Transactions Workshop

Requirements

- *Git / Python3 installed and configured.*
- *Secp256k1 Library (pip3 install secp256k1)*
- *Bitcoin Core set to regtest mode.*

Resources

RPC API Reference

developer.bitcoin.org/reference/rpc

Bitcoin RPC-Auth Generator

jlopp.github.io/bitcoin-core-rpc-auth-generator

Script Wiz

ide.scriptwiz.app

Bitcoin OP Codes

en.bitcoin.it/wiki/Script

Configuring bitcoin.conf

```
## bitcoin.conf file

regtest     = 1
server      = 1

[regtest]
txindex     = 1
rpcbind     = 0.0.0.0
rpcallowip = 0.0.0.0/0
rpcauth     = user:password_hash$salt

## See more options here:
# https://jlopp.github.io/bitcoin-core-config-generator
```

Creating a Wallet

- File → Create Wallet.
- Make it lowercase, no spaces.
- Un-check “Descriptor Wallet”.
- You can also use `createwallet` RPC call.

Useful Commands

- Listunspent
- Getnewaddress
- Gettransaction
- Getaddressinfo
- Signrawtransaction
- Sendrawtransaction
- dumpprivkey

Reading Bytes

- Bitcoin Core prefers to work with transaction data in raw bytes, and display this data as hex-encoded strings.
- Most fields in a transaction are fixed-length, but some are variable (such as arrays and scripts).
- In order to parse variable-length fields and arguments, we precede them with a byte to specify the size or length.
- A single byte can only represent data up to 255 bytes in length.

Varints

- Varints use a prefix byte to signal when the following integer takes up multiple bytes.
- `0xFD`, `0xFE`, `0xFF` are used to signal when the following integer value takes up 2, 4, or 8 bytes.
- Bitcoin Script also uses special opcodes to push large amounts of data onto the stack.

Decoding a Transaction

```
"version": <4 bytes LE>
"vin": [1] [{
  "txid": <32 bytes LE>
  "vout": <4 bytes LE>
  "scriptSig": <var bytes LE>
  "sequence": <4 bytes LE>
}],
"vout": [1] [{
  "value": <8 bytes LE>
  "scriptPubkey": <var bytes>
}],
"locktime": <4 bytes LE>
```

```
01000000010b6072b386d4a773235237f64c1126ac3b240c84b
917a3909ba1c43ded5f51f400000000 8c493046022100bb1ad2
6df930a51cce110cf44f7a48c3c561fd977500b1ae5d6b6fd13
d0b3f4a022100c5b42951acedff14abba2736fd574bdb465f3e
6f8da12e2c5303954aca7f78f3014104a7135bfe824c97ecc01
ec7d7e336185c81e2aa2c41ab175407c09484ce9694b44953fc
b751206564a9c24dd094d42fdbfdd5aad3e063ce6af4cfaaea4
ea14fbbffffffff0140420f00000000001976a91439aa3d569e
06a1d7926dc4be1193c99bf2eb9ee088ac00000000
```

Version	pink	count	indigo
Txid	yellow	length	purple
Vout	green	value	green
script	blue	hash	gold
Sequence	magenta	locktime	magenta

Decoding with Witness

```
"vin": [1] [
  {
    "txid": <32 bytes LE>
    "vout": <4 bytes LE>
    "scriptSig": []
    "sequence": <4 bytes LE>
    "txinWitness": [2] [
      <var-bytes argument>,
      <var-bytes argument>,
    ]
  },
]
```

```
01000000000101055d00a1d0e1cdcfb9fc390bdae64ccaef886
864fddc746e4b002c3d687a1bae00000000ffffffff01385b
a012000000001600144062afb45ce155d7ec727087d04ec8c8e
c422a7c02473044022006 63d77ebe12d9d7d999db32242d2e81
fc6f29dabc936151ea192ae17430ede002206e985ac8ca60c0c
213432a966bb39a1ac7bcbe186cd0ebdc46cdc7d4aaa7717701
210261ed72b33546be5cb6d32079d7387d37dd1ec7ae15ffd87
ba756444223d41d3200000000
```

```
  version : pink     |     count : indigo
     txid : yellow   |    length : purple
     vout : green    |     value : green
   script : blue     |      hash : gold
 sequence : magenta  |  locktime : magenta
                     |  p2w flag : red
```

Calculated Fields

```
## Transaction Fields
“txid”    : reverse(sha256(sha256(raw_tx)))
“size”    : len(raw_tx_plus_witness)
“weight” : len(raw_tx) * 3 + size
“vsize”   : weight // 4 + (weight % 4 > 0)
“txfee”   : sum(vin.values) - sum(vout.values)

## Hash Locks
“pubkey_hash”       : ripemd160(sha256(public_key))
“p2sh_script_hash” : ripemd160(sha256(redeem_script))
“p2w_script_hash”   : sha256(redeem_script)
```

Signatures

- Signatures are created using a partial digest of the spending transaction, called the signature hash.
- **OP_CHECK(MULTI)SIG** enforces that a spending transaction include a valid signature and matching public key.
- When combined with **OP_EQUALVERIFY**, you can limit the spending of an output to specific keys.
- The scope of a signature hash can be configured using special flags.

Signature Flags

- **SIGHASH_ALL:** Include all inputs and outputs in the hash. Any change to the transaction will render the signature invalid.
- **SIGHASH_SINGLE:** Include all inputs in the hash, but only the output that is adjacent to the signature’s input (same index).
- **SIGHASH_NONE:** Include all inputs in the hash, but none of the outputs.
- **ANYONECANPAY:** Modifies the above flags. The signature’s input is the only input included in the hash.

Signature Hash

```
## How to build the signature hash (as defined in BIP 143)
Double SHA256 hash of the following bytes:
    1. version                                                  <4-byte LE>
    2. hashPrevouts: hash256(txid + vin for x in vin)           <32-byte hash>
    3. hashSequence: hash256(sequence for x in vin)             <32-byte hash>
    4. txid of the current input (reversed)                     <32-byte hash>
    5. vout of the current input                                <4-byte LE>
    6. varint + redeem script for the utxo                      <var-byte LE>
    7. output value of the utxo                                 <8-byte LE>
    8. sequence value of the current input                      <4-byte LE>
    9. hashOutputs: hash256(value + script for x in vout)       <32-byte hash>
   10. locktime of the transaction.                             <4-byte LE>
   11. sighash type of the signature.                           <4-byte LE>
## https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki
```

JSON-RPC

Send HTTP requests to Bitcoin's RPC interface.

```
request: {
  type: 'POST',
  headers: {
    Authorization: 'Basic' + base64('user:pass')
    content-type: 'application/json'
  },
  body: serialize({
    Jsonrpc: '1.0',
    id: 'random_id_number',
    method: 'rpc_method_name',
    params: [ 'array', 'of', 'arguments' ]
  })
}
```

Let’s Get Building!

Clone the class repo:

`github.com/cmdruid/bitcoin-programming`

Navigate here:

`contrib/python/transactions`