BLOCKCHAIN BASED DISTRIBUTED KEY PROVISIONING SYSTEM

by

Spandana Etikala

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2019

Approved by:

____

Dr. Fareena Saqib

____

Dr. Jim Conrad

____

Dr. Ronald Sass

ABSTRACT

SPANDANA ETIKALA. Blockchain Based Distributed Key Provisioning System. (Under the direction of DR. FAREENA SAQIB)

In recent days, smart electronics have evolved into ubiquitous computing with connected interfaces such as self-driving automobiles with evolution in sensor network technology and artificial intelligence. Due to these advancements, new features are added into control area network in automobiles but the security and trust in these systems are still questionable because the improvement in automation is paving way for many potential cyber attacks. The closed systems such as in intra-vehicle configuration, the control and command communication between electronic control units require custom architectures to support real time responses that integrate security and safety protocols. The centralized PKI depends on trusted third parties or a centralized server as certification authority to share the public keys and generate digital certificates for asymmetric keys to provide authentication. The centralize CA can be a single point failure in the closed system. To address this problem, we propose a distributed key provisioning system using blockchain technology. We demonstrate the key provisioning scheme using Ethereum blockchain and smart contract for key storage and validation of the public key pair. The key storage and validation process involves all the nodes in the closed system and authenticates by traversing through their copy of blockchain and vote accordingly.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Fareena Saqib for the continuous motivation and distribution of her immense knowledge towards my master's study and thesis research. Her guidance has assisted me throughout my studies, research and writing this dissertation. I wouldn't have imagined my master's thesis research being better without her support.

In addition to my advisor, I would like to thank rest of my thesis committee: Dr. Jim Conrad and Dr. Ronald Sass for their encouragement, valuable inputs and comments. Finally, I would like to thank and respect my family and our team for uplifting me and standing by me through good and bad times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

AES	Advanced Encryption Standard
CA	Certificate Authorities
CAN	Control Area Network
CAN FD	CAN With Flexible Data Rate
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptography
ECDSA	Eliptic Curve Digital Signature Algorithm
EOA	Externally Owned Account
ECU	Electronic Control Units
PKI	Public Key Infrastructure
ECC	Elliptic Curve Cryptography
PCR	Platform Configuration Regidters
PUF	Physical Unclonable Functions
RSA	Rivest-Shamir-Adleman
TPM	Trusted Platform Module

CHAPTER 1: INTRODUCTION

In recent days, smart electronics have evolved into ubiquitous computing with connected interfaces such as selfdriving automobiles with evolution in sensor network technology and artificial intelligence. Due to these advancements, new features are added into control area network in automobiles but the security and trust in these systems are still questionable because the improvement in automation is paving way for many potential cyber attacks. Security integration requires policies and security mechanism implementation custom to the system interfaces, resources and performance requirements. A system implements various security policies for access control, accountability and privacy of data and other system resources. The pillars of a security model are confidentiality, integrity and availability. Cryptographic algorithms are used for encryption to preserve data confidentiality. Keys play a major role in cryptography for achieving secrecy, where the whole system can be compromised with the leakage of keys. Keys must be safely protected, distributed and authenticated to ensure security. Certificate authorities (CA) are used to maintain these keys distribution and key update processes. CA issue digital certificates to store and authenticate the identity of the user. Certificate authorities could be a trusted third party or a centralized server that are the single point of failure if the server or the third party is compromised. In this work, we propose a secure distributed key provisioning method using blockchain technology and smart contracts to store and validate the keys. Instead of giving authority to single user for validation we are using all the nodes in the network to participate in validation.

Cryptography plays a crucial role in security systems."Crypto" refers to secret and "graph" refers to writing. It is a study of mathematical methods used to protect the information to ensure confidentiality and data integrity. There are three main branches in cryptography [3]. They are a) Symmetric cryptography b)Asymmetric cryptography (public key cryptography) c) Cryptographic Protocols. Cryptographic protocols address the cryptographic algorithms related applications. Symmetric and asymmetric algorithms are the basis for secure systems. They use keys to encrypt the messages while sending and decrypted only on the receiver side so that a malicious node sitting in between these two nodes cannot understand the message. The keys must be protected from the hacker and shouldn't be weak and predictable.

In symmetric cryptography, two parties share same secret key known as private key for encryption and decryption. Advanced Encryption Standard (AES), Data encryption standard (DES), etc., are the examples of symmetric encryption. The security relies on the secrecy of the shared key [4]. The problem in these systems is the distribution of keys. The traditional communication protocol cannot guarantee the secure provision of these keys. The secure way to distribute these keys is to program it at the time of manufacturing or installation.

In asymmetric cryptography (public key cryptography) [5] two different keys known as public and private keys are used in encryption and decryption. The public key is derived from the private key, but the reverse process is not possible. In this scheme, the message is encoded with the sender's private key and the receiver's public key using a cryptographic algorithm and sent to receiver. Only the receiver with the private key can decode the encoded message. Thus, eliminating the need to transmit the private key. The public key is shared among everyone to encrypt the data, so that the only user with the private key can decrypt the data. The private key is kept in secret. Some of the examples of asymmetric cryptography are: Rivest-Shamir-Adleman (RSA), Elliptical Curve Cryptography (ECC), etc. RSA is the most widely used scheme but requires huge resources and memory, whereas ECC is best suitable for lightweight embedded nodes. In this work, the Ethereum addresses are generated using Elliptic curve digital signature algorithm (ECDSA).

In public key cryptography, the public key provided by the user must be authenticated to ensure it belongs to the user and not a malicious node which tampered the system and replaced the key. Public Key Infrastructures (PKI) are used for user identification. PKIs [6] are the set of policies and procedures used to generate, authenticate, store and distribute public keys through trusted third parties and issue digital certificates. These certificates are used to identify the user public key. It contains information about the public key, the owner of public key and the digital signature of verifier. Different components of PKI are a) Certificate authorities b) Registration Authorities c) Certificate Repositories. Certificate authorities are trusted third parties used for validating the public key. Registration authorities are used to accept the request for the issuance of digital certificates. Certificate repositories are used to store these keys safely.

We propose a decentralized key provisioning scheme using private Ethereum blockchain with a smart contract for secure key-provisioning, authentication of nodes for systems such as intracommunication between electronic control units (ECUs) of an automotive. The scheme provides a secure key exchange and authentication process using private Ethereum Blockchain and Smart contract for the network with limited number of nodes such as CAN FD. The scheme is implemented on 4 nodes, where 2 nodes are miner nodes.

1.1 Key Contributions

1. Secure key storage in the blockchain.

2. Key provisioning and validation by traversing through the blockchain.

3. Demonstrate the scheme to prevent the attacker from malicious key updates and voting mechanism for key validation.

4. Experimental test bed using Raspberry Pis, Ethereum blockchain and smart

contracts to evaluate the security framework.

1.2 Organization Of Report

The report is organized in the following chapters. Chapter 2 discusses the background of cryptography and public key infrastructure components, threat model and limitations of existing key provisioning schemes and newer technologies that can improve the security and resilience of the infrastructures. Chapter 3 discusses the proposed scheme of Distributed Key Provisioning. Chapter 4 includes details on hardware setup, implementation, results and security analysis of the proposed scheme. Chapter 5 includes conclusion and future work.

CHAPTER 2: BACKGROUND

Key provisioning is the key generation, storage and renewal of cryptographic keys for secured governance [7] as shown in Figure 2.1. Cryptographic systems use one or more keys for encryption and decryption. For example, symmetric cryptography uses a single shared key for both encryption and decryption processes, where as in asymmetric cryptogrpahy public and private key pair is used for encryption and decryption respectively. The security of cryptography schemes relies on the secrecy of keys, hence the keys require a secure key provisioning methodology for new keys enrollment. Cryptographic implmentations are vulnerable to poor key provisioning.

Key management must provide two aspects a) Secrecy of keys b) Guarantee that key is assigned to a specific node, such as digital certificates. Assurance of public keys in public key cryptography is provided by authentication from Public Key Infrastructures. Key provisioning may generate new keys for every session to facilitate forward secrecy that is, the compromise of one message should not affect others.

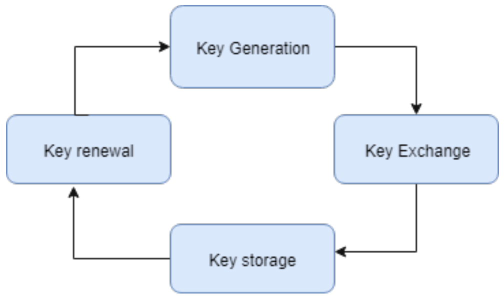


Figure 2.1: Key Provisioning

Security aspects in key provisioning include authentication of legitimate nodes, access control and authorization. Authentication is a mechanism to guarantee the identity of the communicating nodes and mutually agreeing to send or receive the information. Authentication process can be described in two different stages. a) Identification, that is security systems are provided with user's identity in the form of an ID. Then security systems will validate this ID. b) Mutual authentication, that is the user must provide evidence that the ID actually belongs to him. For example, Public Key infrastructures (PKI) provide digital certificates as the user ID and associated key to authenticate the user. These are issued by certificate authorities (CA). Different types of Authentication schemes are:

2.1 PUF Based Authentication

Two identical ICs have intrinsic variations due to imperfections in manufacturing processes. Physical Unclonable Functions (PUF) utilize the inter and intra chip variations to generate unique Challenge-Response Pairs (CRP) which can be used for authentication[8]. [9] describe the authentication based on hardwarebased embedded delay PUF for privacy and mutual authentication by storing path timing information at the verifier.

PUF based authentication contains enrollment phase and verification phase [10]. In the enrollment phase the keys are generated and stored. In verification phase, the keys are validated. The reliability of PUF based authentication depends on reporodducability of CRPs. If there a bit flip due to environmental changes then CRPs cannot be reproduced. PUF based authentication still have to depend on third parties for validation of keys.

2.2 TPM Based Authentication

Trusted Platform Module (TPM) is a specialized on chip core used to generate and store the keys on a tamperresistant nonvolatile memory. [11] TPM supports platform hierarchy, storage hierarchy and endrosment hierarchy. TPM features include true random number generator,symmetric and asymmetric encryption, hmax signatures, digital attestation and verification. Individual keys and data in hierarchies have their own authorization and policy values.

[12] describes TPM for secure key generation and storage over CAN Protocol. TPM supports secure boot and measured boot using PCR registers to compute cumulative hash of the boot file such as device tree and boot image [13].

Hardware security modules are expensive resources for security integration on the closed network or system such as automotive .

2.3 Certificate Authorities And Its Vulnerabilities

Certificate authorities (CA) are responsible for issuing, publishing, verification and revocation of digital certificates [6]. CA implements process for the identification of client and guarantees that the information present in certificate is valid and signs it digitally. If an electronic node needs to verify the public key belongs to a client node, it sends a request to certificate authority using CA's public key. The CA responds to the requesting node with the digital certificate signed its own private key. The requesting node verifies the response and initiate communication with the client node.

The certificate authorities are hierarchical and centralized trusted third parties which can be single point of failure [14]. [15] the attack on ”DigiNotar” a certificate authority in 2011 resulted in a man in the middle attack on Google services [16]. [17] describes compromises of certificate authorities such as Registration Authority compromise, Root CA compromise for issuance of fraudulent certificates. There are security vulnerabilities in hierarchical and centralized model.The centralized certificate authority server is a single point of attack and performance bottleneck for the systems with realtime requirements. Certificate revocation or expiry for key renewals for each node [18]. An alternative approach would be the usage of decentralized schemes for authentication. In this thesis, we investigate blockchain technology for decentralized Key provisioning scheme.

2.4 Distributed/Decentralized Key Management Approaches

Decentralized schemes improve the system security by removing centralized authority of the certificate authority. Some of the decentralized approaches are: a) Pretty Good privacy (PGP) [19]: In this approach user relies on its peers for validation but the main issue with this scheme is key distribution and addition of new nodes into network. b) Certificate transparency [20] is one of the decentralized trust model proposed by Google. In these scheme the certificates are stored and monitored by many parallel servers across the globe.Though they are distributed, they are governed by a single organization. The usage of blockchain technology in distributed key provisioning would overcome the vulnerabilities of PGP and Certificate Transparency.

Some of the blockchain based key provisioning systems are Ceocoin, Certocoin, Keychain and Authcoin. [21] Proposed "Ceocoin" where certificates are stored into blockchain using distributed certificate scheme. [22] Fromknecht et al., introduces distributed PKI using blockchain technology called as "Certocoin" on top of Namecoin blockchain but it is not suitable for lightweight applications. [23] describes blockchain based PKI for lightweight applications using both Emercoin Name Value Services and smart contracts by Ethereum then compared both of them with centralized certificate authorities. [24] proposed blockchain based key distribution system known as "Keychain", where they used ProofOfStake consensus algorithm instead of ProofOfwork algorithm to reduce the usage of resources. They implemented voting scheme to select the users who can write keys into blockchain. [25] introduces blockchain based PKI known as "Authcoin" for validation and authentication. All these schemes have used blockchain for key provisioning and storage of keys.They are used for authentication. Our proposed decentralized Key provisioning scheme is also based on blockchain but compared to other works we are targeting the application of this scheme for closed systems such as intra-vehicle communication such as CANFD. We are using a private blockchain to restrict accessibility by an unauthorized user. Validation of keys involved all the nodes instead of only miners in the network by traversing through blockchain and voting smart contract to improve the security for closed systems.

2.5 An Overview Of Technologies Used

2.5.1 Blockchain Technology

Blockchain is a decentralized and distributed database.[2]. It is a P2P network where all the transactions between computers (nodes) are stored in the form of blocks and linked together with cryptographic hashing. All the hashes of the blocks are encoded into a Merkel Tree as shown in the 2.2. Each node in the chain can have a copy of the whole chain allowing the transparency. It was first introduced by Satoshi Nakamoto in 2008 as a public distributed ledger for Bitcoin cryptocurrency[1]. Later, due to its features, it was extended to many other applications such as Public Key Infrastructure, IOT, etc. Usage of blockhain for key provision would prevent single point failure and make it much more straightforward[21]. Blockchain is immutable, that is any data written into the chain cannot be modified because if one block is modified then the whole chain must be changed as they are linked with each other through the previous hash as shown in Figure 2.2

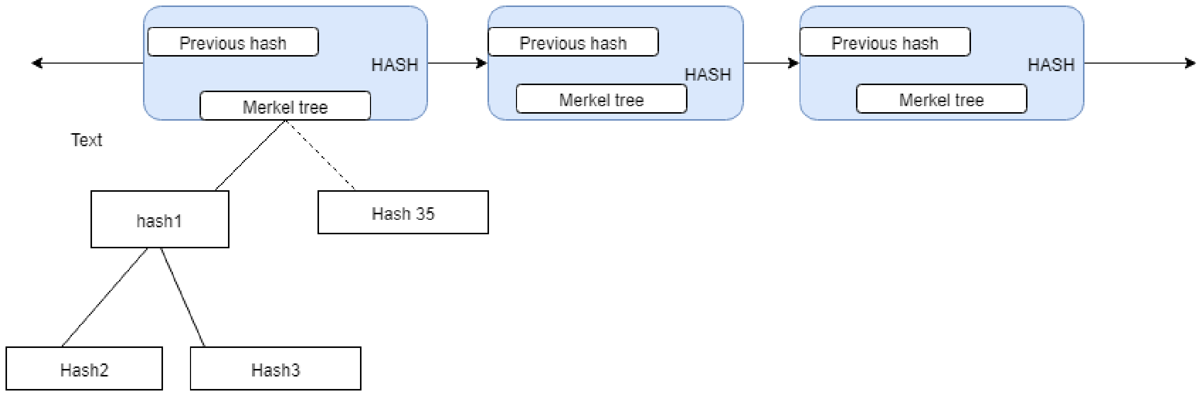


Figure 2.2: Blockchain [1]

There are two types of blockchains a) Public blockchain c) Private blockchain. The major difference between these two types is the accessibility. In public block chain, anyone one with enough resources and internet can join the chain whereas in private chain only a few trusted parties can participate in the network.

Blockchain provides three features[26]. They are:

1. Decentralization: Unlike centralized systems the information is not stored in a single system. In blockchain all the information is shared among the nodes in the network. There is no need for a third party to validate the transactions as shown in 2.3.

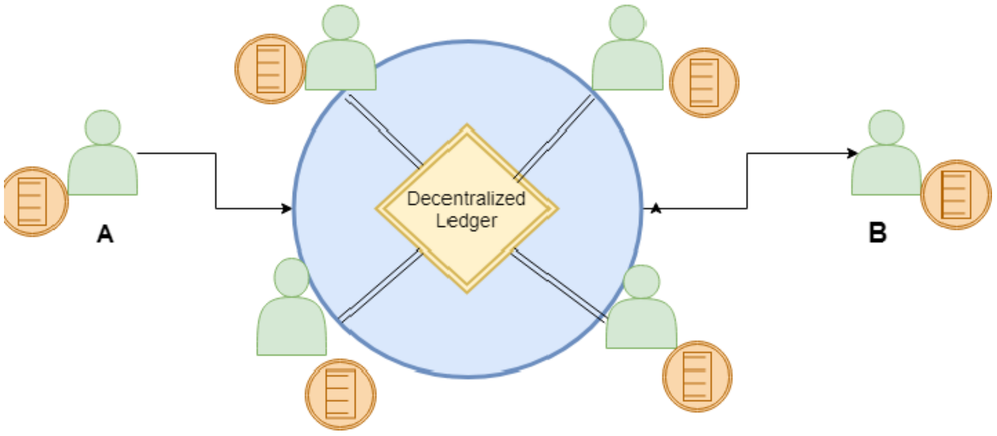


Figure 2.3: Decentralized Mechanism

2. Immutability: Blockchain is immutable, that is any data written into the chain cannot be modified easily because each block is connected with each other using previous hash and each node can have their own copy of the chain. If we try to tamper with one block,the hash value changes resulting in a different value of previous hash that is linked with the next block.

3. Transparency: As blockchain is a record keeping ledger. The whole transactions made by an entity can be seen if we have the public address of that entity.

The crucial part of blockchain is the mining process and it depends upon the hashing algorithms. For example, bitcoin uses SHA256 algorithm. For a given same input to the hashing algorithm results the same output every time. The input can be anything such as integers, strings, time, etc and the output resulted is called “hash” and would always be same length. Miners solve a cryptogrpahic puzzles such as “Proof-Of-Work” to verify the transaction add blocks into the chain. During the resolving of cryptogrpahic puzzle the miner determines a number known as nonce which generates a cryptogrpahic hash less than a threshold value known as difficulty [27]. This is known as consensus. The consensus in the blockchain is used to maintain synchronization of blocks and ensure the block added into the blockchain is true.

Working Of Blockchain:

Following simple example demonstrates how blockchain works using a Bitcoin transaction [28]. Consider two persons, Alice and Bob. Alice wants to send 2 bitcoins to Bob. Then Alice sends the broadcasts message along with the transaction to the network with Bob's public key, amount and signed transaction with Alice's private key.

The miners sitting in the network listens to the message and then validate Alice using its signature to check if the bitcoin sent belongs to Alice or not. After the validation, the miner keep this transaction in a block along with other transactions, previous block output hash and add try to add it to the chain of blocks by solving a cryptographic puzzle known as Proof-Of-work (POW) to generate a valid output hash. Once the block is mined the miner broadcast the newly mined block to the full nodes to update their copy of the chain. The entire process flow of blockchain in shown in Figure 2.4.

Every miner in the block chain contains full copy of the blockchain and everyone in the network trusts the miner with longest chain. In this project, we are using the Ethereum blockchain with a smart contract.

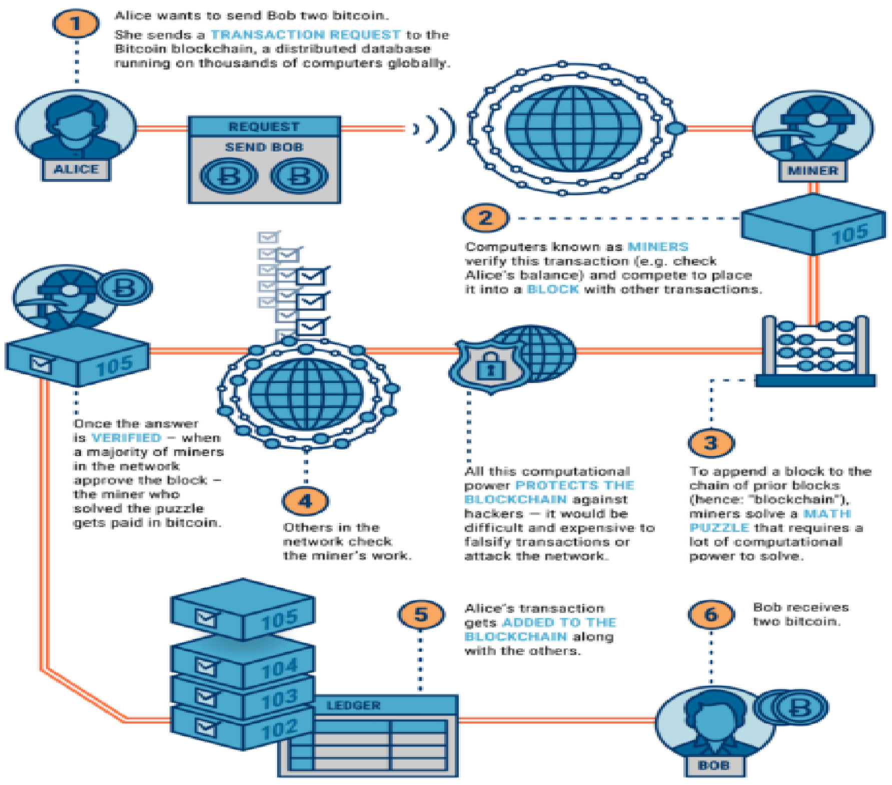

1
Alice wants to send Bob two bitcoin.
She sends a TRANSACTION REQUEST to the Bitcoin blockchain, a distributed database running on thousands of computers globally.
ALICE
REQUEST
SEND BOB
MINER
2
Computers known as MINERS verify this transaction (e.g. check Alice's balance) and compete to place it into a BLOCK with other transactions.
105
Once the answer is VERIFIED – when a majority of miners in the network approve the block – the miner who solved the puzzle gets paid in bitcoin.
4
Others in the network check the miner's work.
All this computational power PROTECTS THE BLOCKCHAIN against hackers – it would be difficult and expensive to falsify transactions or attack the network.
3
To append a block to the chain of prior blocks (hence: "blockchain"), miners solve a MATH PUZZLE that requires a lot of computational power to solve.
5
Alice's transaction gets ADDED TO THE BLOCKCHAIN along with the others.
6
Bob receives two bitcoin.
104
103
102
LEDGER
BOB

Figure 2.4: Blockchain Mechanism [2]

2.5.2 Ethereum and Smart Contract

Ethereum is an open source public blockchain that features smart contracts. It was first introduced by Vitalik Buterin [29]. Compared to Bitcoin Ethereum is faster and uses different ProofOfWork algorithm known as Ethash. It is an improved version of Bitcoin. Ethereum is a "Turing complete machine", unlike bitcoin which supports only electronic cash system application, we can program any decentralized application on Ethereum. It also provides the state of transactions. In Ethereum, we require either "Ether" or "gas" for sending transactions to other nodes and contract. It provides all the features of a blockchain with extra features such as DApps also known as smart contracts that run on top of Ethereum and stored within the blockchain.

Ethereum facilitates two types of accounts. They are: a) Externally owned account (EOA) b) Contract accounts. EOA has ether balance and controlled by public and private keys. The last 20 bytes of the public key are used as an EOA address to send and receive the transactions. They don't have any code associated with them.Contract address also has ether balance, but it is controlled by the code associated with them and can make changes to its memory and state. Contract address can only be activated by an EOA. Each Ethereum full node contains Ethereum virtual machine (EVM) used to execute the smart contracts. As described in [29] each account has four fields: a) nonce to describe number of transactions from the account. b) Current balance in ether. c) Contract code of account if present. d)storage of the account.

Geth and Parity are the two client software for Ethereum. Geth is the official software produced by Ethereum. They are used to interact with the blockchian to add the nodes into the chain, implementing the transactions, mining and to copy the blockchain data. Geth provides remote procedure call APIs [30]. Web3.js is the library used to provide functions to interact with local or remote Ethereum node by using HTTP or IPC.

A smart contact is a special protocol supposed to digitally verify and facilitate the transaction. It is a short reusable code written in a solidity programming language for Ethereum blockchain. Smart contracts help us to avoid the services of the third party. They not only define rules and regulations but also implement them automatically. Smart contracts provide trust as all the documents are encrypted and stored in the shared ledger. Each smart contract is given a contract address when deployed. Users use that address to communicate with thesmart contract. The Figure 2.5 shows various components of Ethereum.

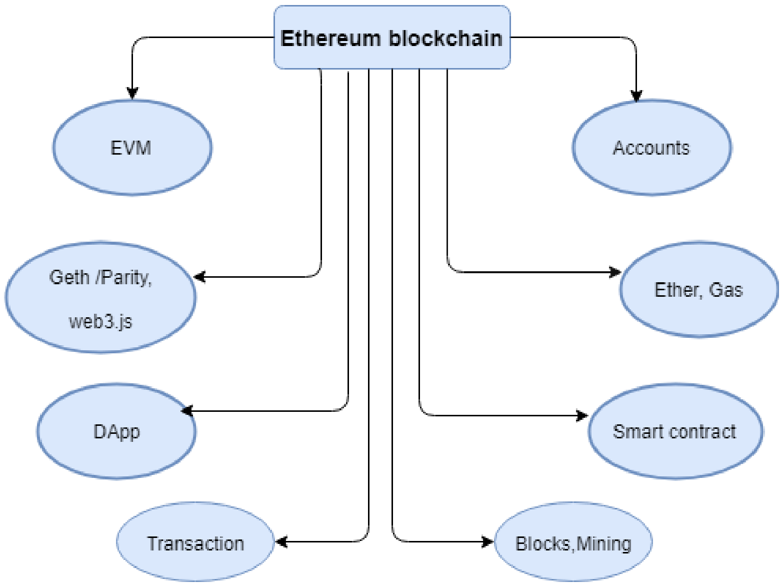


Figure 2.5: Various Components of Ethereum

Transaction Mechanism in Ethereum:

Ethereum transactions contains [31]:

a)from : Account address from which transactions are sent

b) to : Account address to which transaction is sent

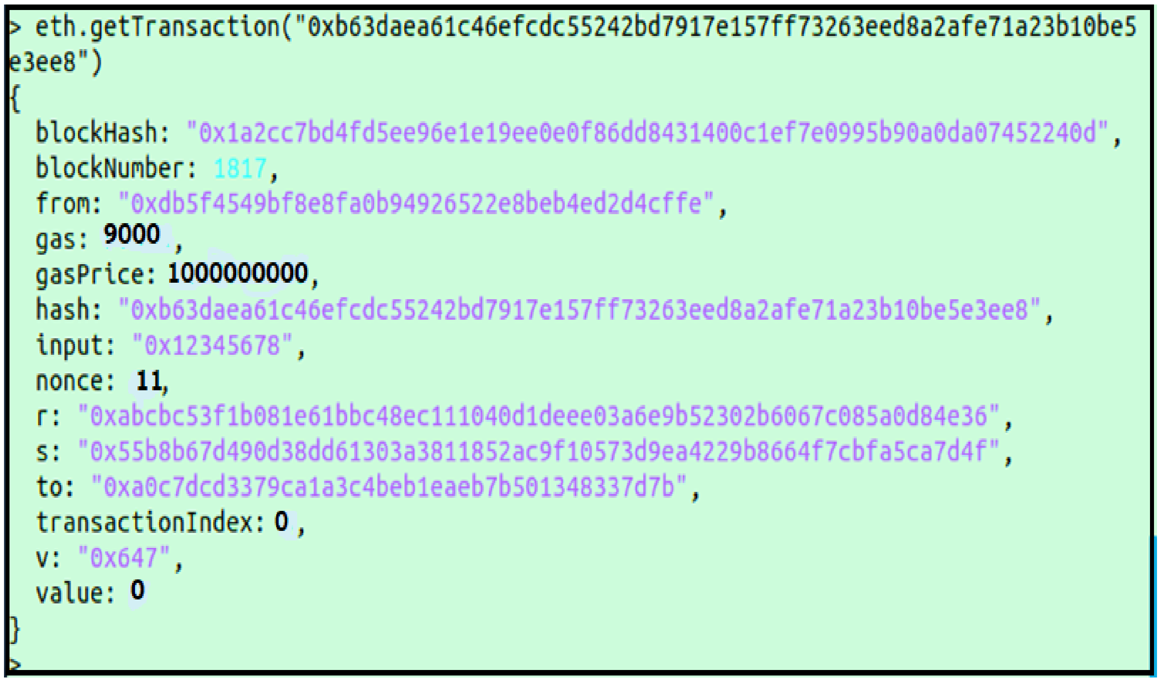
c) value : Amount of ether to be sent

d)input/data : It is mainly used for contracts but can also be used to send some data in hex format to other nodes.

e)gas limit : It gives the information about the amount of gas required to execute a particular transaction.

All the transaction in a blockchain are signed with the keys of sender and receiver for validation. For signing the transactions in Ethereum, there are two types of functions provided by Geth client software. They are: 1) eth.sendTransaction() and 2) eth.sendRawtransaction() [32]. eth.sendTransaction() automatically signs the transaction and performs the serialization required before sending into network. Whereas, eth.sendRawTransaction() is used to send signed transaction if the keys are not handled locally by Geth.

Figure 2.6 shows an example of a transaction receipt in Ethereum using eth.getTransaction():



```
> eth.getTransaction("0xb63daea61c46efcdc55242bd7917e157ff73263eed8a2afe71a23b10be5
e3ee8")
{
  blockHash: "0x1a2cc7bd4fd5ee96e1e19ee0e0f86dd8431400c1ef7e0995b90a0da07452240d",
  blockNumber: 1817,
  from: "0xdb5f4549bf8e8fa0b94926522e8beb4ed2d4cffe",
  gas: 9000 ,
  gasPrice: 1000000000,
  hash: "0xb63daea61c46efcdc55242bd7917e157ff73263eed8a2afe71a23b10be5e3ee8",
  input: "0x12345678",
  nonce: 11,
  r: "0xabcbc53f1b081e61bbc48ec111040d1deee03a6e9b52302b6067c085a0d84e36",
  s: "0x55b8b67d490d38dd61303a3811852ac9f10573d9ea4229b8664f7cbfa5ca7d4f",
  to: "0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b",
  transactionIndex: 0 ,
  v: "0x647",
  value: 0
}
>
```

Figure 2.6: Ethereum Transaction Receipt

Ethereum transactions mainly serve three purposes: a) To Transfer funds, b) deploying of contract using bytecode, c) Interacting with smart contract.

CHAPTER 3: BLOCKCHAIN BASED DISTRIBUTED KEY PROVISIONING

Distributed key provisioning scheme is composed of secure key storage, key update and secure key communication to other legitimate nodes. This scheme of decentralized key management provides much more transparency, security and addresses vulnerabilities of the centralized certification authority in the process of allocation and management of keys. The scheme integrates blockchain based decentralized public key infrastructure for

1. Secure storage of Public keys.
2. Authentication of Public keys.
3. Preventing spoofing and masquerading attacks using private Ethereum blockchain and Externally Owned Address (EOA) of Ethereum.

The proposed distributed key provisioning implements following steps:

1. Enrollment process: The keys generated by each node are added to the blockchain as a new block. The blocks are immutable and each new block is added at the end of the blockchain. The block number is updated and shared with all the peers in the blockchain which are initially set by using static-nodes file. During secure processing and communication, the peer nodes require to validate the public keys from the blockchain to encrypt the payload before transmission.
2. Validation: we propose a voting mechanism using smart contracts to confirm that the validation responses have not been tampered or spoofed.

   The scheme works in the following way:

(a) A request for validation is made using the smart contract that implements voting scheme and integrates it with the blockchain. The requested node key is referenced using the EOA address of the other node.

(b) The other nodes in the network are notified by the smart contract event which is triggered when a proposal request is called for the validation.

(c) Other nodes in the network respond to the request by first traversing through the copy of their blockchain to check whether the public key sent by the node for the verification is present in the chain and then validate it by voting. Then an event is triggered to notify the completion of validation. Before traversing through the blockchain the EOA address of nodes which are trying to vote are checked if they are valid or not.

(d) At the completion of the validation process, the requesting node receives the votes and makes the decision of using the public key for secure communication or blacklisting in case of key compromise.

The malicious node cannot change the votes directly because the voting scheme is in the form smart contract and it is stored in the blockchain which is immutable. Voting smart contract also takes care if malicious node is trying to vote in place of other nodes by voting again in order to change the votes.

Our scheme focuses on key provisioning for a closed network such as CAN FD. The blockchain network with the fewer number of nodes will have less miners compared to other blockchain networks. If the whole validation depends only on the few miners may result in failure if the miner is compromised. To overcome this problem We have involved all the nodes instead of only using miners during the validation. The entire scheme is represented in Figure 3.1:

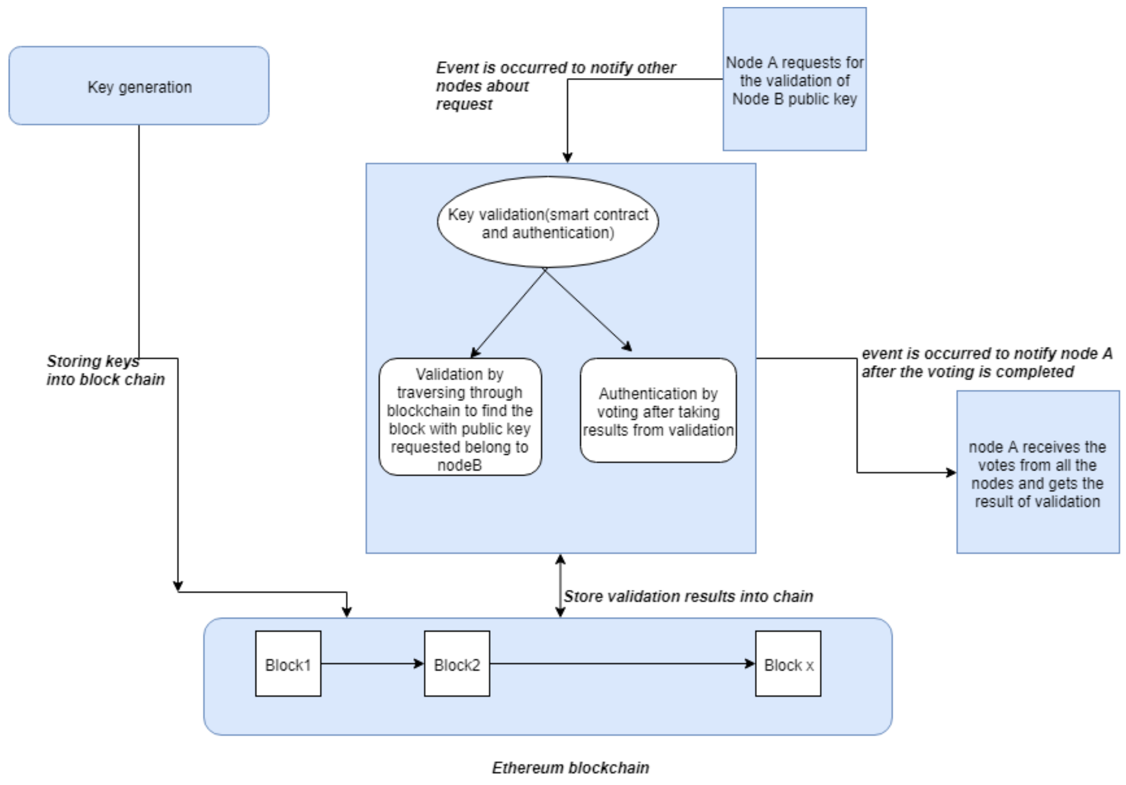

Key generation
Event is occurred to notify other nodes about request
Node A requests for the validation of Node B public key
Key validation(smart contract and authentication)
Storing keys into block chain
Validation by traversing through blockchain to find the block with public key requested belong to nodeB
Authentication by voting after taking results from validation
event is occurred to notify node A after the voting is completed
node A receives the votes from all the nodes and gets the result of validation
Store validation results into chain
Block1
Block2
Block x
Ethereum blockchain

Figure 3.1: Blockchain Based Distributed Key provisioning

We developed two algorithms for voting smart contract and authentication respectively.

Algorithm 1 describes the smart contract that is used for sending requests for validation, key validation and storing of results. The “proposal”is a structure datatype which gives the information about the requests and results of the authentication. Proposal() function is used for sending the request for validation. Voting_Start() function call first checks whether the proposal is completed or not, then checks if the voter has already voted and then caste the votes for validation. The complete code is given in Appendix A.

Algorithm 1 Voting Smart contract

```
1: struct proposal:
   prop_id  //proposal id for keeping track of proposal
   completed  //to check if the proposal has competed or not
   toverify  // Public key to be verified
   recv  // EOA address of node to be verifies
   send  // EOA address of the node which send the proposal
   totalnovotes  // total number of votes casted
   insupport  //incerements if vote is valid
   notsupport  // increments if vote is not valid
2: function PROPOSAL(public key of B, EOA of node B, number of voters)
3:     update the proposal struct
4:     trigger the request voting event
5: end function
6: function VOTING_START(proposal number, public key of nodeB from
   blockchain, EOA of nodeB, Number of nodes in the network)
7:     if proposal is completed or voter has already voted then
8:         revert()
9:     else
10:        if public key sent through proposal == public key derived from blockchain
   then
11:            insupport++
12:        else
13:            notsupport++
14:        end if;
15:        Total++;
16:    end if
17:    if total == number of nodes in the network then
18:        proposal completed = true;
19:        trigger voting_completed event;
20:    end if
21: end function
```

Algorithm 2 describes the proposed authentication scheme. The authentication application is used to traverse through the blockchain, call the smart contract methods using contract instances and respond to events. Gettransactions() function is used to traverse through the blockchain and check if the public key is present or not. Contract_Instance() function is used to call the instance of the smart contract. To request the validation, nodes call Req_Proposal() function, which in turn calls a smart contract function called Proposal() as shown in algorithm 1. A requesting voting event is triggered to notify the other nodes in the network about the proposal request for validation.

Votingstart() function is called by the remaining nodes in the network to start the validation. This function is used to: a) read the the values to be verified from the request event occurred. b) Then validate the EOA addresses of node which is trying to vote.c) Checks number of peers and then calls GetTransactions() function to find the public key associated with the node by traversing through the blockchain. d) Then calls the Voting_Start() smart contract function to vote accordingly. The Get_User() function is used to read the values of proposal structure. The process can be depicted in the Figure 3.2. The complete code is given in Appendix B.

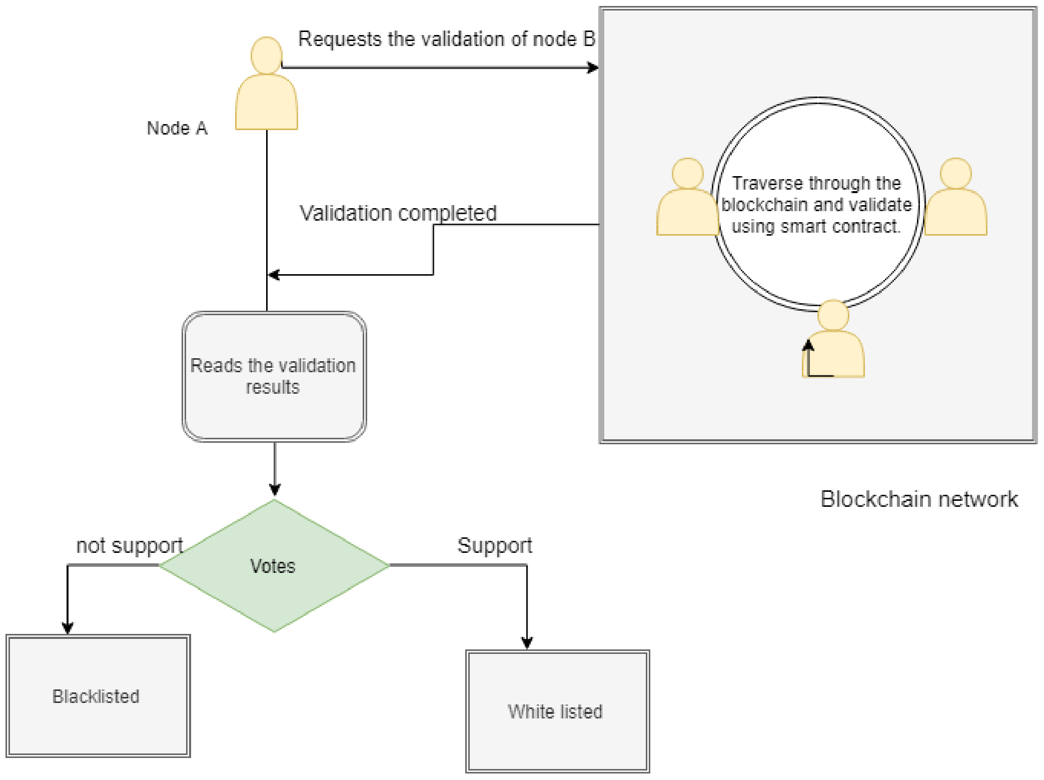

Requests the validation of node B
Node A
Validation completed
Traverse through the blockchain and validate using smart contract.
Reads the validation results
Blockchain network
not support
Votes
Support
Blacklisted
White listed

Figure 3.2: Authentication

Algorithm 2 Authentication

```
function GETTRANSACTIONS(Fromaccount, datatocheck, Startblock, endblock)
var valid = false;
    for $i$ = startblock, $i <=$ endblock ; $i$++ do
        block = eth.getblock(i, true);
        if (block.from == fromaccount) then
            print the block;
            if block.input == datatocheck then
                var valid == true;
            else
                var valid == false; return valid;
            end if
        end if
    end for
end function
function CONTRACT_ INSTANCE
    var contract = web3.eth.contract(contract.abi)
    contractinstance = contract.at("contract address")
    return contractinstance
end function
function REQ_PROPOSAL(Public key to be verified, EOA of public key to be verified)
    contract_ instance();
    call the smart contract's proposal() function;
end function
function VOTINGSTART
    contract_instance();
    Watch the req_ voting() event and take the following values :
    propnum;
    Public key(pk) to be verified
    EOA of pk to be verified
    if address of voter valid then
        var valid = getTransactions(EOA of PK, PK, start block number, end block number)
        if valid = true then
            call voting_start() smart contract function with support
        else
            call voting_start with not support;
        end if
    end if
end function
```

```
38: function VOTING_COMPLETED
        if voting_ completed event == true then
40:         to read voting results:
            call smart contract function get_user1();
42:         call smart contract get_user2();
        end if
44: end function
```

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 System Specifications And Hardware Used

Hardware setup includes a Raspberry Pi based CAN test bed that has been configured with blockchain [33]. The Raspberry Pis are running Raspbian OS and private Ethereum blockchain on each node. Two of the nodes are designated as miner nodes for this testbed that are used to add the new blocks into blockchain. Figure 4.1 shows the hardware setup that includes Raspberry Pi nodes configured with Ethereum blockchain.

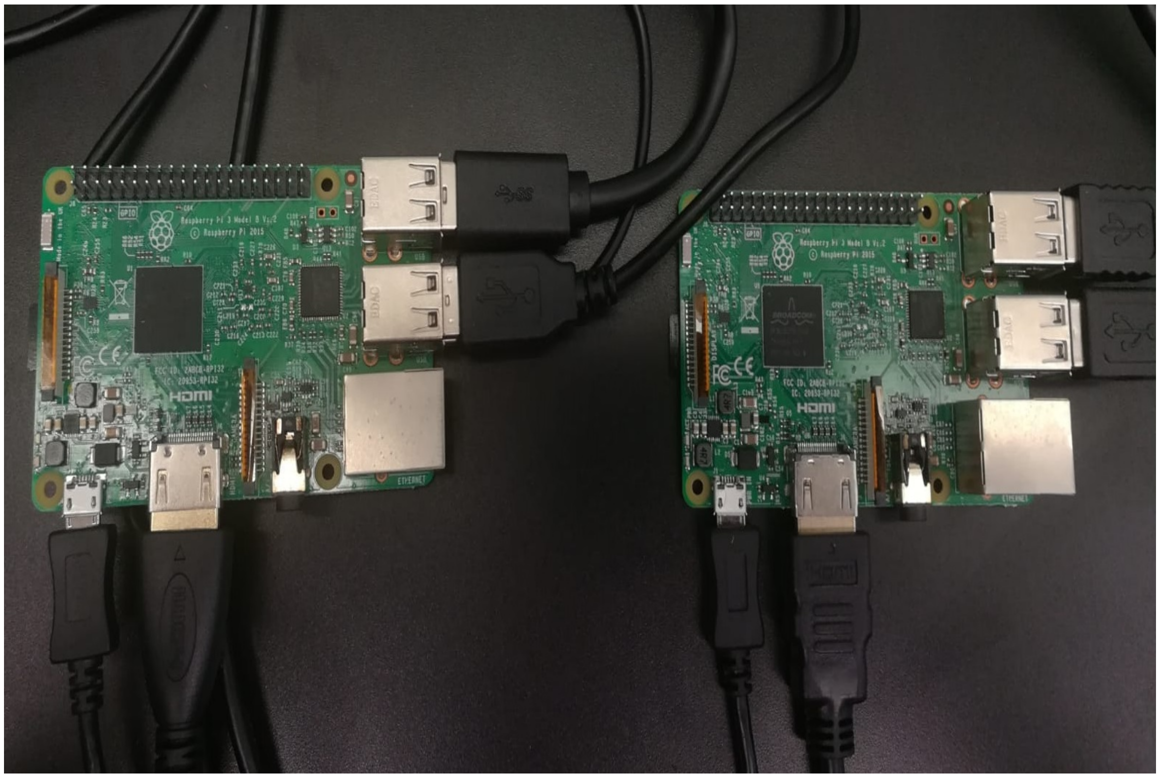


Figure 4.1: Hardware setup for Distributed Key Provisioning

4.2 Setting Up Of Blockchain

The private mode Ethereum blockchain is configured using the following steps:

1. Genesis is the first node in the blockchain. Each blockchain is assigned an id, where the id is stored on all the peer nodes which are participating in that blockchain. In case of Ethereum private blockchain it is stored in a Genesis file including the features of Genesis node, that resides on each node. A snapshot of Genesis file is shown in 4.2, that contains a nonce, which is a random number that is combined with mix hash during the mining using Proof-Of-Work algorithm. Other information includes network id, that is assigned at the time of network initialization. Nodes in Ethereum network use the combination of chain id and network id to stay connected and sign the transactions along with EOA.

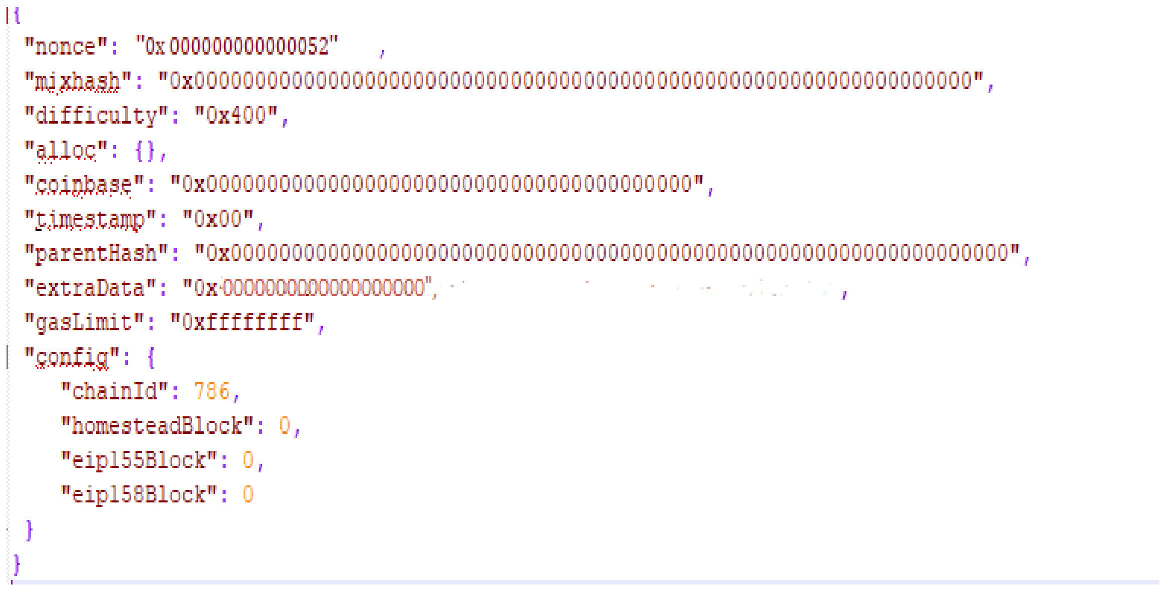
   Difficulty for mining is a threshold value which determines the degree of difficulty for the miners to evaluate a qualified hash in terms of hashing power [1]. This value used to determine the time required for the block generation in the blockchain. Higher the difficulty, more time is required for adding the blocks into blockchain. For private networks with fewer nodes this value is kept low to avoid waiting. In our scheme, we have set our difficulty value as "0x400". Gaslimit is the amount of gas provided by a node to execute a particular operation or a transaction. It is best if assigned high value to execute the operation. Timestamp gives information time at which blocks are added. This value would be the output of a UNIX time().

   The 'alloc' is used to store the wallet addresses if present.The homestead value '0' represents the usage of homested version of Ethereum. The eip155block eip158Block are only used by the developers to suggest the enchancements in the Ethereum. Its value is kept 0 for regular usage.

2. Each node is shared with the Genesis file which is kept up to date and consistent across all the nodes. The new accounts in Ethereum blockchain are created using “accounts new” command at all the nodes. This command create a new EOA account. The nodes are then connected permanently to each other using enode of the node, that is a node id or the public key encoded with URL portion of the node and hostname separated by and stored in datadir folder in a JSON format.

3. The smart contract is a mechanism through which we can combine the security functions such as validation of the block within the blockchain. This capability makes the implementation of smart contract immutable and confirms the authenticity of the security process and make it trusted. The smart contract is a custom process that can be called using function call to the smart contract methods as postprocessing of insertion or retrieval of the blocks in the blockchain. In Ethereum blockchain, the smart contract is compiled to get a bytecode and Application Binary Interface (ABI) and deployed onto the blockchain. A node in the blockchain integrates the smart contract implementation onto the blockchain. The code snippet for deployment is shown in Appendix C.

4. The authentication application is developed to perform validation of keys by traversing through the blockchain and calling smart contract methods. This application is loaded on all the nodes.

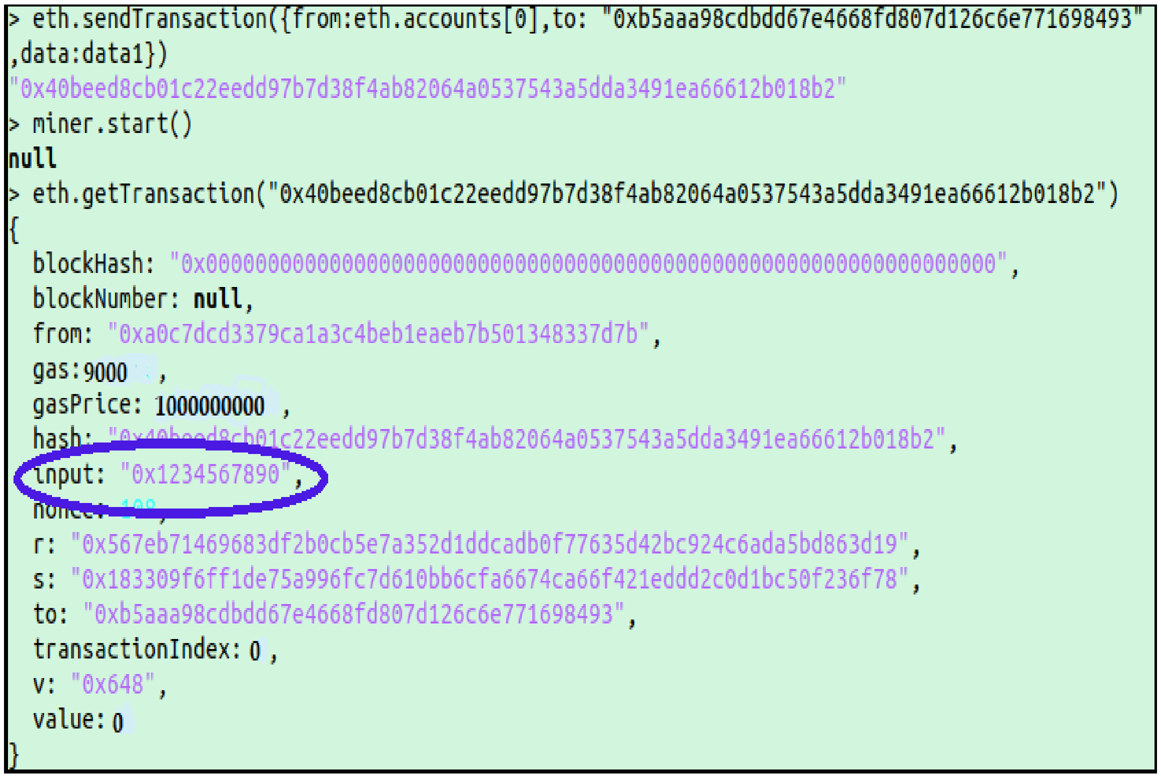


```
{
 "nonce": "0x 0000000000000052"    ,
 "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
 "difficulty": "0x400",
 "alloc": {},
 "coinbase": "0x0000000000000000000000000000000000000000",
 "timestamp": "0x00",
 "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
 "extraData": "0x0000000000000000000",                                   ,
 "gasLimit": "0xffffffff",
 "config": {
    "chainId": 786,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
 }
}
```

Figure 4.2: genesis file

4.3 Key Storage Into The Blockchain

All the keys are stored in the form of transaction in the blockchain. During the validation process, all nodes go through their copy of the blockchain and validate them. Figure4.3 shows the storage of public keys into blockchain of node1 and the highlighted section is the public key of node1. All other nodes are store their public keys as blocks in the blockchain in the similar way.

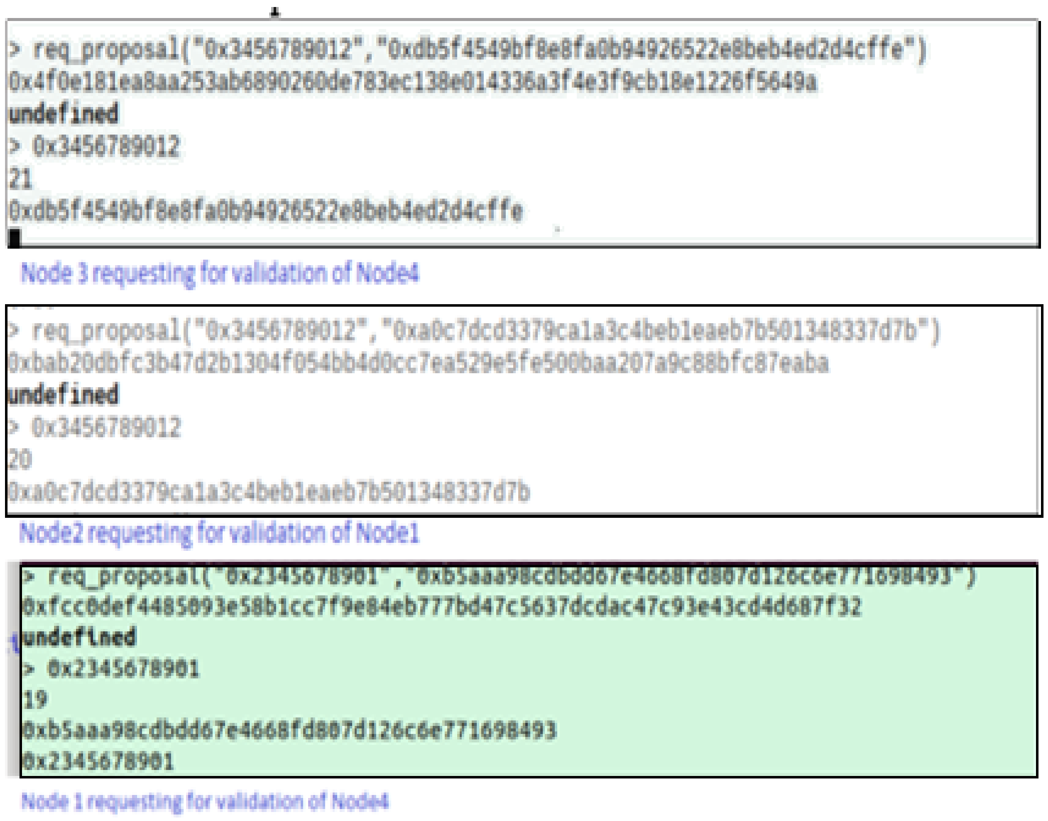


```
> eth.sendTransaction({from:eth.accounts[0],to: "0xb5aaa98cdbdd67e4668fd807d126c6e771698493"
,data:data1})
"0x40beed8cb01c22eedd97b7d38f4ab82064a0537543a5dda3491ea66612b018b2"
> miner.start()
null
> eth.getTransaction("0x40beed8cb01c22eedd97b7d38f4ab82064a0537543a5dda3491ea66612b018b2")
{
  blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",
  blockNumber: null,
  from: "0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b",
  gas:9000 ,
  gasPrice: 1000000000 ,
  hash: "0x40beed8cb01c22eedd97b7d38f4ab82064a0537543a5dda3491ea66612b018b2",
  input: "0x1234567890",
  nonce: 108 ,
  r: "0x567eb71469683df2b0cb5e7a352d1ddcadb0f77635d42bc924c6ada5bd863d19",
  s: "0x183309f6ff1de75a996fc7d610bb6cfa6674ca66f421eddd2c0d1bc50f236f78",
  to: "0xb5aaa98cdbdd67e4668fd807d126c6e771698493",
  transactionIndex: 0 ,
  v: "0x648",
  value: 0
}
```

Figure 4.3: Storing of keys into blockchain

4.4 Validation Request

During the key retrieval from blockchain, the validation process of keys is processed by the node using Req_Proposal() function call. This function in turn, calls a smart contract function called Proposal() with details of the public key to be verified and also Ethereum address of the node. An event is occurred to notify and give details about the proposal number of the request along with the public key to be verified and Ethereum address of node whose public key is to be verified to the other nodes. Figure 4.4 shows request proposals for validation event occurred at nodes when requested. Each request is given a proposal number.



```
> req_proposal("0x3456789012","0xdb5f4549bf8e8fa0b94926522e8beb4ed2d4cffe")
0x4f0e181ea8aa253ab6890260de783ec138e014336a3f4e3f9cb18e1226f5649a
undefined
> 0x3456789012
21
0xdb5f4549bf8e8fa0b94926522e8beb4ed2d4cffe
```

```
> req_proposal("0x3456789012","0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b")
0xbab20dbfc3b47d2b1304f054bb4d0cc7ea529e5fe500baa207a9c88bfc87eaba
undefined
> 0x3456789012
20
0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b
```

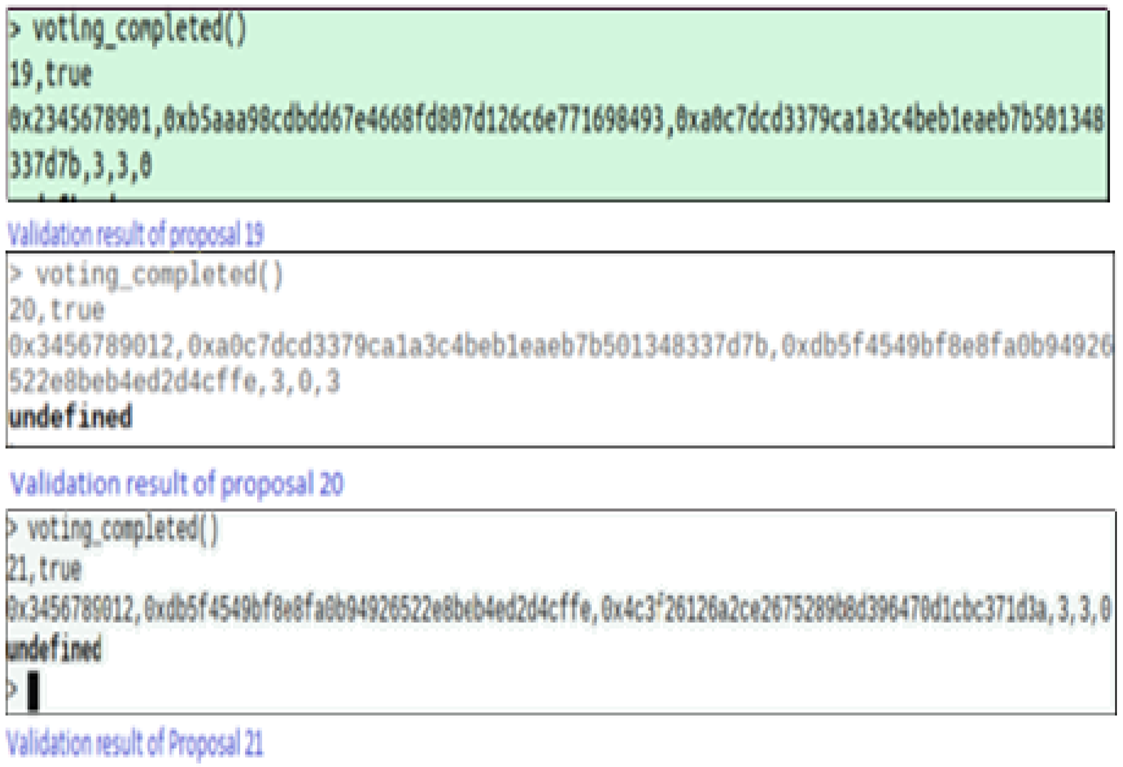
```
> req_proposal("0x2345678901","0xb5aaa98cdbdd67e4668fd807d126c6e771698493")
0xfcc0def4485093e58b1cc7f9e84eb777bd47c5637dcdac47c93e43cd4d687f32
undefined
> 0x2345678901
19
0xb5aaa98cdbdd67e4668fd807d126c6e771698493
0x2345678901
```

Figure 4.4: Requesting For Validation

4.5 Validation

During the validation process, all the nodes traverse through blockchain when the request for validation is made. Each node responds to the vote either in support or not support based on the search results. In the private Ethereum implementation, function call voting_completed() triggers the event to notify the requested node after the search is complete and results are updated in the smart contract. Figure 4.5 shows the result of voting validation of a corresponding proposal number. The result has the following format:

Validation Result = (Validation proposal number, Proposal completed or not, Public key requested for validation, Ethereum address of node of public key to be verified, Ethereum address of the node which requests the validation, Total number of votes, Support votes, Not in support votes).



```
> voting_completed()
19,true
0x2345678901,0xb5aaa98cdbdd67e4668fd807d126c6e771698493,0xa0c7dcd3379ca1a3c4beb1eaeb7b501348
337d7b,3,3,0
```

Validation result of proposal 19

```
> voting_completed()
20,true
0x3456789012,0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b,0xdb5f4549bf8e8fa0b94926
522e8beb4ed2d4cffe,3,0,3
undefined
```

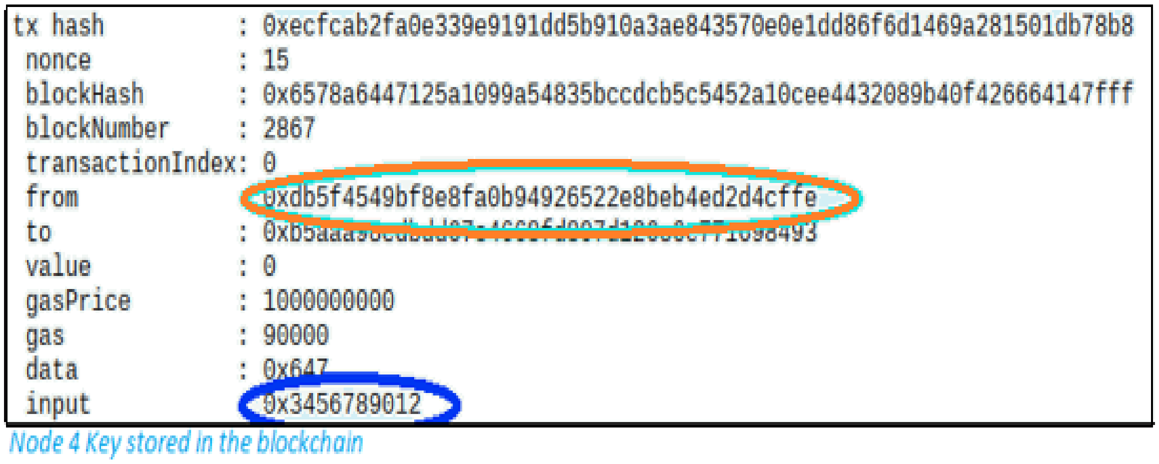
Validation result of proposal 20

```
> voting_completed()
21,true
0x3456789012,0xdb5f4549bf8e8fa0b94926522e8beb4ed2d4cffe,0x4c3f26126a2ce2675289b8d396470d1cbc371d3a,3,3,0
undefined
>
```

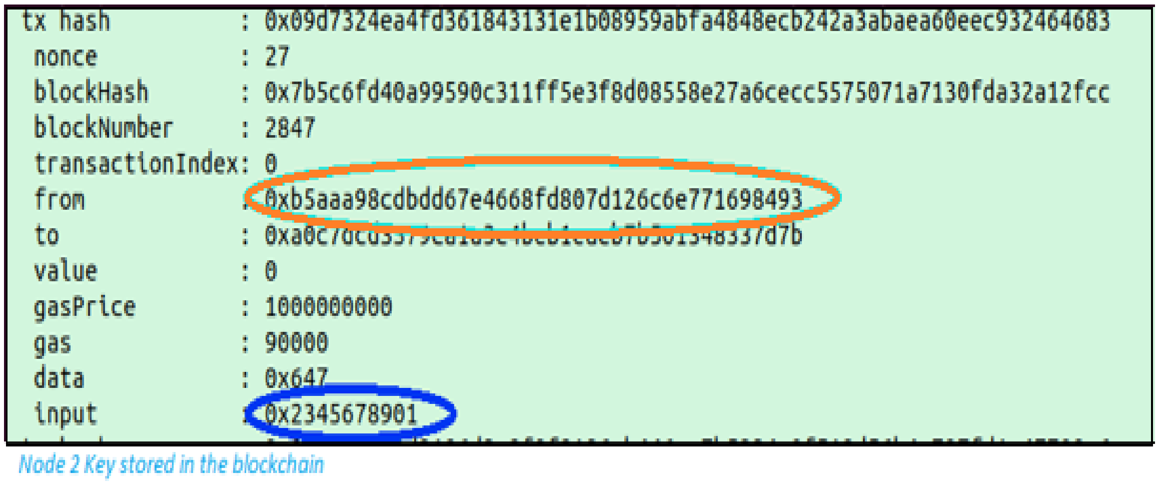
Validation result of Proposal 21

Figure 4.5: Validation(Authentication) Results

Figure 4.5 shows the response of the validation from all the nodes for proposal 20 depicting that all the given votes are not supporting because the key is not valid. It can be seen that the key stored in the blockchain and requested key for validation are different and hence the spoofing in the communication is captured. The correct key stored in the blockchain is shown in Figure4.3. Whereas votes for the other two proposals are in support because the requested key for validation is the same as the key stored in the blockchain when traversed as shown in Figure 4.6. In this way, we can authenticate the public keys sent by the nodes are malicious or not. If all the votes are in support, then the public key is valid else it blacklists the node and requests again for the correct public key and this scheme does not allow a node to vote twice for the same proposal.



```
tx hash          : 0xecfcab2fa0e339e9191dd5b910a3ae843570e0e1dd86f6d1469a281501db78b8
 nonce           : 15
 blockHash       : 0x6578a6447125a1099a54835bccdcb5c5452a10cee4432089b40f426664147fff
 blockNumber     : 2867
 transactionIndex: 0
 from            : 0xdb5f4549bf8e8fa0b94926522e8beb4ed2d4cffe
 to              : 0xb5aaa98cdbdd67e4668fd807d126c6e771698493
 value           : 0
 gasPrice        : 1000000000
 gas             : 90000
 data            : 0x647
 input           : 0x3456789012
```



```
tx hash          : 0x09d7324ea4fd361843131e1b08959abfa4848ecb242a3abaea60eec932464683
 nonce           : 27
 blockHash       : 0x7b5c6fd40a99590c311ff5e3f8d08558e27a6cecc5575071a7130fda32a12fcc
 blockNumber     : 2847
 transactionIndex: 0
 from            : 0xb5aaa98cdbdd67e4668fd807d126c6e771698493
 to              : 0xa0c7dcd[illegible]48337d7b
 value           : 0
 gasPrice        : 1000000000
 gas             : 90000
 data            : 0x647
 input           : 0x2345678901
```

Figure 4.6: Keys Of Node 4 and Node 5 Stored In The Blockchain

4.6 Security Analysis

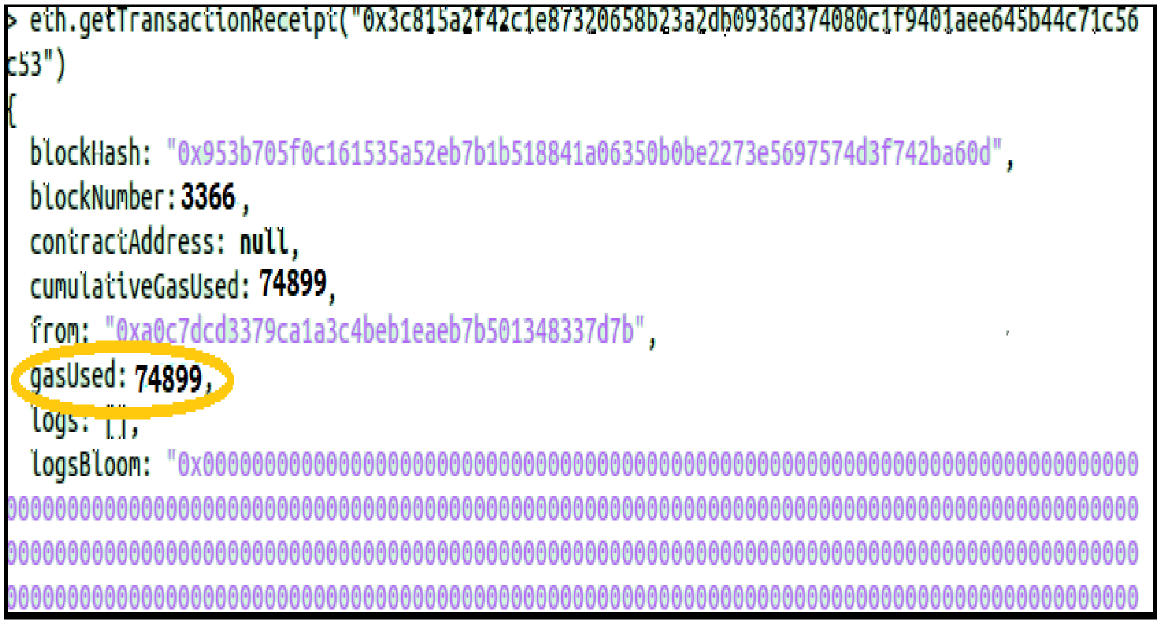
The proposed scheme on distributed key provisioning integrates blockchain technology and builds on the assumption that the Ethereum model of smart contract and blockchain is secure. This section analyses the security of our proposed scheme. The attack model includes key spoofing, key update vulnerabilities, false identity by adding public keys into the blockchain with other Ethereum address, and unauthorized access, where malicious node tries to masquerade a legitimate node to read the keys. The attacks on the voting scheme such as double voting, where the malicious node can change votes by voting again.

To prevent the Unauthorized accessibility into blockchain we are using private blockchain where nodes are added in a trusted environment and have a limited number of nodes involved in the system such as in automotive there are a few electronic control units integrated. To further restrict the authority, IBM hyper ledger can be integrated to provide permissioned blockchain so that unknown identities cannot participate in the network. The nodes are enrolled into the fabric using a trusted membership service provider.

The transaction mechanism in Ethereum prevents the False Identity attack using a nonce. whenever a transaction occurs the nonce attached with Ethereum address is incremented. Even if the malicious node is successful in generating the same address as a victim node it cannot have the same nonce value. During the validation of transaction, the miner checks the nonce value and reject the malicious node.

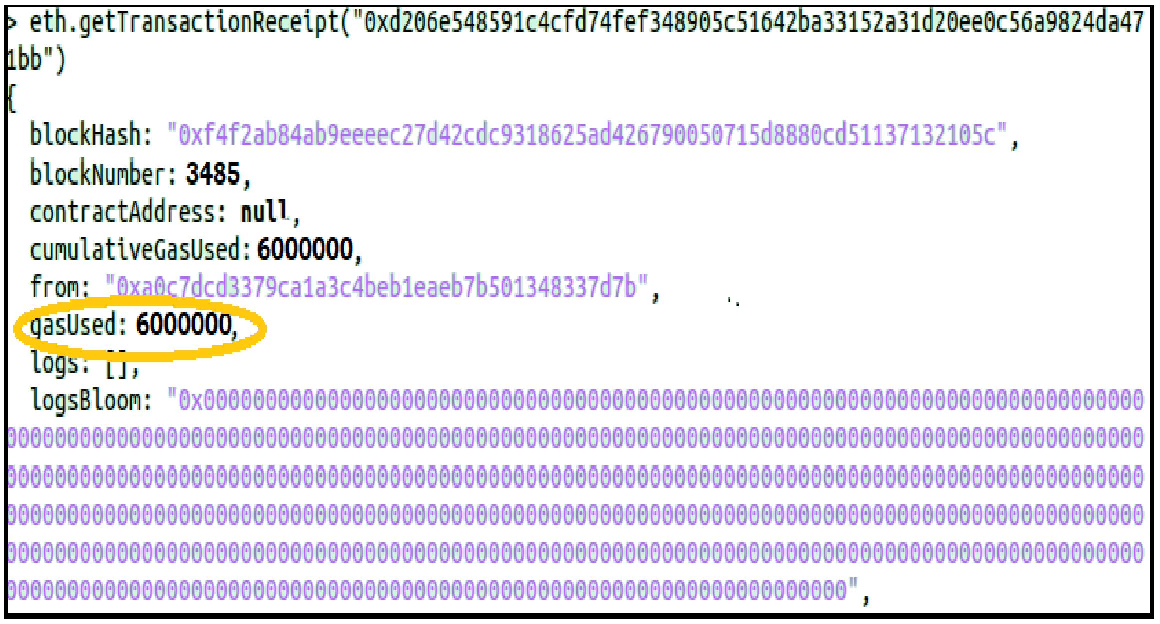
The blocks in the blockchain are linked with each other with their previous hash. If a malicious node tries to change a single block then the whole chain must be changed and by then other nodes rejects this node. In this way, we can prevent Spoofing Attacks.

Figure 4.7 and 4.8, shows the transaction receipt of calling smart contract function called Voting_Start() of a proposal number from Ethereum address of node 1. Each transaction to the smart contract is sent with certain of amount gas. The function uses required gas and sends back the remaining gas to the account. The gas limit set for the sending transaction to Voting_Start() smart function is 6000000. Initially, when the node1 tried to vote once the gas spent value is less that gas limit as shown in Figure 4.7, but when it tried to vote again for the same proposal the gas limit is same as gas spent as shown in Figure 4.8. This means that all the gas send via transaction is spent but not sufficient to execute the transaction where the actual gas required is shown in 4.7. This show that the voting is not executed second time. This feature helps if a malicious node is trying to change the votes. Thus we can prevent Double voting.



```
> eth.getTransactionReceipt("0x3c815a2f42c1e87320658b23a2db0936d374080c1f9401aee645b44c71c56
c53")
{
  blockHash: "0x953b705f0c161535a52eb7b1b518841a06350b0be2273e5697574d3f742ba60d",
  blockNumber: 3366,
  contractAddress: null,
  cumulativeGasUsed: 74899,
  from: "0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b",
  gasUsed: 74899,
  logs: [],
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 4.7: Transaction Receipt of node1 trying to vote first time some proposal



```
> eth.getTransactionReceipt("0xd206e548591c4cfd74fef348905c51642ba33152a31d20ee0c56a9824da47
1bb")
{
  blockHash: "0xf4f2ab84ab9eeeec27d42cdc9318625ad426790050715d8880cd51137132105c",
  blockNumber: 3485,
  contractAddress: null,
  cumulativeGasUsed: 6000000,
  from: "0xa0c7dcd3379ca1a3c4beb1eaeb7b501348337d7b",
  gasUsed: 6000000,
  logs: [],
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000",
```

Figure 4.8: Transaction receipt of node1 trying to vote again for the same proposal

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

We propose a decentralized key provisioning scheme for Elliptic Curve Cryptography (ECC) based secure key storage, update and key sharing mechanism that provides identity and authentication of communicating nodes. This scheme integrates private blockchain and smart contracts to validate the keys. The proposed scheme implements voting mechanism to identify key spoofing and compromise during the sharing process. Unauthorized access and false identity issues are also resolved in the proposed scheme using private blockchain and transaction mechanism of Ethereum respectively. The double voting problem and modification of votes is prevented using smart contract, code snippets are integrated into the blockchain.

For future work, we will investigate permissioned blockchain schemes to restrict the access of blockchain by malicious nodes and improve the performance by using Proof-of-stake algorithm instead of Proof-Of-Work consensus algorithm for lightweight nodes. The scheme will be demonstrated on the CAN FD network with the ECC based encryption and key provisioning applications on the reconfigurable ECU's that have other security mechanisms such as Physical unclonable function responses to generate keys in order to develop a secure key generation.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” https://bitcoin.org/bitcoin.pdf, Mar 2009.

[2] “What is blockchain technology?.” https://www.cbinsights.com/research/what-is-blockchain-technology/, Sep 2018.

[3] C. Paar, “Understanding cryptography.”

[4] S. LANDER, “Advantages & disadvantages of symmetric key encryption.” https://itstillworks.com/advantages-disadvantages-symmetric-key-encryption-2609.html.

[5] M. Rouse, “Assymetric cryptography.” https://searchsecurity.techtarget.com/defi nition/asymmetric-cryptography.

[6] R. Hunt, “Pki and digital certification infrastructure.” Proceedings. Ninth IEEE International Conference on Networks, ICON 2001., Oct 2001.

[7] C AdamsSLloyd, “Understanding pki: Concepts, standards and deployement..”

[8] W. Che, F. Saqib, and J. Plusquellic, “Puf-based authentication,” 2015.

[9] W. Che, M. Martin, G. Pocklassery, V. K. Kajuluri, F. Saqib, and J. Plusquellic, “A privacy-preserving, mutual puf-based authentication protocol.” Cryptography, 2016.

[10] B. Halak, M. Zwolinski, and M. S. Mispan, “Overview of puf-based hardware security solutions for the internet of things,” 10 2016.

[11] “Information technology – trusted platform module library.” https://www.iso.org/standard/66510.html, Jan 2018.

[12] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, “A secure communication framework for ecus.” Advances in Science, Technology and Engineering Systems Journal, 08 2017.

[13] A. Hoeller and R. Toegl, “Trusted platform modules in cyber-physical systems: On the interference between security and dependability.” 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), April 2018.

[14] J. Margulies and M. Berg, “That certificate you bought could get you hacked.” IEEE Security Privacy, Sep. 2016.

[15] K. Zetter, “Diginotar files for bankruptcy in wake of devastating hack.” Wired magazine, Sep 2011.

[16] H. Adkins, “An update on attempted man-in-the-middle attacks.” Google Online Security Blog, 2011.

[17] A. Niemann and J. Brendel, “A survey on ca compromises.” https://docplayer.net/55405696-A-survey-on-ca-compromises.html.

[18] R. Oppliger, “Certification authorities under attack: A plea for certificate legitimation.” IEEE Internet Computing journal, Jan 2014.

[19] A.Abdul-Rahman, “The pgp trust model.” EDI-Forum:TheJournalofElectronicCommerce, 10 1997.

[20] E. K. B. Laurie, A. Langley, “Certificate transparency.” Technical Report, 10 2013.

[21] b. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, “Cecoin: A decentralized pki mitigating mitm attacks.” Future Generation Computer Systems, 10 2017.

[22] C. Fromknecht and D. Velicanu, “A decentralized public key infrastructure with identity retention.” IACR Cryptology ePrint Archive, vol. 2014, pp. 803, 2014., 2014.

[23] A. Singla and E. Bertino, “Blockchain-based pki solutions for iot.” 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Oct 2018.

[24] Y. Hu, Y. Xiong, W. Huang, and X. Bao, “Keychain: Blockchain-based key distribution.” 2018 4th International Conference on Big Data Computing and Communications (BIGCOM), Aug 2018.

[25] B. Leiding, C. Cap, T. Mundt, and S. Rashidibajgan, “Authcoin: Validation and authentication in decentralized networks.” MCIS 2016, Sep 2016.

[26] A.Rosic, “What is blockchain technology? a step-by-step guide for beginners,” 2016.

[27] A Rosic, “Proof of work vs proof of stake: Basic mining guide.” Blockgeeks, 2017.

[28] R. King, “Blockchain explained: The ultimate guide to understanding how blockchain works.” Bitdegree, May 2019.

[29] V. Buterin, “A next-generation smart contract and decentralized application platform.” weusecoins.com white paper, 2014.

[30] S. Sil, “Understanding how ethereum functions and its various components.” https://www.hcltech.com/blogs/understanding-how-ethereum-functions-and-its-various-components, Sep 2017.

[31] C. Chainfund, “Blockchain basics 02: A complete view on ethereum blockchain,” Jul 2018.

[32] Ethereum, “Geth json rpc.” https://github.com/ethereum/wiki/wiki/JSON-RPC.

[33] “Can transreceivers.” http://skpang.co.uk/catalog/pican2-duo-canbus-board-for-raspberry-pi-23-p-1480.html.

[34] Ethereum, “Web3 java script api,.” https://github.com/ethereum/wiki/wiki/JavaScript-API.

APPENDIX A: VALIDATION VOTING SMART CONTRACT

```
   pragma solidity ∧0.5.1; contract mine_contract{
//string memory comparision
function equal(string memory _a, string memory _b) internal pure returns (bool) {
string memory a = _a;
string memory b = _b;
return(keccak256(abi.encode(a)) ==keccak256(abi.encode(b)));

   } uint256 public people_count = 0;
//This event is used to notify the request for for validation.
event req_voting(
string tocheck,
uint prop_num1,
address tocheckaddress
);
//This event is used to notify that the validation is completed.
event voting_finished(
uint proposalnumber,
bool donevoting
);
//This structure defines validation requests,public key to be verified and results in
the form of proposal.
struct proposals{
uint prop_id;
bool completed;
//uint256 toverify;
string toverify;
```

```
address recv;
address send;
uint256 total;
address id;
uint256 insupport;
uint256 notsupport;
}
uint prop_num = 0;
mapping (uint=>proposals) public proposalcheck;
function proposal(string memory _toverify1, address _recv) public{
prop_num++;
proposalcheck[prop_num] = proposals(prop_num, false, _toverify1, _recv, msg.sender,
0, _recv, 0, 0);
emit req_voting(_toverify1, prop_num, _recv);
}
//function is used to cast votes for validation.
function voting_start(uint256 _propnum, string memory _data3, address _tocheck1,
uint no_nodes) public returns(bool){
if (proposalcheck[_propnum].completed == true) revert();
else{ if (msg.sender == proposalcheck[_propnum].id) revert();
else{
  if ( _tocheck1 == proposalcheck[_propnum].recv){
if (equal(_data3, proposalcheck[_propnum].toverify) ) {

  proposalcheck[_propnum].insupport++;}
else {
proposalcheck[_propnum].notsupport++;}
}
proposalcheck[_propnum].id = msg.sender;
proposalcheck[_propnum].total = proposalcheck[_propnum].total+1;
if (proposalcheck[_propnum].total == no_nodes){
proposalcheck[_propnum].completed = true;
}
if( proposalcheck[_propnum].completed == true){
emit voting_finished(_propnum, proposalcheck[_propnum].completed );
}}}}
//used to read propsoal struct value.
function getUser(uint256 _index) public view returns(uint, bool ) {
uint _a =0;
_a++;
return(proposalcheck[_index].prop_id, proposalcheck[_index].completed);
}
//used to read propsal struct values.
function getuser2(uint _index2)public view
returns(string memory, address, address, uint256, uint256, uint256){
return(proposalcheck[_index2].toverify, proposalcheck[_index2].recv, proposalcheck[_index2].send,
proposalcheck[_index2].total, proposalcheck[_index2].insupport,
proposalcheck[_index2].notsupport);
}
}
}
}
```

APPENDIX B: AUTHENTICATION

The following javascript code is used for traversing through block chain, Calling the smart contract instance and execution of contract functions.

```
// This function is used to traverse through the blocks of blockchain and checks if the public key to be verified is present in it.
function get_transactions(myaccount, pub_key, startingblock, lastblocknumber) {
if (lastblocknumber == null) {
lastblocknumber = eth.blockNumber;
console.log(”Using lastblocknumber: ” + lastblocknumber);
}
if (startingblock == null) {
startingblock = lastblocknumber - 1000;
console.log(” Using startingblock: ”+ startingblock);
}
console.log(” Searching for transactions to/from account ´” + myaccount + ” ” within blocks + startingblock + ” and ” + lastblocknumber + ”´”);
var valid = false;
for (var i = startingblock;
i <= lastblocknumber;
i++) {
if (i % 1000 == 0) {
console.log(“Searching for the blocks ” + i);
} var block_details = eth.getBlock(i, true);
if (block_details != null && block_details.transactions != null) {
block_details.transactions.forEach( function(s) {
if (my_account == “ ⋆” || my_account == s.from) {
    console.log( “ transaction hash : ” + s.hash + ”\n”
+ “ nonce value : ” + s.nonce + ”\n”
+ “ blockhash value : ” + s.blockHash + ”\n”
+ “ blockNum : ”+ s.blockNumber + ”\n”
+ “ txIndex: ” + s.transactionIndex + ”\n”
+ “ sender : ” + s.from + ”\n”
+ “ receiver : ” + s.to + ”\n”
+ “ Value : ”+ s.value + ”\n”
+ “ gasPrice : ”+ s.gasPrice + ”\n”
+ “ gas : ”+ s.gas + ”\n”
+ “ data : ” +s.v+”\n”
+ “ input/Key stored : ” + s.input);
data1 = s.input;
if(pub_key == s.input){
var data2 = s.input;
valid = true;
return data2;
} } } ) } }
return valid;
}
var propnum = 9;
var propnum1 =13 ; //for new chain or for new contract address make this value 0.
var tocheck;
var tocheckaddress;
//This function is used to call the smart contract function proposal() for requesting
the validation.
function req_proposal(pub_key1, verif_addr){
```

```
var votinginstance1=contractaccess();
var rrr2 =votinginstance1.proposal.sendTransaction(pub_key1, verif_addr,
{from:eth.accounts[0], gas:3000000},function(err,result){console.log(result);
});
}
//This function is used to cast the votes for validation.
function votingstart(){ var votinginstance1=contractaccess();
var event5=votinginstance1.req_voting({from:eth.accounts[0]},{fromBlock:0, toBlock:'latest'});
//To watch the event when request for validation occurs.
event5.watch( function(error,response) { if(response.args.prop_num1 >= ( propnum1)){
propnum = response.args.prop_num1;
tocheckpublickey = response.args.tocheck;
tocheckaddress = response.args.tocheckaddress;
tocheckaddress1 = "";
tocheckaddress1+=tocheckaddress;
} });
var pubkey1 = get_transactions(tocheckaddress, tocheck, 2500);
var no_voters = web3.net.peerCount;
var valid_addr = web3.isAddress(tocheckaddress);
if(valid_addr ==true){
if( pubkey1 ==true){
var rrr4 =votinginstance1.voting_start.sendTransaction(propnum, tocheckpublickey,
tocheckaddress, no_voters,{from:eth.accounts[0],gas:6000000}
,function(err,result){console.log(result);
}) } else{
tocheck = "wrong";
var rrr4 =votinginstance1.voting_start.sendTransaction(propnum, tocheckpublickey
,tocheckaddress,
no_voters,{from:eth.accounts[0],gas:6000000}
,function(err,result){console.log(result);
}) } }}
   function voting_completed(){ var votinginstance1=contractaccess();
var completed;
var rrr2 =votinginstance1.getUser.call(propnum, {from:eth.accounts[0],gas:300000}
, function(err,result){console.log(result);
completed = result.completed;
});
if (completed == true){ var rrr5 =votinginstance1.getuser2.call(propnum,newline
{from:eth.accounts[0] gas:300000},function(err,result){console.log(result);
});
} var rrr5 =votinginstance1.getuser2.call(propnum,{from:eth.accounts[0]
,gas:300000},function(err,result){console.log(result);
});
}
//Calling the contract instance.
function contractaccess(){
var contract = web3.eth.contract([
{
"constant": false,
"inputs": [
{
" name ": " _propnum ",
" type ": " uint256 "
},
{
“name”: “_data3”,
“type”: “string”
},
{
“name”: “_tocheck1”,
“type”: “address”
}, {
“name”: “no_nodes”,
“type”: “uint256”
}
],
“name”: “voting_start”,
“outputs”: [
{
“name”: “”,
“type”: “bool”
}
],
“payable”: false,
“stateMutability”: “nonpayable”,
“type”: “function”
},
{
“constant”: true,
“inputs”: [
{
```

“name”: “”,

“type”: “uint256”

}

],

“name”: “ecu_address”,

“outputs”: [

{

“name”: ””,

“type”: “address”

}

],

“payable”: false,

“stateMutability”: “view”,

“type”: “function”

},

{

“constant”: true,

“inputs”: [],

“name”: “people_count”,

“outputs”: [

{

“name”: “”,

“type”: “uint256”

}

],

“payable”: false,

“stateMutability”: “view”,

“type”: “function”
},
{
“constant”: true,
“inputs”: [
{
“name”: “_index”,
“type”: “uint256”
}
],
“name”: “getUser”,
“outputs”: [
{
“name”: “”,
“type”: “uint256”
},
{
“name”: “”,
“type”: “bool”
}
],
“payable”: false,
“stateMutability”: “view”,
“type”: “function”
},
{
“constant”: false,

“inputs”: [

{

“name”: “_members”,

“type”: “address”

}

],

“name”: “addPerson”,

“outputs”: [],

“payable”: false,

“stateMutability”: “nonpayable”,

“type”: “function”

},

{

“constant”: true,

“inputs”: [

{

“name”: “_index2”,

“type”: “uint256”

}

],

“name”: “getuser2”,

“outputs”: [

{

“name”: “”,

“type”: “string”

},

{

“name”: “”,

“type”: “address”

},

{

“name”: “”,

“type”: “address”

},

{

“name”: “”,

“type”: “uint256”

}, {

“name”: “”,

“type”: “uint256”

},

{

“name”: “”,

“type”: “uint256”

}

],

“payable”: false,

“stateMutability”: “view”,

“type”: “function”

},

{

“constant”: true,

“inputs”: [

{

“name”: “”,
“type”: “uint256”
}
],
“name”: “proposalcheck”,
“outputs”: [
{
“name”: “prop_id”,
“type”: “uint256”
},
{
“name”: “completed”,
“type”: “bool”
},
{
“name”: “toverify”,
“type”: “string”
},
{
“name”: “recv”,
“type”: “address”
},
{
“name”: “send”,
“type”: “address”
},
{

```
“name”: “total”,
“type”: “uint256”
},
{
“name”: “id”,
“type”: “address”
},
{
“name”: “insupport”,
“type”: “uint256”
},
{
“name”: “notsupport”,
“type”: “uint256”
}
],
“payable”: false,
“stateMutability”: “view”,
“type”: “function”
},
{
“constant”: false,
“inputs”: [
{
“name”: “_toverify1”,
“type”: “string”
},
```

{
“name”: “_recv”,
“type”: “address”
}
],
“name”: “proposal”,
“outputs”: [],
“payable”: false,
“stateMutability”: “nonpayable”,
“type”: “function”
},
{
“inputs”: [],
“payable”: false,
“stateMutability”: “nonpayable”,
“type”: “constructor”
},
{
“anonymous”: false,
“inputs”: [
{
“indexed”: false,
“name”: “tocheck”,
“type”: “string”
},
{
“indexed”: false,

“name”: “prop_num1”,

“type”: “uint256”

},

{

“indexed”: false,

“name”: “tocheckaddress”,

“type”: “address”

}

],

“name”: “req_voting”,

“type”: “event”

},

{

“anonymous”: false,

“inputs”: [

{

“indexed”: false,

“name”: “proposalnumber”,

“type”: “uint256”

},

{

“indexed”: false,

“name”: “donevoting”,

“type”: “bool”

}],

“name”: “voting_finished”,

“type”: “event”

```
} ]);
var votingcontract = contract.at(”0xc8f64abbe44c8141a745e36db6c505eaa663af41”);//This
should be the address of the contract after deployement.
return votingcontract;
}
```

APPENDIX C: DEPLOYING OF SMART CONTRACT

The following code snippet is used for deploying the smart contract [34]. The ABI and byte code of the contract is obtained after compiling the smart contract.

```
var Contract1 = web3.eth.contract(“abi of the contract”);
var mine_contract1 = Contract1.new(
{
from: web3.eth.accounts[0],
data: ’Byte code generated after compiling’
gas: ‘4800000’
}, function (s, contract){
console.log(s, contract);
if (typeof contract.address !== ‘undefined’) {
console.log(‘Contract mined at address: ’ + contract.address + ‘ transaction hash is:
’ + contract.transaction Hash);
}
})
```