Making Use of the Wallet Hierarchy

In the previous exercise, you created an HD wallet by generating a master key from the mnemonic seed phrase, then deriving a public/private key pair using a BIP44 path. With this child node of the HD wallet, you were able to send bitcoin by collecting UTXOs that were sent to the corresponding address and putting them as inputs in the new transaction. You specified the desired outputs, then finally signed and broadcasted the transaction across the bitcoin testnet.

While you certainly created an HD wallet, you only ever used *one* key pair to manage your funds. The purpose of this exercise will be for you to generate and receive funds at multiple bitcoin addresses (all within the same hierarchy), then sign multiple UTXOs with their appropriate private keys. Again, you will broadcast the finalized transaction over the bitcoin testnet.

> *Task:* Review your wallet implementation from the previous exercise, as it will serve as a starting point/reference. It will be important to firmly understand the process of generating an address, supplying the inputs/specifying the outputs to build a transaction, signing a transaction, and broadcasting the transaction on the blockchain.

Address Reuse

Due to the public nature of the bitcoin blockchain, it is trivial for a nosey onlooker to aggregate transactions from an address and learn the details of its financial history. Because of this privacy issue, along with a few other consequences of address reuse, it is best practice to use a new address for every transaction. While this may sound like a cumbersome solution, HD wallets make the process painless.

Unused Address Set

Recall that an HD wallet can derive virtually unlimited key pairs/addresses from a single master key. Also, recall that the address hierarchy structure defined by BIP43 and BIP44 is as follows:

```
m / purpose' / coin_type' / account' / change / address_index
```

By changing any of the fields in the derivation path, a new key pair/address will be generated. The first step of an HD wallet committed to avoiding address reuse is to discover which derived addresses are available (unused). The process for doing so for a given account is as follows:

1. derive an address using the path (starting at address_index = 0)
2. scan the blockchain for transactions associated with this address; respect the *address gap limit* described below as the stopping condition
    - if no transactions are found, the address is considered unused
3. increment address_index and repeat

Address Gap Limit / Stopping Condition

The address gap limit should be set to 20 (this is most common, but somewhat arbitrary). If you scan 20 unused addresses in a row, you can expect there are no used addresses beyond this point and stops searching the address chain. *In determining unused addresses, do not consider the change addresses.*

UTXOs Across Multiple Addresses

Up to this point, you have collected UTXOs from only one address in order to make a transaction. However, UTXOs may be spread across the children of your wallet hierarchy now that you are using a new address each time you receive BTC. Therefore, in order to know the balance of your wallet, you need to calculate the aggregate value of UTXOs belonging to every used derived address. Also, if the UTXOs of one address cannot cover the value of a transaction you are trying to create, you may need to add UTXOs belonging to other children of the wallet hierarchy as well.

Sign Transaction With Multiple UTXOs

The process of signing transactions with UTXOs from multiple child nodes is the most important difference of this exercise compared to the previous one. Because each address of the wallet hierarchy has a corresponding public/private key pair, every child that is associated with a UTXO being used in the transaction must sign. Thus, multiple signatures will be included in a transaction that uses UTXOs associated with different addresses. It is the wallet's responsibility to sign the transaction using the appropriate private keys.

Transitioning Your First Wallet Implementation

Using the above sections as context, you are going to change the wallet implementation you made in the previous exercise, such that you only use unused addresses to receive tBTC and can create transactions using UTXOs that belong to different addresses within the wallet hierarchy. Below are some steps you that may help you, but you are free to go about the making this transition in any way you see fit.

> *Task*: Use the Blockstream API to aggregate unused addresses associated with account 0 of your HD bitcoin wallet. Use the *Unused Address Set* section of this document to help.

> *Task*: Using a bitcoin testnet faucet (i.e. https://bitcoinfaucet.uo1.net), send funds to a newly generated unused child address. Do this three times, such that three addresses within your wallet hierarchy have UTXOs associated with them. After an address receives tBTC, your wallet application should create a new unused address.

> *Task*: Use the Blockstream API to collect UTXOs that belong to each used address of the wallet hierarchy. Keep record of the UTXOs such that you are able to identify which child node is able to spend them. Your application will grab from this set of UTXOs when building a transaction, so you will need to be able to sign with the appropriate child when the time comes.

> *Task*: Use the UTXOs as inputs to send tBTC to a destination address. Use the transaction building process of the previous exercise as reference; the primary difference is that the UTXOs used may not belong to one address. Once enough UTXOs have been added as inputs to cover the cost of the transaction, add an output specifying the destination address/value being sent, and add another output to send change to an unused address within your wallet hierarchy.

> *Task*: Use the child nodes corresponding to the UTXOs included to sign the transaction.

> *Task*: Broadcast the transaction on testnet in the same manner that you did in the previous exercise.