Lab Assignment 1: Bitcoin Key and Address Generation with SageMath

This assignment requires the creation of a SageMath worksheet containing functions for generating both public and private Bitcoin keys, as well as determining their associated addresses. Additionally, it encompasses the generation of private keys in WIF format, suitable for importing into standard wallets.

Attached to this statement is a skeleton with the functions to be implemented. The description of these functions is as follows:

1. Key generation function. **(2.5 points)**

   Implement the `key_gen()` function that generates a pair (private key / public key) of keys for the Bitcoin system. The function should not receive any input parameters and must generate the private key randomly. The function will return a vector of two values. The first will be the private key (an integer in the ring in which the `secp256k1` curve works). The second value in the vector will be the public key (a point on the `secp256k1` elliptic curve).

2. Public key derivation function. **(1 point)**

   Implement the `pk_from_sk(pk)` function that generates the public key from the private key. The function will receive as a parameter `sk`, the private key as obtained from the `key_gen()` function. The function will return the public key (whose type will be a point on the elliptic curve `secp256k1`).

3. Private key export function. **(2.5 points)**

   Implement the `sk_to_wif(sk, network, compressed)` function that, from a private key, returns its value in WIF format. The function will receive three input parameters. The first will be the private key as obtained from the `key_gen()` function, the second the network for which the key is to be exported (which can be MAINNET or TESTNET), and the third a boolean (True/False) indicating whether the export should be done in compressed mode or not. The function will return the private key value in WIF format.

   A detailed description of how to convert a private key to a WIF format private key is available here: `https://en.bitcoin.it/wiki/Wallet_import_format`. Additionally, to implement this function, you can use the `doublehash()` and the `b58encode()` functions found in the *Helpers* section of the SageMath file.

4. Bitcoin address conversion function. **(2.5 points)**

   Implement the `get_address(pk, network, compressed)` function that, from a public key, returns its Bitcoin address in P2PKH format. The function will receive three input parameters. The first will be the public key as obtained from the `key_gen()` function, the second the network for which the key is to be exported (which can be MAINNET or TESTNET), and the third a boolean (True/False) indicating whether the export should be done in compressed mode or not. The function will return the address value.

   To implement this function, you can use the `public_key_to_bc_address()` function found in the *Helpers* section of the SageMath file.

5. Vanity Address Creation. **(1.5 points)** ⚠

   Implement a function that allows the calculation of vanity addresses starting with a specific prefix. The function will receive a character string with the prefix as a parameter and should return the address, the corresponding private key, and the time it took to find the address. If it's not possible to find an address with the specified characteristics, the function should return an error indicating so.

   Next, use this function to find vanity addresses for the prefixes `1TBC` and `qTBC`. Provide the found addresses, their corresponding private keys, and the time it took to obtain them.

Hints

In SageMath, when converting an integer variable, such as `variable = 45`, to its hexadecimal representation as a string, the `hex()` function can be utilized. Executing `print(hex(variable))` will yield the string `'0x2d'`. To reverse this process, converting a string with hexadecimal characters back to its integer representation, the `ZZ()` function in SageMath can be employed. For instance, converting the string `'0x2d'` representing hexadecimal values to an integer can be done using `ZZ('0x2d')`.

It's important to note that each component of a point on an elliptic curve does not have the sage integer type (they are integers in a finite ring). To convert them into integer variables, the same instruction mentioned earlier, `ZZ(variable)`, can be applied.

Furthermore, be aware that certain functions within the helpers section expect data as bytearrays. The docstring of these functions specifies the format of parameters and the return value in the helpers section. In Python 3 (and also in SageMath), converting a string variable containing a hexadecimal representation (e.g., `string = '2F3A'`) to a bytearray can be achieved as follows: `bytearray.fromhex(string)`.

Submission Instructions

Below are the submission instructions for the assignment:

1. The submission of the assignment will consist of the SageMath worksheet file (`.sagews`) with the provided skeleton, where you have implemented the functions as specified in the statement. If you need to add textual explanations to any of the answers, you can include text cells in the same worksheet.

2. Submit the assignment through the virtual campus, in the task labeled "Lab Assignment 1 Submission".

3. The **deadline for submission** of the assignment is **12/02/2024**.