Blockchain Technology Project - SUTDcoin

Group: GOLD EXPERIENCE
Members: Cheng Huan An, Kenneth Sim Jian Hui, Noh Youngmin, Tan Xiang Hao

To set up the environment:

```
python3 -m venv venv
source venv/bin/activate
pip3 install wheel
pip3 install -r requirements.txt
```

Usage

All demonstrations can either be done locally or across multiple computers.

Automated local deployment

"build_local_automation.py" allows for quick local deployment and demonstration of the SUTDCoin network. It will run as many instances as there are in "ports_miner.txt" for miners and "ports_spv.txt" for SPV Clients. The formatting of both files will be touched on later in the report.

It runs both "miner_manage.py" and "spv_client.py" with pre-set arguments, some of which are taken from the "ports_*.txt" files.

Command	Demo
`python3 build_local_automation.py`	Multiple honest miners
`python3 build_local_automation.py -s 1`	Selfish mining demo
`python3 build_local_automation.py -d 1`	Double-spending demo

ports_*.txt format

Both ports_miner.txt and ports_spv.txt have identical formats.

The format in the current repo is as follows:
<port_number>\t<private_key>\t<public_key>\n

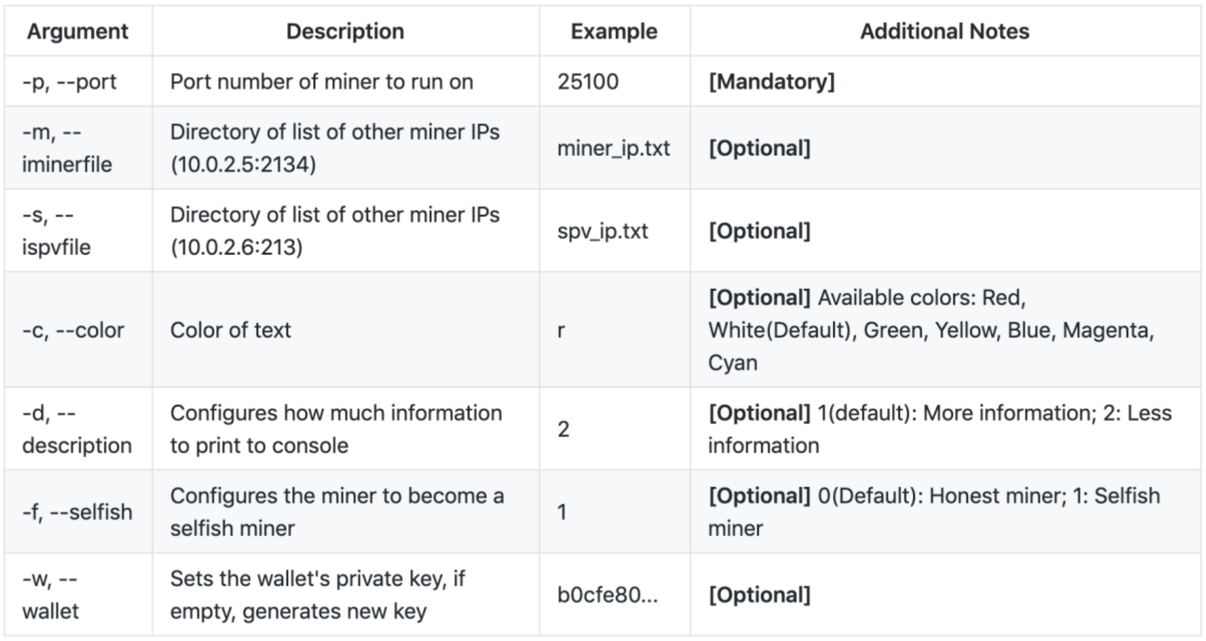
- <port_number> field is mandatory, and the code will run as many instances as there are ports in the file.
- <private_key> field is not mandatory, as the "miner_manage.py" and "spv_client.py" files will generate their own private keys when no input is detected.
- <public_key> field is not used by any of the codes, and is more of a reference for the user for testing.

Do ensure that there is a 'tab character' in between each field, as some IDEs might do 4 spaces.

Network deployment

There are two kinds of clients to be deployed, miners and SPV clients, using “miner_manage.py” and “spv_client.py” respectively.

Miner - “miner_manage.py”

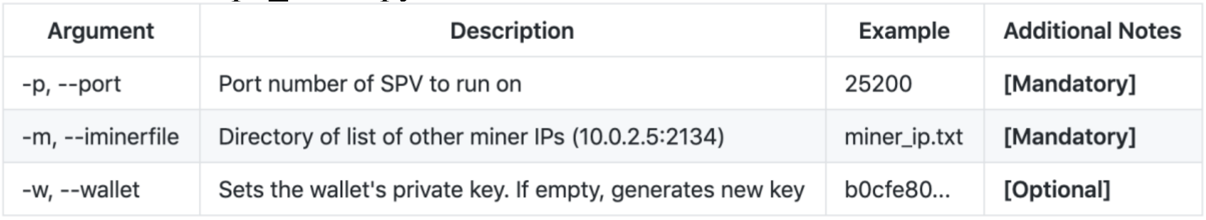


Argument	Description	Example	Additional Notes
-p, --port	Port number of miner to run on	25100	**[Mandatory]**
-m, --iminerfile	Directory of list of other miner IPs (10.0.2.5:2134)	miner_ip.txt	**[Optional]**
-s, --ispvfile	Directory of list of other miner IPs (10.0.2.6:213)	spv_ip.txt	**[Optional]**
-c, --color	Color of text	r	**[Optional]** Available colors: Red, White(Default), Green, Yellow, Blue, Magenta, Cyan
-d, --description	Configures how much information to print to console	2	**[Optional]** 1(default): More information; 2: Less information
-f, --selfish	Configures the miner to become a selfish miner	1	**[Optional]** 0(Default): Honest miner; 1: Selfish miner
-w, --wallet	Sets the wallet's private key, if empty, generates new key	b0cfe80...	**[Optional]**

Sample start up:

- miner_manage.py -p 2200
- miner_manage.py -p 1500 -m miner_ip.txt -c g -s spv_ip.txt -d 2 -w b0cfe80dbda0d882b6d517321b3eb3343c48864ad097c5df
- miner_manage.py -p 1200 -m miner_ip.txt -c r -f 1

SPVClient - “spv_client.py”



Argument	Description	Example	Additional Notes
-p, --port	Port number of SPV to run on	25200	**[Mandatory]**
-m, --iminerfile	Directory of list of other miner IPs (10.0.2.5:2134)	miner_ip.txt	**[Mandatory]**
-w, --wallet	Sets the wallet's private key. If empty, generates new key	b0cfe80...	**[Optional]**

Sample start up:

- spv_client.py -p 2300 -m miner_ip.txt
- spv_client.py -p 1500 -m miner_ip.txt -w c218953cd1e1ebff4cead74f25420dcffd6239ed1f48796f

Endpoints

1. /request_blockchain_header_hash:
    a. Used by both Miner and SPVClient.
    b. Returns an ordered list of header hashes of the longest chain from genesis block.

2. /request_blockchain
    a. Used by both Miner and SPVClient.
    b. Returns an ordered list of blocks of the longest chain from genesis block.
    c. Miner includes list of ordered transactions for every block, while SPVClient does not.

3. /request_full_blockchain
    a. Used by both Miner and SPVClient.
    b. Returns an unordered list of all blocks within client.
    c. Miner includes list of ordered transactions for every block, while SPVClient does not.

4. /request_block/{header_hash}
    a. Used by both Miner and SPVClient.
    b. Returns full information for that particular block.
    c. Miner includes list of ordered transactions, while SPVClient does not.

5. /account_balance
    a. Used by both Miner and SPVClient.
    b. Returns the amount of coins in the queried SPVClient or Miner's wallet.
    c. Miner will retrieve information locally, while SPVClient will ask a random full node/ Miner.

6. /account_balance/{public_key}
    a. Used by both Miner and SPVClient.
    b. Returns the amount of coins in that particular account.
    c. Miner will retrieve information locally, while SPVClient will ask a random full node/ Miner.

7. /send_transaction?receiver={receiver}&amount={amount}
    a. Used by both Miner and SPVClient.
    b. Submits transactions to the network to be processed.

8. /verify_transaction/{txid}
    a. Used by both Miner and SPVClient.
    b. Returns information about the particular transactions, including number of confirmations.
    c. Miner will retrieve and verify locally, while SPVClient will ask a random full node/ Miner for merkle tree's proof and verify locally with the merkle tree root and header hashes.

9. /verify_transaction_from_spv
    a. Used by Miner only. Not meant to be assessed by the user.
    b. Called by SPVClient to get the merkle tree proof data from Miner.

10. /block

    a. Used by Miner only. Not meant to be assessed by the user.
    b. Called by other Miners, able to receive Block objects as Pickles from other Miners in body.

11. /transaction
    a. Used by Miner only. Not meant to be assessed by the user.
    b. Called by other Miners and SPVClient, able to receive Transactions as json in body.

12. /block_header
    a. Used by SPVClient only. Not meant to be assessed by the user.
    b. Called by Miners, able to receive SPVBlock objects as Pickles from other Miners in body.

Utility

The `utility/` folder includes three files:

1. "generate_private_key.py"

Generates a private/ public key pair. E.g.

Private key: 73389712abd6df649d92e4cf5a49c63582cbfdcee9f38932
Public key:
e86f5e99bfe8095defd9f6d801456f2e38e1f5719e6c060dbf2d1b5d7191a002826c5963de797686bdf4
5a9cbbb25fe2

2. "random_transactions.py"

Generates random transactions at random intervals based on the wallet and port information in "ports_miner.txt" and "ports_spv.txt"

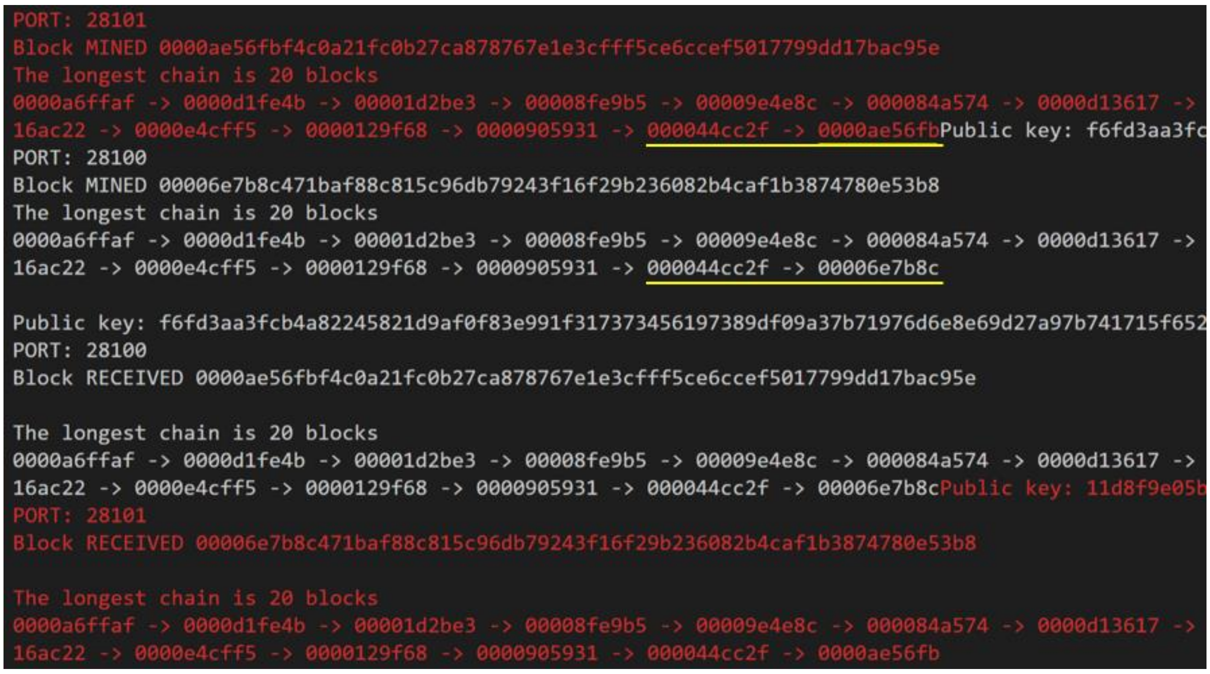
3. "selfish_miner_checker.py"

Sends periodic checks for account balances for the first two entries in "ports_miner.txt", meant to be use in the selfish demo only.

Documentation of displayed features

1. Simulate miners running Nakamoto consensus and making transactions

Implemented features:
- New blocks arrive every few 2-5 seconds, with static difficulty of `00000f`.
- Coinbase transaction of 100 SUTDcoins. Under create_merkle method in "miner.py".
- Transactions occur randomly. See "random_transactions.py".
- Validation checks (no double spending, validated sender, sender must have enough money). See network_block method in "blockchain.py".
- Forks resolved. Example:



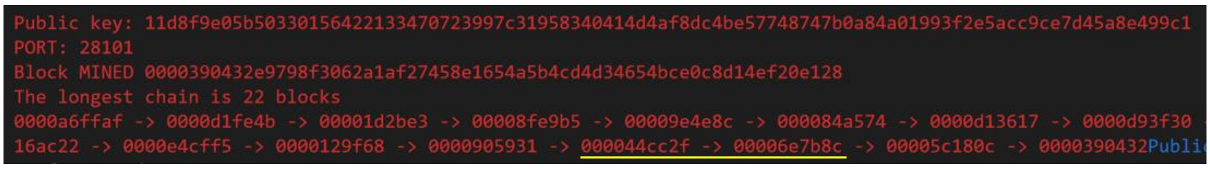
```
PORT: 28101
Block MINED 0000ae56fbf4c0a21fc0b27ca878767e1e3cfff5ce6ccef5017799dd17bac95e
The longest chain is 20 blocks
0000a6ffaf -> 0000d1fe4b -> 00001d2be3 -> 00008fe9b5 -> 00009e4e8c -> 000084a574 -> 0000d13617 ->
16ac22 -> 0000e4cff5 -> 0000129f68 -> 0000905931 -> 000044cc2f -> 0000ae56fbPublic key: f6fd3aa3fc
PORT: 28100
Block MINED 00006e7b8c471baf88c815c96db79243f16f29b236082b4caf1b3874780e53b8
The longest chain is 20 blocks
0000a6ffaf -> 0000d1fe4b -> 00001d2be3 -> 00008fe9b5 -> 00009e4e8c -> 000084a574 -> 0000d13617 ->
16ac22 -> 0000e4cff5 -> 0000129f68 -> 0000905931 -> 000044cc2f -> 00006e7b8c

Public key: f6fd3aa3fcb4a82245821d9af0f83e991f317373456197389df09a37b71976d6e8e69d27a97b741715f652
PORT: 28100
Block RECEIVED 0000ae56fbf4c0a21fc0b27ca878767e1e3cfff5ce6ccef5017799dd17bac95e

The longest chain is 20 blocks
0000a6ffaf -> 0000d1fe4b -> 00001d2be3 -> 00008fe9b5 -> 00009e4e8c -> 000084a574 -> 0000d13617 ->
16ac22 -> 0000e4cff5 -> 0000129f68 -> 0000905931 -> 000044cc2f -> 00006e7b8cPublic key: 11d8f9e05b
PORT: 28101
Block RECEIVED 00006e7b8c471baf88c815c96db79243f16f29b236082b4caf1b3874780e53b8

The longest chain is 20 blocks
0000a6ffaf -> 0000d1fe4b -> 00001d2be3 -> 00008fe9b5 -> 00009e4e8c -> 000084a574 -> 0000d13617 ->
16ac22 -> 0000e4cff5 -> 0000129f68 -> 0000905931 -> 000044cc2f -> 0000ae56fb
```

Red miner originally had the block `0000ae565b` after `000044cc2f` while white miner had the block `00006e7b8c`. The fork is only resolved when one chain becomes longer. The miner(s) with the shorter chain will stop mining on that chain and work on the longer one instead.



```
Public key: 11d8f9e05b5033015642213347072399 7c31958340414d4af8dc4be57748747b0a84a01993f2e5acc9ce7d45a8e499c1
PORT: 28101
Block MINED 0000390432e9798f3062a1af27458e1654a5b4cd4d34654bce0c8d14ef20e128
The longest chain is 22 blocks
0000a6ffaf -> 0000d1fe4b -> 00001d2be3 -> 00008fe9b5 -> 00009e4e8c -> 000084a574 -> 0000d13617 -> 0000d93f30
16ac22 -> 0000e4cff5 -> 0000129f68 -> 0000905931 -> 000044cc2f -> 00006e7b8c -> 00005c180c -> 0000390432Publi
```

In this case, the red miner stopped working on a chain with his original block (`0000ae56fb`) and adopts the longer chain which was built on white's mined block (`00006e7b8c`).

2. Interaction of SPV clients with miners

Implemented features:
- Acts as a wallet, and has both a public and private key.
- Associate key pairs.
- Receive block headers. Block headers are obtained from a separate blockchain.
- Receive and verify transactions.
- Send transactions.

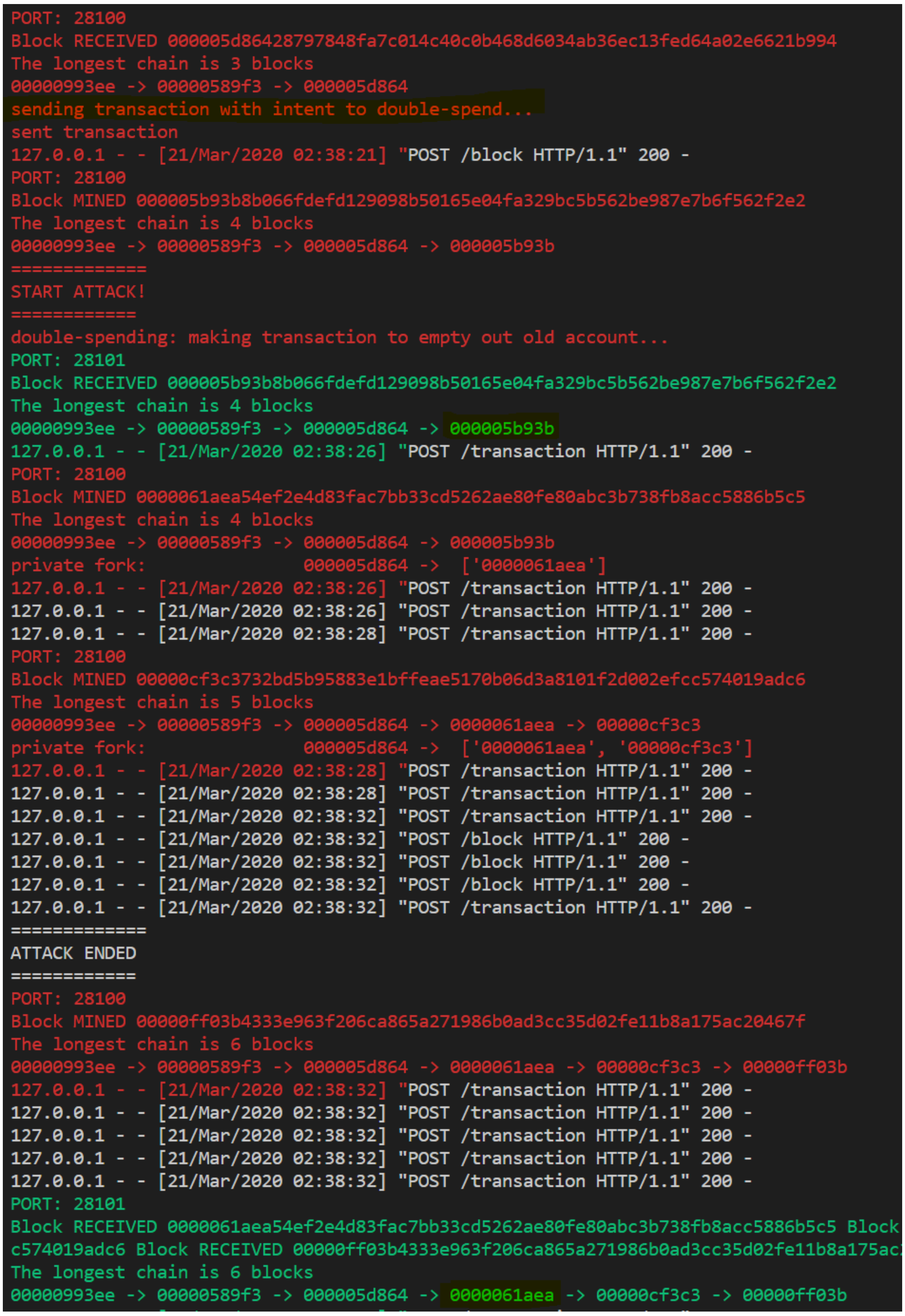
3. Ledger

Implemented features:
- Dictionary that stores public key as keys and account balance as values
- Verifies transactions in the transaction queue
- Updates account values accordingly when all transactions in the queue are verified
- Creates new keys for new recipients

- Rejects transactions when sender does not exist in ledger

4. Double-spending attack
   1) At a specified block in the code, the attacker will send a transaction.
   2) Right after the transaction in (1) is sent, the attacker empties his account by creating a new address and transferring the money to the new account. Subsequent mining will also be carried out under the new address.
   3) When at least one block has been mined since the transaction in (1), the attacker will start to mine blocks with the previous header hash being the block before the one with the transaction we would like to void. The attacker publishes the blocks after three blocks has been mined. If the attack is not successful, the attacker continues mining blocks for his intended fork and publishes them again after 3 blocks. Since attacker has majority hashing power, attacker will eventually overwrite block with bad transaction in (1).

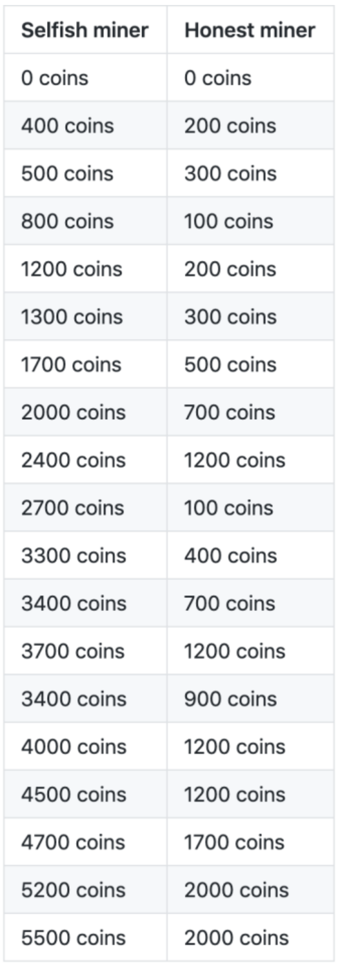
Example output:



```
PORT: 28100
Block RECEIVED 000005d86428797848fa7c014c40c0b468d6034ab36ec13fed64a02e6621b994
The longest chain is 3 blocks
00000993ee -> 00000589f3 -> 000005d864
sending transaction with intent to double-spend...
sent transaction
127.0.0.1 - - [21/Mar/2020 02:38:21] "POST /block HTTP/1.1" 200 -
PORT: 28100
Block MINED 000005b93b8b066fdefd129098b50165e04fa329bc5b562be987e7b6f562f2e2
The longest chain is 4 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 000005b93b
=============
START ATTACK!
============
double-spending: making transaction to empty out old account...
PORT: 28101
Block RECEIVED 000005b93b8b066fdefd129098b50165e04fa329bc5b562be987e7b6f562f2e2
The longest chain is 4 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 000005b93b
127.0.0.1 - - [21/Mar/2020 02:38:26] "POST /transaction HTTP/1.1" 200 -
PORT: 28100
Block MINED 0000061aea54ef2e4d83fac7bb33cd5262ae80fe80abc3b738fb8acc5886b5c5
The longest chain is 4 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 000005b93b
private fork:              000005d864 ->  ['0000061aea']
127.0.0.1 - - [21/Mar/2020 02:38:26] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:26] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:28] "POST /transaction HTTP/1.1" 200 -
PORT: 28100
Block MINED 00000cf3c3732bd5b95883e1bffeae5170b06d3a8101f2d002efcc574019adc6
The longest chain is 5 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 0000061aea -> 00000cf3c3
private fork:              000005d864 ->  ['0000061aea', '00000cf3c3']
127.0.0.1 - - [21/Mar/2020 02:38:28] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:28] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /block HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /block HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /block HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
=============
ATTACK ENDED
============
PORT: 28100
Block MINED 00000ff03b4333e963f206ca865a271986b0ad3cc35d02fe11b8a175ac20467f
The longest chain is 6 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 0000061aea -> 00000cf3c3 -> 00000ff03b
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
127.0.0.1 - - [21/Mar/2020 02:38:32] "POST /transaction HTTP/1.1" 200 -
PORT: 28101
Block RECEIVED 0000061aea54ef2e4d83fac7bb33cd5262ae80fe80abc3b738fb8acc5886b5c5 Block
c574019adc6 Block RECEIVED 00000ff03b4333e963f206ca865a271986b0ad3cc35d02fe11b8a175ac
The longest chain is 6 blocks
00000993ee -> 00000589f3 -> 000005d864 -> 0000061aea -> 00000cf3c3 -> 00000ff03b
```

Block following `000005d864` was originally `000005b93b` but is `0000061ea` after attack.

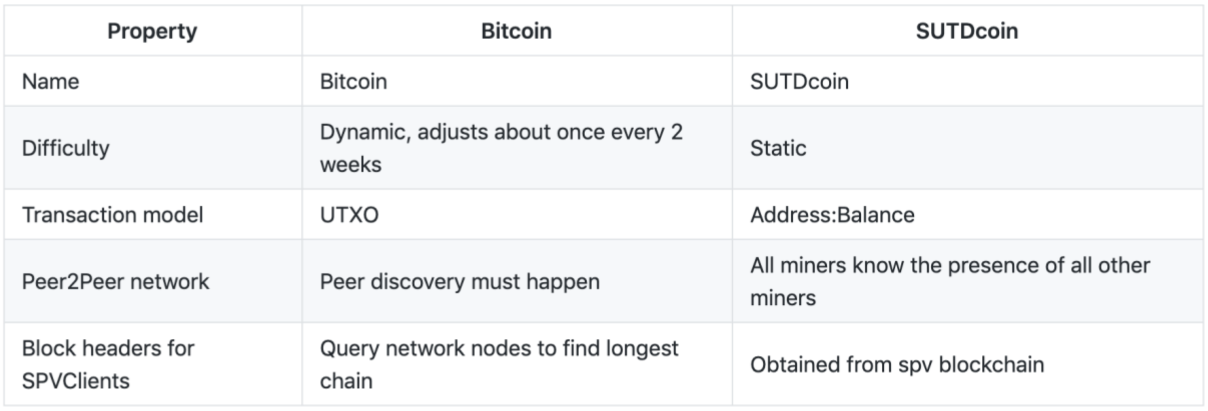
5. <u>Selfish-mining</u>



Selfish miner	Honest miner
0 coins	0 coins
400 coins	200 coins
500 coins	300 coins
800 coins	100 coins
1200 coins	200 coins
1300 coins	300 coins
1700 coins	500 coins
2000 coins	700 coins
2400 coins	1200 coins
2700 coins	100 coins
3300 coins	400 coins
3400 coins	700 coins
3700 coins	1200 coins
3400 coins	900 coins
4000 coins	1200 coins
4500 coins	1200 coins
4700 coins	1700 coins
5200 coins	2000 coins
5500 coins	2000 coins

- The Selfish miner holds n number of blocks after mining before releasing it to the network.
- The Selfish miner will try to mine at least one block faster than the rest of the network before releasing it to the network, this is to force the rest of the miners to switch to its fork.
- In the demo, the Selfish miner will reset its collection when it realises that the other miners have already n+1 blocks, and is unable to catch up.

Major differences between Bitcoin and SUTDcoin



Property	Bitcoin	SUTDcoin
Name	Bitcoin	SUTDcoin
Difficulty	Dynamic, adjusts about once every 2 weeks	Static
Transaction model	UTXO	Address:Balance
Peer2Peer network	Peer discovery must happen	All miners know the presence of all other miners
Block headers for SPVClients	Query network nodes to find longest chain	Obtained from spv blockchain