Criptografía de curva elíptica

¿Cómo funciona Bitcoin?

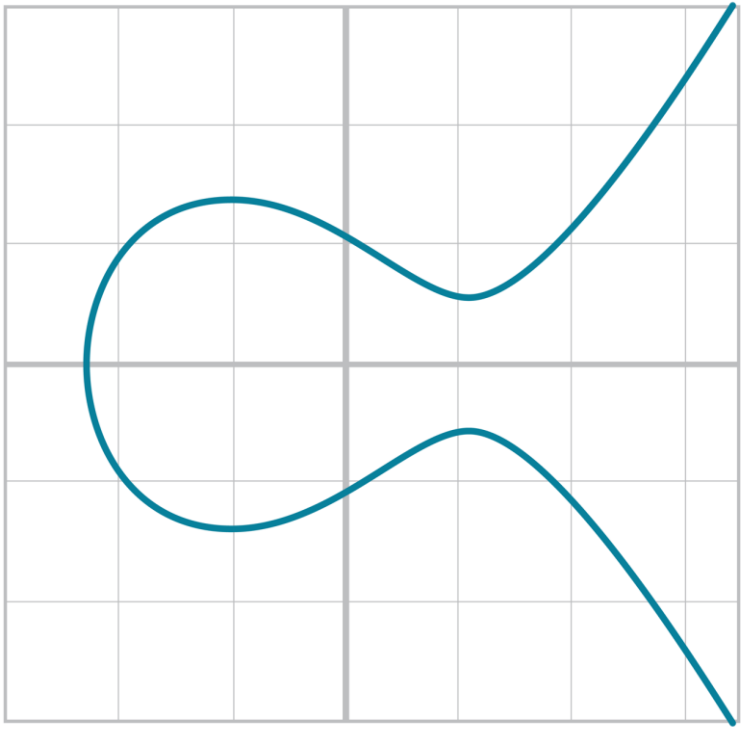
Curvas Elipticas

Definición de una curva elíptica:

$$\{ (x,y) : y^2 = x^3 + ax + b \} + \{ I \}$$

Curvas Elipticas

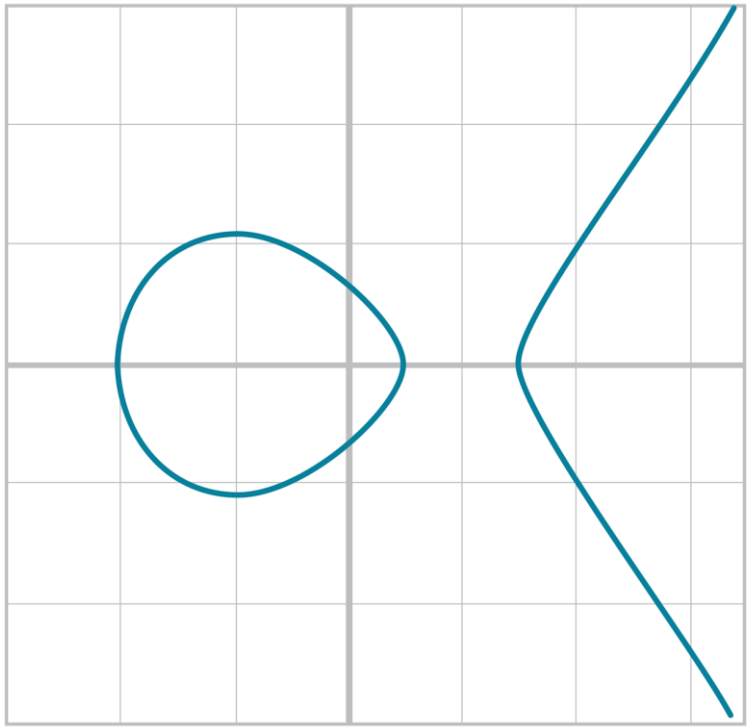
Ejemplos



https://github.com/jimmysong/programmingbitcoin/blob/master/ch02.asciidoc

Curvas Elipticas

Ejemplos

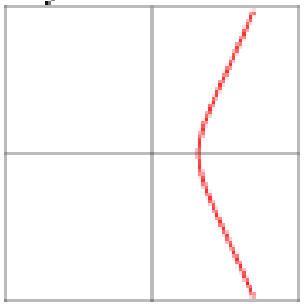


https://github.com/jimmysong/programmingbitcoin/blob/master/ch02.asciidoc

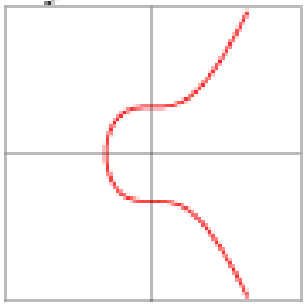
Curvas Elipticas

Ejemplos

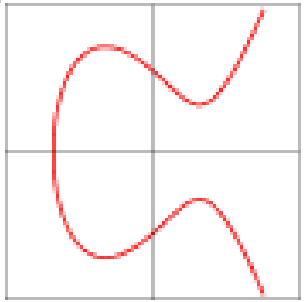
$y^2 = x^3 - 1$



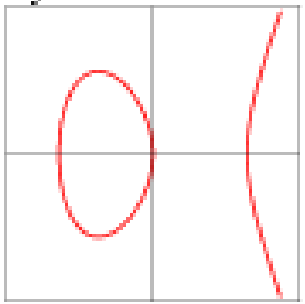
$y^2 = x^3 + 1$



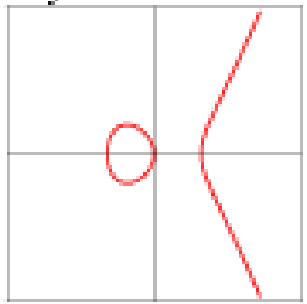
$y^2 = x^3 - 3x + 3$



$y^2 = x^3 - 4x$



$y^2 = x^3 - x$

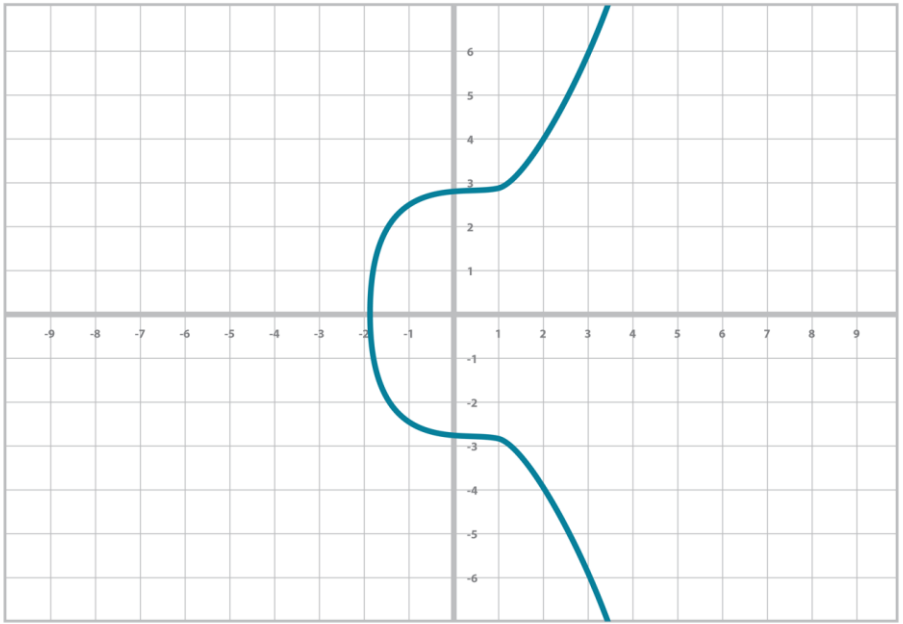


https://mathworld.wolfram.com/EllipticCurve.html

Curva de Bitcoin

secp256k1

$$y^2 = x^3 + 7$$



https://github.com/jimmysong/programmingbitcoin/blob/master/ch02.asciidoc

Curvas Elipticas

Lo que nos interesa son puntos de la curva

Curvas Elipticas

```python
class Point:

    def __init__(self, x, y, a, b):
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        # x being None and y being None represents the point at infinity
        # Check for that here since the equation below won't make sense
        # with None values for both.
        if self.x is None and self.y is None:
            return
        # make sure that the elliptic curve equation is satisfied
        # y**2 == x**3 + a*x + b
        if self.y**2 != self.x**3 + a * x + b:
            # if not, throw a ValueError
            raise ValueError('({}, {}) is not on the curve'.format(x, y))

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y \
            and self.a == other.a and self.b == other.b
```

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

Operaciones con puntos

Suma de dos puntos en curva elíptica

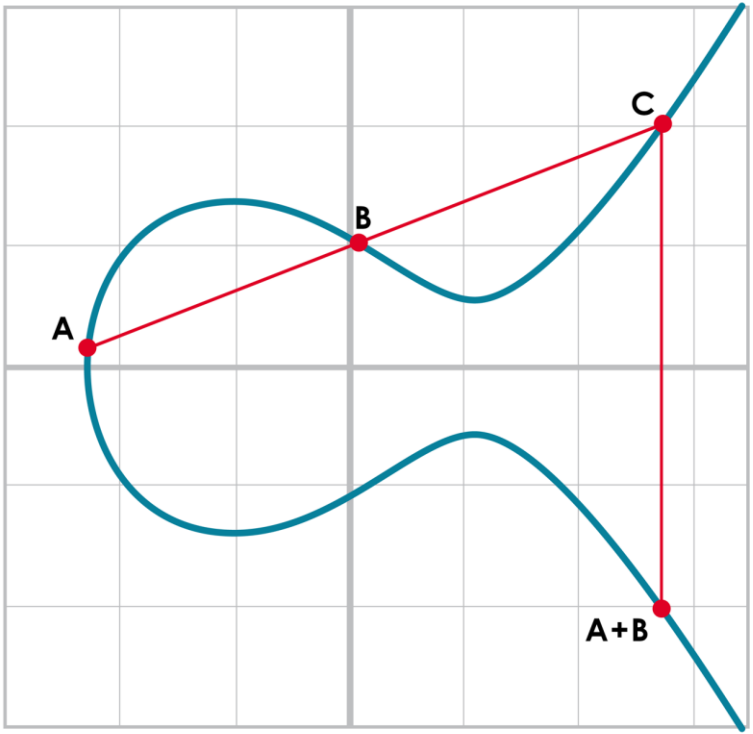
La operación más importante:

- Suma de dos puntos
- Veremos que esto nos permite tener un grupo

Suma de dos puntos

Intuición:

1. Dos puntos definen una recta
2. Recta intersecta la curva en tercer punto
3. Imagen en espejo de axis y de la intersección es nuestra suma

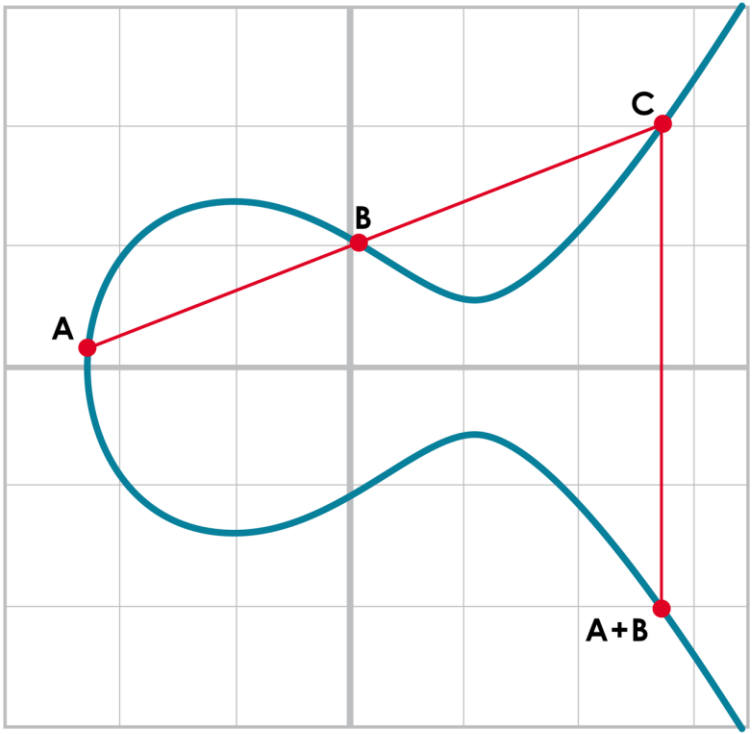


Suma de dos puntos

Intuición:

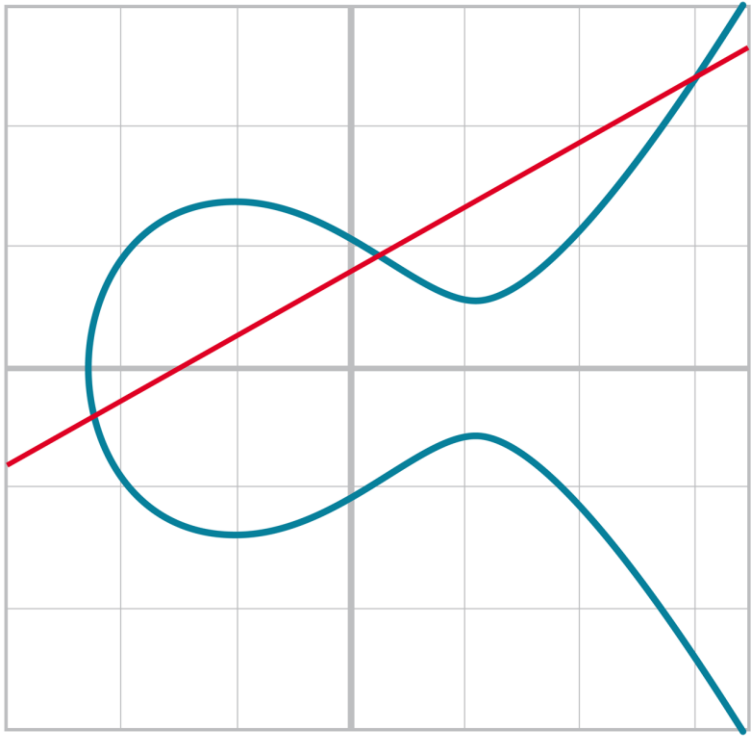
1. Dos puntos definen una recta
2. Recta intersecta la curva en tercer punto
3. Imagen en espejo de axis y de la intersección es nuestra suma

Pero esto no ocurre siempre!



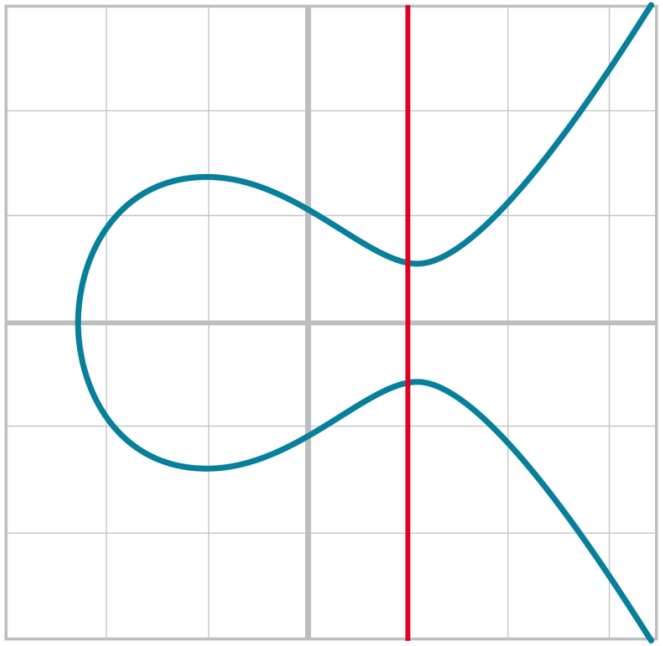
La curva y la recta

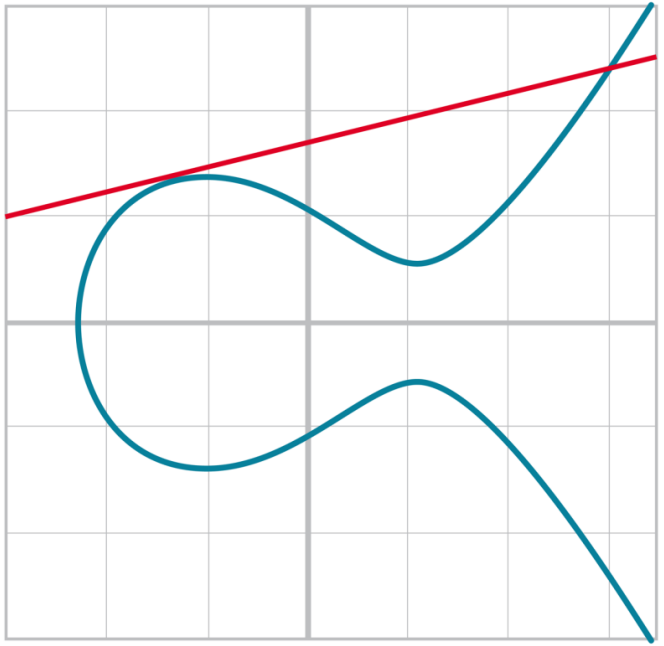
Caso 1: tres puntos de contacto



La curva y la recta

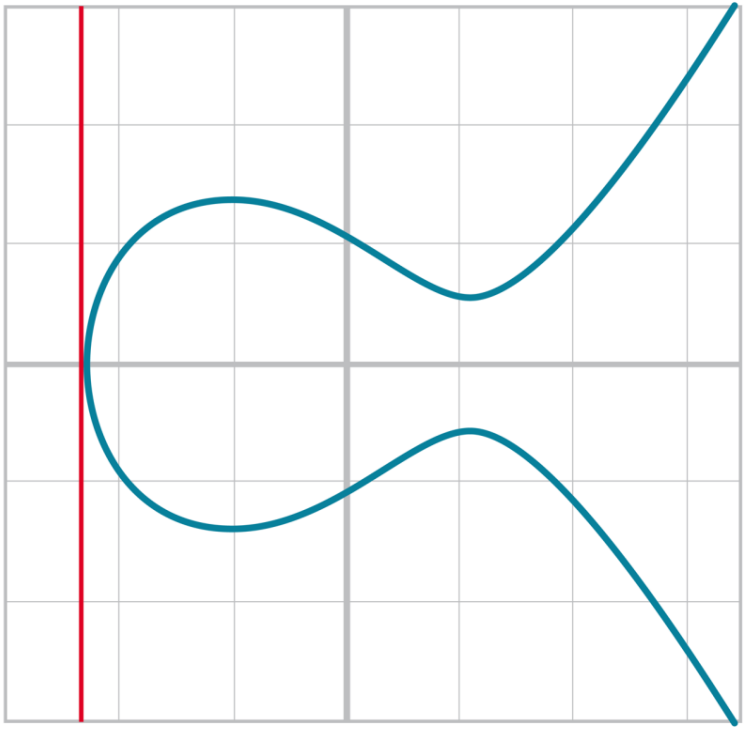
Caso 2: dos puntos de contacto





La curva y la recta

Caso 3: un punto de contacto



Suma de dos puntos

Propiedades de suma de números:

- *Elemento neutro:* existe un elemento 0 t.q. $0 + A = A + 0 = A$, para todo A
- *Conmutatividad:* $A + B = B + A$
- *Asociatividad:* $(A + B) + C = A + (B + C)$
- *Invertibilidad*: existe $-A$ t.q. $A + (-A) = (-A) + A = 0$, para cada A

Suma de dos puntos

Elemento neutro

$$\{ (x,y) : y^2 = x^3 + ax + b \} + \{ I \}$$

Suma de dos puntos

Elemento neutro

$$\{ (x,y) : y^2 = x^3 + ax + b \} + \{ I \}$$

Suma de dos puntos

Elemento neutro

$$\{ (x,y) : y^2 = x^3 + ax + b \} + \{ I \}$$ 

I + A = A, para cada A en la curva

Suma de dos puntos

Elemento inverso

$$\{ (x,y) : y^2 = x^3 + ax + b \} + \{ I \}$$

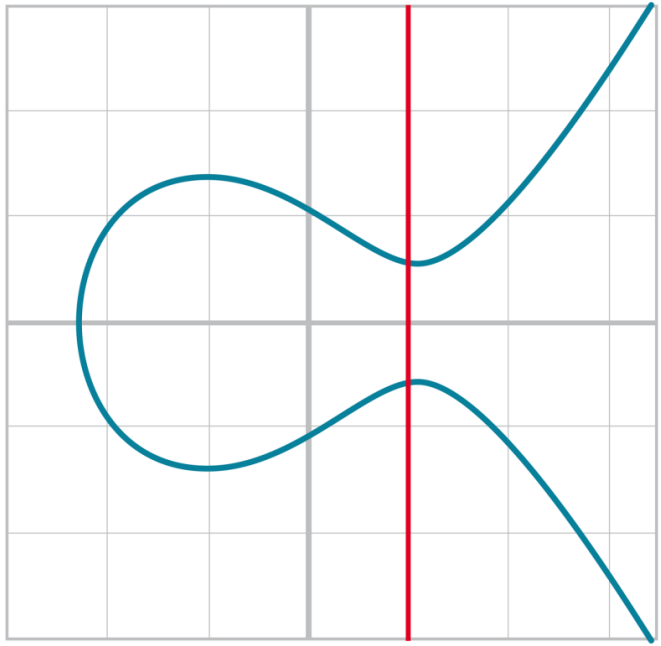
A = (x,y)  -A = (x,-y)

A + (-A) = I

Suma de dos puntos

Elemento inverso

Intuición: recta por los dos punto, el tercer punto de intersección (reflejado) es la suma

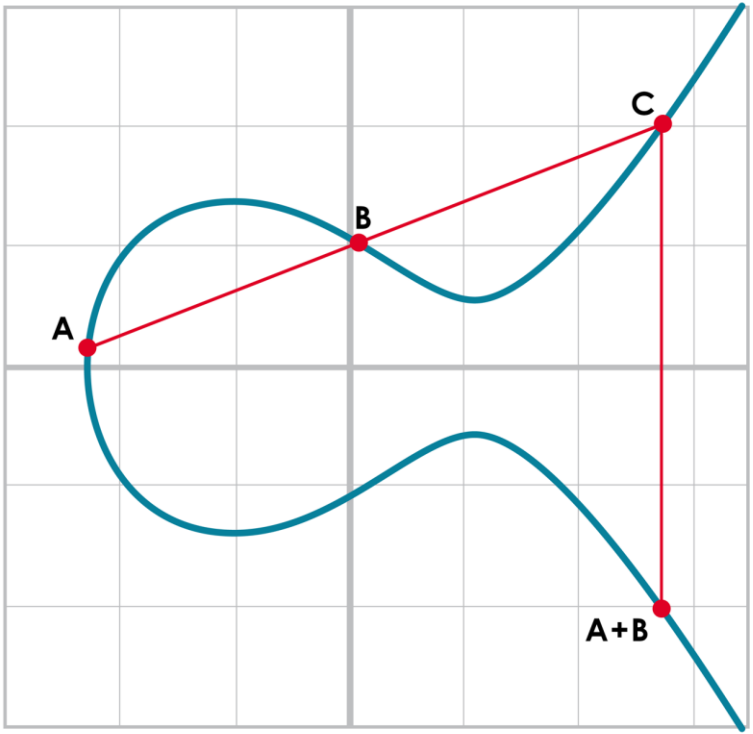


Suma de dos puntos

El caso más importante

Intuición:

1. Dos puntos definen una recta
2. Recta intersecta la curva en tercer punto
3. Imagen en espejo de axis y de la intersección es nuestra suma



Suma de dos puntos

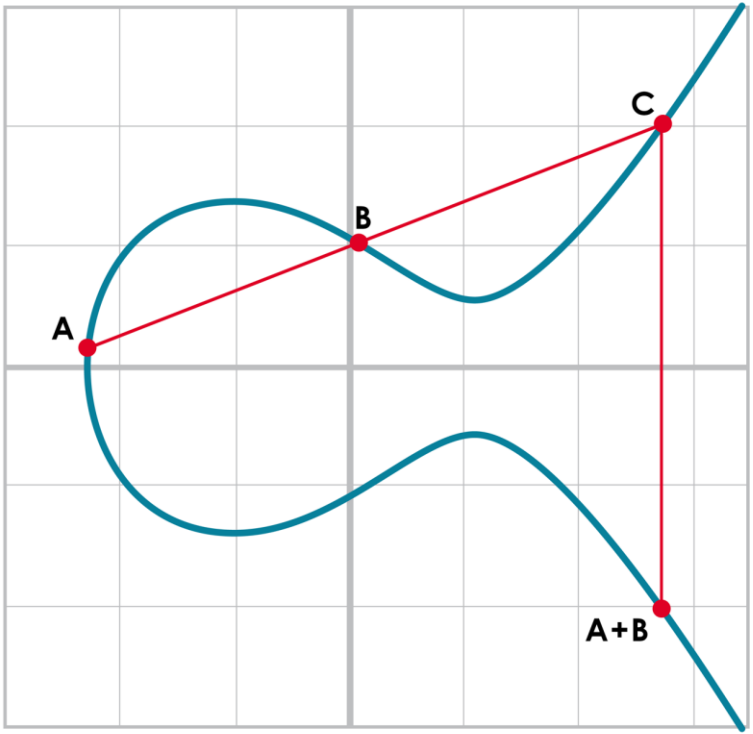
El caso más importante

¿Cómo calcular las coordenadas de A + B?

P1 = A = (x1,y1)
P2 = B = (x2,y2)
P3 = A + B = (x3,y3)

P1 + P2 = P3

¿Si conozco las coordenadas de P1 y P2, como computar las coordenadas de P3?



Suma de dos puntos

El caso más importante

P1 + P2 = P3

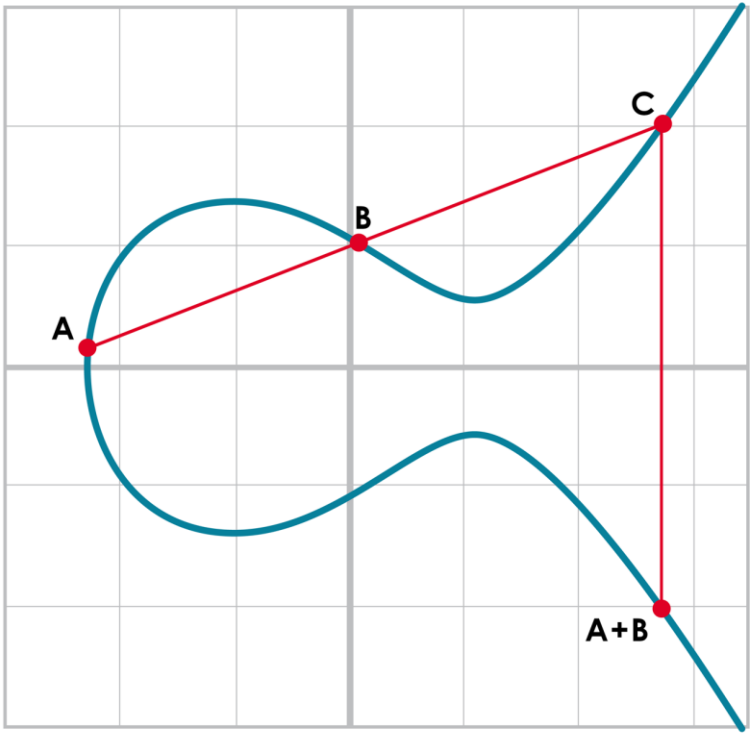
P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

x1 ≠ x2

y = s(x1- x) + y1

s = $\frac{y2-y1}{x2-x1}$

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



Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

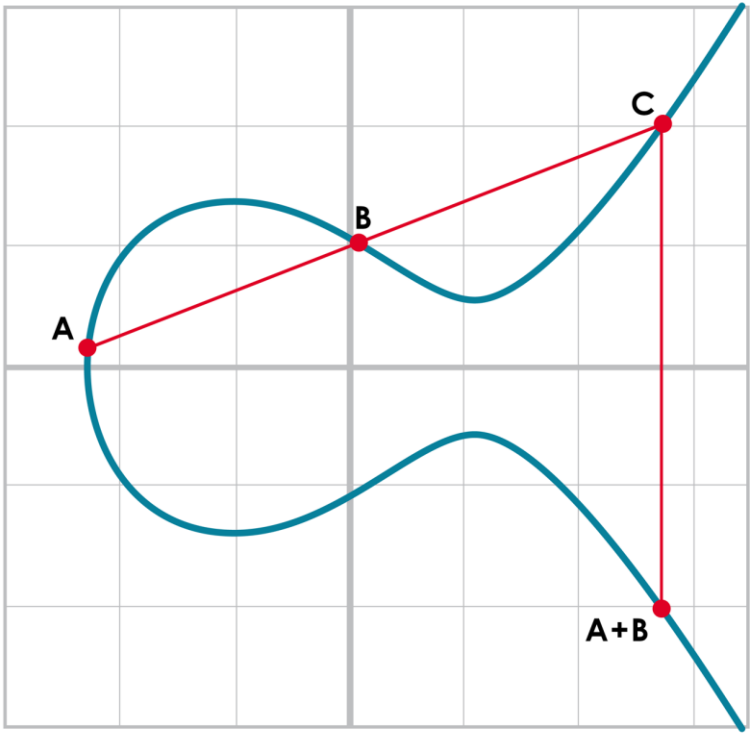
x1 ≠ x2

y = s(x1- x) + y1

s = $\frac{y2-y1}{x2-x1}$

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

$y^2 = (s(x - x1) + y1)^2 = x^3 + ax + b$



Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

$s = \frac{y2-y1}{x2-x1}$ $\qquad y^2 = x^3 + ax + b$

$y^2 = (\, s(x - x1) + y1\,)^2 = x^3 + ax + b$

$$x^3 - s^2x^2 + (a + 2s^2x1 - 2sy1)x + b - s^2x1^2 + 2sx1y1 - y1^2 = 0$$

Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

$s = \frac{y2-y1}{x2-x1}$ $\qquad y^2 = x^3 + ax + b$

Igual a cero para
x = x1,x2,x3

$y^2 = (s(x - x1) + y1)^2 = x^3 + ax + b$

$x^3 - s^2x^2 + (a + 2s^2x1 - 2sy1)x + b - s^2x1^2 + 2sx1y1 - y1^2 = 0$

Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

$s = \frac{y2-y1}{x2-x1}$ $\qquad y^2 = x^3 + ax + b$

$y^2 = (s(x - x1) + y1)^2 = x^3 + ax + b$

$x^3 - s^2x^2 + (a + 2s^2x1 - 2sy1)x + b - s^2x1^2 + 2sx1y1 - y1^2 = 0$

(x-x1)(x-x2)(x-x3) = 0

Igual a cero para
x = x1,x2,x3

Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

$s = \frac{y2-y1}{x2-x1}$ $\qquad y^2 = x^3 + ax + b$

$y^2 = (s(x - x1) + y1)^2 = x^3 + ax + b$

Vieta!!!
https://en.wikipedia.org/wiki/Vieta%27s_formulas

$x^3 - s^2x^2 + (a + 2s^2x1 - 2sy1)x + b - s^2x1^2 + 2sx1y1 - y1^2 = 0$

(x-x1)(x-x2)(x-x3) = 0

Suma de dos puntos

El caso más importante

P1 + P2 = P3

P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

$s = \frac{y2-y1}{x2-x1}$ $\qquad y^2 = x^3 + ax + b$

$y^2 = (s(x - x1) + y1)^2 = x^3 + ax + b$

Vieta!!! https://en.wikipedia.org/wiki/Vieta%27s_formulas

$x^3 - s^2x^2 + (a + 2s^2x1 - 2sy1)x + b - s^2x1^2 + 2sx1y1 - y1^2 = 0$

$x^3 - (x1+x2+x3)x^2 + (x1x2 + x1x3 + x2x3)x - x1x2x3 = 0$

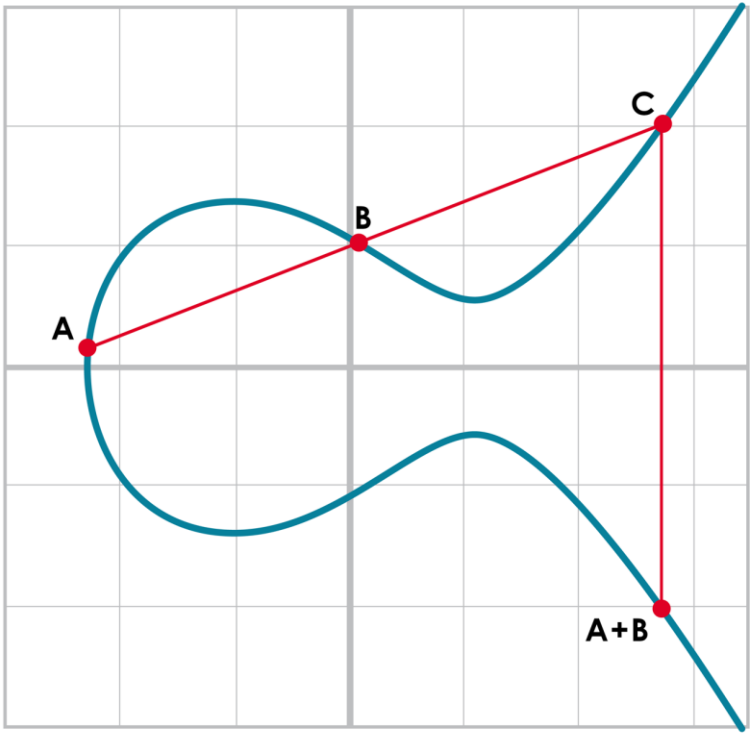
$s^2 = x1 + x2 + x3$

Suma de dos puntos

El caso más importante

P1 + P2 = P3
P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)
x1 ≠ x2

x3 = $s^2 - x1 - x2$
y3 = s(x1 - x3) – y1

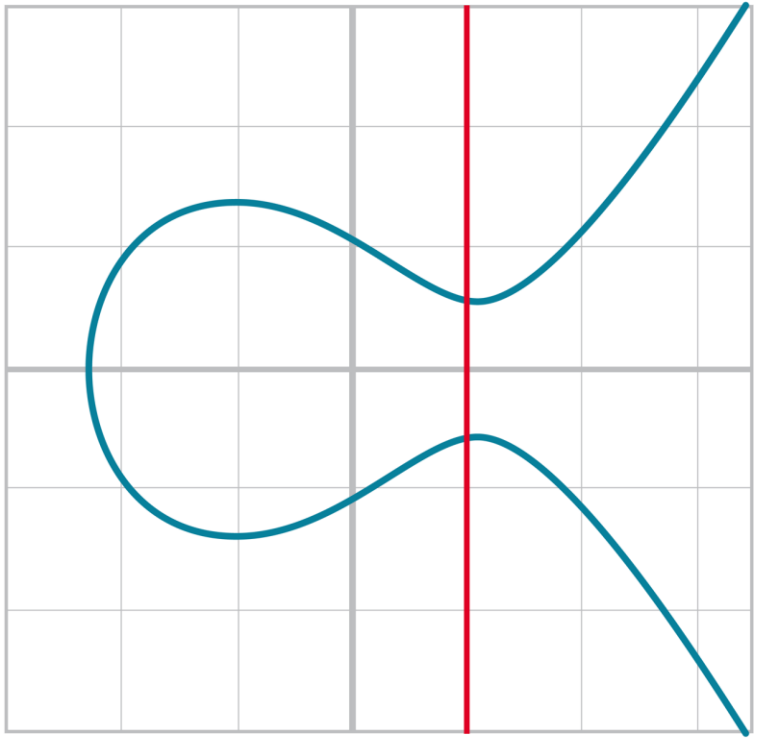


Suma de dos puntos

Otros casos

P1 + P2 = P3
P1 = (x1,y1) = P2 = (x2,y2), P3 = (x3,y3)
x1 = x2, y1 ≠ y2

P3 = I



Suma de dos puntos

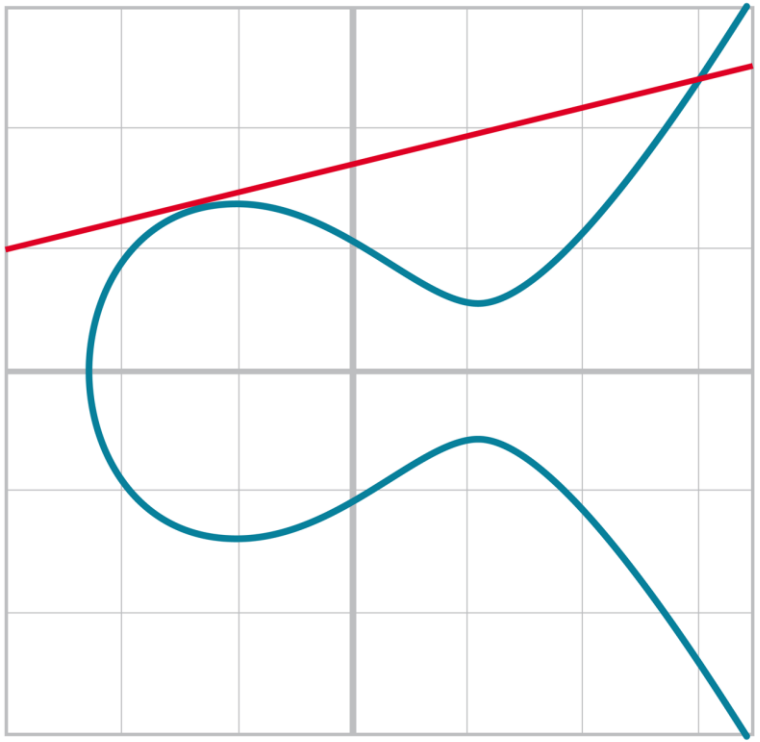
Otros casos

P1 + P2 = P3
P1 = P2 = (x1,y1)

Es un limite de dos puntos distintos

s de la recta tangente: derivada

s = $(3x1^2 + a)/(2y1)$
x3 = $s^2 - 2x1$
y3 = s(x1 – x3) – y1



Suma de dos puntos

Último caso

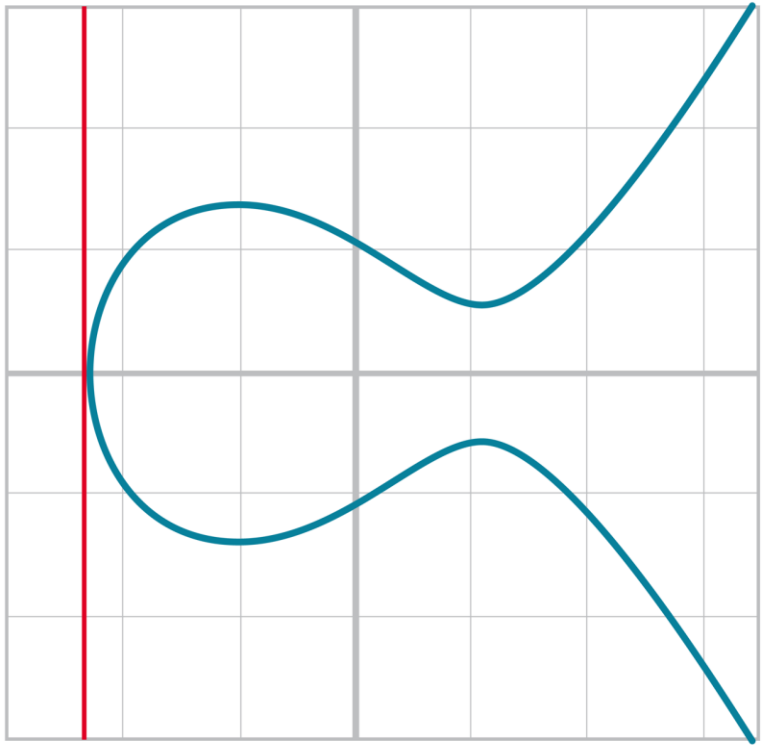
P1 + P2 = P3
P1 = P2 = (x1,y1)

Es un limite de dos puntos distintos

s de la recta tangente: derivada

s = $(3x1^2 + a)/(2y1)$
x3 = $s^2 - 2x1$
y3 = s(x1 – x3) – y1

y1 = 0 ⟹ P1 + P1 = I



Suma de dos puntos

Propiedades de suma de números:

- *Elemento neutro:* existe un elemento 0 t.q. 0 + A = A + 0 = A, para todo A
- *Conmutatividad:* A + B = B + A
- *Asociatividad:* (A + B) + C = A + (B + C)
- *Invertibilidad*: existe –A t.q. A + (-A) = (-A) + A = 0, para cada A

Fácil de demostrar ahora!

Suma de dos puntos

Implementación

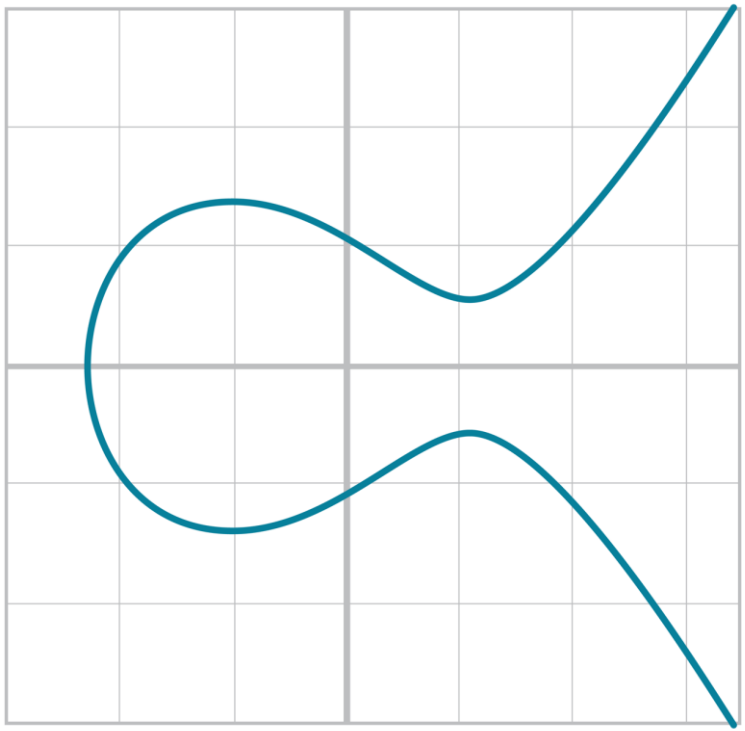
TODO.md | Programa.md | curves.py

```python
class Point:

    ...

    def __add__(self, other):
        if self.a != other.a or self.b != other.b:
            raise TypeError('Points {}, {} are not on the same curve'.format(self, other))
        # Case 0.0: self is the point at infinity, return other
        if self.x is None:
            return other
        # Case 0.1: other is the point at infinity, return self
        if other.x is None:
            return self

        # Case 1: self.x == other.x, self.y != other.y
        # Result is point at infinity
        if self.x == other.x and self.y != other.y:
            return self.__class__(None, None, self.a, self.b)

        # Case 2: self.x ≠ other.x
```

Curvas Elipticas

Nuestros gráficos son sobre reales

En práctica: curvas son sobre **cuerpos finitos**



Cuerpos finitos

Un cuerpo finito es una tupla F = (G,+,·,0,1) con estas propiedades:

1) *Clausura* : $a + b \in G$, y $a \cdot b \in G$, para cada $a,b \in G$
2) *Neutro aditivo*: $0 \in G$, y $0 + a = a$, para cada $a \in G$
3) *Neutro multiplicativo*: $1 \in G$, y $1 \cdot a = a$, para cada $a \in G$
4) *Inverso aditivo*: para cada $a \in G$, existe $-a \in G$ t.q. $a + (-a) = 0$
5) *Inverso multiplicativo*: para cada $a \in G$, $a \neq 0$, existe $a^{-1} \in G$, t.q. $a \cdot a^{-1} = 1$

Cuerpos finitos

Prototipo de cuerpos finitos:

$F_p = \{0,1,2,\ldots,p-1\}, p \text{ un número } \boldsymbol{primo}$

$a +_p b = (a+b)\,\%\,p \quad (modulo\ p)$

$a \cdot_p b = (a \cdot b)\,\%\,p \quad (modulo\ p)$

Cuerpos finitos

Inversos

¿Cómo se define el inverso aditivo?

-a % p $(a \in F_p \rightarrow -a = (p - a) \in F_p)$

E.g. $F_p = \{0,1,\ldots,18\}$, p = 19

-14 = 5 $\in F_p$

14 $+_p$ 5 = (14 + 5 = 19) % 19 = 0

Cuerpos finitos

Inversos

¿Cómo se define el inverso multiplicativo?

$$a \cdot_p b = (a \cdot b) \,\%\, p \quad (modulo\ p)$$

Pequeño teorema de Fermat: Si p es primo, y a > 0 es natural, entonces:

$$\boldsymbol{a^{p-1} \equiv 1\ (mod\ p)}$$

$$a^{-1} = a^{p-2} \,\%\, \text{p}$$

$$a \cdot_p a^{-1} = (a \cdot a^{p-2}) \,\%\, \text{p} = 1$$

Cuerpos finitos

Exponenciales

Pequeño teorema de Fermat: Si p es primo, y a > 0 es natural, entonces:

$$a^{p-1} \equiv 1 \ (mod\ p)$$

$a^k \in G = a^k$ % p

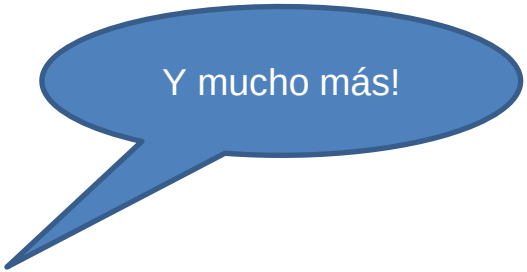
k >> p -> k = m · (p-1) + n

a^k % p = $a^{m \cdot (p-1)}$ % p · a^n % $p = a^n$ % p = $a^{k\%p}$ % p

Cuerpos finitos

Implementación

```python
class FieldElement:

    def __init__(self, num, prime):
        if num >= prime or num < 0:
            error = 'Num {} not in field range 0 to {}'.format(
                num, prime - 1)
            raise ValueError(error)
        self.num = num
        self.prime = prime

    def __repr__(self):
        return 'FieldElement_{}({})'.format(self.prime, self.num)

    def __eq__(self, other):
        if other is None:
            return False
        return self.num == other.num and self.prime == other.prime

    def __ne__(self, other):
        # this should be the inverse of the == operator
        return not (self == other)
```



Curvas elípticas

En el salvaje

Curvas elípticas se estudian sobre cuerpos finitos, y no sobre los reales

$y^2 = x^3 + 7$ *sobre* F_{103}

(17,64) está en la curva:

$y^2 = 64^2 \,\%\, 103 = 79$

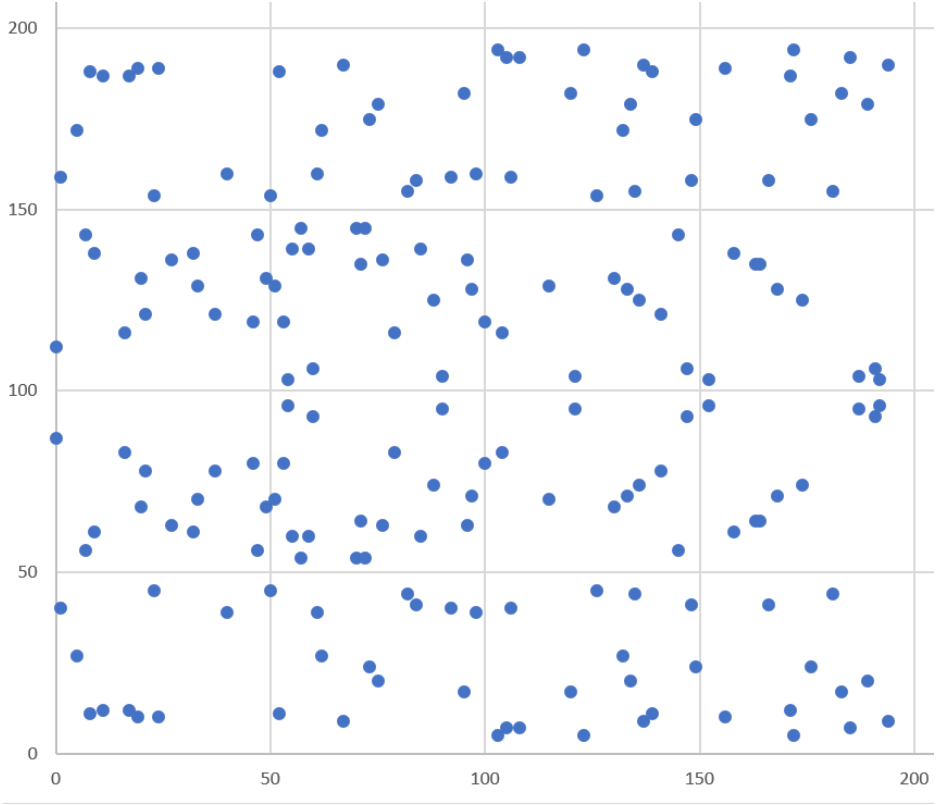
$x^3 + 7 = 17^3 + 7 \,\%\, 103 = 79$

Curvas elípticas

Curva $y^2 = x^3 + 7$ sobre F_{223}

Simétrica sobre la mitad

¿Cuántos puntos puede tener una curva sobre un campo finito?



Curvas elípticas

Suma de dos puntos

¿Cambia algo sobre los campos finitos?

P1 + P2 = P3
P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

1) x1 ≠ x2

x3 = $s^2 - x1 - x2$
y3 = s(x1 - x3) – y1
s = $\frac{y2-y1}{x2-x1}$

Curvas elípticas

Suma de dos puntos

¿Cambia algo sobre los campos finitos?

P1 + P2 = P3
P1 = (x1,y1), P2 = (x2,y2), P3 = (x3,y3)

1) x1 ≠ x2

x3 = $s^2 - x1 - x2$ % p
y3 = s(x1 - x3) – y1 % p
s = $\frac{y2-y1}{x2-x1}$ % p

2) P1 = P2

x3 = $s^2 - 2x1$ % p
y3 = s(x1 – x3) – y1 % p
s = (3$x1^2$ + a)/(2y1) % p

Multiplicación escalar

La operación más importante en el ecc

P

P + P

P + P + P

P + P + P + P

.

.

.

Multiplicación escalar

La operación más importante en el ecc

$P = 1 \cdot P$

$P + P = 2 \cdot P$

$P + P + P = 3 \cdot P$

$P + P + P + P = 4 \cdot P$

.

.

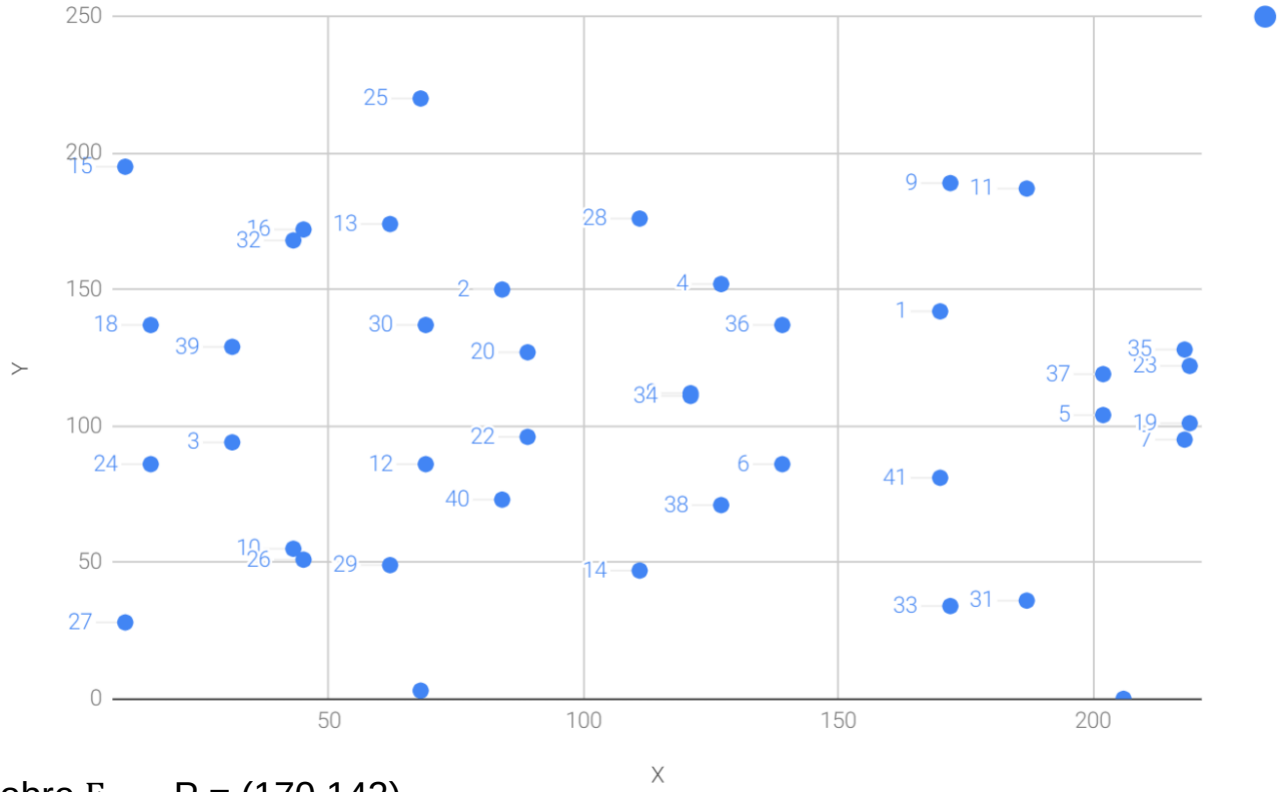
.

$a \cdot P$, para cualquier a número natural

Multiplicación escalar!!!

Multiplicación escalar

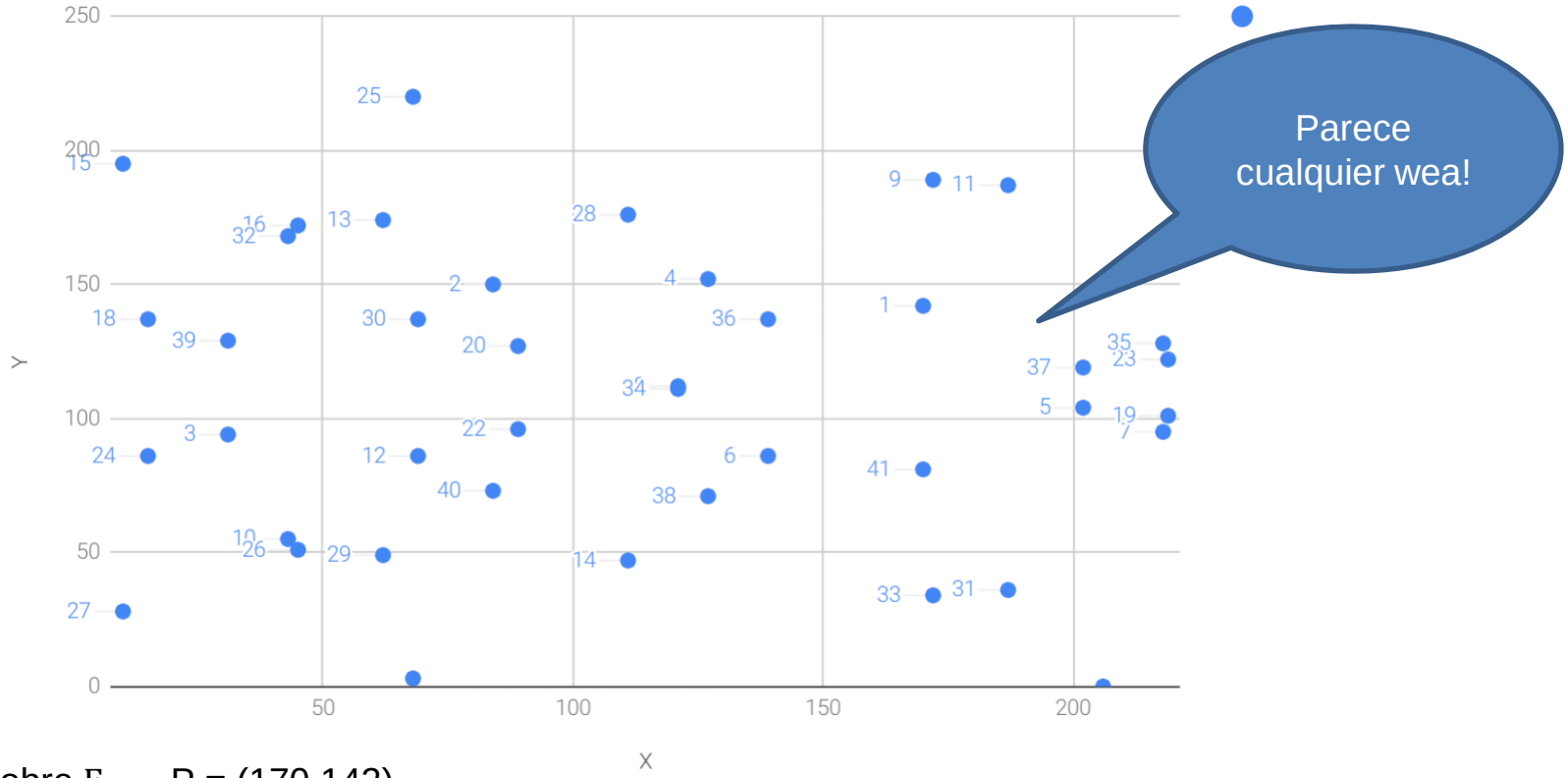
¿Cómo se ve esta operación gráficamente?



$y^2 = x^3 + 7$ sobre F_{223}, P = (170,142)

Multiplicación escalar

¿Cómo se ve esta operación gráficamente?



$y^2 = x^3 + 7$ sobre F_{223}, P = (170,142)

Multiplicación escalar

Problema de logaritmo discreto

$R = e \cdot P$

Si tengo P y R, el mejor algoritmo para encontrar e tiene que revisar todos los e posibles

I.e. No hay una forma analítica para la división escalar

The discrete log problem: invertir producto escalar en cualquier grupo

Una de las preguntas más conocidas en computación: ¿existe un algoritmo polinomial para resolver el problema de logaritmo discreto?

Multiplicación escalar

Una propiedad fundamental

Tengo una curva elíptica sobre un campo finito $\mathrm{F_p}$:

- Número de punto de mi curva es finito, digamos m
- Tomo un G en la curva:
- El conjunto {1 · G, 2 · G, 3 · G,…} es finito (P+Q está en la curva para P,Q en la curva)
- Entonces existe un n t.q. n · G = I (punto en el infinito)
- ¿Por qué?

Multiplicación escalar

Una propiedad fundamental

Tengo una curva elíptica sobre un campo finito $\mathrm{F_p}$:

- Número de punto de mi curva es finito, digamos m
- Tomo un G en la curva:
- El conjunto {1 · G, 2 · G, 3 · G,…} es finito (P+Q está en la curva para P,Q en la curva)
- Entonces existe un n t.q. n · G = I (punto en el infinito)
- ¿Por qué?

La curva con suma de puntos es un **grupo** finito

{1 · G, 2 · G, 3 · G,…, n · G} es un subgrupo cíclico de los punto en la curva

- ¿Cómo demuestro esto?

Multiplicación escalar

Problema de logaritmo discreto

$R = e \cdot P$

Si tengo P y R, el mejor algoritmo para encontrar e tiene que revisar ***todos los e posibles***

Máximo e es el número de punto en la curva en realidad

El algoritmo brute-force se considera exponencial!!!

- Tema de representación de los números

Multiplicación escalar

Problema de logaritmo discreto

$R = e \cdot P$

¿Por qué se llama el *logaritmo* discreto? (Y no la división.)

En matemática un grupo es $G = (A, \cdot_G)$, y no $(A, +_G)$

Entonces: $P \cdot_G P \cdot_G P \cdot_G P \cdot_G P \cdot_G P = P^7$

Estamos resolviendo $P^7 = R$, conociendo P y R

Entonces buscamos $7 = \log_P R$

Multiplicación escalar

Un algoritmo inteligente

Double and add

$151 \cdot P$

$151 = 10010111_2$

$151 \cdot P = 2^7 \cdot P + 2^4 \cdot P + 2^2 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

Multiplicación escalar

Un algoritmo inteligente

$151 \cdot P = 2^7 \cdot P + 2^4 \cdot P + 2^2 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

$151 = 10010111_2$

P

1) SUM = I
2) SUM += P
3) P += P (= 2P)
4) SUM += P
5) P += P (= 4P)
6) SUM += P
7) P += P (= 8P)
8) P += P (= 16P)
9) SUM += P
10) P += P (= 32P)
11) P += P (= 64P)
12) P += P (= 128P)
13) SUM += P

$SUM = 2^7 \cdot P + 2^4 \cdot P + 2^2 \cdot P + 2^1 \cdot P + 2^0 \cdot P$

Multiplicación escalar

Un algoritmo inteligente

$e \cdot P = (d_1 d_2 \ldots d_k)_2 \cdot P$

```
current = P
sum = I

for i = k downto 1
        if d_k == 1
                sum += current
        current += current
```

Curva elíptica de Bitcoin

secp256k1

¿Qué define a una curva elíptica en el mundo real?

1) $y^2 = x^3 + ax + b$ → Parámetros a y b

2) Sobre un cuerpo finito F_p → orden del cuerpo p

3) Un punto especial G = (G_x,G_y) que está en la curva

4) El subgupo {G,2G,3G,…,nG = I} generado por G → orden n del subgrupo

Curva elíptica de Bitcoin

secp256k1

¿Qué define a una curva elíptica en el mundo real?

1) $y^2 = x^3 + ax + b$ → Parámetros **a** y **b**

2) Sobre un cuerpo finito F_p → orden del cuerpo **p**

3) Un punto especial **$G = (G_x, G_y)$** que está en la curva

4) El subgupo {G,2G,3G,…,nG = I} generado por G → orden **n** del subgrupo

Curva elíptica de Bitcoin

secp256k1

¿Qué define a una curva elíptica en el mundo real?

1) $y^2 = x^3 + ax + b$ → Parámetros **a** y **b**

2) Sobre un cuerpo finito F_p → orden del cuerpo **p**

3) Un punto especial **$G = (G_x, G_y)$** que está en la curva

4) El subgupo {G,2G,3G,…,nG = I} generado por G → orden **n** del subgrupo

Todo ocurre aquí!

Curva elíptica de Bitcoin

secp256k1

¿Que ocurre?

- $y^2 = x^3 + ax + b$ sobre F_p define un grupo finito (puntos con la suma)
- El subgupo {G,2G,3G,…,nG = I} generado por G es un subgrupo de este grupo

Problema de logaritmo discreto vamos a mirar en {G,2G,3G,…,nG = I}

Si n es primo podemos definir inversas de un escalar en el campo F_n**!!!**

Curva elíptica de Bitcoin

secp256k1

1) $y^2 = x^3 + 7$ → Parámetros **a = 0** y **b = 7**

2) Sobre un cuerpo finito F_p → orden del cuerpo **$p = 2^{256} - 2^{32} - 977$**

3) Un punto especial **$G = (G_x, G_y)$** que está en la curva

G_x = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
G_y = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8

4) El subgupo {G,2G,3G,…,nG = I} generado por G → orden **n** del subgrupo

n = 0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

Curva elíptica de Bitcoin

secp256k1

1) $y^2 = x^3 + 7$ → Parámetros **a = 0** y **b = 7**

2) Sobre un cuerpo finito F_p → orden del cuerpo **p = 2^{256} - 2**

3) Un punto especial **G = (G_x,G_y)** que está en la curva

G_x = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce2 …b16f81798
G_y = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a685 …99c47d08ffb10d4b8

4) El subgupo {G,2G,3G,…,nG = I} generado por G → orden n del subgrupo

n = 0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

n es un número primo

Curva elíptica de Bitcoin

secp256k1

$p = 2^{256} - 2^{32} - 977$

$n = 0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141$

Los dos son casi 2^{256} -- las coordenadas y escalares se pueden expresar en 256 bits

Acuérdense del sha256 = 256 bits!!!

Curva elíptica de Bitcoin

Llaves

$y^2 = x^3 + 7$

n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

G = (G_x,G_y)

Llave secreta: un número *e* de 256 bits (en realidad en [0,n])
Llave pública: el punto P = e · G en nuestra curva

e es mi secreto
P publico y sirve como mi identidad!

Curva elíptica de Bitcoin

Llaves

$y^2 = x^3 + 7$

n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

G = (G_x,G_y)

Llave secreta: un número *e* de 256 bits
Llave pública: el punto P = e · G en nuestra curva

¿Si conozco P y G, porque no puedo reconstruir *e*?

Curva elíptica de Bitcoin

Llaves

$y^2 = x^3 + 7$

n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

$G = (G_x, G_y)$

Llave secreta: un número *e* de 256 bits
Llave pública: el punto $P = e \cdot G$ en nuestra curva

¿Si conozco P y G, porque no puedo reconstruir *e*?

Discrete log of size **n!!!**

Curva elíptica de Bitcoin

Llaves

$y^2 = x^3 + 7$

n = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141

G = (G_x,G_y)

Llave secreta: un número *e* de 256 bits
Llave pública: el punto P = e · G en nuestra curva

¿Cómo genero *e*?

1) e = randint(0,n) ← pero con rand de verdad, no de Python porfa!
2) e = sha256(b'mi secreto # 7893') ← "brain wallet"
3) e = sha256(sha256(b'mi secreto # 7893') + b'sal') ← "brain wallet con sal"

Curva elíptica de Bitcoin

¿Cómo firmo a mi documento?

Llave secreta: un número e de 256 bits
Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$

1. Elegir k aleatoriamente (pero aleatoriamente de verdad) en [0,n]
2. Calcular el punto $F = k \cdot G = (F_x, F_y)$
3. Sea $r = F_x$
4. Si r == 0 goto 1.
5. Sea $s = k^{-1}(z + r \cdot e)$
6. Si s == 0 goto 1.

Mi firma es (r,s)

Curva elíptica de Bitcoin

¿Cómo firmo a mi documento?

Llave secreta: un número *e* de 256 bits
Llave pública: el punto P = e · G en nuestra curva
Documento: d → firmo *z = hash256(d)*

Todos los cálculos en F_n

1. Elegir *k* aleatoriamente (pero aleatoriamente de verdad)
2. Calcular el punto $F = k \cdot G = (F_x, F_y)$
3. Sea $r = F_x$
4. Si r == 0 goto 1.
5. Sea $s = k^{-1}(z + r \cdot e)$
6. Si s == 0 goto 1.

Mi firma es *(r,s)*

Curva elíptica de Bitcoin

¿Cómo firmo a mi documento?

Llave secreta: un número *e* de 256 bits
Llave pública: el punto P = e · G en nuestra curva
Documento: d → firmo *z = hash256(d)*

1. Elegir *k* aleatoriamente (pero aleatoriamente [illegible]rdad)
2. Calcular el punto $F = k \cdot G = (F_x, F_y)$
3. Sea $r = F_x$
4. Si r == 0 goto 1.
5. Sea $s = k^{-1}(z + r \cdot e)$
6. Si s == 0 goto 1.

Mi firma es *(r,s)*

Todos los cálculos en F_n
(n es primo)

$k^{-1} = k^{(n-2)}$ % n (Fermat)
$s = k^{-1}(z+re)$ % n

Curva elíptica de Bitcoin

¿Cómo verifico una firma?

Recibo: La firma (r,s)
Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$

1. Calculo $u = s^{-1}z$
2. Calculo $v = s^{-1}r$
3. Calculo $C = u \cdot G + v \cdot P = (C_x, C_y)$

Return $C_x == r$

Curva elíptica de Bitcoin

¿Cómo verifico una firma?

Recibo: La firma (r,s)
Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$

Todos los cálculos en F_n
(n es primo)

1. Calculo $u = s^{-1}z$
2. Calculo $v = s^{-1}r$
3. Calculo $C = u \cdot G + v \cdot P = (C_x, C_y)$

Return $C_x == r$

Curva elíptica de Bitcoin

¿Cómo verifico una firma?

Recibo: La firma (r,s)
Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$

1. Calculo $u = s^{-1}z$
2. Calculo $v = s^{-1}r$
3. Calculo $C = u \cdot G + v \cdot P = (C_x, C_y)$

Return $C_x == r$

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$
Verificación: $u = s^{-1}z$, $v = s^{-1}r$, $C = u \cdot G + v \cdot P = (C_x, C_y)$

$$
\begin{aligned}
C &= u \cdot G + v \cdot P \\
&= u \cdot G + ve \cdot G \\
&= (u + ve) \cdot G \\
&= (s^{-1}z + s^{-1}re) \cdot G \\
&= s^{-1}(z + re) \cdot G
\end{aligned}
$$

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$ → $k = s^{-1}(z + re)$
Verificación: $u = s^{-1}z$, $v = s^{-1}r$, $C = u \cdot G + v \cdot P = (C_x, C_y)$

$$
\begin{aligned}
C &= u \cdot G + v \cdot P \\
&= u \cdot G + ve \cdot G \\
&= (u + ve) \cdot G \\
&= (s^{-1}z + s^{-1}re) \cdot G \\
&= s^{-1}(z + re) \cdot G
\end{aligned}
$$

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$ → $k = s^{-1}(z + re)$
Verificación: $u = s^{-1}z$, $v = s^{-1}r$, $C = u \cdot G + v \cdot P = (C_x, C_y)$

$$
\begin{aligned}
C &= u \cdot G + v \cdot P \\
&= u \cdot G + ve \cdot G \\
&= (u + ve) \cdot G \\
&= (s^{-1}z + s^{-1}re) \cdot G \\
&= s^{-1}(z + re) \cdot G \\
&= k \cdot G \\
&= F
\end{aligned}
$$

Curva elíptica de Bitcoin

¿Cómo firmo a mi documento?

Llave secreta: un número e de 256 bits
Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d → firmo $z = hash256(d)$

1. Elegir k **aleatoriamente!!!!**
2. Calcular el punto $F = k \cdot G = (F_x, F_y)$
3. Sea $r = F_x$
4. Si r == 0 goto 1.
5. Sea $s = k^{-1}(z + r \cdot e)$
6. Si s == 0 goto 1.

Mi firma es (r,s)

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d_1, d_2 → firmo $z_1 = hash256(d_1)$, $z_2 = hash256(d_2)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$

$kG = (r,y)$
$s_1 = k^{-1}(z_1 + r \cdot e)$, $s_2 = k^{-1}(z_2 + r \cdot e)$
$s_1/s_2 = (z_1 + r \cdot e)/(z_2 + r \cdot e)$
$s_1(z_2 + r \cdot e) = s_2(z_1 + r \cdot e)$
$s_1z_2 + s_1re = s_2z_1 + s_2re$
$s_1re - s_2re = s_2z_1 - s_1z_2$
$e = (s_2z_1 - s_1z_2)/(rs_1 - rs_2)$

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d_1, d_2 → firmo $z_1 = hash256(d_1)$, $z_2 = hash256(d_2)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$

kG = (r,y)

$s_1 = k^{-1}(z_1 + r \cdot e)$, $s_2 = k^{-1}(z_1 + r \cdot e)$

$s_1/s_2 = (z_1 + r \cdot e)/(z_2 + r \cdot e)$

$s_1(z_2 + r \cdot e) = s_2(z_1 + r \cdot e)$

$s_1z_2 + s_1re = s_2z_1 + s_2re$

$s_1re - s_2re = s_2z_1 - s_1z_2$

$e = (s_2z_1 - s_1z_2)/(rs_1 - rs_2)$

Curva elíptica de Bitcoin

¿Por qué funciona esto?

Llave pública: el punto $P = e \cdot G$ en nuestra curva
Documento: d_1, d_2 → firmo $z_1 = hash256(d_1)$, $z_2 = hash256(d_2)$
Firma: (r,s); $r = F_x$; $F = k \cdot G$, $s = k^{-1}(z + r \cdot e)$

kG = (r,y)

$s_1 = k^{-1}(z_1 + r \cdot e)$, $s_2 = k^{-1}(z_1 + r \cdot e)$

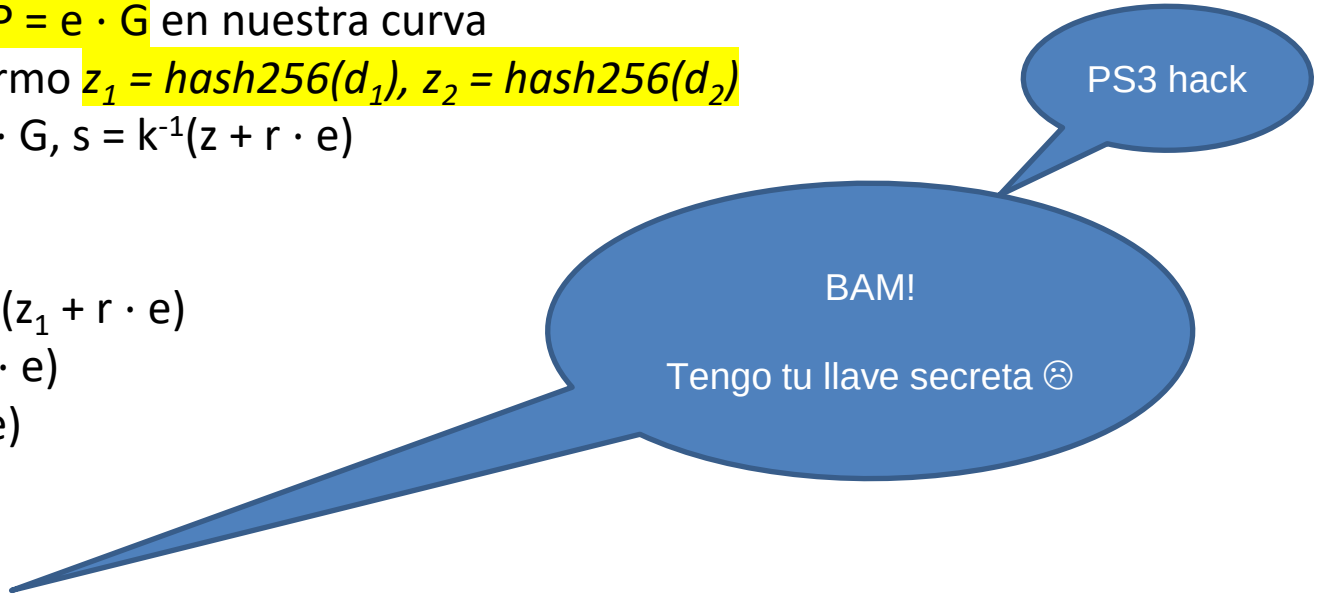
$s_1/s_2 = (z_1 + r \cdot e)/(z_2 + r \cdot e)$

$s_1(z_2 + r \cdot e) = s_2(z_1 + r \cdot e)$

$s_1z_2 + s_1re = s_2z_1 + s_2re$

$s_1re - s_2re = s_2z_1 - s_1z_2$

$e = (s_2z_1 - s_1z_2)/(rs_1 - rs_2)$



Serialización

Objetos que se transfieren por la red:

1. Llave pública
2. La firma
3. Una dirección de Bitcoin (wait for it)

Todos estos objetos tenemos que transferir en cierto formato:

- Nuestra implementación de ECC, llaves, y firmas van a ser objetos de Python
- Esto no es algo que podemos pasar a otra persona que quizás no está utilizando Python

Quiere decir: tenemos que serializar la **llave pública y la firma**

Formato SEC

Standards for Efficient Cryptography

Existe un estándar para serialización de llaves en ECDSA: SEC

SEC permite dos tipos de serialización:

1. Llave no comprimida (uncompressed SEC format)
2. Llave comprimida (compressed SEC format)

Formato SEC

Uncompressed SEC format

P = (x,y) es una llave pública

x, y son números de 256 bits = 32 bytes (1 byte = 8 bits = 2 números hex)

Serialización SEC no-comprimida de P =(x,y):

1. prefix = 0x04
2. x_{SEC} = x in 32 bytes big-endian int
3. y_{SEC} = y in 32 bytes big-endian int

Return prefix + x_{SEC} + y_{SEC} (+ es concatenación de bytes)

Formato SEC

Uncompressed SEC format

P = (x,y) es una llave pública

x, y son números de 256 bits = 32 bytes (1 byte = 8 bits = 2 números hex)

Serialización SEC no-comprimida de P =(x,y):

1. prefix = 0x04
2. x_{SEC} = x in 32 bytes big-endian int
3. y_{SEC} = y in 32 bytes big-endian int

```
047211a824f55b505228e4c3d5194c1fcfaa15a456abdf37f9b9d97a4040afc073dee6c8906498
4f03385237d92167c13e236446b417ab79a0fcae412ae3316b77
```

- 04 - Marker
- x coordinate - 32 bytes
- y coordinate - 32 bytes

Formato SEC

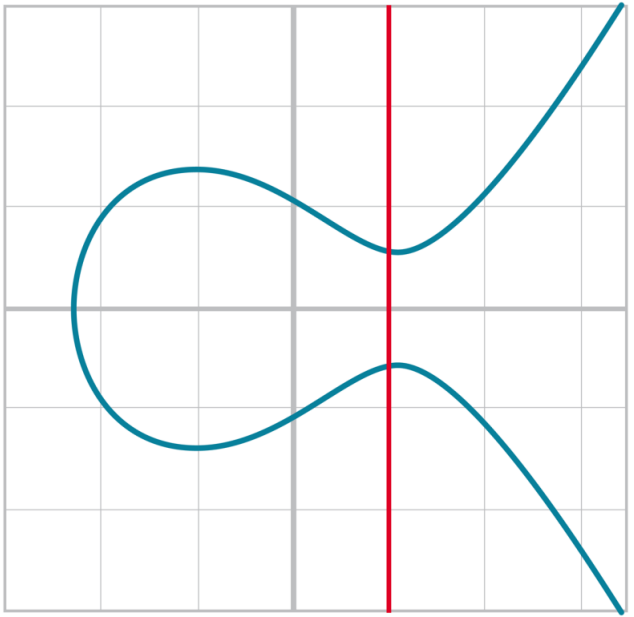
Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

Si conozco el x, hay solo dos candidatos para y!!!

(x,y) , (x,-y)

Para codificar esto necesito: x + 1 bit para decir si uso y o -y



Formato SEC

Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

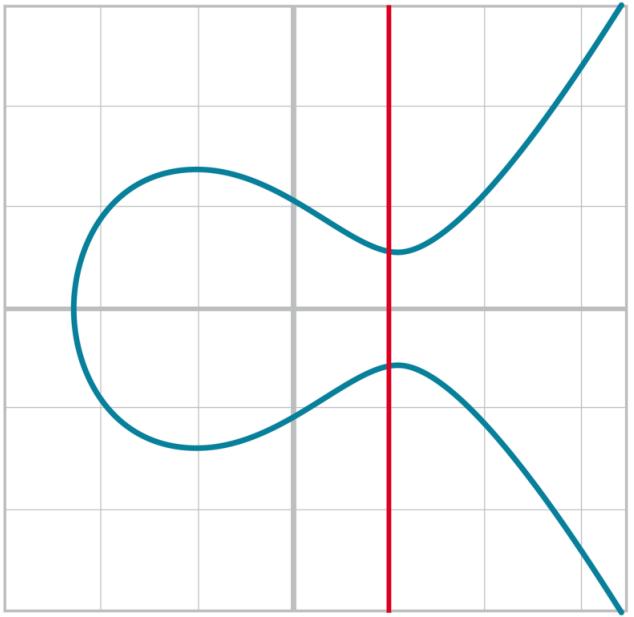
Si tengo x → (x,y), (x,-y) son buenos

-y = (p-y) % p, porque trabajamos sobre F_p

Si tengo x → (x,y), (x,p-y) son buenos

p un número primo mayor que 2

y par → p-y impar
y impar → p-y par



Formato SEC

Compressed SEC format

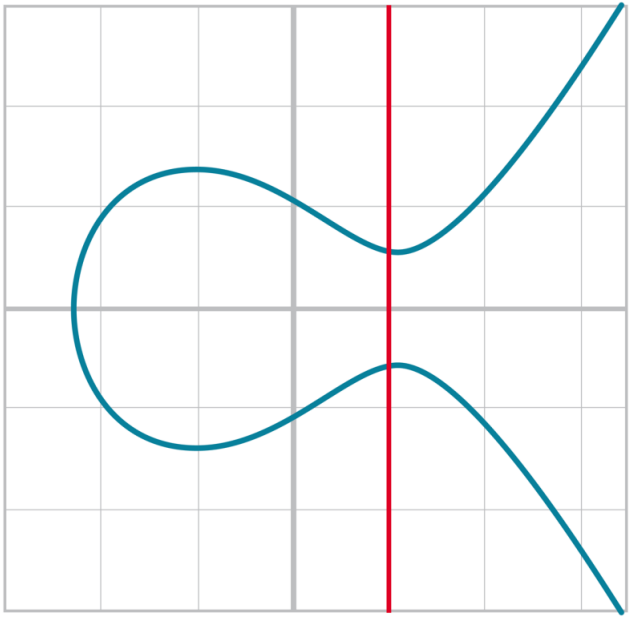
P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

Si tengo x → (x,y), (x,-y) son buenos

y par → p-y impar
y impar → p-y par

SEC comprimido:

- x
- Paridad de y (par o impar)



Formato SEC

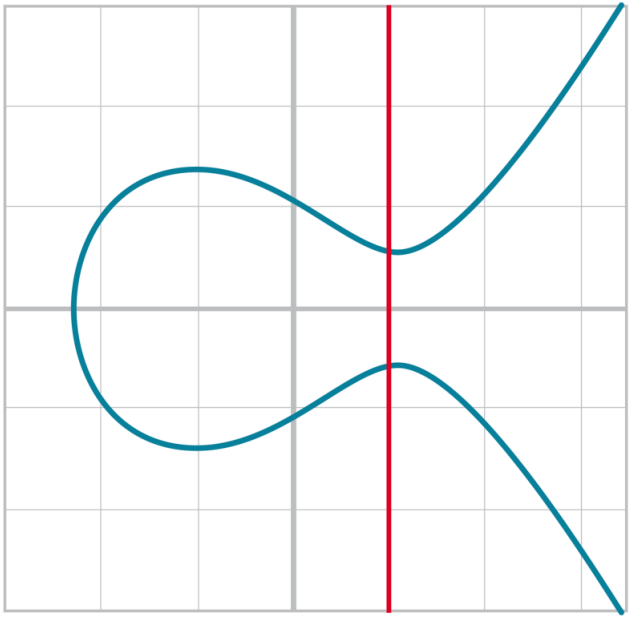
Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

SEC comprimido:

1. prefix = 0x02 si y es par, 0x03 si no (1 byte)
2. x_{SEC} = x en 32 bytes big-endian

Return prefix + x_{SEC} (33 bytes)



Formato SEC

Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

SEC comprimido:

1. prefix = 0x02 si y es par, 0x03 si no (1 byte)
2. x_{SEC} = x en 32 bytes big-endian

Return prefix + x_{SEC}

```
0349fc4e631e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278a

        - 02 if y is even, 03 if odd - Marker
        - x coordinate - 32 bytes
```

Formato SEC

Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

Tengo x, y la paridad de y: ¿cómo computo el y en F_p?

Estamos resolviendo la ecuación w^2 = v en el campo F_p

El truco: asumamos que p % 4 = 3 (como en la curva de BitCoin)

p % 4 = 3 → (p+1) % 4 = 0 → (p+1)/4 es un número natural

Formato SEC

Compressed SEC format

Estamos resolviendo la ecuación w^2 = v en el campo F_p

p % 4 = 3 → (p+1) % 4 = 0 → (p+1)/4 es un número natural

Pequeño teorema de Fermat: w^{p-1} % p = 1

$w^2 = w^2 \cdot 1 = w^2 \cdot w^{p-1} = w^{p+1}$ → p es primo > 2 → p+1 es par

$$w = w^{(p+1)/2} = w^{2(p+1)/4} = (w^2)^{(p+1)/4} = v^{(p+1)/4}$$

w^2 = v

Formato SEC

Compressed SEC format

P = (x,y) es una llave pública, y vive en una curva $y^2 = x^3 + ax + b$

SEC comprimido:

1. prefix = 0x02 si y es par, 0x03 si no (1 byte)
2. x_{SEC} = x en 32 bytes big-endian

$y_{SEC} = x_{SEC}^{(p+1)/4}$

If prefix == 0x02 AND y_{SEC} % 2 == 0 → P = (x_{SEC},y_{SEC}), else P = (x_{SEC},p - y_{SEC})

If prefix == 0x03 AND y_{SEC} % 2 == 0 → P = (x_{SEC},p - y_{SEC}), else P = (x_{SEC},y_{SEC})

Formato DER

Digital Encoding Rules

Para mandar la firma usamos DER (por culpa de Satoshi de nuevo, o de OpenSSH en 2008):

firma = *(r,s), r* y *s* son de 32 bytes; esto no se puede comprimir

1. *prefix = 0x30*
2. *len = largo* del resto de la firma (en hex; normalmente 0x44, o 0x45)
3. *marker = 0x02*
4. *r* en big-endian, si *r[0] ≥ 0x80 r = 0x00 + r* (concatenación de bytes)
5. marker = 0x02
6. s en big-endian, si *s[0] ≥ 0x80 s = 0x00 + s*

Return prefix + len + marker + len(r) + r + marker + len(s) + s

Formato DER

Digital Encoding Rules

Para mandar la firma usamos DER (por culpa de Satoshi de nuevo, o de OpenSSH en 2008):

firma = *(r,s), r* y *s* son de 32 bytes; esto no se puede comprimir

1. *prefix = 0x30*
2. *len = largo* del resto de la firma (en hex; normalmente 0x44, o 0
3. *marker = 0x02*
4. *r* en big-endian, si *r[0] ≥ 0x80 r = 0x00 + r* (concat ación de bytes)
5. marker = 0x02
6. s en big-endian, si *s[0] ≥ 0x80 s = 0x00 + s*

¿Qué es esto?

Return prefix + len + marker + len(r) + r + marker + len(s) + s

Formato DER

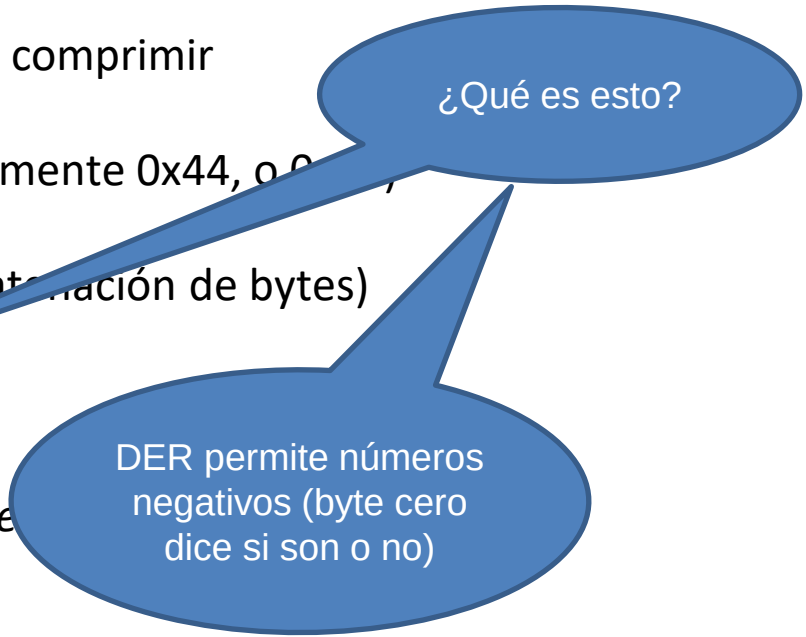
Digital Encoding Rules

Para mandar la firma usamos DER (por culpa de Satoshi de nuevo, o de OpenSSH en 2008):

firma = *(r,s), r* y *s* son de 32 bytes; esto no se puede comprimir

1. *prefix = 0x30*
2. *len = largo* del resto de la firma (en hex; normalmente 0x44, o 0
3. *marker = 0x02*
4. *r* en big-endian, si *r[0] ≥ 0x80 r = 0x00 + r* (concat ación de bytes)
5. marker = 0x02
6. s en big-endian, si *s[0] ≥ 0x80 s = 0x00 + s*

Return prefix + len + marker + len(r) + r + marker + le

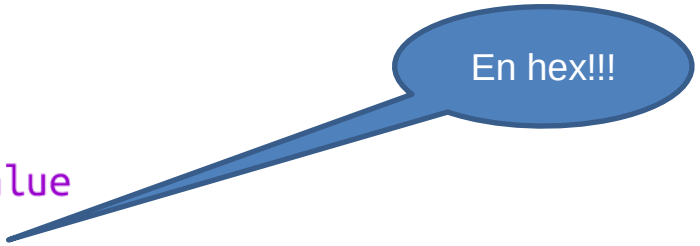


Formato DER

Digital Encoding Rules

```
3045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf213
20b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d7f67801
c31967743a9c8e10615bed
```

- 30 - Marker
- 45 - Length of sig
- 02 - Marker for r value
- 21 - r value lenth
- 00ed...8f - r value
- 02 - Marker for s value
- 20 - s value length
- 7a98...ed - s value



Direcciones de BitCoin

Una llave en SEC (comprimido o no) se puede acortar (de 33/64 a 20 bytes):

- La vamos a acortar y poner más seguridad utilizando *ripemd160*
- *ripemd160* es una función de hash con salida de 160 bits

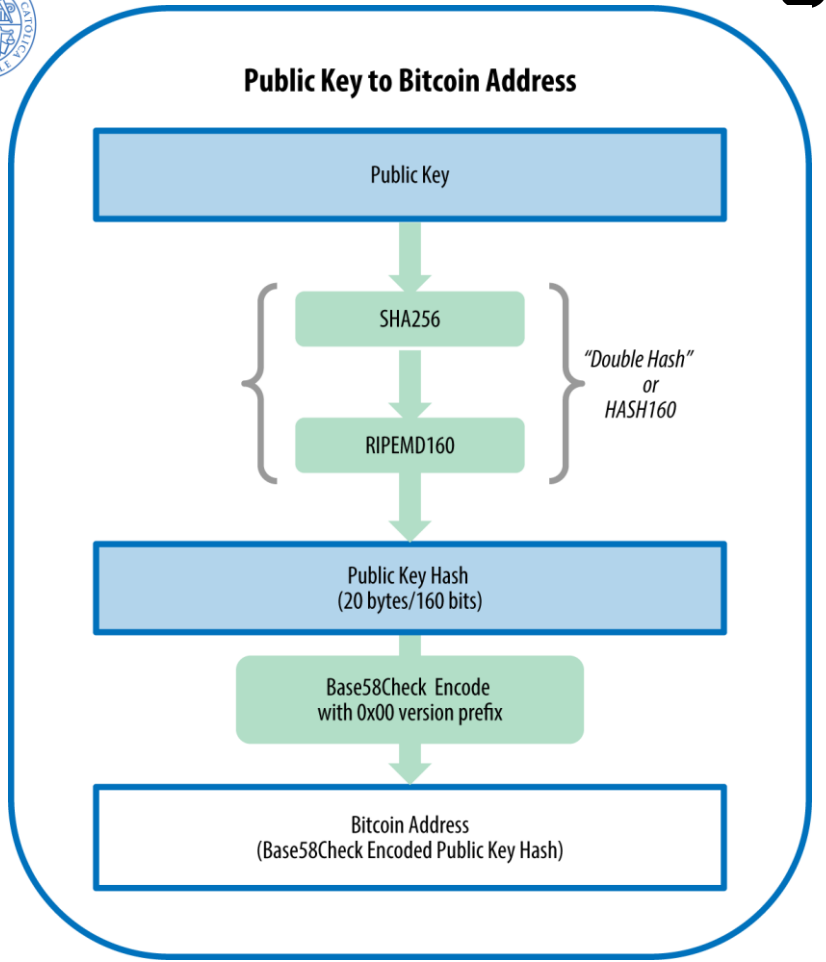
Una llave puede ser más legible si es más corta:

- Vamos a acortar la representación usando la base 58
- [0-9] + [a-z] + [A-Z]
- Menos: 0/O, I/l (cero, O mayúscula, I mayúscula, l minúscula)

¿Cómo definimos una llave de Bitcoin así?

Una dirección de Bitcoin

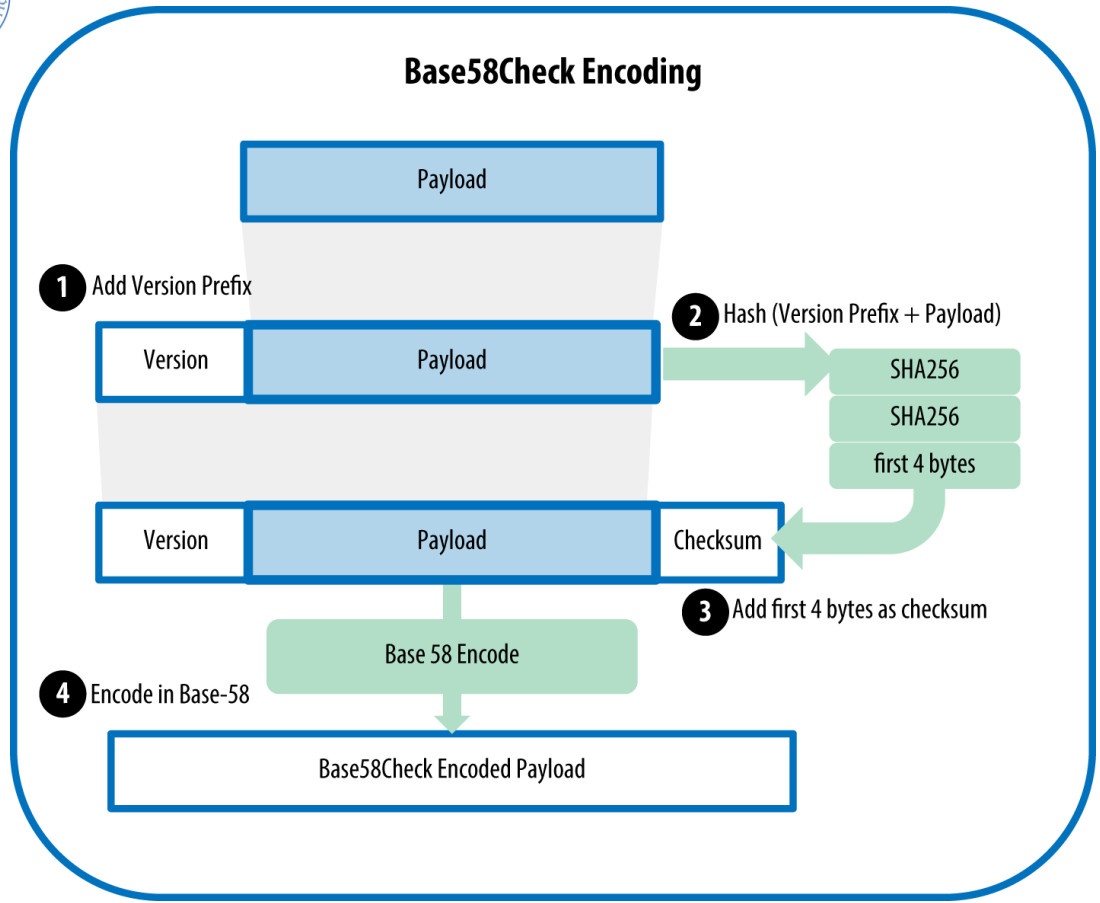
No es una llave publica



https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc

Una dirección de Bitcoin

No es una llave publica



https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc

Direcciones de BitCoin

P una llave en SEC

¿Cómo saco la dirección de Bitcoin?

1. *version = 0x00* para mainnet, y *0x6f* para testnet
2. *key = ripemd160(sha256(P))*
3. *checksum = sha256(sha256(version + key))*[:4]
4. *res = version + key + checksum*

Return encode_base58(res)

Direcciones de BitCoin

¿Cómo se ve esto?

```
Private key: 0x69b0392170b2b4809788bd619997ed88371ddd6d6e3fa0524d1eb49325498936
Testnet address: n1VietsybSPTN3wuAWuXvUDtTc7D4GGVDj
Mainnet address: 1LymMqnznQxCawUHSwwA6Z1ZbcWW4Nxb1a
```

Llaves de BitCoin

Una nota sobre base58

```python
# the alhabet we use
BASE58_ALPHABET = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'

# compute base58 encoding of s
def encode_base58(s):
    count = 0
    for c in s:  # count the number of zeros at the front
        if c == 0:
            count += 1
        else:
            break
    num = int.from_bytes(s, 'big')
    prefix = '1' * count
    result = ''
    while num > 0:  # figure out which symbol to use
        num, mod = divmod(num, 58)
        result = BASE58_ALPHABET[mod] + result
    return prefix + result  # prepend the zeros that conversion deleted
```

Llaves de BitCoin

Base58 se va

Viene Bench32 (ya está en segwit – BIP 0173)

Referencias

Lectura:

- Jimmy Song, Programming Bitcoin, chapters 1—4
- https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/

Todos los imágenes recuperados en:
https://github.com/jimmysong/programmingbitcoin/blob/master
https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc