

Programming Project #2

Due: 11:59pm on Mon., **Oct. 29, 2018**

Submit via Gradescope code: **9RZGVZ**

In this assignment you will create a transaction called a cross-chain atomic swap, allowing two entities to securely trade ownership over cryptocurrencies on different blockchains. We will provide starter code for this using python-bitcoinlib, a free, low-level Python 3 library for manipulating Bitcoin transactions.

1 Introduction

1.1 Cross-chain Atomic Swap

In this assignment you will implement key parts of the code for a cross-chain atomic swap between two parties, Alice and Bob. Alice has bitcoin on BTC Testnet3, the standard Bitcoin testnet that Project 1 used. Bob has bitcoin on BCY Testnet, Blockcypher's Bitcoin testnet which is mined and maintained exclusively by Blockcypher. They want to trade ownership of their respective coins securely, something that can't be done with a simple transaction because they are on different blockchains.

The idea here is to set up transactions around a secret x , that only one party (Alice) knows. In these transactions only $H(x)$ will be published, leaving x secret. Transactions will be set up in such a way that once x is revealed, both parties can redeem the coins sent by the other party. If x is never revealed, both parties will be able to retrieve their original coins safely, without help from the other.

This method also works between other cryptocurrencies and altcoins, for example trading Bitcoin for Litecoin.

1.2 How does it work?

Please refer to the Bitcoin wiki page on Atomic cross-chain trading:

https://en.bitcoin.it/wiki/Atomic_cross-chain_trading.

Please read the algorithm describing on a high level how the algorithm works to better understand what we'll be doing for this assignment.

2 Setup

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. (a) Create BTC testnet keys for Alice and Bob. You can use `keygen.py` and place it into `keys.py`. Please make sure to create different keys for Alice and Bob, you wouldn't want

them to be able to forge each others' transactions!

- (b) Give Alice's address bitcoin on BTC testnet3. You can use the same faucet from Project 1, <https://coinfaucet.eu/en/btc-testnet/>.
- 3. (a) Sign up for an account with Blockcypher to get an API token here: <https://accounts.blockcypher.com/>.
(b) Create BCY testnet keys for Alice and Bob and place into keys.py.

```
curl -X POST https://api.blockcypher.com/v1/bcy/test/addr?token=$YOURTOKEN
```


(c) Give Bob's address bitcoin on the Blockcypher testnet (BCY).

```
curl -d '{"address": "BOBS_BCY_ADDRESS", "amount": 1000000}' \
https://api.blockcypher.com/v1/bcy/test/faucet?token=$YOURTOKEN
```
- 4. Split the test coins by using split_test_coins.py (filling out the relevant fields in the file). This is a good way to test that you can send bitcoin on both blockchains!
- 5. Fill in the variables in swap.py.
- 6. Read swap.py, alice.py, and bob.py. Compare to the pseudocode in https://en.bitcoin.it/wiki/Atomic_cross-chain_trading. This will be very helpful in understanding this assignment.

3 Submitting your code

Please submit all code for this assignment. Make sure design_doc.txt is filled out and your code verifies when run with broadcast_transactions=False. Please create a single .tar or .zip file that includes all your deliverables for all three exercises. Submit via Gradescope.

4 Exercises

Exercise 1. Consider the ScriptPubKey necessary for creating a transaction necessary for a cross-chain atomic swap. This transaction must be redeemable by the recipient (if they have a secret x that corresponds to $\text{Hash}(x)$), or redeemable with signatures from both the sender and the recipient.

Write this ScriptPubKey in coinExchangeScript in swap_scripts.py.

Exercise 2. Write the accompanying ScriptSigs:

- (a) Write the ScriptSig necessary to redeem the transaction in the case where the recipient knows the secret x . Write this in coinExchangeScriptSig1 in swap_scripts.py.
- (b) Write the ScriptSig necessary to redeem the transaction in the case where both the sender and the recipient sign the transaction. Write this in coinExchangeScriptSig2 in swap_scripts.py.

Exercise 3. Run your code! We aren't requiring that the transactions be broadcasted, as that requires some waiting to validate transactions. Running with broadcast_transactions=False

will validate that `ScriptSig + ScriptPK` return true. Try this for `alice_redeems=True` as well as `alice_redeems=False`.

OPTIONAL: Try with `broadcast_transactions=True`, which will make the code sleep for an appropriate amount of time to post everything to the blockchain and verify correctly. Warning: will take 20-60 minutes to run.

Exercise 4. Fill in `design.doc.txt`. Write a short (1-3 paragraph) design document about this project. Please include the following:

- (a) An explanation of what you wrote and how the `coinExchangeScript` work.
- (b) Briefly, how the `coinExchangeScript` you wrote fits into the bigger picture of this atomic swap.
- (c) Consider the case of Alice sending coins to Bob with `coinExchangeScript`:
Why can Alice always get her money back if Bob doesn't redeem it?
Why can't this be solved with a simple 1-of-2 multisig?