Analisi della crittografia su curve ellittiche

Docente: Francesco Romani
Studente: Nicola Venturi, 546468

UNIVERSITÀ DI PISA

Indice

1 Introduzione

La crittografia su curve ellittiche, o ECC, è un sistema di crittografia a chiave pubblica utilizzato per lo scambio di chiavi segrete e messaggi cifrati attraverso l'uso di opportuni protocolli.

2 Le curve ellittiche

2.1 Le curve ellittiche su numeri reali

Le curve ellittiche sui numeri reali sono curve algebriche definite sul campo $\mathbb{R}$. Per essere definite, tali curve devono soddisfare l'equazione in forma normale di Weistrass e comprendere un elemento neutro $\mathcal{O}$, detto anche punto all'infinito su y

$$E(a,b) = \{(x,y) \in \mathbb{R}^2 | y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$$

dove a,b $\in \mathbb{R}$.

Inoltre, se la curva ellittica E(a,b) soddisfa l'equazione $(4a+27b) \neq 0$ assume le caratteristiche di un **gruppo abeliano**, la cui definizione viene tralasciata in quanto non fondamentale per questo progetto.
Le curve ellittiche sui numeri reali possono assumere una delle seguenti forme a seconda del numero di radici reali che questa presenta e in entrambi i casi presentano una simmetria orizzontale.

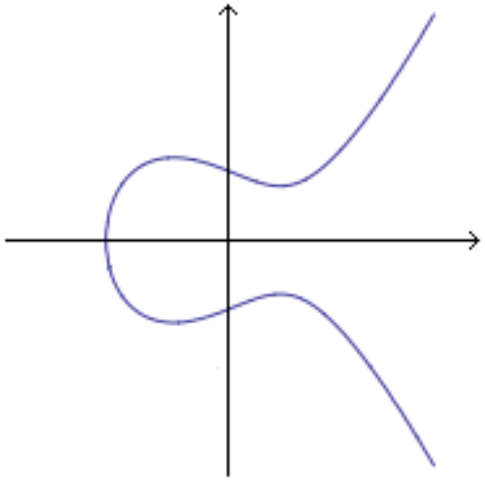

Figura 1: Curva con un'unica radice reale

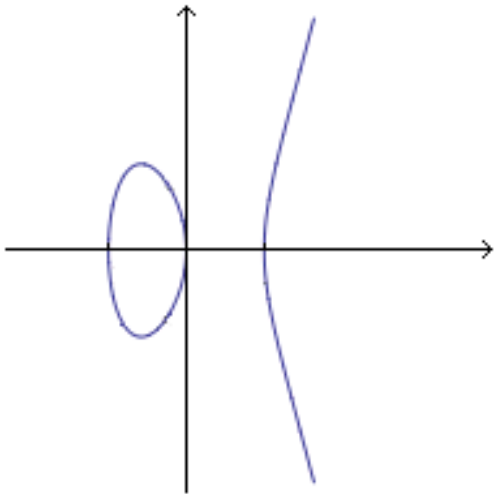

Figura 2: Curva con tre radici reali

Da un punto di vista crittografico è rilevante l'operazione di somma di due punti appartenenti ad una curva. A questo scopo sottolineiamo che ogni retta interseca la curva in al più 3 punti. Se una retta interseca la curva E(a,b) nei punti P e Q allora questa lo intersecherà anche nel punto R, nel caso in cui la retta sia verticale allora il terzo punto sarà il punto all'infinito $\mathcal{O}$.
La somma è quindi così definita: viene considerata la retta t passante per i punti P e Q, o la tangente passante per P nel caso in cui P e Q coincidano; si determina il punto di intersezione R della retta t con la curva, o tra la tangente e la curva; infine si definisce P+Q come il punto simmetrico a R rispetto all'asse delle ascisse, ovvero P+Q=-R.

La formula della somma ci permette di calcolare le coordinate del punto T=P+Q a partire dalle coordinate di P e Q.
Poniamo P=(x_p, y_p) e Q=(x_q, y_q) due punti distinti della curva e tali che P≠-Q, allora abbiamo

$$\begin{cases} x_t = \lambda^2 - x_p - x_q \\ y_t = y_p + \lambda(x_p - x_t) \end{cases}$$

con $\lambda = \frac{y_q - y_p}{x_q - x_p}$ coefficiente angolare della retta passante per P e Q, nel caso in cui P e Q siano distinti, e $\lambda = \frac{3{x_p}^2 + a}{2y_p}$ nel caso in cui P=Q. Se y_p=0 allora la tangente in P è una retta verticale e questo implica 2P=$\mathcal{O}$.

2.2 Curve ellittiche su campi finiti

Le curve ellittiche su campi finiti sono le curve usate in pratica per realizzare la crittografia su curve ellittiche, poiché offrono un'aritmetica precisa e veloce da calcolare, cosa che non avviene con con le curve definite sui numeri reali.
Sono definite due famiglie di curve ellittiche su campi finiti: la prima sono le ***curve ellittiche prime***, utilizzate per implementazioni software di questa tipologia di crittografia, la seconda sono le ***curve ellittiche binarie***, utilizzate invece per implementazioni a livello hardware.

2.2.1 Curve ellittiche prime

Le curve ellittiche prime sono curve con variabili e coefficienti ristrette ad elementi di $\mathbb{Z}_p$, ovvero tutti gli elementi compresi tra 0 e p-1. La caratteristica del campo è p, per questo motivo ci limiteremo a considerare campi $\mathbb{Z}_p$ con p>3 e le curve saranno descritte dall'equazione

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

con a,b$\in \mathbb{Z}_p$.

Una curva ellittica prima è rappresentata da una nuvola di punti, che vengono riportati nel sottospazio $0 \leq x \leq p-1$, $0 \leq y \leq p-1$. I valori di y nelle curve ellittiche definiti su numeri reali, come affermato precedentemente, sono simmetrici rispetto all'asse delle x, ma questo non può avvenire nelle curve ellittiche su campi finiti, infatti possiamo notare che $-y mod p = p-1$, quindi tutti i punti sono compresi in $0 \leq y \leq p-1$ e abbiamo una simmetria rispetto alla retta $y = p/2$.

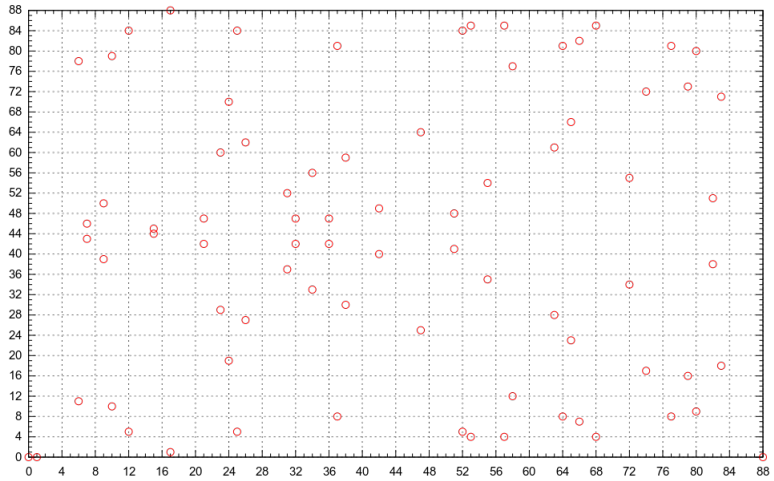

Figura 3: $y^2 = x^3 - x$ in $\mathbb{Z}_{89}$

Per verificare che il polinomio $(x^3 + ax + b) mod p$ non abbia radici multiple, dobbiamo verificare l'equazione $(4a^3 + 27b) mod p \neq 0$, affermando in questo modo che i punti della curva $E_p(a,b)$ formano un gruppo abeliano finito rispetto all'operazione di addizione.

Un parametro importante dal punto di vista della sicurezza delle applicazioni crittografiche è rappresentato dall'ordine di una curva, ossia il numero di punti che la compongono. Si può dimostrare che esattamente (p-1)/2 elementi di $\mathbb{Z}_{\text{l}}$ sono *residui quadratici*.

Definizione: In ***teoria dei numeri***, un numero intero q si dice ***residuo quadratico*** modulo p se esiste esiste un intero x t.c. $x^2 \equiv q$ mod p.

Viene ora in nostro aiuto il seguente teorema che permette di porre un limite superiore all'ordine di una curva:

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

2.2.2 Curve ellittiche binarie

Le curve ellittiche binarie sono la seconda famiglia di curve utilizzata dalla ECC. I coefficienti e le variabili di tale curve assumono valore nel campo $\mathbb{GF}(2^m)$, costituito da 2^m elementi che si possono pensare come tutti gli interi binari di m cifre e su cui si può operare mediante l'aritmetica polinomiale modulare.

Dato che la caratteristica del campo $\mathbb{GF}(2^m)$ è 2, le curve ellitticche binarie saranno definite nel modo seguente:

$$E_{2^m} = \{(x,y) \in \mathbb{GF}(2^m)^2 | y^2 + xy = x^3 + ax + b\} \cup \{\mathcal{O}\}$$

con a,b$\in \mathbb{GF}(2^m)$.

Come per le curve ellittiche prime, anche l'ordine delle curve binarie può essere stimato utilizzando il teorema di Hasse, che fornisce la seguente stima

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

3 Perché usare ECC?

3.1 Il problema del logaritmo discreto

La sicurezza della crittografia su curve ellittiche si basa sul problema del logaritmo discreto su curve ellittiche, funzione one-way trap-door[1] equivalente al logaritmo discreto per l'algebra modulare di RSA.

Questo problema consiste nel trovare un valore **k**, dati due punti **Q** e **R**, tale che valga l'eguaglianza **Q**=**kR** e ciò risulta computazionalmente difficile in quanto non è ancora stato trovato un algoritmo che risolva questo problema in tempo polinomiale.

3.2 Vantaggi dell'ECC

La crittografia su curve ellittiche, a differenza del cifrario RSA e del protocollo DH, non è soggetta ad attacchi basati sull'algoritmo ***index-calculus***, tipologia di attacchi che per chiavi composte da **b** bit risolve il problema del logaritmo discreto per l'algebra modulare in tempo $O(2^{\sqrt{blogb}})$, che risulta sub-esponenziale. Attualmente, il miglior attacco noto al problema del logaritmo discreto per le curve ellittiche è ***Pollard*** ρ che richiede in media $O(2^{b/2})$ operazioni per chiavi da **b** bit e risulta pienamente esponenziale.

A parità di lunghezza delle chiavi, i protocolli basati su ECC sono molto più sicuri rispetto ad RSA e protocollo DH, considerati questi equivalenti da un punto di vista di sicurezza poiché entrambi sono soggetti agli attacchi descritti precedentemente. Questo implica che per garantire lo stesso livello di sicurezza la crittografia su curve ellittiche richiede chiavi di lunghezza minore.
Nella tabella seguente vengono mostrate a parità di livello di sicurezza le lunghezze in bit delle chiavi dei tre protocolli.

[1] funzione facilmente calcolabile dati determinati parametri, ma difficile da invertire

RSA e DH	ECC
1024	160
2048	224
3072	256
7680	384
15360	512

Tabella 1: Lunghezze in bit della chiave a parità di sicurezza

Altro aspetto importante di cui tener conto è l'efficienza computazionale di un sistema crittografico.
RSA e ECC, tenendo conto di chiavi della stessa lunghezza, sono confrontabili, ma considerando che per offrire lo stesso livello di sicurezza ECC può usufruire di chiavi più corte e questo mostra altri vantaggi a favore di questo sistema crittografico in termini di efficienza: elaborazioni più veloci ed implementazioni più compatte.

4 Scopo e test del progetto

4.1 Scopo

Il progetto si propone di usare il Python per condurre uno studio accademico sull'implementazione di una classe ausiliaria Point, dell'algoritmo di Koblitz, per la trasforma-zione di un messaggio in un punto della curva, e il protocollo per lo scambio di messaggi cifrati testandoli poi sue due curve proposte dal NIST. La prima è la curva denominata secp192-r1 che opera in modulo su un numero primo a 192 bit, la seconda è la curva denominata secp256-r1 che invece opera su un primo a 256 bit.

4.1.1 Algoritmo di Koblitz

L'algoritmo di Koblitz è uno degli algoritmi con probabilità più bassa di fallimento nel trasformare un messaggio $m < p$in un punto della curva ellittica prima $E_p(a, b)$.
Usando m come valore per l'ascissa del punto, la probabilità di trovare un punto della curva è pari alla probabilità che $m^3 + am + b$ mod p sia un residuo quadratico, ossia pari a 0,5.
Per questo motivo si fissa un intero h t.c. $(m+1)h < p$ e si considerano come possibili valori dell'ascissa del punto della curva gli interi x=mh+i con $0 \leq i \leq h-1$.
Per ogni x si prova a calcolare $y = \sqrt{(x^3 + ax + b) mod p}$, se la radice esiste si prende P_m(x,y) come punto sulla curva corrispondente al messaggio m, altrimenti si procede con un nuovo valore di x incrementando il valore di i.
L'algoritmo termina trovando una radice, o raggiungendo il limite superiore di i e concludendo che non è stato trovato un punto della curva in cui trasformare il messaggio. Per risalire al messaggio m a partire dal punto P_m si calcola m=$\lfloor x/h \rfloor$.

Poiché la probabilità di trovare una radice è pari a 1/2 questo significa che la probabilità di successo dell'algoritmo è pari a 1-$\frac{1}{2^h}$. Nel caso la dimensione del messaggio m sia troppo grande da poter trasformare in un punto della curva, il

messaggio può essere diviso in blocchi opportunamente lunghi senza alterarne il significato.

4.1.2 Protocollo per lo scambio di messaggi cifrati

Gli utenti per scambiarsi messaggi cifrati devono accordarsi su una curva ellittica prima $E_p(a, b)$, su un punto generatore B, di ordine n elevato, appartenente alla curva e su un intero h per la trasformazione dei messaggi in punti della curva. Successivamente ogni utente genera la propria coppia $<K_{prv}, K_{pub}>$ scegliendo un intero casuale n_u<n come chiave privata e pubblicando $P_U = n_U$B come chiave pubblica. Descriviamo ora la fase di cifratura e di decifrazione:

- **Cifratura**: Mitt trasforma il messaggio m nel punto P_m della curva, sceglie un intero casuale r e calcola i due punti V=rB, W=P_m+rP_D dove P_D è la chiave pubblica di Dest e invia la coppia di punti <V,W>.
- **Decifrazione**: Dest riceve la coppia di punti <V,W> e ricostruisce il punto P_m con la sua chiave privata n_D, calcolando W-n_DV=P_m+rP_D-n_DV=P_m+r(n_DB)-n_D(rB)=P_m, così può recuperare il messaggio m dal punto trovato.

4.2 Test

I test sulle due curve definite precedentemente sono stati svolti in modo incrementale, partendo da piccole stringhe costituite da pochi caratteri, passando poi a stringhe formate da più di un centinaio di caratteri per arrivare ad immagini di alcuni kB. Il test finale è stato condotto su sulla seguente immagine che ha una dimensione di 7,0 kB.

Figura 4: Immagine di test finale

L'immagine per essere cifrata è stata letta come una stringa e decodificata in ASCII. Successivamente, ognuno dei 9364 caratteri di cui era stata composta è stata trasformata in un punto della curva, partendo da un valore del paramentro h per l'algoritmo di Koblitz pari ad 1, fino ad arrivare in modo incrementale al

valore di h che permettesse di generare i punti corrispondenti ai caratteri della stringa mostrati nella figura successiva.

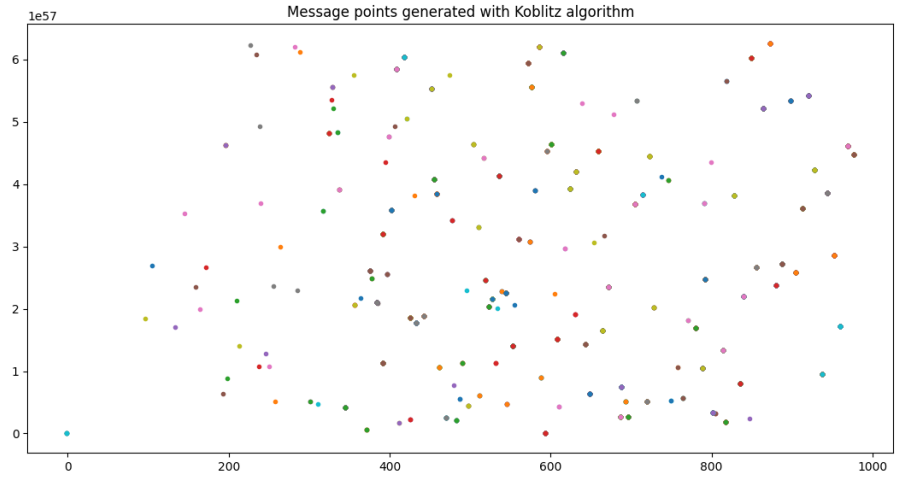

Figura 5: Punti generati sulla curva secp192-r1

I caratteri uguali vengono trasformati in punti uguali, ma verranno successivamente cifrati in punti diversi della curva grazie all'intero r generato in modo casuale in un intervallo [2, n-1], con n ordine del punto generatore della curva come spiegato precedentemente. Le coppie di punti generati dalla cifratura del messaggio sono mostrati nella seguente figura.

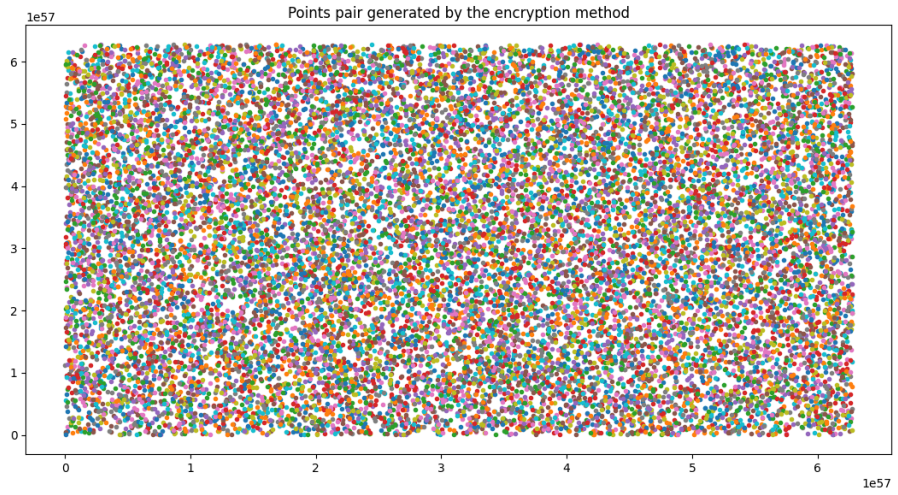

Figura 6: Coppie di punti generati dalla cifratura del messaggio

La decifrazione del messaggio genererà nuovamente i punti mostrati in Figura 5, dai quali verranno poi recuperati i caratteri per ricostruire l'immagine decifrata che corrisponderà alla Figura 4 mostrata precedentemente.

5 Appendice

All'interno della seguente appendice sarà presentato il codice delle classi utilizzate per condurre i test descritti precedentemente.

5.1 Point

La classe definisce un oggetto di tipo Point che rappresenta un punto della curva ellittica

```
class Point(object):

    def __init__(self, x=0, y=0):
        self.X=int(x)
        self.Y=int(y)

    def getCoordinate(self):
        return (self.X, self.Y)

    def setX(self, value):
        self.X=int(value)

    def getX(self):
        return self.X

    def setY(self, value):
        self.Y=int(value)

    def getY(self):
        return self.Y

    def equals(self, P):
        if self.X==P.X & self.Y==P.Y:
            return True
        else:
            return False
```

5.2 KoblizAlgorithm

La classe definisce un oggetto di tipo KoblizAlgorithm per la trasformazione di un messaggio in un punto della curva

```
import math
from Point import Point
from Exception.NonValidHException import NonValidHException

class KoblitzAlgorithm(object):

    def __init__(self, a, b, p):
        self.p=p
        self.a=a
        self.b=b

    def trasform_message(self, m, h):
        if (m+1)*h>=self.p:
            raise NonValidHException
        else:
            Pm=Point(-1,-1)
            for i in range(h):
                x=(m*h+i)%self.p
```

```
                z=(pow(int(x),3,self.p)+int(x)*self.a+self.b)%self.
    p
                if self.__quadratic_residue(z)==1:
                    y=self.__tonelli(z, self.p)
                    Pm.setX(x)
                    Pm.setY(y)
                    break
            return Pm

    def __quadratic_residue(self, z):
        return pow(z, (self.p-1)//2, self.p)
```

Al suo interno, la classe utilizza anche l'algoritmo di Tonelli-Shanks per il calcolo della radice quadrata in modulo. Tale algoritmo calcola la radice quadrata mod p in tempo polinomiale se l'intero p è un numero primo, caso in cui rientrano i test condotti.

```
def __tonelli(self, n, p):
        q = p - 1
        s = 0
        while q % 2 == 0:
            q //= 2
            s += 1
        if s == 1:
            return pow(n, (p + 1) // 4, p)
        for z in range(2, p):
            if p - 1 == self.__quadratic_residue(z, p):
                break
        c = pow(z, q, p)
        r = pow(n, (q + 1) // 2, p)
        t = pow(n, q, p)
        m = s
        t2 = 0
        while (t - 1) % p != 0:
            t2 = (t * t) % p
            for i in range(1, m):
                if (t2 - 1) % p == 0:
                    break
                t2 = (t2 * t2) % p
            b = pow(c, 1 << (m - i - 1), p)
            r = (r * b) % p
            c = (b * b) % p
            t = (t * c) % p
            m = i
        return r
```

5.3 ECC

La classe definisce un oggetto di tipo ECC che rappresenta una curva ellittica oltre ai relativi metodi per la cifratura e la decifrazione di un messaggio

```
import math
import random
from Point import Point
from Exception.PointToInfiteException import PointToInfiteException
from Exception.InvalidParameterException import
    InvalidParameterException

"""Define the main method to exchange message on Elliptic Curve"""
class ECC(object):

```

```
    """a, b: value of elliptic curv Ep(a,b), p: curve order, B:
    curve point choosen"""
    def __init__(self, a, b, p, B,n,h, seed):
        self.a=a
        self.b=b
        self.p=p
        self.B=B
        self.n=n
        self.h=h
        self.seed=seed
        random.seed(self.seed)

    """Method to generate the inverse -A of a point A"""
    def __gen_Inverse(self, A):
        W=Point(A.getX(), self.p-A.getY())
        return W

    """Method used to calculate the gradient of passing line for A
    and B"""
    def __calc_lmb(self, A, B):
        xa=A.X
        ya=A.Y
        if A==B:
            if ya==0:
                raise PointToInfiteException
            else:
                yaopp=pow(2*ya, self.p-2,self.p)
                lmb=((3*xa**2+self.a)*yaopp)%self.p
                return lmb
        else:
            xb=B.X
            yb=B.Y
            if (xb-xa)==0:
                raise PointToInfiteException
            else:
                diffOpp=pow(int(xb-xa), self.p-2, self.p)
                lmb=((yb-ya)*diffOpp)%self.p
                return lmb

    """Method used to calculate the sum of two points"""
    def __sum_points(self, A, B):
        gradient=self.__calc_lmb(A,B)

        xa=A.X
        ya=A.Y
        if A==B:
            xc=(gradient**2-2*xa)%self.p
            yc=(-ya+gradient*(xa-xc))%self.p
            C=Point(xc, yc)
            return C
        else:
            xb=B.X
            yb=B.Y
            xc=(gradient**2-xa-xb)%self.p
            yc=(-ya+gradient*(xa-xc))%self.p
            C=Point(xc,yc)
            return C

    """Method which execute the redoubling method to calculate the
    product of an integer k per a point A"""
    def __redoubling_method(self, A, k):
        if k==1 :
```

```
            return A
        kbin=bin(k)
        kbin=kbin[2:]
        D=A
        Q=Point()
        len=kbin.__len__()
        for i in kbin[:len-1]:
            D=self.__sum_points(D,D)
            if i=="1":
                if Q.X==0 & Q.Y==0:
                    Q.setX(D.X)
                    Q.setY(D.Y)
                else:
                    Q=self.__sum_points(Q,D)

        return Q

    """Method to encypt Pm and generate the pair <V,W> to send"""
    def encrypt(self, r, Pdest, Pm):
        V=self.__redoubling_method(self.B,r)
        Q=self.__redoubling_method(Pdest, r)
        W=self.__sum_points(Pm,Q)
        return (V,W)

    """Method to decrypt and get Pm from pair <V,W>"""
    def decrypt(self, pair, kprv):
        V=pair[0]
        W=pair[1]
        Vf=self.__redoubling_method(V,kprv)
        Vff=self.__gen_Inverse(Vf)
        Pm=self.__sum_points(W,Vff)
        return Pm

    """Method to generate a public key from a private key given"""
    def __kpub_generator(self, kprv):
        return self.__redoubling_method(self.B, kprv)

    """Method to generate keys pair"""
    def keys_generator(self):
        N=len(bin(self.n))
        if (N<160 & N>223):
            raise InvalidParameterException
        c=random.randint(2, self.n-1)
        d=(c%(self.n-1))+1
        Q=self.__kpub_generator(d)
        return (d,Q)
```

6 Bibliografia

Riferimenti bibliografici

[1] Anna Bernasconi, Paolo Ferragina, Fabrizio Luccio, *Elementi di Crittografia*, Pisa University Press, 2014

[2] *Digital Signature Standard (DSS)*, NIST, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`

[3] *Elliptic Curve Cryptography*, `https://en.wikipedia.org/wiki/Elliptic-curve_cryptography`

[4] *Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters*, Daniel R. L. Brown, Certicom Research, January 2010

[5] *Text Message Encoding Based on Elliptic Curve Cryptography and a Mapping Methodology*, Omar Reyad, Computer Science Branch, Faculty of Science, Sohag University, Egypt

[6] *Algoritmo di Tonelli-Shanks*, `https://rosettacode.org/wiki/Tonelli-Shanks_algorithm#Python`