IIC3272 — Criptomonedas y contratos inteligentes

Tarea 2: Serialización y direcciones de Bitcoin

Entrega: Jueves – 25 – Septiembre – 23:59

1. Cosas administrativas

Cada tarea en este curso equivale a 20 % de la nota final. En total vamos a tener 6 tareas, pero la peor tarea que solucionan no cuenta para la nota final. Quiere decir que pueden botar una tarea.

En esta tarea se les pide implementar ciertos métodos en las clases implementadas durante la actividad sobre las curvas elípticas, y después ejecutar una secuencia de comandos que imprime ciertos resultados.

El programa con su solución hay que subir por el buzon de canvas.

Uso de materiales (o soluciones) encontradas en Internet está permitido. Solo se les pide citar la fuente que utilizaron. No se aplica ninguna penalidad para su uso. Si entienden como usar una billetera electrónica comercial para resolver la tarea tampoco se aplica una penalidad.

2. La tarea

En esta tarea vamos a extender nuestra implementación de elementos de criptografía de curva elíptica, para poder transmitir llaves públicas en el formato SEC, transmitir las firmas en el formato DER, y generar direcciones de Bitcoin.

El código desarrollado durante las clases nos permite definir los elementos del campo finito utilizado en Bitcoin, definir puntos de la curva elíptica secp256k1, y firmar/verificar mensajes utilizando esta curva elíptica. En esta tarea recibirán el código extendido con algunas funciones como el hash160 que nos ayuda en calcular direcciones de Bitcoin, o transformación de una secuencia de bytes a base 58.

Objetivo de la tarea es extender tres de las clases definidas en el código de siguiente manera:

- En la clase `S256Point` (acuérdense que una llave pública es simplemente un punto de esta curva) tienen que implementar el método:

  `def sec(self, compressed=True)`

  que produce serialización SEC del punto con cual estamos trabajando. El parámetro `compressed` nos dice si vamos aplicar el formato comprimido o descomprimido de SEC. El objeto que devuelve el método debe ser en formato bytes.

- En la clase `Signature` deberán implementar el método:

  `def der(self)`

  que devuelve el formato DER de la firma, en formato bytes.

- Volviendo a la clase `S256Point`, se les pide generar llaves de Bitcoin correspondiente a la llave pública con cual estamos trabajando. Para esto, deberían implementar el método:

  `def address(self, compressed=True, testnet=False)`

  que genera la dirección de Bitcoin basada en la llave pública con cual estamos trabajando (i.e. el objeto `S256Point`).

  El parámetro `compressed` nos dice si vamos a generar la dirección desde el formato SEC comprimido, o desde el formato SEC no comprimido. Para convertir el punto de la curva al formato deseado, deben utilizar el método `sec(...)` implementado en el ejercicio 1 arriba.

  El parámetro `testnet` nos dice si vamos a generar una dirección de mainnet o del testnet de Bitcoin. Como un sanity check, cuando generan una dirección de mainnet, dicha debería empezar con un 1, y una dirección de testntet con n o m. Más sobre los prefijos pueden leer en `https://en.bitcoin.it/wiki/List_of_address_prefixes`.

3. Entrega

Su entrega debe ser una archivo `zip` (con el nombre que quieran) que contenga un único archivo `hw2.py`.

Para evaluar su trabajo en el código dejamos un test con la salida esperada. El puntaje será lo siguiente:

1. **SEC** [**2 puntos**].
2. **DER** [**2 puntos**].
3. **Direcciones** [**2 puntos**].

Code hints:

Para convertir un número a bytes en el formato big-endian, pueden usar:

```
num.to_bytes(32, byteorder='big')
```

Para el formato DER, recuerden eliminar los 0 al inicio:

```
bytes.lstrip(b'\x00')
```

Para calcular base58 encoding, pueden usar:

```
encode_base58(bytes)
```