Configurable Smart Contracts Automation for EVM based Blockchains

ZAIN UL ABEDIN Technical Team Lead, Nugenesis Pty, Ltd Australia, zainmustafaaa@hotmail.com

Muhammad Shujat Ali IT Manager, Orange Networks, Shujat@organnetworks.com

Ashraf Ali The Institute of Management Sciences, ashraf@pakaims.edu.pk

Sana Ejaz Assistant Prof & In-Charge Mphil (CS) M.Phil CS, FAST National University of Computer and Emerging Sciences, sani.ejaz.a@gmail.com

Summary
Electronic voting machines (EVMs) are replacing research ballots due to the errors involved in the manual counting process and the lengthy time required to count the votes. Even though these digital recording electronic systems are advancements, they are vulnerable to tampering and electoral fraud. The suspected vulnerabilities in EVMs are the possibility of tampering with the EVM's memory chip or replacing it with a fake one, their simplicity, which allows them to be tampered with without requiring much skill, and the possibility of double voting. The vote data is shared among all network devices, and peer-to-peer verification is performed to ensure the vote data's authenticity. To successfully tamper with the system, all of the data stored in the nodes must be changed. This improves the proposed system's efficiency and dependability. Elections and voting are fundamental components of a democratic system. Various attempts have been made to make modern elections more flexible by utilizing digital technologies. The fundamental characteristics of free and fair elections are intractability, immutability, transparency, and the privacy of the actors involved. This corresponds to a few of the many characteristics of blockchain-like decentralized ownership, such as chain immutability, anonymity, and distributed ledger. This working research attempts to conduct a comparative analysis of various blockchain technologies in development and propose a 'Blockchain-based Electronic Voting System' solution by weighing these technologies based on the need for the proposed solution. The primary goal of this research is to present a robust blockchain-based election mechanism that is not only reliable but also adaptable to current needs.

Keywords: *Smart Contracts, Automation, EVM, Blockchains, Solidity, Ethereum, Hyperledger, JSON*

1. Introduction

At this point, the use of technology in meeting human needs has become commonplace. As most people today do not trust their governments, the increasing use of technology has created new challenges in the democratic process, making elections critical in a modern democracy. Elections have enormous power in determining the fate of a country or organization. Blockchain technology is one solution that can be used to reduce voting problems. The blockchain is a distributed, immutable, and transparent ledger that cannot be altered.

Manuscript received October 5, 2023
Manuscript revised October 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.10.18

1.1 Benefits of Blockchain

A. Greater Trust

As a member of a people who are part network, you can belief that you will receive relevant and reliable information after blockchain, and that only network participants that you also expressly granted access will have direct exposure to your sensitive blockchain records.

B. Greater Security

Data accuracy must be agreed upon by all participants in the network, and all confirmed transactions are irreversible as they are completely recorded. Nobody, an action no one, not even the system administrator, can delete it.

C. More Efficiency

Time consuming record reconciliations are eliminated with a distributed network shared by network users. A smart contract is a set of rules that can be located on the blockchain and executed automatically speed up transaction.

1.2 Blockchain Network Types

There are numerous ways to build a blockchain network. They may be created individually, collectively, publicly, or privately.

1.2.1 Blockchain Networks Open to the Public

A public blockchain, like the one used by Bitcoin, is open to everyone. It requires a lot
of computing power, offers little to no privacy during transactions, and has poor Security. These are important factors for the blockchain use cases in the sector.

1.2.2 Private Blockchain Networks

A commercial blockchain network is a decentralized P-to-P network, just like a public blockchain network. On the other hand, the network is run by a single organization that controls who can participate, implements a consensus process, and manages the shared ledger. Depending on the use case, this can significantly raise participant trustworthiness. A private blockchain can be used and even organized on-site within a company's firewall.

1.2.3 Permissioned Blockchain Networks

A permissioned blockchain network is typically set up by businesses that build private blockchains. It’s crucial to keep in mind that public blockchain networks can have permissions as well. This restricts who is allowed to use the network and what transactions they can conduct. Respondents must first receive an invitation or authorization to participate.

1.2.4 Consortium Blockchains

The maintenance of a blockchain could be shared among several businesses. That pre-selected entities decide who can yield to transactions or admittance data. When everyone needs to have approval and share ownership of the blockchain, a consortium blockchain is the best option.

1.2.5 Blockchain Security

Risk management methods are available for blockchain networks. After creating a business blockchain app, it's critical to have a well-thought-out security plan that incorporates cyber security frameworks, declaration services, and finest practices to decrease the risk of spells and fraud.

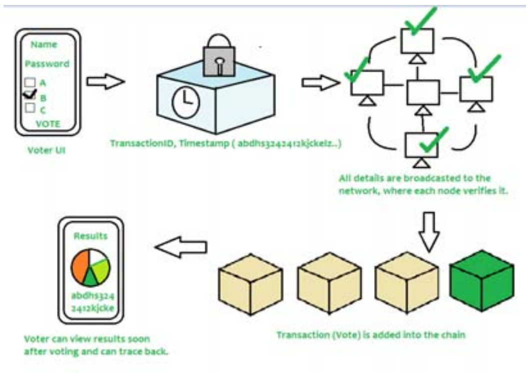

Name
Password
A
B
C
VOTE
Voter UI
TransactionID, Timestamp (abdhs3242412kjckelz..)
All details are broadcasted to the network, where each node verifies it.
Results
abdhs324 2412kjcke
Voter can view results soon after voting and can trace back.
Transaction (Vote) is added into the chain

Figure 1: Working of Blockchain

1.3 Smart Contracts

Contracts are object-oriented languages equivalents of classes, and they can store captured formal variables. Away from each other from the general-purpose types used in traditional programming languages, such as text, integers, static or dynamic arrays, a crucial type in Solidity is the address, which identifies users (EOAs) and extra contracts' locations. Contracts can also have functions that are called from outside the contract. Function convertors can also be applied to many functions to conduct declarative preparatory checks (for example, data validation), mirroring aspects of aspect-oriented programming [5].

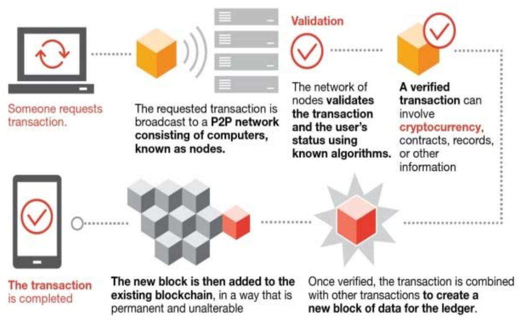

Validation
Someone requests transaction.
The requested transaction is broadcast to a P2P network consisting of computers, known as nodes.
The network of nodes validates the transaction and the user's status using known algorithms.
A verified transaction can involve cryptocurrency, contracts, records, or other information
The transaction is completed
The new block is then added to the existing blockchain, in a way that is permanent and unalterable
Once verified, the transaction is combined with other transactions to create a new block of data for the ledger.

Figure 2: A Simple Voting System is represented by a contract extract

Smart contracts may be specified and encoded very easily thanks to the usage of high-level languages. It's worth noting, though, that smart contracts' potential they are essential infrastructure because they are used as mediators in open systems.

The following table shows how Ethereum and Hyperledger Composer differ in their smart contract syntax:

Table 1: Difference in the Syntax of Smart Contracts between Hyperledger Composer and Ethereum

Parameters	**Hyper ledger Composer**	**Ethereum**
Determine the patient information field's contract state.	Asset Medical Record recognised by recordId -> Patient holder o String recordId -> asset	Defined as a State Variable: bytes32 recordId; Patient private owner;
Define a patient as a contractual party	Participant is defined as: participant Patient recognized by patientId, which is made up of the following elements: string	Defined as State variable or struct: struct Patient{ bytes32 name;

	patientId, string name, integer age, and string gender.	uint age; Gender gender;}
The method that can be used to execute the contract functionality	Defined as Transaction: transaction setPatientName{ o String name }	Defined as Function: function setPatientName(bytes32 name) public{ patient.
A condition that specifies who can access the contract method	Defined as rule in access policy file : Rule PatientSeeUpdateOwnMedicalRecord { description: "Patient can read and update their own record only" participant(t): "org.medical.Patient" operation: ALL resource(v): "org.medical.record" condition:(v.owner.getIdentifier() == t.getIdentifier()) action: ALLOW }	Defined as a Modifier: address public recordOwner; modifier onlyOwner() { require(msg.sender == recordOwner); _; }

Automated Analysis of Smart Contracts

Methodologies for analyzing code on some input data, dynamic code analysis [LSCL12] executes the program but only analyses a subset of all potential execution paths. Static code analysis can offer comprehensive coverage without executing the program and can be performed fast on tiny code segments. A static analysis usually consists of three steps:

(1) creating an integral image (IR) for a more in-depth analysis than text analysis, using algorithms like control- and data access assessment (synonym, constant, and type propagation [ASU07]), taint analysis [TPF+09], symbolic execution, and abstract interpretation;

(2) Enriching the IR with additional information [Wög05];

(3) Vulnerability identification based on a database of patterns, which define as formal specifications of the contract's intended functionality are seldom accessible, we do not discuss formal verification techniques in this section.

Table 2: Smart Check Parameters

Name	Description
Balance equality	Contract logic can be manipulated by an opponent by sending ether against their will. Utilize balances with non-strict inequality
Unchecked external call	There is no verification on the return value. Check method return values at all times.
DoS by external contract	Expect other calls to be intentionally ignored.
send instead of transfer	It is important to verify send's return value. Use transfer, which is akin to throwing if (!send());
Reentrancy	After all local state modifications, outside contracts should be contacted.
Malicious libraries	Using outside libraries might be risky. Avoid external code dependencies, and check every project-related code.
Using tx.origin	A malevolent contract may take action on behalf of a user. For authentication, use msg.sender.
Transfer forwards all gas	All gas is sent by a.call.value ()(), enabling the callee to call back. Use a.transfer(); it only delivers 2300 gas to the callee (insufficient for a callback)
Integer division	Rounding down reduces the quotient. Keep track of it, particularly the ether and token balances.
Locked money	Ether is delivered to the contract, but it cannot be withdrawn. Add a withdrawal feature or refuse payments
Unchecked math	Integer overflow and underflow are conceivable without further checks. Utilize SafeMath
Timestamp dependence	Timestamps can be changed by miners. Make environment-independent vital code.
Unsafe type inference	The lowest integer type is selected through type inference. Explicitly specify types
Byte array	byte[] requires more than bytes
Costly loop	The block gas limit may be exceeded by expensive calculation within loops. Avoid loops with numerous or arbitrary stages.
Token API violation	Where the ERC20 standard anticipates a bool, the contract throws. Rather, return false.
Compiler version not fixed	Future compiler versions can build contracts. Give the precise compiler version.

private modifier	The private modifier simply prevents remote programmes from altering the variable, not to conceal its value.
Redundant fallback function	Fallback for payment rejection is unnecessary. To conserve space, disable the feature that causes payments to be automatically denied.
Style guide violation	Confusion results from inconsistent capitalization. Start event names with uppercase and function names with lowercase.
Implicit visibility level	By default, functions are made public. Keep things clear: Declare visibility level explicitly

Conoscenti et al. (2016) published an SLR at the end of 2016 on the use and flexibility of blockchain with regard to the Internet of Things and other peer-to-peer technologies. Additionally, they claimed that data theft might be detected using the blockchain without the requirement for a centralized reporting mechanism. An SLR outlining the rising impact of blockchain on service schemes was published in 2017 by Seebacher and Schüritz. According to the results of their SLR, blockchain is essential to the functioning of a core service. Most recently in 2018, Reyna et al. (2018) compiled a list of appropriate works in order to identify existing issues and areas for development in the combination of blockchain and IoT sectors. They discovered that the current state of the two technologies together confronts six difficulties: scalability, security, privacy, smart contracts, legal issues, and agreement. They also provided a study to highlight the benefits of integrating blockchains with IoT devices.

2. Methodology

The underlying difficulty that many blockchain-based systems face is scalability [69]. As the chain increases in size, the blockchain validation technique gets more sophisticated and time-consuming. To overcome this issue, we decided to split the blockchain into two parts: smart contracts and transactions. One is for storing and executing smart contracts, while the other is for keeping track of transactions. Parties must first agree on business terms, which are then converted into smart contracts, or contracts that may be executed automatically.

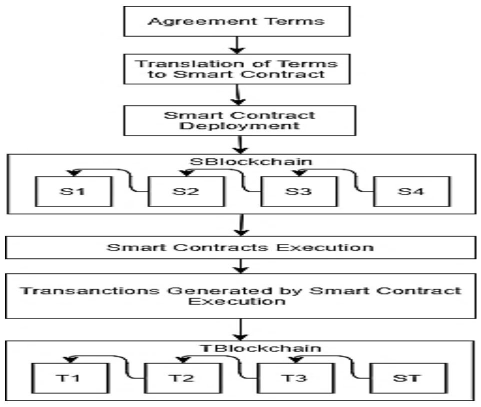

Agreement Terms
Translation of Terms to Smart Contract
Smart Contract Deployment
SBlockchain
S1
S2
S3
S4
Smart Contracts Execution
Transanctions Generated by Smart Contract Execution
TBlockchain
T1
T2
T3
ST

Figure 3: Blockchain-based Smart Contract Management Systems.

Solidity is a high-level object-oriented programming language that may be used to create smart contracts. Account operations in the Ethereum state are governed by smart contracts, which are computer programs. Solidity, a programming language using curly brackets, was created for the Ethereum Virtual Machine (EVM). It is influenced by C++, Python, and JavaScript. Additional details on the languages that have influenced Solidity can be found in the portion on linguistic impacts. Solidity's statically typed nature it supports sophisticated user-defined types, libraries, and inheritance, among other things. You can design contracts using Solidity for things like voting, crowdsourcing, blind bidding, and multi-signature wallets. Use the most recent Solidity version when deploying contracts. Only the most recent version receives security updates, unless there are exceptional circumstances. Furthermore, new features and breaking modifications are added on a regular basis.an algorithm capable of reading the Solidity Smart Contract and then converting it to a JSON file Implemented Solidity version 1.1, which is stored in the JSON file as Solidity version 1.1, as well as any functions it may have, such as a main function with a body of its own, which is also known to that JSON file. Then there will be a second algorithm that will read the JSON file that the first algorithm has created. After that, the second algorithm will parse the JSON file and convert it to a Solidity Smart Contract. In simple words there will be a two-way conversion Smart contract to JSON and from JSON to Smart contract providing the end user with no need of writing any line of code. The JSON object will be able to be updated from any platform or web application so that the user will be able to add features to it that they like and make other changes as required through the automation process that I am trying to build.

The algorithms I am going to write for this project will not be based on any defined standards that already exist. The reason for this is there has not been much work done in creating automated smart contracts like this so my algorithms will be totally custom made for this purpose. Once the algorithms are somewhat successfully made and the dependency of a third party developer is removed there will be no more accidental errors or anything of that sort. In short there will be no human error and we will have a much more cost efficient method of creating contracts in the open source community. This is what I want to achieve and therefore I will research and describe a standard for input variables and function parameters in a JSON structure with the smart contract and make that bundle name i.e. template. Furthermore this system will consider the whole platform where for the end-user, user-interface also included. So that the platform for automation will understand the requirements for that particular template and will provide a nice user interface to the end user which will be based on the input requirement i.e. for boolean it will show the checkboxes etc. So the user will only need to enter the required configurable information for the template and platform transform the smart contract with user provided configurations. End user can also deploy that smart contract onto the ethereum blockchain by providing the account.

2.1 A JSON Parser for Solidity

With the introduction of frameworks such as Truffle and services such as oraclize.it, developing smart contracts in Ethereum has become easier. Using oracles to access information outside of the blockchain, in particular, has become simple enough for contract development. However, the majority of the data that is fed into the blockchain via Oracle transactions is in JSON format. To process the data supplied by an oracle, a smart contract must parse a JSON object. String processing is especially costly on the Ethereum blockchain. As a result, the JSON parser should be "lightweight," requiring little computation to parse and process a JSON string. Because there was no JSON parser for Solidity that I was aware of, I decided to write my own. As a starting point, I ported the code from jsmn to Solidity. Jsmn's main design consideration is to parse a JSON string in a single pass while avoiding copying substrings along the way. As a result, the parser only generates meta data on the provided string, which can later be used to locate and access objects.

2.2 No copying, single pass, fixed memory

The parser operates by parsing the supplied string once, character by character. Along the way, it generates tokens that each identify an object in the string and indicate where it begins and ends. A token has the following structure:

```
struct Token {
  JsmnType jsmnType;
  uint start;
  bool startSet;
  uint end;
  bool endSet;
  uint8 size;
}
```

The JsmnType encodes the type of the token. Valid values are:

```
enum JsmnType { UNDEFINED, OBJECT, ARRAY, STRING, PRIMITIVE}
```

It is worth noting that number, boolean, and null are all treated as JsmnType. PRIMITIVE. They can be distinguished by evaluating the token's first character. The next two values are start and end. They encode the beginning and end positions of the substring that identifies the object. The size of an object indicates how many sub-objects it has (that is the number of children in the JSON hierarchy). For technical reasons, there are two more variables (start Set and endSet) that are false on initialization but become true once the parse has set the values for start or end. They are required because Solidity has a habit of setting all variables to their default values.

2.3 Solidity to JSON Converter

The version of this Solidity programming language will be first to convert into JSON file which is of high significance as both the algorithms will be working on converting files from one form to the other. Rest of the code may differ but version is the single parameter which needs to remain constant as the compiler of Solidity should be the same.

2.4 Solidity to JSON Converter Data Flow

First, open the Read source code file. Sol files are translated by Solidity version 1.1, and the translator validates the syntax. If the source code is valid, the translator will map to the JSON file and write the JSON file according to the object formatter.

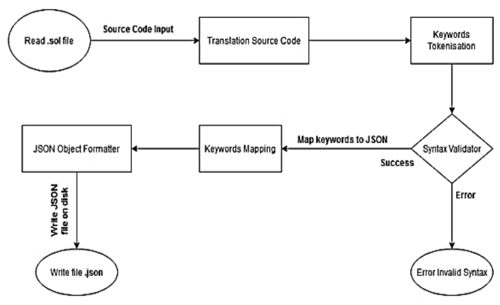


Figure 4: Solidity to JSON Converter Data Flow

2.5 Working Procedure of Solidity to JSON Converter

```
const fs = require('fs-extra');
const path = require('path');
const strip = require('strip-comments');
const parent = [];
const jsonObject = {};
let source = fs.readFileSync(path.resolve(process.cwd(), 'source', 'Ballot.sol'));
source = strip(source.toString());
source = source.split('\n').filter(_ => _ !== '').map(_ => _.trim()).filter(_ => _ !== '');
for(let i=0; i<source.length; i++) {
  const statement = source[i].split(' ').filter(_ => (_ !== '{' || _ !== '}'));
  parseToJson(statement)
}
function parseToJson(statement) {
  switch(statement[0]) {
    case 'pragma': {
      jsonObject['version'] = statement.join(' ');
      break;
    }
    case 'contract': {
      jsonObject[statement[0]] = { name: statement[1], type: statement[0] };
      break;
    }
    case 'function': {
      if(jsonObject.functions) {
        jsonObject.functions.push(statement[1].split('(')[0]);
        jsonObject.method[statement[1].split('(')[0]] = {
          type: 'function',
          header: statement.join(' ').split('{')[0].trim(),
          statements: [],
        }
      } else {
        jsonObject.functions = [ statement[1].split('(')[0] ];
        jsonObject.method = {};
        jsonObject.method[statement[1].split('(')[0]] = {
          type: 'function',
          header: statement.join(' ').split('{')[0].trim(),
          statements: [],
        }
      }
      break;
    }
    case 'struct': {
      jsonObject[statement[1]] = { type: statement[0], statements: [] }
      break;
    }
  }
  if(parent.length > 1) {
    if(jsonObject[parent[parent.length - 1]]) {
      jsonObject[parent[parent.length - 1]].statements.push(statement.join(' '));
    }

    if(jsonObject.method && jsonObject.method[parent[parent.length - 1]]) {
      jsonObject.method[parent[parent.length                              -
1]].statements.push(statement.join(' '));
    }
  }
  if(parent.length === 1) {
    if(statement[statement.length - 1].split(';').length === 2)
      if(!jsonObject.globalDeclarations)      jsonObject.globalDeclarations      =
[ statement.join(' ') ];
      else jsonObject.globalDeclarations.push(statement.join(' '));
  }
  for(let _ of statement) {
    if(_ === '{') parent.push(statement[1]?.split('(')[0]);
    if(_ === '}') parent.pop();
  }
}
console.log (jsonObject.method);
```

A contract is a grouping of Solidity code (the contract's functions) and data (the contract's state) recorded at a single Ethereum blockchain location. The statement **unit** stored Data; declares an **unit** state variable named stored Data (unsigned integer of 256 bits). You may think of it as a single database slot that you can query and modify with database management code operations. The functions set and get, which can be used to alter or retrieve the value of the variable, are defined by the contract in this situation. This is seldom used to retrieve a current contract member (such as a status variable). You may go to Prefix just by typing its name. In contrast to other languages, removing it is not just a question of style; it results in an entirely different means of accessing the member, but more on that later.

2.6 JSON to Solidity Converter

In this step the JSON file will be converted into the solidity. jsmnSol is a port of the jsmn JSON parser to Solidity, originally written in C. Its main purpose is to parse small JSON data on chain. Because string handling is complicated in Solidity and particularly expensive the usage should be restricted to small JSON data. However, it can help to reduce calls to oracles and deal with the responses on chain.

2.7 JSON to Solidity Converter Data Flow

First, open the Read source code file. JSON files are retranslated by Solidity version 1.1, and the translator validates the syntax. If the source code is valid, the translator will map to the JSON file and write the Solidity file according to the object formatter.

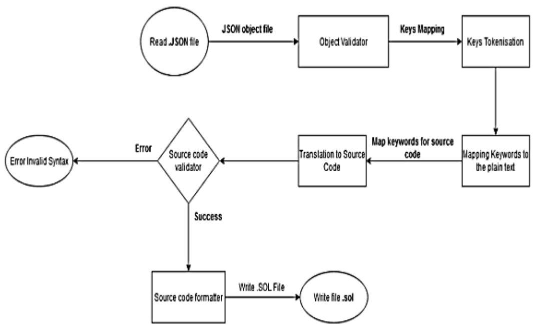
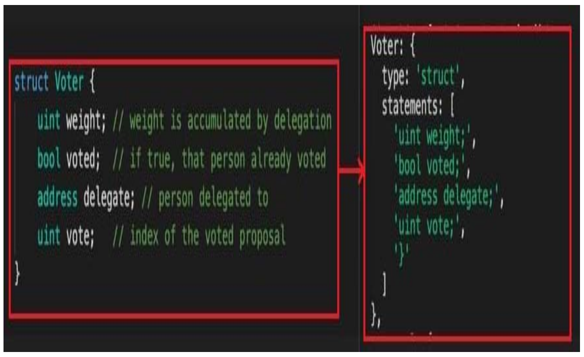


Figure 5: JSON to Solidity Converter Data Flow

2.8 Working Procedure of Solidity to JSON Converter

```
const fs = require('fs-extra');
const path = require('path');
const strip = require('strip-comments');
const parent = [];
const jsonObject = {};
let source = fs.readFileSync(path.resolve(process.cwd(), 'source', 'Ballot.sol'));
source = strip(source.toString());
source = source.split('\n').filter(_ => _ !== '').map(_ => _.trim()).filter(_ => _ !== '');
for(let i=0; i<source.length; i++) {
  const statement = source[i].split(' ').filter(_ => (_ !== '{' || _ !== '}'));
  parseToJson(statement)
}
function parseToJson(statement) {
  switch(statement[0]) {
    case 'pragma': {
      jsonObject['version'] = statement.join(' ');
      break;
    }
    case 'contract': {
      jsonObject[statement[0]] = { name: statement[1], type: statement[0] };
      break;
    }
    case 'function': {
      if(jsonObject.functions) {
        jsonObject.functions.push(statement[1].split('(')[0]);
        jsonObject.method[statement[1].split('(')[0]] = {
          type: 'function',
          header: statement.join(' ').split('{')[0].trim(),
          statements: [],
        }
      } else {
        jsonObject.functions = [ statement[1].split('(')[0] ];
        jsonObject.method = {};
        jsonObject.method[statement[1].split('(')[0]] = {
          type: 'function',
          header: statement.join(' ').split('{')[0].trim(),
          statements: [],
        }
      }
      break;
    }
    case 'struct': {
      jsonObject[statement[1]] = { type: statement[0], statements: [] }
      break;
    }
  }
  if(parent.length > 1) {
    if(jsonObject[parent[parent.length - 1]]) {
      jsonObject[parent[parent.length - 1]].statements.push(statement.join(' '));
    }

    if(jsonObject.method && jsonObject.method[parent[parent.length - 1]]) {
      jsonObject.method[parent[parent.length                                   -
1]].statements.push(statement.join(' '));
    }
  }
  if(parent.length === 1) {
    if(statement[statement.length - 1].split(';').length === 2)
      if(!jsonObject.globalDeclarations)    jsonObject.globalDeclarations     =
[ statement.join(' ') ];
      else jsonObject.globalDeclarations.push(statement.join(' '));
  }
  for(let _ of statement) {
    if(_ === '{') parent.push(statement[1]?.split('(')[0]);
    if(_ === '}') parent.pop();}}
console.log (jsonObject.method);
```



```
struct Voter {
    uint weight; // weight is accumulated by delegation
    bool voted;  // if true, that person already voted
    address delegate; // person delegated to
    uint vote;   // index of the voted proposal
}
```

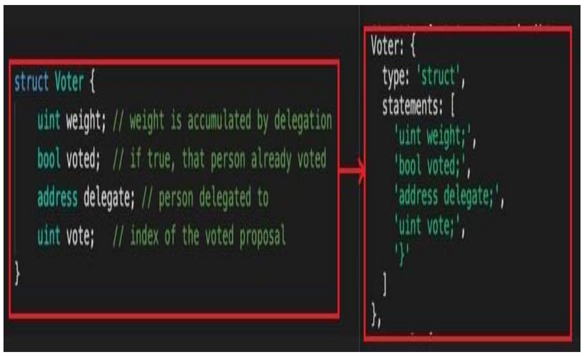
```
Voter: {
  type: 'struct',
  statements: [
    'uint weight;',
    'bool voted;',
    'address delegate;',
    'uint vote;',
    '}'
  ]
},
```

3. Results and Discussion

Investigating the algorithm behavior on the Ballot smart contract, in which users have written to vote for a specific purpose. We have a solidity voter structure where the weight corresponds to some delegation voted status in bool and the voter address as delegate in the end structure is maintaining the voter index with variable vote.

3.1 JSON Implementation of the Structure

On the right side, we can see the JSON implementation of the structure, where the Voter type is struct and some voter struct statements are written in an array with the statements key.



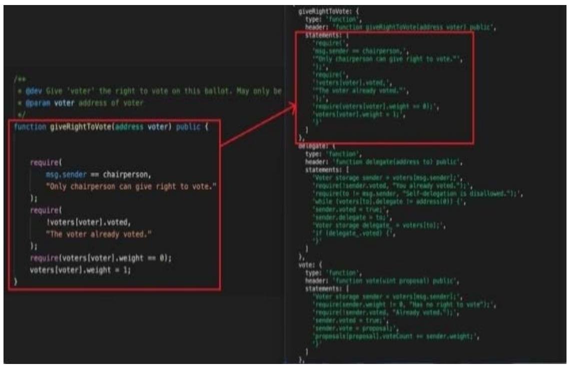
```
struct Voter {
    uint weight; // weight is accumulated by delegation
    bool voted;  // if true, that person already voted
    address delegate; // person delegated to
    uint vote;   // index of the voted proposal
}
```

```
Voter: {
  type: 'struct',
  statements: [
    'uint weight;',
    'bool voted;',
    'address delegate;',
    'uint vote;',
    '}'
  ]
},
```

Figure 6: JSON Implementation of the Structure

3.2 Ballot Smart Contract

Ballot smart contract have some global declarations including chairperson as address with having some lead privileges also a mapping with voter structure maintaining with unique key (address). Algorithm analysis the smart contract and make the global declarations separated in the. JSON implementation with key global Declarations having set of arrays as declarations.



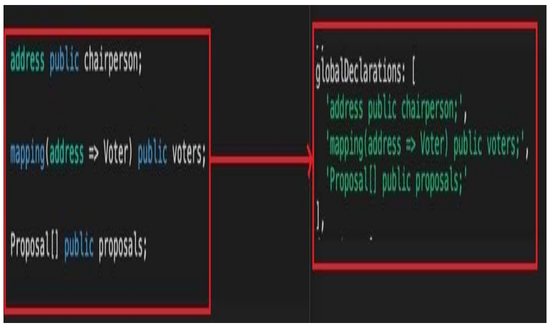
```
/**
 * @dev Give 'voter' the right to vote on this ballot. May only be
 * @param voter address of voter
 */
function giveRightToVote(address voter) public {


    require(
        msg.sender == chairperson,
        "Only chairperson can give right to vote."
    );
    require(
        !voters[voter].voted,
        "The voter already voted."
    );
    require(voters[voter].weight == 0);
    voters[voter].weight = 1;
}
```

```
giveRightToVote: {
  type: 'function',
  header: 'function giveRightToVote(address voter) public',
  statements: [
    'require(',
    'msg.sender == chairperson,',
    '"Only chairperson can give right to vote."',
    ');',
    'require(',
    '!voters[voter].voted,',
    '"The voter already voted."',
    ');',
    'require(voters[voter].weight == 0);',
    'voters[voter].weight = 1;',
    '}'
  ]
},
delegate: {
  type: 'function',
  header: 'function delegate(address to) public',
  statements: [
    'Voter storage sender = voters[msg.sender];',
    'require(!sender.voted, "You already voted.");',
    'require(to != msg.sender, "Self-delegation is disallowed.");',
    'while (voters[to].delegate != address(0)) {',
    'sender.voted = true;',
    'sender.delegate = to;',
    'Voter storage delegate_ = voters[to];',
    'if (delegate_.voted) {',
    '}'
  ]
},
vote: {
  type: 'function',
  header: 'function vote(uint proposal) public',
  statements: [
    'Voter storage sender = voters[msg.sender];',
    'require(sender.weight != 0, "Has no right to vote");',
    'require(!sender.voted, "Already voted.");',
    'sender.voted = true;',
    'sender.vote = proposal;',
    'proposals[proposal].voteCount += sender.weight;',
    '}'
  ]
},
```

Figure 7: Ballot Smart Contract

3.3 Smart Contract Multiple Functions

Also, in the smart contract we have multiple functions one of the major function names as give Right to Vote so we have to pass an address as a parameter and that user can be able to vote in a system. In a JSON implementation of function, we have some properties like key as function give Right to Vote type as function, header with function public access and parameter details and statements as array of function instructions.



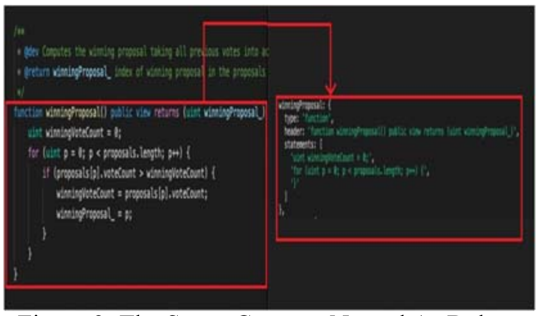
```
address public chairperson;

mapping(address => Voter) public voters;

Proposal[] public proposals;
```

```
globalDeclarations: [
  'address public chairperson;',
  'mapping(address => Voter) public voters;',
  'Proposal[] public proposals;'
],
```

Figure 8: Smart Contract Multiple Functions

3.4 The Smart Contract Named as Delegate

Another main function of Balloting in the smart contract named as delegate so we have to pass an address as a parameter and that user can be voted in a system. In a JSON implementation of function we have some properties like key as function delegate type as function, header with function public access and parameter details and statements as array of function instructions.



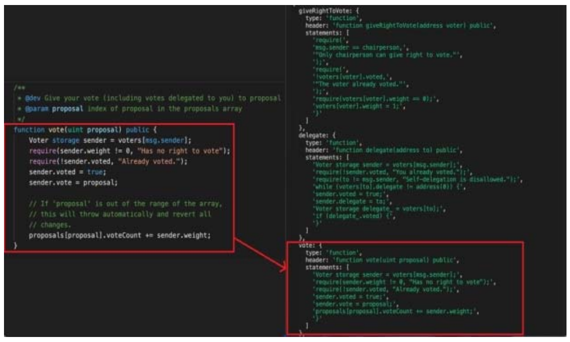
```
/**
 * @dev Computes the winning proposal taking all previous votes into ac
 * @return winningProposal_ index of winning proposal in the proposals
 */
function winningProposal() public view returns (uint winningProposal_)
    uint winningVoteCount = 0;
    for (uint p = 0; p < proposals.length; p++) {
        if (proposals[p].voteCount > winningVoteCount) {
            winningVoteCount = proposals[p].voteCount;
            winningProposal_ = p;
        }
    }
}
```

```
winningProposal: {
  type: 'function',
  header: 'function winningProposal() public view returns (uint winningProposal_)',
  statements: [
    'uint winningVoteCount = 0;',
    'for (uint p = 0; p < proposals.length; p++) {',
    '}'
  ]
},
```

Figure 9: The Smart Contract Named As Delegate

3.5 The Smart Contract Named as Vote Maintaining The Index of Voting

Another main function of Balloting in the smart contract named as vote Maintaining the index of voting. In a JSON implementation of function, we have some properties like key as function vote type as function, header with function public access and parameter details and statements as array of function instructions.



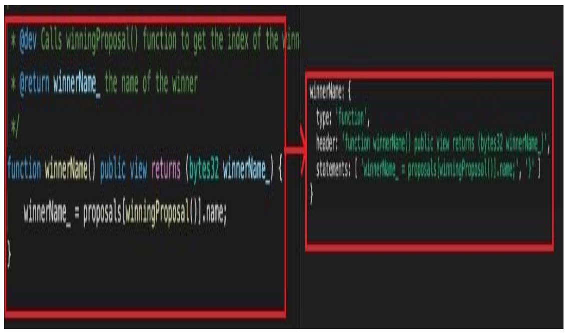
```
/**
 * @dev Give your vote (including votes delegated to you) to proposal
 * @param proposal index of proposal in the proposals array
 */
function vote(uint proposal) public {
    Voter storage sender = voters[msg.sender];
    require(sender.weight != 0, "Has no right to vote");
    require(!sender.voted, "Already voted.");
    sender.voted = true;
    sender.vote = proposal;

    // If 'proposal' is out of the range of the array,
    // this will throw automatically and revert all
    // changes.
    proposals[proposal].voteCount += sender.weight;
}
```

```
giveRightToVote: {
  type: 'function',
  header: 'function giveRightToVote(address voter) public',
  statements: [
    'require(',
    'msg.sender == chairperson,',
    '"Only chairperson can give right to vote."',
    ');',
    'require(',
    '!voters[voter].voted,',
    '"The voter already voted."',
    ');',
    'require(voters[voter].weight == 0);',
    'voters[voter].weight = 1;',
    '}'
  ]
},
delegate: {
  type: 'function',
  header: 'function delegate(address to) public',
  statements: [
    'Voter storage sender = voters[msg.sender];',
    'require(!sender.voted, "You already voted.");',
    'require(to != msg.sender, "Self-delegation is disallowed.");',
    'while (voters[to].delegate != address(0)) {',
    'sender.voted = true;',
    'sender.delegate = to;',
    'Voter storage delegate_ = voters[to];',
    'if (delegate_.voted) {',
    '}'
  ]
},
vote: {
  type: 'function',
  header: 'function vote(uint proposal) public',
  statements: [
    'Voter storage sender = voters[msg.sender];',
    'require(sender.weight != 0, "Has no right to vote");',
    'require(!sender.voted, "Already voted.");',
    'sender.voted = true;',
    'sender.vote = proposal;',
    'proposals[proposal].voteCount += sender.weight;',
    '}'
  ]
},
```

Figure 10: The Smart Contract Named as Vote Maintaining The Index of Voting

3.6 Winning Proposal Type Function

Winning Proposal calculating the highest voted structure and returning the value as a result. In a JSON implementation of function, we have some properties like key as function winning Proposal type as function, header with function public access and parameter details and statements as array of function instructions. WinnerName is a getter function which returns the winner name as in result so anyone can call this function without spending the gas fee to see the winner name. In a JSON implementation of function, we have some properties like key as function winnerName type as function, header with function public access and parameter details and statements as array of function instructions.



```
* @dev Calls winningProposal() function to get the index of the winn
* @return winnerName_ the name of the winner
*/
function winnerName() public view returns (bytes32 winnerName_) {
  winnerName_ = proposals[winningProposal()].name;
}
```

```
winnerName: {
  type: 'function',
  header: 'function winnerName() public view returns (bytes32 winnerName_)',
  statements: [ 'winnerName_ = proposals[winningProposal()].name;', '}' ]
}
```

Figure 11: function winning Proposal type as function

4. Conclusion

The emerging smart contracts have become a hot research topic in both academia and industry due to the rapid development of blockchain technologies. Smart contracts' immutability and irreversibility can help people exchange money, shares, intellectual property, and other assets in a transparent, conflict-free manner while avoiding third-party interference. As a result, smart contracts will become increasingly common in financial and social systems in the near future. We present an overview of smart contracts in this paper, including their concept, architecture, and application scenarios. In this research also discuss the smart contract's challenges and present its future trends. In the future, we intend to conduct additional research on parallel blockchain and related smart contract applications.

This proposed system is intended to provide secure data and a trustworthy election among democratic citizens. Blockchain will be publicly verifiable and distributed in such a way that it cannot be corrupted. In this research work, a security mechanism based on blockchain technology is designed to ensure the integrity of vote data in an electronic voting system. The original blockchain architecture proposed by Satoshi Nakamoto has been modified to meet our needs for an electronic voting system. A prototype implementation of the same is also being developed.

4.1 Future Recommendation/Work

- ➢ It is planned to develop an encryption methodology, preferably using hybrid techniques, in the future to improve the security of our model and its resistance to snooping attacks.
- ➢ Security measures will also be developed to allow our model to withstand Denial of Service (DoS) attacks that could affect different layers of the network stack.
- ➢ It is also planned to upgrade the system to handle a large number of connected devices and to improve the model's concurrency.
- ➢ The model will be optimised to reduce the time required to register votes and verify transactions.
- ➢ To supplement the functionality of the fingerprint authentication system, alternative biometric authentication mechanisms such as iris scanning, face and voice recognition can be used.
- ➢

References

[1] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaria, "To blockchain or not to blockchain: That is the question," IT Professional, vol. 20, no. 2, pp. 62–74, 2018.

[2] S. Leible, S. Schlager, M. Schubotz, and B. Gipp, "A review on blockchain technology and blockchain projects fostering open science," Frontiers in Blockchain, vol. 2, Nov. 2019. [Online].

[3] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. March- esi, “A massive analysis of ethereum smart contracts empirical study and code metrics,” IEEE Access, vol. 7, pp. 78 194–78 213, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2921936

[4] D. Macrinici, C. Cartofeanu, and S. Gao, “Smart contract applications within blockchain technology: A systematic mapping study,” Telematics and Informatics, vol. 35, no. 8, pp. 2337–2354, Dec. 2018. [Online].

[5] L. W. Cong and Z. He, “Blockchain disruption and smart contracts,” Tech. Rep., Mar. 2018. [Online].

[6] B. Bodó, D. Gervais, and J. P. Quintais, “Blockchain and smart contracts: the missing link in copyright licensing?” International Journal of Law and Information Technology, vol. 26, no. 4, pp. 311–336, 2018. [Online]. M. Alharby and A. van Moorsel, “Blockchain based smart contracts : A systematic mapping study,” in Computer Science & Information Technology (CS & IT). Academy & Industry Research Collaboration Center (AIRCC), Aug. 2017. [Online].

[7] C. Sillaber, B. Waltl, H. Treiblmaier, U. Gallersdörfer, and M. Felderer, “Laying the foundation for smart contract development: an integrated engineering process model,” Information Systems and e- Business Management, Feb. 2020. [Online].

[8] V. Buterin, “A next-generation smart contract and de- centralized application platform,” 2015.

[9] B. K. Mohanta, S. S. Panda, and D. Jena, “An overview of smart contract and use cases in blockchain technology,” in 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE, Jul. 2018. [Online]. Available: https://doi.org/10.1109/ icccnt.2018.8494045

[10] “Smart contracts implementation, applications, benefits, and limitations,” Journal of Information Engineering and Applications, Sep. 2019. [Online]. Available: https://doi.org/10.7176/jiea/9-5-07

[11] L. M. Bach, B. Mihaljevic, and M. Zagar, “Comparative analysis of blockchain consensus algorithms,” in 2018 41st International Convention on Information and Com- munication Technology, Electronics and Microelectron- ics (MIPRO). IEEE, 2018, pp. 1545–1550.

[12] “Nem ecosystem blockchain - because together, ev- erything is possible.” https://nem.io/, (Accessed on 07/02/2021).

[13] E. Elrom, “Neo blockchain and smart contracts,” in The Blockchain Developer. Springer, 2019, pp. 257–298.

[14] “Neo smart economy,” https://neo.org/, (Accessed on 07/02/2021).

[15] C. Dannen, Introducing Ethereum and solidity. Springer, 2017, vol. 318.

[16] G. Wood et al., “Ethereum: A secure decentralised gen- eralised transaction ledger,” Ethereum project yellow paper, vol. 151, no. 2014, pp. 1–32, 2014.

[17] A. M. Antonopoulos and G. Wood, Mastering ethereum: building smart contracts and dapps. O’reilly Media, 2018.

[18] “Home ethereum.org,” https://ethereum.org/en/, (Ac- cessed on 07/02/2021).

[19] V.-O. Ossip, “Ethereum blockchain and hyperledger burrow blockchain comparative analysis.”

[20] “Hyperledger burrow – hyperledger,” https://www.hyperledger.org/use/hyperledger-burrow, (Accessed on 07/02/2021).

[21] B. Xu, D. Luthra, Z. Cole, and N. Blakely, “Eos: An architectural, performance, and economic analysis,” Retrieved June, vol. 11, p. 2019, 2018.

[22] M. T. Quasim, M. A. Khan, F. Algarni, A. Alharthy, and G. M. M. Alshmrani, “Blockchain frameworks,” in Decentralised Internet of Things. Springer, 2020, pp. 75–89.

[23] “Home – eosio blockchain software & services,” https://eos.io/, (Accessed on 07/02/2021).S. D. Lerner, “Rsk,” 2015.

ZAIN UL ABEDIN
Technical Team Lead
Nugenesis Pty, Ltd Australia
zainmustafaaa@hotmail.com

Muhammad Shujat Ali
IT Manager, Orange Networks,
Lahore
shujat@orangnetworks.com

Ashraf Ali
Assistant Professor & In-Charge M.Phil (CS)
The Institute of Management Sciences,
ashraf@pakaims.edu.pk

Sana Ejaz
M.Phil (CS) – FAST National University of
Computer and Emerging Sciences
Lahore
sani.ejaz.a@gmail.com