Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Informatica

Crittografia basata su curve ellittiche e implementazione di funzioni di libreria per Cryptokit

Tesi di Laurea in Sicurezza e Crittografia

Relatore:
Chiar.mo Dott.
UGO DAL LAGO

Presentata da:
MARGHERITA LAZZARINI

Co-relatore:
Chiar.mo Dott.
DAVIDE ALIFFI

I Sessione
Anno Accademico 2011 - 2012

Alla mia famiglia e agli amici,
che mi hanno sostenuto in questo percorso

Indice

Elenco delle figure

Capitolo 1

Prima parte

1.1 Introduzione

La crittografia sulle curve ellittiche (ECC) costituisce una tipologia di crittografia a chiave pubblica basata su curve ellittiche definite su campi finiti. Una curva ellittica è una curva algebrica: possiede un'operazione di somma fra punti appartenenti alla curva, rispetto alla quale essa è un gruppo abeliano il cui elemento neutro è costituito dal punto all'infinito O. L'utilizzo di questo metodo crittografico è stato proposto da Neal Koblitz e Victor S. Miller nel 1985.

Le curve ellittiche sono inoltre utilizzate in diversi metodi di fattorizzazione di numeri interi, come ad esempio il metodo di Lenstra.

Definizione 1.1 (Classe di resto)**.** Una classe di resto $[a]_m$ costituisce l'insieme degli interi congrui ad a modulo m.

E.g. $[5]_7 = \{x \in \mathbb{Z} | 7|x-5\}$

$\mathbb{Z}/p\mathbb{Z}$ (o $\mathbb{Z}_p$) denota l'insieme delle classi di resto modulo p.

Definizione 1.2 (Campo)**.** Un campo è una struttura algebrica composta da un insieme non vuoto K e da due operazioni binarie interne, le quali soddisfano le proprietà associativa e commutativa, l'esistenza dell'elemento neutro e degli inversi.

Definizione 1.3 (Campo finito)**.** Un campo finito F_q denota un campo con un numero q di elementi.

In particolare, se $F_q = \mathbb{Z}/q\mathbb{Z}$, F_q contiene le classi di resto modulo q.

Definizione 1.4 (Caratteristica di un campo)**.** La caratteristica di un campo è definita come il più piccolo naturale n tale che $1 + 1 + 1 + ... + 1$ (n volte)$= 0$. Se non esiste, la caratteristica è 0 per definizione.

E.g. $\mathbb{Z}/p\mathbb{Z}$ ha caratteristica p con p primo.

Sia K un campo con caratteristica diversa da 2 e 3, e sia $x^3 + ax + b$ un polinomio senza radici multiple, con a,b appartenenti a K. Una curva ellittica su K è costituita dai punti (x,y) che soddisfano l'equazione

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

assieme ad un punto O, chiamato punto all'infinito.

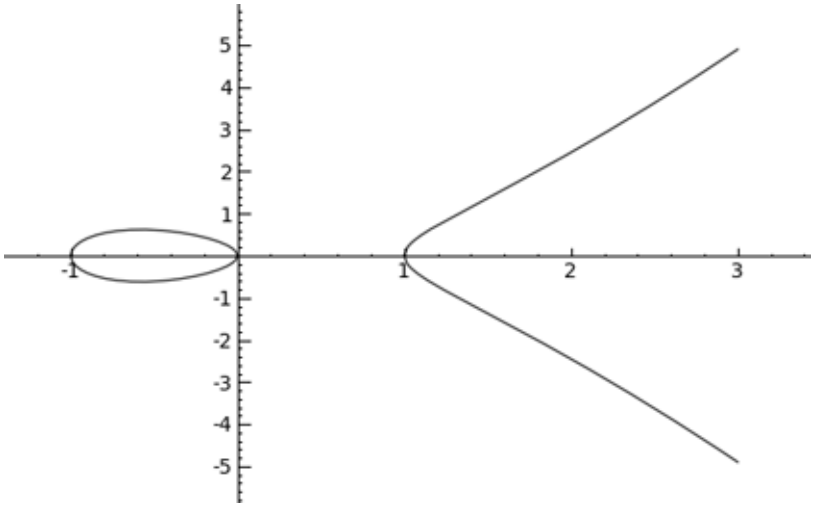

Figura 1.1: La curva $y^2 = x^3 - x$

Osservazione 1.
Se K è un campo di caratteristica 2, allora una curva ellittica su K è l'insieme dei punti che soddisfa l'equazione:

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

assieme al punto all'infinito O. In questo caso, $x^3 + ax + b$ può anche avere radici multiple.

1.1.1 Curve ellittiche su $\mathbb{R}$

Prima di discutere le curve ellittiche definite su campi finiti, ovvero le curve di interesse crittografico, assumiamo per il momento che $K = \mathbb{R}$, ovverosia che E sia una curva ordinaria nel piano cartesiano e che quindi i punti $\in E$ abbiano coordinate reali.

Sia E una curva ellittica su $\mathbb{R}$, e siano $P, Q \in E$. E' possibile definire una somma sui punti di E, in modo da ottenere un gruppo:

1. Se $P \equiv O$, allora $-P = O$ e $P + Q = Q$, cioè O è l'elemento neutro del gruppo formato dai punti della curva.

2. -P ha la stessa ascissa di P e la ordinata opposta di P. E.g. Se $P(x, y)$, $-P(x, -y)$

3. La retta passante per P e Q interseca la curva in un terzo punto R. Si definisce $P + Q \equiv -R$. Se $P(x_1, y_1)$, $Q(x_2, y_2)$ e $P + Q(x_3, y_3)$, valgono le seguenti formule:

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

$$y_3 = -y1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$

Osservazione 2.
Se $P \equiv Q$, la retta passante per P e Q è tangente alla curva e si ha

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

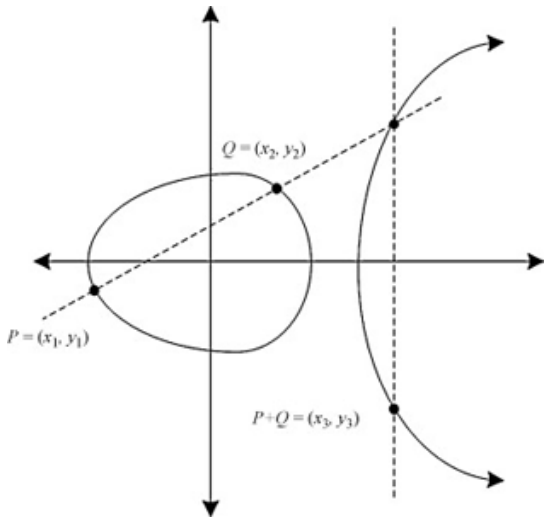

Figura 1.2: Somma di due punti su una curva ellittica

$$y_3 = -y1 + \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3)$$

con a coefficiente della x in E.

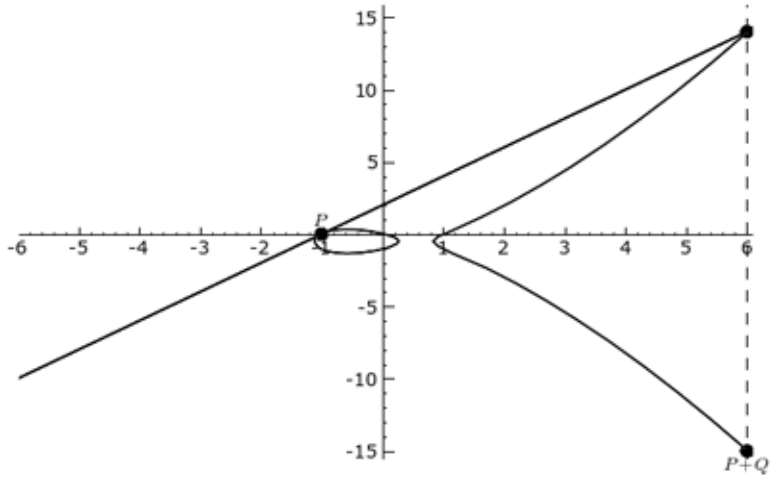

Figura 1.3: Somma di due punti coincidenti sulla curva $y^2 + y = x^3 - x$

1.1.2 Curve ellittiche in campi finiti

Sia F_q un campo finito avente $q = p^r$ elementi, e sia E una curva ellittica definita su F_q. Allora i punti appartenenti ad E formano un gruppo abeliano finito, dotato di operazione somma come definita in precedenza, e avente come elemento neutro il punto all'infinito O.

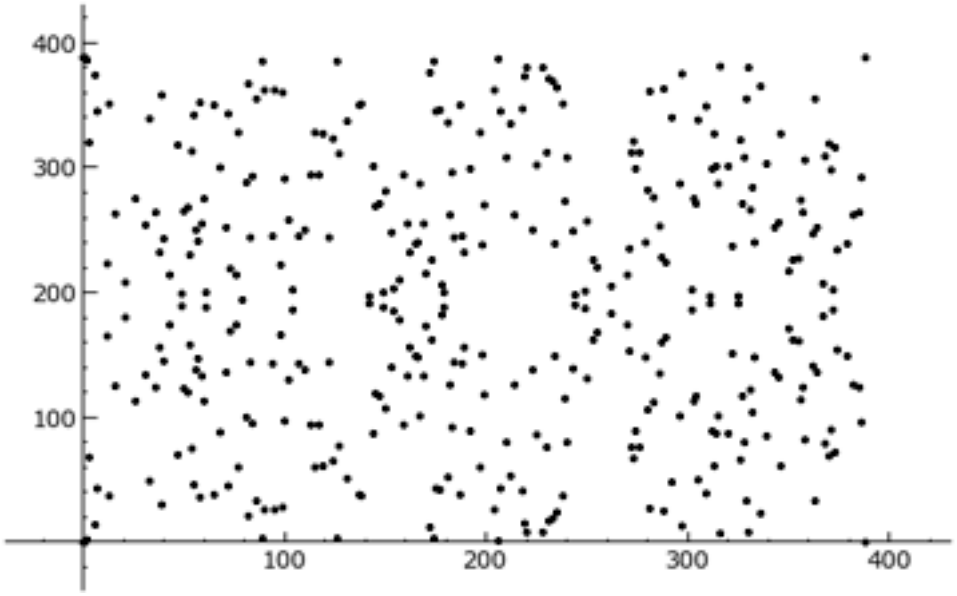

Figura 1.4: La curva $y^2 + y = x^3 - x$ definita sul campo finito F_{389}

Il grafico della curva definita su F_q non è più una curva liscia. Tutte le operazioni, quali addizione, sottrazione, divisione e moltiplicazione coinvolgono interi compresi fra 0 e $p - 1$, poichè le operazioni si intendono in aritmetica modulare. Pertanto, le formule che valgono per $\mathbb{R}$ devono essere riadattate per le curve ellittiche sui campi finiti mediante le seguenti operazioni aritmetiche:

$$x_3 = \left((y_2 - y_1) * (x_2 - x_1)^{-1}\right)^2 - x_1 - x_2 \mod q$$

$$y_3 = -y1 + \left((y_2 - y_1) * (x_2 - x_1)^{-1}\right) * (x_1 - x_3) \mod q$$

Mentre per $P \equiv Q$:

$$x_3 = \left((3x_1^2 + a) * (2y_1)^{-1}\right)^2 - 2x_1 \mod q$$

$$y_3 = -y1 + \left((3x_1^2 + a) * (2y_1)^{-1}\right) * (x_1 - x_3) \mod q$$

dove per $(x_2 - x_1)^{-1}$ o $(2y_1)^{-1}$ si intende l'inverso moltiplicativo nel campo F_q.

Definizione 1.5 (Inverso moltiplicativo)**.** Data la classe di resto $[a]_q$ in $\mathbb{Z}_q$, $[a]_q^{-1} = [x]_q$ tale che $[a]_q * [x]_q = [1]_q$.

E.g. $[5]_7 * [3]_7 = [15]_7 = [1]_7$

Una curva ellittica di questo tipo può avere, al più, $2q + 1$ punti (il punto all'infinito più $2q$ coppie (x,y) con $x,y \in F_q$ e che soddisfano l'equazione $y^2 = x^3 + ax + b$). Per contare i punti su una curva ellittica, un importante risultato è stato raggiunto con la formulazione del teorema di Hasse.

Teorema 1 (Teorema di Hasse)**.** *Sia N il numero di punti $\in F_q$ su una curva ellittica definita su F_q. Allora*

$$|N - (q + 1)| \leq 2\sqrt{q}$$

1.2 Crittografia ellittica

1.2.1 Multipli di punti

Definizione 1.6 (Ordine di un elemento di un gruppo)**.** Dato un gruppo G e un suo elemento e si definisce ordine di e il minimo numero intero i per il quale $e^i = I$ (dove I è l'elemento neutro di G).

Definizione 1.7 (Ordine di un punto in una curva ellittica)**.** L'ordine N di un punto P su una curva ellittica è il più piccolo intero tale che $NP = O$.

Proposizione 1.
Sia F_q, ove $q = p^f$, un campo finito con caratteristica p primo, e sia $F(X)$ un polinomio irriducibile di grado f definito su F_p. Allora, due elementi di F_q possono essere moltiplicati o divisi in $O(\log^3 q)$ operazioni binarie. Se k è un intero positivo, allora un elemento di F_q può essere elevato alla k-esima potenza in $O(\log k \log^3 q)$ operazioni bit.

Come nel problema del logaritmo discreto, l'elevamento alla k-esima potenza in F_q può essere effettuato con il metodo delle quadrature ripetute in $O(\log k \log^3 q)$ operazioni bit, allo stesso modo si dimostra che il punto $kP \in E$ può essere trovato in $O(\log k \log^3 q)$ operazioni bit grazie al metodo del raddopio ripetuto.

E.g. Per trovare $100P = 2(2(1 + 2(2(2(1 + 2P)))))$

Infatti il calcolo della somma di due punti richiede meno di 20 operazioni (fra moltiplicazioni, addizioni, divisioni e sottrazioni). Quindi, per la proposizione 1 ogni addizione di punti richiede $O(\log^3 q)$. Siccome il metodo del raddopiamento ripetuto richiede $O(\log k)$ passi, possiamo concludere che le coordinate di kP possono essere calcolate con $O(\log k \log^3 q)$ operazioni.

1.2.2 Codifica del testo in chiaro

Vogliamo far sì che al nostro testo in chiaro corrisponda un punto P_m sulla curva ellittica E definita in un campo finito F_q. Non esiste un algoritmo deterministico che trova molti punti su una curva ellittica arbitraria in un tempo

polinomiale ($\log q$). Perciò si utilizza un algoritmo probabilistico con possibilità di fallimento molto basse. Supponiamo k grande abbastanza in modo che la probabilità di fallimento nell'integrazione del testo in chiaro m sia di 1 su 2^k (e.g. $k = 50$), e $0 \leq m < M$ con M che rappresenta l'alfabeto del nostro testo in chiaro. Supponiamo inoltre che F_q (con $q = p^r$) sia tale che $q > Mk$. Rappresentiamo gli interi da 1 a Mk nella forma $mk + j$ con $1 \leq j \leq k$. Mentre, per predisporre una corrispondenza 1-1 tra questi interi e gli elementi di F_q, scriviamo m come intero in base p e consideriamo le cifre di $(m)_p$ come un polinomio di grado $r - 1$ (con r = numero di cifre di $(m)_p$).

$$m = (a_0 a_1 ... a_{r-1}) \mapsto x_m = a_0 x^{r-1} + a_1 x^{r-2} + ... + a_{r-1}$$

Non è detto che x_m sia un punto sulla curva. Per questo, si cerca un punto x_1 vicino a x_m in modo efficiente tale che $x_m \leq x_1 \leq x_m + k$. Una volta trovato il punto di coordinate (x_1,y_1), è possibile risalire al testo in chiaro conoscendo k:

$$x_m = \left\lfloor \frac{x_1}{k} \right\rfloor$$

1.2.3 Scelta di un punto e di una curva

Vengono ora illustrati due metodi per la scelta di una curva ellittica E e di un punto $B \in E$.

- Una volta scelto F_q avente caratteristica > 3, siano x, y, a tre elementi di F_q scelti in modo random. Poniamo $b = y^2 - (x^3 + ax)$ e controlliamo che $x^3 + ax + b$ non abbia radici multiple, che equivale a

  $$4a^3 + 27b^2 \neq 0$$

  Altrimenti, scegliamo altri x, y, a. A questo punto, $B = (x, y)$ e $E : y^2 = x^3 + ax + b$.

- Viene scelta una curva ellittica E su Q e il relativo punto all'infinito O. Dopodichè viene scelto un primo grande p e si effettua la riduzione di E e B modulo p; più precisamente, i coefficienti dell'equazione di E vengono ridotti mod p. Se si effettua un cambio di variabili portando l'equazione

risultante su F_p alla forma $y^2 = x^3 + ax + b$, la cubica $x^3 + ax + b$ non ha radici multiple (eccetto che per alcuni p piccoli), e così è data una curva ellittica, denotata con E mod p su F_p.

Osservazione 3.
Se il numero N di punti di $F_q \in E$ è il prodotto di numeri primi relativamente piccoli, il metodo di Pohlig-Silver-Hellman può essere usato per risolvere il problema del logaritmo discreto. Per trovare N, può essere utilizzato l'algoritmo di Schoof; pubblicato da Renè Schoof nel 1985, costituisce una svolta teorica in quanto è il primo algoritmo deterministico e polinomiale per contare i punti di $F_q \in E$. Tuttavia, la sua implementazione risulta alquanto inefficiente.

1.3 Problema del logaritmo discreto su curve ellittiche

Data una curva ellittica E definita su un campo F_q, e dato $B \in E$, risolvere il problema del logaritmo discreto su E in base B significa, dato un punto $P \in E$, trovare un intero x tale che $xB = P$ (se x esiste). Si pensa che il problema del logaritmo discreto su curve ellittiche sia più intrattabile del logaritmo discreto in campi finiti. Questo è vero specialmente nel caso in cui la caratteristica del campo sia 2, e vi sono ragioni pratiche, sia di matrice hardware che software, per le quali conviene scegliere F_{2^r}. Vi sono metodi speciali per risolvere il logaritmo discreto in F_2^* grazie ai quali il crittosistema è vulnerabile agli attacchi, a meno che non venga scelto un r molto grande. L'analogo crittosistema basato su curve ellittiche in F_{2^r} è sicuro con valori di r significativamente più piccoli.

Inoltre, a parità di dimensione del campo, gli algoritmi conosciuti per le curve ellittiche sono ancora meno efficienti che nel caso degli interi. Di conseguenza, a parità di sicurezza questa crittografia richiede chiavi pubbliche di dimensione inferiore, e quindi più facilmente utilizzabili rispetto a quelle utilizzate dal metodo RSA (chiavi pubbliche da 256 bit nella crittografia ellittica dovrebbero fornire un livello di sicurezza comparabile ad una chiave pubblica RSA da 3072 bit).

Per questo la crittografia ellittica costituisce attualmente il più valido sistema di crittografia a chiave pubblica alternativo a RSA, il quale un giorno potrebbe diventare non sicuro.

1.4 Algoritmi noti

Vengono descritti ora algoritmi analoghi a quelli noti per lo scambio di chiavi basati sul problema del logaritmo discreto.

1.4.1 Analogo di Diffie-Hellman

L'algoritmo di Diffie-Hellman è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. Si suppone che le due parti, A e B, scelgano un campo finito F_q e una curva ellittica E definita su di esso. Dopodichè, scelgono un punto $B \in E$ di ordine molto grande, e lo rendono pubblico. A sceglie un intero a da mantenere segreto, calcola $aB \in E$ e lo rende pubblico. B fa lo stesso: sceglie b segreto, calcola $bB \in E$ e lo rende pubblico. La chiave segreta è costituita da

$$P = abB \in E$$

Trovare a conoscendo B e aB significherebbe risolvere il problema del logaritmo discreto su curve ellittiche, in quanto sembra che non ci sia modo di calcolare direttamente abB conoscendo solo aB e bB. Questo è noto come problema di Diffie-Hellman.

1.4.2 Analogo di Massey-Omura

L'algoritmo di Massey-Omura è una tecnica crittografica per la trasmissione di un messaggio m su un canale non sicuro. Si suppone m tradotto in un punto $P_m \in E$ su F_q, tali che E sia una curva ellittica pubblica e q grande. Si suppone inoltre che sia stato calcolato e reso pubblico il numero N di punti di E. Ogni utente del sistema sceglie in modo random un intero e che mantiene segreto, tale che $1 < e < N$ e che $GCD(e, N) = 1$; dopodichè calcola

$$d = e^{-1} mod N$$

in modo che $de = 1 \mod N$. Se A vuole mandare P_m a B:

- A manda a B $e_A P_m$

- B manda ad A $e_B e_A P_m$
- A calcola
$$d_A e_B e_A P_m = e_B P_m$$
e lo rimanda a B
- B può leggere P_m calcolando
$$d_B e_B P_m = P_m$$

1.4.3 ECDSA

L' ECDSA (Elliptic Curve Digital Signature Algorithm) è una variante al DSA (Digital Signature Algorithm) che utilizza la crittografia basata su curve ellittiche.

Supponiamo che A voglia mandare un messaggio firmato m a B. Inizialmente, A e B concordano una curva ellittica $E : y^2 + y = x^3 + ax + b$ definita su un campo F_q e un punto $G \in E$ con coordinate (x_G, y_G). Sia inoltre n l'ordine del punto G e sia L_n la lunghezza di $(n)_2$.

A possiede la coppia di chiavi pubblica-privata
$$(d_A, Q_A)$$
dove d_A è un intero random scelto nell'intervallo $[1, n-1]$, $Q_A = d_A G$. A segue i seguenti passi:

- A calcola $e = HASH(m)$ dove HASH è una funzione one-way (e.g. SHA-1), e siano z i primi L_n bit di e.
- A sceglie $k \in [1, n-1]$
- A calcola kG avente coordinate (x_1, y_1) e $r = x_1 \mod q$
- A calcola $s = k^{-1}(z + rd_A) \mod n$.

La firma è la coppia (r, s).

B, per autenticare la firma di A, inizialmente deve validare la sua chiave pubblica Q_A:

1. Controlla che le coordinate di Q_A siano valide e che $Q_A \not\equiv O$
2. Controlla che $Q_A \in E$
3. Controlla che $nQ_A \equiv O$

Dopodichè, B, dopo aver verificato che r e s siano interi $\in [1, n-1]$, verifica che:

$$s^{-1} \cdot z \cdot G \cdot + s^{-1} \cdot r \cdot Q_A = r$$

Nel caso l'uguaglianza sia rispettata, la firma è verificata. Infatti:

$$s^{-1} \cdot z \cdot G \cdot + s^{-1} \cdot r \cdot Q_A = s^{-1}\big(z \cdot G + r \cdot d_A \cdot G\big) = G \cdot s^{-1}\big(z + r \cdot d_A\big)$$

Dimostriamo che

$$G \cdot s^{-1}\big(z + r \cdot d_A\big) = k \cdot G s^{-1}\big(z + r \cdot d_A\big) = ks = k^{-1}\big(z + r \cdot d_A\big)$$

Confronto con DSA

Ad un livello di sicurezza di 80 bit (il che significa che un attacco richiede la generazione di 2^{80} firme per trovare la chiave privata) la dimensione di una chiave pubblica DSA è almeno 1024 bit, mentre la dimensione di una chiave pubblica ECDSA è di 160 bit per via dell'attacco del compleanno. Mentre invece, la dimensione della firma è la stessa: $4t$ bit, dove t è il livello di sicurezza misurato in bit.

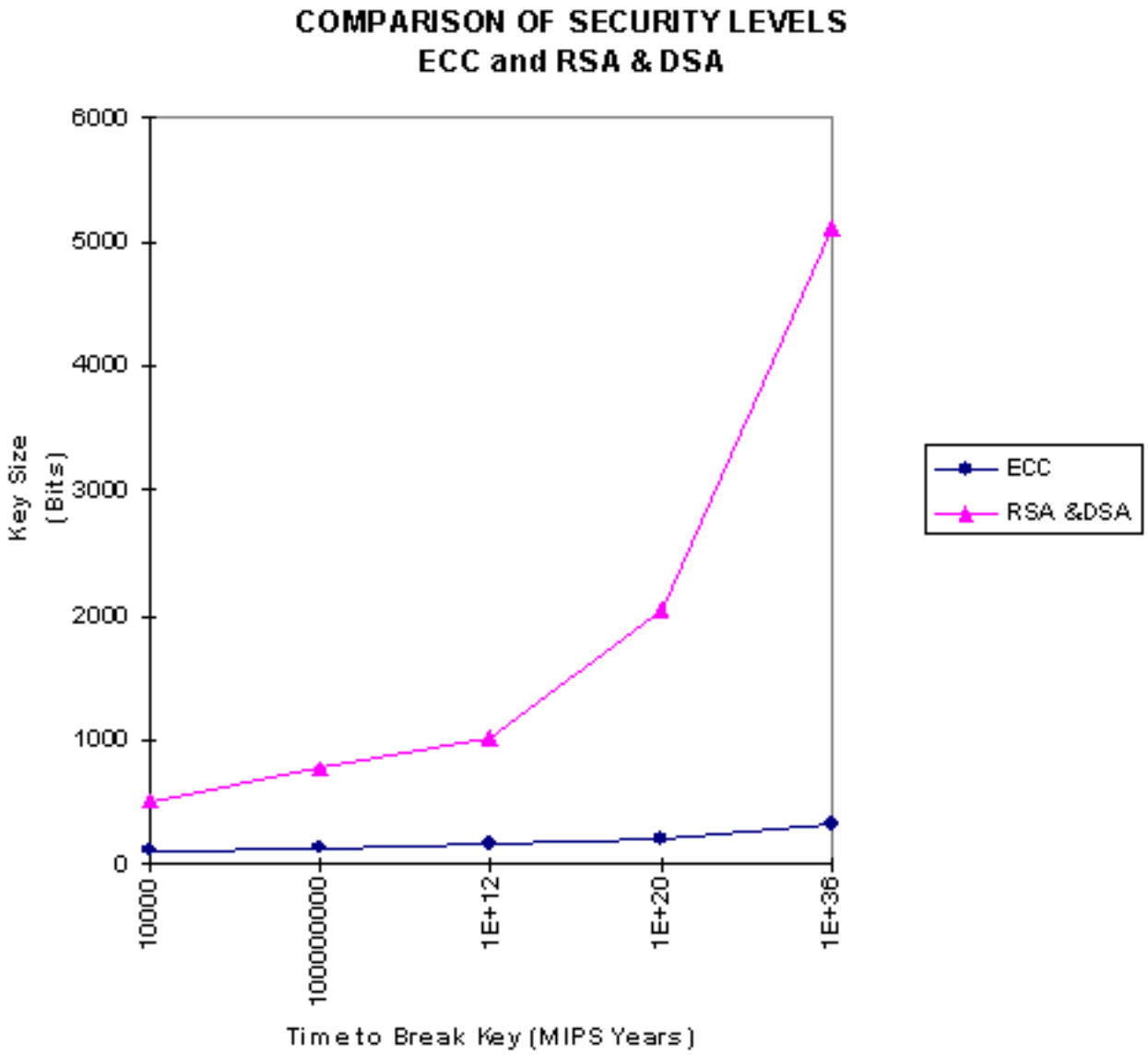

Figura 1.5: Confronto ECDSA e DSA

Capitolo 2

Seconda parte

2.1 La libreria Cryptokit

Cryptokit è una libreria in OCaml scritta da Xavier Leroy, ricercatore presso INRIA (Institut national de recherche en informatique et en automatique) che fornisce una serie di primitive crittografiche utilizzabili per implementare protocolli di crittografia all'interno di applicazioni. Le primitive includono:

- Cifrature a chiave simmetrica:
  - AES
  - DES
  - Triplo DES
  - ARCfour
- Crittografia a chiave pubblica:
  - RSA
  - Algoritmo Diffie-Hellman per lo scambio di chiavi
- Funzioni hash e MAC:
  - SHA-1
  - MD5

  - MAC basato su AES e DES

- Generazione di numeri casuali
- Compressioni:
  - base 64
  - esadecimale
  - compressione Zlib

La peculiarità della libreria è costituita dal fatto che possono essere utilizzati cifrari e funzioni hash addizionali, in quanto i meccanismi crittografici basilari sono forniti da classi generiche facilmente ricomponibili. Le trasformazioni crittografiche dei flussi di dati da crittografare sono quindi modularizzate, in uno stile "Lego-like".

In particolare, Cryptokit contiene:

- Interfaccie general purpose, quali *transform* che rappresenta una corrispondenza arbitraria da sequenze di caratteri a sequenze di caratteri (come cifratura, decifratura, compressione/decompressione, e codifica di dati binari in testo), oppure *hash*, funzione che mappa sequenze di caratteri di lunghezza arbitraria in stringhe più piccole e di lunghezza fissata.
- Utilities per numeri random e schemi padding:
  - il modulo *Random* fornisce generatori di numeri casuali e pseudocasuali per generare chiavi crittografiche, nonces, o challenges.
  - il modulo *Padding* definisce un'interfaccia generica per inglobare i dati in input in un numero fissato di blocchi.
- Primitive crittografiche semplificate:
  - il modulo *Cipher* implementa i cifrari simmetrici AES, DES, Triple-DES, ARCfour e Blowfish, presentati come trasformazioni parametrizzate data una chiave segreta e una direzione, che indica se deve essere eseguita una cifratura o una decifratura.

  - il modulo *Hash* implementa funzioni hash non dotate di chiavi (SHA-1, SHA-256, RIPEMD-160 e MD5), conosciute anche come funzioni digest.
  - il modulo *MAC* implementa funzioni per l'autenticazione di messaggi, che corrispondono a funzioni hash dotate di chiave, alcune delle quali parametrizzate da una chiave segreta. Il modulo MAC fornisce due funzioni basate su SHA-1 e MD5, e 4 funzioni basate sui cifrari a blocchi AES DES e 3DES.
  - il modulo *RSA* implementa la crittografia a chiave pubblica RSA. Questo tipo di crittografia supporta sia la crittografia a chiave pubblica sia la firma digitale.
  - il modulo *DH* implementa l'algoritmo Diffie Helman per lo scambio delle chiavi. Esso è un protocollo tramite il quale due parti possono stabilire un segreto condiviso (tipicamente per la crittogrfia simmetrica o MAC) scambiandosi dei messaggi con la garanzia che anche se un attacker intercettasse la conversazione, non potrebbe ricostruire il segreto condiviso. L'algoritmo DH si basa sulla difficoltà del calcolo del logaritmo discreto ed è vulnerabile agli attacchi di tipo man-in-the-middle.

- Interfacce avanzate e componibili per cifrari a blocchi e cifrari a flusso:
  - Il modulo *Block* fornisce classi che implementano cifrari a blocchi, modalità di concatenazione e inglobamento di un cifrario a blocchi come una trasformazione generale o come una funzione hash. Le classi sono facilmente componibili per facilitare l'integrazione di nuovi cifrari a blocchi.
  - il modulo *Stream* fornisce classi che implementano il cifrario a flusso ARCfour, e l'inglobamento di un cifrario a flusso come trasformazione generale. Anche in questo caso le classi sono facilmente componibili per facilitare l'integrazione di nuovi cifrari a flusso.

- Codifiche e compressioni di dati:

- il modulo *Base64* supporta la codifica e decodifica di dati binari nel formato in base 64, usando solo caratteri alfanumerici che possono essere trasmessi in maniera sicura per email o negli URL.
- il modulo *Hexa* supporta la codifica e decodifica di dati binari come stringhe esadecimali. E' un formato comune per trasmettere chiavi in forma testuale.
- il modulo *Zlib* supporta la compressione e decompressione dei dati usando la libreria zlib. L'algoritmo utilizzato è la compressione Lempel-Ziv come in gzip e zip. Mentre la compressione in se non è crittografia, è spesso usata prima della crittografia per nascondere regolarità nel testo in chiaro e per ridurre le dimensioni del testo cifrato.

- Tipi di errori e utility varie.

2.2 Implementazione di funzioni su curve ellittiche

La libreria è stata integrata con funzioni crittografiche basate su curve ellittiche attraverso l'implementazione di due moduli:

- **EC**: funzioni basilari su curve ellittiche
- **ECDH**: algoritmo per lo scambio delle chiavi secondo lo schema Diffie-Hellman

2.2.1 Modulo EC: funzioni di base su curve ellittiche

Prevede la definizione di due strutture:

- Una curva ellittica è composta da una coppia di interi, a e b, che rappresentano i coefficienti nell'equazione $y^2 = x^3 + ax + b$ definita su un campo finito F_n , con n primo

```
type elliptic_curve = {
        a: int;
        b: int;
}
```

- Un punto appartenente ad una curva ellittica è anch'esso composto da una tripla di interi: le coordinate del punto (intere in quanto le curve a scopi crittografici sono definite su campi finiti) e il campo inf, che assume valori: 0 se è il punto all'infinito della curva, 1 se non lo è, -1 in caso di errore

```
type ec_point = {
        x: int;
        y: int;
        inf: int;
}
```

La funzione *ec_sum* somma due punti dati in input su una curva ellittica utilizzando le formule per la somma di due punti in F_n. Nel caso uno dei due punti sia il punto all'infinito della curva, che rappresenta l'elemento neutro del gruppo abeliano finito composto dalla curva e dall'operazione di somma, viene ritornato l'altro punto. L'output è un punto appartenente alla curva.

```
let ec_sum p1 p2 n ec=
        if p1.inf = 0 then p2 else
        if p2.inf = 0 then p1 else
        let (u1,v1,gcd) = eucl_ext (p1.x - p2.x) n in
        if (gcd<>1 && gcd<>n) then
                {
                x= -1;
                y= -1;
                inf= -1
                }
        else
        if gcd=1 then
```

```
let s = (p1.y - p2.y) * u1 in
let x3 = modulo ((ex s 2)- p1.x -p2.x) n in
let y3 = modulo (s* (p1.x - x3) -p1.y) n in
        {
        x=x3;
        y=y3;
        inf=1
        }
else
let (u1,v1,gcd)=eucl_ext (p1.y + p2.y) n in
if (gcd<>1 && gcd<>n) then
        {
        x= -1;
        y= -1;
        inf= -1
        }
else
if gcd = n then {x =0; y=1; inf = 0} else
let s=(3*(ex p1.x 2) + ec.a)*u1 in
let x3 = modulo ((ex s 2) - 2*p1.x) n in
let y3 = modulo (s*(p1.x-x3) -p1.y) n in
        {
        x=x3;
        y=y3;
        inf=1
        }
```

La funzione ricorsiva *ec_mult* moltiplica un punto su una curva ellittica per un intero utilizzando l'algoritmo efficiente del raddoppiamento ripetuto. L'output è un punto sulla curva.

```
let rec ec_mult k p n ec =
        if p.x = p.y && p.y = p.inf && p.inf = -1 then {x=0;y=1;inf=0}
        else
```

```
match k with
        |0 ->{x=0;y=1;inf=0}
        |1-> p
        |_-> let y=ec_mult (k/2) p n ec in
        if k mod 2 = 0 then ec_sum y y n ec else
        ec_sum p (ec_sum y y n ec) n ec
```

Funzioni ausiliarie

La funzione ricorsiva *ex* implementa l'elevazione a potenza per numeri interi. L'algoritmo utilizzato è efficiente: permette di effettuare, al più, $\log(exp)$ moltiplicazioni.

```
let rec ex b e =
        if e=1 then b
        else
        if e mod 2 = 0 then ex (b*b) (e/2)
                else b*(ex (b*b) (e/2))
```

La funzione *random_integer* genera un intero random di dimensione passata in input, minore di k. Se non viene specificato il seme *rng*, viene utilizzato quello di default letto da /dev/random. Viene richiamata la funzione implementata all'interno del modulo RSA *random_nat*.

```
let random_integer ?(rng = Random.secure_rng) nbits k =
        let a = RSA.random_nat ~rng nbits in
        (abs(Nat.nth_digit_nat a 0)) mod k
```

Le funzioni *eucl* e *eucl_ext* implementano l'algoritmo di Euclide esteso per il calcolo del MCD fra due numeri e, allo stesso tempo, dell'inverso moltiplicativo. In particolare, *eucl_ext x1 x2*, ritorna $x_1u_1 + x_2u_2 = u_3 = MCD(x_1, x_2)$.

```
let rec eucl (u1,u2,u3) (v1,v2,v3) =
        match v3 with
        |0 -> if u3>0 then (u1,u2,u3) else (-u1,-u2,-u3)
```

```
        | _ -> let q=u3/v3 in
        let (t1,t2,t3)=(u1-q*v1,u2-q*v2,u3-q*v3) in
        eucl(v1,v2,v3)(t1,t2,t3)


let eucl_ext x1 x2 =
        let (u1,u2,u3)=(1,0,x1) in
        let (v1,v2,v3)=(0,1,x2) in
        eucl(u1,u2,u3)(v1,v2,v3)
```

La funzione modulo, calcola il modulo tra due numeri.

```
let modulo a n =
        if a>=0 then a mod n
        else a mod n + n
```

Funzioni per il test

- *new_point*: crea un punto di coordinate passate come input

- *new_curve*: crea una curva con parametri a e b passati come input

2.2.2 Modulo ECDH: Algoritmo analogo a Diffie-Helman

Il modulo implementa una variante dell'algoritmo Diffie-Hellman per lo scambio di chiavi su un canale non sicuro. Le funzioni sono speculari a quelle presenti nel modulo DH, il quale ne costituisce la versione classica, basata sul problema del logaritmo discreto.

Viene definita la struttura contenente i parametri su cui le parti coinvolte devono accordarsi per iniziare lo scambio di informazioni.

```
type ec_params = {
        ec: elliptic_curve;
        p: ec_point;
```

```
        q: int;
}
```

La funzione *new_parameters* sceglie una curva ellittica a random su F_q, data la lunghezza dimensione in bit di q, e un punto appartenente ad essa.

```
let new_parameters ?(rng = Random.secure_rng) q numbits =
        let r_a = EC.random_integer ~rng 4 q in
        let r_x = EC.random_integer ~rng numbits q in
        let r_y = EC.random_integer ~rng numbits q in
        let r_b = (EC.modulo (EC.ex r_y 2) q) - (EC.modulo ((EC.ex r_x
        r_x)) q) in
                {
                ec = {EC.a = r_a; EC.b = r_b};
                p = {EC.x = r_x; EC.y = r_y; EC.inf = 1};
                q = q
                }
```

La funzione *private_secret* genera il segreto privato di ogni parte coinvolta.

```
let private_secret ?(rng = Random.secure_rng) para numbits =
        let a = EC.random_integer ~rng numbits para.q in
        a
```

La funzione *message* genera il messaggio da mandare all'altra parte.

```
let message para priv_num =
        let pub_info = EC.ec_mult priv_num para.p para.q para.ec in
        pub_info
```

La funzione *shared_secret* genera il segreto condiviso da entrambe le parti.

```
let shared_secret para priv_num other_msg =
        let key = EC.ec_mult priv_num other_msg para.q para.ec in
        key
```

2.2.3 Test

La libreria mette a disposizione un eseguibile di testing che verifica la correttezza delle funzioni implementate.

Test delle funzioni basilari su curve ellittiche

Vengono creati una curva e un punto su di essa. Il primo test confronta che il raddoppiamento del punto sia corretto, ovvero che sia uguale ad una soluzione nota. Il secondo test, invece, confronta 2 * 2p e 4p; si passa il test se coincidono.

```
let _ =
        testing_function "Elliptic-curves";
                let c = EC.new_curve 1 1 in
                let p = EC.new_point 3 10 in
                let q = 23 in
                let sol = EC.new_point 7 12 in
                let t = EC.ec_mult 2 p q c in
                let r = EC.ec_mult 4 p q c in
                let s = EC.ec_mult 2 t q c in
                test 1 t sol;
                test 2 r s;;
```

Test del modulo ECDH

Vi sono due parti coinvolte: entrambe generano un intero segreto e un messaggio da inviare alla rispettiva controparte. Si passa il test se le chiavi finali generate da ambo le parti sono uguali.

```
let _ =
        testing_function "Elliptic curves Diffie-Hellman";
                let par = ECDH.new_parameters 23 1 in
                let a = ECDH.private_secret par 3 in
                let aP = ECDH.message par a in
                let b = ECDH.private_secret par 3 in
```

```
let bP = ECDH.message par b in
let key_a = ECDH.shared_secret par a bP in
let key_b = ECDH.shared_secret par b aP in
test 1 key_a key_b;;
```

Appendice A

Utilizzi della crittografia ellittica

La crittografia basata su curve ellittiche è stata introdotta nel 1985. Ad oggi, dopo vari anni di studi, è considerata una tecnologia sicura ed efficiente, ed è stata incorporata in molti prodotti e adottata in vari sistemi crittografici.

Campi e curve raccomandati dal NIST

Il FIPS (Federal Information Processing Standards) è una collezione di standard e linee guida rilasciate dal NIST (National Institute of Standards and Technology) per uso governativo. FIPS 186-2 include ECDSA, con raccomandazioni su specifici campi finiti e curve ellittiche dotate di proprità particolari per ottimizzare le performance. E' accertato che nessuna di queste appartene alla classe delle curve anomale e supersingolari, sucscettibili agli attacchi. Viene consigliato un totale di 15 curve ellittiche in 10 campi finiti (5 campi finiti su un primo p e 5 campi finiti su 2^k). Per ognuno di essi è generata una curva pseudo-casualmente utilizzando il metodo SHA-1, come specificato dagli standard. Inoltre, per ognuno dei campi binari è selezionata una curva di Koblitz, le quali ammettono una moltiplicazione scalare rapida dei punti su di esse.

Tutte le curve scelte hanno come cofattori 1, 2 o 4 per assicurare l'efficienza nella computazione. Di conseguenza, le chiavi pubbliche e private hanno approssimativamente la stessa lunghezza in bit. Ogni campo è scelto in modo che la lunghezza del suo ordine in bit sia almeno due volte la lunghezza dei cifrari a blocchi a chiave privata. Questo perchè una ricerca esaustiva di un cifrario a blocchi

di k bit richiede lo stesso tempo della soluzione di ECDLP usando l'algoritmo di Pollard in un campo finito di ordine $2k$.

Smart Cards

Un esempio di utilizzo pratico della crittografia ellittica è costituito dalle smart card, dispositivi hardware delle dimensioni di una carta di credito con potenzialità di elaborazione e memorizzazione dati ad alta sicurezza. Ad esempio, sono utilizzate per le schede telefoniche, per i pagamenti elettronici, per l'identificazione, e per molte altre applicazioni. In commercio se ne trovano di vari tipi:

- Memory cards: sono le meno costose e meno funzionali. Contengono una memoria EEPROM e una ROM, e spesso si limitano a prevenire la sovrascrittura o la cancellazione dei dati salvati. Un tipico esempio è costituito dalle carte telefoniche prepagate.

- Micorprocessor cards: l'architettura di questo tipo di smart card include una CPU, una RAM, una ROM ed una EEPROM. Tipicamente, il sistema operativo è contenuto della ROM.

- Coprocessor cards: sono come le carte dotate di microprocessore, con l'aggiunta di un coprocessore atto a svolgere le operazioni crittografiche per ridurre notevolmente i tempi di calcolo.

- Contactless smart cards: anch'esse contengono un microprocessore. Si distinguono in quanto i contatti del chip non sono visibili sulla superficie della carta.

L'algoritmo ECDLP (Elliptic Curve Discrete Logarithm Problem) garantisce una buona sicurezza con chiavi di dimensioni relativamente ridotte. Dal momento che le chiavi sono più corte, la memoria richiesta per salvarle è minore e, conseguentemente, i tempi di trasmissione dei dati tra la carta e il dispositivo sono ridotti. Inoltre, molte smart card che normalmente richiederebbero un coprocessore dedicato per i calcoli crittografici, non lo richiedono nel caso venga utilizzata la crittografia ellittica. Un altro aspetto di rilevanza pratica riguarda

la generazione delle chiavi pubbliche: tipicamente sono generate esternamente e, in seguito, trasferite sulla smart card, in quanto è un'operazione costosa. Nel caso di ECC, invece, il tempo necessario a generare una chiave è ridotto, e può essere compiuto direttamente nella carta se viene fornito un generatore di numeri casuali efficiente.

Open SSL

Open SSL è un'implementazione open source di SSL e TLS, protocolli crittografici che permettono una comunicazione sicura end-to-end su reti TCP/IP fornendo autenticazione, integrità dei dati e cifratura operando al di sopra del livello di trasporto. Le librerie di base, in linguaggio C, eseguono le funzioni crittografiche principali, e sono disponibili per la maggior parte dei sistemi operativi unix-like oltre che per Microsoft Windows.

Open SSL contiene le primitive per la crittografia ellittica. In particolare nel 2002 l'azienda SUN Microsystems contribuì all'implementazione aggiungendo:

- funzionalità basate sull'attuale internet-draft di IETF, che spiecifica l'utilizzo della crittografia ellittica all'interno del protocollo SSL

- implementazione dell'algoritmo di Diffie-Helman basato sullo standard ANSI X9.63

- supporto su cambi binari polinomiali e la sottostante libreria aritmetica

Bibliografia

[1] Neal Koblitz. A Course in Number Theory and Cryptography. Springer-Verlag, 1987

[2] Johannes A. Buchmann. Introduction to Cryptography. Springer-Verlag, 2^{nd} edition, 2004

[3] Elisabeth Oswald. Introduction to Elliptic Curve Cryptography. Institute for Applied Information Processing and Communication

[4] Wade Trappe - Lawrence C. Washington. Crittografia con Elementi di Teoria dei Codici. Pearson Paravia Bruno Mondadori, Seconda edizione, 2009

[5] Matthew England. Elliptic curve cryptography. Heriot-Watt University, 2006

[6] Lawrence C. Washington. Elliptic Curves: Number Theory and Cryptography. Chapman Hall, 2^{nd} edition, 2008.

[7] Sorgente di Cryptokit. http://forge.ocamlcore.org/projects/cryptokit/

[8] Sorgente di OpenSSL. ftp://ftp.openssl.org/snapshot/

Ringraziamenti

Ringrazio i miei genitori e i miei colleghi. In particolare, Luca Toscano, Simona Arma, Luca Serravalli, Isabella Taronna, Andrea Catalini, Alessandro Veronesi, Daniele Bellini, Andrea 'Patatino' Melis, Alberto Paladino, Dennis Olivetti e Simone Rondelli.