CS 251: Bitcoin and Cryptocurrencies Fall 2016

Project 1

due: 2016-10-19 23:59 via email to cs251ta@cs.stanford.edu

Introduction

In this assignment you'll create several transactions and post them to the Bitcoin blockchain. We will provide starter code for this using `bitcoinj`, a free and popular Java library for interacting with Bitcoin. You are free to create and post the transactions using an alternate library and language if you want, but you must submit your code in any case.

Getting started

1. Download the starter code from the course website and and import it into your favorite IDE. You can use maven to download the required dependencies.
2. Familiarize yourself with Bitcoin's scripting system.
3. Peruse the `bitcoinj` API and the starter code. You should check out the `ScriptTransaction` class and the example in `PayToPubKey`.
4. 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 http://tpfaucet.appspot.com/. It is courteous to send the testnet coins back to the faucet after you are done experimenting with them.
5. You must implement the transactions by specifying the Scripts in the specific subclasses. You will not receive credit if you create the transactions using a different tool. We will test your implementation.
6. Email the TA list (cs251ta@cs) to receive some real bitcoin that you can play around with. In your email, please provide a Bitcoin address that you own. You can generate this address by running the `printAddress` unit test.
7. You can use the transaction hashes to track your transactions on a block explorer tool such as https://test-insight.bitpay.com/ (testnet) or https://insight.bitpay.com/ (mainnet).
8. **Important**: The transaction for exercise 1 should be done on the **mainnet.** The transactions for exercises 2 and 3 may be done on the **testnet** (you can also do them on mainnet if you want, but you will need to submit them directly to a mining pool which allows non-standard transactions).

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 "README" 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 and email it to the address at the top of this page. The title of your email should be "Project 1 Submission".

Exercises

1. Generate an address whose standard Base58Check representation starts with 1 and then at least the first four letters of your surname in lowercase. If your surname is shorter than four letters, please append as many 'X' characters as necessary. If it contains an 'l' please use 'L' instead as the 'l' is dropped in Base58Check to avoid confusion with '1'. You may generate this address either using `bitcoinj` or using an external generator. Send some bitcoins to this address using a standard Pay2PubKeyHash transaction and then redeem them.

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

   x+y = (first half of your suid) and x-y = (second half or 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]
   Create and redeem the transaction. The redemption script should be as small as possible. That is, a valid script sig should consist of simply pushing two integers *x* and *y* to the stack. Make sure you use OP_ADD and OP_SUB in your script.

3. 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. Create and redeem the transaction and make sure that the script 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.

Notes

1. The starter code has the tests in ScriptTests.java commented out. Once you have implemented a transaction type, uncomment the corresponding test and run it.
2. When running the tests, be careful that you set `useMainNet` to the appropriate value depending on whether you want to put your transaction on mainnet or testnet.
3. There may be times when your code will hit an error but the process will not terminate. This will prevent you from re-running your code because the old process will have a lock on your wallet. To fix this you need to manually kill the process. On most UNIX platforms, the following command should kill all of your java processes that contain the string "Project1" in their arguments:

   ```
   ps -xo 'pid,command' | grep -E '^[0-9]+ [^ ]*/java .*\bProject1\b' | cut -d ' ' -f 1 | xargs kill -9
   ```
4. To generate a public address to which the TAs can send you some bitcoin, you can use the `printAddress` unit test. In particular, don't ask the TAs to send bitcoin to your vanity address.
5. It sometimes happens that the transactions generated by the unit tests don't make it out onto the Bitcoin network (or onto testnet). After running a test, look up the transaction hash in a blockchain explorer to verify whether the transaction was picked up by the network. If it was, you should see it on a sight like insight.bitpay.com within a few minutes. If you think your transaction didn't make it onto the network, you can post the transaction data manually using the "broadcast transaction" feature at the bottom of the page. Make sure that all your transactions have been posted successfully before submitting their hashes.