iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 7 **Issue:** IX **Month of publication:** September 2019

DOI: http://doi.org/10.22214/ijraset.2019.9105

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.177
Volume 7 Issue IX, Sep 2019- Available at www.ijraset.com

Voting Application using Blockchain Technology

Ranjana Vittal[1]
[1]*Electronics and Communication Department, PES University*

Abstract: ***The aspect of electoral fraud and lack of transparency in the conventional voting system employed in India is a major problem that requires attention. The objective is to implement a secure e-voting system based on blockchain technology that counteracts the above-mentioned issues. The aim is to fight electoral malpractices by developing a system which will keep data secure in digital voting and prevent tampering of data while introducing a level of transparency in the system. One might argue that online voting is susceptible to database manipulation, however the adoption of blockchain technology solves this by its inherent property of decentralization wherein data is not stored in a central database, rather distributed over a vast network of peered systems. Electoral fraud in the form of voter impersonation and voting multiple times can be solved by the use of biometric voter authentication which has been implemented in the prototype of an Electronic Voting Machine (EVM). This paper discusses the design and implementation of the system proposed.***
Keywords: Blockchain, Ethereum, Smart Contract, Voting Application, Peering, Node, Arduino

I. INTRODUCTION

1) As a country of 1.3 billion people, more than 800 million of whom are eligible to vote, India is the world's largest democracy. In the current voting process, the voter has to show his voter ID card when he goes to cast his vote. This is a time consuming process since the authority in the polling station has to check the voter ID card against the list present, confirm its authenticity and then allow the person to cast his vote. This still does not guarantee a fair process due to fake voter ID cards. Thus, to avoid problems of this kind, our system of fingerprint based voting machine using blockchain technology will simplify the voting process and the person need not carry his voter ID which contains his personal details.

2) Coming to vote tampering, data held in traditional ledgers are maintained by a single organization which can tamper with the stored data or add data fraudulently. Even if the organization would not make malicious changes to the data, there is a possibility that external agents like hackers could manipulate it to their own means.

3) In order to ensure that no single organization can manipulate the database, each authorised entity can store a redundant copy of the database. Every entity in the network can be assured that their copy of the database is legitimate and updated by comparing it with others in the network and implementing the consensus algorithm by which the transactions are validated, ensuring that the data contained with them are exact copies of one another. This is where blockchains come into picture to prevent tampering once the data goes into the blockchain.

II. IMPLEMENTATION

A. Blockchain, Ethereum and Smart Contract

1) A blockchain is a growing list of blocks which are packages of data that carry permanently etched data on the blockchain network. These blocks are linked and secured using hash cryptography. It is similar to a database which stores information, but the difference is that data is located in a network of personal computers called nodes and there is no central entity controlling the data.

2) Each block contains transaction data, a timestamp and hash pointer as a link to a previous block. The initial block is known as the Genesis block and is hardcoded into the software. A hash function is used to map any alphanumeric data of arbitrary size to data of fixed size. The copies of the transaction are hashed, the hashes are paired and again hashed and this is continued until only one hash remains which is called Merkle root.

3) In this model, a private blockchain has been developed in Ethereum which is an open source, distributed, blockchain-based ecosystem used to store the votes cast. Ethereum implements a virtual machine. The virtual machine resides in the Ethereum node (participant in the blockchain network with a copy of the ledger) application known as a wallet.

4) Ethereum can understand a broad range of instructions making it possible to manage digital or 'smart' contracts. A smart contract is a computer protocol which digitally facilitates and verifies the performance of credible transactions which are trackable and irreversible. By using Solidity programming language, instructions are written which tell the Ethereum virtual machine in the node application to enforce the terms of the contract consuming and releasing a digital token called 'ether' as

part of the transaction. Each contract has an associated hexadecimal address. The contract cannot be tampered with because of the cryptographic integrity of data on the Ethereum blockchain.

5) In the smart contract (referred to as “voting.sol” ahead) used in this model, a field called “mapping” has been used which acts as the cryptographic hash. In this model, the mapping key is the name of the party standing in the election which is stored as type bytes32 and the value is unsigned integer in order to store vote count. To store the list of parties, an array of bytes32 is used. The constructor is called once when the contract is deployed to the blockchain and an array of parties contesting in the election is passed.
6) There are three methods used in the smart contract code. One to return the total votes a party has received so far, second to cast a vote and third to increment the vote count for the specified party.
7) For smart contracts, Ethereum Virtual Machine is the runtime environment. It is a 256-bit register stack and is the fundamental consensus mechanism for Ethereum. Each node in the Ethereum network runs an implementation of the Ethereum Virtual Machine and executes the same instructions. Smart contracts are compiled to Ethereum Virtual Machine bytecode and deployed to the blockchain.
8) In order to deploy the smart contract to the blockchain, an account needs to be created in Ethereum. The account addresses are composed of the prefix "0x" which is a common identifier for hexadecimal and contain 40 hexadecimal digits, with 2 digits to representing a byte. For example, 0xb894E5eB0aa36794cE839633ff2BA04249559278.
9) Contract addresses have the same format, but they are determined by sender and creation transaction nonce which is a random number issued in an authentication protocol to make sure that old communications are not used again in replay attacks. User and contract accounts cannot be distinguished given just an address for each without any blockchain data.
10) In the private blockchain developed in Ethereum, read permissions can be public or restricted to an extent while write permissions are centralized to a single entity thereby ensuring security.
11) The first step to build the private blockchain is creating a genesis JSON file. The genesis file is used to initiate the first block of the chain. It is hardcoded with certain parameters that will define the characteristics of the rest of the blockchain. All the nodes in the private blockchain are initiated using the same Genesis file. Given below is a figure showing the Genesis file parameters used.

```
"config": {
    "chainId": 1476,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0
},
"difficulty": "10",
"gasLimit": "2000000",
"alloc": {
    "0xbA95723CeBEfc4c79aeDec70BaA99Bc1859535bA": {
        "balance": "1000000000000000000000000"
    },
    "0x139CDeF62B158DfE9B06d6746d7deF35B02961d3": {
        "balance": "1200000000000000000000000"
    }
}
```

Fig 1: Genesis File

12) The Genesis File parameters are explained as follows:
a) Config: This has the blockchain identifier number which can be any number we want. The rest of the parameters under it like homesteadBlock, eip155Block, eip158Block, byzantiumBlock which relate to blockchain versioning and forking are set to 0 since a new private blockchain is being started.
b) Difficulty: It dictates how difficult it is to validate the transaction and add it to the blockchain. A low value has been kept here in the project just for demonstration purpose, but the difficulty values can go upto 10000 on test networks.
c) gasLimit: Gas is a unit in Ethereum which tells how much "work" a set of actions takes to perform. gasLimit is the amount of gas each block can use, so this value has been kept high enough so that it does not slow down the network.
d) alloc: It has account addresses with some ether balance allocated to every account and each of these addresses are used to deploy the contract onto the blockchain from one particular polling booth (terminal). Ether is like a token for any transaction to take place. The balance can be any higher order number (x1023).

13) Geth is the command line interface to run an Ethereum node. It is launched with the JavaScript Console that provides a runtime environment bringing out a JavaScript application program interface (API) to interact with the node. This JavaScript Console API includes web3 Decentralised API. Each Ethereum node will act like a polling booth in an election scenario. There are two steps involved in this:

a) Instantiating Data Directory: geth --datadir ./DataDir1 init ./Genesis.json
This specifies where the blockchain, keystore, etc needs to be saved.
--datadir is used to configure the location of the data directory.

b) Starting the Ethereum Node: geth --datadir ./DataDir1 --networkid 1634 console 2
Network ID can be any number of our choice, but other peers joining the network must use the same ID. This ensures privacy of the network.

14) The output of this command will open the geth Javascript console. In the Javascript console, we compile and deploy the contract to the blockchain as well as add other peers to our network. A screenshot of the console is given below:

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.2-stable/darwin-amd64/go1.10
coinbase: 0x126d157d0c7834db349b1020e5b111a90fa63817
at block: 0 (Thu, 01 Jan 1970 05:30:00 IST)
 datadir: /Users/RanjanaVittal/my-eth-chain/peer3DataDir
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txp
ool:1.0 web3:1.0
```

Fig 2: Geth JavaScript Console

B. Peering

1) Peering is a process by which multiple networks connect and exchange traffic. This useful concept allows direct traffic hand off between two interacting parties, without having to involve an external agent to carry that traffic for them. Peering between two networks is instantiated by a two-way physical connection that is an exchange of routing information through the Border Gateway Protocol (BGP).

2) Private peering is performed by creating a direct physical connection between two networks. The connection is made from only one network to another network.

3) Peering is used to add more nodes to our private blockchain network, that is, to create a scenario of multiple polling booths in an election where each Ethereum node(enode) is representative of a polling booth. Each node is communicating with the other node and are in sync.

4) Suppose two votes are cast at the same time in different polling booths/enodes of the same constituency, they are logged into the blockchain without collision since each vote has a separate transaction ID and has a different source enode.

5) Peering is done as per the steps discussed below.

a) Instantiating a new data directory:
geth --datadir ./peer2DataDir1 init ./Genesis.json
The same command which was used to initiate the data directory of the first node is applicable here.

b) Launching the second peer:
geth --datadir ./peer2DataDir1 --networkid 1634 --port 30304 console 2>> myeth2.log
The second peer is launched using the same network ID as the first node but using a different port number. The first node is using port 30303. This command opens the Javascript console.

Fig 3: IP Address and Port Number of the two peered nodes

c) In the Javascript console of the first node, admin.nodeInfo.enode will give the enode address.

d) In the Javascript console of the second peer, admin.addPeer() with the enode address as the parameter will make sure that the this peer is now a part of our private network.

e) In the Javascript console of the second peer, admin.peers will give an output which will confirm that the two nodes are now peered and fit for communication.

6) The nodes on the private ethereum network communicate with each other over IPC (Inter Process Communication). In the ethereum space, IPC normally involves geth creating a IPC pipe (which is represented by the file $HOME/.ethereum/geth.ipc). BootNodes are used as an entry to the network from where a node discovers the other nodes in that network.

7) Similarly, any number of nodes can be peered, initiated using the same network ID and a different port.

8) Data is decentralized across its peered nodes. Every node is independent; if one node fails, the network can still be run by the other nodes. Therefore, this model is more resilient, transparent, flexible, distributed and have more incentivized structure than existing software models.

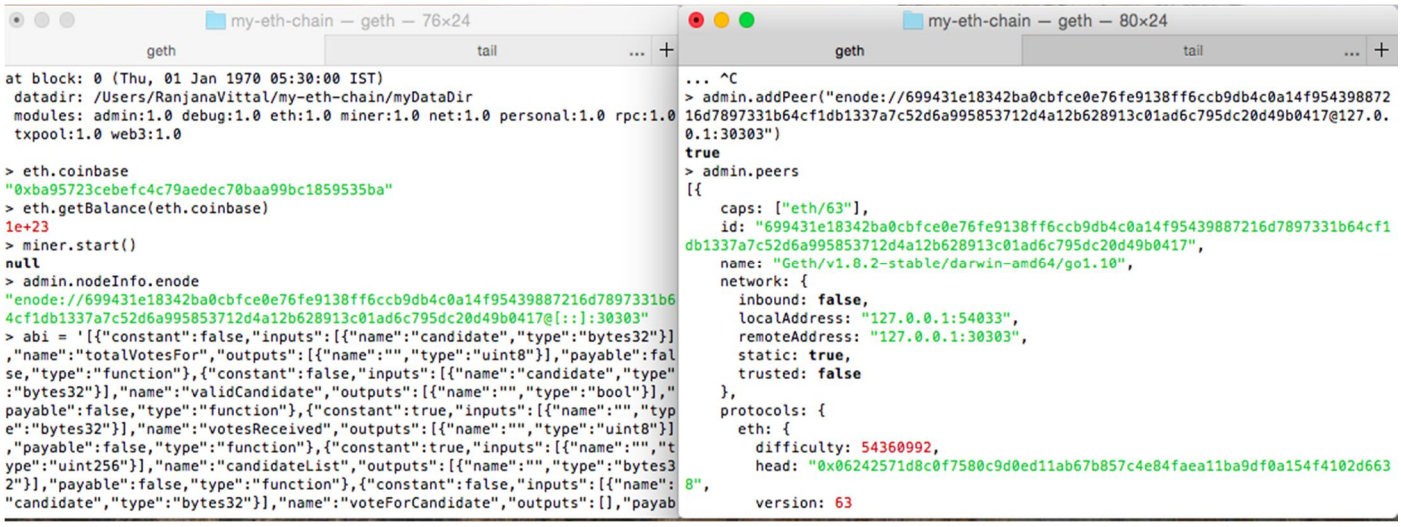


Fig 4: Two nodes peered

C. Contract Compilation and Deployment

1) To compile the contract code written in solidity language, a npm module (package manager for the scripting language Javascript) called solc is required. This library is used within the node console to compile the contract.

2) The contract is saved with the extension '.sol'.

3) After initialising the solc and web3 objects, the contract code is loaded from .sol to a string variable and then compiled.

4) Once the code is successfully compiled and the contract object is printed in the node console, two important fields are obtained - ABI and bytecode.

a) ABI (Application Binary Interface):
This is an interface or template which provides details regarding the methods available in the contract. In the future, this ABI definition is required to interact with the contract.

b) Bytecode:
It is obtained when the source code in voting.sol is compiled. This is the code which will be deployed to the blockchain.

5) The ABI and bytecode thus obtained are key elements for deployment of the contract. They are keyed in the console for deployment.

6) Then, the account corresponding to this node is unlocked using the personal.unlockAccount() function through which the contract will be deployed into the blockchain from this particular node.

7) Once deployed, the methods in the smart contract can be called to either cast a vote or display the vote count.

8) The information required for an account in Node 2 accessing this deployed contract are the ABI and its deployed address obtained from Node 1. The ABI and deployed address are keyed in the console to deploy the contract in Node 2. Node 2 can use the methods of the contract similar to Node 1 and the two will be in sync.

D. Interacting with the Contract through a Web Application

1) To connect to the blockchain nodes from the web application Ethereum's Javascript API called Web3.js is used which implements the Generic JSON RPC. This is a lightweight and stateless remote procedure call (RPC) protocol and a data interchange format. It is transport agnostic which means that it can be used over HTTP, sockets and various message passing environments as well as within the same process. Connection and interaction with a remote or local ethereum node can be done using HTTP or IPC connection.

2) The front end web user interface is built using HTML and JavaScript. The Election commission can monitor the proceeding in real time using this interface and only they will have access to the votes stored in the blockchain in order to protect the identity of a voter and in continuum of the concept of secret ballot.

3) The Javascript API acts as a bridge between the web application and the blockchain. In the code a web3 object has been defined and declared the JSON-RPC endpoint which by default is http://localhost:8545. After which two functions have been defined which internally call the methods defined in the smart contract. The code collects the vote from the HTML page and calls the method defined in the smart contract using RPC, to transact it into the blockchain.

E. Electronic Voting Machine Prototype

1) The Electronic Voting Machine Prototype has been created by using the microcontroller Arduino Genuino Uno, Fingerprint Module R305, 16x2 LCD, Buzzer and Sliding Switch. Given below is the circuit diagram and an image of the prototype.

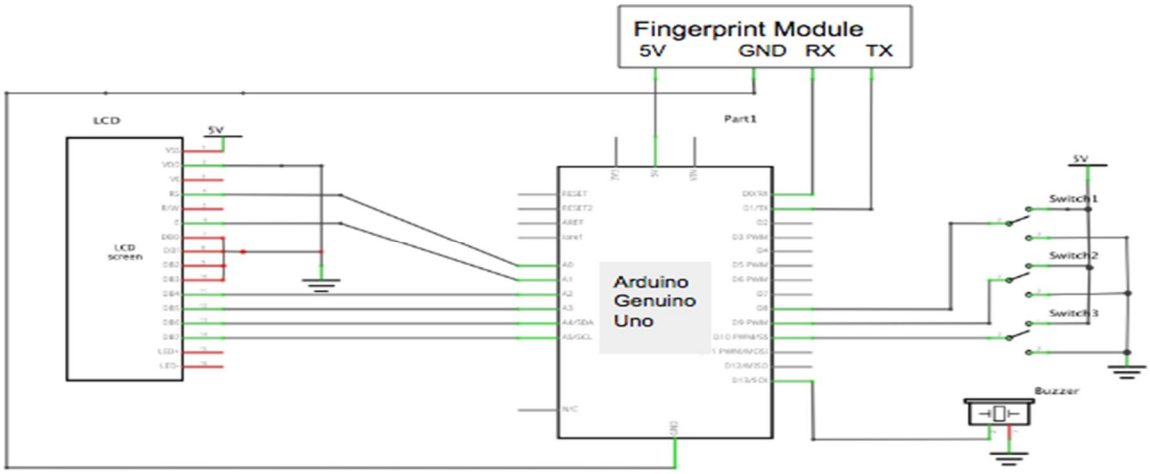


Fig 5: Circuit Diagram of Electronic Voting Machine Prototype

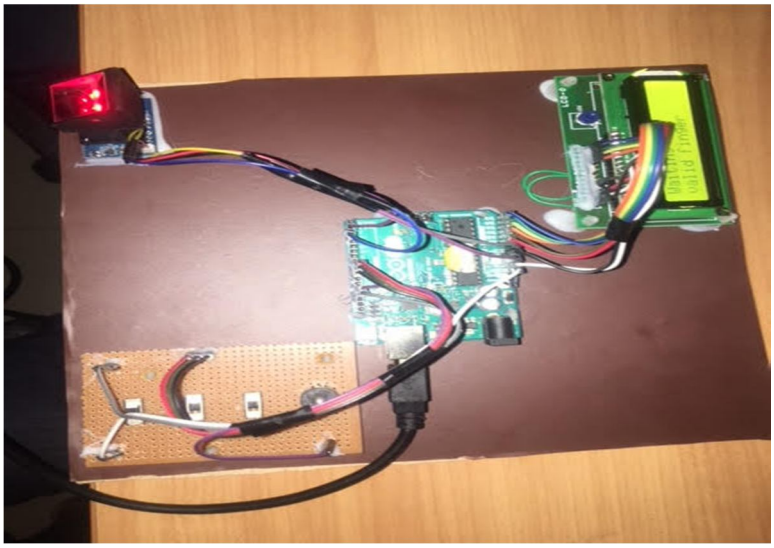


Fig 6: Prototype of Electronic Voting Machine

2) First, the voter needs to be enrolled in order to cast his/her vote. The fingerprint of a voter is stored in the memory unit of the fingerprint module. There is also a feature to delete the stored fingerprint in case the voter has changed his place of residence and is then a part of a different constituency.

3) The voter places his/her finger on the module. If it matches the pre-stored values, then the voter is allowed to cast a vote using the sliding switches. Once the fingerprint is authenticated the voter casts his vote and he should not be allowed to vote again. In case the fingerprint does not match, a buzzer beeps and disallows the voter from casting a vote.

4) Sliding switches with the names of the contesting parties (A, B, C etc) are provided to the voter to cast their vote. The input received through the sliding switch is sent to the serial port of the computer.

5) A script runs on the host machine and constantly monitors the serial port for inputs from the sliding switch. For instance, when Switch 1 is high, the script reads 1 in the serial port and fills 'Party A' in the text box of the HTML form. Similarly, when Switch 2 is high, the script reads 2 in the serial port and fills 'Party B' in the text box of the HTML form and so on.

6) Therefore, when the input/vote is sensed at the serial port, it fills the 'Party Name' textbox of the HTML document with values corresponding to that input and auto-submits the form.

7) A web page is also created to display the vote count and this vote count is stored in the private blockchain.

III.CONCLUSIONS

In developing nations like India, there is a genuine problem of vote tampering and a lack of transparency in elections. The focus of the paper has been tackling these very issues with the use of a secure, trustworthy technology like Blockchain. In the system that is proposed in the paper, a smart contract deployed on a private blockchain provides the functionality of securing each vote into a blockchain by means of a transaction and subsequently allowing the blockchain to be queried for the vote count at any point in time. The prototype developed adeptly represents an election scenario in a constituency with five polling stations, consisting of one polling booth in each station. Each polling booth acts as a peer node in the private blockchain network and has a copy of the main blockchain. In addition to this, the prototype of EVM is enabled with biometric (fingerprint) voter authentication which will ensure impossibility to vote twice or to commit electoral fraud.

REFERENCES

[1] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564.

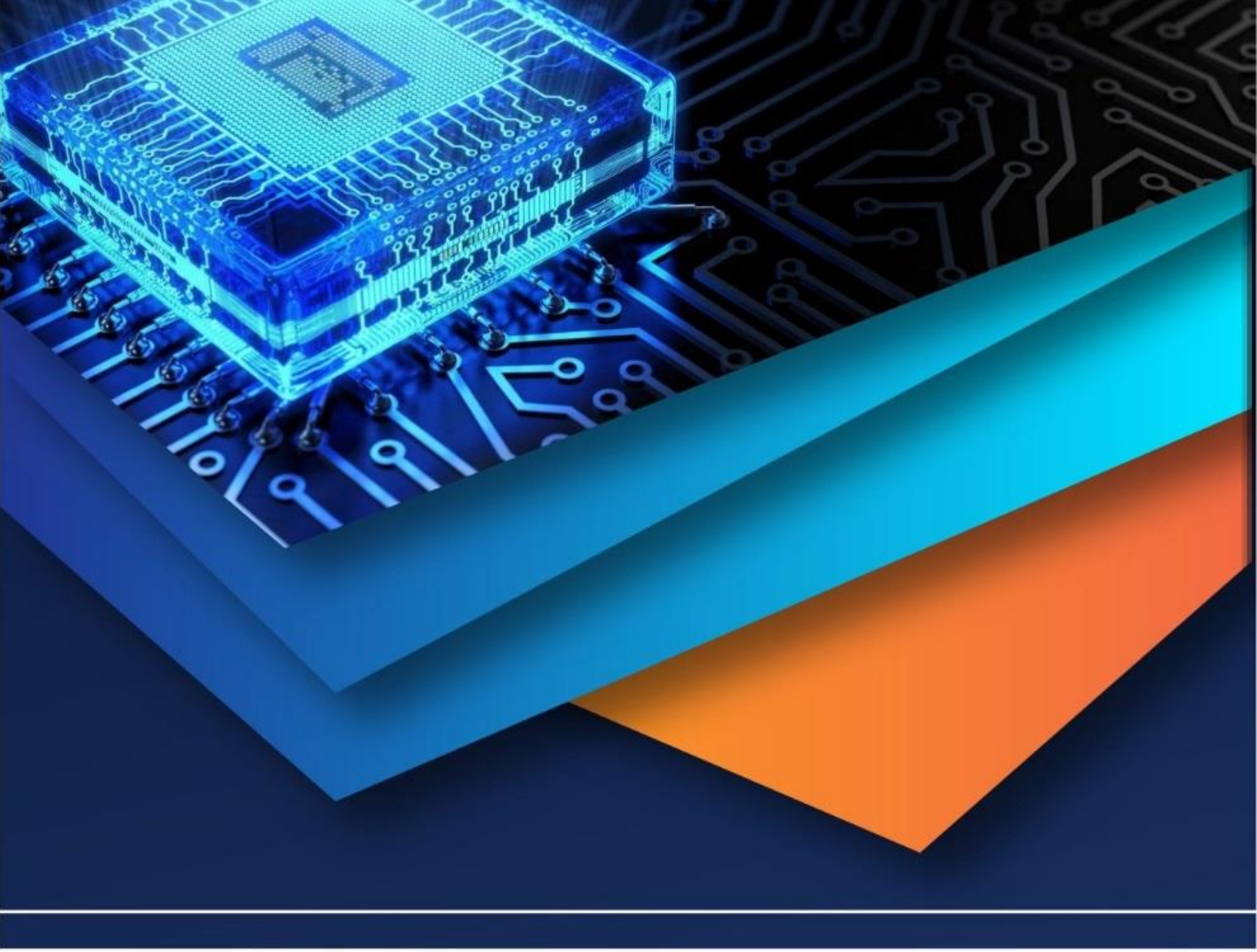
[2] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," in *IEEE Access*, vol. 4, pp. 2292-2303, 2016.

[3] I. Weber et al., "On Availability for Blockchain-Based Systems," 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), Hong Kong, 2017, pp. 64-73.

[4] C. K. Frantz and M. Nowostawski, "From Institutions to Code: Towards Automated Generation of Smart Contracts," 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), Augsburg, 2016, pp. 210-215.

[5] Blais, A., E. Guntermann, et al. (2016). "Linking Party Preferences and the Composition of Government: A New Standard for Evaluating the Performance of Electoral Democracy." Political Science Research and Methods: 17.

[6] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Renesse, Emin Gün Sirer. "Decentralization in Bitcoin and Ethereum Networks", In Proc. of: Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer, 2018.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY