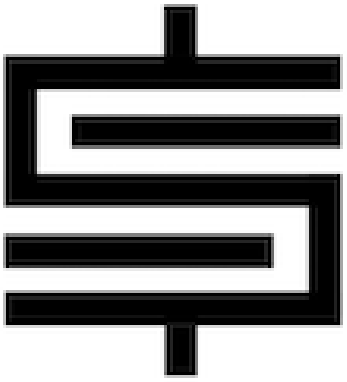


ISTD - 50.037 : Blockchain Technology

Project: SUTDCOIN

Team Name: CorkBlock

Team Members: Aiden Chia, Eugene Chia, Ong Chin Tak

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Table of Contents

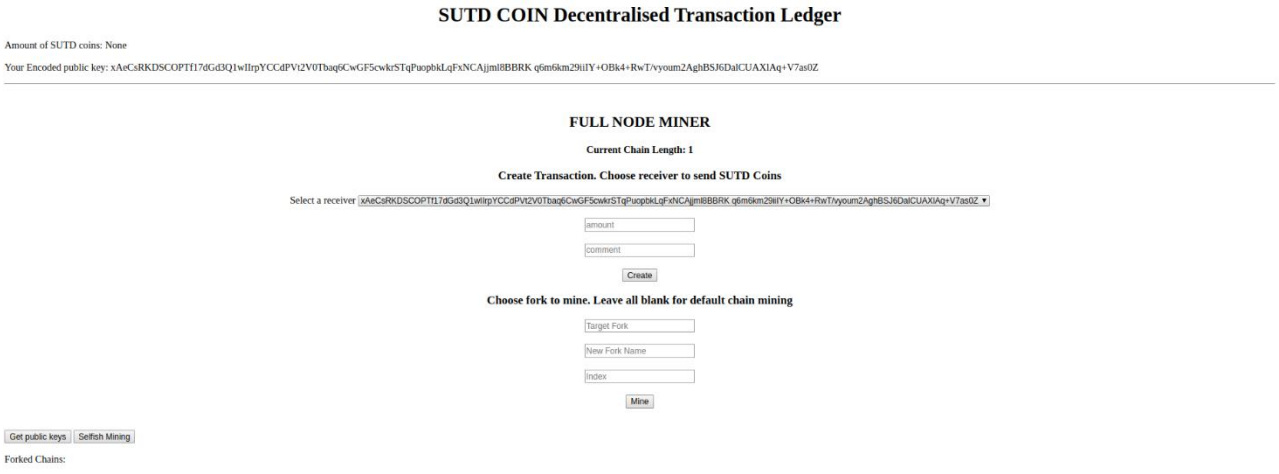
Introduction

SUTDCoin is a blockchain network adapted from the Bitcoin model built using Python and Flask. There are two types of nodes – full nodes (Miner) and SPV nodes (SPV Client)

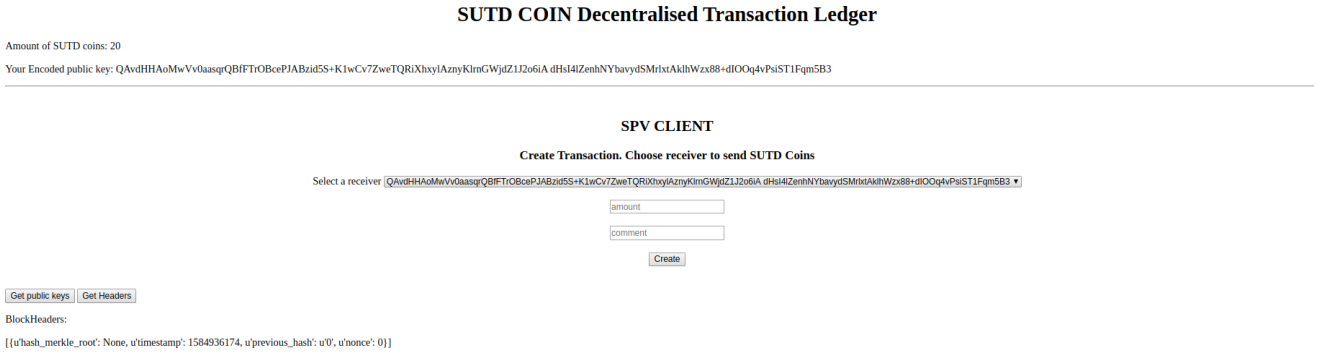
Typical Transaction Flow

1. Transaction is created by SPV Client or Miner
2. Miner mines transaction with proof of work and validates it before adding to main chain
3. Miner checks for consensus and reaches out to other nodes to see if it has the longest chain. If it has the longest chain, it updates other miners with the blockchain details (main chain, coins, locked_coins). Otherwise, it does not.
4. Whenever blocks are mined, coins are not directly added to account balance, but into locked_coins that are only released after a specified LOCKTIME. E.g. if LOCKTIME = 6, coins will only be released 6 blocks after the block is mined.
5. All miners have the updated blockchain and account balance now.

Full Node Miner User Interface



SPV Node Miner User Interface



System Requirements

- OS of any linux flavor
- Python installed

Steps to set up simple SPV-Node server

Assumptions

Miner node running on port 8000 is the first miner node to exist

Steps

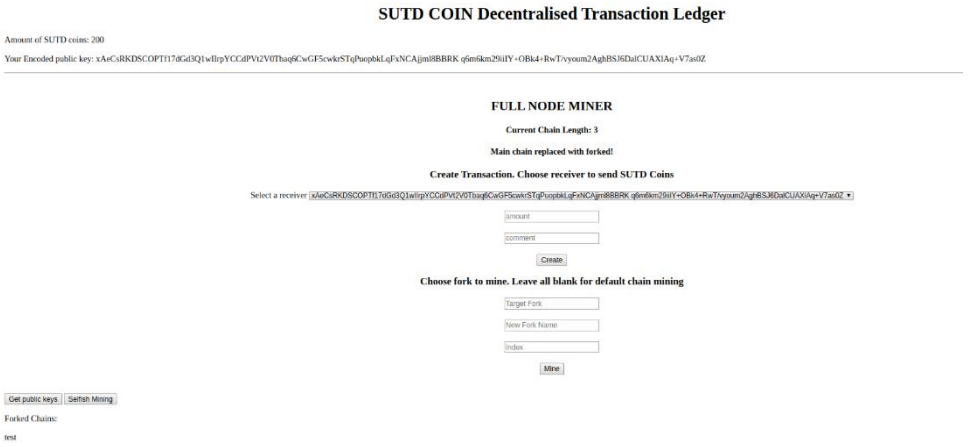
Steps:	Code:	Comments
1. Install relevant packages:	pip install –r requirements.txt	
2. Run miner node 1 on port 8000:	env FLASK_APP=server.py flask run --port 8000	You may run multiple nodes on multiple ports and connect them using the commands in commands file
3. Run SPV node on port 8002	env FLASK_APP=spv_server.py flask run --port 8002	(SPV node's default connected full node is miner node running on port 8000
4. Register SPV node with miner node 1	curl -X POST \http://127.0.0.1:8002/spv_register_with \-H 'Content-Type: application/json' \ -d '{"node_address": http://127.0.0.1:8000}'	*For demonstration purposes, it is recommended to set LOCKTIME and TARGET to low values.
(Optional): To set up additional miner node: 5. Run new miner node on port 8001 or any port of your choice	env FLASK_APP=server.py flask run --port 8001	
6. Register new miner node with miner node 1	curl -X POST \http://127.0.0.1:8001/register_with \-H 'Content-Type: application/json' \-d '{"node_address": http://127.0.0.1:8000}'	

Replicate a simple mining process

1. Go to main page on http://127.0.0.1:8000
2. Create a transaction by filling in 0 for both amount and comment as miner will not have any coins in the beginning.
3. Return to main page and leave all inputs empty. Press the mine button to mine.
4. Refresh the main page. You should have 100 coins now as part of the mining reward.

Replicate fork resolution

1. Create a transaction and mine it as in the simple mining process above
2. Main chain will have 2 blocks now – genesis block and newly added block
3. Create another transaction with 0 values for amount and comment
4. Return to main page and fill in “main” for “Target Fork”, “new_fork” for “New Fork Name” and “index” as blank
5. Press the mine button to mine.
6. Refresh the main page. The main chain is now replaced with the “new_fork” as new_fork is of a greater length than the original main chain!



Replicate payment between miners to spv clients:

1. Create a transaction and mine it as in the simple mining process above
2. Miner should have 100 coins now.
3. On miner home page, set receiver to “spv client public key address”. Change amount to “50” and set comment to any value you like. Add transaction.
4. Go back to main page. Leave all inputs empty and mine the block.
5. Refresh both main pages. Miner should have 150 coins now (+100 coins from mining and –50 from transfer). SPV Node should have 50 coins now from the transaction.
6. Now, we attempt to transfer money back to miner
7. On SPV Node homepage, press “get public keys” button to get all public keys from miner node.

8. On SPV Node homepage, set receiver to “miner node public key address”. Change amount to “10” and set comment to any value you like. Add transaction. Transaction is now sent to miner node.
9. Go back to miner node main page. Leave all inputs empty and mine the block.
10. Refresh both main pages. Miner should have 260 coins now (+100 coins from mining and +10 from transfer). SPV Node should have 40 coins now (-10 from transfer).

Retrieve headers from miners

1. On SPV Node homepage, press “Get Headers” button
2. Refresh homepage and headers will be updated.

Replicate transaction resending protection

1. Change value of *“LOCKTIME”* in Blockchain.py line 52 to 6 (Remember to change it back to 0 for other demonstrations)
2. Carry out multiple rounds of simple mining process with 0 for both amount and comment.
3. Notice than amount of coins miner node has does not increase until after 6 blocks. This prevents double spending. If main chain is replaced by forked chain before 6 blocks is up, locked coins will be invalidated.

51% attack Demonstration

Concept

When a group of miners control more than 50% of the mining power, they can ignore a transaction and create a new fork. By working on the new fork, the new fork becomes longer than the original chain and the chain is replaced. The original transaction is lost and the coins are not deducted. The sender will be able to spend the coins again.

Steps to replicate

1. Change value of "*LOCKTIME"* in Blockchain.py line to 6 (Remember to change it back to 0 for other demonstrations)
2. Set up a single miner node server
3. Create a transaction and mine it as in the simple mining process above. This will be the transaction we want to invalidate.
4. Create a few transactions with 0 values for amount and comment
5. Return to main page and fill in "main" for "Target Fork", "attack" for "New Fork Name" and "0" for "index"
6. Press the mine button to mine.
7. Now, the new fork will be of the same length as the original chain but does not contain the original transaction.
8. Fill in "attack" for "Target Fork". Press the mine button to mine again. (Assuming we have more than 50% of the mining power, we can mine new blocks faster than the original chain)
9. Refresh the main page. Now, the main chain is now replaced with the "attack" fork as it is of a greater length than the original main chain! Original transaction is lost and coins can be double spent.

Selfish Mining attack Demonstration

Concept

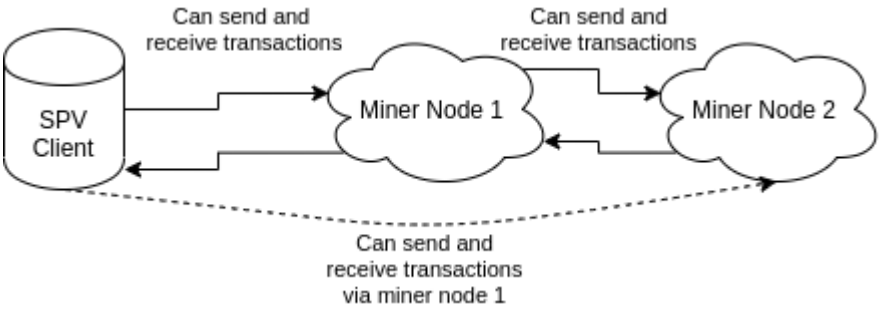
Selfish mining is a strategy for miners to collude to increase their revenue and deny others of their effort.

When the miner finds a new block, instead of announcing the new block, the miner hides the new block and continues to generate a few new blocks on a forked chain. When the private forked chain is longer than the main chain, the miner reveals his forked chain to other miners and the main chain is replaced by the private forked chain.

Steps to replicate

1. Set up a single miner node server
2. Create 4-5 transactions with 0 values amount and comment
3. Press selfish mining button
4. Refresh the main page. Notice that 3 blocks are added at the same time to the blockchain (Check out the current chain length! Number of coins have incremented by 300 too!)

Conceptual Overview of blockchain implementation



Actors

Miners – Miners are full nodes which store a whole blockchain and their individual transactions in the form of a merkle tree. They can mine blocks, send and receive transactions and also store additional information like the account balance.

SPV Client – SPV Clients are light nodes which do not store full blocks but only store the headers. They can also send and receive transactions but cannot mine blocks. Given transactions and their proofs, SPV Clients can verify transactions using the verify_proof method.

Differences between our SUTDcoin and Bitcoin

1. Instead of relying on the UTXO method of spending and receiving coins, we have implemented a method similar to that used in Ethereum. Each miner's public key is paired with the amount of coins that they current possess in a dictionary that is an attribute of the blockchain, much like a smart contract.
2. Methods are required to be called regularly to ensure that the blockchain's main fork is the longest one.
3. Instead of deciding on keeping the proof of work for some of the blockchains and releasing it all at once at a later date in order to invalidate the main chain, miners in our SUTDcoin are able to decide which fork they want to add their blocks by adding in the name of the fork into the block addition method.
4. SPV Nodes can only interact with a single selected miner node. However, they can send and receive transactions from other miner nodes via the directly linked miner node.
5. Fork Resolution via the resolve() method on Blockchain class is run every single time after a block addition to main chain or any fork to see which is the longest chain to use as the main chain. Consensus method is also run every time to determine if other miner nodes have longer main blockchain lengths.
6. Some commmunication functionality has been simplified for demonstration purposes.

Code Documentation

Block

Parameters taken in

1. Index: it is the unique ID of the block
2. Transactions: Merkle Tree
3. Time Stamp: Time Generation of the block
4. Previous Hash: Hash of the previous block in the chain

Methods

1. Compute_hash: Returns the hash of the block instance by first converting it into JSON string.

Blockchain

Attributes

1. TARGET: difficulty of PoW algorithm (Increase value to increase difficulty)
2. REWARD: reward for mining blocks
3. LOCKTIME: number of blocks to be mined before reward is released
4. self.unconfirmed_transactions = []: data yet to get into blockchain
5. self.locked_coins = []: blocks are mined but yet to release coins until LOCKTIME over. Locked_coins will be lost if current chain is replaced.
6. self.chain = []: List of all of the blocks in the block chain
7. self.forked_chains = {}: dictionary of the various forks of the chain, includes both main chain too.
8. self.create_genesis_block(): Method to create the genesis block
9. self.coins = {}: dictonary of the miner's public keys and the amount of coins they have

Methods

1. add_new_transaction(self, merkle_tree):
    a. Validates merkle tree before adding it to the unconfirmed_transactions list
2. validate_transaction(self, merkle_tree):
    a. first checks if the transaction's public key can verify the transaction
    b. Then checks if the transaction amount is less than the sender's coin balance.
3. create_genesis_block(self):
    a. generate genesis block and appends it to the chain. The block has index 0, previous_hash as 0, and a valid hash.
4. last_block(self):
    a. Retrieve's the most recent block that was added to the chain.
5. proof_of_work(self, block):
    a. Function that tries different values of the nonce to get a hash that satisfies our difficulty criteria.
6. add_block(self, miner, block, proof, target_fork="main", new_fork=None, index=None):
    a. A function that adds the block to the chain after verification.

   b. Verification includes:
      i. Checking if the proof is valid.
      ii. The previous_hash referred in the block and the hash of a latest block in the chain match.
   c. Miners can choose which fork to add to.
   d. If new_fork name is specified, create new fork based on existing targeted fork chain and index and add to new fork chain
   e. If no new fork_name is specified, block adds to the targeted chain. Default chain is the main chain.
   f. Index only matters for new fork creation
   g. If last 3 arguments not supplied, add block to main chain
7. validate(self, block, block_hash):
   a. Check if block_hash is valid hash of block and satisfies the difficulty criteria.
8. add_transacted_coins(self, block, miner):
   a. Adds the locked coins of the previously approved transactions to the miner's wallet after the specified number of lock down blocks have been added to the block chain.
9. release_locked_coins(self):
   a. Checks if the number of lock-down blocks have been added to the blockchain.
10. resolve(self):
   a. Checks for the longest chain and sets it as the main chain for the blockchain.

Miner

Attributes

1. public_key: base64 encoded public key of miner

Methods

1. mine(self, blockchain, target_fork="main", new_fork=None, index=None)
   a. serves as an interface to add the pending transactions to the blockchain by adding them to the block and figuring out proof of work.
2. get_coins(self, blockchain);
   a. returns the current balance of the number of coins in the miner's wallet.