Leveraging blockchain technique for predicting and transmitting data over pervasive environment

Aditya Tripathi, Anurag Choudhury, Karan Sharma

Dr. Rahul Mishra, Dr. Tapas Kumar Maiti

***Abstract*---** **This document describes the workings and implementation of blockchain across three nodes using smart contracts. This setup of the blockchain can be tweaked to include any number of devices (more than 1). Tools such as Python, Flask, and Postman have been used to implement the same.**

***Keywords*---** **Blockchain, Smart contracts, Decentralized network, Python, Flask, Postman**

I. INTRODUCTION

Transparency and trust are essential in today's digital world to carry out safe and effective transactions. On the other hand, trust is frequently severely hindered by old centralized systems, which can result in problems like fraud, data manipulation, and inefficiency. By establishing a decentralized, immutable record that facilitates safe and transparent transactions without the need for middlemen, blockchain technology represents an evolutionary change.

Using the Python programming language, we will create a "smart blockchain." In a "smart blockchain" network, there is a Block Producer Smart Contract (BPSC). The sender must first transfer the cryptocurrency to the BPSC wallet for the transaction to be successful in the network; the BPSC will then automatically send the same cryptocurrency to the wallet of the final recipient.

A. Working of Smart Blockchain

Any successful transaction in a smart blockchain is dependent upon the presence of one or more Block Producer Smart Contracts (BPSC) within the decentralized network. For a transaction to be successful on the network, the sender must first send the cryptocurrency to BPSC's wallet, after which BPSC will automatically transmit it to the recipient. With the help of this straightforward action, the BPSC was able to obtain accurate and flawless data about every network transaction. As a result, the BPSC was able to record this data into a new block and add it to the end of the blockchain, eliminating the need to assign tasks to block producers and miners.

Smart blockchain also offers the ability to categorize transactions according to their subject. For instance, it permits the trade of multiple coin types within a network made up of separate blockchains. The network's BPSC count will be increased to achieve this. The fact that transactions might be assigned to different BPSCs with unrelated subjects is both a drawback and an important factor. For instance, if we want a smart contract to provide a reward for each of the 100 successful transactions, then the cryptocurrency transactions and the total number of successful transactions for that project should be recorded, registered, and maintained by a single BPSC and added to a single blockchain.

B. Benefits of Smart Blockchain

The correct and precise transaction details are recorded and saved at the time of the transaction. This eliminates the need to create several alternatives from a block, as is the case with Bitcoin, Ethereum, and other networks, and allows miners to select the right and legitimate block with minimal financial expense, energy consumption, and the use of multiple proof methods.

C. Uses of Smart Blockchain

Every new smart contract can be easily registered and deployed on a separate blockchain, which is what we can do with networks like Ethereum, EOS, and others. In this manner, a BPSC will be launched simultaneously with the new smart contract, and all the smart contract's transactions will be recorded and registered into the BPSC's ledger right away. In this scenario, the dedicated blockchain will function independently and won't require any data from the Ethereum blockchain, EOS, etc. This BPSC can be coded independently and launched in between, or its codes can be imported into the appropriate smart contract codes.

II. METHODOLOGY

A. Technologies Used

Python is used as the de facto programming language.

Flask is used to make each node act as a server and connect them with each other.

Postman is used to test the API of a webserver created by Flask using Python.

B. Testing Environment

Three nodes on a single device (ports 5001, 5002, and 5003) are implemented using three Python scripts. This can also be performed on different devices simply by executing scripts on different devices and making minor changes to them.

SHA256 has been used as a hashing function across all nodes by default to maintain uniformity. Different hashing functions can also be used across nodes.

There is no proof-of-work (PoW) or proof-of-stake (PoS) mechanism in this blockchain since there is no consensus.

C. Potential Outcome

Smart blockchain is based on the concept of smart contracts spread across multiple nodes. Since all the nodes are interconnected, changes made on either of the nodes are synced across all the nodes.

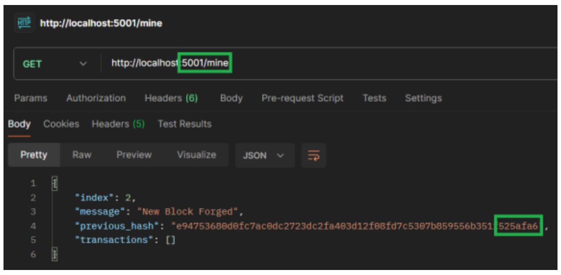
A smart contract gains permanence once it is recorded on the blockchain network. That is, no human being will be able to alter the contract's original text in any way if the blockchain network is operational. The same applies to block producer smart contracts (BPSCs).

III. IMPLEMENTATION

The "smart blockchain" supports features such as mining a new block, attaching the block to the chain, and inserting custom data into the chain. All basic blockchain features have also been implemented in this system.

A. Mining of Blocks

Blocks can be mined from either of the nodes.



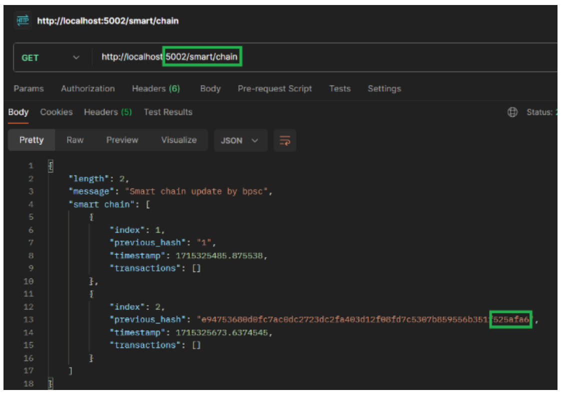
```
GET http://localhost:5001/mine
```

```
{
    "index": 2,
    "message": "New Block Forged",
    "previous_hash": "e94753680d0fc7ac0dc2723dc2fa403d12f08fd7c5307b859556b351f525afa6",
    "transactions": []
}
```

Figure 1: Mining of Block on Node 1

B. Synchronization of Blocks

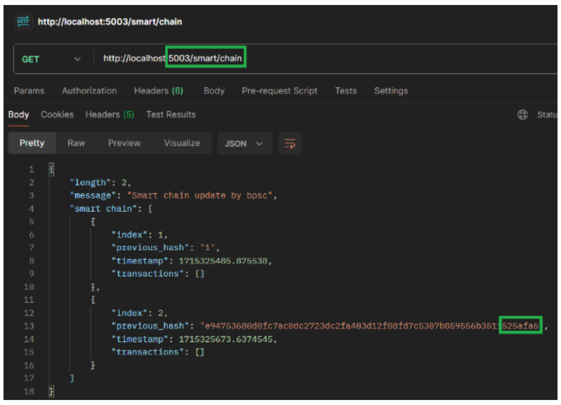
Blocks mined on either node will be synchronized across the entire blockchain.



```
GET http://localhost:5002/smart/chain
```

```
{
    "length": 2,
    "message": "Smart chain update by bpsc",
    "smart chain": [
        {
            "index": 1,
            "previous_hash": "1",
            "timestamp": 1715325485.875538,
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "e94753680d0fc7ac0dc2723dc2fa403d12f08fd7c5307b859556b351f525afa6",
            "timestamp": 1715325673.6374545,
            "transactions": []
        }
    ]
}
```

Figure 2: Sync of Blocks on Node 2



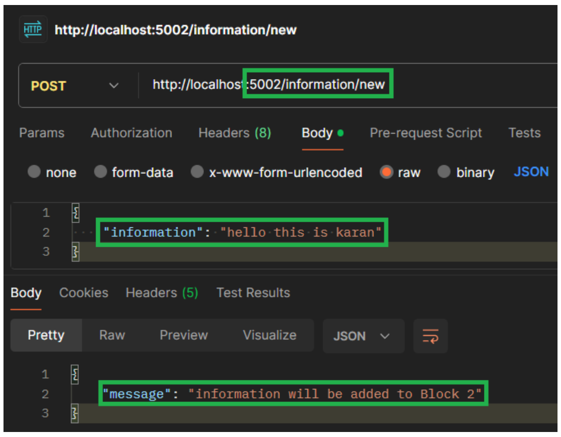
```
GET http://localhost:5003/smart/chain
```

```
{
    "length": 2,
    "message": "Smart chain update by bpsc",
    "smart chain": [
        {
            "index": 1,
            "previous_hash": "1",
            "timestamp": 1715325485.875538,
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "e94753680d0fc7ac0dc2723dc2fa403d12f08fd7c5307b859556b351f525afa6",
            "timestamp": 1715325673.6374545,
            "transactions": []
        }
    ]
}
```

Figure 3: Sync of Blocks on Node 3

C. Insertion of Custom Data in Blocks

Custom data and transactions can be inserted and mined into blocks in JSON format.


http://localhost:5002/information/new

POST http://localhost:5002/information/new

Params Authorization Headers (8) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON

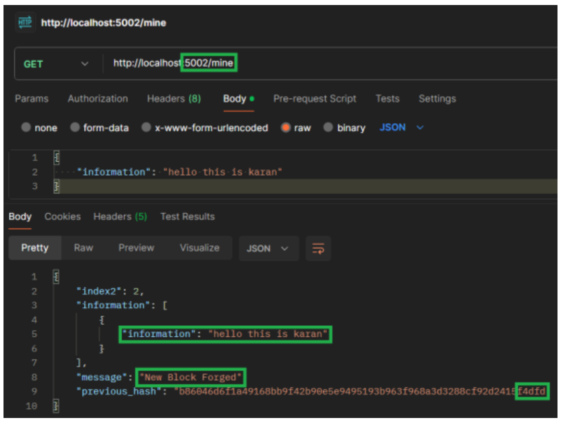
```
{
    "information": "hello this is karan"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "information will be added to Block 2"
}
```

Figure 4: Inserting Block with Custom Message


http://localhost:5002/mine

GET http://localhost:5002/mine

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

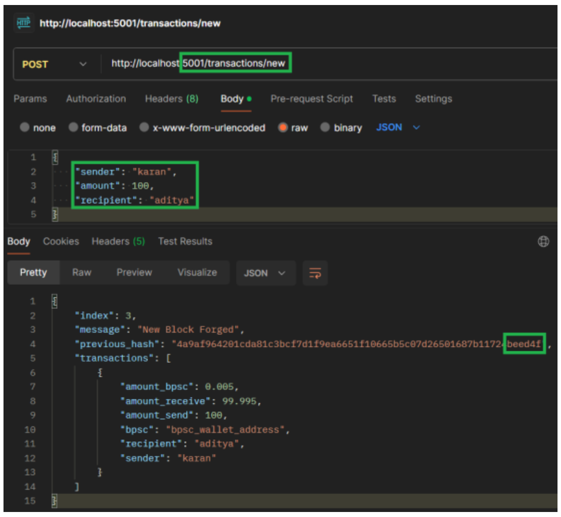
```
{
    "information": "hello this is karan"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "index2": 2,
    "information": [
        {
            "information": "hello this is karan"
        }
    ],
    "message": "New Block Forged"
    "previous_hash": "b86046d6f1a49168bb9f42b90e5e9495193b963f968a3d3288cf92d2415f4dfd
}
```

Figure 5: Mining Block with Custom Message


http://localhost:5001/transactions/new

POST http://localhost:5001/transactions/new

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
{
    "sender": "karan",
    "amount": 100,
    "recipient": "aditya"
}
```

Body Cookies Headers (5) Test Results

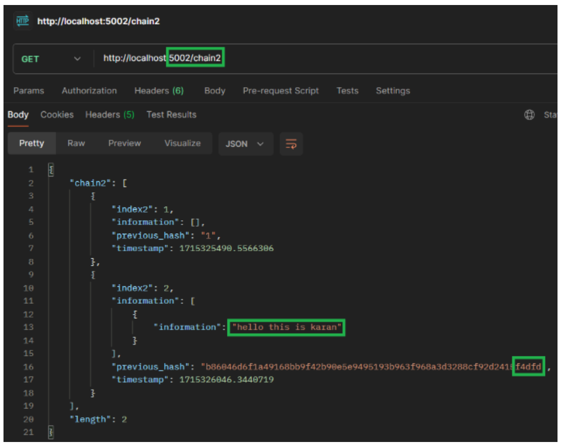
Pretty Raw Preview Visualize JSON

```
{
    "index": 3,
    "message": "New Block Forged",
    "previous_hash": "4a9af964201cda81c3bcf7d1f9ea6651f10665b5c07d26501687b1172beed4f",
    "transactions": [
        {
            "amount_bpsc": 0.005,
            "amount_receive": 99.995,
            "amount_send": 100,
            "bpsc": "bpsc_wallet_address",
            "recipient": "aditya",
            "sender": "karan"
        }
    ]
}
```

Figure 6: Inserting Block with Custom Transaction

D. Synchronization of Custom Data in Blocks

Custom data and transactions will be synchronised across all nodes. Synchronisation works like in the previous section.


http://localhost:5002/chain2

GET http://localhost:5002/chain2

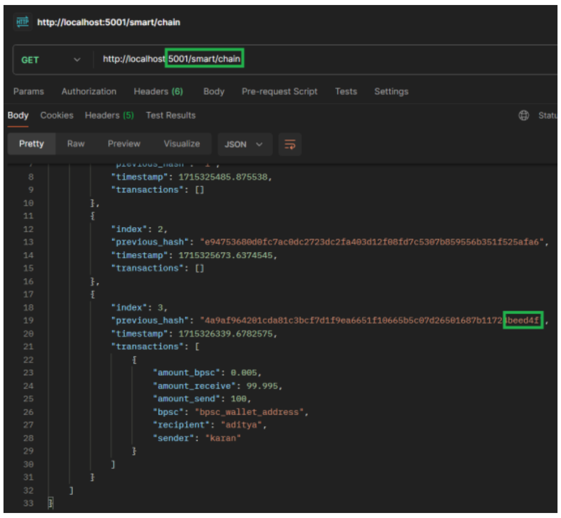
Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "chain2": [
        {
            "index2": 1,
            "information": [],
            "previous_hash": "1",
            "timestamp": 1715325490.5566306
        },
        {
            "index2": 2,
            "information": [
                {
                    "information": "hello this is karan"
                }
            ],
            "previous_hash": "b86046d6f1a49168bb9f42b90e5e9495193b963f968a3d3288cf92d2415f4dfd",
            "timestamp": 1715326046.3440719
        }
    ],
    "length": 2
}
```

Figure 7: Synchronization of Custom Message on Node 2


http://localhost:5001/smart/chain

GET http://localhost:5001/smart/chain

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
                "timestamp": 1715325485.875538,
                "transactions": []
            },
            {
                "index": 2,
                "previous_hash": "e94753680d0fc7ac0dc2723dc2fa403d12f08fd7c5307b859556b351f525afa6",
                "timestamp": 1715325673.6374545,
                "transactions": []
            },
            {
                "index": 3,
                "previous_hash": "4a9af964201cda81c3bcf7d1f9ea6651f10665b5c07d26501687b1172beed4f",
                "timestamp": 1715326339.6782575,
                "transactions": [
                    {
                        "amount_bpsc": 0.005,
                        "amount_receive": 99.995,
                        "amount_send": 100,
                        "bpsc": "bpsc_wallet_address",
                        "recipient": "aditya",
                        "sender": "karan"
                    }
                ]
            }
        ]
}
```

Figure 8: Synchronization of Custom Transaction on Node 3

IV. REFERENCES

- https://hackernoon.com/what-is-smart-blockchain-4b134275e90f
- https://hackernoon.com/how-to-build-a-smart-blockchain-that-prevents-double-spending-a-step-by-step-guide-vw9m33aq
- https://python.plainenglish.io/implementing-a-smart-blockchain-with-python-3183e0c1052e