¿Cómo funciona Bitcoin?

Parte 1: Criptografía de Curva Elíptica

Bitcoin es una moneda digital que basa su funcionamiento en distintas herramientas criptográficas. Para asegurar la propiedad de un usuario y que éste es el único titular y poseedor de alguna cantidad de Bitcoin se recurre a la criptografía asimétrica o de llave pública.

Se puede encontrar en Internet noticias que hablan sobre el hackeo y posterior robo de wallets (tanto cold /hot wallets) en bitcoin y lo inseguro que resultan manejar estos.
Todas estas notas que pueden o no ser ciertas, no son sobre hackeos sobre la red ni sobre el protocolo de Bitcoin.

Todos estos hackeos se deben a brechas en ciberseguridad (un teléfono o computadora con malware), ingeniería social (Pishing) o estafas (hardware wallets maliciosas o intermediarios no confiables) que obtienen la clave privada de la víctima para luego mover esos fondos.

En ningún caso reportado se logró romper el algoritmo de la firma digital. Pues para lograrlo no existen herramientas matemáticas más que la fuerza bruta y es computacionalmente inviable. Por esta razón es muy importante la selección de la clave privada y evitar que sea replicable fácilmente (el equivalente a usar 1234 como pin de un celular).

Este tipo de algoritmos y herramientas criptográficas son ampliamente conocidas y estudiadas. Se aplican en distintas tecnologías de uso cotidiano y dependiendo de las características necesarias (como un alto grado de encriptado para un uso militar, por ejemplo) se recurren a distintas herramientas matemáticas. Un estándar usado en distintas industrias desde finanzas hasta almacenamiento es el uso de RSA (que aplica la factorización de números enteros).

Bitcoin usa una alternativa a RSA. El algoritmo de firma digital mediante curva elíptica. El problema que no tiene solución analítica se conoce como logaritmo discreto. Al igual que la factorización de números enteros en RSA su resolución implica usar fuerza bruta.
La Curva Elíptica muestra ser más eficiente en los tamaños de clave e

igualmente seguro que RSA. Esta cualidad (el tamaño de la clave) es crítica para la firma masiva de transacciones pues mientras más espacio en memoria ocupe, el blockchain necesitará mas cantidad de memoria en disco para almacenarse.

Bitcoin utiliza un protocolo que esta definido y estandarizado por el SEC (standards for efficient cryptography): Secp256k1.

Esta define la clave con un tamaño único de 160 bits. Por poner otro ejemplo para comparar los autos eléctricos de Tesla basan su seguridad en un algoritmo de curva elíptica (distinto a Bitcoin) Curve25519 con una clave de 256 bits. Ese es el estándar de tamaño de clave para la seguridad de hoy.

> **Note**
>
> La base matemática que usaremos para explicar y entender como funciona la criptografía de curva elíptica:

```
- Expresiones algebraicas en geometría de curvas planas.

- Derivadas para cálculo de máximos/mínimos para realizar análisis.

- Cuerpos fínitos para la representación del sistema.
```

Copiar

Curva Elíptica

La siguiente ecuación representa una generalización de la curva elíptica:

$$y^2 = x^3 + Ax + B$$

Es la ecuación de Weiestrass.

Proposición

Se busca definir el uso de la Curva Elíptica como un grupo Abeliano, simplificando:
Un grupo Abeliano cumple las operaciones de SUMA y MULTIPLICACION y sus propiedades como la conmutatividad:

$$A + B = B + A$$

$$A * B = B * A$$

Para lo cual es una condición la existencia de un elemento neutro

$$a + 0 = a$$

Se busca demostrar que las operaciones sobre la curva elíptica cumplen con estas dos: conmutatividad y elemento neutro.

Consideramos los puntos P y Q que satisfacen la ecuación de la curva elíptica:

$P = (x_1, y_1)$ y $Q = (x_2, y_2)$

Si trazamos una recta entre ambos puntos P y Q obtenemos un 3er punto F que es la intersección de esta recta con la curva elíptica.

$$F = (x_3, y_3)$$

Este punto F (intersección recta y elipse) se define como la suma de los puntos P y Q (que están en la recta):

$$P + Q = F$$

Sus coordenadas $F = (x_3, y_3)$ se pueden obtener de dos maneras: o resolviendo las ecuaciones para encontrar el punto en que se cruzan o usando las relaciones matemáticas siguientes.

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = (x_1 - x_3) * \lambda - y_1$$

Donde λ es un factor que depende de los puntos P y Q.

- Si P y Q son distintos puntos $x_1 \neq x_2$

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

- Si P y Q son el mismo punto $x_1 = x_2$ & $y_1 \neq 0$

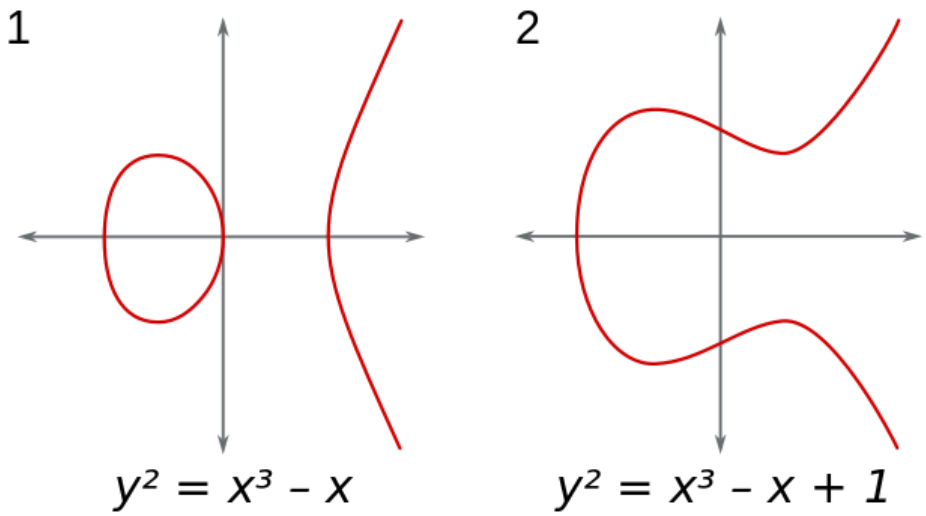
$$\lambda = \frac{3 * x_1^2 + a}{2y_1}$$

Con estas fórmulas se puede calcular el punto $F(x_3, y_3)$ sin la necesidad de resolver la intersección de ecuaciones.

La simetría de puntos para la curva elíptica se da en el eje x:

$$P = (x_1, y_1)$$

$$-P = (x_1, -y_1)$$



Demostración de la Suma

Demostraremos que las ecuaciones son válidas para calcular el punto F.

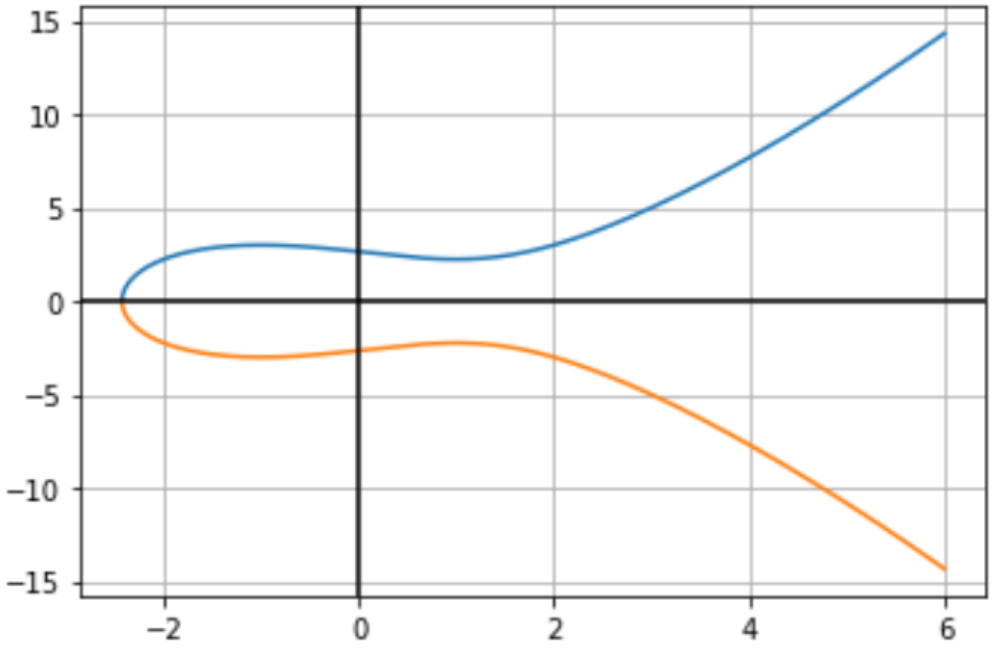
Caso 1.

Consideremos la siguiente curva, un caso particular de la ecuación General de Weiestrass:

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

Graficamos la curva usando Python.

```
import numpy as np
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
plt.plot(x,y,x,-1*y)
plt.grid()
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



El primer paso de análisis es encontrar el dominio de la variable independiente x. Esto es para asegurarnos que las coordenadas de los puntos P y Q que se elijan pertenezcan a la curva.

> **Note**
>
> Bitcoin usa el conjunto de números enteros para evaluar los cálculos. No existen números con decimales en sus funciones. Son todos enteros por qué un computador para representar un valor con decimales recurre a una notación de punto flotante.
>
> Redondeando el número a cierta cantidad de decimales (no se puede representar infinitos decimales en un computador) se genera un error y este se propaga al hacer operaciones como la multiplicación o suma.

Para encontrar el dominio de 'x' despejamos la variable 'y'.

$$y = \pm\sqrt{x^3 - 3x + 7}$$

Los valores dentro del radical deben ser mayor/igual a cero.

$$x^3 - 3x + 7 \geq 0$$

Para encontrar sus raíces rápidamente recurrimos a Python

```python
import numpy as np
```

```
coeff = [1, 0, -3, 7]
print(np.roots(coeff))
```

```
[-2.42598876+0.j 1.21299438+1.18914511j 1.21299438-1.18914511j]
```

Las raíces que obtenemos rápidamente con numpy son del conjunto de los números complejos. La primera raíz -2.42 no tiene parte imaginaria por lo que la podemos tomar como puramente real.
Las otras dos raíces son conjugadas, haciendo un poco de álgebra podemos llegar a la siguiente expresión.

$$(x + 2.42)(x^2 - 2.43x + 2.89) \geq 0$$

Solo para este ejemplo usamos números redondeados. Pues escogimos la curva de la Elipse al azar.

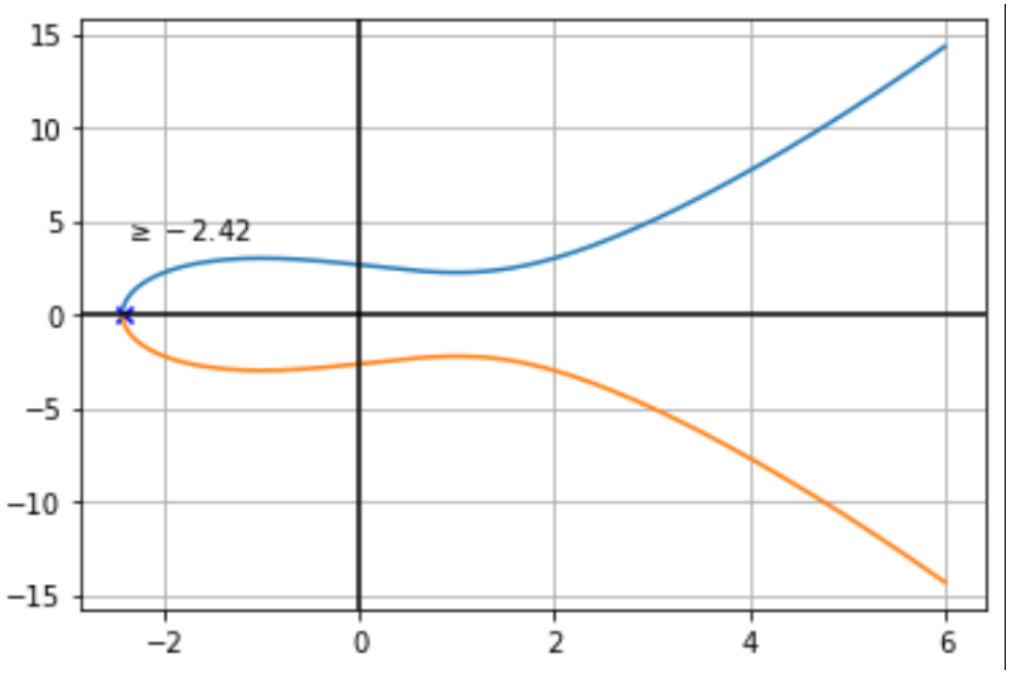
Resolviendo el sistema de inecuaciones llegamos a la siguiente condición.

$$x \geq -2.42$$

Si graficamos el punto.

PYTHON Copiar

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)

plt.plot(x,y,x,-1*y)
plt.grid()

plt.scatter([-2.42],[0],color='blue',marker='x')
plt.text(-2.42,4,r'$\geq-2.42$')
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



Al graficar la curva y el punto encontrado antes es evidente que no existe puntos con un 'x' menor a -2.42 que pertenezcan a la curva.

De esta manera podemos seleccionar los puntos P y Q y realizar el cálculo para encontrar su suma $F = P + Q$

Para P, tomamos un valor 'x=-2':

$$P(x, y) = -2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-2)^3 - 3 * (-2) + 7 = -8 + 6 + 7 = 5$$

$$y = \sqrt{5}$$

Quedando el punto $P(-2, \sqrt{5})$

Para otro punto Q, tomamos un valor de 'x=-1':

$$Q(x, y) = -1, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

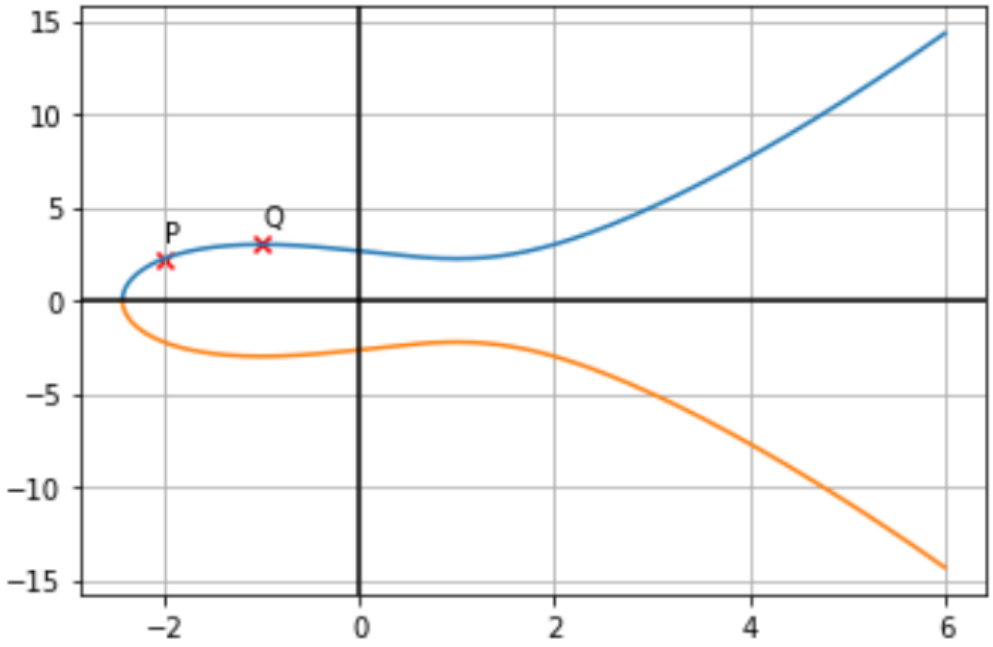
$$y^2 = (-1)^3 - 3 * (-1) + 7 = -1 + 3 + 7 = 9$$

$$y = 3$$

Quedando $Q(-1, 3)$

Graficamos los puntos P y Q sobre la curva elíptica.

```python
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
plt.plot(x,y,x,-1*y)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')

plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



El punto F es la intersección de la curva con una recta entre los puntos P y Q.

Para encontrar la recta entre dos puntos recordamos esta relación de la geometría analítica:

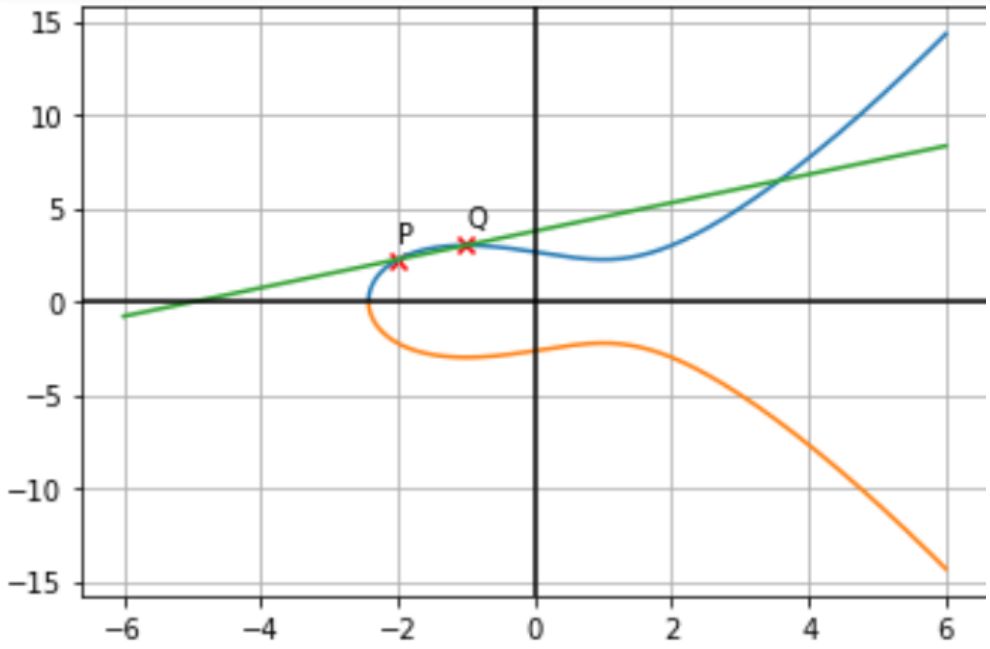
$$y - y_2 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_2)$$

Reemplazando los valores llegamos a la recta:

$$y = (0.76)x + 3.76$$

Si graficamos esta recta encontraremos la intersección de ambas curvas.

```python
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)

#recta
y2=0.76*x+3.76
plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')
plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')

plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



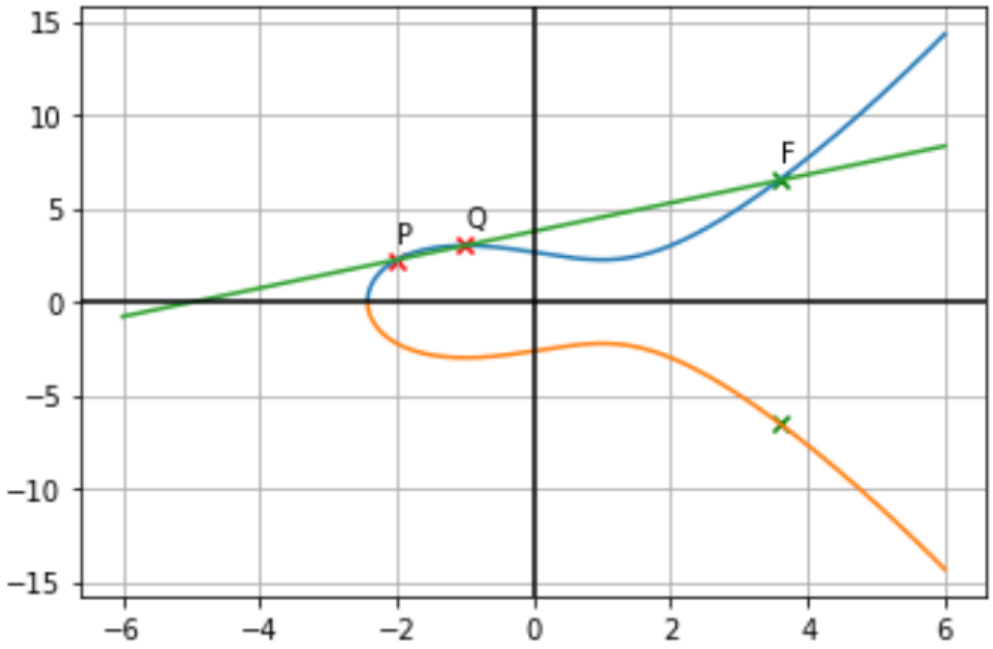
El punto F se puede calcular encontrando las raíces del sistema de ecuaciones de la recta y curva. Se puede usar Python para calcularlo, sin embargo tiene un alto costo computacional.

```python
import sympy as sym
x=sym.Symbol('x')
y=sym.Symbol('y')

resp=sym.solve([0.76*x+3.76-y,-3*x+x*x*x+7-y*y], dict=True)
print(resp)
```

[{x: -1.99684273373310, y: 2.24239952236285}, {x: -1.00000000000000, y: 3.00000000000000}, {x: 3.57444273373310, y: 6.47657647763715}]

También se pueden obtener si usamos las relaciones con λ del punto Proposición. Se obtienen con pocas operaciones.

```python
x1,y1=-2,np.sqrt(5)
x2,y2=-1,3

alpha=(y2-y1)/(x2-x1)


x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1

x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)

#recta
y2=0.76*x+3.76
plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')

plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')
plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.scatter([x3,x3],[y3,-y3],color='green',marker='x')
plt.text(x3,-1*y3+1,'F');
print(x3,-y3)
```



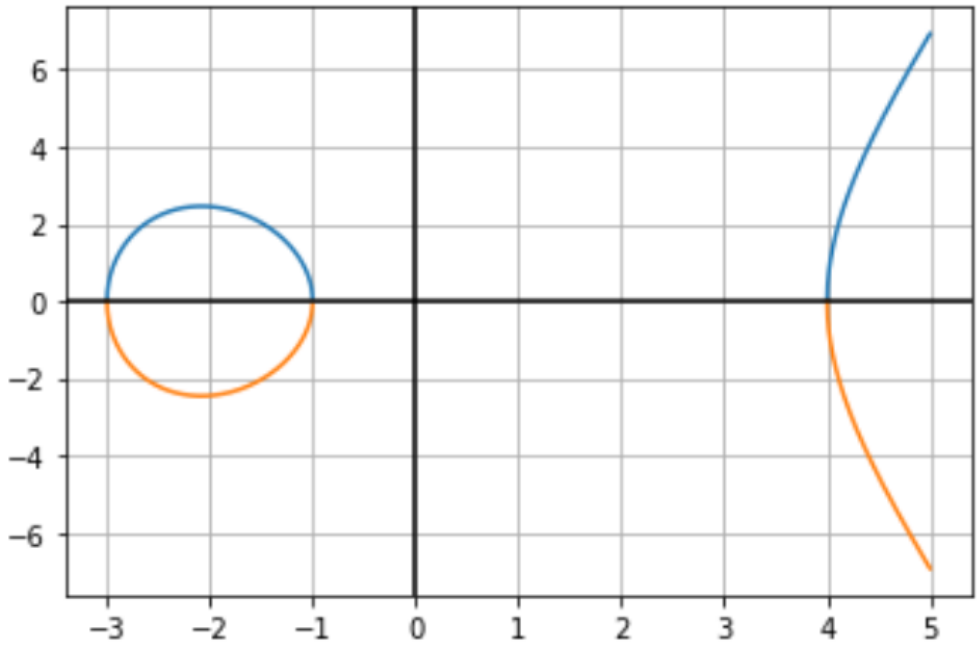
En los ejemplos las coordenadas encontradas para F son cercanas, no son precisamente iguales. Esto por el error de redondeo.

Caso 2.

Otra variante de la ecuación general de Weiestrass es:

$$y^2 = x^3 - 13x - 12$$

```python
x = np.arange(-5,5,0.001)
y = np.sqrt(x*x*x-13*x-12)
plt.plot(x,y,x,-1*y)
plt.grid()

plt.axhline(0,color='black') &amp;nbsp;#horizontal line
plt.axvline(0,color='black'); &amp;nbsp;#horizontal line
```



Para este caso el análisis se realiza de la misma manera.
Si despejamos la variable 'y' tenemos:

$$y = \pm\sqrt{x^3 - 13x - 12}$$

Los valores dentro del radical deben ser mayor/igual a cero.

$$x^3 - 13x - 12 \geq 0$$

```python
import numpy as np

coeff = [1, 0, -13, -12]
print(np.roots(coeff))
```

```
[ 4. -3. -1.]
```

Factorizando tenemos:

$$(x-4)(x+3)(x+1) \geq 0$$

Con las condiciones:

$x \geq 4, x \geq -3$ y $x \geq -1$

que se pueden expresar de la siguiente forma:

$D_x = [-3, -1] \cup [4, \infty[$

Los puntos P y Q se escogen de manera que se encuentren en la región izquierda.

$$P(x,y) = -5/2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-5/2)^3 - 13 * (-5/2) - 12 = 39/8$$

$$y = \sqrt{39/8}$$

Quedando $P(-5/2, \sqrt{39/8})$

Para Q:

$$Q(x,y) = -3/2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-3/2)^3 - 13 * (-3/2) - 12 = 33/8$$

$$y = \sqrt{33/8}$$

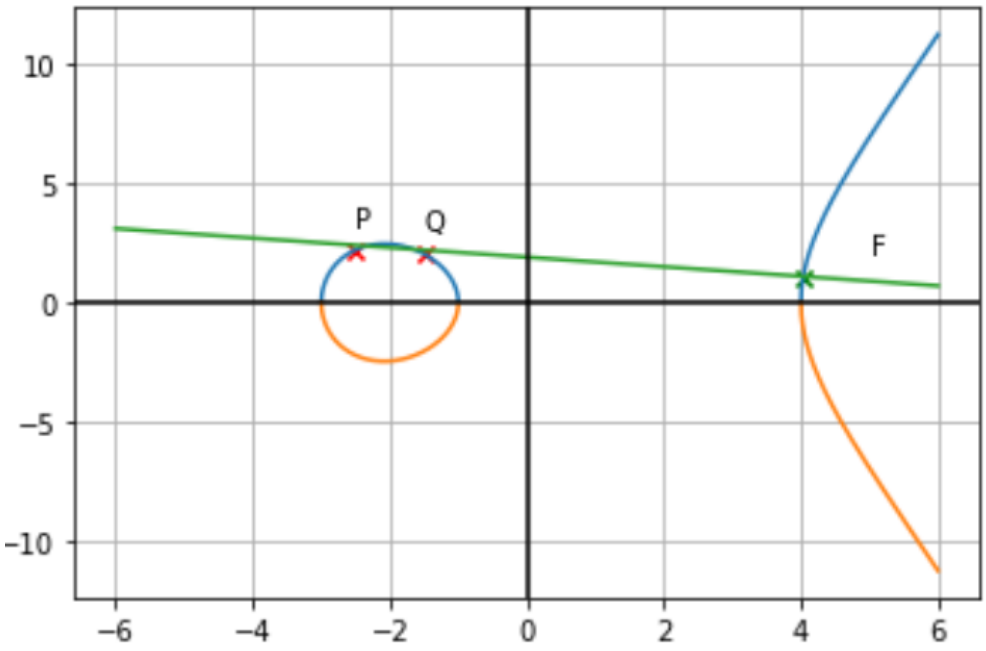
Quedando $Q(-3/2, \sqrt{33/8})$

La recta que pasa por P y Q quedaría:

$$y = -0.2 * x + 1.9$$

```python
x1,y1=-5/2,np.sqrt(39/8)
x2,y2=-3/2,np.sqrt(33/8)

alpha=(y2-y1)/(x2-x1)

x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1

x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-13*x-12)
y2 = -0.2*x + 1.9
plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()

plt.scatter([-5/2,-3/2],
[np.sqrt(39/8),np.sqrt(33/8)],color='red',marker='x')

plt.text(-5/2,np.sqrt(39/8)+1,'P')
plt.text(-3/2,np.sqrt(33/8)+1,'Q')
plt.axhline(0,color='black')
```

```
plt.axvline(0,color='black')
plt.scatter([x3],[-y3],color='green',marker='x')
plt.text(x3+1,-y3+1,'F');
print(x3,-y3)
```



Caso 3

En los dos anteriores casos demostramos que la suma de dos distintos puntos P y Q en la curva elíptica cumplen las relaciones para encontrar x_3, y_3.

Este caso se muestra para demostrar que puede darse la suma de un mismo punto (es decir que P y Q son iguales): $P + P = 2 * P$

La ecuación que teníamos para encontrar la recta que pasa por dos puntos en este caso no es útil. Para encontrar la recta tangente al punto P derivamos la ecuación de la curva elíptica:

$$y^2 = x^3 - 13x - 12$$

despejando 'y'

$$y = \pm\sqrt{x^3 - 13x - 12} = \pm(x^3 - 13x - 12)^{\frac{1}{2}}$$

$$dy = \frac{1}{2}(x^3 - 13x - 12)^{-\frac{3}{2}}(3x^2 - 13)dx$$

Igualamos la derivada de la función de cero para encontrar un máximo.

$$y' = \frac{1}{2}(x^3 - 13x - 12)^{-\frac{3}{2}}(3x^2 - 13) = 0$$

$$3x^2 - 13 = 0$$

$$x = \pm\sqrt{\frac{13}{3}} = -2.08$$

El resultado que buscamos se encuentra en la región izquierda de la curva, por lo cual el punto x con signo negativo es el valor a usarse.

Evaluamos en la función:

$$y^2 = x^3 - 13x - 12$$

$$y^2 = (-2.08)^3 - 13(-2.08) - 12 = -9.02 + 27.06 - 12 = 6.04$$

$$y = \sqrt{6.04} = 2.45$$

Finalmente el punto P:

$$P(x, y) = -2.08, 2.45$$

Para encontrar la recta tangente recurrimos a la siguiente forma:

$$y = mx + b$$

Donde m es la pendiente y representa a la derivada de la curva elíptica evaluada en el punto P.

$$y\prime = m = \frac{1}{2}((-2.08)^3 - 13(-2.02) - 12)^{-\frac{3}{2}}(3(-2.08)^2 - 13)$$

$$m = \frac{1}{2}(24)^{-\frac{3}{2}}(-0.021)$$

$$m = -0.0021$$

Reemplazando en la ecuación de la recta tangente:

$$2.45 = (-0.0021)(-2.08) + b$$

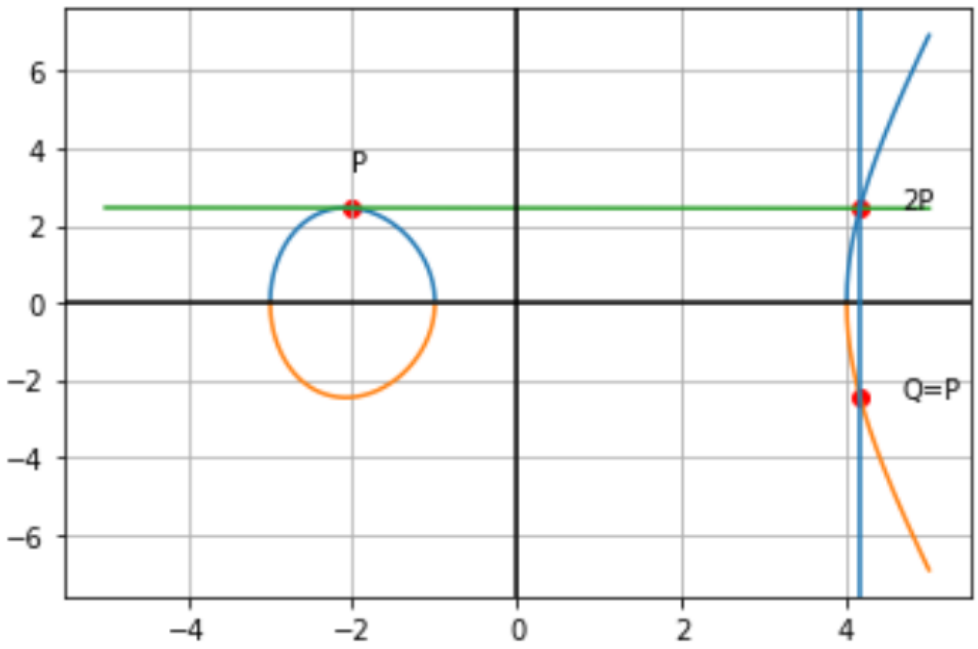
$$b = 2.45$$

Finalmente:

$$y = (-0.0021)x + 2.45$$

```python
x = np.arange(-5,5,0.001)
y = np.sqrt(x*x*x-13*x-12)

y_2 = (-0.0021)*x+2.45
x1,y1=-2.08,2.45
x2,y2=x1,y1
```

```
alpha = (3*x1*x1-13)/(2*y1)
x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1

plt.plot(x,y,x,-1*y)
plt.plot(x,y_2)
plt.grid()
plt.axhline(0,color='black')  #horizontal line
plt.axvline(0,color='black')  #horizontal line
plt.scatter([-2.02,x3,x3],[2.45,y3,-y3],color='red')
plt.text(-2.02,2.45+1,'P')
plt.text(x3+0.51,y3,'Q=P')
plt.text(x3+0.51,-y3,'2P')
plt.axvline(x3,ymin=-y3,ymax=y3);  #horizontal line
print(x3,y3)
```

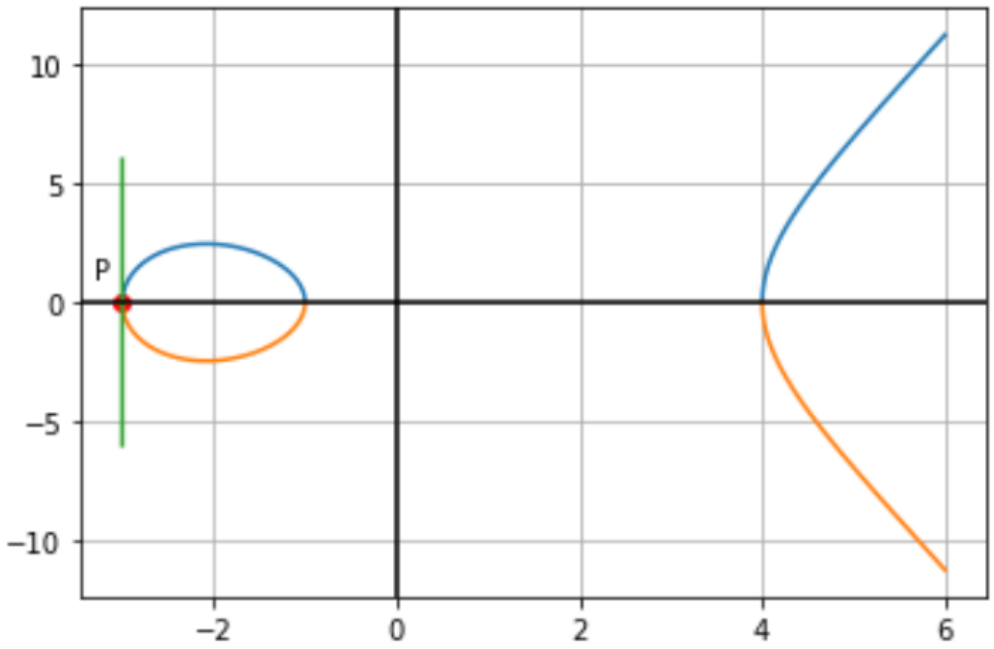


Elemento neutro en la suma.

Si la pendiente de la recta no intersecta a la curva elíptica es porque tiene una pendiente de 90 grados (m=0). Es el equivalente a sumar cero pues no cambia la expresión.

```
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-13*x-12)

y2 = -3*np.ones(len(x))
plt.plot(x,y,x,-1*y)
```

```
plt.plot(y2,x)
plt.grid()
plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.scatter([-3],[0],color='red');
plt.text(-3-0.3,1,'P')
```



$$P(x1, y1) + 0 = P(x1, y1)$$

Finalmente el elemento neutro junto con la forma de operar la suma en la curva elíptica queda demostrado.

Multiplicación

Al demostrar la suma de un mismo punto, de cierta forma se demuestra la multiplicación por dos: $P + P = 2 * P$. Se puede extender el concepto para un punto P y un número entero natural k.

$$k * P = P + \underbrace{\ldots}_{k_{veces}} + P$$

Se define la multiplicación como una suma repetida.

Cuerpos Finitos

Un ordenador no puede trabajar con infinita cantidad de números. Usando la matemática modular se puede representar un conjunto

infinito a partir de otro finito.
Para mostrar un ejemplo entendible hablemos de las horas del día.

En un reloj encontramos un conjunto finito de números «0,1,2,..,22 y 23». Se puede representar cualquier número de horas, por decir, 110 hrs. usando el siguiente concepto:

¿Si partimos de 0 hrs que hora marca un reloj transcurridas 110 hrs?

Por intuición se llega a la conclusión: $110hrs = 4 * 24hrs + 14hrs$ Es decir que transcurrieron 4 días (dando una vuelta completa a todos los números del 0 al 23) y sobran hasta las 14 hrs.

Una forma de expresarlo es mediante matemática modular:

$14 = 110 mod 24$

De esta forma ya se puede expresar cualquier número de horas con el conjunto de 24 cifras de un reloj.

De la misma forma Bitcoin usa criptografía sobre cuerpos finitos (y no sobre los números reales o enteros) para representar los números muy grandes.

Los parámetros de Bitcoin

Bitcoin hace el uso de una curva especial que viene definida en el documento normativo Standards for Efficient Cryptography (SEC) (Certicom Research:

http://www.secg.org/collateral/sec2_final.pdf)

Los valores de las constantes A, B como los rangos de números del cuerpo finito y otras constantes se eligieron matemáticamente, por esta razón las curvas se denominaron secp256k1 y usan la k por 'Koblitz'. Otra familia muy usada es secp256r1 con la r por 'Random' pues se eligieron sus constantes al azar.

Para Bitcoin la ecuación Weiestrass tiene $A = 0$ y $B = 7$:

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

Estos números se expresan en Hexadecimal:

a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007

Con un número base generador:
G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

El cuerpo finito es un conjunto de números enteros, en el caso de Bitcoin el límite es un número primo muy grande.

$$P = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Este número se expresa en hexadecimal.

P = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

Si transformamos al sistema decimal corresponde al número:

P = 115792089237316195423570985008687907853269984665640564039457584007908834671663

Los números P y G son únicos y comunes en el protocolo. Se escogieron para que se cumpla la relación modular de manera eficiente.

Orden n

Otro valor constante que se especifica en el estándar 'Secp256k1' es el orden 'n'. Como vimos la e cuación general de la curva elíptica $y^2 = x^3 + Ax + B$ tiene un Dominio y Rango. Los parámetros de la curva que usa Bitcoin $y^2 = x^3 - 7$ se escogieron de manera que se tenga un amplio rango de puntos en la curva. El valor 'n' representa el máximo valor de puntos que contiene la curva.

n = 115792089237316195423570985008687907852837564279074904382605163141518161494337

En HEX.

n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

Los puntos en la curva pueden tener 'n' valores, dentro del rango $[1, n-1]$.

El estándar secp256k1 detalla que la clave privada debe ser un número entero de 256 bits. La clave puede tomar valores tan grandes hasta $2^{256}-1$. Siendo precisos tenemos que puede tomar n valores, con un $n < 2^{256}-1$.
La diferencia es mínima y se debe a que no todos los puntos del campo finito pertenecen a la curva por lo que se debe verificar que una clave generada aleatoriamente sea menor a $n-1$.

El orden $n = 1,1579... \times 10^{77}$ es un número extremamente grande de claves privadas. Un calculo usando física muestra que todo el universo que podemos observar (incluyéndonos) tiene ~10^{82} átomos. Es decir que el límite de usuarios de Bitcoin es casi del tamaño del universo.

Para entender que tan seguro es el sistema ponemos números:

- La computadora más poderosa hasta 2022 era Frontier. Realiza 1×10^{18} operaciones por segundo. Supongamos que todos en el mundo (10 mil millones de humanos) contamos con esta computadora por igual, sumarían 1×10^{10} computadoras en paralelo haciendo un total (si todos nos dedicamos a lo mismo) de $1 \times 10^{28} \frac{direcciones}{segundo}$ (suponiendo que demore un solo proceso para hallar la solución).
- El tiempo que se demoraría en verificar la totalidad de direcciones se calcula
  $1 \times 10^{77} direcciones * \frac{1 segundo}{1 \times 10^{28} direcciones} = 1 \times 10^{49} segundos * \frac{1 año}{3.154 \times 10^{7} seg} = 3.15 \times 10^{43} años$
- La edad de la tierra son $4 \times 10^{9} años$ y la del universo $13.6 \times 10^{9} años$. Es virtualmente imposible romper Bitcoin por Fuerza Bruta.

Parte 2: Generando claves Pública/Privada

El sistema criptográfico se basa en dos claves: una pública y otra privada. Que se relacionan mediante la multiplicación sobre curva elíptica.

$$K = k * G$$

K mayúscula es la clave pública
k minúscula es la clave privada

G el punto generador

Para generar una clave pública K primero se necesita conseguir una clave privada 'k' (minúscula). Luego se multiplica escalarmente con el punto Generador G (que es constante) para producir otro punto 'K' (mayúscula) que pertenece a la curva.

> **Note**
>
> El punto generador es un parámetro de la curva elíptica secp256k1. Representa un punto $G(x, y)$ en la curva.
> Se puede expresar de dos maneras:
>
> - Sin comprimir.
> G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
> - Con compresión. Si ambos puntos pertenecen a la curva, si se despeja la variable 'y' se puede computar a partir de 'x': $y^2 mod P = (x^3 - 7) mod P$. De modo que el valor de G se expresa:
> G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
>   Mostrando solo un valor de la coordenada 'x' e indicandolo si 'y' es par o impar con el primer byte '02' o '03' respectivamente.

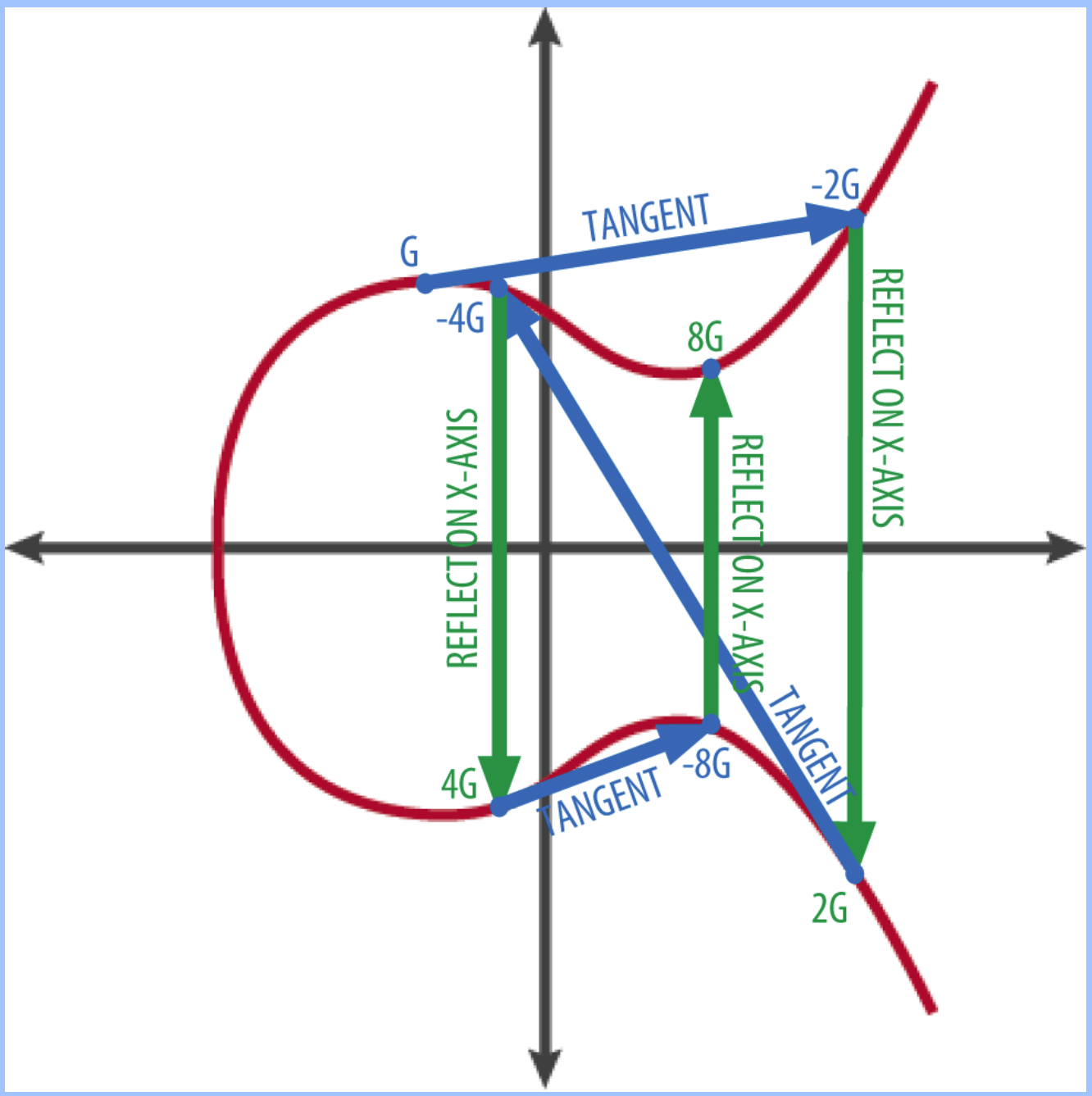
Como hemos visto la multiplicación en curva elíptica esta definida como la suma de G veces el valor de 'k'. Esta operación que demostramos en la parte 1 esta definida en un solo sentido. No existe formula para la división sobre curva elíptica. Y resolver este problema (encontrar k a partir de K y G) se conoce como el problema del logaritmo discreto. No existe forma de resolverlo, ni analítica ni computacionalmente que no sea usando fuerza bruta (y como vimos es inútil).

El número de 'k' de la clave privada produce otro número K. Este también se puede expresar de manera comprimida y no comprimida. Prefiriendo la forma comprimida de manera mas amplia.

Note

En la primera parte se demostró como funciona la suma y mutiplicación sobre curvas elípticas. Para encontrar la clave Pública en Bitcoin $K = k * G$, G representa a un punto $P(x, y)$ y 'k' es un número escalar (que debe ser aleatorio) de 256 bits y menor al orden n.

Para evaluar este producto internamente Bitcoin usa un algoritmo muy eficiente: doblado y suma (double and add). Multiplicar el punto generador G por el valor de un escalar 'k' se puede visualizar en la siguiente gráfica. Amplía el mismo método de encontrar la intersección de una recta con la curva.



El conjunto de números que puede tomar la clave privada es [1,n-1]. Si se toman los primeros valores $(1,2,\dots)$ que son triviales cualquier persona puede acceder a estas. Es por esta razón que es *crítico* elegir un número que no se vuelva a repetir. La incertidumbre se tenga para generar estos números se llama Entropía. Mientras mayor sea la fuente de entropía para encontrar la clave privada será mucho más difícil de adivinar.

Direcciones Bitcoin

Para obtener una Dirección Bitcoin y recibir un pago se debe transformar la clave pública (HEX) obtenida a un formato que se pueda compartir entre humanos.

Este número tiene el siguiente procedimiento para convertirse en una dirección Bitcoin.
1. Se obtiene un Hash SHA256 a la clave pública de 256 bits (expresada en HEX).
Esto se realiza por dos razones: enmascarar la clave pública y tener su firma digital.
2. Se obtiene un segundo Hash RIPEMD160.
Se realiza como una medida extra de seguridad y para reducir el tamaño de la firma digital a 160 bits.
3. La salida es una cadena de texto con tamaño constante de 160 bits o 20 bytes (8 bits = 1 byte).
4. Esta salida usa un encode llamado Base58Check.
La razón de esta es expresar la dirección de Bitcoin en letras y con una revisión de errores.

El encode se refiere a como se van a expresar esos 20 bits. Si usamos un sistema decimal va de 0 a 9. El sistema HEX (hexadecimal) usa 0-9 y A-F (16 valores). Lo mas lógico sería usar una forma limitada de ASCII.
Base58 es una codificación que usa:
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Base58Check es una forma de evitar la mala manipulación al copiar direcciones Bitcoin.

1. Hace un doble SHA256 y copia los primeros 4 bytes al final para que se constate que es correcto.
2. Suma un byte al inicio para identificar que tipo de dirección es.

Tipos de direcciones – Prefijo Base58

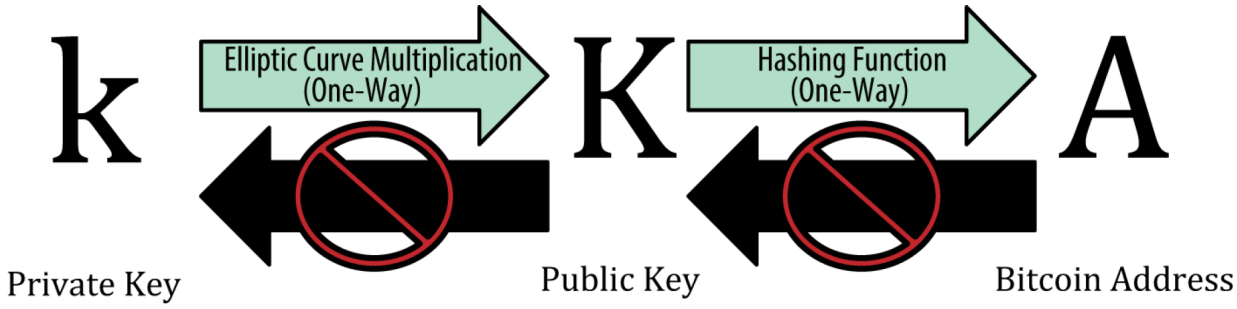
- Dirección normal: 1
- Hash-to-Script: 3
- Testnet: m o n

Y finalmente se añade un checksum a la base58 para evitar errores de copia.

Al final se tiene una dirección WIF (comprimida) que es el estándar

> 16y8bMjqVXgWjT8LhskyZHHHpFcse2ZNqc

No existe forma de encontrar la clave privada partiendo del Address, ni de la clave pública.



Aplicando Python (ejemplos)

Para este ejemplo usamos una librería especializada en la manipulación criptográfica.

```
import cryptos


#generamos una clave privada
valid_private_key = False
while not valid_private_key:
#El método que se usa para la generación de un número randomico internamente
#usa RSA, tecnicas mas apropiadas para cryptografia que los paquetes random
normales
```

```
        private_key = cryptos.random_key()
        decoded_private_key = cryptos.decode_privkey(private_key, 'hex')
        valid_private_key =  0 < decoded_private_key < cryptos.N


print('El número Clave Privada HEX: '+private_key)
print('El número Clave Privada DEC: '+str(decoded_private_key))

wif_encoded_private_key = cryptos.encode_privkey(valid_private_key, 'wif')

print("Private Key (WIF): ", wif_encoded_private_key)

compressed_private_key = wif_encoded_private_key + '01'

print("Private Key Compressed (hex) is: ", compressed_private_key)

wif_compressed_private_key =
cryptos.encode_privkey(cryptos.decode_privkey(compressed_private_key,
'hex'), 'wif')

print("Private Key (WIF-Compressed) is: ", wif_compressed_private_key)

public_key = cryptos.fast_multiply(cryptos.G, decoded_private_key)

(public_key_x, public_key_y) = public_key

if (public_key_y % 2) == 0:
    compressed_prefix = '02'
else:
    compressed_prefix = '03'

hex_compressed_public_key = compressed_prefix + cryptos.encode(public_key_x,
16)

print("Compressed Public Key (hex) is:", hex_compressed_public_key)

# Generate bitcoin address from public key

print("Bitcoin Address (b58check) is:",
cryptos.pubkey_to_address(public_key))

# Generate compressed bitcoin address from compressed public key

print("Compressed Bitcoin Address (b58check) is:",
cryptos.pubkey_to_address(hex_compressed_public_key.encode()))
```

```
El número Clave Privada HEX:
9dc22e88e66d4186b3f0198d2ddcba897472083098d5bf353690dbdf666f5d80 El número
Clave Privada DEC:
71356206729291301012444016341937156419989089833072981744418296592959320055168 Private Key (WIF): 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
Private Key Compressed (hex) is:
5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf01 Private Key (WIF-Compressed) is: 5HpHagT65Xw47KFbBT5CLHdr6kpbWGhTRVobx8uuURJB96mYAtA
Compressed Public Key (hex) is:
025d77e664a368715af5008ee7a6e5c08a9cc1ddabcdbb99bc9ebdfbf6ba83a7c0 Bitcoin
Address (b58check) is: 13edqfDceGxV5HKa8wPZ2a8iX3KyNHXj27 Compressed Bitcoin
Address (b58check) is: 16y8bMjqVXgWjT8LhskyZHHHpFcse2ZNqc
```

Estas direcciones son únicas en Bitcoin. Se puede buscar (por prueba y error) y obtener direcciones que tengan algún carácter en particular.

Trampeando Direcciones

Supongamos que tenemos la dirección del ejemplo anterior.

> 16y8bMjqVXgWjT8LhskyZHHHpFcse2ZNqc

Algún atacante buscará reemplazar la dirección para robar un pago. Pero si reemplaza otra dirección muy distinta el cambio sería evidente. Por lo que pueden buscar generar una dirección que tenga algunas cifras parecidas, pues mucho usuarios solo verifican pocas letras de las direcciones para hacer el envío pues es engorroso chequear todas las letras para un humano, peor si hay Qr de por medio. Supongamos que el atacante considera que será suficiente que empiecen igual para que alguien lo notara.

> 16y8xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Se puede hacer un script de Python que lo haga.

```python
import cryptos

#generamos una clave privada

valid_private_key = False


patron = '16y8'
while not valid_private_key:
```

```
    #El método que se usa para la generación de un número randomico
internamente
    #usa RSA, tecnicas mas apropiadas para cryptografia que los paquetes
random normales
    private_key = cryptos.random_key()
    decoded_private_key = cryptos.decode_privkey(private_key, 'hex')
    valid_private_key =  0 < decoded_private_key < cryptos.N
    public_key = cryptos.fast_multiply(cryptos.G, decoded_private_key)
    (public_key_x, public_key_y) = public_key
    if (public_key_y % 2) == 0:
        compressed_prefix = '02'
    else:
        compressed_prefix = '03'
    hex_compressed_public_key = compressed_prefix +
cryptos.encode(public_key_x, 16)
    BTC_dir = cryptos.pubkey_to_address(hex_compressed_public_key.encode())
    if(BTC_dir[:len(patron)]==patron):
        print(str(decoded_private_key))
        valid_private_key = True
    else:
        valid_private_key = False
```

Arrojando en el resultado en mi laptop en 58.6 segundos.

Clave privada:

- d274aa016e072ee5bbc75b91f0e2233cf139d6d2babd714c2cd817bc8fefebf3
- Direccion Fake: 16y8ZMrSVbLNh13N9GXpiEJLEYmzmvcXQZ

que comparado con el original

- 16y8bMjqVXgWjT8LhskyZHHHpFcse2ZNqc

Pueden llevar a fraudes por reemplazo.

Direcciones de Vanidad

Como vimos una Dirección de Bitcoin será la combinación de letras minúsculas/mayúsculas y números. Buscando por prueba y error se pueden conseguir direcciones que tengan algún patrón especial.

Por ejemplo, encontremos una dirección que empiece por: (juampi->jpi)

- 1jpixxxxxxxxxxxxxxxxxxxxxx

Con el script anterior se encontró en 7 min 19.9 segundos.

- Clave privada: 191116501fa4097346c84792fa749e17c56623094ff817bfff20fd3ec5d49b0b
- Dirección Vanidad: 1jpiPHuNiXRkE7MRiS7p6cB7XB8T9N1cY

Las direcciones de vanidad permiten identificar y reducir falsificaciones además de dar ser una exclusividad (de ahí el nombre de vanidad). Es un arma de doble filo pues con suficiente poder computacional puede convertirse en objetivo de falsificaciones. Mientras más letras se requiera como condición más difícil es encontrar su clave privada correspondiente.

Estas direcciones de Vanidad se pueden obtener solamente con prueba y error por lo que generarlos puede significar un trabajo similar a la minería en cuanto el consumo intensivo de recursos computacionales (y de energía).

Parte Final: Recomendaciones.

Como vimos Bitcoin es un sistema muy seguro que se basa en código y matemáticas.

Quita el factor de CONFIANZA en que una transacción será válida y aumenta otro factor: el VERIFICAR si la transacción esta en un bloque.

El poseedor de la Clave Privada es el único que puede afirmar que tiene control de una dirección de Bitcoin. Por lo que se pueden perder bitcoins en caso de:

1. Enviar una transacción a una dirección desconocida o errónea.
2. Perder la clave privada.

Por lo cuál este aspecto es fundamental para tener en cuenta.

La clave privada se puede robar. Por esto no debe compartirse ni filtrarse por Internet. Como una computadora puede ser hackeada existen formas de guardar una clave privada más allá de tener un archivo .txt:

- Usar una aplicación. Este método es peligroso porque se debe confiar que el programador no es malicioso.
- Almacenarlo cifrado.

- Almacenarlo en una computadora que nunca se conecta a Internet (solo se podría robar con un hackeo físico).
- Usar hardware especializado. Existen chips que no son simplemente memorias, sino un circuito de seguridad que no se puede extraer ni físicamente.

Un método que es muy usado por su costo vs. beneficio es un monedero ninja safu.

Consiste en generar una clave privada usando un método BIP39 que permite almacenar la clave privada como 12 o 24 palabras más una contraseña.
Estas palabras se almacenan acuñando sobre un metal anti corrosivo (como el acero 304) que sea resistentes a una variedad de ambientes extremos.

Finalmente terminamos este documento con las siguientes frases:

SI NO SON TUS CLAVES NO SON TUS BITCOINS.
NO CONFIES, VERIFICA.
NO NECESITAS DAR TUS DATOS (KYC) PARA USAR BITCOIN.