EE576 Cybersecurity: Homework 5 (100 points)

You can team up with at most one other student in the class to complete this homework.

1. (10 points) Read the paper "Blockchain and Smart Contracts for Internet of Things" assigned in the course module. Identify two differences on the use of blockchain in Bitcoin versus in the IoT domain.

2. The goal of the following problem is to help you to learn Bitcoin Script in writing basic transactions and simple smart contracts. This assignment is based on a similar assignment from a graduate-level course on bitcoin and cryptocurrencies in Stanford.

Resources:

1. You should be familiar with Bitcoin script. Its basic structure was covered in class and chapter 3 of the book "Bitcoin and Cryptocurrency Technologies". Full documentation can be found here.
2. A skeleton code in Python has been provided with this document. You will need at least Python 3 to run all the scripts correctly. Please consult http://learn.perl.org/installing/ to install Python on your own machine. If you do not know Python, you might read any Python tutorial available online such as the one at http://scipy-lectures.org/language/python_language.html to familiarize with the basic syntax.
3. Download the skeleton code archive and extract all the code to a directory. Navigate to that directory and run `pip install -r requirements.txt` to install the required dependencies.
4. For this homework, you will test your code using the Bitcoin test network ("testnet").

Exercises (you will need to provide a written answer for all underlined parts):

1. (3 points) Generate a Testnet private key and address with `keygen.py`. Copy and paste the private key and address in the appropriate place in `config.py`. Explain how the private key and address are used in Bitcoin. How do you know that this is an address for the Testnet blockchain but not the production Bitcoin blockchain?
2. Go to the Testnet faucet https://coinfaucet.eu/en/btc-testnet/ to obtain coins for free. Note that faucets will often rate-limit requests for coins based on Bitcoin and IP addresses, so try not to lose your bitcoin too often. Also, 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.
3. (3 points) Write down the transaction hash and the faucet address as you will need it for the next step. Go to a block explorer such as https://live.blockcypher.com/ to see the transaction that corresponds to this hash. How much BTC did you get and what is the transaction fee?
4. (3 points) The faucet will give you one spendable output, but we would like to have multiple outputs to spend 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_to_send` is the transaction hash of the faucet transaction from the previous step, `utxo_index` 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 (say 8), and `amount_to_send` is the amount of BTC you have minus a fee, say 0.001 BTC. If you don't include a fee, it's likely that your transaction will never be added to the blockchain. Run the program `split_test_coins.py`. Cut and

paste the bitcoin transaction in your report and provide a short description of what are in that transactions. Also, confirm that this transaction is posted with a block explorer.

5. In this part, you will create several transactions and post them to the Bitcoin Testnet blockchain. To publish each transaction created for the exercise, 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. If the scripts you write aren't valid, an exception will be thrown before they're published. Also make sure your transaction is posted by checking with the block explorer. 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 and also upload a separate PDF report containing the answers for every part.
    i. 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.
    ii. (a) 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 the larger student id in your team

        and

        x - y = second half of the larger student id in your team

        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 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.

    iii. (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 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.

For each of the 3 exercises in part 5, you will receive 10 points for the correct idea as explained in the report, 10 points for a working python execution and 7 points for providing evidence of a posted transaction through a block explorer.