Homework 2: Serialization and Bitcoin addresses

Deadline: November 18th

1. Administration

Each homework in this course will contribute 20 % to the final grade. There will be 4 homework assignments and one final project. If needed, we might have a bonus assignment, which can replace your worst homework, but not the final project.

In this assignment we will need to modify an existing Python program in order to implement some functionalities of elliptic curve cryptography, and use your implementation to generate Bitcoin addresses which are valid on both the Bitcon mainnet and the Bitcoin testnet.

You should extend the library hw2.py, which you receive together with this assignment.

The library hw.py has no implementation for certain methods, and your assignment asks to implement these in order to generate the data that is asked from you in this homework. It will not be necessary for you to send in the .py file, but you can do so if you wish. What will be asked is an output of your program.

Your solution should be sent to the following email:

- domagojvrgoc@gmail.com

There is no restriction or penalty for using materials you found online. It is a good practice to mention those if you use them. This will not result in any penalty to your homework.

2. The homework

In this homework assignment, we will extend our implementation of ECC cryptography so that we are able to: (i) transmit public keys in the SEC format; (ii) transmit digital signatures in the DER format; and (iii) generate valid Bitcoin addresses.

The code we already developed allows us to define the elements of the finite field used in Bitcoin, define points on the curve secp256k1, and sign messages using this curve. The file hw2.py contains all of these elements, as well as some functions such as hash160, or a function for transforming a sequence of bytes into base 58, which will allow us to compute Bitcoin addresses

The objective of the homework is to extend the classes defined in hw2.py in the following manner:

- In the class `S256Point` (recall that a public key is simply a point on this curve) you have to implement the method:

  `def sec(self, compressed=True)`

  which produces the SEC serialization of the point we are working with. The parameter `compressed` tells us whether we wish the compressed or the uncompressed SEC format. The object returned by this method should be in byte format (i.e. raw).

- In the class `Signature` you should implement the method:

  `def der(self)`

  that returns teh DER format of our signature, in byte format.

- Coming back to the class `S256Point`, you are asked to generate the Bitcoin address that corresponds to the public key we are working with (recall, our object is a point). For this, you should implement the method:

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

  which generates the Bitcoin address based on the public key we are working with (i.e. the object of the class `S256Point`).

  The parameter `compressed` tells us if we will generate the address from the compressed SEC format, or from the uncompressed one. To convert the point into the desired format, you should use the method `sec(...)` that you implemented in the exercise above.

  The parameter `testnet` tells us if we are going to generate a mainnet or a testnet address. As a sanity check, when you generate a mainnet address, its first digit should be 1, and for a testnet address n or m. You can read more about these prefixes at `https://en.bitcoin.it/wiki/List_of_address_prefixes`.

To evaluate your work, this time you need not hand in the code, but rather the output of your program when the following commands are executed:

1. **SEC** [**3 points**].

```
# Generating my private key:
secret = hash256(b'BitcoinSucks')
intSecret = int(secret.hex(),16)

privKey = PrivateKey(intSecret)

# Displaying the public key in the two SEC formats:
print('Uncompressed SEC format: ',privKey.point.sec(False).hex())

print('Compressed SEC format: ',privKey.point.sec(True).hex())
```

2. **DER** [**1 point**].

```
# Signing a message:
message = hash256(b'This course is boring!')
z = int(message.hex(),16)

signature = privKey.sign(z)


# What is the signature: raw vs DER
print('Raw signature: ',signature)

print('DER signature: ',signature.der().hex())
```

3. **Direcciones** [**3 points**].

```
# Bitcoin address assuming compressed SEC format for the public key
testnet = privKey.point.address(compressed = True, testnet = True)
mainnet = privKey.point.address(compressed = True, testnet = False)

print('Testnet address: ',testnet)
print('Mainnet address: ',mainnet)
```

In order to make sure you don't mix up the order of execution, the file hw2.py already contains the commands you should execute.

Bonus 1. [**3 points**] Using the testnet address generated above, obtain any amount of testcoins using some Bitcoin testnet faucet. A testnet faucet is a service that can send you some testcoins for free so that you can play with them in your implementations. A good option is `https://bitcoinfaucet.uo1.net/send.php`. To confirm that you actually received the testcoins, you can enter a block explorer such as `https://live.blockcypher.com/`, select the option BTC testnet, and search for the address generated in exercise 3 above. There, you should find a transaction that sends the funds to the specified address.

Bonus 2. [**You feel cool**] Using the methods you just implemented, you can now generate any number of valid Bitcoin addresses. Congratulations!