In [1]:

```
class Ed(object):
    def __init__(self,p, a, d , ed = None):
        assert a != d and is_prime(p) and p > 3
        K         = GF(p)

        A =  2*(a + d)/(a - d)
        B =  4/(a - d)

        alfa = A/(3*B) ; s = B

        a4 =  s^(-2) - 3*alfa^2
        a6 =  -alfa^3 - a4*alfa

        self.K = K
        self.constants = {'a': a , 'd': d , 'A':A , 'B':B , 'alfa':alfa ,
        self.EC = EllipticCurve(K,[a4,a6])

        if ed != None:
            self.L = ed['L']
            self.P = self.ed2ec(ed['Px'],ed['Py'])  # gerador do gru
        else:
            self.gen()

    def order(self):
        # A ordem prima "n" do maior subgrupo da curva, e o respetivo cofa
        oo = self.EC.order()
        n,_ = list(factor(oo))[-1]
        return (n,oo//n)

    def gen(self):
        L, h = self.order()
        P = O = self.EC(0)
        while L*P == O:
            P = self.EC.random_element()
        self.P = h*P ; self.L = L



    def is_edwards(self, x, y):
        a = self.constants['a'] ; d = self.constants['d']
        x2 = x^2 ; y2 = y^2
        return a*x2 + y2 == 1 + d*x2*y2

    def ed2ec(self,x,y):       ## mapeia Ed --> EC
        if (x,y) == (0,1):
            return self.EC(0)
        z = (1+y)/(1-y) ; w = z/x
        alfa = self.constants['alfa']; s = self.constants['s']
        return self.EC(z/s + alfa , w/s)

    def ec2ed(self,P):         ## mapeia EC --> Ed
        if P == self.EC(0):
            return (0,1)
        x,y = P.xy()
        alfa = self.constants['alfa']; s = self.constants['s']
        u = s*(x - alfa) ; v = s*y
        return (u/v , (u-1)/(u+1))
```

```python
def sign(message):
    private_key = Ed25519PrivateKey.generate()
    signature = private_key.sign(message)
    public_key = private_key.public_key()
    return signature, public_key

def verify(signature, public_key, message):
    public_key.verify(signature, message)
```

In [2]:

```python
p = 2^255-19
K = GF(p)
a = K(-1)
d = -K(121665)/K(121666)

ed25519 = {
'b'  : 256,
'Px' : K(15112221349535400772501151409588531511454012693041857206046113283
'Py' : K(46316835694926478169428394003475163141307993866256225615783033603
'L'  : ZZ(2^252 + 27742317777372353535851937790883648493), ## ordem do subg
'n'  : 254,
'h'  : 8
}

E = Ed(p,a,d,ed=ed25519)
```

In [3]:

```python
from cryptography.hazmat.primitives.asymmetric.ed25519 import Ed25519Privat



sig, pk = sign(b'Hello!')

try:
    verify(sig, pk, b'Hello!')
except:
    print("Erro na verificação")
```