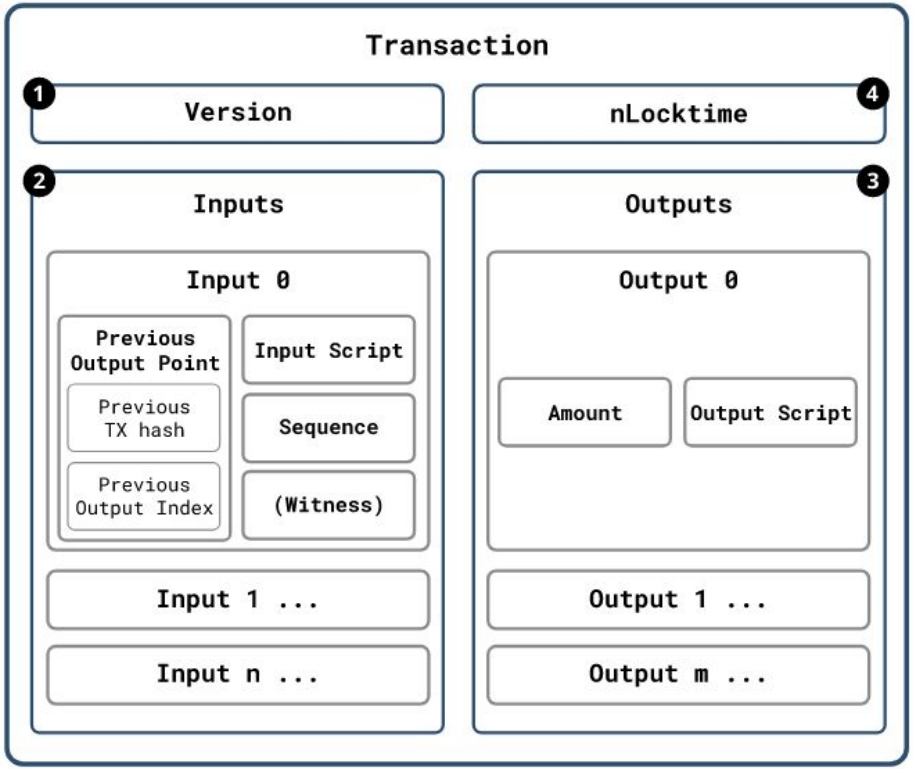
Wallets

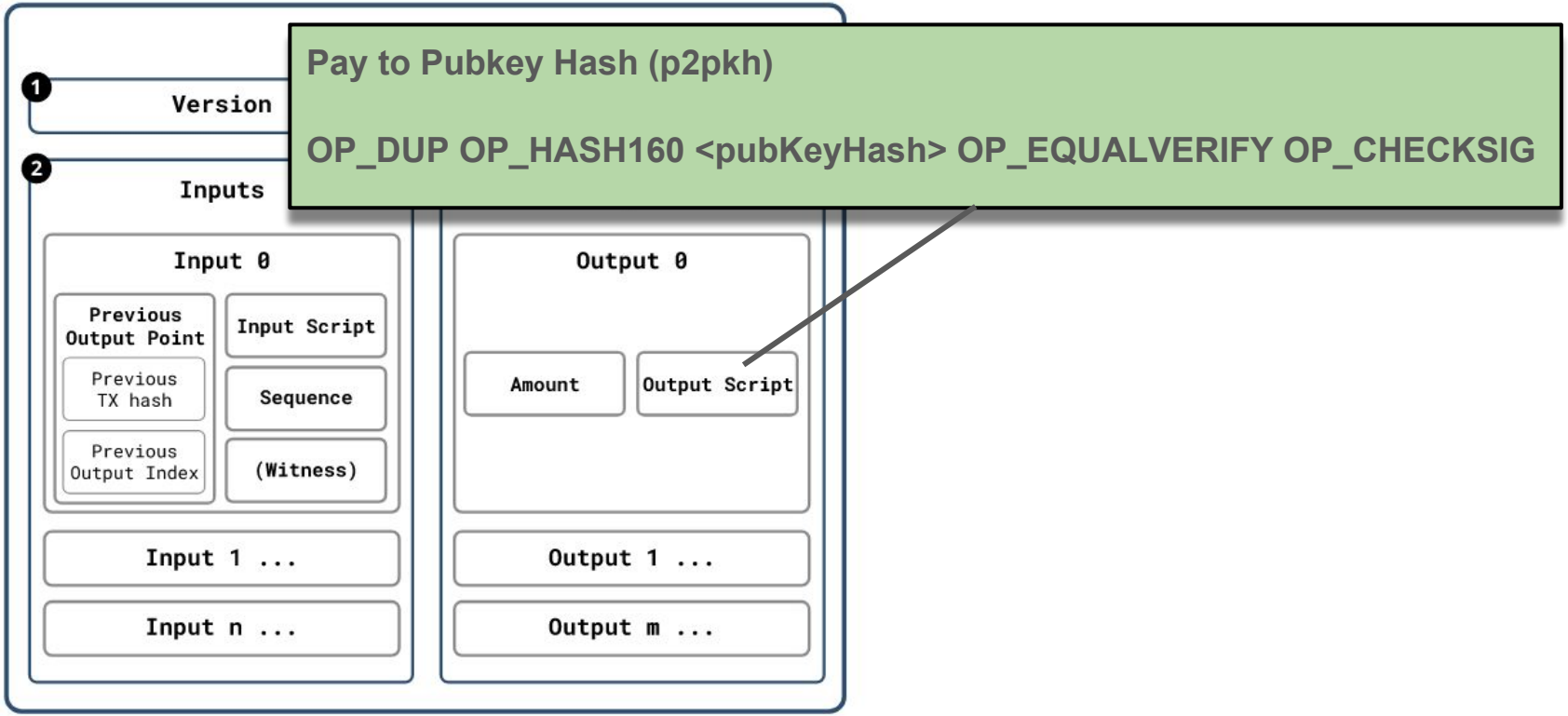
What is in a wallet?

- Privacy considerations
- Key management?
- Different account types - p2pkh, p2wpkh, multisig
- A lot of BIPs!
- Receive addresses and change addresses
  - Gap limits
- Accessing relevant data from the blockchain
  - Transactions that touch any of your receive or change addresses
- Managing your UTXOs
- Building, signing, and broadcasting transactions

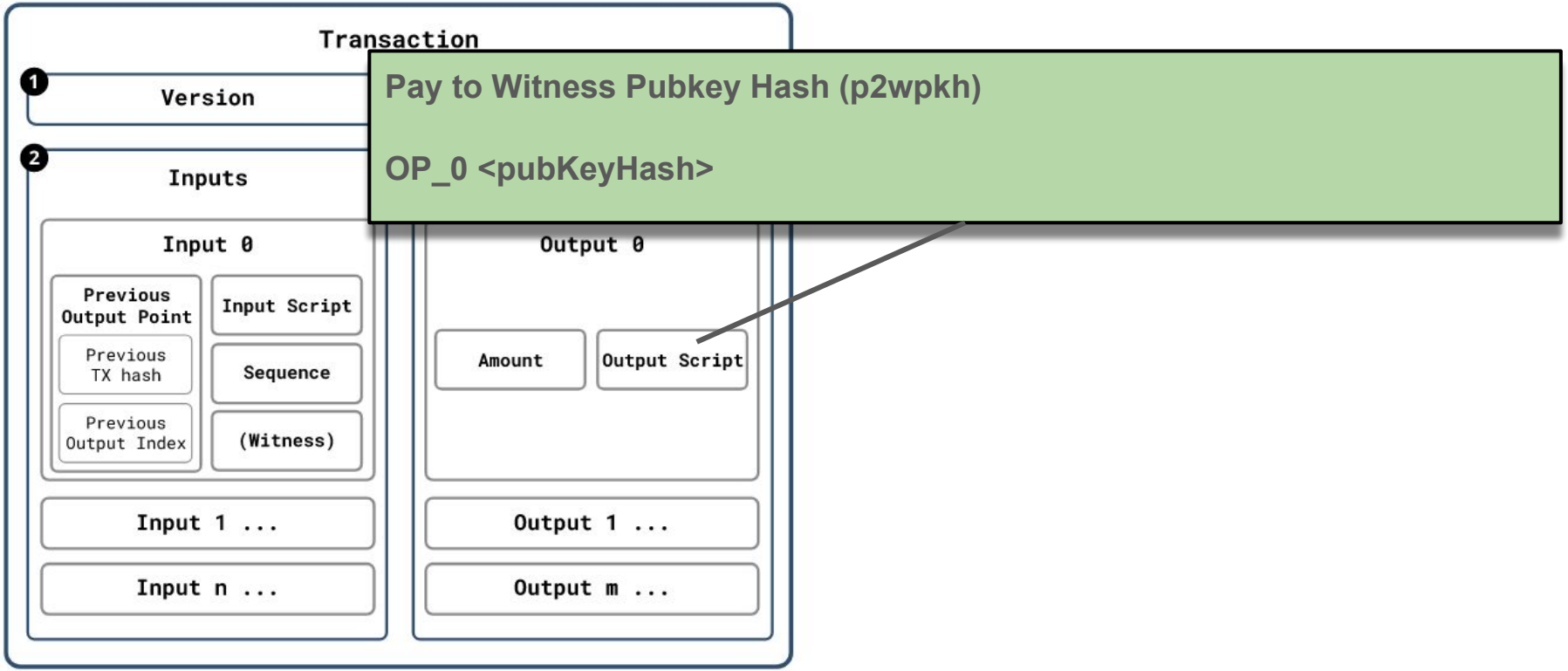
Refresher: what is an output?



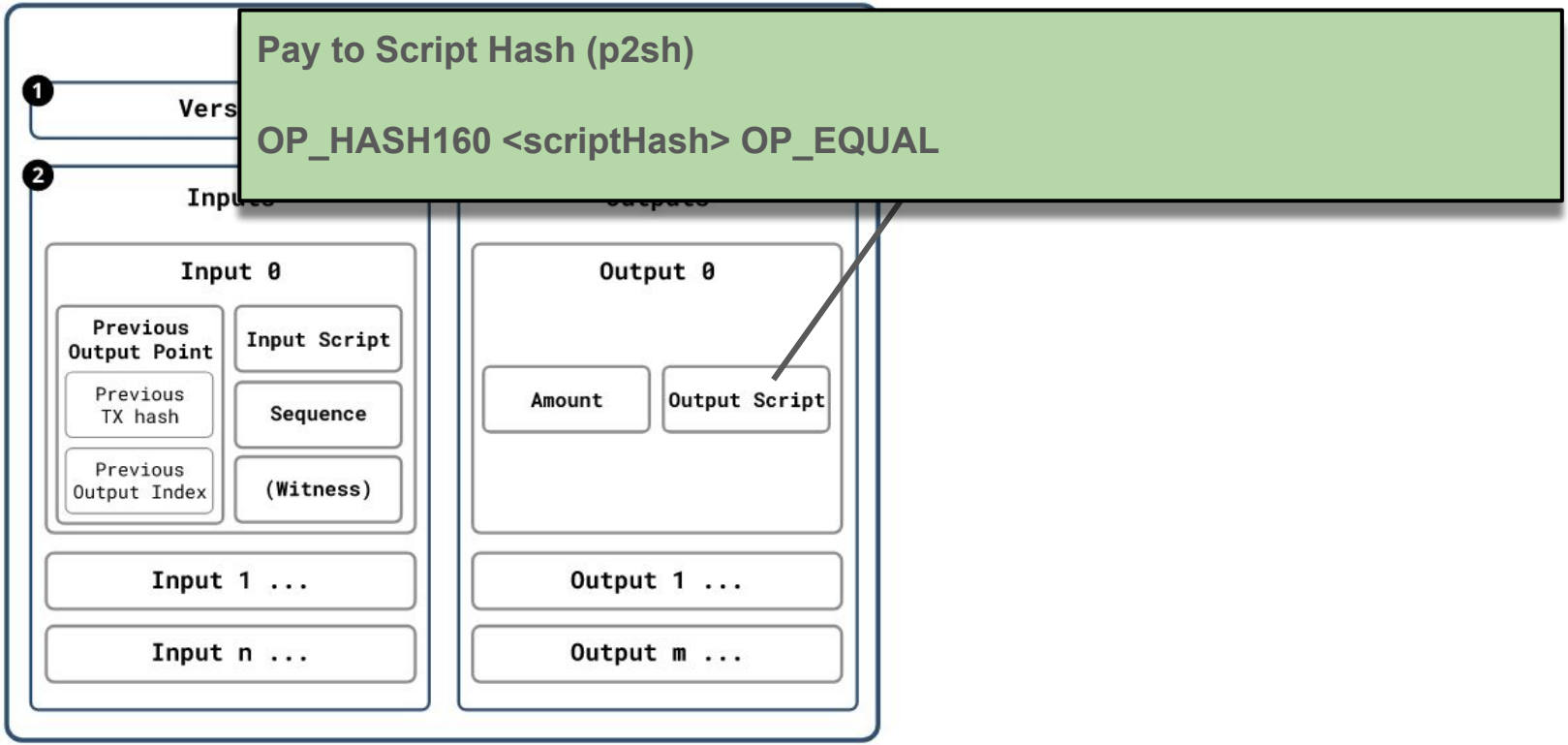
Refresher: what is an output?



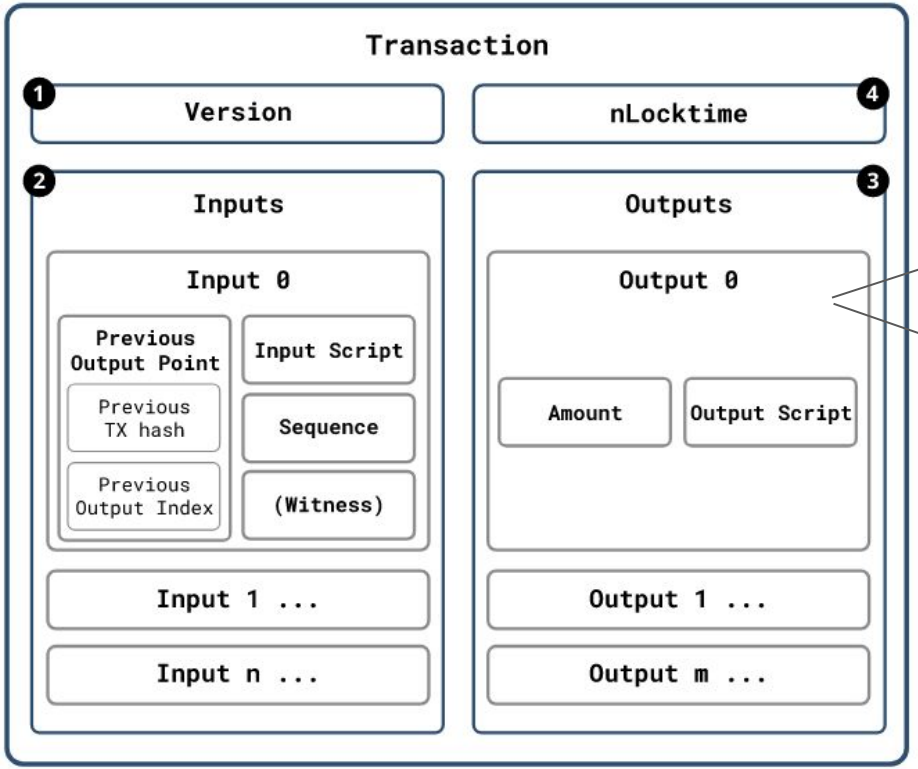
Refresher: what is an output?



Refresher: what is an output?



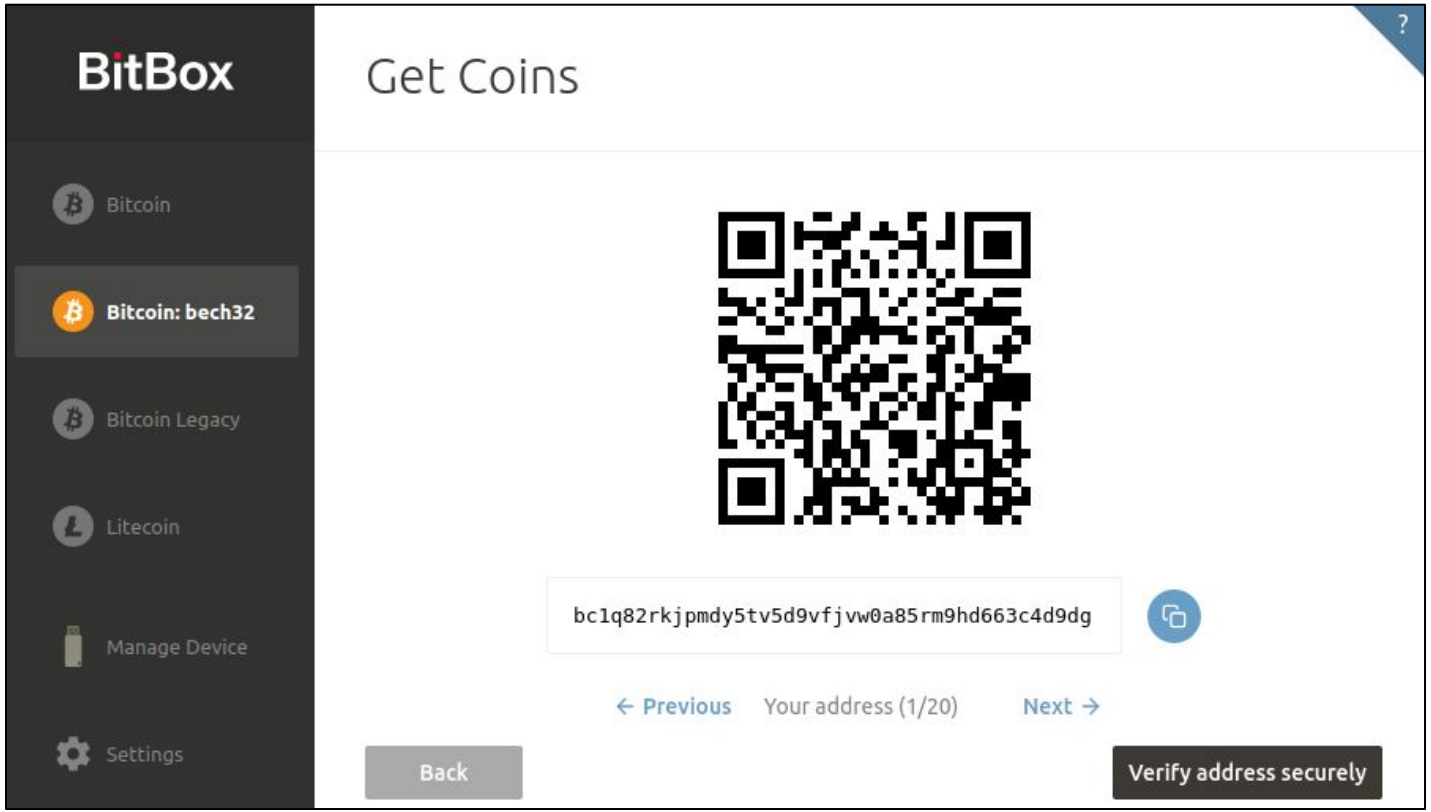
Refresher: what is an output?



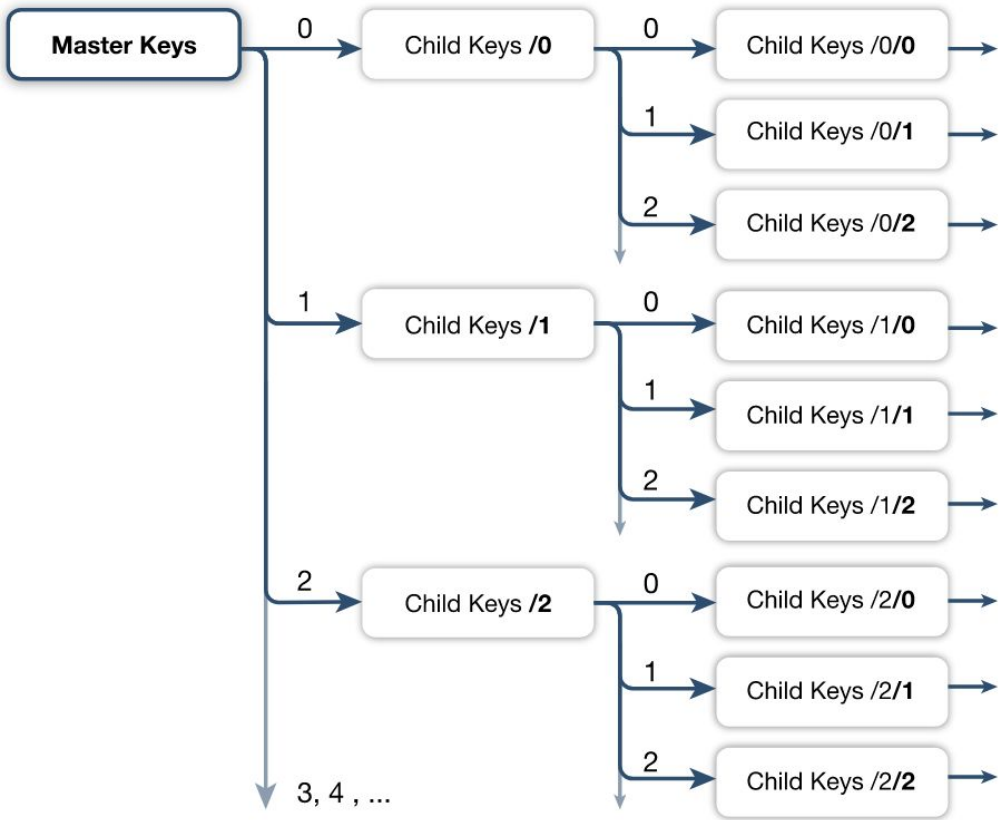
If unspent, called an **UTXO**:
Unspent Transaction Output

You have control of
the output, if you are able to
provide the correct input script

How to organize keys, accounts, scripts/addresses?



BIP32 - Hierarchical Deterministic Key derivation



HD wallets (BIP32) can deterministically derive an indefinite number of fresh addresses from a single wallet secret.

HD Tree

- Fresh addresses to improve privacy.
- HD Tree is derived from Master Keys.
- HD Tree can be reconstructed from master Keys (given tree structure).

Master keys

- Derived from HD root secret.

Subtrees

- Allow separation of keys for accounts/usages.
- Selective key sharing.

BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH (BIP84):

Root XPRV / 84' / 0' / 2' / 0 / 9

points to the 10th key controlling p2wpkh outputs of your 3rd account.

BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH:

Account XPUB / 0 / 9

to reconstruct an output script, generate a receive address

BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH:

Account XPRV / 0 / 9

to spend outputs matching the output script

Output Scripts vs Account Type vs Addresses

Addresses encode Output Scripts.
There is an encoding for all standard output scripts.

Account Types:
Legacy (1..)
Multisig (3…)
Native Segwit (bc1…)

One address format per account type?

What is an account balance?

What is an account balance?

Sum of all UTXOs found under

XPUB / 84' / 0' / 0' / <change> / <index>

What is an account balance? - Pseudocode

```
1    def get_balance(xpub):
2        return sum_utxos(xpub, change=False) + sum_utxos(xpub, change=True)
```

```
4   def sum_utxos(xpub, change):
5       xpub = xpub.Child(1 if change else 0)
6       gap_limit = 20
7       result_sum = 0
8       gap, index = 0, 0
9       for gap < gap_limit:
10          current = xpub.child(index)
11          # construct a p2wpkh output script: OP_0 OP_PUSH20 pubkey_hash
12          pubkey_hash = hash160(current.publicKey.serializeCompressed())
13          output_script = bytearray([0, 20]) + pubkey_hash
14
15          utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
16          result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))
17
18          if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
19              gap = 0
20          else:
21              gap += 1
22          index += 1
23      return result_sum
```

```
4   def sum_utxos(xpub, change):
5       xpub = xpub.Child(1 if change else 0)
6       gap_limit = 20
7       result_sum = 0
8       gap, index = 0, 0
9       for gap < gap_limit:
10          current = xpub.chi
11          # construct a p2wp
12          pubkey_hash = hash
13          output_script = bytearray([0, 20]) + pubkey_hash
14
15          utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
16          result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))
17
18          if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
19              gap = 0
20          else:
21              gap += 1
22          index += 1
23      return result_sum
```

Validate!

- **SPV proofs: were the tx really mined in blocks?**
- **Blocks had sufficient proof of work?**

```
4   def sum_utxos(xpub, change):
5       xpub = xpub.Child(1 if change else 0)
6       gap_limit = 20
7       result_sum = 0
8       gap, index = 0, 0
9       for gap < gap_limit:
10          current = xpub.child(index)
11          # construct a p2wpkh output script: OP_0 OP_PUSH20 pubkey_hash
12          pubkey_hash = hash160(current.publicKey.serializeCompressed())
13          output_script = bytearray([0, 20]) + pubkey_hash
14
15          utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
16          result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))
17
18          if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
19              gap = 0
20          else:
21              gap += 1
22          index += 1
23      return result_sum
```

Blockchain Indexing Techniques

Problem:

How do we keep track of our coins (UTXOs)?

Given an output script, how do we find all relevant transactions?

- Transactions sending money to it (creating UTXOs)
- Transactions spending it (destroying UTXOs)

We need **`blockchain.scripthash.get_history(output_script_hash)`**, i.e. something that indexes relevant transactions.

Blockchain Indexing Techniques

Three broad areas:

- Personal Index using a full node
- Personal Index using SPV
- Full Index (personal or shared)

Personal: tracks only the utxos and transactions related to your own addresses.

Personal Index - Bitcoin Core

1. Add an output script (based on a public key) to the watchlist.

2. When a new block arrives, scan all transactions in it:
   a. If a transaction output script matches, add the output to our UTXO set, and the transaction to the transaction list.

   b. If a transaction spends a matching output, remove it from our UTXO set, and add the transaction to the transaction list.

Our balance is the sum of the UTXOs.

What about Recovery?

Tracking outputs is lightweight, but rescanning the past is not.

Need to rescan the whole chain, which can take a long time.

If willing to lose the transaction history, one can scan only the full UTXO set.

→ Recently released **scantxoutset** RPC call in Bitcoin Core. Still experimental, not yet in use.

SPV - Bloom Filters and Neutrino

Same indexing as before, but process only a small subset of the blockchain.

Bloom Filters (BIP37): download trimmed blocks with only relevant transactions

Neutrino (BIP157,158): download full blocks, but only relevant ones

(Good for relatively low traffic clients, like Lightning)

Full Index

A full index maintains the {address: transactions} index for all outputs.

Full Index

A full index maintains the {address: transactions} index for all outputs.

Often what SPV wallets connect to.

Full Index

A full index maintains the {address: transactions} index for all outputs.

Often what SPV wallets connect to.

Slow to build index, adds currently ~35GB more data.

ElectrumX

de-facto standard protocol

blockchain.address.get_history

Return the confirmed and unconfirmed history of a bitcoin address.

> blockchain.address.get_history(**address**)
>
> **address**
>
> > The address as a Base58 string.

Response

> A list of confirmed transactions in blockchain order, with the output of *blockchain.address.get_mempool* appended
> the list. Each transaction is a dictionary with keys *height* and *tx_hash*. *height* is the integer height of the block the
> transaction was confirmed in; if unconfirmed then *height* is 0 if all inputs are confirmed, and -1 otherwise. *tx_hash* t
> transaction hash in hexadecimal.

Response Examples

```
[
  {
    "height": 200004,
    "tx_hash": "acc3758bd2a26f869fcc67d48ff30b96464d476bca82c1cd6656e7d506816412"
  },
  {
    "height": 215008,
    "tx_hash": "f3e1bf48975b8d6060a9de8884296abb80be618dc00ae3cb2f6cee3085e09403"
  }
]
```

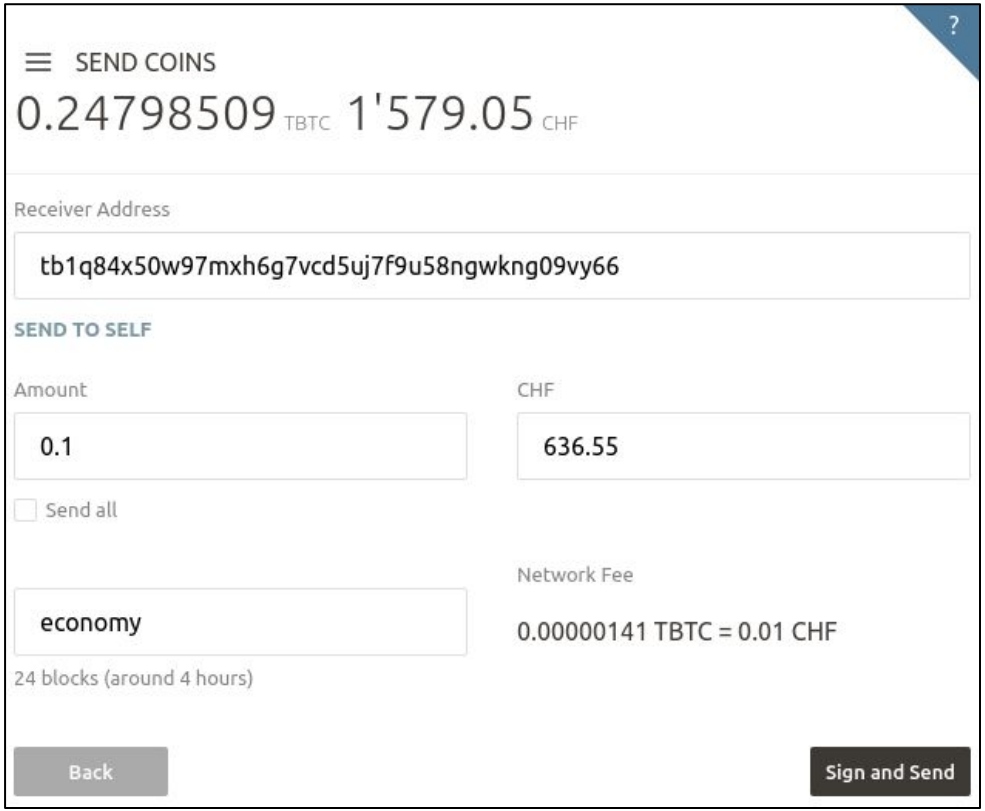
libbitcoin

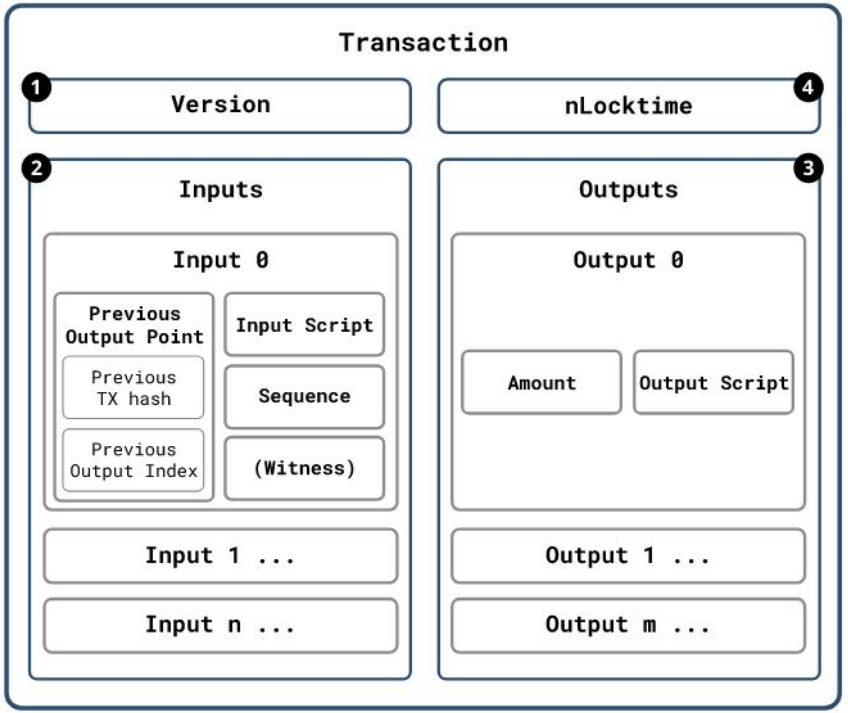
```
$ bx fetch-history 134HfD2fdeBTohfx8YANxEpsYXsv5UoWyz
```

```
transfers
{
    transfer
    {
        received
        {
            hash 97e06e49dfdd26c5a904670971ccf4c7fe7d9da53cb379bf9b442fc9427080b3
            height 247683
            index 1
        }
        spent
        {
            hash b7354b8b9cc9a856aedaa349cffa289ae9917771f4e06b2386636b3c073df1b5
            height 247742
            index 0
        }
        value 100000
    }
}
```

> The `spent` property indicates that the received amount has been spent. The `spent.height` property indicates the block height at which the spend transaction is confirmed.

Transaction Creation





Transaction Creation - Spendable Outputs

From our UTXOs, filter the ones we can safely spend

Transaction Creation - Spendable Outputs

```
// SpendableOutputs returns all unspent outputs of the wallet which are eligible to be spent. Those
// include all unspent outputs of confirmed transactions, and unconfirmed outputs that we created
// ourselves.
func (transactions *Transactions) SpendableOutputs() map[wire.OutPoint]*wire.TxOut {
    result := map[wire.OutPoint]*SpendableOutput{}
    for outPoint, txOut := range transactions.unspentOutputs {
        tx, height := transactions.TxInfo(outPoint.Hash)

        confirmed := height > 0

        if confirmed || transactions.allInputsOurs(dbTx, tx) {
            result[outPoint] = txOut,
        }
    }
    return result
}
```

Transaction Creation - Coin Selection

Requirements:

- Input values must cover the output values + mining fee
  - Annoying dependency: the fee also depends on the number of inputs and outputs
- There can be no dust output

Nice to have:

- Reduce UTXO bloat, avoid small change outputs
- Reduce fees: small number of inputs and outputs
- Privacy: make it hard to identify the change output.

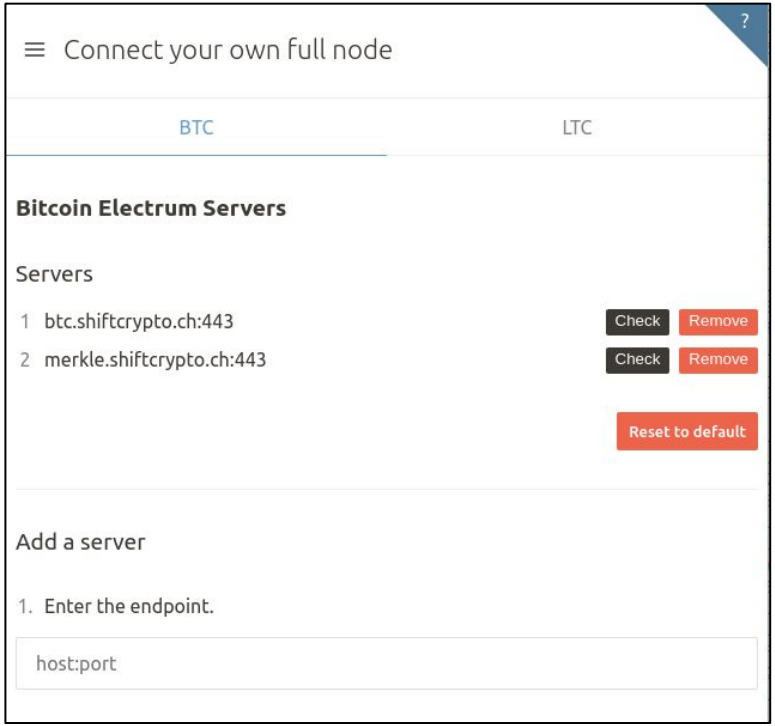
No selection policy works for all.

Transaction Creation - Coin Selection

- Bitcoin Core's algorithm until recently was very involved
- Turned out that a simple random draw performed better in many of the metrics
- Since last release, using Branch&Bound + Random Draw
  - Optimizing for exact matches to cut costs
  - http://murch.one/wp-content/uploads/2016/11/erhardt2016coinselection.pdf

Privacy Considerations

Support powering the wallet with your own full node!



Privacy Considerations

A lot about how a wallet behaves leaks information on the chain

- Types of scripts in use (if there are only a few segwit users, they stand out), and their mix
- Transaction composition can give you away
  (coin selection, input/output ordering)
- All wallets should ideally behave the same way.
- Lightning will help

Others:

- Third party servers to be minimized. Decorrelate with onchain actions.