

Programming Project #1

Due: 11:59pm on Mon., **Oct. 15, 2018**

Submit via Gradescope code: **9RZGVZ**

In this assignment you will create several transactions and post them to the Bitcoin testnet blockchain. We will provide starter code for this using python-bitcoinlib, a free, low-level Python 3 library for manipulating Bitcoin transactions.

1 Project Background

1.1 Overview of Block Explorer Results

Rather than having you download the entire testnet blockchain and run a bitcoin client on your machine, we will be making use of an online block explorer to upload and view transactions. The one that we will be using is called BlockCypher, which features a nice web interface as well as an API for submitting raw transactions that the starter code uses to broadcast the transactions you create for the exercises. After completing and running the code for each exercise, BlockCypher will return a JSON representation of your newly created transaction, which will be printed to your terminal. An example transaction object along with the meaning of each field can be found at BlockCypher's developer API documentation at <https://www.blockcypher.com/dev/bitcoin/#tx>. Of particular interest for the purposes of this project will be the `hash`, `input`, and `output` fields.

1.2 Anatomy of a Bitcoin Transaction

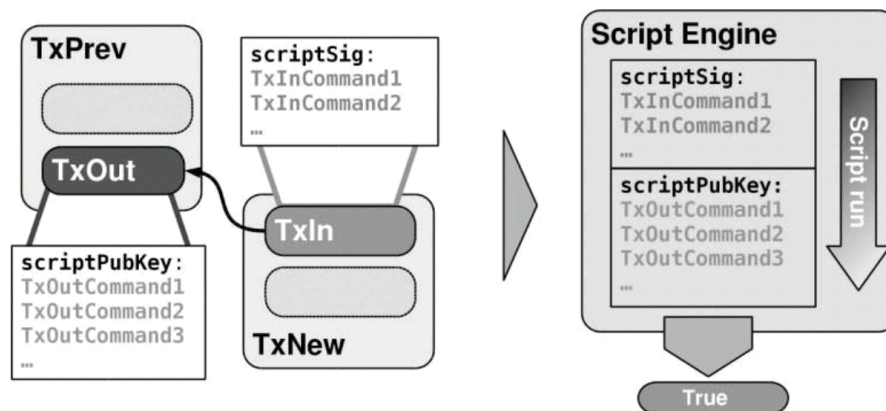


Figure 1: Each `TxIn` references the `TxOut` of a previous transaction, and a `TxIn` is only valid if its `scriptSig` outputs `True` when prepended to the `TxOut`'s `scriptPubKey`.

While you will not be required to write any Python code yourself aside from setting some

parameters and implementing the `scriptPubKey` and `scriptSig` scripts, an understanding of the surrounding code and the terminology that is used will be useful.

Bitcoin transactions are fundamentally a list of outputs, each of which is associated with an amount of bitcoin that is “locked” with a puzzle in the form of a program called a `scriptPubKey` (also sometimes called a “smart contract”), and a list of inputs, each of which references an output of an existing transaction and includes the “answer” to that output’s puzzle in the form of a program called a `scriptSig`. Validating a `scriptSig` consists of appending the associated `scriptPubKey` to it, running the combined script and ensuring that it outputs `True`. Most transactions are “PayToPublicKeyHash” or “P2PKH” transactions, where the `scriptSig` is a list of the recipient’s public key and signature, and the `scriptPubKey` performs cryptographic checks on those values to ensure that the public key hashes to the recipient’s bitcoin address and the signature is valid.

Each transaction input is referred to as a `TxIn`, and each transaction output is referred to as a `TxOut`. The situation for a transaction with a single input and single output is summarized by Figure 1.

The sum of the bitcoin in the inputs to a transaction must not exceed the sum of the outputs for the transaction to be valid, and the difference between the total input and total output is implicitly taken to be a transaction fee, as a miner can modify a received transaction and add an output to their address to make up the difference before including it in a block. For this project, all transactions you create will consume one input and create one `PayToPublicKeyHash` output that sends an amount of bitcoin back to the testnet faucet. For each exercise, you will want to take the fee into account when specifying how much to send and subtract a bit from the amount in the output you’re sending from, say .001 BTC. **If you don’t include a fee, it’s likely that your transaction will never be added to the blockchain, and since BlockCypher will delete transactions that remain unconfirmed after a day or two it’s very important that you include a fee and make sure that your transactions are eventually confirmed.**

1.3 Script Opcodes

The opcodes of the Bitcoin stack machine are documented on the Bitcoin wiki [1], and when composing programs for your transactions’ `scriptPubKeys` and `scriptSigs` you may specify opcodes by using their names verbatim. For example, below is an example of a function that returns a `scriptPubKey` that cannot be spent, but rather acts as storage space for an arbitrary piece of data that someone may want to save to the blockchain using the `OP_RETURN` opcode.

```
def save_message_scriptPubKey(message):
    return [OP_RETURN,
            message]
```

Examples of some opcodes that you will likely be making use of include `OP_DUP`, `OP_CHECKSIG`, `OP_EQUALVERIFY`, and `OP_CHECKMULTISIG`, but you will end up using additional ones as well.

2 Getting started

1. Download the starter code from the course website, navigate to the directory and run `pip install -r requirements.txt` to install the required dependencies. Make sure that you are using Python 3.

2. Make sure you've understood the structure of Bitcoin transactions and read the references in the Recommended Reading section if you would like more information.
3. Implement code for the exercises below, using the Bitcoin test network (testnet) to test your code (as well as offline testing). You can obtain testnet coins for free from <https://coinfaucet.eu/en/btc-testnet/>. It is courteous to send the testnet coins back to the faucet after you are done experimenting with them, and each exercise ends with returning the coins to the faucet.
4. You must implement each transaction specified in the exercises below by writing a scriptPubKey script that locks a certain amount of bitcoin as part of a first transaction as well as a scriptSig script that redeems the first transaction and sends it back to the faucet. You will not receive credit unless you explicitly write your scripts at the opcode level (no calling python-bitcoinlib functions to do it for you).
5. You can use the transaction hashes to track your transactions on a block explorer tool such as <https://live.blockcypher.com/btc-testnet/>.

3 Setup

1. Generate a testnet private key and address with `keygen.py`. Copy and paste the private key in the appropriate place in `config.py`.
2. Go to the faucet, paste in the address, and get some testnet BTC (note that faucets will often rate-limit requests for coins based on Bitcoin address and IP address, so try not to lose your bitcoin too often). ~~Copy and paste the faucet's address into the appropriate place in config.py and note~~ Note the transaction hash the faucet provides, as you will need it for the next step. Viewing the transaction in a block explorer will also let you know which output of the transaction corresponds to your address, and you will need this information for the next step as well.
3. The faucet will give you one spendable output, but we would like to have multiple outputs to spend, at least 3 per exercise and preferably more in case we accidentally lock some with invalid scripts. Edit the parameters at the bottom of `split_test_coins.py`, where `txid` is the transaction hash of the faucet transaction from the previous step, `utxo_id` is 0 if your output was first in the faucet transaction and 1 if it was second, and `n` is the number of outputs you want your test coins split evenly into, and run the program with `python split_test_coins.py`. A perfect run through this assignment would require `n = 3`, one for each exercise, but if you anticipate accidentally locking an output due to a faulty script a couple times per exercise then you might want to set `n` to something higher like 8 so that you don't have to wait to access the faucet again or have to try with a different Bitcoin address.
4. If it's successful, you should get back some information about the transaction. Note the transaction hash, as each exercise will be spending an output from this transaction and will refer to it using this hash.

4 Exercises

To publish each transaction created for the exercises, edit the parameters at the bottom of the file to specify which transaction output the solution should be run with along with the amount to send in the transaction. Note the transaction hash of the created transaction and write it to `transactions.py`. If the scripts you write aren't valid, an exception will be thrown before they're published.

After completing each exercise, look up the transaction hash in a blockchain explorer to verify whether the transaction was picked up by the network. Make sure that all your transactions have been posted successfully before submitting their hashes.

Exercise 1. Open `ex1.py` and complete the scripts labelled with TODOs to redeem an output you own and send it back to the faucet with a standard PayToPublicKeyHash transaction.

Exercise 2. (a) Generate a transaction that can be redeemed by the solution (x, y) to the following system of two linear equations:

$$x + y = (\text{first half of your suid}) \quad \text{and} \quad x - y = (\text{second half of your suid})$$

[to ensure that an integer solution exists, please change the last digit of the two numbers on the right hand side so the numbers are both even or both odd]. (b) Redeem the transaction. The redemption script should be as small as possible. That is, a valid scriptSig should consist of simply pushing two integers x and y to the stack. Make sure you use OP_ADD and OP_SUB in your scriptPubKey.

Exercise 3. (a) Generate a multi-sig transaction involving four parties such that the transaction can be redeemed by the first party (bank) combined with any one of the 3 others (customers) but not by only the customers or only the bank. **You may assume the role of the bank for this problem so that the bank's private key is your private key and the bank's public key is your public key. Generate the customers' keys using `keygen.py` and paste them in `ex3a.py`.** (b) Redeem the transaction and make sure that the scriptPubKey is as small as possible. You can use any legal combination of signatures to redeem the transaction but make sure that all combinations would have worked.

5 Submitting your code

For all exercises, submit the source code as well as the transaction hashes. Your transaction hashes should be in a file called `transactions.py`. and listed one per line in the same order as the exercises. Please create a single .tar or .zip file that includes all your deliverables for all three exercises. Submit via Gradescope as explained at the top of this document.

6 Recommended Reading

1. Bitcoin Script: <https://en.bitcoin.it/wiki/Script>
2. Bitcoin Transaction Format: <https://en.bitcoin.it/wiki/Transaction>
3. <https://privatekeys.org/2018/04/17/anatomy-of-a-bitcoin-transaction/>