Blockchain API (Information Sharing Website as Demo)

Kiran More, Siddharth Kale, Sampatlal Jangid, Vedant Mahajan, Prashil Jambhulkar

Department of Computer Engineering

***Abstract* —** Our goal is to build an application that allows users to share information by posting. Since the content will be stored on the blockchain, it will be immutable and permanent. Users will interact with the application through a simple web interface.

Keywords *— Blockchain, Hashing ,Consensus, Proof Of Work.*

I. Introduction

As there is rapid growth in healthcare,new companies are coming forward to manufacture high demand products such as masks ,PPE kits,sanitizers,etc which are very important that their qualities are good and products are certified . But it's very difficult for hospitals to track the product whether it is of good quality , is it certified, etc. So by using blockchain technology we get a transparent mechanism to store and distribute data and is paving the way for solving serious data privacy, security, and integrity issues in healthcare.

This section introduces you to general blockchain concepts, such as a,transactions,blocks, Cryptographic security and consensus.

[1] Blockchain: A blockchain is an append-only database of transactions that is distributed to all participants in a blockchain network. There is no owner, administrator, or centralized data storage. Participants do not necessarily belong to the same enterprise or organization. Another term for blockchain is distributed ledger, because the database can be thought of as an electronic ledger of transactions (state changes) to the data.

This distributed ledger is:

- **Shared:** Each participant has a copy of the database that is demonstrably identical to all other copies in the blockchain network.
- **Auditable:** The blockchain provides an immutable (unalterable) history of all transactions that uses block hashes to detect and prevent attempts to alter the history.
- **Secure:** All changes are performed by transactions that are signed by known participants.

Cryptographic and signing mechanisms provide additional security for the transactions on the blockchain. These features work together with a consensus mechanism to provide “adversarial trust” among all participants in a blockchain network

Transaction: A transaction is a change to the shared state of the blockchain database. For example: 1)Transferring an asset, such as transferring product from manufacturer to distributor etc 2) Changing an item’s location when tracking provenance on a supply chain 3) Updating personal data, such as patient information in an electronic medical record Transactions are gathered into blocks, with one or more transactions per block.

Working of Blockchain: You can think of a blockchain as a series of state changes, or log of transactions, that affect the state of the data on a shared distributed ledger. Each block on the blockchain contains transaction data and a header with a timestamp, signer, and hash value. The hash value is a cryptographic signature or digital fingerprint that uniquely identifies the data in that block, as well as the block’s position in the blockchain. Each block’s hash includes the hash value of the previous block, which makes it very difficult for a malicious user to change a previous block. Importantly, the hash algorithm takes an input string of any length and produces fixed-length output. Common blockchain hash algorithms include RIPEMD and SHA-2, (such as SHA256 and SHA512). A blockchain starts with a genesis block. Depending on the blockchain platform, the first block could be empty (except for its hash value) or contain initial settings for the blockchain.

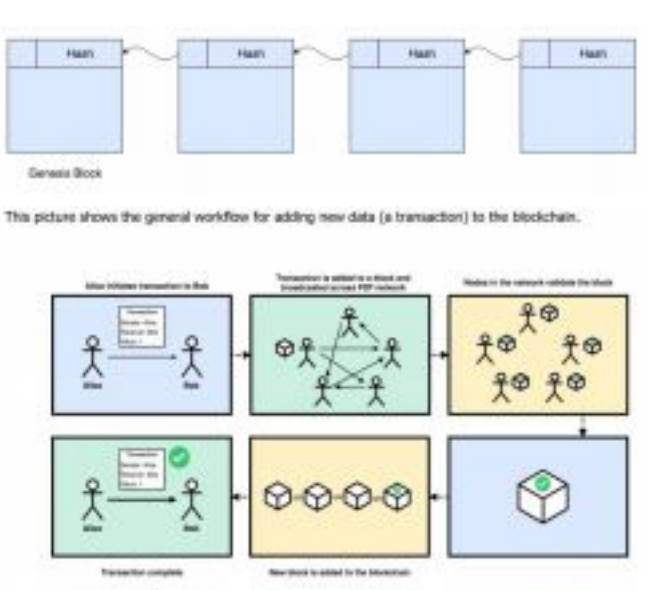

Fig : General workflow of adding new data(transaction) to the blockchain

Blockchain permissions: Blockchain networks have the following types of general permissions: A public blockchain has no restrictions on participation. It allows anyone to join and submit transactions. A signing

mechanism, such as public/private keys, identifies each submitter. Ethereum and Bitcoin are examples of public blockchains. A consortium blockchain is partly private or decentralized. It restricts which nodes can join the network, and can further control which nodes can participate in consensus. However, any participant can submit signed transactions. This network type supports policy-based transaction permissions. For example, Quorum is an Ethereum-powered consortium blockchain that is intended for use by banking and financial industries. A private blockchain is "permissioned" with access control features. It can have separate controls to restrict who can join, submit transactions, and participate in consensus. This network type supports policy-based transaction permissions. In addition, a private blockchain: 1) Specifies the type of transactions that "transactors" can sign 2) Restricts "address space access" to a limited set of "transactors" 3) Supports policy-based transaction permissioning

Consensus:A blockchain network uses consensus to determine whether a block is valid and should be added to the blockchain. The consensus algorithm determines how the participants decide and how many participants must agree. Types of consensus include: 1) **Practical Byzantine Fault Tolerance (PBFT)**, where participants elect a leader to validate transactions. As long as malicious participants are less than 50% of the network, they are overruled by the other participants. 2) **Proof of Work (PoW)**, a lottery-like system where the participant who finishes a computational task first (such as solving a cryptographic puzzle) is chosen to publish a block. Bitcoin uses Proof of Work consensus. 3) **Proof of Stake (PoS)**, another lottery-like system that requires a stake (such cryptocurrency or computational power). As each participant votes on whether a block is valid, the vote is weighted by the size of the participant's stake. 4) **Proof of Elapsed Time (PoET)** is a type of consensus that uses random wait times that are generated from a trusted source. The wait time determines who can propose and publish blocks. The participant with the shortest wait time wins the right to validate the block. PoET consensus improves energy use and resource consumption as compared to PoW or PoS consensus

When not to use a blockchain: Blockchain technology is designed for applications that need a decentralized, distributed database with no central control or owner. Participants can be "mutually distrusting," such as competitors in the same business space. The traditional approach is a centralized relational database (RDBMS) under the control of a single managing authority. A centralized RDBMS is appropriate for data with a single owner (such as a corporation or government entity) where all contributors are trusted. Performance is another issue. A blockchain platform is designed for security and decentralization, but not necessarily speed. For example, traditional stock market transactions must occur at a much faster rate than is possible on current blockchain platforms, where each participant must process every transaction.

II. Literature Review //changes(own lang)

Name of Authors: Satoshi Nakamoto. **Title/Year of Publication:** Bitcoin: A Peer-to-Peer Electronic Cash System. **Abstract:** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone. **Data structure and Algorithms:** Merkle tree as hash based data structure. Validation and Consensus Algorithm. **Limitations:** 1) Large computing power required for proof of work consensus Algorithm. 2) Smaller Proof of Work blockchains that remain extremely vulnerable to 51% attacks.

Name of Authors: Lin Chen ??, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi Department of Computer Science, University of Houston, TX 77004, USA **Title/Year of Publication:**On Security Analysis of Proof-of-Elapsed-Time (PoET) October 2017 DOI: 10.1007/978-3-319-69084-1_19 Conference: International Symposium on Stabilization, Safety, and Security of Distributed Systems. **Project:** Blockchain Technology**. Algorithms:** Proof Of Elapsed Time (POET). **Abstract:**As more applications are built on top of blockchain and public ledger, different approaches are developed to improve the performance of blockchain construction. Recently Intel proposed a new concept of proof of-elapsed-time (PoET), which leverages trusted computing to enforce random waiting times for block construction. However, a trusted computing component may not be perfect and 100% reliable. It is not clear, to what extent, blockchain systems

based on PoET can tolerate failures of trusted computing components. The current design of PoET lacks rigorous security analysis and a theoretical foundation for assessing its strength against such attacks. To fulfill this gap, we develop a theoretical framework for evaluating a PoET based blockchain system, and show that the current design is vulnerable in the sense that adversary can jeopardize the blockchain system by only compromising $\Theta(\log \log n / \log n)$ fraction of the participating nodes, which is very small when n is relatively large. Based on our theoretical analysis, we also propose methods to mitigate these vulnerabilities.

III. Methodology/Experimental

A. Method

1) Store transactions into blocks: We'll be storing data in our blockchain in a format that's widely used: JSON. Here's what a product stored in blockchain will look like.

```
{
"product ID": "uniquely identifies the product",
"type of product": "Masks,PPE kits,Sanitizers etc",
"location of product": "location"
"timestamp": "The time at which the content or product was created"
}
```

The transactions are stored packed into blocks. A block can contain one or many transactions. The blocks containing the transactions are generated frequently and added to the blockchain. Because there can be multiple blocks, each block should have a unique ID:

```
class Block:
    def __init__(self, index, transactions, timestamp):
        """
        Constructor for `Block` class.
        :param index: Unique ID of block.
        :param transactions: List of transaction.
        :param timestamp: Time of generation of block.
        """
        self.index = index
        self.transactions = transaction
        self.timestamp = timestamp
        self.previous hash = previous__hash
```

2) Adding Digital fingerprints to the blocks: We'd like to prevent any kind of tampering in the data stored inside the block, and detection is the first step to that. To detect if the data in the block has been tampered with, you can use cryptographic hash functions.

A hash function is a function that takes data of any size and produces data of a fixed size from it (a hash), which is generally used to identify the input.

The characteristics of an ideal hash function are:

1) It should be easy to compute.

2) It should be deterministic, meaning the same data will always result in the same hash.

3) It should be uniformly random, meaning even a single bit change in the data should change the hash significantly. The consequence of this is: It is virtually impossible to guess the input data given the hash. (The only way is to try all possible input combinations.)

If you know both the input and the hash, you can simply pass the input through the hash function to verify the provided hash.

3)Chain the blocks: We need a way to make sure that any change in the previous blocks invalidates the entire chain. One way to do this is to create dependency among consecutive blocks by chaining them with the hash of the block immediately previous to them. By chaining here, we mean to include the hash of the previous block in the current block in a new field called previous_hash.

Okay, if every block is linked to the previous block through the previous_hash field, what about the very first block? That block is called the genesis block and it can be generated either manually or through some unique logic. Let's add the previous_hash field to the Block class and implement the initial structure of our Blockchain class.

Now, if the content of any of the previous blocks changes: The hash of that previous block would change.This will lead to a mismatch with the previous_hash field in the next block. Since the input data to compute the hash of any block also consists of the previous_hash field, the hash of the next block will also change.

4) Implement a proof of work algorithm: There is one problem, though. If we change the previous block, the hashes of all the blocks that follow can be re-computed quite easily to create a different valid blockchain. To prevent this, we can exploit the asymmetry in efforts of hash functions to make the task of calculating the hash difficult and random. Here's how we do this: Instead of accepting any hash for the block, we add some constraint to it. Let's add a constraint that our hash should start with "n leading zeroes" where n can be any positive integer.

Unless we change the data of the block, the hash is not going to change, and of course we don't want to change existing data. So, we'll add some dummy data that we can change. Let's introduce a new field in our block called nonce. A nonce is a number that we can keep on changing until we get a hash that satisfies our constraint. The nonce satisfying the constraint serves as proof that some computation has been performed. The number of zeroes specified in the constraint determines the difficulty of our proof of work algorithm (the greater the number of zeroes, the harder it is to figure out the nonce).

Also, due to the asymmetry, proof of work is difficult to compute but very easy to verify once you figure out the nonce (you just have to run the hash function again):

5) Add blocks to the chain:

To add a block to the chain, we'll first have to verify that: The data has not been tampered with (the proof of work provided is correct). The order of transactions is preserved (the previous_hash field of the block to be added points to the hash of the latest block in our chain).

Mining: The transactions will be initially stored as a pool of unconfirmed transactions. The process of putting the unconfirmed transactions in a block and computing proof of work is known as the mining of blocks. Once the nonce satisfying our constraints is figured out, we can say that a block has been mined and it can be put into the blockchain.

6)Create interfaces:

Now it's time to create interfaces for our blockchain node to interact with the application we're going to build. We'll be using a Python microframework called Flask to create a REST API that interacts with and invokes various operations in our blockchain node.We need an endpoint for our application to submit a new transaction. This will be used by our application to add new data (posts) to the blockchain:

Here's an endpoint to return the node's copy of the chain. Our application will be using this endpoint to query all of the data to display:

Here's an endpoint to request the node to mine the unconfirmed transactions (if any). We'll be using it to initiate a command to mine from our application itself:

7) Establish consensus and decentralization: Up to this point, the blockchain that we've implemented is meant to run on a single computer. Even though we're linking blocks with hashes and applying the proof of work constraint, we still can't trust a single entity (in our case, a single machine). We need the data to be distributed, we need multiple nodes maintaining the blockchain. So, to transition from a single node to a peer-to-peer network, let's first create a mechanism to let a new node become aware of other peers in the network:

A new node participating in the network can invoke the register_with_existing_node method (via the /register_with endpoint) to register with existing nodes in the network. This will help with the following:

Asking the remote node to add a new peer to its list of known peers.

Initializing the blockchain of the new node with that of the remote node.

Resyncing the blockchain with the network if the node goes off-grid.

However, there's a problem with multiple nodes. Due to intentional manipulation or unintentional reasons (like network latency), the copy of chains of a few nodes can differ. In that case, the nodes need to agree upon some version of the chain to maintain the integrity of the entire system. In other words, we need to achieve consensus. A simple consensus algorithm could be to agree upon the longest valid chain when the chains of different participating nodes in the network appear to diverge. The rationale behind this approach is that the longest chain is a good estimate of the most amount of work done :

Next, we need to develop a way for any node to announce to the network that it has mined a block so that everyone can update their blockchain and move on to mine other transactions. Other nodes can simply verify the proof of work and add the mined block to their respective chains :

The announce_new_block method should be called after every block is mined by the node so that peers can add it to their chains.

IV. Results and Discussions

Blocks can't be modified once added; in other words, it is appended only.
There are specific rules for appending data to it.
Its architecture is distributed.
Enforcing these constraints yields the following benefits:
Immutability and durability of data
No single point of control or failure
A verifiable audit trail of the order in which data was added

V. Limitations

There are two primary disadvantages to Proof of Work systems. The first is that they waste energy, which is bad for the environment. As computers perform extra computational work, additional electricity is used. This can add up to an extremely large amount of excess electricity consumption.

VI. Future Scope

API can be used for:- Asset Management: Trade Processing and Settlement, Insurance: Claims processing. Payments: Cross-Border Payments , Your car/ smartphone,Blockchain music, Passports Birth, wedding, and death certificates, Personal Identification.

Top solve limitations of Proof of work Consensus algorithm Proof of Elapsed Time can be Implemented.

VII. Conclusion

Since the content will be stored on the blockchain, it will be immutable and permanent. Users will interact with the application through a simple web interface.

Acknowledgment

Our project required guidance and assistance from people and we are extremely privileged to have got this all along the completion of the project. All that we have done is somewhere due to such supervision and assistance and We would not forget to thank them. We respect and thank our director sir for providing us an opportunity to do the project work in Vishwakarma Institute of Technology and giving us all the support and guidance which guided us to the right path . We owe deep gratitude to our project guide Prof. Sangita Lade who took keen interest in our project work and guided us all along by providing all the necessary information for developing a good system.

References

[1] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System; BN Publishing: La Vergne, TN, USA, 2008.

Benlamri:On Security Analysis of Proof-of-Elapsed-Time (PoET) [2] Mackey, T.K.; Kuo, T.T.; Gummadi, B.; Clauson, K.A.; Church, G.; Grishin, D.; Obbad, K.; Barkovich, R.;Palombini, M. 'Fit-for-purpose?'—Challenges and opportunities for applications of blockchain technology in the future of healthcare. BMC Med. 2019, 17, 68.

[3] Siyal, A.; Junejo, A.; Zawish, M.; Ahmed, K.; Khalil, A.; Soursou, G. Applications of Blockchain Technology in Medicine and Healthcare: Challenges and Future Perspectives.

[4] Fedor Muratov, Andrei Lebedev, Nikolai Iushkevich, Bulat Nasrulin, Makoto Takemiya -- YAC: BFT Consensus Algorithm for Blockchain