



2-10-2024

# Session 5: ECDSA

**Nombre:** Álvarez García Brandon Azarael

**Nombre de la materia:** Selected topics in cryptography

**Grupo:** 7CM1

**Nombre de la profesora:** Dra. Sandra Díaz Santiago

## Avances.

### Ejercicios de programación

1. Develop a function to configure the public parameters of the protocol ECDSA. Remember that the public parameters are: a prime number  $p$ , the parameters of an elliptic curve  $a, b \in \mathbb{Z}_p^*$ , a generator point  $G \in E(a, b)$  and the order of  $G$ ,  $q$ . These parameters MUST NOT be generated at random. Please consider that the user must be able to establish them without editing the source code.

Para este ejercicio realice una clase llamada ECDSA la cual creo un constructor para definir los parámetros que ocuparemos.

```
1 import math
2 import random
3
4 usage new *
5
6 class ECDSA:
7     new *
8     def __init__(self, p, a, b, G, q):
9         self.p = p
10        self.a = a
11        self.b = b
12        self.G = G
13        self.q = q
```

Y reutilizamos el código de hace dos prácticas, para la suma de puntos, la comprobación de puntos, y la multiplicación de puntos por el escalar.

```
13 def isPoint(self):
14
15     x, y = self.G
16
17     y2 = (math.pow(x, 3) + self.a * x + self.b) % self.p
18     sr = (math.pow(y, 2)) % self.p
19
20     if (y2 == sr):
21         return True
22     else:
23         return False
24
25 2 usages new *
26 def euclides_extendido(self, a, b):
27     x0, x1, y0, y1 = 1, 0, 0, 1
28     while b != 0:
29         q, a, b = a // b, b, a % b
30         x0, x1 = x1, x0 - q * x1
31         y0, y1 = y1, y0 - q * y1
32     return x0 % self.p
```

```

53 def sum_points(self, P, Q):
54     x1, y1, z1 = P
55     x2, y2, z2 = Q
56
57     if z1 == 0:
58         return Q
59     if z2 == 0:
60         return P
61
62     if P == Q:
63         s = (3 * x1 ** 2 + self.a * z1 ** 2) * self.euclides_extendido(2 * y1 * z1, self.p) % self.p
64     else:
65         # Fórmula para suma de puntos
66         s = (y2 * z1 - y1 * z2) * self.euclides_extendido(x2 * z1 - x1 * z2, self.p) % self.p
67
68     x3 = (s ** 2 - x1 * z2 - x2 * z1) % self.p
69     y3 = (s * (x1 * z2 - x3) - y1 * z2) % self.p
70     z3 = (z1 * z2) % self.p
71
72     return (x3, y3, z3)

```

```

54 def scalar_multiplication(self, k, P):
55
56     result = (0, 1, 0) # Punto al infinito
57     addend = P
58
59     while k > 0:
60         if k % 2 == 1:
61             result = self.sum_points(result, addend)
62             addend = self.sum_points(addend, addend)
63             k //= 2
64
65     return result

```

Yaqui se crea la función para la generación de llaves y el guardado de la llave publica en un archivo de texto.

```

67 def generateKeys(self):
68
69     # Generate private key
70     d = random.randint(1, self.q)
71
72     # Deconstructing A
73     x1, y1, z1 = self.G
74     A = (x1, y1)
75
76     # Deconstructing B
77     xb, yb, zb = self.scalar_multiplication(d, self.G)
78     B = (xb, yb)
79
80     # Generate public key
81     kpub = (self.p, self.a, self.b, self.q, A, B)
82
83     # Save public key in a text file
84     file = open("PublicKey.txt", "w")
85     file.write(str(kpub))
86
87     return kpub, d

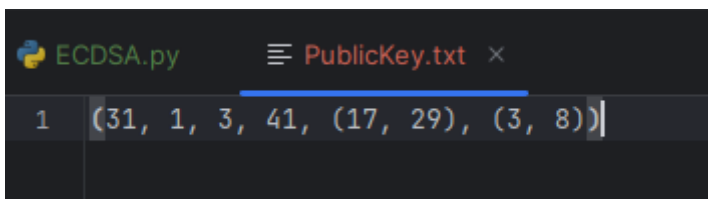
```

### Código en ejecución.

```
90 test = ECDSA(31, 1, 3, (17, 29, 1), 41)
91
92 P = test.generateKeys()
93
94 print(f'{P}')
```

```
D:\Programs\Anaconda\envs\PersonalProjects\python.exe
(31, 1, 3, 41, (17, 29), (3, 8))

Process finished with exit code 0
```



ECDSA.py    PublicKey.txt ×

```
1 (31, 1, 3, 41, (17, 29), (3, 8))
```