\sqrt{\frac{n\pi}{2}}\right)$	Fully exponential

Tabella 6.2 Complessità risolutiva dei problemi IFP, DLP, ECDLP

Anzitutto si può osservare che i problemi IFP e DLP sono sostanzialmente equivalenti, pertanto sono confrontabili anche i livelli di sicurezza offerti dai corrispondenti algoritmi crittografici. Inoltre, come già anticipato nel Cap. 4, risulta evidente che l'ECDLP è un problema molto più arduo degli altri due. Ciò significa che, a parità di *bitlength* della chiave, gli algoritmi basati su EC sono molto più sicuri dei sistemi RSA/DSA. In altre parole per garantire lo stesso livello di sicurezza di RSA/DSA gli algoritmi ECC possono usare chiavi molto più piccole. Queste considerazioni sono state formalizzate dal NIST attraverso una tabella di equivalenza fra la sicurezza offerta da algoritmi simmetrici e la

sicurezza offerta dai vari algoritmi a chiave pubblica [25]. Essa è riportata in Tabella 6.3.

Bits di sicurezza	Algoritmo simmetrico	ECC size (bit)	RSA/DSA size (bit)
80	Skipjack	160	1024
112	3DES	224	2048
128	AES-128	256	3072
192	AES-192	384	7680
256	AES-256	512	15360

fonte: NIST SP 800-57

Tabella 6.3 Confronto di sicurezza fra ECC e RSA (NIST)

La tabella va letta in questo modo. Consideriamo ad esempio la seconda riga: la sicurezza offerta da un cifrario simmetrico con chiave a 112 bit (come 3DES) è equivalente alla sicurezza offerta da un sistema ECC a 224 bit (ECC-224) e da un sistema RSA a 2048 bit (RSA-2048). I valori 224 e 2048 corrispondono al *bitlength* rispettivamente dell'ordine del punto base P e del modulus n dell'istanza IFP.

Le cifre parlano da sole: al crescere del livello di sicurezza, le dimensioni (in bit) dei sistemi RSA/DSA crescono molto più rapidamente delle dimensioni dei sistemi ECC. Incrociando i dati della Tabella 6.1, ad esempio, si può osservare che fra soli cinque anni i sistemi RSA, per essere considerati sicuri dovranno raddoppiare la *bitlenght* del modulus n.

6.1.1 Dimensioni delle Chiavi

Vediamo ora come le dimensioni, in bits, di RSA e di ECC riportate in Tabella 6.3 si riflettono direttamente sulla dimensione delle chiavi. Una chiave pubblica RSA consiste nella coppia (n,e) (vedi Par. 2.2) dove n è il modulus mentre e è l'esponente di cifratura. Un valore tipico di e corrisponde a $2^{16} + 1$, quindi la chiave pubblica di un sistema RSA a 1024 bits richiederà 131 = 128 +3 bytes (1024 bits = 128 bytes). Similmente un sistema RSA a 15360 bits avrà una chiave pubblica di 1923 bytes.

Nei sistemi ECC la chiave pubblica corrisponde ad un punto Q della Curva Ellittica ($Q = dP$), rappresentato dalle coordinate (x,y). Entrambe le coordinate hanno *bitlength* pari alla dimensione del campo F_q ed essa è spesso uguale all'ordine del punto base P. Di conseguenza un sistema ECC a 160 bits avrà una chiave pubblica di circa 40 = 20 + 20 bytes (160 bits = 20 bytes). Analogamente un sistema ECC a 512 bits utilizzerà una chiave pubblica di 128 bytes. In Tabella 6.4 sono riportate le dimensioni (in bytes) delle chiavi pubbliche relative a sistemi ECC e RSA che forniscono livelli di sicurezza equivalenti.

Bits di sicurezza	Chiave pubblica ECC	Chiave Pubblica RSA	Rapporto ECC/RSA
80	40	131	1:3
112	56	259	1:4
128	64	387	1:6
192	96	963	1:10
256	128	1923	1:15

Tabella 6.4 Confronto fra chiavi pubbliche RSA/ECC (bytes)

Ancora una volta si può vedere che al crescere del livello di sicurezza aumenta il rapporto fra le dimensioni della chiave RSA e le dimensioni della chiave ECC. La riduzione delle dimensioni della chiave introdotta dall' ECC si rivela cruciale in ambienti con risorse computazionali limitate come palmari, PDA e dispositivi wireless in generale mentre introduce miglioramenti poco significativi in un contesto PKI che impiega certificati X.509. La dimensione tipica di un certificato X.509, infatti, è di circa 1000 bytes, pertanto cambiare la chiave pubblica del proprietario da RSA-1024 a ECC-160 riduce la dimensione del certificato di meno del 10%.

6.1.2 Dimensioni della Firma

Le osservazioni fatte per la chiave pubblica ci consentono di confrontare anche le dimensioni della Firma Elettronica. In un sistema RSA-1024 la Firma Elettronica corrisponde ad un valore di 1024 bits (vedi Par. 2.2) che quindi è rappresentabile

con 128 bytes. Analogamente la firma di un sistema RSA-15360 richiede 1920 bytes.

Vediamo i sistemi ECC equivalenti. La firma ECDSA di un sistema a 160 bit è costituita da una coppia di valori da 160 bits, per un totale di 40 bytes. Similmente in un sistema a 512 bits la firma occuperà 128 bytes.

Ritroviamo perciò le stesse proporzioni viste con la chiave pubblica e quindi valgono le medesime considerazioni.

6.1.3 Prestazioni

Nei paragrafi precedenti si è visto come il passaggio da un sistema RSA ad un equivalente ECC comporti una sensibile riduzione della dimensione della chiave pubblica e della Firma Elettronica. E' lecito aspettarsi che questo si rifletta positivamente anche sulle prestazioni, infatti così avviene. Premesso che ogni comparazione numerica dipende fortemente dalla qualità dell'implementazione, dalla piattaforma usata e dal livello di ottimizzazione impiegato, in letteratura vi sono numerosi risultati sperimentali che testimoniano la maggiore efficienza dei sistemi ECC. Vediamone alcuni.

Palmare RIM BlackBerry

Il palmare BlackBerry della Research In Motion (RIM) costituisce un'estensione del tradizionale desktop. Esso dispone di funzioni come calendario, e-mail, modem integrato, giochi, oltre ad essere aperto ad applicazioni di terze parti. Come in tutti i dispositivi wireless particolare attenzione è stata posta alla cifratura delle trasmissioni. In [5, vol 2, n°1] si possono leggere i risultati di alcuni test comparativi eseguiti dalla RIM. I tempi riportati in Tabella 6.5 si riferiscono al livello di sicurezza di 128 bit equivalenti (vedi Tabella 6.3) e sono stati raccolti con il modello BlackBerry 7230.

Operation	ECC-256	RSA-3072
Key Generation	166 ms	N/A*
Encrypt / Verify	150 ms	52 ms
Decrypt / Sign	168 ms	8s

* Tempo troppo lungo da misurare

Tabella 6.5 Confronto tempi ECC vs RSA su BlackBerry

Come è facile osservare, il sistema RSA è più performante solo nella fase *Encrypt/Verify*. Questo perché l'esponente di cifratura *e* ha spesso un valore particolarmente basso. Nelle altre due fasi invece la soluzione ECC è molto più veloce.

SSL

Ricerche condotte dalla Sun Microsystems hanno evidenziato come l'impiego dell'ECC in SSL (Secure Socket Layer) migliori sensibilmente le prestazioni. I dettagli dello studio sono consultabili in [8], qui riassumiamo alcuni risultati numerici. In sostanza gli autori della ricerca hanno aggiunto le funzionalità ECC all'implementazione OpenSSL e hanno poi confrontato la nuova cifratura con quella RSA. In Tabella 6.6 si possono leggere i tempi misurati con il comando OpenSSL *speed* su due piattaforme differenti: un Linux PDA con processore StrongARM a 200 MHz e un Ultra-80 Sun Server con processore UltraSPARC II a 450 MHz.

Platform	Bits di sicurezza	RSA		ECDSA	
		Encr / Ver	Decr / Sign	Encr / Ver	Decr / Sign
Sun Ultra-80	80	1.7	32.1	13.0	6.8
	112	6.1	205.5	18.1	9.2
Linux PDA	80	10.8	188.7	46.5	24.5
	112	39.1	1273.8	76.6	39.0

Tabella 6.6 Confronto di tempi ECDSA vs RSA con OpenSSL (ms)

Anche in questi test RSA si dimostra più veloce nella fase di verifica ma le differenze diminuiscono all'aumentare del livello di sicurezza (Security bits). D'altra parte ECDSA è molto più performante nelle operazioni di firma/decifratura.

Sempre nei laboratori di Sun Microsystems la versione ECC-*enhanced* di OpenSSL è stata impiegata per misurare le prestazioni del web server Apache con

traffico HTTPS [9]. I risultati ottenuti mostrano che, con *workloads* realistici, l'Apache può gestire fino al 31% di richieste HTTPS in più quando si usa ECC-160 al posto di RSA-1024. Al livello di sicurezza richiesto dal 2010 invece l'utilizzo di ECC-224 al posto di RSA-2048 introduce miglioramenti che vanno ben oltre il 120%. Tra le varie misurazioni effettuate in [9] si è scelto di visualizzare, vedi Figura 6.1, i tempi richiesti dall'Apache per soddisfare una richiesta HTTPS, al variare della dimensione della pagina web.

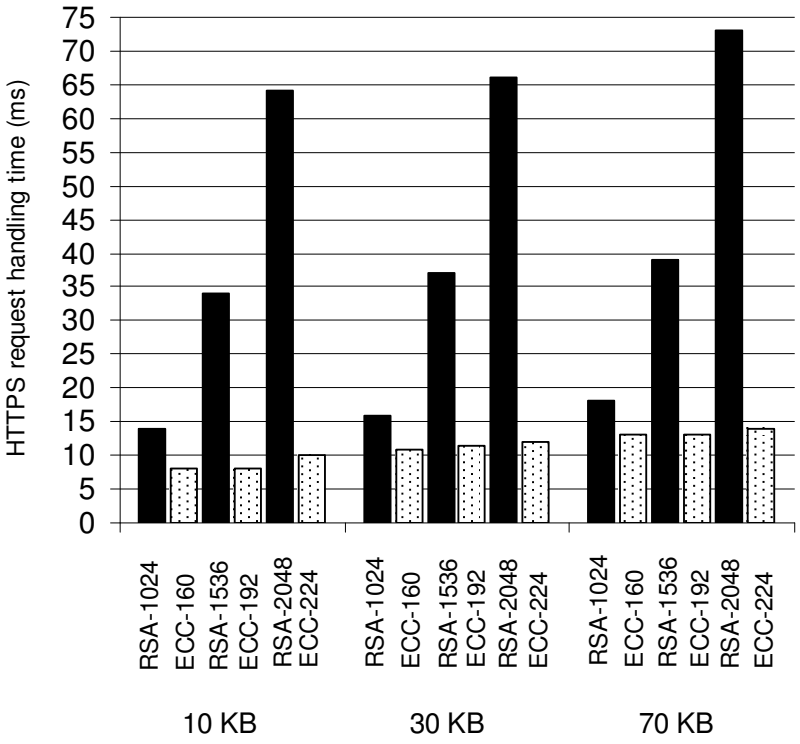

Figura 6.1 Tempi per richieste HTTPS

Una volta di più si vede che il divario di prestazioni fra ECC e RSA cresce all'aumentare del livello di sicurezza.

6.2 Conclusioni

Esattamente vent'anni fa nasceva la Crittografia basata sulle Curve Ellittiche. Inizialmente il suo studio è rimasto confinato negli ambienti accademici, ma in poco tempo ha suscitato l'interesse degli organismi internazionali (ANSI, ISO, etc.) e di numerose società che operano nel campo della sicurezza informatica e delle telecomunicazioni. Il motivo di tanto interesse ormai lo conosciamo bene, l'ECC si basa su un problema matematico (ECDLP) assai difficile, molto più difficile di quelli che hanno determinato il successo di RSA e DSA. Questo significa che ECC garantisce la stessa sicurezza con chiavi molto più piccole. In questi anni, inoltre, l'ECC è stata oggetto di numerose ricerche e numerosi sono stati gli studi sui possibili attacchi all'ECDLP. Tutto ciò ha contribuito ad aumentare la credibilità dell'ECC.

Attualmente i sistemi a chiave pubblica tradizionale (RSA, DSA, Diffie-Hellman) sono in assoluto i più utilizzati e la diffusione dell'ECC è ancora marginale, ma tre semplici considerazioni fanno prevedere un impiego sempre maggiore di tale soluzione:

- la diffusione di dispositivi wireless è in continua crescita
- i requisiti di sicurezza sono sempre più stringenti
- all'aumentare del livello di sicurezza richiesto cresce sensibilmente l'efficienza dei sistemi ECC rispetto a RSA/DSA.

In attesa di nuove soluzioni crittografiche, quindi, le prospettive per l'ECC sono promettenti. Può anche essere che nel frattempo venga scoperto un algoritmo polinomiale per risolvere l'IFP o meglio ancora l'ECDLP! Stiamo a vedere.

Appendice A

Concetti Matematici

In questa Appendice vengono ripresi i concetti matematici sui quali si basano gli algoritmi di crittografia asimmetrica. Le nozioni base di Teoria dei Numeri, di Algebra Astratta e di Complessità hanno utilità di carattere generale, mentre il concetto di discriminante viene utilizzato nello studio della Crittografia basata sulle Curve Ellittiche (ECC).

A.1 Teoria dei Numeri

Anzitutto ricordiamo che un intero $p \geq 2$ è *primo* se e solo se i suoi divisori sono 1 e p, altrimenti è detto *composito*. Due interi a e b, invece, sono detti *primi relativi* (o *coprimi*) se $\gcd(a,b) = 1$. Il fatto che l'intero a *divide* l'intero b si rappresenta con la notazione $a|b$. Definiamo ora il concetto di *congruenza*, molto importante nell'ambito dell'aritmetica in campi finiti.

Definizione A.1 (Congruenze) Dato un intero positivo m e altri due interi a e b, si dice che a *è congruente* a b *modulo* m, e si scrive $a \equiv b (\mathrm{mod}\, m)$, se e solo se $a \bmod m = b \bmod m$ cioè se il resto della divisione fra a ed m coincide con il resto della divisione fra b ed m; è altresì facile dimostrare che

$$a \equiv b (\mathrm{mod}\, m) \Leftrightarrow m|(b - a).$$

L'intero m è chiamato *modulus*.

Il concetto delle congruenze introduce l'*aritmetica modulare* dove appunto si usa la congruenza ≡ al posto dell'uguaglianza =. Questo significa che, dato il modulus m, ogni intero x viene sostituito da $x \bmod m$, ossia viene ridotto modulo m.

Definizione A.2 (*Funzione di Eulero* $\varphi(m)$) Se indichiamo con Z_m l'insieme degli interi modulo m $\{0,1,2, \ldots, m - 1\}$, la funzione $\varphi(m)$ ritorna il numero di elementi di Z_m che sono primi relativi ad m.

La funzione di Eulero sottolinea un aspetto molto importante dell'aritmetica modulare, infatti se in Z_m le operazioni di addizione e moltiplicazione vengono

eseguite modulo m, per ogni intero $a \in Z_m$ si definisce *inverso moltiplicativo* quel numero $b \in Z_m$ tale che $ab \equiv 1 \pmod{m}$. Se tale inverso esiste esso è unico, si indica con a^{-1} e a si dice *invertibile* (o *unità*). Inoltre vale il seguente:

Teorema A.1 Un intero $a \in Z_m$ è invertibile $\Leftrightarrow \gcd(a,m)=1$.

La funzione $\varphi(m)$, quindi, conta il numero di elementi di Z_m che ammettono inverso moltiplicativo; l'insieme di tali elementi viene rappresentato con $\mathbf{Z}_m^*$.

Appare evidente che se m è primo allora $\varphi(m) = m - 1$ e $\mathbf{Z}_m^* = \{1,2, \ldots, m-1\}$. Se invece m è dato dal prodotto di due numeri primi p e q, $m = pq$, è facile provare che $\varphi(m) = \varphi(p)\varphi(q) = (p-1)(q-1)$.
Citiamo ora un importante Teorema dimostrato da Eulero nel 1760:

Teorema A.2 (Eulero) Se $a \in \mathbf{Z}_m^*$ allora $a^{\varphi(m)} \equiv 1 \pmod{m}$

Un altro risultato notevole della Teoria dei Numeri, nasce dall'esigenza di risolvere simultaneamente più congruenze e porta al seguente

Teorema A.3 (del Resto Cinese)
Siano $n_1, n_2, \ldots, n_k$ interi coprimi fra loro tali che $N = n_1 n_2 \ldots n_k$ e siano $a_1, a_2, \ldots, a_k$ k interi qualunque. Allora, il seguente sistema di congruenze

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \quad \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

ammette un'unica soluzione modulo N pari a $x_0 = \sum_{i=1}^{k} a_i M_i y_i$,

dove $\forall i = 1,2,\cdots,k$ $M_i = N/n_i$ e y_i è tale che $M_i y_i \equiv 1 \pmod{n_i}$.

In diverse situazioni inoltre si è interessati a stabilire quali elementi di Z_m sono dei quadrati. A tale proposito si definisce

Definizione A.3 Siano dati il primo p e $x \in Z_p$. x è un *residuo quadratico modulo* p se esiste un $y \in Z_p$ tale che $y^2 \equiv x \pmod{p}$.

Un utile formalismo viene fornito dal *Simbolo di Legendre*.

Definizione A.4 Dati il primo p e l'intero a, si definisce *Simbolo di Legendre*:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \textit{se } a \equiv 0 \pmod{p} \\ 1 & \textit{se a è un residuo quadratico} \bmod p \\ -1 & \textit{se a non è un residuo quadratico} \end{cases}$$

Un efficiente metodo per calcolare il simbolo di Legendre è fornito dal seguente risultato, noto con il nome di *Criterio di Eulero*:

Teorema A.4 Se p è un primo, per un qualsiasi intero a vale

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p$$

A.2 Algebra Astratta

A.2.1 Gruppi

Definizione A.5 Un *gruppo* $(G, *)$ è costituito da un insieme di elementi G e da un operazione binaria $*$ che soddisfa le seguenti proprietà:

(i) Chiusura. $\forall a,b \in G$ vale $a * b \in G$.

(ii) Associatività. $\forall a,b,c \in G$ vale $a * (b * c) = (a * b) * c$

(iii) Identità. Esiste un elemento, detto *elemento identità*, $e \in G$ tale che $a * e = e * a = a \quad \forall a \in G$.

(iv) Inverso. $\forall a \in G$ esiste un elemento b tale che $a * b = b * a = e$

Quando vale anche la seguente:

(v) Commutatività. $\forall a,b \in G$ vale $a * b = b * a$

il gruppo è chiamato *abeliano*, o commutativo.

Se l'operazione binaria è l'addizione, il gruppo è detto *additivo*, l'elemento identità viene indicato con 0 e l'inverso di a con il simbolo $-a$. Se invece l'operazione considerata è la moltiplicazione, il gruppo si dice *moltiplicativo* e l'elemento identità e l'inverso di a vengono rappresentati rispettivamente con 1 e a^{-1}.

Definizione A.6 Dato il gruppo $(G, *)$ e H un sottoinsieme di G, se H è gruppo rispetto l'operazione $*$ allora si dice che $(H, *)$ è un *sottogruppo* di $(G, *)$.

Definizione A.7 Se G è un gruppo finito il numero di elementi di G è detto *ordine di G* e viene indicato con $\#G$.

Vediamo ora alcuni esempi di gruppi. L'insieme degli interi $\mathbb{Z}$ forma, con l'operazione di addizione, un gruppo additivo. L'insieme Z_m , con l'operazione di somma modulo m, costituisce un gruppo finito di ordine m. Lo stesso Z_m, però, non forma un gruppo con l'operazione di moltiplicazione, poiché non tutti gli elementi di Z_m ammettono inverso. L'insieme $\mathbb{Z}_m^*$ invece è un gruppo moltiplicativo di ordine $\varphi(m)$ con elemento identità 1.
Per le considerazioni successive useremo la notazione relativa ai gruppi moltiplicativi.

Definizione A.8 Dato il gruppo G, *l'ordine di un elemento* $a \in G$ è il più piccolo intero positivo n tale che $a^n = 1$, ammesso che esista. Se tale numero non esiste per definizione l'ordine di a è ∞.
Se quindi n è l'ordine di a, allora l'insieme

$$A = \left\{ a^k \,\middle|\, 0 \le k \le n \right\}$$

è un sottogruppo di G, *generato da a*, indicato con il simbolo $\langle a \rangle$.

Definizione A.9 Un gruppo G si dice *ciclico* se esiste un elemento $g \in G$ tale che $\forall\ a \in G$ esiste un intero i per il quale si ha $a = g^i$; l'elemento g è chiamato *generatore di G*.

In particolare si dimostra che se m è primo allora $\mathbb{Z}_m^*$ è ciclico.

Un Teorema molto importante, che correla l'ordine di un gruppo a quello dei suoi sottogruppi, e quindi all'ordine dei sottogruppi del tipo $\langle a \rangle$, con $a \in G$, è il seguente:

Teorema A.5 (Lagrange) Se G è un gruppo finito e H è un suo sottogruppo allora l'ordine di H divide l'ordine di G, in breve $\#H \mid \#G$. Pertanto se $a \in G$ l'ordine di a divide $\#G$.

Vediamo ora come è possibile generare nuovi gruppi a partire da altri

Definizione A.10 Dati due gruppi G_1 e G_2 si definisce *somma diretta* l'insieme di coppie ordinate formate da elementi di G_1 e G_2:

$$G_1 \oplus G_2 = \{(g_1, g_2) \mid g_1 \in G_1 \wedge g_2 \in G_2\}$$

Coppie ordinate posso essere sommate, a livello di componente:

$$(g_1, g_2) + (h_1, h_2) = (g_1 + h_1, g_2 + h_2)$$

In questo modo $G_1 \oplus G_2$ costituisce un gruppo con (0,0) come elemento identità. Analogamente si può definire la somma diretta di più di due gruppi.

A.2.2 Anelli e Campi

Definizione A.11 Un *anello* $(R,+,\cdot)$ è costituito da un insieme R, insieme a due operazioni binarie + (addizione) e · (moltiplicazione) definite su R che soddisfano:

(i) $(R,+)$ è un gruppo abeliano con elemento identità 0.

(ii) L'operazione · è chiusa: $\forall\, a,b \in R$ vale $a \cdot b \in R$

(iii) L'operazione · è associativa. $\forall\, a,b,c \in R$ vale $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

(iv) Identità di · Esiste un elemento indicato con 1 tale che
$a \cdot 1 = 1 \cdot a = a \;\; \forall\, a,b \in R$.

(v) Distributività di · rispetto + . $\forall\, a,b,c \in R$ vale
$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ e $(b + a) \cdot c = (b \cdot c) + (a \cdot c)$

Se vale anche

(vi) L'operazione · è commutativa. $\forall\, a,b \in R$ vale $a \cdot b = b \cdot a$

l'anello viene detto *commutativo*.

Esempi di anelli commutativi sono l'insieme degli interi $\mathbb{Z}$, l'insieme dei numeri reali $\mathbb{R}$ e anche l'insieme Z_m, con le operazioni di addizione e moltiplicazione modulo *m*.

Definizione A.12 Un *campo* è un anello commutativo *F* nel quale ogni elemento $a \in F$ diverso da zero è invertibile.

Gli insiemi dei numeri reali e numeri razionali $\mathbb{R}$ e $\mathbb{Q}$ sono esempi di campi con le consuete operazioni di addizione e moltiplicazione; i numeri interi $\mathbb{Z}$ invece non costituiscono un campo, poiché gli unici elementi che ammettono inverso moltiplicativo sono 1 e -1. Per quanto riguarda l'insieme Z_m degli interi modulo *m*, si dimostra che l'anello Z_m è un campo se e solo se *m* è primo [10].

Definizione A.13 La *caratteristica* di un campo *F* è 0 se la somma $\overbrace{1+1+1+\cdots+1}^{m}$ non è mai uguale a 0, per qualunque valore *m*; altrimenti la caratteristica coincide con il più piccolo intero *m* tale che $\sum_{i=1}^{m} 1 = m \cdot 1 = 0$.

La caratteristica di un campo *F* viene indicata con il simbolo char(*F*).

Tornando agli esempi visti sopra, si capisce che $\mathbb{R}$ e $\mathbb{Q}$ sono campi con caratteristica uguale a 0, mentre se *m* è primo il campo Z_m ha caratteristica pari a *m*. Più in generale, si dimostra che la caratteristica di un qualunque campo *F* è 0 oppure un numero primo *p* e si dice che *F* contiene rispettivamente l'insieme dei razionali $\mathbb{Q}$ oppure il campo Z_p [10].

Definizione A.14 Un sottoinsieme *H* di un campo *F* è un *sotto-campo di F* se è a sua volta un campo rispetto alle stesse operazioni (+, ·); in tal caso *F* viene detto *campo estensione* di *H*. Se $H \neq F$ allora *H* è un *sotto-campo proprio*, altrimenti se *F* non ammette sotto-campi propri viene chiamato *campo primo*.

Definizione A.15 L'*anello dei polinomi* $\mathbb{F}[x]$ con x sul campo F è l'insieme di tutte le espressioni formali $f(x) = a_0 + a_1 x + \cdots + a_n x^n$, $(n \geq 0)$ dove $a_i \in F$.

Definizione A.16 Il campo F è *algebricamente chiuso* se ogni $f(x) \in \mathbb{F}[x]$ ha una radice in F.

Definizione A.17 Dato L campo estensione di F, un elemento $a \in L$ si definisce *algebrico su* K se esiste un polinomio non nullo $f(x)$ con coefficienti in K tale che $f(a) = 0$. Gli elementi di L che non sono algebrici vengono chiamati *trascendentali*. Il campo L si dice *estensione algebrica di* K se ogni elemento di L è algebrico su K.

Definizione A.18 La *chiusura algebrica* di un campo K si indica con $\overline{K}$ ed è tale che:

- $\overline{K}$ è estensione algebrica di K.
- $\overline{K}$ è algebricamente chiusa.

Si può dimostrare che ogni campo K ammette chiusura algebrica e che tale chiusura è unica, a meno di isomorfismi (vedi paragrafo successivo).

Nel caso K sia il campo finito F_p, si dimostra che vale

$$\overline{F}_p = \bigcup_{n \geq 1} F_{p^n}$$

Come esempi, si osservi che il campo dei numeri reali $\mathbb{R}$ non è algebricamente chiuso (il polinomio $x^2 + 1 = 0$ non ammette zeri reali). Il *Teorema Fondamentale dell'Algebra*, invece, dimostra che il campo dei numeri complessi $\mathbb{C}$ è chiuso ed è chiusura algebrica di $\mathbb{R}$.

A.2.3 Isomorfismi

Dati due anelli R e S, una mappatura θ: $R \to S$ è detta *omomorfismo* se, $\forall\ a,b \in R$ valgono:

$$\theta(a + b) = \theta(a) + \theta(b)$$
$$\theta(ab) = \theta(a)\ \theta(b)$$

Informalmente si dice che un omomorfismo fra R e S è una mappatura che conserva l'addizione e la moltiplicazione.
E' facile dimostrare che se 0_R e 0_S sono rispettivamente l'elemento 0 di R e di S, si ha $\theta(0_R) = 0_S$; l'analogo per l'identità però non vale, cioè non è necessariamente verificato che $\theta(1_R) = 1_S$.

Definizione A.19 Si definisce *isomorfismo* un omomorfismo θ: $R \to S$ biiettivo, che quindi crea una corrispondenza uno-a-uno fra gli elementi di R e quelli di S. In tal caso R e S si dicono *isomorfici* e si scrive $R \cong S$.

In sostanza due anelli sono isomorfici se hanno la stessa struttura, sebbene con una rappresentazione diversa dei rispettivi elementi.

A.2.4 Campi finiti

Definizione A.20 Un *campo finito* è un campo F che contiene un numero finito n di elementi.

Una caratteristica molto importante dei campi finiti è riassunta dal seguente

Criterio di esistenza ed unicità dei campi finiti

(i) se F è un campo finito, allora contiene p^m elementi dove p è un numero primo ed m un intero maggiore o uguale a 1.

(ii) per ogni potenza p^m, con p primo, esiste un unico (a meno di isomorfismi) campo finito di ordine p^m. Tale campo si indica con F_{p^m} o $GF(p^m)$[6].

[6] La notazione *GF* è abbreviazione di Galois Field, in onore del matematico francese Evariste Galois.

Ricordiamo che se p è un numero primo Z_p è un campo e perciò un qualsiasi campo F_p di ordine p è isomorfico a Z_p, cioè $F_p \cong Z_p$. Più in generale si dimostra che un qualunque campo finito F è estensione di Z_p, per un particolare intero primo p.

Infine se un campo finito F_q ha ordine $q = p^n$, con p primo, la caratteristica di F_q è pari a p.

A.3 Discriminante di un polinomio

Il *discriminante* di un polinomio è il prodotto dei quadrati delle differenze delle radici. Quindi, considerato il polinomio di grado n:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

si definisce discriminante:

$$\Delta = a_n^{2n-2} \prod_{i<j}^{n} (x_i - x_j)^2$$

dove le x_i sono radici del polinomio.

Ne consegue, che imporre $\Delta \neq 0$ significa imporre che il polinomio non abbia radici multiple.

In particolare, lo studio delle curve ellittiche porta ad analizzare il polinomio cubico $x^3 + ax + b$. Se indichiamo le sue radici con x_1, x_2, x_3, semplici calcoli provano che il corrispondente discriminante vale:

$$\Delta = ((x_1 - x_2)(x_1 - x_3)(x_2 - x_3))^2 = -(4a^3 + 27b^2)$$

A.4 Complessità degli Algoritmi

L'efficienza di un algoritmo viene misurata in base alle risorse che utilizza. Tipicamente la risorsa considerata è il *tempo* ma può anche essere la memoria richiesta o il numero di processori impiegati. E' ragionevole aspettarsi che le risorse utilizzate crescano al crescere delle dimensioni dei valori di *input*, perciò l'efficienza può considerarsi una funzione di tali dimensioni. In questo contesto le dimensioni corrispondono al numero di *bits* necessari a rappresentare in notazione binaria il valore di input. Ricordo che un intero n è rappresentabile con ℓ bits,

dove $\ell = \lfloor \log n \rfloor + 1$; tale valore viene chiamato *bitlength* di *n*. Diamo ora una definizione di tempo di esecuzione (*running time*) che prescinde dalla piattaforma sulla quale implementare l'algoritmo.

Definizione A.21 Il *running time* di un algoritmo, con un particolare input, è uguale al numero di passi elementari eseguiti.

Poiché è difficile dare un'espressione esatta del running time, si fa ricorso alla notazione asintotica e più precisamente alla notazione *big-O* e *little-o*. Considerate due funzioni *f* e *g* definite sui positivi interi e a valori reali positivi, diamo le seguenti definizioni.

Definizione A.22 Vale $f(n)=O(g(n))$ se esistono due costanti positive *c* ed *L* tali che $f(n) \leq cg(n)$ per ogni $n \geq L$.

Definizione A.23 Vale $f(n)=o(g(n))$ se per ogni costante positiva *c* esiste una costante *L* tale che $f(n) \leq cg(n)$ per ogni $n \geq L$.

In sostanza, $f(n)=O(g(n))$ comporta che asintoticamente $f(n)$ è limitata da $g(n)$, a meno di un fattore costante. La scrittura $f(n)=o(g(n))$, invece, significa che $f(n)$ diventa trascurabile rispetto a $g(n)$, per *n* che tende a $+\infty$.

Classi di Complessità

Dato un algoritmo *A* con un input *n*, di $\lfloor \log n \rfloor + 1$ bits, definiamo alcune classi di complessità:

Definizione A.24 *A* è un algoritmo *polynomial-time* se il suo running time è $O((\log n)^c)$ con $c > 0$.

Definizione A.25 *A* è un algoritmo *exponential-time* se il suo running time non è nella forma $O((\log n)^c)$, per qualunque $c > 0$.

A loro volta gli algoritmi *esponenziali* si dividono in

- *subexponential-time* se il loro running time è $O(2^{o(\log n)})$
- *fully-exponential-time* se il running time non è nella forma $O(2^{o(\log n)})$.

Senza voler troppo generalizzare, vanno considerati efficienti gli algoritmi polynomial-time, mentre sono ritenuti inefficienti gli algoritmi exponential-time.
Vediamo ora un esempio di algoritmo *subexponential-time*: sia *A* un algoritmo che ha come input un elemento di un campo finito F_n o un intero *n* (quindi la dimensione dell'input è *O*(log *n*)). Se il running time di *A* è nella forma:

$$L_n[\alpha, c] = O(e^{(c+o(1))\ (\log\ n)^{\alpha}\ (\log\ \log\ n)^{1-\alpha}})$$

allora *A* è un algoritmo *subexponential-time*. Si osservi che per $\alpha = 0$ $L_n[0,c]$ diviene un'espressione polinomiale in *log n* e quindi *A* diventa polynomial-time. Nel caso $\alpha = 1$, invece, $L_n[1,c]$ è un espressione esponenziale in $\log n$ e *A* è un algoritmo *fully-exponential-time*. Il parametro α quindi consente di stimare il grado di efficienza di un algoritmo *subexponential-time*. L'espressione $L_n[\alpha,c]$ viene utilizzata per valutare gli algoritmi di risoluzione dei problemi IFP e DLP.

Appendice B

Legge di Gruppo delle Curve Ellittiche

Si consideri una curva ellittica *E*/*K*. Le formule algebriche per la cosiddetta *Legge di Gruppo* possono essere derivate dalla descrizione geometrica (vista nel Capitolo 3). Di seguito vengono presentate tali formule nel caso che il campo *K* abbia caratteristica diversa da 2 e 3 (ad esempio $K = GF(p)$) , oppure abbia caratteristica uguale a 2 (campi del tipo $K = GF(2^m)$). Se char(*K*) = 2, bisogna distinguere fra curva supersingolare e curva non-supersingolare. Solo per i campi *K* con $char(K) \neq 2,3$ vengono esplicitati i passaggi matematici.

B.1 Legge di Gruppo nel caso char(*K*) ≠ 2,3

Legge di Gruppo per *E*/*K*: $y^2 = x^3 + ax + b$, char(*K*) ≠ 2,3

a) Consideriamo prima la somma di due punti distinti P ≠ Q

Riferendoci alla Figura 3.4, siano (x_1, y_1) e (x_2, y_2) le coordinate rispettivamente dei punti *P* e *Q*. La retta *L* che passa per i punti *P* e *Q* ha pendenza

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Analizziamo prima il caso $x_1 \neq x_2$. Quindi l'equazione della retta *L* è $y = m(x - x_1) + y_1$. Per trovare le coordinate del punto di intersezione con la curva *E*, bisogna risolvere l'equazione cubica $(m(x - x_1) + y_1)^2 = x^3 + ax + b$ che può essere scritta come

$$0 = x^3 - m^2 x^2 + (\cdots)x + c \qquad (1)$$

Di tale cubica però, si conoscono già le due radici x_1 e x_2, quindi se fattorizziamo il polinomio della (1) si ottiene

$$x^3 - m^2 x^2 + (\cdots)x + c = (x - x_1)(x - x_2)(x - t) = x^3 - (x_1 + x_2 + t)x^2 + \cdots$$

dove *t* è la terza radice da calcolare. Si ha $t = m^2 - x_1 - x_2$ e quindi il punto R' ha coordinate

$$x = m^2 - x_1 - x_2$$
$$y = m(x - x_1) + y_1$$

Infine se si *riflette rispetto l'asse delle x* si ottiene il punto $R = (x_3, y_3) = P + Q$

$$x_3 = m^2 - x_1 - x_2$$
$$y_3 = m(x_1 - x_3) - y_1$$

Se invece $x_1 = x_2$ (e quindi $y_1 = -y_2$) , la retta L è verticale e quindi interseca E nel punto all'infinito Θ. Il simmetrico di Θ rispetto all'asse delle x è sempre Θ, pertanto si ottiene $P + Q = \Theta$.

b) Vediamo ora il caso $P = Q = (x_1, y_1)$.

In questa situazione bisogna considerare la retta L tangente alla curva E nel punto P. La pendenza m di tale retta si ricava con la derivazione implicita:

$$2y\frac{dy}{dx} = 3x^2 + a, \quad \text{quindi} \quad m = \frac{dy}{dx} = \frac{3x_1^2 + a}{2y_1}$$

Se $y_1 = 0$ la retta L è verticale e si definisce, come in precedenza, $P + P = \Theta$.

Se invece $y_1 \neq 0$, l'equazione della retta L è $y = m(x - x_1) + y_1$. Procedendo come nel caso $P \neq Q$ si ottiene l'equazione cubica

$$0 = x^3 - m^2x^2 + (\cdots)x + c \qquad (4.7)$$

Questa volta si conosce la sola radice x_1 che però è doppia (infatti L è tangente ad E in P).

Quindi, ripetendo i calcoli visti prima si ottiene il punto $R = (x_3, y_3) = P + P$, dove

$$x_3 = m^2 - 2x_1$$
$$y_3 = m(x_1 - x_3) - y_1$$

c) Infine consideriamo il caso $Q = \Theta$

La retta che passa per P e Θ è verticale ed interseca la curva E nel punto P', che è proprio il simmetrico di P rispetto all'asse delle x. Pertanto quando si riflette il punto P' rispetto all'asse delle x, per ottenere il punto somma $P + \Theta$, si ricava nuovamente P. Quindi, per ogni $P \in E(K)$ si ha

$$P + \Theta = P$$

Riassumendo, valgono le seguenti proprietà:

1. Identità: $P + \Theta = P \ \forall P \in E(K)$
2. Negativo: dato il punto $P = (x, y)$, allora $(x, y) + (x, -y) = \Theta$. Il punto $(x, -y)$ è indicato come $-P$ ed è chiamato *negativo* (*opposto*) di P; se $P \in E(K)$ anche $-P$ appartiene alla curva. Inoltre, per definizione $-\Theta = \Theta$.
3. Addizione: Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ punti appartenenti alla curva E/K, con $P \neq \pm Q$. Allora $R = P + Q = (x_3, y_3)$, dove

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

4. Raddoppio: Sia $P = (x_1, y_1) \in E(K)$ e $P \neq -P$. Allora $R = P + P = (x_3, y_3)$, dove:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \qquad \text{e} \qquad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

B.2 Legge di Gruppo nel caso char(K) = 2

Legge di Gruppo per *non-supersingolare E/GF*(2^m): $y^2 + xy = x^3 + ax^2 + b$

1. Identità: $P + \Theta = P \ \forall P \in E(GF(2^m))$
2. Negativo: sia $P = (x, y) \in E(GF(2^m))$, allora $(x, y) + (x, x + y) = \Theta$. Il punto $(x, x + y)$ è indicato come $-P$ ed è chiamato *negativo* di P; se $P \in E(GF(2^m))$ anche $-P$ appartiene alla curva. Inoltre $-\Theta = \Theta$.
3. Addizione: Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ punti appartenenti a $E(GF(2^m))$, con $P \neq \pm Q$. Allora $R = P + Q = (x_3, y_3)$, dove

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad \text{e} \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

con $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

4. Raddoppio: Sia $P = (x_1, y_1) \in E(GF(2^m))$ e $P \neq -P$. Allora $R = P + P = (x_3, y_3)$, dove

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \qquad \text{e} \qquad y_3 = x_1^2 + \lambda x_3 + x_3$$

con $\lambda = x_1 + y_1/x_1$.

Legge di Gruppo per *supersingolare E/GF*(2^m): $y^2 + cy = x^3 + ax + b$

1. Identità: $P + \Theta = P \ \forall P \in E(GF(2^m))$
2. Negativo: sia $P = (x, y) \in E(GF(2^m))$, allora $(x, y) + (x, \ y + c) = \Theta$. Il punto $(x, \ y + c)$ è indicato come $-P$ ed è chiamato *negativo* di P; se $P \in E(GF(2^m))$ anche $-P$ appartiene alla curva. Inoltre $-\Theta = \Theta$.
3. Addizione: Siano $P = (x_1, y_1)$ e $Q = (x_2, y_2)$ punti appartenenti a $E(GF(2^m))$, con $P \neq \pm Q$. Allora $R = P + Q = (x_3, y_3)$, dove

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + x_1 + x_2 \qquad \text{e} \qquad y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + y_1 + c$$

4. Raddoppio: Sia $P = (x_1, y_1) \in E(GF(2^m))$ e $P \neq -P$. Allora $R = P + P = (x_3, y_3)$, dove

$$x_3 = \left(\frac{x_1^2 + a}{c}\right)^2 \qquad \text{e} \qquad y_3 = \left(\frac{x_1^2 + a}{c}\right)(x_1 + x_3) + y_1 + c$$

Sebbene le regole di addizione appena viste abbiano espressioni algebriche poco intuitive, il seguente Teorema stabilisce proprietà molto importanti di cui godono [33].

Teorema

L'addizione di punti in una curva ellittica E soddisfa le seguenti proprietà:

1. *Commutativa*: $P_1 + P_2 = P_2 + P_1 \ \ \forall P_1, P_2 \in E$
2. *Esistenza dell'identità*: $P + \Theta = P \quad \forall P \in E$
3. *Esistenza dell'opposto*: $\forall P \in E$ esiste un $P' \in E$ tale che $P + P' = \Theta$. Il punto P' viene indicato con $-P$.
4. *Associatività*: $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \ \ \forall P_1, P_2, P_3 \in E$

In altre parole i punti di una curva ellittica E, insieme all'operazione di addizione, formano un gruppo abeliano, con Θ elemento identità.

Bibliografia

[1] R. Akhtar, “An Introduction to Elliptic Curves”, Miami University

[2] I. Blake, G. Seroussi, N. P. Smart, “Advances in Elliptic Curve Cryptography”, Cambridge University Press, 2005.

[3] D. Boneh, R. Venkatesan, “Breaking RSA may be easier than factoring”

[4] Certicom, “Elliptic Curve Cryptosystem”, http://www.certicom.com

[5] Certicom, Educational newsletter “Code & Cipher”, http://www.certicom.com/codeandcipher

[6] W. Chou, “Elliptic Curve Cryptography and Its Applications to Mobile Devices”, http://www.cs.umd.edu/Honors/reports/ECCpaper.pdf

[7] W. Diffie, M. Hellman, “New Directions in Cryptography”, 1976, http://citeseer.ist.psu.edu

[8] V. Gupta, S. Gupta, S. Chang, “Performance Analysis of Elliptic Curve Cryptography for SSL”, http://research.sun.com/projects/crypto

[9] V. Gupta et al., “Speeding up Secure Web Transactions Using Elliptic Curve Cryptography”, http://research.sun.com/projects/crypto

[10] R. Howlett, “An Undergraduate Course in Abstract Algebra”, http://www.maths.usyd.edu.au/u/bobh/UoS/rfwhole.pdf

[11] IETF, “ECC Cipher Suites for TLS”, www.ietf.org/internet-drafts/draft-ietf-tls-ecc-12.txt

[12] IETF, “ECP Groups for IKE and IKEv2”, www.ietf.org/internet-drafts/draft-ietf-ipsec-ike-ecp-groups-02.txt

[13] G.A. Jones, J.M. Jones “Elementary Number Theory”, Springer, 2005

[14] N. Koblitz, “A Course in Number Theory and Cryptography”, Springer, 1994

[15] N. Koblitz, A. Menezes, “A Survey of Public-key Cryptosystems” , http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications

[16] N. Koblitz, A. Menezes, S. Vanstone, “The State of Elliptic Curve Cryptography”, Design Codes and Cryptography, 19,173-193, 2000

[17] J. Krasner, “Using Elliptic Curve Cryptography for Enhanced Embedded Security”, http://www.embedded-forecast.com/EMF-ECC-FINAL1204.pdf

[18] K. Lauter, “The Advantage of Elliptic Curve Cryptography for Wireless Security”, IEEE Wireless Communications, February 2004.

[19] J. Lopez, R. Dahab, “An Overview of Elliptic Curve Cryptography”,

[20] A. Menezes, D. Hankerson, S. Vanstone, “Guide to Elliptic Curve Cryptography”, Springer, 2004

[21] A. Menezes, “Evaluation of Security Level of Cryptography: The Elliptic Curve Discrete Logarithm Problem”, http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1028_ecdlp.pdf

[22] A. Menezes, P. van Oorschot, S. Vanstone, “Handbook of Applied Cryptography”, CRC Press 1997.

[23] T.K. Meng, “Curves for the Elliptic Curve Cryptosystem”, National University of Singapore, Thesis 2000/2001

[24] NIST, FIPS 186-2 “Digital Signature Standard”, 2000

[25] NIST, SP 800-57 “Reccomendation for Key Management”, August 2005

[26] E. Oswald, “Introduction to Elliptic Curve Cryptography”, http://www.iaik.tu-graz.ac.at/aboutus/people/oswald/

[27] H.J.J. Riele, “Factoring Large Numbers: Fun or Applied Science?”, http://dbs.cwi.nl

[28] A. Shikfa, “Billinear Pairings over Elliptic Curves”, Ecole doctorale STIC de Nice Sophia-Antipolis, Research Master Thesis, 2005

[29] J.H Silverman, “The Arithmetic of Elliptic Curves”, Spinger-Verlag, 1986

[30] S. Singh, “Codici e Segreti”, Rizzoli, 1999

[31] W. Stallings, “Crittografia e Sicurezza delle Reti”, McGraw-Hill, 2004

[32] D.R. Stinson, “Cryptography, Theory and Practice”, CRC Press, 1995

[33] L. C. Washington, “Elliptic Curves, Number Theory and Cryptography”, CRC Press, 2003.

[34] R. Zuccherato, “Using a PKI Based upon Elliptic Curve Cryptography”, http://www.entrust.com, 2004