Programming 2 project
Bitcoin

Huang Haoyi, Jin Lingfeng, Palma Leonardo Nicola,
Yu Angela, Zheng Gioia

June 2023

1 Introduction

Our goal is to develop a simple peer-to-peer blockchain simulator in order to make us able to see the decentralized transaction system in action, also to verify the consistency and the security of a blockchain-based model.

The advantages of building such a system using the blockchain technology are the following:

Decentralization By using a peer-to-peer network, the system can eliminate the need for a central authority.

Transparency Blockchain provides transparency by mantaining a public ledger of all transactions. This transparency ensures that all participants have access to the same information, reducing possibility of fraud or manipulation.

Security The use of cryptographic techniques to secure transactions and maintain the integrity of the system.

Consensus and Trust Blockchain enables trust between participants without the need for intermediaries. The immutability of transactions and the consensus mechanisms ensures that the system operates on agreed-upon rules and eliminates the need for trust in a centralized entity.

2 Organization

In the simulation all agents play a role in the simulation as a node in a blockchain system.

The three types of agent are **User**, **Miner** and **Malicious**, difined as classes that extend the **class Thread.**

Every agent in the simulation has a wallet, which is an object of **class Wallet** and his **Public and Private Keypair**.

2.1 User

An user represents a participant in the blockchain network which performs various operations:

2.1.1 Perform a transaction

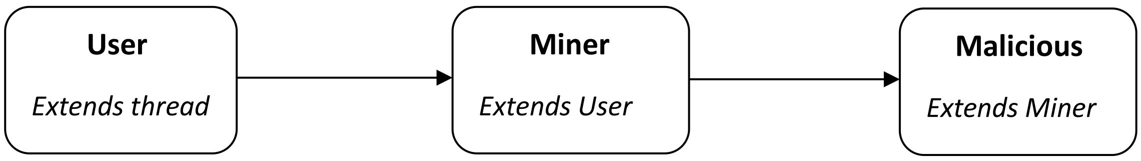
implemented in the method of name **do_transactions(),** in which an user checks its **UTXO** (discussed later) and perform a transaction of a random amount if that transaction is possible. These transactions will be signed using **Digital Signature** machanism and broadcasted to the network and put into the **mempool** (details in section 4) of the blockchain.

After performed a transaction, the user will **sleep()** for a random amount of time and repeat the task until the flag **isInterrupted** is set to True.

2.1.2 Verify the validity of a Block

An user verifies the validity of a new block added to a blockchain by checking its fields such as Pointer to the previous block and the **proof of work**.(discussed in further paragraph)

The User class is defined as a **subclass of class Thread** to allow concurrent execution, and it is also the **super-class** of class **Miner** and **Malicious**.

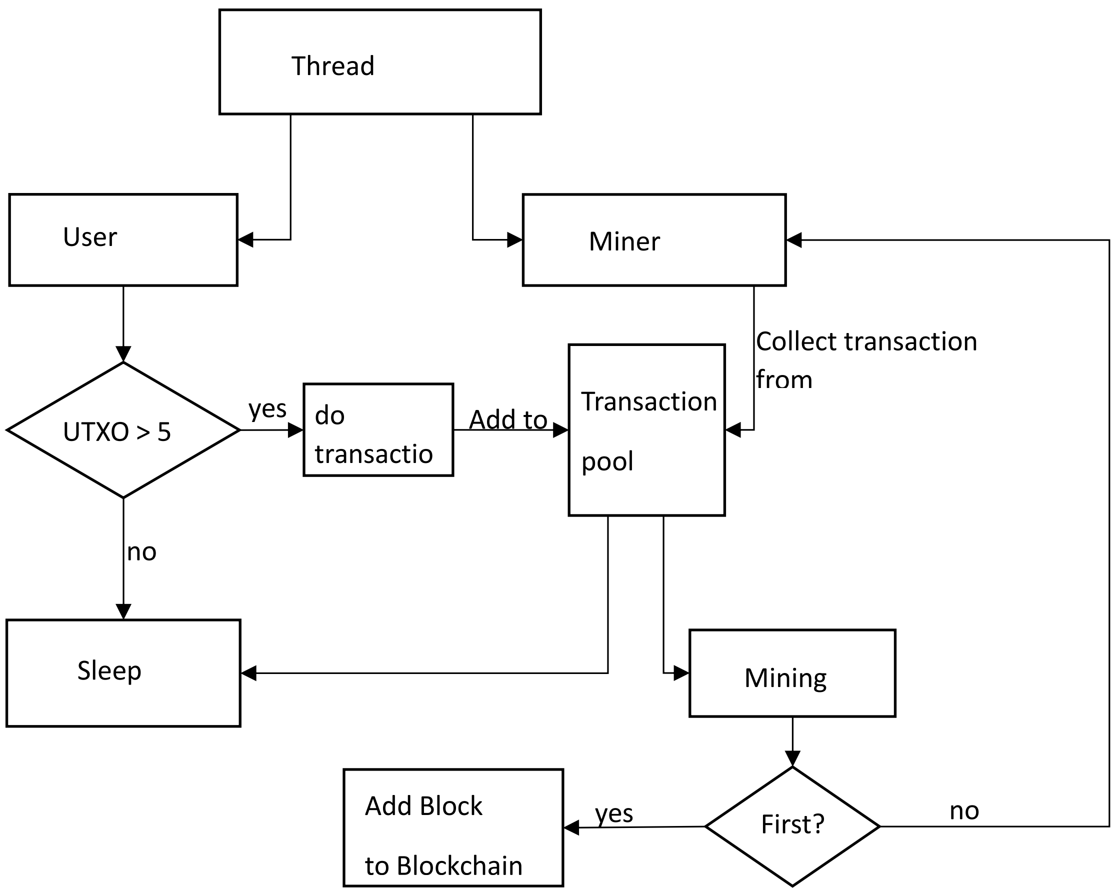


2.2 Miner

The miner agent plays a crucial role in a blockchain system, by mining new blocks. The Miner class extends the User class, inheriting its functionalities while adding specific mining capabilities.

2.2.1 Mining

Miners collect transactions from the **transaction pool**, create blocks by solving cryptographic puzzles called **proof of work**, and add them to the blockchain. They will receive a **reward** from the blockchain system and also **fees** from Users when performing transactions, in order to make their transactions included in the new block and then validated.



2.3 Malicious

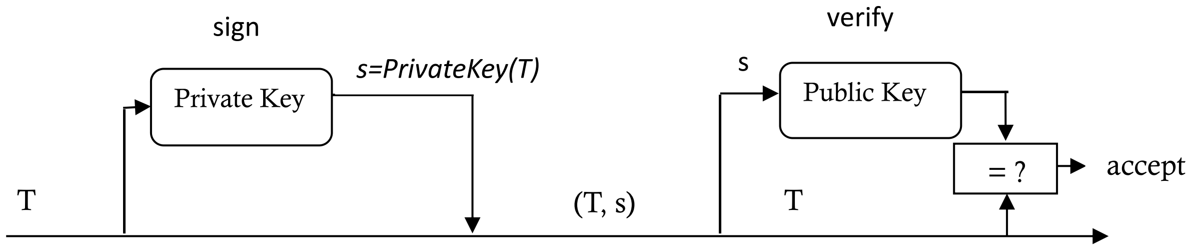
The Malicious agent represents a potential threat to the blockchain system. It is designed to simulate malicious behaviour, such as attempting to manipulate transactions**, disrupt consensus** or trying to **fork a different blockchain**. This agent helps to evaluate the system's **resilence** and **security** against various malicious scenarios.

3 Transactions

The **Transaction class** represents a transaction between two parties in a blockchain network. In the simulation, transactions are performed by user agents in a **randomized** controlled way. A transaction contains information about the sender , recipient and transaction details, such as **PublicKey** of the sender, transaction amount, **wallet address** of recipient and the **digital signature**.

3.1 Digital Signature

When a coin owner transfer the coin to another user, it needs to digitally sign the transaction obtained by **hashing** transaction informations using his **PrivateKey.** The receiver can verify the transaction by verifying the digital signature using sender's PublicKey.



This solves the problem of: *Does agent x really sent his coins to agent y?*

But the problem of **double-spending** remains unsolved, in this decentralized system there is no central authority that keeps track of all transactions, so the only way to avoid this problem is to make all transactions **public to every user** so that all participants can agree on a single history line of transactions, this line is constructed by a chain of units containing transactions, called **Blockchain**.

4 Block

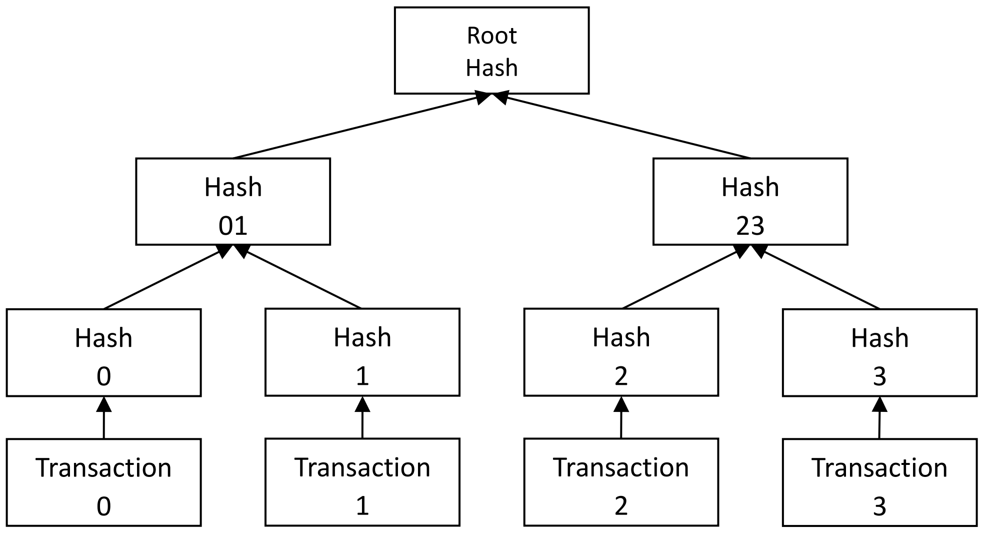
Every instance of the **class Block** represents a block in the blockchain network, it contains information about the transactions, **hash**, **hash of the previous block**, **nonce**, miner information and the **timestamp**.

In the simulator, each block contains 8 transactions, in which the first one is always the **coinbase**: where the Miner puts his information to receive rewards from the Blockchain.

4.1 Merkle Root

Each block contains a field named Merkle Root in which represents the root hash of a particular data structure called **Merkle Tree**, in which every leaf is the hash of a transaction in the block and every parent node is the **hash of the concatenation** of left and right child.

It enables the network user to check whether the transaction was included in the block or not, also the Merkle Root hash guarantees the **integrity** and security of each block in a peer-to-peer network.



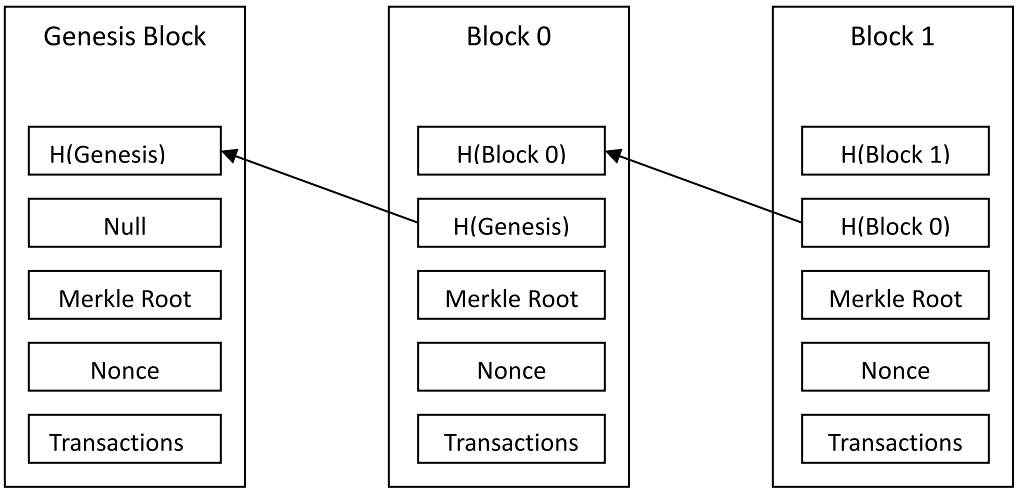
4.2 Nonce

The Nonce field is a integer number that represents the **proof-of-work**, calculated by solving **mathematical cryptographic tasks**, this is required to every block in order to be added to the blockchain. It consists a number in which makes the **hash of the block start with as many zeros** as many the difficulty of the blockchain is set to. The only way to find this number when a modern hash function like **SHA-256** is used is to **brute-force** random numbers until it gets the right one.

The first Miner that comes up with the valid nonce wins this cryptographic competition, his block will be added to the blockchain, and he will receive a reward from the Blockchain system to **incentivate** mining activities. Since calculating the proof of work is a task that asks a lot of **computational power**, the history of transactions that every agree on is **the chain containing the most number of blocks**, and this is extremely difficult for some malicious agents to change since they have to compete with all computation power of the network.

4.3 Genesis Block

The Genesis block represents the starting point of every blockchain and it **contains the first transactions** of the blockchain system, it does not point to any previous block since it is the first one and it is direclty **validated by the blockchain** system when the simulation starts.



5 Payment Validation

When a new block is mined and added to a blockchian, the transactions contained in that block should be validated if the chain that the block is adding to is the longest, this check is done by the blockchain system implemented as **class Blockchain**, and it contains all existing chains of blocks, each of them is stored as **ArrayList of Blocks.**

The validation of transactions is processed in following steps:

1 Check the validity of the block by checking if its hash starts with difficulty number of zeros.

2 Find the chain where the block is adding to, this is done by iterating through all existing forks and compare the hash of last block of a chain to the field hash of previous block.

3 If the chain is the one with the most number of blocks, transctions contained in this new block will be executed, the system will extract the wallet object of sender and receiver from a HashMap using key as walletAddress, and increment or decrement the balance based on the transaction amount (including fees).

4 The miner will be rewarded some coin, this reward number and the difficulty number are field of class Blockchain.

5 After the process, validated transactions will be removed from the mempool in order to make a transaction can be contained only in one block of the same blockchain.

6 Update the longestChain attribute, this field is added for efficiency reason, so that a call to get the longest blockchain does not require to iterate through all existing forks, instead we can just return this attribute and update it when a new block is added to the longest chain.

6 Simulations

6.1 implementation

Simulations are implemented in the main of **class TestBlockChain**, in the simulation we can create **genesis block** using different transactions to change the number of coins circulating in the network. We can create users by giving them an initial balance using the **overloaded constructor** and store all users(including miners and malicious) in a list structure that implements the **iterable interface** and start them all in once by using a **ForEach** loop.

The duration of the simulation can be controlled by making the main thread **sleep(time)**, when the main thread wakes up from the simulation it interrupts all user thread of the list.

6.2 Results

Now we can execute the main method of class TestBlockChain and change parameters specified above to analyze result in different scenarios.

We print the balance of each user and the size of the longest blockchain after each simulation, and we analysize the security of the system in presence of malicious agents using the following formula:

$$\boldsymbol{S}\% = \left(\mathbf{1} - \frac{\boldsymbol{Chain_{malicious}.\,size}}{\boldsymbol{Chain_{longest}.\,size}}\right) * \mathbf{100}$$

These are some results of the simulation:

6.2.1 Simple case where total # of users is a small number

The following result is obtained by setting the parameters:

- *Difficulty = 2*
- *Simulation Duration = 5000 (ms)*

# of users	*# of miners*	*# of malicious*	*Length of malicious chain*	*Length of longest Chain*	*S (%)*
5	*3*	*2*	*24*	*29*	*17.2*

Table 1: Simulation 6.2.1

As we can observe in the result, when we have:

# of malicious ≈ # of miners

The **security factor S** of the system is a very small number thus the system is not reliable in this case. Let’s simulate the case when we have more users to the network.

6.2.2 Changing the number of users

With same constant parameters as 6.2.1

# of users	*# of miners*	*# of malicious*	*Length of malicious chain*	*Length of longest Chain*	*S (%)*
50	*30*	*3*	*37*	*286*	*87.1*
100	*50*	*10*	*41*	*490*	*91.6*

Table 2: Simulation 6.2.2

We increased the number of users and miners, in other words we **increased the total computational power of the network** so that malicious agent have less probability to win since they have to compete with all miners in the network and the security factor of the system is significantly increased.

6.3.3 Increase the level of diffuclty

Constant parameters:

- *Difficulty = 4*
- *Simulation Duration = 10000 (ms)*

# of users	*# of miners*	*# of malicious*	*Length of malicious chain*	*Length of longest Chain*	*S (%)*
50	*30*	*3*	*15*	*65*	*76.9*
100	*50*	*10*	*11*	*63*	*82.5*

Table 3: Simulation 6.3.3

By increasing the **difficulty** we actually increased the **average mining time** of a block, the security factor S decreased and it approaches to a value ≈ 80%.

Thanks for reading

This project was made by:

Huang Haoyi

huang.2076810@studenti.uniroma1.it

Jin Lingfeng

jin.2043920@studenti.uniroma1.it

Palma Leonardo Nicola

palma.2043253@studenti.uniroma1.it

Yu Angela

yu.2074736@studenti.uniroma1.it

Zheng Gioia

zheng.2040439@studenti.uniroma1.it