December 03, 2023

PROJECT DOCUMENTATION

A complete documentation for the blockchain simulation that can simulate the behavior of a real-world blockchain and how transactions are processed in it.

Huy Le & Nikki Dulaney

CS 4850

Table of Contents

Overview

With today's advancement in technology, blockchain has become popular in the field of digital currency, offering a private and secure digital transaction between payers and payees without third-party interference. While blockchain technology is considered secure due to its decentralized nature, its networks are not immune to cyberattacks. This is a hybrid project in which the team aims to conduct both research and development in order to build and safeguard a blockchain.

This project encompasses two primary facets: research and development, each addressing the intricate aspects of blockchain vulnerabilities. From a research perspective, the project focused on methodologies for detecting vulnerabilities, offer an in-depth analysis of potential security weaknesses, and propose strategic preventive measures.

On the development front, we have created a Blockchain Simulation designed to mimic the behavior of a real-world blockchain with integrated security features. This simulation, developed utilizing Python and web development tools, serves as a practical representation of our research. The functionality and effectiveness of the Blockchain Simulation will be showcased in a detailed Presentation Video.

Complementing the simulation is a comprehensive Simulation Report. This document articulates the methodology employed, elaborates on the significance of hashing, digital signatures, Proof-of-Work concepts, transaction validation processes, and mining operations. Additionally, it provides an extensive explanation of the underlying code, along with a discussion on the observed results and subsequent recommendations.

This report presents an overall view of the project, starting with the planning and requirements. It delves into the project discussions section, offering insights into the team's methods, resources, and rationale behind the development of the Blockchain Simulation. Continuing, there is the simulation report, which provides a practical perspective on the Blockchain Simulation developed with security in mind. Next, the simulation design, security, and functionality are explained. Finally, the testing plan and report are provided.

Planning

This project aims to impart foundational knowledge about blockchain technology and its associated vulnerabilities. It will feature a practical blockchain simulation, complete with integrated security measures. The goal is to combine research with hands-on experience in blockchain security.

Team meetings will typically occur on Tuesdays or Thursdays. If meeting in-person, the team will meet from 3-5pm at the KSU Lawrence V. Johnson Library. If meeting virtually, the team will meet through Discord voice chat. Huy Le will schedule the meeting and Nikki Dulaney will take the meeting minutes. Work commitment will be at least 2 hours per week for each person and an average of 8 hours per week for the whole team.

Version Control

All versions of the project documentation will be stored and managed on the team's KSU OneDrive. OneDrive keeps all changes and updates are systematically tracked so everyone can work together seamlessly on documents. It is a great tool to make sure the team keeps on track of who changed what, since each team member's edits and contributions can be tracked.

All versions of the blockchain simulation and showcase website will be stored and controlled on the team's organizational GitHub repositories. GitHub is easy to learn and use, as the user interface is intuitive, and plenty of resources are available online. All team members will be able to see a consistent view of the project. Each team member has a full copy of the repository on their local machine that they can then edit. Each file has a history, allowing all team members to see all changes that occurred, when, and by who. It is easy to navigate through the many different versions of the files that were created. GitHub allows team members to open and close issues, which is useful for targeting problems. If necessary, relocating all of the files to a new repository will not be difficult.

Requirements

Assumptions

When creating a Blockchain Simulation and writing papers about security and blockchain, it is assumed that users of the product have a sufficient level of proficiency in the English language to read instructions, descriptions, and other text provided. It is assumed that users of the product have working vision to view the simulation and research cases. It is assumed that users will have a stable internet connection when accessing the showcase website. Finally, it is assumed that all team members have a device to conduct testing with.

Deliverables

In the scope of this project, the team is tasked with conducting in-depth research on blockchain technology and effectively presenting their findings. Additionally, the team is responsible for designing and developing a public showcase website. A critical component of the project involves the creation of a Blockchain Simulation, which must be tested for functionality. Upon completion of the simulation, the team is required to compose a comprehensive report detailing the development process, functionality, and key insights derived from the Blockchain Simulation.

The Blockchain Simulation

The front-end for the user client will have three pages:

1. The wallet page features a generate keys function which generates a public key and, private key creating a new wallet for the user.
2. The make transaction page includes a transaction form, a generate transaction function, and a cancel or confirm transaction function.
3. The view transaction page provides a text input asking for a blockchain node URL and a view transaction function to display the latest transaction on the blockchain.

The front-end for the miner client will have two pages:

1. Blockchain page contains sections for an unmined transaction table and a mined transaction table, and functionality for getting new transactions, mining transactions, and achieving consensus protocol.
2. Configure page allows miners to add more miner nodes to the blockchain network.

The back-end will require resources to create a block, generate transactions and signatures, verify signatures; and resources to perform mining, hashing, Proof-of-Work, and consensus protocol.

More details on the front-end and back-end are provided in the simulation report, which will contain information on the methodology. The report also contains explanations for the code, the results, and suggestions for future improvement.

Project Discussions

The team is developing a blockchain simulation that simulates the behavior of a real-world blockchain and how transactions are processed in it. The blockchain will be managed in a distributed P2P network, which allows for decentralization, anonymity, and consensus. Decentralization allows for enhanced security and transparency, and as such is one of the core principles of blockchain technology. P2P networks allow users to interact without needing to trust a central authority. The algorithm for consensus protocol that will be implemented is Proof-of-Work (PoW), which requires Miners to solve complex mathematical problems in order to validate a block in the blockchain. This makes it time consuming and computationally expensive to perform attacks such as 51% attacks and Sybil attacks.

The simulation back-end will be written in Python. The team chose Python for its readable syntax, which allows collaborators to understand and maintain the code, as well as for its extensive libraries and frameworks. The front-end will be written in HTML for creating webpages, Bootstrap for aesthetic of webpages, and JavaScript to handle the back-end.

Front-end for Users

Wallet Generator Page: To generate public key and private key. These keys will serve as a wallet for the user.

Make Transaction Page: To send tokens, the user client must know the following transaction details: sender's public key and private key, the recipient's public key, and the amount they wish to send. There will be a confirmation form that will pop up after clicking 'Generate Transaction'. Once confirmed, the transaction is added to the mining queue and appears in the unmined transactions table on the specified node URL.

The confirmation form will contain the user's input, the transaction signature, and a blockchain node URL that can be edited.

View Transactions Page: To enter the Blockchain Node URL to view transactions. The information shown will be the latest version of the blockchain which is a mined transaction table.

Front-end for Miners

Blockchain Page: Contains the not yet mined transaction table and the mined transaction table on the blockchain. Additionally, there will be a 'Refresh' button to get the newly created transaction, a 'Mine' button to mine the transaction block, and a 'Transaction' button to perform consensus protocol making sure all nodes have the latest version of the blockchain.

Configuration Page: To add more miners to the blockchain network.

Back-end

Create a block resource: Develop data structure and methods to create a block, store transactions in a block. A block must contain the block hash, previous block hash, time stamp, and data (transaction).

Create a resource to generate transactions and signatures: A set of methods to handle newly created transactions.

Create a resource for signature verification: A set of methods to verify digital signatures on incoming transactions to confirm the validity of transactions.

Implement mining: The process in which miners compete solving for PoW to validate a block and add the block to the blockchain.

Implement hashing and PoW: Implement a cryptographic hash function and develop the PoW algorithm that specifies the criteria for valid proof.

Configure network nodes: Define the nodes' structure and behavior; implement a set of methods so consensus protocol can be achieved.

Code Organization

The BlockchainSM folder contains 2 folders: Blockchain and Client. Blockchain has a blockchain.py file and a templates folder with 2 files: blockchain.html and configure.html. These files contain the source code relating to the miner. The Client folder has a client.py file and a templates folder with 3 files: wallet.html, make_transaction.html, and view_transaction.html. These files contain the source code relating to the user. The code is organized in a modular way, meaning concerns are separated into distinct files and folders and methods are sectioned by comments.

In the Blockchain folder, the blockchain.py file contains the main logic for the miner and the blockchain; the templates folder contains the HTML file for displaying blockchain related information and interaction and the HTML file for configuring and managing nodes. The blockchain.py file sets routes in a logical order (first show the unmined and mined tables, then a configuration page for adding nodes). The methods follow an order that is logical according to functionality.

In the Client folder, the client.py file contains the logic relating to the user, which includes wallet and transaction handling; the templates folder contains the HTML files for displaying key pairs of the wallet, creating new transactions, and viewing transactions. In the client.py file, the routes are set up in order of how a user would visit pages. (First generate keys to create a wallet, then make transaction, then view transaction). The methods occur in a logical order; keys generation must occur before transaction generation, as transactions rely on private and public keys. The HTML templates are organized in order of how the pages are displayed.

Simulation Report

Methodology

The team coded in python and utilized the Flask web framework for developing the back-end of the blockchain simulation. The Crypto library was used for cryptographic operations, which includes the key generation, signing, and hashing processes.

Implementation Details

Display data operation:

JavaScript was used for the front-end to handle button click event calling methods from the back-end.

AJAX function was implemented to perform requests fetching the data from the back-end.

The URL inside an AJAX function calls for the designated route from the back-end implemented using Flask.

DataTable was implemented which is a jQuery plugin to display the transaction tables.

Ellipsis was utilized to prevent text overflow on DataTable.

Web-app and routing operation:

The `flask ` library was imported from the Flask web framework to provide a lightweight and flexible environment for building python web applications.

The `jsonify` function generates JSON responses from Flask allowing it to send JSON data to clients.

The `request` function helped the handling of incoming HTTP requests. It provided access to the data sent by clients, allowing the back-end to process and respond accordingly.

The `render_template` function renders HTML templates, allowing the HTML pages to be dynamically generated based on the state of the simulation.

The '@app.route("/")' tells Flask what URL should trigger the function that follows.

Cryptographic operation:

The `pycrypto` library was imported to develop methods for cryptographic operations such as RSA-2048 key generation, transaction signature verification, and SHA-256 hashing processes.

The `RSA` class from `Crypto.PublicKey` was used for key pair generation.

The `PKCS1_v1_5` class from `Crypto.Signature` played a crucial role in implementing the PKCS #1 v1.5 signature scheme for signing transactions.

The `hash` class with the ‘SHA’ function performed the hashing operations.

Dependencies and Utilities:

In addition to Flask and Crypto, the team also utilized the following modules from the Python Standard Library:

binascii: Used for binary-to-text and text-to-binary conversions.

collections: Used for creating ordered dictionaries (`OrderedDict`), which were valuable for maintaining the order of elements in certain data structures.

uuid: Used specifically for utilizing the `uuid4` function. This function generates universally unique identifiers (UUIDs) and played a crucial role in uniquely identifying entities within the application.

json: Used for handling JSON data, important for communication between the back-end and front-end components.

hashlib: Used for generating hash functions, ensuring data integrity and security.

requests: Used for enabling HTTP requests to external entities, such as blockchain nodes.

urllib.parse: Used for parsing and constructing URLs; essential for handling and processing URLs.

The Code – Miner client (Blockchain client)

Back-end | blockchain.py

The ’Blockchain’ class manages nodes in the network and initializes the blockchain with an empty list of transactions and a genesis block. Each block contains a list of transactions, a nonce, a timestamp, and the hash of the previous block. Defines constants for the sender of mining rewards, the reward amount, and the mining difficulty.

Methods for the blockchain:

‘create_block’ method adds a new block to the chain with a specified block number, timestamp, transactions, nonce and previous hash.

‘hash’ method computes the hash of a block to ‘valid_chain’ validates the integrity of a chain.

‘verify_transaction_signature’ method verifies the digital signature of a transaction.

‘valid_proof’ method validates the PoW for a block.

‘proof_of_work’ method implements a PoW algorithm to find a nonce that satisfies the difficulty criteria.

‘register_node’ method adds a new node to the blockchain network.

‘valid_chain’ method retrieves the latest update from the other node with the longest blockchain.

‘resolve_conflicts’ method ensures that all nodes in the network have the longest valid chain and blockchain on all nodes are the same.

Flask App Setup initializes a Flask application and enables CORS to handle requests from different domains.

‘/’ route renders the blockchain page.

‘/configure’ route renders the node configuration page.

‘/transactions/new’ route calls for ‘new_ transaction’ method to handle the creation of new transactions.

‘/transactions/get’ route calls for ‘get_transaction’ method to retrieve the unmined transaction.

‘/mine’ route calls 'mine’ method to perform Proof-of-Work algorithm, and solve cryptographic puzzle by finding the nonce of the block.

‘/chain’ route calls ‘chain’ method to get the mined block and add the block to the blockchain.

‘/nodes/get’ route calls ‘retrieve_nodes’ method to retrieve the list of registered nodes in the network.

‘/nodes/register’ route calls ‘handle_nodes’ method to add new nodes to the network.

‘/nodes/resolve’ route calls ‘consensus_protocol’ method to achieve consensus protocol by invoking ‘resolve_conflicts’ method.

Front-end | blockchain.html

The JavaScript invokes the ‘get_transaction’ method from blockchain.py upon clicking ‘Refresh’ button, which populates the unmined transactions table. Button click event for ‘Mine’ invokes 'mine’ method from blockchain.py and populates the mined transactions table in the current node. ‘Transaction’ button invokes ‘consensus_protocol’ method from blockchain.py, which resolves conflict between nodes and gets the latest version of blockchain for all nodes.

Front-end | configure.html

The JavaScript handles the button click event for the ‘Add Node’ button. It invokes ‘retrieve_nodes’ method from the blockchain.py that adds a list of new blockchain nodes in the network.

The Code - User client

Back-end | client.py

Flask App Setup initializes a Flask application and enables CORS to handle requests from different domains.

'/' route renders the wallet page.

'/make/transaction' route renders the make transaction page.

'/view/transaction' route renders the view transaction page.

'/wallet/new' route calls for 'new_wallet' method to create a wallet by generating 2048-bit RSA key pair.

'/generate/transaction' route calls for 'generate_transaction' method to retrieve the transaction details user put.

Front-end | wallet.html

The JavaScript handles the button click event for the 'Generate Keys' button. It invokes the 'new_wallet' method from the client.py to generate key pairs.

Front-end | make_transaction.html

The JavaScript handles the button click event for the 'Generate Transaction' button. It Invokes the 'generate_transaction' method from the client.py to retrieve the transaction inputs. Button click event for 'Confirm Transaction' invokes 'new_transaction' method from blockchain.py to handle the creation of a new transaction.

Front-end | view_transaction.html

The JavaScript handles the button click event for the 'View Transaction' button. It Invokes the 'chain' method blockchain.py to get the mined block and add the block to the blockchain.

The flow of blockchain operation

‘hash’ takes a block as input, converts it to a string, and generates its SHA-256 hash. A hash uniquely identifies each block in the blockchain. ‘verify_transaction_signature’ verifies the digital signature of a transaction by using the sender’s public key to import the key, hash the transaction details, and verify the signature against the hash. If verification is successful, the transaction is valid. Transaction handling consists of the submission, validation, and processing of transactions within the blockchain. The ‘submit_transaction’ method takes the sender’s public key, recipient’s public key, transaction signature, and transaction amount as parameters. It adds a new transaction to the unmined transactions list and checks if the transaction is a reward for mining or a regular transaction. If it is a reward, it is added directly; else, it verifies the digital signature before adding it to the transaction list. The /transactions/get endpoint retrieves the unmined transactions; the ‘get_transaction’ method returns a JSON response containing the list of unmined transactions.

'valid_proof’ takes transactions, last_hash (hash of the last block), and nonce as parameters, concatenates these values, encodes them, and then computes the SHA-256 hash. The method checks if the characters at the beginning of the hexadecimal string (leading characters of the hash) match the predefined difficulty criteria. If the hash meets the difficulty criteria, it returns True, indicating a valid proof. ‘proof_of_work’ is part of the mining process. It iteratively increments the nonce until valid proof is found; to do this, it repeatedly calls ‘valid_proof’ until the hash meets the difficulty criteria. When a miner invokes the mining endpoint (/mine), the PoW algorithm executes.

The ‘proof_of_work’ method is called to find a nonce that meets the difficulty criteria. After a valid nonce is found, a new transaction is added to reward the miner using blockchain.submit_transaction. Then, the miner creates a new block with the calculated nonce and the hash of the last block. The details of the new block (message, block number, transactions, nonce, and previous hash) are included in the response. In the ‘valid_chain’ method, transactions are validated during the validation of the entire blockchain. Transactions are used to calculate the PoW for a block; if the proof is invalid, the entire chain is invalid.

In summary, miners solve cryptographic puzzles to create a new block. Miners play a crucial role in maintaining blockchain integrity by validating transactions and adding new blocks. The process is resource-intensive and time-consuming as it ensures that miners must expend computational effort to find a nonce that results in a hash meeting the difficulty criteria. This makes it impractical for attackers to compromise the network.

Results

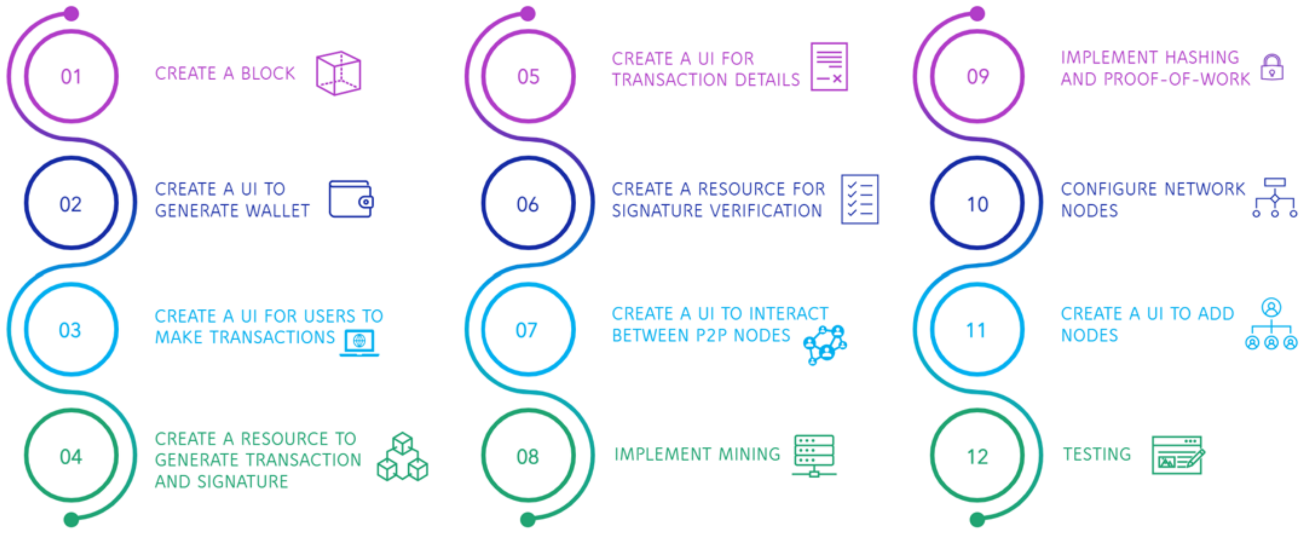
The blockchain simulation demonstrates a secure and robust system. The use of hashing and digital signatures ensures the integrity and authenticity of transactions within the blockchain. More security is provided by the PoW consensus mechanism, making the blockchain validated and more resilient. The user client functions are all successful. That includes wallet generation, transaction generation and confirmation, and transaction viewing. The miner client functions are all successful, including the refresh functionality for the unmined transactions table and mined transactions table, the mine functionality, and add node functionality.

Suggestions

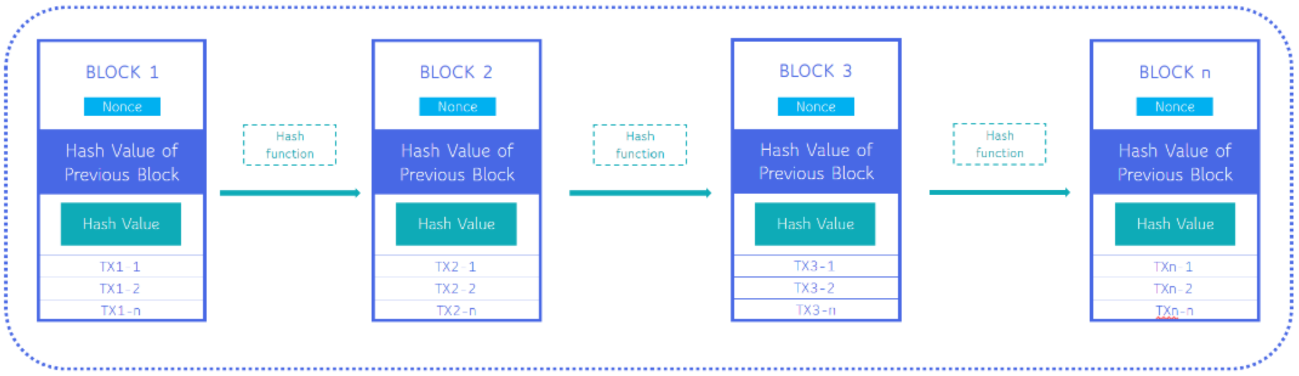
The blockchain could explore an alternative consensus mechanism such as Proof-of-Stake (PoS). With PoS, a participant must show that they own a certain amount of cryptocurrency tokens that are native to the blockchain in order to validate transactions on the crypto network. PoS is more energy efficient, and the validator node equipment is significantly less expensive overall compared to the equipment that is necessary to mine popular PoW cryptocurrencies at a profit.

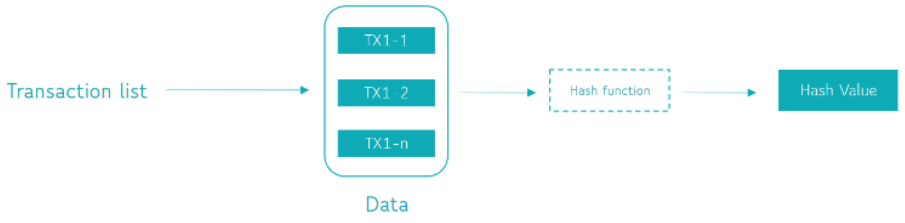
Simulation Design

Blockchain Simulation Development Process


01
CREATE A BLOCK
02
CREATE A UI TO GENERATE WALLET
03
CREATE A UI FOR USERS TO MAKE TRANSACTIONS
04
CREATE A RESOURCE TO GENERATE TRANSACTION AND SIGNATURE
05
CREATE A UI FOR TRANSACTION DETAILS
06
CREATE A RESOURCE FOR SIGNATURE VERIFICATION
07
CREATE A UI TO INTERACT BETWEEN P2P NODES
08
IMPLEMENT MINING
09
IMPLEMENT HASHING AND PROOF-OF-WORK
10
CONFIGURE NETWORK NODES
11
CREATE A UI TO ADD NODES
12
TESTING

Blockchain Structure





A blockchain is a chain of blocks that contains data. These blocks are linked together using hash cryptography. Each block contains the hash of the previous block.

The first block is called Genesis Block, where the previous hash will be 00000000 because there is no previous block to be assigned.

Inside a block there is:

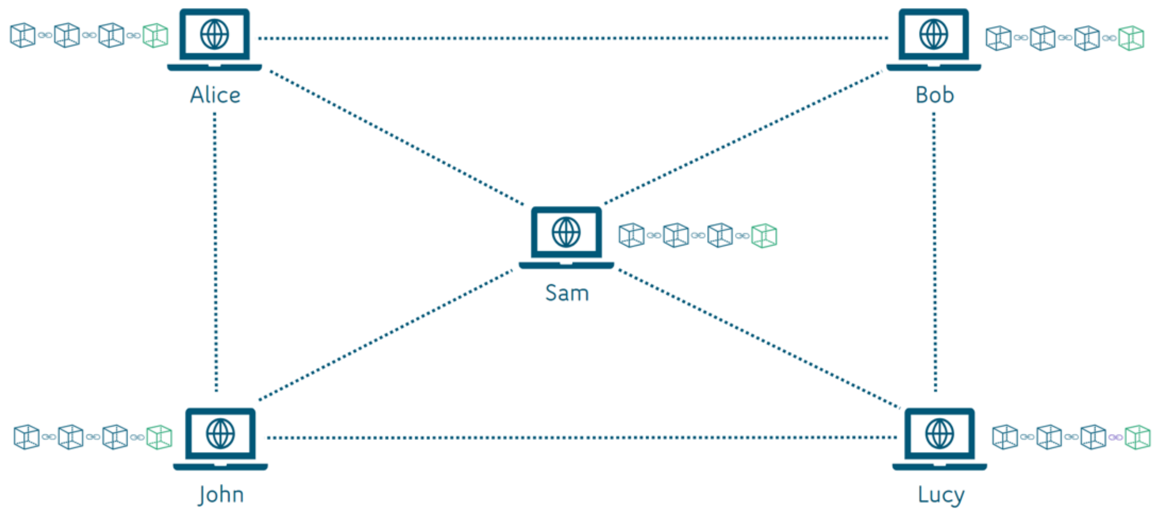
1. Data (transaction)

2. Previous block hash

3. Hash of the current block (also known as fingerprints of the block. A representation of that block)

The second block will contain the data, hash of the genesis block, and a new hash assigned for block 2, continuing in this pattern for the next blocks.

Distributed P2P Network and Consensus Protocol



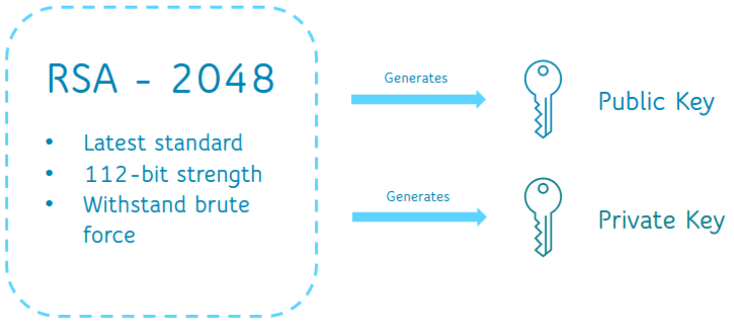
The blockchain is typically managed by a peer-to-peer (P2P) network. A P2P network is a distributed network containing a set of nodes that are interconnected together.

For example, in this distributed network, if Alice purchased a luxury watch from Bob for 5 ETH, a new block (green) containing that transaction information will be added to the block. The block will be created on Alice's node and added into Alice’s blockchain, that newest blockchain will be replicated among all other nodes within this P2P Network. This replication of blockchain for all nodes to have the same latest version is called Consensus Protocol.

Consensus Protocol is the core component of the blockchain. It determines how a decentralized computer network reaches agreement on which transactions are valid and which are not. There are various types of consensus mechanisms used in the blockchain network such as Proof-of-Work (Pow), Proof-of-Stake (PoS), and Proof-of-Burn (PoBr). For this project blockchain simulation, the team will be implementing Proof-of-Work.

Security in Blockchain Simulation

RSA-2048 Algorithm

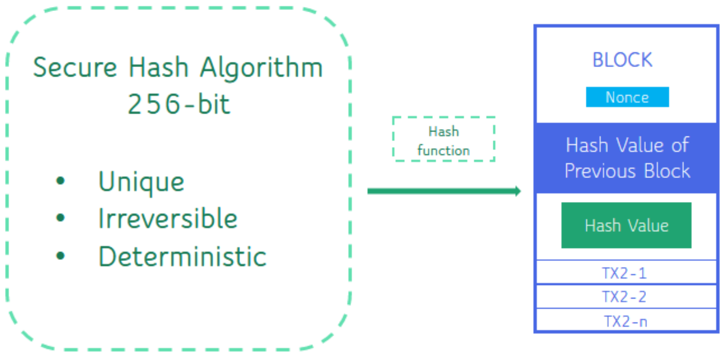


In blockchain technology, the RSA algorithm is the most widespread and used key generation algorithm. For the blockchain simulation, the team will implement the RSA-2048 algorithm to generate these public and private keys.

RSA-2048 uses a very large key size (2048 bits), which makes it extremely difficult to predict or replicate the private key, even if the random number generator is somewhat weak. The sheer complexity and size of RSA-2048 keys provide a buffer against the risks posed by weak randomness.

RSA-2048 is the current industry standard, offering a high level of security deemed adequate for the foreseeable future. With a security strength of 112 bits, RSA-2048 provides robust protection against brute force attacks, rendering it computationally impractical to breach.

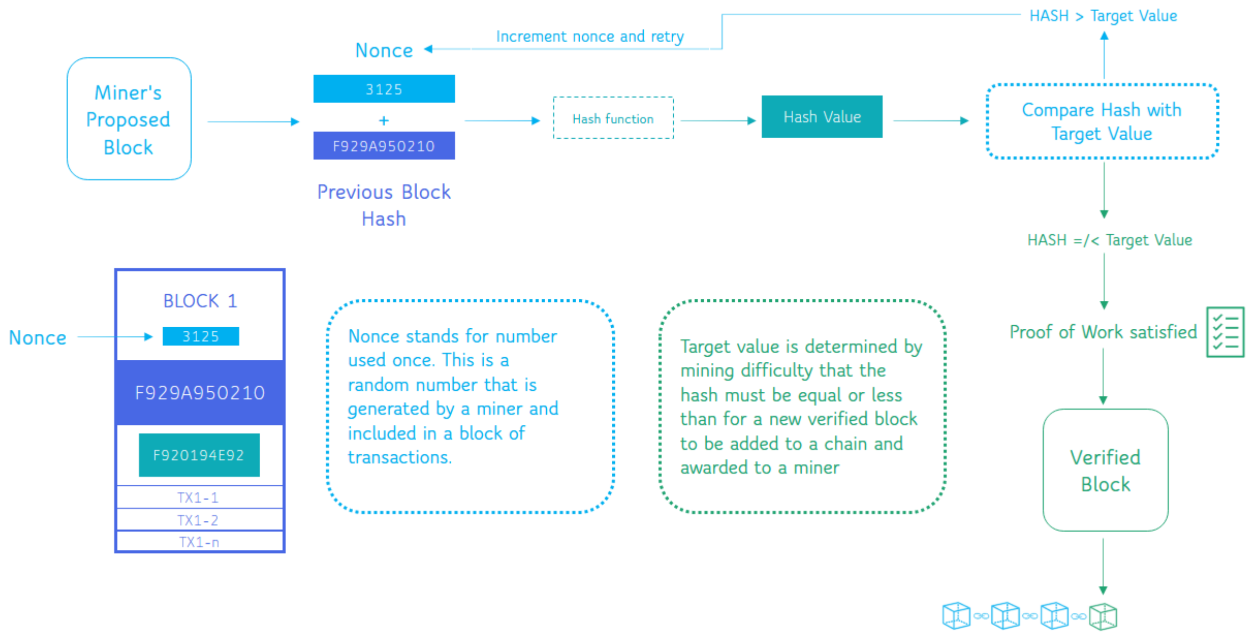
SHA-256 Hash Function



The simulation will be implemented with the SHA-256 hash function, a core component in generating block hashes and transaction signatures. SHA-256 can produce unique, irreversible, and deterministic hashes. Notably, Bitcoin has utilized SHA-256 since its inception.

The use of SHA-256 ensures that even small changes in input (such as transaction data or nonce value) lead to completely unpredictable and vastly different hash outputs. This unpredictability mitigates the risks associated with weak randomness in the creation of block hashes and transaction identifiers.

Mining and Proof-of-Work


Miner's Proposed Block
Nonce
3125
+
F929A950210
Previous Block Hash
Increment nonce and retry
Hash function
Hash Value
Compare Hash with Target Value
HASH > Target Value
HASH =/< Target Value
Proof of Work satisfied
Verified Block
Nonce
BLOCK 1
3125
F929A950210
F920194E92
TX1-1
TX1-2
TX1-n
Nonce stands for number used once. This is a random number that is generated by a miner and included in a block of transactions.
Target value is determined by mining difficulty that the hash must be equal or less than for a new verified block to be added to a chain and awarded to a miner

The team’s simulation will implement the Proof-of-Work (PoW) consensus mechanism. PoW, supported by the SHA-256 hash function, is integral to the blockchain network. Notably, Bitcoin has consistently used PoW, and it was previously implemented by Ethereum and other well-known blockchain platforms.

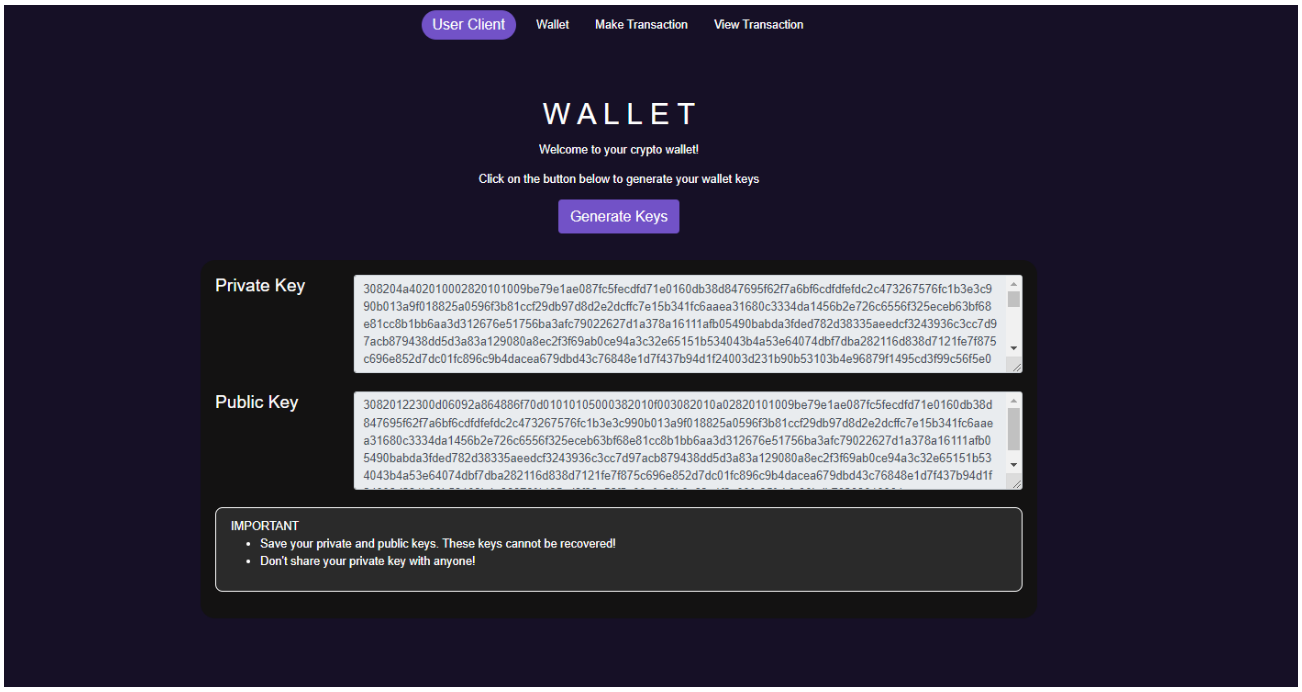
To successfully execute a 51% attack in a PoW system, attackers would need to gain control of more than 50% of a blockchain network's mining power. Miners performing PoW process involves solving complex mathematical problems to validate and add new blocks to the blockchain, ensuring that adding fraudulent blocks is computationally demanding and unfeasible for most attackers.

Blockchain Simulation: How It Works

User Client

1. Initiating a Transaction

To make a transaction, users must first create a wallet. This is accomplished by generating a pair of keys through the 'Generate Keys' function. The 'Generate Keys' button will call for RSA-2048 algorithm method which generates unique 2048-bit public and private keys. The private key should be kept private by the user because it is used to prove the ownership of the wallet. The public key, derived from the private key, is publicly known and acts as the address to perform cryptocurrency transactions.



Welcome to your crypto wallet!

Click on the button below to generate your wallet keys

Private Key

```
308204a402010002820101009be79e1ae087fc5fecdfd71e0160db38d847695f62f7a6bf6cdfdfefdc2c473267576fc1b3e3c9
90b013a9f018825a0596f3b81ccf29db97d8d2e2dcffc7e15b341fc6aaea31680c3334da1456b2e726c6556f325eceb63bf68
e81cc8b1bb6aa3d312676e51756ba3afc79022627d1a378a16111afb05490babda3fded782d38335aeedcf3243936c3cc7d9
7acb879438dd5d3a83a129080a8ec2f3f69ab0ce94a3c32e65151b534043b4a53e64074dbf7dba282116d838d7121fe7f875
c696e852d7dc01fc896c9b4dacea679dbd43c76848e1d7f437b94d1f24003d231b90b53103b4e96879f1495cd3f99c56f5e0
```

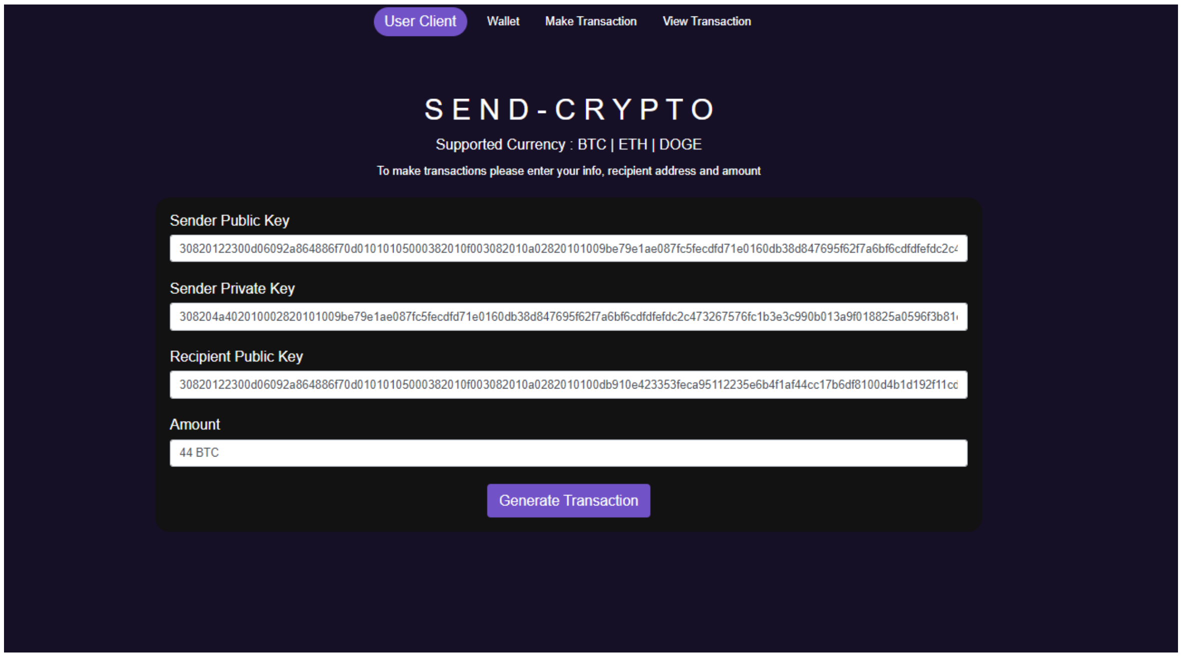
Public Key

```
30820122300d06092a864886f70d01010105000382010f003082010a02820101009be79e1ae087fc5fecdfd71e0160db38d
847695f62f7a6bf6cdfdfefdc2c473267576fc1b3e3c990b013a9f018825a0596f3b81ccf29db97d8d2e2dcffc7e15b341fc6aae
a31680c3334da1456b2e726c6556f325eceb63bf68e81cc8b1bb6aa3d312676e51756ba3afc79022627d1a378a16111afb0
5490babda3fded782d38335aeedcf3243936c3cc7d97acb879438dd5d3a83a129080a8ec2f3f69ab0ce94a3c32e65151b53
4043b4a53e64074dbf7dba282116d838d7121fe7f875c696e852d7dc01fc896c9b4dacea679dbd43c76848e1d7f437b94d1f
```

IMPORTANT
- Save your private and public keys. These keys cannot be recovered!
- Don't share your private key with anyone!

2. Creating a Transaction

For example, to transfer an amount of Ethereum (ETH), Bitcoins (BTC) or Dogecoin (DOGE), both sender and receiver must have their respective wallets. With both parties having their wallets, the sender can proceed to the 'Make Transaction' page. Here, the sender inputs the amount to be transferred, their public and private keys, and the recipient's public key. Selecting the 'Generate Transaction' button initiates the process, preparing the transaction for the next block.



SEND-CRYPTO

Supported Currency : BTC | ETH | DOGE

To make transactions please enter your info, recipient address and amount

Sender Public Key

30820122300d06092a864886f70d01010105000382010f003082010a02820101009be79e1ae087fc5fecdfd71e0160db38d847695f62f7a6bf6cdfdfefdc2c

Sender Private Key

308204a402010002820101009be79e1ae087fc5fecdfd71e0160db38d847695f62f7a6bf6cdfdfefdc2c473267576fc1b3e3c990b013a9f018825a0596f3b81

Recipient Public Key

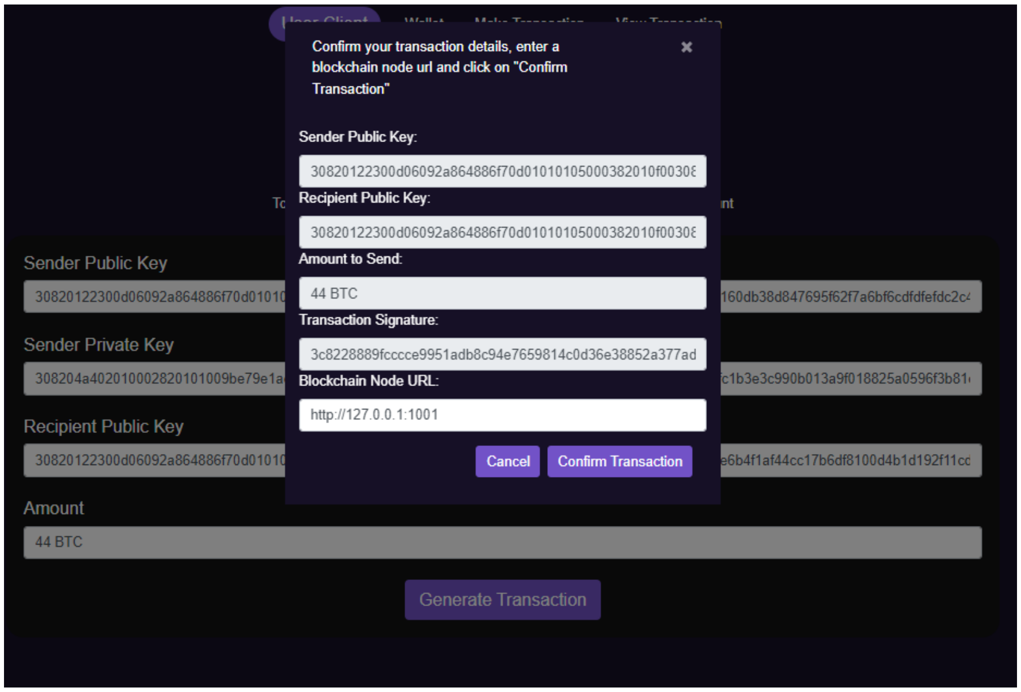
30820122300d06092a864886f70d01010105000382010f003082010a0282010100db910e423353feca95112235e6b4f1af44cc17b6df8100d4b1d192f11cc

Amount

44 BTC

3. Transaction Confirmation

Upon clicking the 'Generate Transaction' button, a confirmation form appears, displaying the entered details, a transaction signature, and the blockchain node URL where the transaction will be transmitted. Nodes in the blockchain network use the sender's public key to validate the transaction signature. A successful verification confirms the transaction's origin. The transaction, once confirmed, is queued in the next block for mining, visible on the designated blockchain node URL in the not yet mined transactions table.



Confirm your transaction details, enter a blockchain node url and click on "Confirm Transaction"

Sender Public Key:

30820122300d06092a864886f70d01010105000382010f00308

Recipient Public Key:

30820122300d06092a864886f70d01010105000382010f00308

Amount to Send:

44 BTC

Transaction Signature:

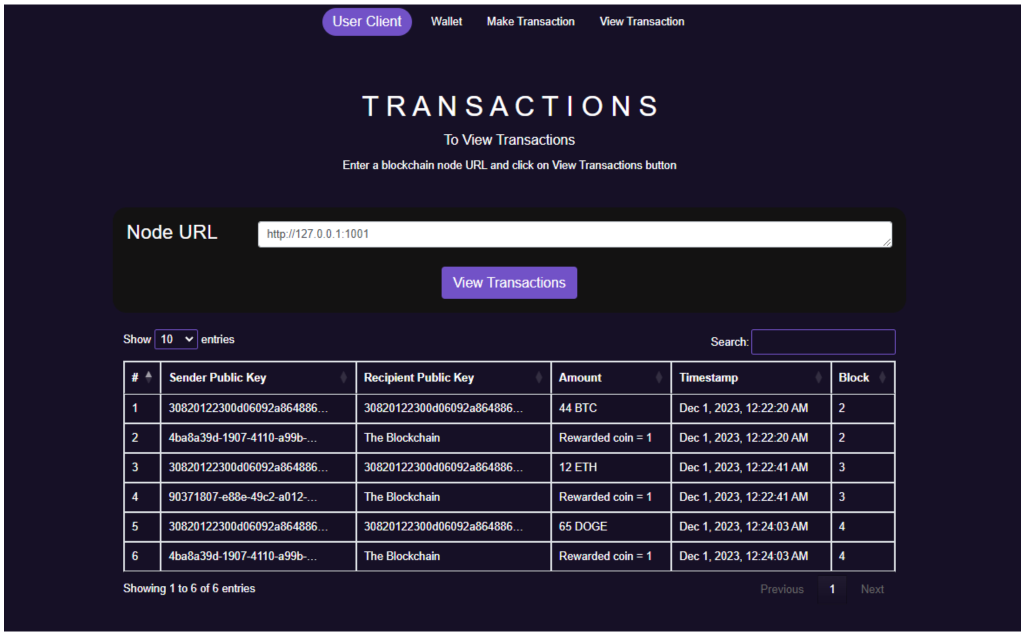
3c8228889fcccce9951adb8c94e7659814c0d36e38852a377ad

Blockchain Node URL:

http://127.0.0.1:1001

4. Viewing the Transaction

To view the transaction, the user can input the blockchain node URL and click 'View Transactions' button. This displays a list of mined transactions, including specific details of each.



TRANSACTIONS

To View Transactions

Enter a blockchain node URL and click on View Transactions button

Node URL: http://127.0.0.1:1001

Show 10 entries

Search:

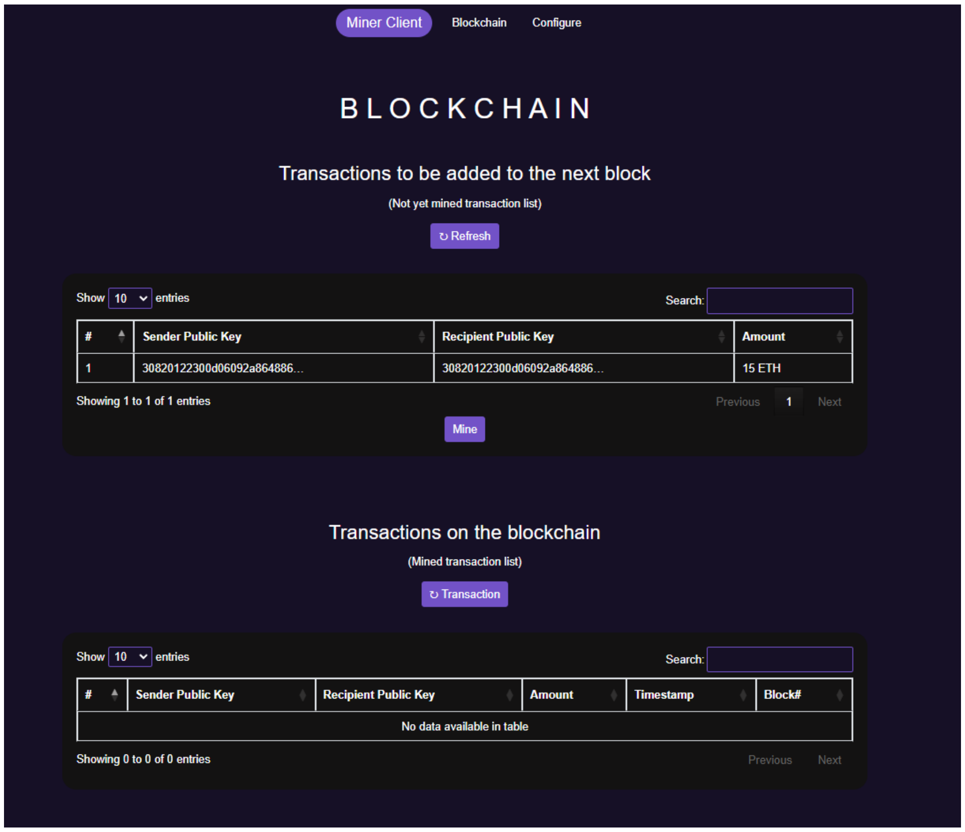
#	Sender Public Key	Recipient Public Key	Amount	Timestamp	Block
1	30820122300d06092a864886...	30820122300d06092a864886...	44 BTC	Dec 1, 2023, 12:22:20 AM	2
2	4ba8a39d-1907-4110-a99b-...	The Blockchain	Rewarded coin = 1	Dec 1, 2023, 12:22:20 AM	2
3	30820122300d06092a864886...	30820122300d06092a864886...	12 ETH	Dec 1, 2023, 12:22:41 AM	3
4	90371807-e88e-49c2-a012-...	The Blockchain	Rewarded coin = 1	Dec 1, 2023, 12:22:41 AM	3
5	30820122300d06092a864886...	30820122300d06092a864886...	65 DOGE	Dec 1, 2023, 12:24:03 AM	4
6	4ba8a39d-1907-4110-a99b-...	The Blockchain	Rewarded coin = 1	Dec 1, 2023, 12:24:03 AM	4

Showing 1 to 6 of 6 entries

Miner Client

1. Mining Process

Once a transaction is initiated, such as Alice sending Bob 10 ETH, the transaction appears in the not yet mined transactions table (Transaction to be added to the next block). However, it remains unconfirmed until mined. When a miner, for example, John ONE with node 192.0.0.1:1001, they can update the unmined transactions table to view pending transactions by clicking on ‘Refresh’ button. Upon selecting the 'Mine' button, the transaction undergoes Proof-of-Work, validated by the Consensus Mechanism, and subsequently appears in the mined transactions table (Transactions on the blockchain table).



BLOCKCHAIN

Transactions to be added to the next block

(Not yet mined transaction list)

Show 10 entries Search:

#	Sender Public Key	Recipient Public Key	Amount
1	30820122300d06092a864886...	30820122300d06092a864886...	15 ETH

Showing 1 to 1 of 1 entries

Transactions on the blockchain

(Mined transaction list)

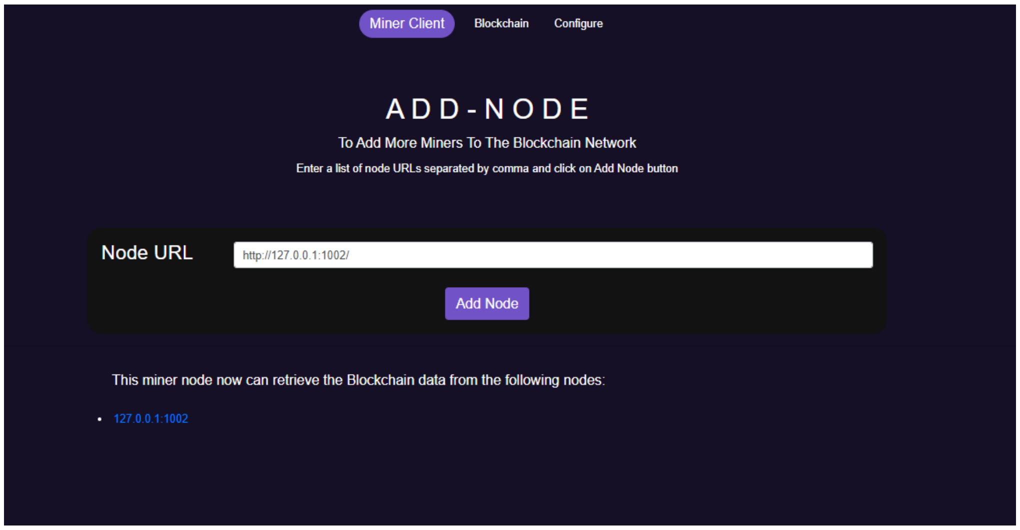
Show 10 entries Search:

#	Sender Public Key	Recipient Public Key	Amount	Timestamp	Block#
No data available in table					

Showing 0 to 0 of 0 entries

2. Network Security Through Increased Mining

Enhancing blockchain security involves expanding the network of miners. Additional miners, like John TWO and John THREE, with nodes 192.0.0.1:1002 and 192.0.0.1:1003, can join the blockchain network upon adding those nodes from an existing member, such as John ONE. However, John TWO and John THREE also need to be aware of John ONE node by adding 192.0.0.1:1001 on their end.



ADD-NODE

To Add More Miners To The Blockchain Network

Enter a list of node URLs separated by comma and click on Add Node button

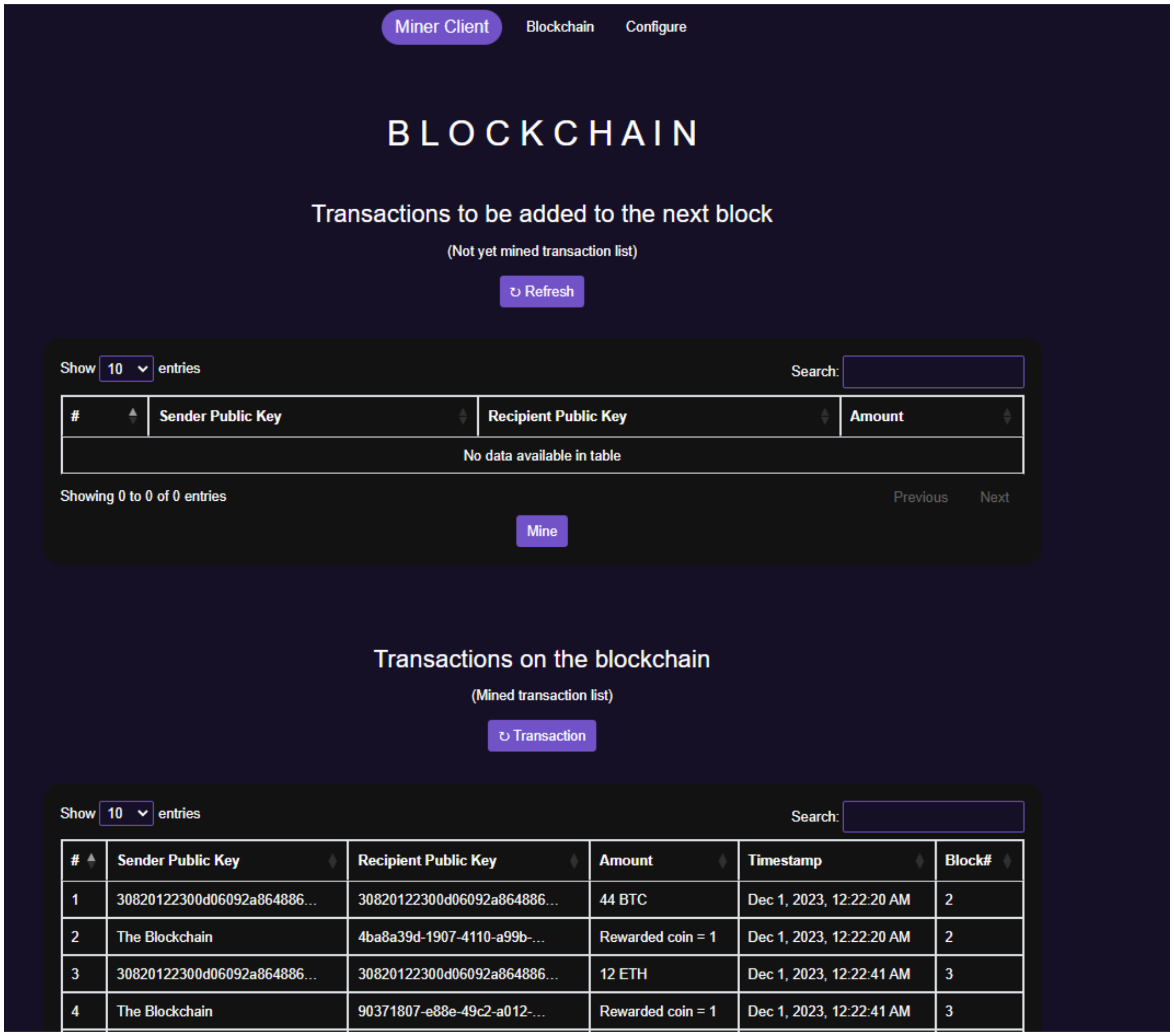
Node URL: http://127.0.0.1:1002/

This miner node now can retrieve the Blockchain data from the following nodes:

- 127.0.0.1:1002

3. Consensus Protocol

If the Consensus Protocol operates correctly by clicking 'Transaction' button on mined transaction list, all miners (like the three Johns) in the network will display identical lists of mined transactions upon refreshing their respective mined transactions tables.



BLOCKCHAIN

Transactions to be added to the next block

(Not yet mined transaction list)

Show 10 entries Search:

#	Sender Public Key	Recipient Public Key	Amount
No data available in table			

Showing 0 to 0 of 0 entries Previous Next

Transactions on the blockchain

(Mined transaction list)

Show 10 entries Search:

#	Sender Public Key	Recipient Public Key	Amount	Timestamp	Block#
1	30820122300d06092a864886...	30820122300d06092a864886...	44 BTC	Dec 1, 2023, 12:22:20 AM	2
2	The Blockchain	4ba8a39d-1907-4110-a99b-...	Rewarded coin = 1	Dec 1, 2023, 12:22:20 AM	2
3	30820122300d06092a864886...	30820122300d06092a864886...	12 ETH	Dec 1, 2023, 12:22:41 AM	3
4	The Blockchain	90371807-e88e-49c2-a012-...	Rewarded coin = 1	Dec 1, 2023, 12:22:41 AM	3

Testing

Test Scope

The testing scope will focus on the following functionalities: USER CLIENT: Generate Keys; USER CLIENT: Generate Transaction; USER CLIENT: Cancel; USER CLIENT: Confirm Transaction; USER CLIENT: View Transactions; MINER CLIENT: Transactions to be Added – Refresh; MINER CLIENT: Mine; MINER CLIENT: Transactions on the Blockchain – Transaction; MINER CLIENT: Add Node

Test Environment

Environment Variables

To run and test the simulation, it is required to install libraries, and edit User Variables Path by adding the paths to Python311 and Python311\Scripts.

Operating Systems

The simulation will be tested on Windows 11 and Windows 10.

Browsers

The simulation will be tested on Microsoft Edge and Google Chrome.

Configurations

At least 3 Python Node Configurations will be added: 2 User nodes and at least 1 Miner node.

The User client back-end script will be client.py. The parameters for User node 1 and User node 2 are -p 2401 and -p 2402, respectively.

The Miner client back-end script will be blockchain.py. The parameters for Miner node 1, Miner node 2, and Miner node 3 are -p 1001, -p 1002, and -p 1003, respectively.

Test Schedule

The team will conduct testing and troubleshooting throughout the development process. The testing table will be updated weekly, excluding during Fall Break (the week of November 20, 2023), starting from November 2, 2023.

Defects

On 11/02/2023, Huy Le reported 'Confirm Transaction' not working (SSL error).

On 11/22/2023, Huy Le reported 'Mine' not working (500 error).

On 11/27/2023, Huy Le reported consensus protocol and 'View Transactions' not working (404 error).

Solutions

On 11/16/2023, Huy Le fixed 'Confirm Transaction' not working (400 error).

Reason for defect: The blockchain back-end fails to retrieve the user input data from generate transaction.

Solutions:

- Fixed by removing the unexpected argument 'transaction' in Submit Transaction method
- Replaced the 'name' of Transaction Signature input text field to be the same as the id

On 11/27/2023, Huy Le fixed 'Mine' not working (500 error).

Reason for defect: The blockchain back-end fails to perform mining due to inconsistent keys in block creation and hashing.

Solutions:

- Fixed by changing the key in the block and hash dictionary inside the create_block method

On 11/28/2023, Huy Le fixed consensus protocol and 'View Transactions' not working (404 error).

Reason for defect: The blockchain back-end fails to call consensus method.

Solutions:

- Updated consensus method where it did not invoke resolve node conflict method
- Updated the code where there was a typo in HTML causing the mined transaction table to not display

TESTING

Testing Table

Date	Test Case	Pass	Fail
11/28/2023	USER CLIENT: Generate Wallet	X	
	USER CLIENT: Generate Transaction	X	
	USER CLIENT: Cancel	X	
	USER CLIENT: Confirm Transaction	X	
	USER CLIENT: View Transactions	X	
	MINER CLIENT: Refresh	X	
	MINER CLIENT: Mine	X	
	MINER CLIENT: Transaction	X	
	MINER CLIENT: Add Node	X	

Operating Systems

- Windows 10
- Windows 11

Browsers

- Google Chrome
- Microsoft Edge

Conclusion

The final report comprehensively outlines the planning phase and the specific requirements essential for the successful execution of the project. This is followed by an in-depth discussion covering various critical aspects of the project. The section titled "Project Discussions" delves into a detailed analysis of the team’s research and development stages. It methodically examines the procedural methodologies employed and the functionality of the blockchain technology utilized in the project. This analysis is further supplemented by an insightful Simulation Report.

The Simulation Report, a pivotal section of the project, meticulously details the methodology employed and the implementation processes of the blockchain simulation. It encompasses a wide range of topics such as hashing techniques, digital signatures, Proof-of-Work (PoW) mechanisms, transaction validation protocols, and mining processes. Additionally, this report presenting the results obtained and proposing suggestions for future enhancements and developments. Following this, the report elucidates the simulation design, focusing on its security features and overall functionality. Moreover, it provides comprehensive information regarding the testing methodologies applied. Concluding the report is a well-structured References page, which lists all the resources and materials utilized in the creation and development of the blockchain simulation.

References

This reference section documents the various resources that were instrumental in the development of our blockchain simulation. The resources are categorized based on their specific contribution to the project.

Blockchain Back-end Development:

1. Python Tutorial: Build a Blockchain - This article from Medium provides a step-by-step guide on creating a basic blockchain in Python.

(https://medium.com/coinmonks/python-tutorial-build-a-blockchain-713c706f6531).

2. Building a Blockchain using Python - JavaTpoint offers a comprehensive tutorial on blockchain development with Python.

(https://www.javatpoint.com/building-a-blockchain-using-python).

3. Create Simple Blockchain using Python - GeeksforGeeks presents a simplified approach to blockchain creation.

(https://www.geeksforgeeks.org/create-simple-blockchain-using-python/).

4. Learn Blockchains by Building One - This Medium article delves into the practical aspects of blockchain technology.

(https://medium.com/@vanflymen/learn-blockchains-by-building-one-117428612f46).

5. Building a Blockchain in Python - Analytics Vidhya discusses blockchain building blocks in Python.

(https://www.analyticsvidhya.com/blog/2022/06/building-a-blockchain-in-python/).

6. *How to Create a Simple Blockchain with Python* - Educative provides insights into creating a basic blockchain.

(https://www.educative.io/answers/how-to-create-a-simple-blockchain-with-python).

Hashing and RSA Key Generation:

1. PyCrypto Project - PyPI's PyCrypto library for cryptographic functionality in Python.

(https://pypi.org/project/pycrypto/).

2. Generating an RSA Key - A guide on RSA key generation using PyCryptodome.

(https://www.pycryptodome.org/src/examples#generate-an-rsa-key).

3. Generating Encrypted Key Pairs in Python - A tutorial on creating encrypted RSA keys.

(https://dev.to/aaronktberry/generating-encrypted-key-pairs-in-python-69b).

4. Python RSA Keys Guide - EasyDevGuide's tutorial on RSA keys in Python.

(https://www.easydevguide.com/posts/python_rsa_keys).

5. Python Code Snippet for RSA Key Generation - A useful Gist for generating RSA keys.

(https://gist.github.com/gabrielfalcao/de82a468e62e73805c59af620904c124).

6. Python-RSA Usage - Documentation on using the Python-RSA library.

(https://stuvel.eu/python-rsa-doc/usage.html).

Web Applications for Python:

1. Building a Web Application from Scratch with Flask and Python - A Medium article by Alexander Obregon on Flask-based web applications.

(https://medium.com/@AlexanderObregon/building-a-web-application-from-scratch-with-flask-and-python-f25f1f638aec).

2. Creating Your First Web Application Using Flask and Python 3 - DigitalOcean's tutorial on Flask web apps.

(https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3).

3. Flask Tutorial on Templates - Official Flask documentation on using templates in web applications.

(https://flask.palletsprojects.com/en/3.0.x/tutorial/templates/).