Programming Homework 2

Foundations of Blockchain Technology and cryptocurrencies

Fall 2019

Sharif Blockchain Lab
Sharif University of Technology
Department of Electrical Engineering

Deadline: 1st Day 23:55

- Read each problem completely before coding. Provide exactly what the questions ask for.
- Study Bitcoin scripts and their details thoroughly before starting[1]. Working with the scripts, even with the tools provided is challenging and any small mistake can mislead you for hours. Since the blocktime is 10 minutes, it will take you long to see the result and debug your code.
- Problems 1 and 2 have a base code provided. All you need to do is complete the **TODO** sections.
- In problems 3 and 4, you need to provide a solution to a dilemma and restructure the code from problem 2.
- In problem 5, you will write a script in Python using all the knowledge you gained in the last 4 parts.
- You will submit your codes and also your transaction IDs for each problem.
- If you have difficulties in submitting your answers, ask your questions in **Quera**.

[1] You can use the lecture notes or this link.

Getting Started:

Before we begin, let us go over a few things first:

- To use the given code, either use PyCharm and automatically install the requirements or run **pip install -r requirements.txt** to do so. Note that the codes are only compatible with **Python3**.
- All codes must be in the same folder, including the codes you will add. Otherwise they won't work. The codes provided are already in the same folder, But when you add your own files, take this into consideration.
- We are going to submit transactions on the Bitcoin network and learn the Bitcoin script along the way. Since Bitcoins are a bit costly, we will be working with the testnet[2] instead.
- First we are going to generate key pairs for you, Alice and Bob on the Bitcoin Testnet. Using lib/keygen.py, generate private keys for my_private_key, alice_secret_key_BTC and bob_secret_key_BTC, and record these keys in lib/config.py. Note that Alice and Bob's keys will only come into play for question 6. Please make sure to create different keys for Alice and Bob, you wouldn't want them to be able to forge each others' transactions!
- Next, we want to get some test coins for my_private_key and alice_secret_key_BTC. To do so:
  1. Go to the Bitcoin Testnet faucet **this faucet**[3] and paste in the corresponding addresses of the users. Note that faucets will often rate-limit requests for coins based on Bitcoin address and IP address, so try not to lose your test Bitcoin too often.
  2. Record the transaction hash the faucet provides, as you will need it for the next step. Viewing the transaction in a block explorer **blockcypher** will also let you know which output of the transaction corresponds to your address, and you will need this utxo index for the next step as well.
- The **faucet** doesn't allow you to try too many times and it might block or delay your Bitcoin address and/or IP address. Since you are going to need a lot of TX outputs for the following parts, we suggest you use **split_test_coins.py** to split the given output to many parts. A perfect run needs around 15 outputs but we suggest 30 to give room for error. Split Alice's coins as well.
- Next, we are going to create generate key pairs for Alice and Bob on the BlockCypher testnet.
  1. Sign up for an account with Blockcypher to get an API token here.
  2. Create BCY testnet keys for Alice and Bob and place into lib/config.py.(run this command in cmd!)

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

[2]The testnet is a network that functions similar to the mainnet of Bitcoin, but the bitcoins in it are free. It's purpose is to help developers test their codes and play around with their tools before trying them out on the mainnet. You can read more about it here

[3]A faucet is a website that gives free testnet coins to developers.

- Give Bob's address bitcoin on the Blockcypher testnet (BCY) to get some funds.
  curl -d '{"address": "BOBS_BCY_ADDRESS", "amount": 1000000}'
  https://api.blockcypher.com/v1/bcy/test/faucet?token=$YOURTOKEN
  Note: if you are using windows cmd use this code instead
  curl -d "{\"address\": \"BOBS_BCY_ADDRESS\", \"amount\": 1000000}"
  https://api.blockcypher.com/v1/bcy/test/faucet?token=$YOURTOKEN

- Let's also split Bob's coins using split_test_coins.py. Make sure to edit the parameters at the bottom of the file. Each time you are switching between the Bitcoin and BlockCypher testnets, make sure to visit lib/config.py and adjust the network type variable. Make sure to record the transaction hash.

- Be very careful about uppercase and lowercase letters. Python is case sensitive and so is the framework where the codes are prepared upon.

Acknowledgment

The code provided is based on a course project code in CS251 Stanford. We like to thank the instructors and their assistants, whose endeavors made the preparation of this homework easier.

Problem 1 - Standard Bitcoin Script (50 points)

To start, we are going to write a standard transaction script that is used everywhere. Major parts of the code is already provided in **ex1.py**. Complete the **TODO** parts and run the code. You will get the transaction data. Copy the transaction hash and use it to find the transaction in **blockcypher**. If it is confirmed, copy this hash to **transactions.py**. You will repeat similar steps in the next problems as well.

Hint: Just put the appropriate script commands and variables in **return** part of the function. For example:

return [OP_DUP,address,OP_HASH160,OP_EQUALVERIFY] (This is not the right answer!!)

Problem 2 - Linear Redeemer (100 points)

A student has suggested his own protocol to redeem an output. Redeeming will happen when the solution (x, y) to the following set of linear equations have been found and submitted in the ScriptSig:

$$x + y = the\ first\ half\ of\ your\ studentID$$

$$x - y = the\ second\ half\ of\ your\ studentID$$

(If x and y turn out to be fractional, add or subtract one value from your Student ID to make sure the solution is integer.)

In this part, you will edit **ex2a.py** and **ex2b.py**, where in **ex2a.py** you will make a transaction that can only be redeemed with the solution and in **ex2b.py**, you will redeem it to verify that your script works. Your scripts should be as small as possible. Note that the ScriptSig must only consist of pushing x and y to the stack. Don't forget to add the hashes of the produced transactions to **transactions.py**.

Problem 3 - Bitcoin company (200 points)

Faraz and Ata want to start a company together. They have decided to put their companies funds in the Bitcoin blockchain to invest in cryptocoins and also show how up to date they are. They decided to use the MULTISIG feature of the Bitcoin script, such that if both sides agree, the transaction is confirmed.
After some years the company grew and now has 5 other shareholders. They changed their policy and now each transaction is confirmed if:
i. Faraz and Ata both agree on the transaction.
ii. Either Faraz or Ata and 3 other shareholders agree on the transaction.(note that shareholders can not spend the funds without agreeing with Ata or Faraz)

1.Using conditional branches (OP_IF) implement a script code for the company's policy by creating two transactions for the idea In this solution, the first one creates an output that can only be redeemed by the way you proposed. The second transaction redeems that output, with the method you proposed, to verify your idea works.

2.Now suppose Faraz and Ata are having problems and never can agree on a transaction (case **i** never happens); implement a script code for the case **ii** without using conditional branches (OP_IF).

Copy the code from **ex2a.py** to **ex31a.py** and **ex32a.py** and copy the content of **ex2b.py** to **ex31b.py** and **ex32b.py**. Modify the codes to make the two transactions.
The modification may require you to change parts beyond than the **TODO** parts, possibly even functions that are defined in **utils.py**. Don't change **utils.py** though. Copy the function declaration to the new files and proceed from there.
Don't forget to copy the transaction hashes to **transactions.py**.

Problem 4 - Happy Birthday (250 points)

1. Uncle Saeed decided to give a birthday present for his nephew, Hamed, so when his birthday comes, he can use the money. After getting introduced to Bitcoins, he has become extremely fond of this technology and decided to put the funds in the Bitcoin blockchain. Propose a way he can make sure Hamed can't use the bitcoins sent to her address before his birthday, even if he has the private key.

After this, copy the code from **ex2a.py** and **ex2b.py** to **ex4a.py** and **ex4b.py** respectively. Modify the codes to make two transactions. One whose output cannot be redeemed until a specific time in the future. One who redeems that output after the specified time. Again, like before, write the hashes of these transactions in **transcations.py**

The modification might mean you have to change codes further than the **TODO** parts, possibly even functions that are defined in **utils.py**. Don't change **utils.py** though. Copy the function declaration to **ex4a.py** or **ex4b.py** and proceed from there.(150 points)

2. His other uncle, Homayoun, wants to write a happy birthday letter to Hamed. He found out that you can put messages on bitcoin's blockchain and they last for as long as bitcoin lasts. He really liked the idea and decided to put the happy birthday message on bitcoin.

Write a python code (**happybirthday.py**) to help Homayoun by getting a costume message as an input and putting it on bitcoin's blockchain. Try your code with 'Happy Birthday Hamed' and save the transaction hash in **transactions.py**

Problem 5 - I have it (200 points)

Salar wrote an incredible article recently and decided to have it published in Nature in a few months. Mohammad told him that if he proves he has the article right now, Mohammad will give him 400$. Salar has known Mohammad for a long time and knows that if he gives the article to him, he will publish it with his own name. Yet he still wants that 400$. Easy enough, he computes the hash of the article he wrote and gives it to MOhammad, and tells him to check if the hash is valid when it is published. Mohammad doesn't accept this and says he is not organized enough to keep the hash for a few months and find it at that time and will probably lose the hash. Salar knows Mohammad likes Bitcoins. He also wants the 400$. Salar thought of this solution:

- Salar uses the $SHA256$ hash of the file to construct an address (using the algorithm that can generate an address based on public keys) and sends 1 satoshi to it

Part 1 - 100 points

Write a Python scripts (**fileverify.py**) who take an input file and implement the two ideas. Try your script with the file **data.hex**. Save the transaction hashes in **transactions.py**.

Part 2 - 100 points

Now, If there are 10 articles and Salar needs to prove he has written all of them beforehand, what should he do? Clearly he can do the same steps 10 times. Can you suggest an easier, better and cheaper solution?
Explain your idea and implement it in a script called **multifileverify.py**, which takes n, the number of files and then takes n files and using your idea, it achieves Parsa's goal. You don't need to test it, but make sure it works.

Problem 6 - Atomic Swap (300 points)

Last but not least, you will create a transaction called a cross-chain atomic swap, allowing two entities to securely trade ownership over cryptocurrencies on different blockchains. In this case, Alice and Bob will swap coins between the Bitcoin testnet and BlockCypher testnet. As you recall from setup, Alice has bitcoin on BTC Testnet3, and Bob has bitcoin on BCY Testnet. 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. Before you start, make sure to read swap.py, alice.py, and bob.py. Compare to the pseudocode in here(open your vpn :D). This will be very helpful in understanding this assignment. Note that for this question, you will only be editing ex6.py and you can test your code by running python swap.py.

1. Consider the ScriptPubKey required 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 Hash(x)), or redeemable with signatures from both the sender and the recipient. Write this ScriptPubKey in coinExchangeScript in ex6.py.

2. Write the accompanying ScriptSigs:

   (a) Write the ScriptSig necessary to redeem the transaction in the case where the re- cipient knows the secret x. Write this in coinExchangeScriptSig1 in ex6.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 ex6.py.

3. Run your code using python swap.py. We aren't requiring that the transactions be broad- casted, as that requires some waiting to validate transactions. Running with broad- cast transactions=False will validate that ScriptSig + ScriptPK return true. Try this for alice redeems=True as well as alice redeems=False.

Submission

You must submit the files, compressed in one zip file named HW2_STID.zip:

- config.py
- ex1.py
- ex2a.py
- ex2b.py
- ex31a.py
- ex31b.py
- ex32a.py
- ex32b.py
- ex4a.py
- ex4b.py
- happybirthday.py
- fileverify.py
- multifileverify.py
- ex6.py
- swap.py
- alice.py
- bob.py
- transactions.py
- For problems 3 to 6, explain your ideas in a file and send the PDF file as report.pdf (explain your ideas for 1uestion 5 and 6 with details)