# BLOCKCHAIN AND ACCESS CONTROL

*A project report submitted as a part of Major-II*
*By*

**Zeya Umayya**
**2K16/CSE/20**

*under the supervision of*

**Asst. Prof. Divyashikha Sethia**

*in partial fulfilment of the requirements*
*for the award of the degree of*

**Master of Technology**

**Department Of Computer Science and Engineering**
**DELHI TECHNOLOGICAL UNIVERSITY**

**July 2018**

# DECLARATION

I, **Zeya Umayya**, 2K16/CSE/20 student of M.Tech (Computer Science & Engineering), hereby declare that the project dissertation titled **"Blockchain and Access Control"** which is submitted by me to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any degree, Diploma Associateship, Fellowship or other similar title or recognition.

**Zeya Umayya**
2K16/CSE/20

# CERTIFICATE

I hereby certify that the Dissertation titled **"Blockchain and Access Control"** which is submitted by **Zeya Umayya**, 2K16/CSE/20, Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

**Asst. Prof. Divyashikha Sethia**
Project Guide
Department of Computer Science and Engineering
Delhi Technological University

Place: Delhi

Date: 17 July 2018

# ACKNOWLEDGEMENTS

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of the life. My greatest thanks are to my parents who bestowed ability and strength in me to complete this work.
I owe profound gratitude to my project guide Ms.Divyashikha Sethia who has been a constant source of inspiration to me throughout this project. It was her competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. Her calm, collected and professionally impeccable style of handling situations not only steered me through every problem but also helped me to grow as a matured person.
I am also thankful to her for trusting my capabilities to develop this project under her guidance.

Date

Zeya Umayya

# ABSTRACT

Blockchain and IoT technologies have gained much popularity nowadays in almost every sector. The blockchain is a decentralized public ledger mainly for cryptocurrency transactions. But because of its important inherent security features, it is being explored in other sectors also, e.g., IoT. Although IoT has evolved greatly through the years but providing fine-grained access control for its users is still a major issue. To tackle such an issue, Ouaddah et. al. has given an idea of using a token-based access control system with the help of second-generation blockchain technology,i.e., Ethereum, and smart contracts. But they have not provided any method for selective access control. CP-ABE(Ciphertext-Policy Attribute-Based Encryption) schemes are known for providing fine-grained role-based access control. So, the inherent features of blockchain technology and CP-ABE schemes can be combined to provide a fine access control mechanism in IoT.

In this work, a more secure and privacy-preserving access control system is deployed Ethereum blockchain with the help of smart-contracts(SC) and CP-ABE for IoT. CP-ABE is used to provide a fine-grained access control to protect IoT data. Implementation and testing of the system is done on a Raspberry Pi2 device and on Linux 16.04 based system. Two smart contracts have been created named as PolicyContract and AuthList. PolicyContract SC is deployed by IoT end, and AuthList SC is deployed by requester end. IoT device acts as a service-end(SE) and other user devices such as PCs act as a requester-end(RE). IoT end is just a geth-client, itâĂŹs not a full ethereum node. Mining work is done on a PC, so it is a full node. Every transaction submitted by IoT is mined by PC miners. IoT device defines access policies using CP-ABE for its resources and converts these access policies to the PolicyContract SC. RE submits its attribute list to SE and SE checks if attribute list satisfies the access policy. If it satisfies then, SE sends create the token transaction, and a token and RE's secret key is added to AuthList SC via PolicyContract SC. RE receives the token and its secret by SE. Whenever RE wants to access a particular resource A, it provides the appropriate token to SE. SE checks for token validity and gives access to that resource A. Resources are encrypted with CP-ABE. RE decrypts the resource using its secret key. After this SE removes the token from AuthList SC. In case SE detects malicious activities by an RE, it will revoke its access by removing the token from AuthList SC.The said access control system is implemented on a Raspberry Pi2 device and on a Dell PC with acceptable timings for access check, token creation and validation.

# Contents

# List of Tables

# List of Figures

# ABBREVIATIONS

| | |
|---|---|
| **SC** | Smart Contract |
| **ETH** | Ether |
| **BTC** | Bitcoin |
| **USD** | United States Dollar |
| **PoW** | Proof of Work |
| **PoS** | Proof of Stake |
| **PBFT** | Practical Byzantine Fault Tolerance |
| **PoA** | Proof of Activity |
| **Tx** | Transaction |
| **SE** | Service End |
| **RE** | Requester End |
| **DAO** | Decentralized Autonomous Organization |
| **ACL** | Access Control List |
| **RBAC** | Role Based Access Control |
| **ABAC** | Attribute Based Access Control |
| **DAC** | Discretionary Access Control |
| **OrBAC** | Organization Based Access Control |
| **CLI** | Command Line Interface |
| **ABE** | Attribute-Based Encryption |
| **kP-ABE** | Key Policy Attribute-Based Encryption |
| **CP-ABE** | Ciphertext Policy Attribute-Based Encryption |

# Chapter 1

# INTRODUCTION

An overview of the overall idea behind the thesis, motivation of the thesis and problem statement is presented in below sections.

## 1.1 Overview

The Internet of Things(IoT) devices communicate with each other and to different devices via wired or wireless connections to provide services. IoT devices collect data using connected sensor and actuators from the environment around them. Further, this collected data goes under some processing and is used for some kind of decision making. Decision making can be done either by other connected IoT devices or some non-IoT devices. There are many real-life examples of IoT services which are useful and make daily human life easier and fast. For example, a temperature sensor connected to an IoT device will continuously provide temperature status of the room. These IoT devices are smart enough to change the temperature of the room automatically, or it can be given some kind of notification to change the temperature of the room using some user help. Nest Thermostat [2] is an example of such application. IoT devices are also being used in industrial areas to monitor important events so that failure can be avoided and necessary measures can be taken according to the data provided by IoT devices. Currently, a huge number of such IoT devices are connected to the Internet. According to Cisco[3] report there will be more IoT devices connected to the Internet by 2020 than the total human population on earth. Although this is good for the IoT market but it also raises concerns over the communication done among these devices and with the Internet. These concerns include access control, data security, privacy and communication issues. To tackle the issue of fine-grained access control and privacy several techniques using blockchain technology has been proposed [4][5][6].

Blockchain is basically a decentralized public ledger mainly used as a backend technology for cryptocurrencies. Satoshi Nakamoto developed the first cryptocurrency in the year 2008 which uses the blockchain[7]. Blockchain evolved with time, and now it has its second generation known as Ethereum blockchain and smart contracts. Smart contracts[8] are digital contracts in the form of code which are executed and deployed on the blockchain to enforce an agreement between untrusted parties in the absence of any third party. So the properties of blockchain and smart contracts can be used to provide fine-grained access control and privacy-preserving system for IoT users. However, the access policy lacks selective role-based access control. Hence we look into the CP-ABE(ciphertext-policy attribute-based encryption) scheme. Ciphertext-Policy Attribute-based Encryption (CP-ABE)[10] is a fine-grained encryption technique which can provide selective access control. Although it is computationally expensive, it has been proved feasible on resource-constrained devices such as a mobile device and IoT device. Different variants of CP-ABE scheme has been given based on the access policy used. Example of access policies is-tree-based access policy, AND-gate with a multivalued-attribute based access policy, AND-gate with positive and negative attributes based access policy, etc.

## 1.2 Motivation

According to European Commision[8], "Internet of Things (IoT) represents the next step towards the digitization of our society and economy, where objects and people are interconnected through communication networks and report about their status and/or the surrounding environment". IoT is playing an important part in almost every sector as the communication done between these devices are helpful in decision making and preventing catastrophic events from happening. For example fire alarms, automatic doors, etc. IoT devices are able to sense, actuate, process and store the data. IoT has made the cities smart by collaborating with these smart devices. Such an example can be traffic management, citizen observatory, energy management and flood detection and prediction systems using smart grids.

Access control is mainly provided and managed by centralized authorities; it can be manufacturers, service providers or governments. These centralized parties are able to gain unauthorized access and therefore collecting and analyzing sensitive user data. These unauthorized actions pose a threat to user privacy and data security. So IoT devices need a decentralized system which can provide all IoT services with user privacy, data security and with a fine-grained access control mechanism. Blockchain is such promising technologies. [12]Blockchain functions without any central authority. The blockchain is growing at a faster rate. The first generation of blockchain mainly deals with Bitcoin. Ethereum and smart contracts come under the second generation of the blockchain. Smart Contracts which are executed and deployed on top of the blockchain are also impacting public sectors, e.g., finance, healthcare, real-state, etc. The decentralized nature of blockchain makes it very useful for trustless parties to interact[27]. Furthermore, CP-ABE(ciphertext-policy attribute-based encryption) is used to provide fine-grained access control using different types of access structure. It has many variants based on access structure used. CP-ABE using Tree access-structure[10] and AND-gate with multivalued attributes[11] can be used in IoT devices. CP-ABE will also secure data by encrypting it using pairing-based cryptography.

In this work, we provide an architecture and its implementation comprising of fine-grained access control, user privacy and data security with the help of Ethereum blockchain, smart contracts and CP-ABE. Any type of interaction between IoT devices and users will be done through Ethereum blockchain and smart contracts. Access will be token-based. Legitimate users will be able to access data provided by IoT devices using valid tokens. Smart contracts will save tokens and provide it to users on demand.

## 1.3 Problem Statement

Access control in IoT is a challenging issue. Most of the solutions given till date are based on a centralized party. But relying on a centralized entity is very risky for user data. These centralized workstations store user sensitive information. These data can be shared with healthcare providers. Users want their data to be secure. So to provide such system not only a flexible access control system is needed but it is also vital to make sure that this access control system exchanges data in a secure and intact way. Data or services while giving access to requesters should not be compromised. Blockchain technology works without any central authority. An access control system, solving above said problems over blockchain could be built. FairAccess[1][5] is such a system to provide access control using tokens, but it does not specify any selective access control method. Also it does not provide any method to secure data. CP-ABE(ciphertext-policy attribute-based encryption)[10] is a cryptographic method which provides fine-grained access control, data security and preserves privacy. Features of both CP-ABE and FairAccess can be merged to build such a system.

# Chapter 2

# Literature Review

## 2.1 Blockchain

Blockchain technology came into existence after Bitcoin cryptocurrency was introduced to the world by an anonymous person named Satoshi Nakamoto in the year 2008. In his paper[7], a peer to peer electronic cash system was developed. This system was developed in such a manner that it did not involve any central authority for transaction completion. Thus a new decentralized system was developed. The technology behind this cash system is now known as blockchain. Blockchain can be seen as a distributed database of transaction logs. This ledger stores all transactions and digital events happened on the network between two parties.

Two fundamental properties of blockchain technology are-

1. It is public, which means its visibility is open to all as it is stored on the network and also it is not controlled by any single entity. Figure-2.1[12] shows the difference between centralized system and decentralized system.
2. It is encrypted. It uses the public-private key concept to make it secure.

## 2.2 Blockchain Characteristics

Blockchain technology has following characteristics[12]:

- *Decentralized and Distributed :* Before blockchain all types of transactions or exchange of services or information needed a trusted intermediary to validate their transactions. The intermediaries involved must guarantee successful and secure operations and should detect security breaches. But blockchain does not need any central authority to perform any financial transaction or data exchange operations by using decentralized public ledger which is immutable and incorruptible. This immutable public ledger records all transactions ever happened on blockchain in a distributed manner. Everyone on the peer-to-peer network can see this ledger.
- *Immutable and Nonrepudiable:* Blockchain is a chain of blocks. The first block is known as genesis block. After genesis block, each block is having a hash of the previous block. So any data once written to the blockchain, cannot be changed because this will change hash of the block and thus the block next to it will notice the difference between stored hash value and current hash value of the block, and it will not accept it as a valid block, it will break the chain. Thus blockchain is immutable. Every transaction is stored on the blockchain. Thus any peer cannot deny any transaction done by him/her. This ensures non-repudiability.

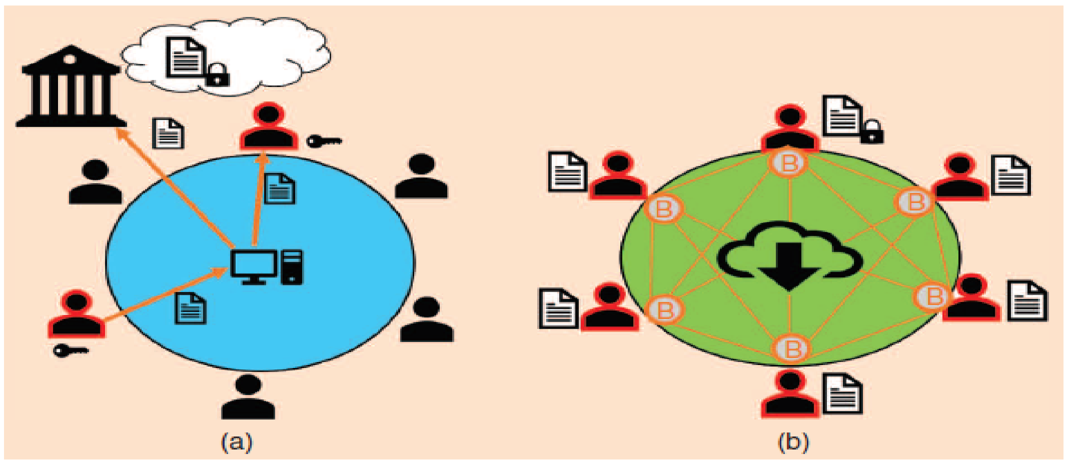


Figure 2.1: (a)Centralized Vs (b)Decentralized System[12]

- *Transparent and Verifiable:* The decentralized and distributed nature of blockchain makes it transparent to its users. Each transaction is validated by the peers using predefined consensus algorithms.
- *Chronological and Time-stamped:*Transactions in a block are arranged in a chronological way to prevent double-spending. Every block has a Unix time-stamp. Having a time-stamp makes it difficult for the adversary to change the block.
- *Trustless Operations:* In blockchain, transactions are processed, validated and successfully added to blockchain without any involvement of third party or central authority. This property makes blockchain trustless.
- *Cryptographically Secure:* In blockchain cryptography is mainly used for two reasons:
  1. To secure the identity of its users
  2. Hashes are used to make blocks immutable.

## 2.3 Blockchain Architecture

The basic architecture of a blockchain will be explored in this section. The blockchain is a ledger of transactions comprising blocks, digital signatures, consensus algorithms, etc.

### 2.3.1 Structure of a Block

Blockchain records all its transactions or events-happened on a database called block. Each block contains hundreds of transactions. A transaction which is not yet a part of a block will be mined by miners. After mining these transactions are added to blocks. The basic structure of a block consists of block header and data part. The meta-data of block header has following entries[15]:

1. *Hash of previous block:* Hash of the previous block is stored on the next block to connect the chain.
2. *Version:* Current version of block.

3. *Timestamp:* The time of block creation.
4. *size:* Size may differ based on the type of blockchain used. For Bitcoin, its maximum value is up to 1Mb, but for Ethereum it's is not fixed.
5. *Difficulty:* The difficulty level for creating nonce. It is different for each block so that each nonce will be different.
6. *Nonce:* In Bitcoin it is a 4-byte number which contains a number of leading zeroes. It is different for each block. The nonce is used to prevent double spending. For Bitcoin, it can also be known as PoW nonce. In Ethereum two types of nonce are used. First one is Account nonce, it stores the number of transactions done by the account. The second one is PoW nonce; it is used for satisfying the current difficulty level.
7. *Merkle Root:* In cryptography, Merkle root is the digital fingerprint of all of its nodes. Blockchain stores Merkle root hash value of all the transactions present in a block. Merkle root hash value is evaluated by repeatedly by calculating the hash of a pair of nodes until there is only one hash value left, which is known as Merkle Root. It is used to verify if a given transaction is a part of the block. It's a part of both BTC(Bitcoin) and ETH(Ethereum).

After header part, a block contains the transactions added to it. Figure-2.2 shows a basic structure of blockchain.

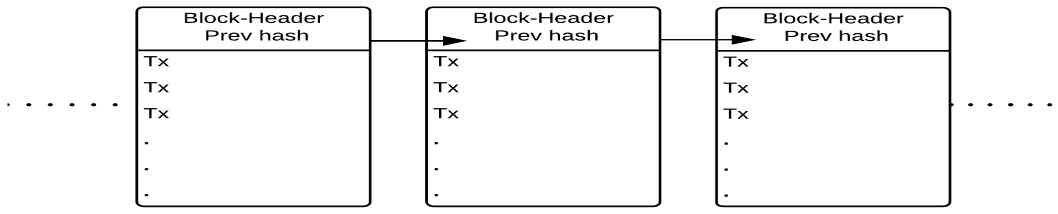


Figure 2.2: Chained Blocks

### 2.3.2 Digital Signature

A digital signature[15] is used to authenticate transactions on the blockchain. Hash values are created by digital signatures,i.e., Bitcoin uses SHA-256 algorithm, and Ethereum uses the ECDSA(Elliptic curve digital signature algorithm).

### 2.3.3 Decentralized Network

Blockchain nodes communicate through the decentralized network[15]. Blockchain has two types of nodes- full nodes and partial nodes. Full nodes store the whole blockchain, and they participate in mining work, while partial nodes only store a part of the blockchain. Partial nodes are also called lightweight nodes. These nodes connect to miners and only store relevant part of the blockchain on their side. When a new transaction is submitted by a node, it's broadcasted on the network. Miner nodes receive these transactions into a block. When PoW is done, miner broadcasts this block to other miners. Miners check for the validity of the block. Block is said to be valid if all transactions in it are not spent twice. Valid blocks are then added to the blockchain. After this phase, miners start working on the next block and use the hash of newly added block as the previous hash value in the new blocks. Nodes follow the longest chain. When two miners simultaneously broadcast the same next block with different versions, nodes may get the two blocks in different orders. In such a scenario, nodes work on the very first one they received and add it to their chain, but they keep a copy of the other block in case the chain with the other node gets longer. If this happens then, nodes switch to the longest valid chain. Therefore, always the chain with the longest length is considered valid.

### 2.3.4 Consensus Algorithms

Consensus algorithms are used to ensure that all nodes are working with the same version of the blockchain. These also prevent double spending in the blockchain. Objectives of consensus algorithm are to have a collaborative, cooperative, inclusive and agreement seeking method. Blockchain does not have a central entity to make decisions, so it needs an algorithm to reach consensus. The very first consensus algorithm is Proof of Work(PoW)[7] used in Bitcoin. Other examples are- PoS(Proof of Stake), PBFT(Practical Byzantine Fault Tolerance) and PoA(Proof of Activity)[13], etc. In this section, PoW and PoS will be discussed.

- *Proof of Work:* In PoW miners have to solve a puzzle for a new block addition to the blockchain. The puzzle is cryptographic. According to the puzzle, miners have to find such a nonce, which when appended to the transaction, gives a hash value with the agreed-upon value of a number of leading zeroes. Finding such a nonce takes time. The number of leading zeroes is generally known as the difficulty in PoW. In BTC, every block validation takes almost 10 minutes. This process requires a great amount of computational power. After PoW, the addition of block to blockchain is done as discussed in the previous section.
- *Proof of Stake:* PoS is created to minimize the computational power needed in PoW. In PoS a new block creator is chosen based on its wealth,i.e., the number of coins it holds. These number of coins are also defined as stake. A miner can mine only up to the capacity of its coins. For example, miners having 10% of the coins will be able to mine only 10% of the blocks. Currently Ethereum currently uses PoW, but it is planning to switch to PoS. Ethereum's PoS is known as Casper[14].

## 2.4 Types of Blockchain

After the emergence of Bitcoin, blockchain technology has grown into many forms. It is mainly of three types- public blockchain, private blockchain, and federated or consortium blockchain. Each type has different variations. From a business point of view, there are two types of blockchain- open and closed[15]. The public is of open type, and close type consists of private and consortium.

### 2.4.1 Public Blockchain

Public blockchains are generally based on PoW and PoS consensus algorithms. As the name suggests, these blockchains are open source and permissionless. Here permissionless means three things, first, anybody with the blockchain code can become a miner and start mining using consensus method. Second, anybody can submit a transaction on the network, and the valid ones will be added to the blockchain. Third, everyone will be able to read transactions on the blockchain. Some examples of public blockchain are- Bitcoin, Ethereum, Monero, Dash, Litecoin, etc. Table-2.1 points out some differences between Bitcoin and Ethereum[40][45].

- *Bitcoin:* First introduced by Satoshi Nakamoto[7] in the year 2008, and first mined in the year 2009. It is based on the PoW consensus protocol. It is decentralized, open source public ledger. It does not involve any central bank or entity for decision making. A Bitcoin user has to maintain an address, a private key, wallet to store its credentials and a Bitcoin managing computer software[15].

- *Ethereum:* Ethereum is another open source public ledger based distributed computing network. Ethereum was first introduced in the year 2013 by Vitalik Buterin, a developer, and researcher, and it went online in the year 2015. It has an added feature known as smart contracts. Smart contracts are code which are executed on the blockchain. A smart contract can store data. In 2016, DAO(Decentralized Autonomous Organization) was instantiated by Ethereum as a smart contract. It was supposed to work like a venture capital fund for crypto and decentralized space. At that time it received 12.7 Ethers(around $150M at that time) via online crowdfunding. But in June 2016, due to a vulnerability in DAO code, one-third of the funding got stolen. As a result, Ethereum was hard forked into two chains to get back the stolen Ethers. The first chain was named Ethereum Classic(ETC) which virtually retrieved the stolen Ethers and second chain kept the old name as Ethereum(ETH).

Table 2.1: Bitcoin Vs Ethereum

| | *Bitcoin* | *Ethereum* |
|---|---|---|
| Inventor | Satoshi Nakamoto in 2009, 6331.890 USD/BTC | Vitalik Buterin, Joseph Lubin, Gavin Wood, etc. in 2015,433.91 USD/ETH |
| Scripting Language | Its limited in functionalities. | Its turing complete. |
| Block Size | Limited to 1MB. | No such limit. |
| Block addition time | around 10 minutes | around 20 seconds |
| Consensus Algorithm | Proof of Work | Currently uses both PoW and PoS. It is planning to completely switch to PoS. |

### 2.4.2 Private Blockchain

Private blockchain as the name suggests is a property of an individual or an organization. The right to read/write is given selectively. In this type, the consensus is reached when its organizing body settles on an agreement. Thus mining is done by these individuals. They can give other users mining rights, but it's based on the will of organizers. Again there is a single authority which is in-charge of everything. It is still debatable if it should be called blockchain because its ideas are quite contrary to why blockchain was made. In this type of setting, not everyone can become full node or partial node with its own will. Not everyone can submit transactions on the private blockchain, or review the blockchain. However, it has certain security advantages, and as part of the organization, it does not need public users to read/write its database[16]. Some examples are-Multichain, Chain Core, and Hyperledger, etc.

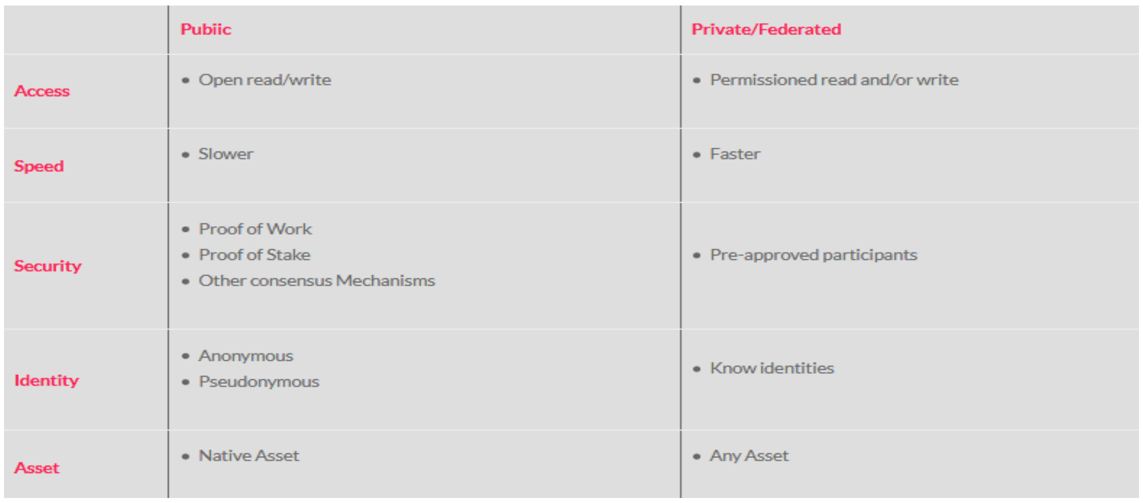
- **Multichain:** Multichain provides an API and a CLI(command line interface) to setup Multichain. It is made to give a permissioned blockchain. It does not have the problems of mining, openness, and privacy. Multichain is developed to support different blockchains at the same time. Block incentives and transaction charges are null. According to [17], deployment is rapid, assets are unlimited, and permissions are fine grained.
- **Hyperledger:** It was created by Linux Foundation in 2015. Its main aim is to provide a collaborative platform for open-source blockchain development. Currently, Hyperledger has different frameworks,e.g., Fabric, Sawtooth, Indy, Burrow, and Iroha. Fabric helps in creating private channels. Sawtooth uses a PoET(Proof of Elapsed Time) algorithm to reach consensus. Indy helps in creating digital identities. The Burrow project includes a permissioned, smart-contract interpreter built in part to the specification of the Ethereum Virtual Machine (EVM). It has different tools like Cello, Explorer, Composer, and Quilt. These tools are used as a ledger interoperability solution, for collaboration, for creating a user-friendly web application[18].

### 2.4.3 Federated/Consortium Blockchain

Federated blockchain has a mixture of both flavors - private and public blockchain[19]. Its more like a private blockchain because it is also controlled by some central entity. But here the central entity is not a single authority but a group of entities. Federated blockchain provides more privacy in transactions. These are faster and light. These consume less energy as compared to the public blockchain. Multiple organizations control it, so it comes under the permissioned blockchain. Some examples of blockchain are R3(banks), EWF(Energy) and Corda, etc.

- **R3:** It is a blockchain-based company whose allies are worlds big financial institutions. Banks are trying to use blockchain technology features to have a secure and fast payment system.

Figure-2.3[19] lists some difference between private, public and consortium/federated blockchain. Federated and private are almost similar in the sense that federated is under the leadership of multiple organizations while private is under a single organization.



| | Pubiic | Private/Federated |
|---|---|---|
| Access | • Open read/write | • Permissioned read and/or write |
| Speed | • Slower | • Faster |
| Security | • Proof of Work<br>• Proof of Stake<br>• Other consensus Mechanisms | • Pre-approved participants |
| Identity | • Anonymous<br>• Pseudonymous | • Know identities |
| Asset | • Native Asset | • Any Asset |

Figure 2.3: Public and Private/Federated Blockchain[19]

## 2.5 Smart Contracts

The term ’smart contract’ was coined by Nick Szabo in 1994[26]. In blockchain world, a smart contract is a piece of code which is executed on blockchain to enforce some rules between parties without any central entity. Smart contracts once deployed on blockchain cannot be changed. So once terms of a contract are decided, no party can alter or modify it. Bitcoin can also provide smart contract features, but its scripting language is very restrictive. However, Ethereum provides a Turing-complete language to program smart contracts. Some capabilities of smart contracts are-

- It may work in a multi-signature fashion.
- It manages agreements between parties.
- It can be used to store data about a particular application,e.g., membership information, etc.

An example of smart contract usage in IoT world can be shown in Figure-2.4[27]. Blockchain and smart contracts manage parcel delivery to your home address. Using sensor devices parcel is tracked at every step. Whenever a new sensor reads it, its location is broadcasted, and other IoT nodes verify it. Thus every event is recorded in the blockchain.

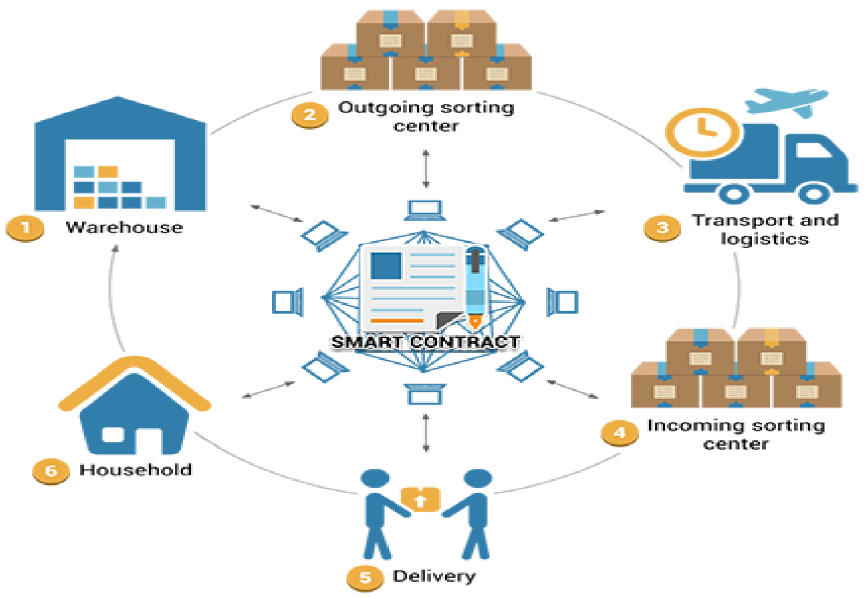

1 Warehouse
2 Outgoing sorting center
3 Transport and logistics
4 Incoming sorting center
5 Delivery
6 Household
SMART CONTRACT

Figure 2.4: Smart Contract In IoT-Use Case[27]

## 2.6 Access Control in IoT

IoT has become a megatrend for industries and consumer products. Authentication and identity management are the main enablers for IoT access control. Access control is a mechanism to identify its users. It also decides which users should be given access to its resources like- files, devices, and sensors, etc. Access in modern operating systems is provided based on the level of its users. For example- superuser is able to access more system resources and files as compared to regular users. Access control mechanism in IoT is necessary to allow only legitimate users and parties to use resources or files. It helps in creating business models such as sensor-as-a-service. Access control is used to share IoT device data selectively to allow predictive maintenance and protection of the sensitive data.

IoT devices are low-power, low-bandwidth and distributed in nature. So before applying standard access control methods like- ACL(access control list), RBAC(Role based access control), ABAC(attribute-based access control) or UCON(usage control models), a deep understanding and applicability should be studied.

- **Access control list:** In this model, an access control matrix is maintained. This matrix, as shown in Table-2.2, has columns indicating resources available and rows indicating the users who can be given access to these resources. On a high level, all access control models try to present the access control matrix using an abstraction that makes managing access control on system level easier. In IoT devices, it is hard to keep the access control matrix updated in a distributed system. And the more the number of users are, the more it becomes difficult to provide fine-grained access control[20]. ACL is sometimes called DAC(Discretionary Access Control).
- **Role based access control:** RBAC uses the abstract version of access control list. Users have roles, and according to their roles permissions are assigned to access resources or

Table 2.2: Access Control Matrix

| S.No. | *File1* | *File1* | *File1* |
|---|---|---|---|
| User1 | RW | | R |
| User2 | | RWX | R |
| User3 | | R | RW |

files. So roles have permissions associated with them, and also resources have permissions associated with them. Access is provided if permissions of both user and resource match. An advantage of RBAC from a user perspective is that roles can be associated with them easily. But the disadvantage of such a system is realized when new roles are added, and the whole system is updated to reflect changes. In RBAC, generally, control servers are centralized.

- **Attribute based access control:** ABAC uses a more abstract version of access control list. Each user has an access policy made of his/her attributes. Attributes can be email-id, user location, job title, date of birth, etc. Likewise, an access policy is associated with each resource. When a user submits an access request, his/her attributes are checked against the resource access policy; if they match, access is given otherwise it is denied. ABAC has almost same problems as RBAC.

## 2.7 CP-ABE

An Attribute-Based Encryption (ABE) is an encryption scheme, where different users have specific attributes and can decrypt a given ciphertext which is associated with an access policy of these attributes. Characteristics of a user, e.g., his name or date of birth can be used for access control of important resources and information. Schemes such as [21] [22] are an example of Identity-Based Encryption (IBE), where does not disclose the identity of the decryptor in any case. Canetti et al.[23] proposed the first ABE scheme inspired by IBE. In IBE schemes there is a one-to-one relationship between an encryptor and a decryptor and the schemes assign only one decryptor for an encryptor. Whereas the ABE schemes assign, many decryptors to a single encryptor by assigning some common attributes to decryptors such as mail-id, gender, age and so on. ABE schemes have two variants, first Key-Policy ABE (KP-ABE) and second Ciphertext-Policy ABE (CP-ABE). KP-ABE [21][23] is a scheme such that it associates each user's private key with an access structure. However, in CP-ABE schemes an access-structure is defined for each ciphertext, which means that an encrypting party can decide who should or should not be allowed to access the ciphertext.

According to the survey of the existing techniques presented by Hwang et al. [24] an ideal ABE scheme must have the following capabilities:

- *Data confidentiality:* Any unauthorized participant cannot find out any information about the encrypted data.
- *Fine-grained access control:* For access control to be flexible, the access rights, even for the users of the same group, are different.
- *Scalability:* The overall performance of an ABE scheme will not go down with the total number of approved participants. Thus we can say that an ABE scheme can deal with the case where the number of the authorized users increases dynamically.
- *Attribute or user-based revocation:* If any participant leaves the system, then his access rights will be revoked by the ABE scheme. Similarly, attribute revocation is inevitable.
- *Accountability:* In all previous ABE schemes, the dishonest/illegal users were able to directly distribute some part of the transformed or original keys such that nobody will know the real distributor of these keys. Accountability should prevent the above problem which is called key abuse.
- *Collusion resistance:* The unauthorized users cannot decrypt the secure data by combining their attributes to match the access policy.

## 2.7.1 Preliminary Construction in CP-ABE

- **Bilinear Group**
  Bilinear groups make the CP-ABE scheme secure against various attacks. The algebraic groups are called bilinear groups, which are groups with bilinear map.
  *Definition (Bilinear map).* Assume $G1, G2$, and $G3$ are three multiplicative cyclic groups of prime order p. A bilinear mapping is done as follows
  $e : G1 \times G2 \to G3$
  *e* is a deterministic function; it takes one element from each group *G1* and *G2* as input, and then produces an element of group *G3*, which satisfies the following criteria
  1. *Bilinearity :* For all
  $$x \in G1, y \in G2, a, b, \quad e(x^a, y^b) = e(x, y)^{ab}.$$
  2. *Non degeneracy:* $e(g_1, g_2) \neq 1$ where $g_1$ and $g_2$ are generators of group G1 and G2 respectively.
  3. e must be computed efficiently.

- **Details of the Algorithms**
  Construction of the CP-ABE scheme is realized using following four algorithms[10]:
  *Setup:* The setup algorithm will select a bilinear group G0 of order p, which is prime, and its generator g. Then, it will select two random exponents $\alpha, \beta \epsilon Z_p$. The public key is as follows:
  $PK = (G0, g, h = g^{\beta}; f = g1/\beta; e(g; g)^{\alpha})$
  and the master key is
  $MK = (\beta, g^{\alpha})$
  *Keygen:* The key generation algorithm will take a set of attributes S as an input and produces a secret key. The algorithm selects a random number $r \epsilon Z_p$, then random number $r_j \epsilon Z_p$ for every attribute $j \epsilon S$. And then it computes the user secret key as

$SK = (D = g^{(\alpha+r)/\beta}; \forall j \epsilon S : D_j = g^r.H(j)^{r_j}; D_j = g_j^r)$
*Encrypt:* The encryption algorithm encrypts any message M under the given tree access structure T. The encryption algorithm first selects a polynomial qx for every node x (including leaves) in the given access tree. The polynomials are chosen in the following way in a top-down fashion, starting from the root node R. For every node x in the access tree, set the degree dx of the polynomial qx one less than the threshold kx of that node, that is, dx = kx - 1. For the root node R of the access tree, this algorithm chooses a random number $s \epsilon Zp$ and sets the value of qR(0) to s. Then, it chooses dR other points of the polynomial qR randomly to define it completely. For any other node x, it sets $q_x(0) = q_{parent(x)}(index(x))$ and chooses dx other points randomly to define qx completely. Let, Y be the set of leaf nodes in the access tree. Then, the ciphertext is constructed by giving the access structure T and computing
$CT = (T, \bar{C} = Me(g;g)^{\alpha s}, C = h^s, \forall y \epsilon Y : C_y = g^{q_{y(0)}},$
$C'_y = H(att(y))^{q_{y(0)}})$
*Decrypt:*First define DecryptNode(CT, SK, x), a recursive algorithm, that takes a ciphertext CT and a private key SK as input. The algorithm is linked with a set of attributes named as S, and a node x from T. Suppose the node x is a leaf node then set i = att(x) and define as follows: If $i \epsilon S$, then compute

DecryptNode(CT;SK;x) $= e(D_i, C_x)/e(D'_i, C'_x)$
$= e(g^r.H(i)^{r_i}, h^{q_x(0)})/e(g^{r_i}, H(i)^{q_x(0)})$
$= e(g,g)^{r.q_x(0)}$

If $i \notin S$, then set DecryptNode(CT, SK, x) = NULL. Now consider the case when x is a non-leaf node. The algorithm DecryptNode(CT, SK, x) then proceeds as follows: DecryptNode(CT, SK, z) will be called for all nodes z that are children of x, and then stores the output as Fz. Suppose Sx is an arbitrary kx-sized set of child nodes z such that $Fz \neq \perp$. If there are no such sets then that node will not be satisfied, and thus NULL value is returned. Otherwise, compute

$$\begin{aligned} Fx &= \prod_{z \epsilon S_x} F_z^{i,S'_x(0)} \\ &= \prod_{z \epsilon S_x} (e(g,g)^{r.q_z(0)})^{i,Sx(0)} \\ &= \prod_{z \epsilon S_x} (e(g,g)^{r.q_{parent(z)}(index(z))})^{i,S_x(0)} \\ &= \prod_{z \epsilon S_x} e(g,g)^{r.q_x(i).i,S'_x(0)} \\ &= e(g,g)^{r.q_x(0)}, (using \quad polynomial \quad interpolation) \end{aligned}$$

Above calculated value of Fx is given to DecryptNode() function for further evaluation. Main decryption algorithm can be defined using DecryptNode() function. The algorithm begins by simply calling the function on the root node R of the tree T. If the tree is satisfied by access structure S, set
$A = DecryptNode(CT; SK; r)$

$= e(g,g)^{r \cdot q_R(0)} = e(g,g)^{rs}$.
The algorithm performs final decryption by computing

$$C'/(e(C,D)/A) = C'/e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{r.s}) = M$$

- **CP-ABE Toolkit**
  The CP-ABE toolkit provides a set of programs implementing a ciphertext-policy attribute-based encryption scheme. CP-ABE toolkit uses the PBC library for algebraic operations. The code is divided into two different packages, named as libbswabe (a library implementing the core crypto operations) and cpabe(higher level functions and user interface). The CP-ABE toolkit provides four command line tools to be used to perform the different operations of the CP-ABE scheme. These command line tools are developed for easy invocation by larger systems in addition to manual usage. [36]

  1. cpabe-setup: This command will generate a public key and a master secret key.
  2. cpabe-keygen: This command will generate a user private key using the given set of attributes.
  3. cpabe-enc: This command will encrypt the input file according to the associated access policy, which is an expression regarding attributes.
  4. cpabe-dec: It decrypts a file using a private key.

  Apart from above four algorithms, I have added another algorithm to check if requester end's attributes satisfy service end's policy. This algorithm is named as *cpabe-pcheck*. It takes the public key, the requester's attributes and service's policy as input and gives output stating whether the policy is satisfied or not.

## 2.8 Summary

Different access control models have been discussed in this chapter. RBAC is a better than ACL model in a sense that RBAC provides more security. RBAC works at a more granular level than ACL. RBAC does not only see the identity of a user but sees his/her role in the organization. Between RBAC and ABAC, ABAC has more flexibility. ABAC provides more fine-grained access to access resources. RBAC is not dynamic in nature as compared to ABAC. RBAC is not able to deal with contextual data like device type, user location, and date, etc. Therefore ABAC is more suitable to use in IoT devices than RBAC and ACL. But all these models are centralized. CP-ABE is a cryptographic encryption method for securing data and giving a fine-grained access control. CP-ABE is different from ABAC in a sense that CP-ABE gives the authorization power to the data itself than to the control system and also ABAC does not secure data rather it focuses mainly on providing an access control system. CP-ABE can be used in both centralized and decentralized manner[25].

Blockchain is a decentralized, distributed, immutable, non-repudiable, transparent, verifiable, time-stamped and trustless public ledger. Blockchain and smart contract can be merged with CP-ABE to be used in IoT devices to provide a secure, decentralized, transparent and privacy preserving access control system.

# Chapter 3

# Related Work

## 3.1 Bitcoin-Blockchain based solutions

Different access control models have been proposed lately, but mostly these are based on a centralized approach. For example, Seitz et. al.[32] have proposed an access control method for IoT. Their method is flexible, fine-grained and suitable for low memory and power devices. But it depends on a central authority.

Maesa et. al.[31] have used Bitcoin framework to enforce access policies. In their method, access control policies and their hashes are stored in some form. Users can see access policies stored on the blockchain and can pass their access tokens temporarily to others. If any user is denied the access, he/she can see the reason behind it by checking access policy on the blockchain.

Ouaddah et. al.[4] have proposed an access control system for IoT based on Bitcoin framework. They have used OrBAC(Organization Based Access Control) model. In their scheme, the token is used as a means to provide access to IoT resources. Raspberry Pi2 board with a dedicated camera is used as a resource. For Bitcoin-wallet bitcoinj is used and for test the network, regtest is used[5]. Their scheme is named as FairAccess. FairAccess does not provide any method to secure the resource, and the scripting language used by Bitcoin is not flexible and turing complete as scripting language of Ethereum.

## 3.2 Ethereum-Blockchain based solutions

Azaria [33] have proposed a framework for maintenance of medical records and access control management using blockchain technology. Their scheme is named as MedRec, which uses multiple contracts to manage such a system like [28]. Fujimura et. al.[34] proposed a similar system in which it controls the actions via video rights management. Zyskind et. al.[35] presented another system to protect personal data by storing it in DHT network. It is off-Blockchain. Only a pointer to this data is kept on the blockchain.

Zhang et. al.[28] proposed a framework using multiple smart contracts to provide access control in IoT. They created two main contracts named as judge contract(JC) and register contract(RC). JC implements a misbehavior judging method. JC receives mis-behavior information from ACC (access control contracts), it judges and submits a request for the penalty from misbehaving entity. RC is used to register, delete or update these methods. Some study is done by

Christidis et. al.[29] and Gupta et. al.[30] into blockchain and smart contracts for its applicability in IoT. According to Christidis et. al.[29], "the blockchain-IoT combination is powerful and can cause significant transformations across several industries, paving the way for new business models and novel, distributed applications."

In another work, Ouaddah et al.[1] just presented the idea of using smart contract in FairAccess. But their scheme does not secure the resource itself. Another such access control framework is given by Pinno et. el.[6]. Their scheme is named as ControlChain. ControlChain presents the idea to use dedicated blockchains for different purposes. For example- rules blockchain, relationships blockchain, context blockchain and accountability blockchain. Rules blockchain stores the authorization rules defined by the resource owners. Relationship blockchain stores all the information related to user credentials and their relationships. Context blockchain keeps data collected from sensor, actuators or other IoT devices.

## 3.3 CP-ABE in IoT and Blockchain

CP-ABE has been proved feasible on smart devices and IoT devices. Ambrosin et. al.[42] developed a C-based ABE library for android operating system. In their work, they have shown the feasibility of using CP-ABE and KP-ABE on smart devices. Sethia et. al.[41] presesnted a CP-ABE based revocation scheme for IoT devices. In another work, Sethia et. al.[43] have developed a CP-ABE based secure health sensor node using Raspberry Pi device. Touati et. al.[44] also used CP-ABE in IoT applications. Thus CP-ABE is feasible on IoT devices.

Jemel et. al.[46] used CP-ABE and blockchain to protect data residing on cloud systems. They used Multichain[17] and CP-ABE toolkit to create a timely-CP-ABE and blockchain based solution. In their scheme blockchain is used to verify legitimacy of users. Similarly, Yuan et. al.[47] used CP-ABE and blockchain to secure electronic documents. They used blockchain as an audit log and CP-ABE to encrypt data. Jemel et. al.[46] and Yuan et. al.[47] used CP-ABE over blockchain for cloud based memory and resource enriched systems. In our scheme, we are using CP-ABE over blockchain for IoT devices which are resource constrained.

# Chapter 4

# Proposed Work

In this chapter, firstly the architecture and work-flow of the idea given by Ouaddah et. al.[1] is explained, then our proposed architecture with CP-ABE is explained.

## 4.1 FairAccess

FairAccess 1.0[5] is based on Bitcoin framework. In FairAccess 2.0[1], authors have demonstrated the feasibility of using smart contracts. Both FairAccess versions are based on OM-AM(Objectives, Models, Architectures, and Mechanisms) framework[39]. Objective and model layers depict the required properties and architecture, and mechanism layers depict the methods to be used in achieving those properties.

### 4.1.1 OM-AM Framework

As its name suggests, it has four layers. Each layer is explained below in the context of FairAccess[1][5]. Figure-4.1 shows different layers of the OM-AM framework.

1. **Objective:** The main objectives are- transparency, user-driven, flexibility, anonymity, unlinkability, pseudonymity, revocation, scalability and IoT privacy and security. Access control model should be chosen while keeping in mind these objectives.

2. **Model:** FairAccess does not use any specific access control model. In FairAccess 1.0[5], OrBAC model is used.

3. **Architecture:** In FairAccess framework, its architecture has different entities- resource owner, requester entity, IoT resources and AMP(authorization manager point).

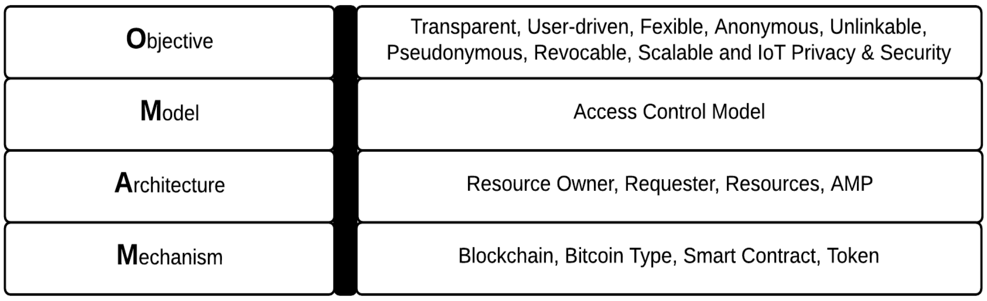
4. **Mechanism:** FairAccess 1.0[5] uses bitcoin blockchain, multisig scripts and tokens to build the access control system. FairAccess 2.0[1] uses second generation of blockchain which is smart contracts.

### 4.1.2 Entities in FairAccess

It has following entities:

- **User:** It can be either a resource owner or a requester requesting for IoT resource.

- **FairAccess Node:** It is responsible for carrying out all verification work,e.g., verifying a transaction. It can be a miner node.



| | |
|---|---|
| Objective | Transparent, User-driven, Fexible, Anonymous, Unlinkable, Pseudonymous, Revocable, Scalable and IoT Privacy & Security |
| Model | Access Control Model |
| Architecture | Resource Owner, Requester, Resources, AMP |
| Mechanism | Blockchain, Bitcoin Type, Smart Contract, Token |

Figure 4.1: OM-AM Framework

- **Wallet:** A wallet stores the user credentials. In the FairAccess framework, a wallet is playing the role of AMP(authorization manager point). A web application can be considered as a wallet. Resource owner submits his access policy to the wallet, it converts them into transactions and then it is sent to be deployed on the network.

- **Address:** Address is the hash of public key used by resource owner or requester. A user can create his/her address. The public-private key pair is generated by ECDSA(Elliptic Curve Digital Signature Algorithm).

- **Transaction:** Interaction between all users and the FairAccess node is done through transactions using addresses. FairAccess has following types of transactions:
  1. **GrantAccess Transaction:** This transaction is done by resource owner to deploy the smart contract. The smart contract contains the access policy for the resource. The source address is resource owner's address.
  2. **RequestAccess Transaction:** This transaction is done by the requester to communicate with the smart contract.
  3. **GetAccess Transaction:** This transaction is done by the requester to use its token to access the IoT device or resource.

- **Smart Contract:** A smart contract is created to carry out the access control functionalities. FairAccess creates an SC named PolicyContract. It resides on blockchain with a unique address. Requester uses this address to call PolicyContract.

- **Authorization Token:** A token is created by resource owner for requester if requester fulfills the access policy requirements of the IoT resource. Requester maintains an authorization token list(AuthTknList).

- **Block & Blockchain:** A blockchain consists of blocks. FairAccess assumes blockchain as a database to store its smart contract and transactions within a block.

### 4.1.3 Work-flow of FairAccess

FairAccess performs its process in the following manner:

1. **Initialization:** Access policies are defined by resource owners to be further converted into contracts and to get a unique address for each resource. It is assumed that requester knows the access policy of each resource.

2. **Grant Access:** In this step, PolicyContract is deployed on the blockchain via a grant access transaction.

3. **Request Access:** In this step, requester submits a request to access a particular resource device. The device then gives the address of the contract for that particular resource. Requester evaluates its response and submits a transaction to access the resource.

4. **Executing Contract:** PolicyContract is executed to check if requester fulfills the policy or not. And according to the positive output, token creation is done by calling CreateTokenContract. CreateTokenContract will create a token and token will be added to requester's token list. If the response of policy check is negative, request to access resource is denied.

5. **Get Access:** Once the token is created, the requester can request to get access by submitting its token.

6. **Token Validation:** Submitted token in step5 is validated, and access to the resource is given according to the output of validation check.

## 4.2 Proposed Architecture

### 4.2.1 OM-AM Framework

The proposed architecture follows the same OM-AM framework as explained in section-4.1.1. All objectives are the same except for a newly added objective - fine-grained access control and data security. The second layer after the objective is to use access control model. The architecture is made from service end(SE), requester end(RE) and miners. The mechanism used is Ethereum blockchain, smart contract, CP-ABE, and tokens. Table-4.1 shows the OM-AM framework of the proposed architecture.

Table 4.1: **O**bjective-**M**odel-**A**rchitecture-**M**echanism

| | |
|---|---|
| **Objective** | Fine Grained Access, Transparent, User-driven, Fexible, Anonymous, Unlinkable, Pseudonymous, Revocable, Scalable and IoT Data Security & Privacy |
| **Model** | Access Control Model |
| **Architecture** | Service End(SE), Requester End(RE) and Miners |
| **Mechanism** | Ethereum Blockchain, Smart Contract, CP-ABE and Tokens |

### 4.2.2 Entities of Proposed Architecture

- **Service End(SE):** It is the IoT resource end. It is in the form of data provided by Raspberry Pi2 device. The IoT device collects data from sensors and actuators. Data is encrypted using CP-ABE. SE generates the user secret key using CP-ABE and provides to its requesters. SE is a lightweight Ethereum node.
- **Requester End(RE):** This end wants to access IoT data. It can be both, either lightweight node or full node and participate in mining.
- **Miner Node:** It is responsible for carrying out all verification work,e.g., verifying a transaction. This is an Ethereum full node. Every transaction submitted either by SE or RE is mined by miners on the node.
- **Wallet:** A wallet stores the user credentials. Service end submits his access policy to the wallet, it converts them into transactions and then it is sent to be deployed on the network. Requester end can submit transactions to request or verify token through his/her wallet.
- **Address:** Address is the hash of public key used by SE or RE. A user can create his/her address. In Ethereum blockchain public-private key pair is generated by ECDSA(Elliptic

Curve Digital Signature Algorithm). Wallets and smart contracts have their unique addresses. Address of an Ethereum smart contract or wallet is of 20 bytes. Communication between nodes is done using these addresses.

- **Transaction:** Interaction among SE, RE or miners is done through transactions using addresses. The proposed method has following types of transactions:
  1. **DeploySC Transaction:** This transaction is done by SE to deploy the smart contract. The smart contract contains the access policy of the SE. The source address is SE's address, and the target address is left null.
  2. **CheckAccess Transaction:** This transaction is done by the requester to communicate with the smart contract. In this transaction, RE submits his/her attribute list to SE requesting to access an IoT resource.
  3. **AccessCheckOutput Transaction:** This transaction is done by SE to notify RE that his/her attributes have been checked against access policy. RE receives the response message.
  4. **CreateToken Transaction:** If the response of CheckAccess is positive, SE submits the CreateToken transaction to create a token.
  5. **RequestDevice Transaction:** This transaction is done by RE to access IoT device.
  6. **VerifyToken Transaction:** In response to RequestDevice transaction, SE submits VerifyToken transaction to check the validity of token. If the token is valid, SE gives the data encrypted with CP-ABE.
  7. **RevokeAccess Transaction:** This transaction is done by SE to revoke access from the requester. Access revocation is done in three cases, when:
     (a) SE discovers misbehavior/maliciousness of RE.
     (b) Token is invalid.
     (c) RE has used its token.

- **Smart Contract:** A smart contract is created to carry out the access control functionalities. There are two contracts in proposed method, first PolicyContract and second AuthList contract. Both reside on blockchain with a unique address. RE uses the addresses to call contracts.

- **Token:** A token is created by SE for RE if RE fulfills the access policy requirements of the IoT resource. RE maintains an authorization token list(AuthTknList) in the form of a smart contract. In the proposed method, the token has five fields. Table-4.2 shows token fields with description.

- **Ethereum Blockchain:** A blockchain consists of blocks. It stores smart contracts and transactions within a block.

### 4.2.3 Work-Flow

Proposed access control architecture executes in following manner:

1. **SC Deployment:** Access policies are defined by SE to be further converted into contracts and to get a unique address for each resource. It is assumed that requester knows the access

Table 4.2: Structure of a Token

| ***S.No.*** | ***Token Field*** | ***Description*** |
|---|---|---|
| 1. | Address | It holds the address of RE. |
| 2. | DeviceId | It holds the identity number of resource device. |
| 3. | TimeStamp | It holds the time-value of token creation. |
| 4. | Validity | It holds the time-period upto which token is valid. |
| 5. | Resource | It holds the resource name. |

policy of each resource. In this step, SE submits a transaction to deploy the contract named PolicyContract. RE deploys the AuthList contract. For every resource/service and requester, a contract is deployed by the respective entity.

2. **Request Access:** In this step RE submits a CheckAccess transaction with its attribute list as an argument. CheckAccess transaction triggers an event on SE side. SE side will perform policy check operation using CP-ABE's function *cpabe-pcheck*. The response of policy check will be submitted in the form of AccessCheckOutput transaction. This transaction triggers an event on RE side to inform that policy has been checked. If the response is positive, SE will create RE's secret key using CP-ABE's function *cpabe-keygen* and adds this key to AuthList SC.

3. **Token Creation:** If the response of step2 is positive, SE will submit the CreatToken transaction. This transaction will create token and add it to RE's AuthList contract. Thus RE will receive its token and secret key for requested resource in its AutList contract.

4. **Request Device:** In this step, RE submits a request to access the resource by presenting its token. This request triggers an event on SE side. SE submits VerifyToken transaction and sends the response to RE. If the response is negative, SE deletes the token from PolicyContract and AuthList and sends a negative response. If the response is positive, RE further submits RequestDevice transaction to access the resource.

5. **Grant Access:** SE provides the IoT resources encrypted with CP-ABE. It uses CP-ABE's function *cpabe-enc* to encrypt the resource. RE side decrypts the resource by calling CP-ABE's function *cpabe-dec*.

6. **Delete Token:** After giving access to the resource, SE submits RevokeAccess transaction to delete the token from its list as well as AuthList.

### 4.2.4 Block Diagram of Architecture

A block diagram of architecture is shown in figure-4.2. It represents two parties in architecture, first the IoT-resource side, in which multiple resources with their access policies as AP and resource names as RS are secured using CP-ABE. Second party contains different requesters as RQ with their attribute lists. Both requesters and resource-ends are communicating via Ethereum blockchain which consists of smart contracts and transactions.

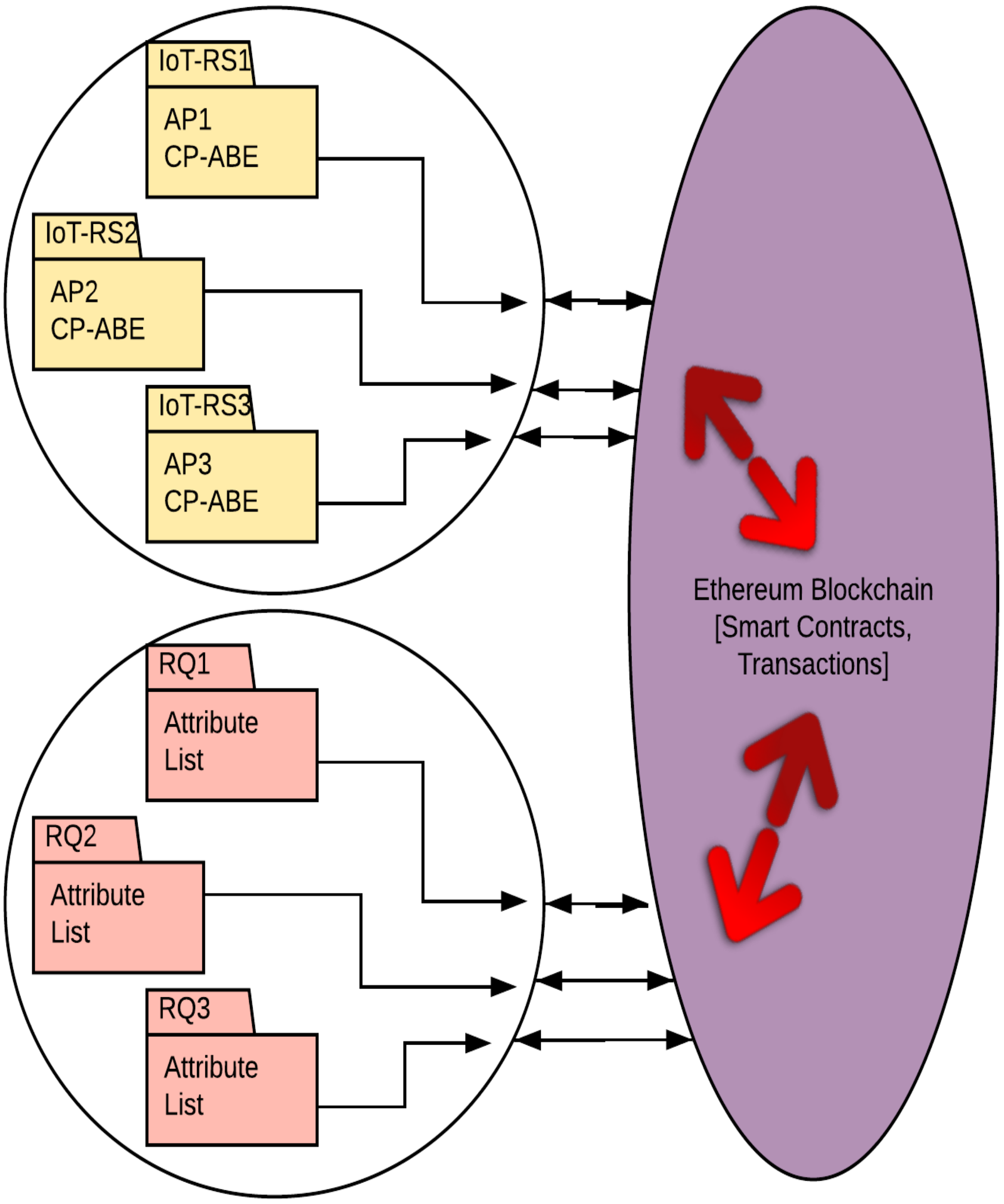


Figure 4.2: Block Diagram of Architecture[RS(Resource), RQ(Requester)]

### 4.2.5 Flow-Chart

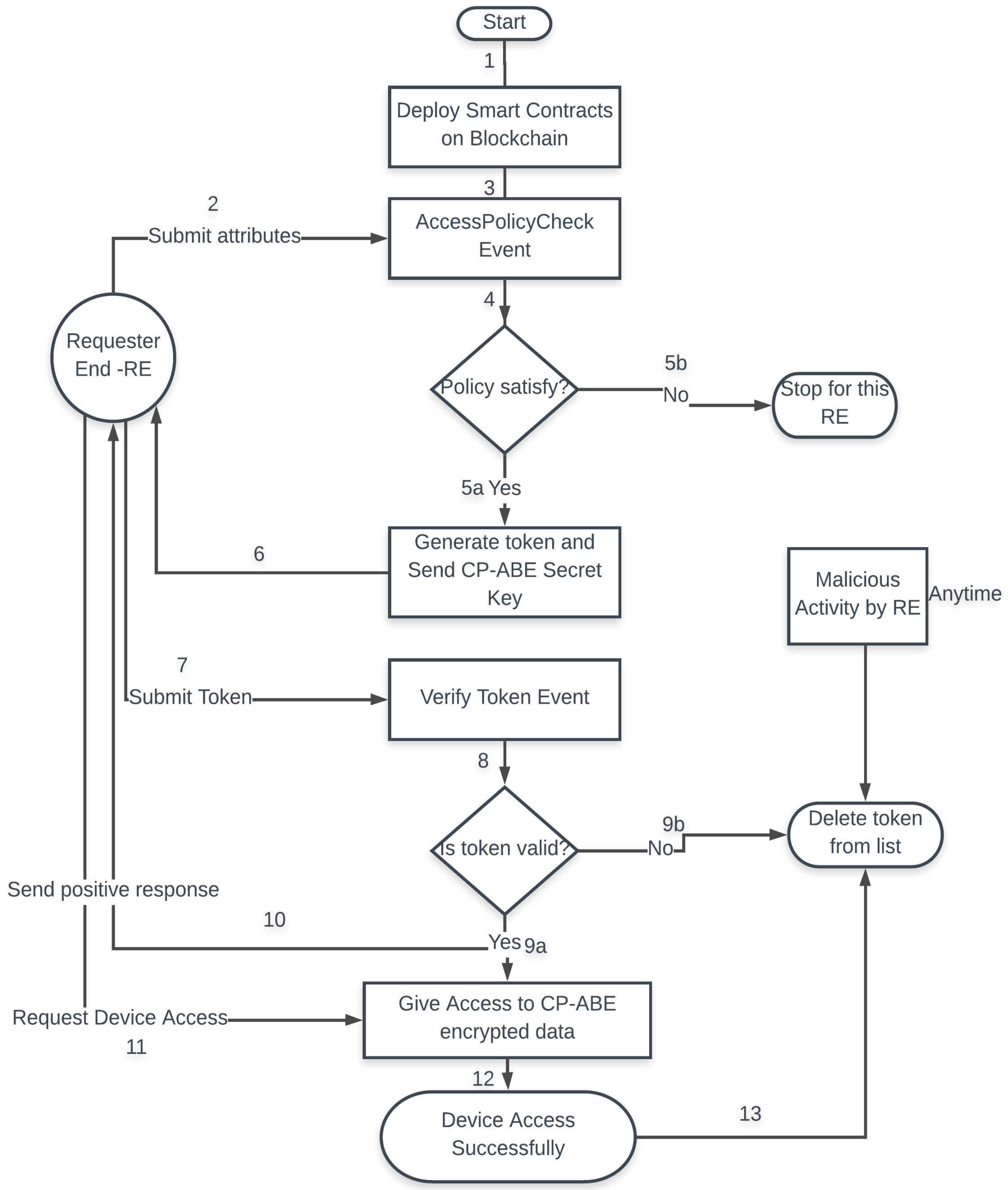


Figure 4.3: Flow-Chart

### 4.2.6 Sequence-Diagram

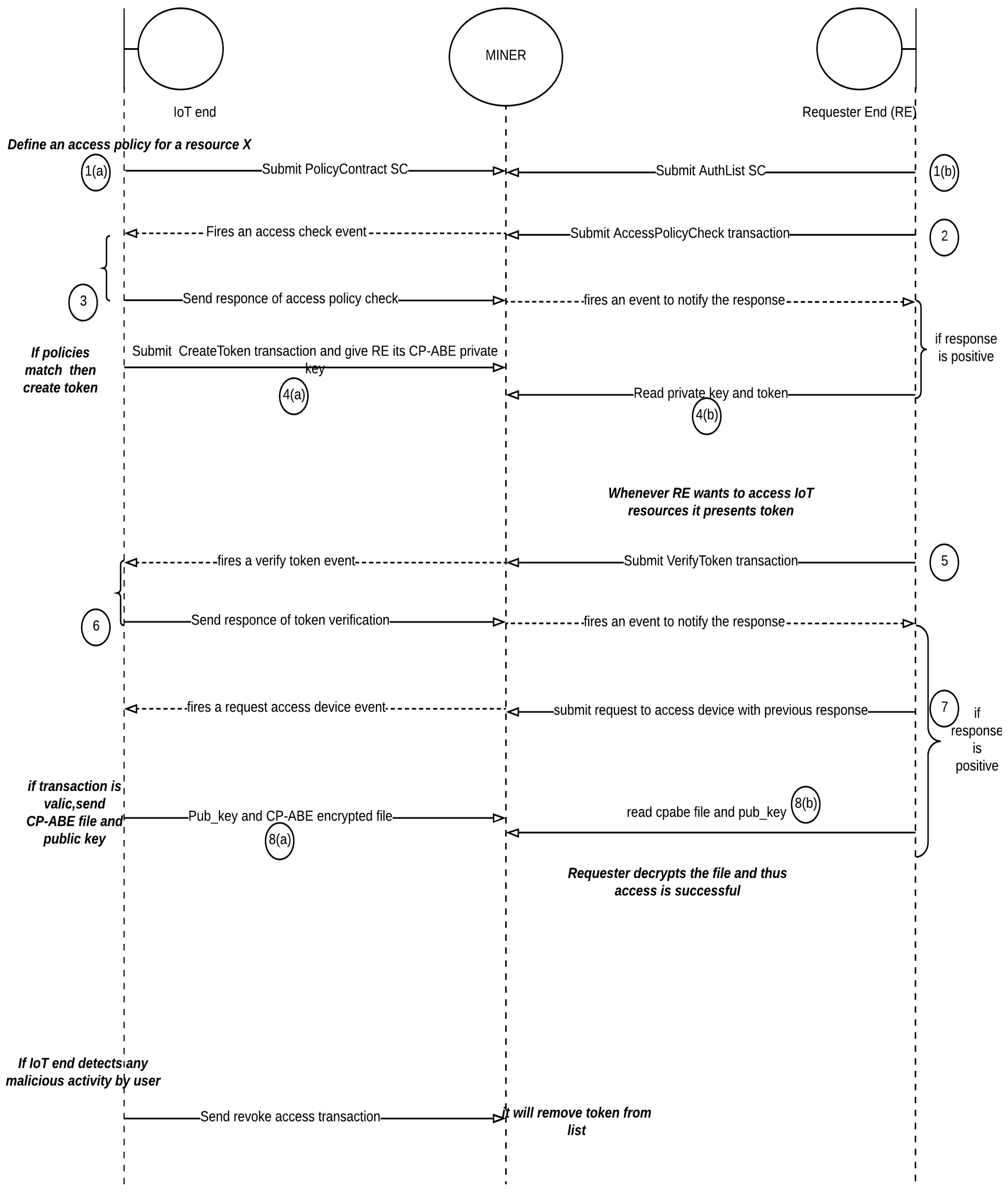

MINER
IoT end
Requester End (RE)
Define an access policy for a resource X
1(a)
Submit PolicyContract SC
Submit AuthList SC
1(b)
Fires an access check event
Submit AccessPolicyCheck transaction
2
3
Send responce of access policy check
fires an event to notify the response
if response is positive
If policies match then create token
Submit CreateToken transaction and give RE its CP-ABE private key
4(a)
Read private key and token
4(b)
Whenever RE wants to access IoT resources it presents token
fires a verify token event
Submit VerifyToken transaction
5
6
Send responce of token verification
fires an event to notify the response
fires a request access device event
submit request to access device with previous response
7
if response is positive
if transaction is valic,send CP-ABE file and public key
Pub_key and CP-ABE encrypted file
8(a)
read cpabe file and pub_key
8(b)
Requester decrypts the file and thus access is successful
If IoT end detects any malicious activity by user
Send revoke access transaction
It will remove token from list

Figure 4.4: Sequence-Diagram

# Chapter 5

# Detailed Implementation

## 5.1 Tools and Technology Used

### 5.1.1 System Setup

Experiental setup of our implementation is given as follow:

1. Raspberry Pi2 Device(SE):
   (a) OS - Linux Raspbian
   (b) Memory - 1 GB
   (c) CPU - ARMv7
2. Dell PC 1(For mining/RE):
   (a) OS - Linux 16.04
   (b) Memory - 4 GB
   (c) CPU - Intel Core i5
3. Dell PC 2(As a monitor screen for Raspberry Pi2 Desktop)
4. Software Used:
   (a) Ethereum private network
   (b) Solidity to write Ethereum smart contracts.
   (c) NodeJs and Web3 to interact with Ethereum blockchain.
   (d) Geth - a CLI(command line interface) to run an Ethereum full node
   (e) CP-ABE toolkit

### 5.1.2 Physical Layout

Figure-5.1 shows the layout of the implementation. Raspberry Pi2 device is connected to a monitor screen(Dell PC 2) using a HDMI-to-VGA cable.

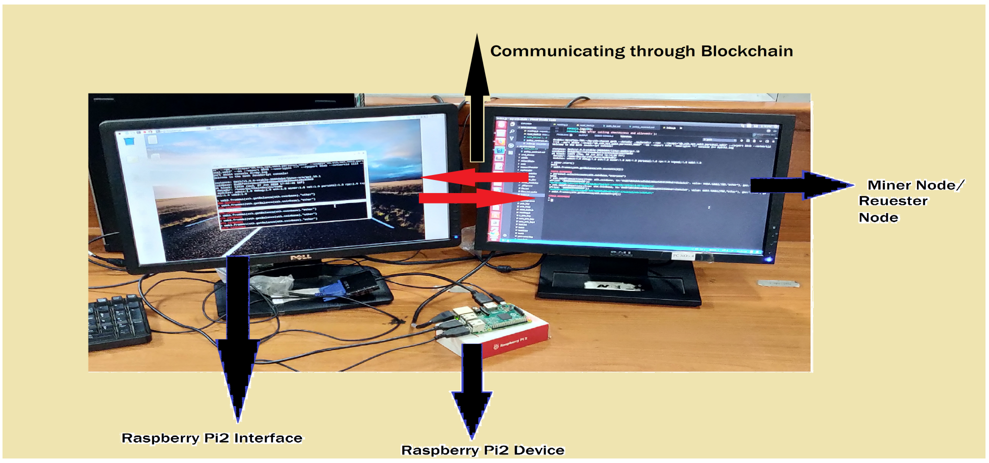
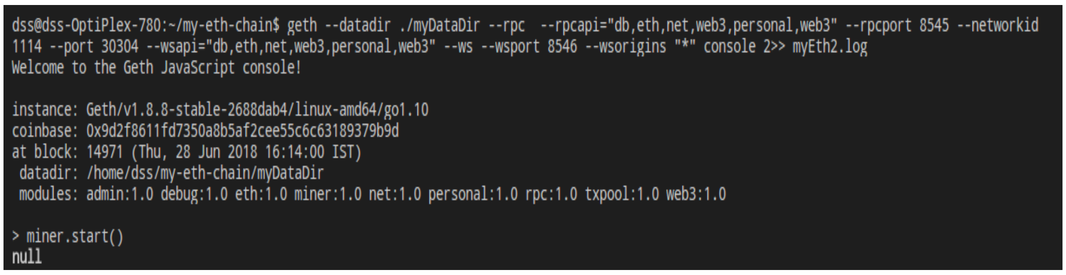


Figure 5.1: Physical Layout

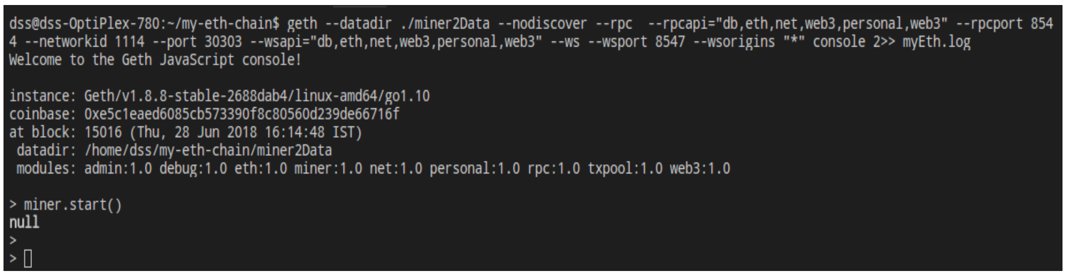
## 5.2 Setting-up Ethereum Blockchain

For Ethereum blockchain, an Ethereum private network is created on a Dell PC. After creating a genesis.JSON file, data directory is instantiated. Once data directory is created, start an Ethereum peer node. Figure-5.2 and figure-5.3 shows two full nodes running as miners. Figure-5.4 shows that mining is being done on both miners. A client on IoT device will be created with the same genesis.JSON file, but this end will not mine. Geth client will run on Raspberry Pi2 device only as a lightweight node. Figure-5.5 shows geth client running on Raspberry Pi2 device.



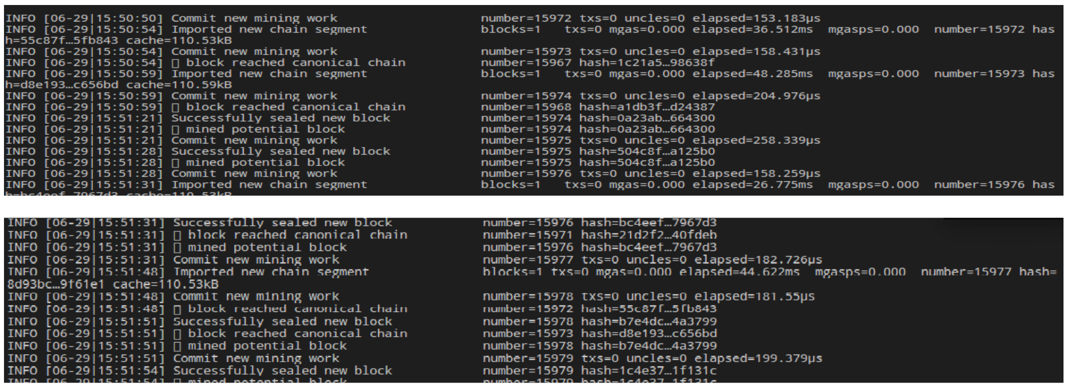
```
dss@dss-OptiPlex-780:~/my-eth-chain$ geth --datadir ./myDataDir --rpc  --rpcapi="db,eth,net,web3,personal,web3" --rpcport 8545 --networkid
1114 --port 30304 --wsapi="db,eth,net,web3,personal,web3" --ws --wsport 8546 --wsorigins "*" console 2>> myEth2.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.10
coinbase: 0x9d2f8611fd7350a8b5af2cee55c6c63189379b9d
at block: 14971 (Thu, 28 Jun 2018 16:14:00 IST)
 datadir: /home/dss/my-eth-chain/myDataDir
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> miner.start()
null
```

Figure 5.2: Geth client-Miner1



```
dss@dss-OptiPlex-780:~/my-eth-chain$ geth --datadir ./miner2Data --nodiscover --rpc  --rpcapi="db,eth,net,web3,personal,web3" --rpcport 854
4 --networkid 1114 --port 30303 --wsapi="db,eth,net,web3,personal,web3" --ws --wsport 8547 --wsorigins "*" console 2>> myEth.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.8-stable-2688dab4/linux-amd64/go1.10
coinbase: 0xe5c1eaed6085cb573390f8c80560d239de66716f
at block: 15016 (Thu, 28 Jun 2018 16:14:48 IST)
 datadir: /home/dss/my-eth-chain/miner2Data
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> miner.start()
null
>
> ▯
```

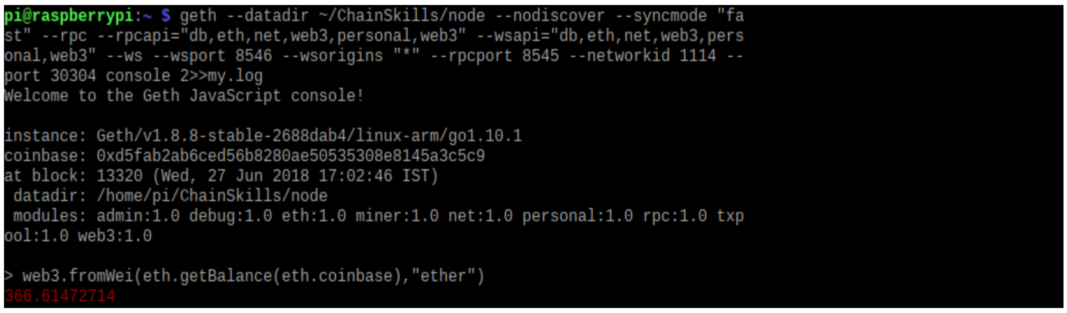
Figure 5.3: Geth client-Miner2



```
INFO [06-29|15:50:50] Commit new mining work                number=15972 txs=0 uncles=0 elapsed=153.183µs
INFO [06-29|15:50:54] Imported new chain segment            blocks=1   txs=0 mgas=0.000 elapsed=36.512ms  mgasps=0.000  number=15972 has
h=55c87f…5fb843 cache=110.53kB
INFO [06-29|15:50:54] Commit new mining work                number=15973 txs=0 uncles=0 elapsed=158.431µs
INFO [06-29|15:50:54] 🔗 block reached canonical chain      number=15967 hash=1c21a5…98638f
INFO [06-29|15:50:59] Imported new chain segment            blocks=1   txs=0 mgas=0.000 elapsed=48.285ms  mgasps=0.000  number=15973 has
h=d8e193…c656bd cache=110.59kB
INFO [06-29|15:50:59] Commit new mining work                number=15974 txs=0 uncles=0 elapsed=204.976µs
INFO [06-29|15:50:59] 🔗 block reached canonical chain      number=15968 hash=a1db3f…d24387
INFO [06-29|15:51:21] Successfully sealed new block         number=15974 hash=0a23ab…664300
INFO [06-29|15:51:21] 🔨 mined potential block              number=15974 hash=0a23ab…664300
INFO [06-29|15:51:21] Commit new mining work                number=15975 txs=0 uncles=0 elapsed=258.339µs
INFO [06-29|15:51:28] Successfully sealed new block         number=15975 hash=504c8f…a125b0
INFO [06-29|15:51:28] 🔨 mined potential block              number=15975 hash=504c8f…a125b0
INFO [06-29|15:51:28] Commit new mining work                number=15976 txs=0 uncles=0 elapsed=158.259µs
INFO [06-29|15:51:31] Imported new chain segment            blocks=1   txs=0 mgas=0.000 elapsed=26.775ms  mgasps=0.000  number=15976 has
```

```
INFO [06-29|15:51:31] Successfully sealed new block         number=15976 hash=bc4eef…7967d3
INFO [06-29|15:51:31] 🔗 block reached canonical chain      number=15971 hash=21d2f2…40fdeb
INFO [06-29|15:51:31] 🔨 mined potential block              number=15976 hash=bc4eef…7967d3
INFO [06-29|15:51:31] Commit new mining work                number=15977 txs=0 uncles=0 elapsed=182.726µs
INFO [06-29|15:51:48] Imported new chain segment            blocks=1 txs=0 mgas=0.000 elapsed=44.622ms  mgasps=0.000  number=15977 hash=
8d93bc…9f61e1 cache=110.53kB
INFO [06-29|15:51:48] Commit new mining work                number=15978 txs=0 uncles=0 elapsed=181.55µs
INFO [06-29|15:51:48] 🔗 block reached canonical chain      number=15972 hash=55c87f…5fb843
INFO [06-29|15:51:51] Successfully sealed new block         number=15978 hash=b7e4dc…4a3799
INFO [06-29|15:51:51] 🔗 block reached canonical chain      number=15973 hash=d8e193…c656bd
INFO [06-29|15:51:51] 🔨 mined potential block              number=15978 hash=b7e4dc…4a3799
INFO [06-29|15:51:51] Commit new mining work                number=15979 txs=0 uncles=0 elapsed=199.379µs
INFO [06-29|15:51:54] Successfully sealed new block         number=15979 hash=1c4e37…1f131c
```

Figure 5.4: Mining log of Miner1 and Miner2

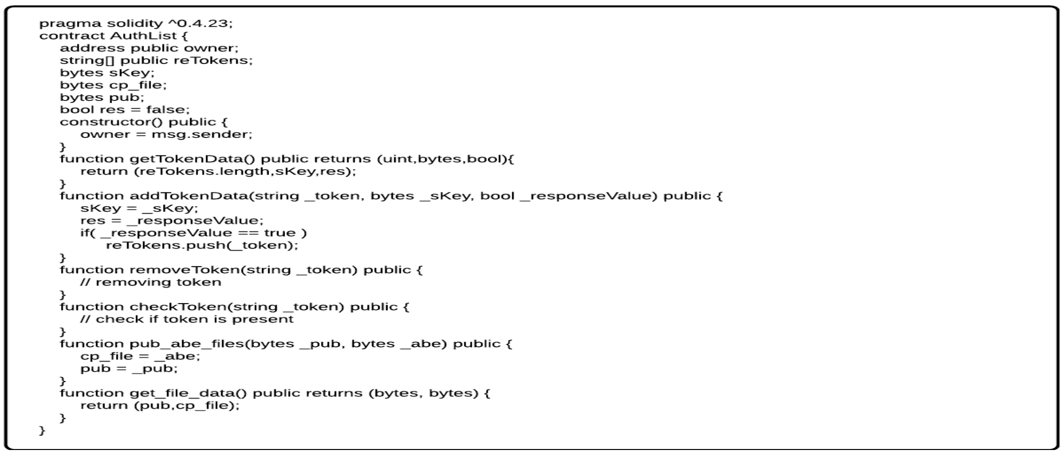


```
pi@raspberrypi:~ $ geth --datadir ~/ChainSkills/node --nodiscover --syncmode "fa
st" --rpc --rpcapi="db,eth,net,web3,personal,web3" --wsapi="db,eth,net,web3,pers
onal,web3" --ws --wsport 8546 --wsorigins "*" --rpcport 8545 --networkid 1114 --
port 30304 console 2>>my.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.8-stable-2688dab4/linux-arm/go1.10.1
coinbase: 0xd5fab2ab6ced56b8280ae50535308e8145a3c5c9
at block: 13320 (Wed, 27 Jun 2018 17:02:46 IST)
 datadir: /home/pi/ChainSkills/node
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txp
ool:1.0 web3:1.0

> web3.fromWei(eth.getBalance(eth.coinbase),"ether")
366.61472714
```

Figure 5.5: Geth client-Raspberry Pi2

## 5.3 Contracts

An example of Solidity code for PolicyContract and AuthList is shown in figure-5.7. These code are just an abstract of contracts, it shows how functions are added in smart contract.



```
pragma solidity ^0.4.23;
contract AuthList {
    address public owner;
    string[] public reTokens;
    bytes sKey;
    bytes cp_file;
    bytes pub;
    bool res = false;
    constructor() public {
        owner = msg.sender;
    }
    function getTokenData() public returns (uint,bytes,bool){
        return (reTokens.length,sKey,res);
    }
    function addTokenData(string _token, bytes _sKey, bool _responseValue) public {
        sKey = _sKey;
        res = _responseValue;
        if( _responseValue == true )
            reTokens.push(_token);
    }
    function removeToken(string _token) public {
        // removing token
    }
    function checkToken(string _token) public {
        // check if token is present
    }
    function pub_abe_files(bytes _pub, bytes _abe) public {
        cp_file = _abe;
        pub = _pub;
    }
    function get_file_data() public returns (bytes, bytes) {
        return (pub,cp_file);
    }
}
```

Figure 5.6: Abstract of AuthList

```
pragma solidity ^0.4.23;
contract AuthList {
    function getTokenData() public returns (uint,bytes,bool) {}
    function addTokenData(string,bytes,bool) public  {}
    function removeToken(string) public  {}
    function checkToken(string) public {}
    function pub_abe_files(bytes,bytes) public {}
    function get_file_data() public returns (bytes, bytes) {}
}
contract PolicyContract {
    struct Token {
        address req;
        uint deviceId;
        uint timestamp;
        uint validity;
        string rs;
    }
    Token[] tokens;
    //mapping (address => Token) public tokens;
    //mapping (address => mapping (uint => Token)) public tokens;
    address public owner;
    uint devId;
    string devPolicy;
    bool getAccess_request = true;
    constructor(uint _devId, string _devPolicy) public {
        owner = msg.sender;
        //owner = _owner_address;
        devPolicy = _devPolicy;
        devId = _devId;
    }
    function getDevId() public view returns (uint,string) {
        return (devId,devPolicy);
    }
    function getTokenData() public view returns (uint, address, uint, uint, uint, string) {

        return (tokens.length, tokens[0].req ,tokens[0].deviceId, tokens[0].timestamp, tokens[0].validity,tokens[0].rs );
    }
    // ===== Properties used to check access =====
    event requestCheckAccess(string accessStructure, address requester);
    event accessChecked(bool granted, address requester);
    function checkAccess(string _acStruct) public  {
        //return _acStruct;
        requestCheckAccess(_acStruct, msg.sender);
    }
    function accessCheckOutput(bool _granted, address _requester) public returns (bool,address) {
        emit accessChecked(_granted, _requester);
        return (_granted,_requester);
    }
    // ===== Finished properties ====
    // ===== Properties used to request access =====
    event requestAccessToken(bytes accessStructure, address requester);
    function requestAccess(bytes acStruct) public {
        emit requestAccessToken(acStruct, msg.sender);
    }
    function createToken (string _file, address _requester, address _authList, bytes _secKey, bool _response) public {
        require(msg.sender == owner);
        // === added token-data to its structure

        tokens.push( Token( {
            req: _requester,
            deviceId: devId,
            timestamp: now,
            validity: 10,
            rs: _file
        } ) );

        //===create an instance of AuthList contract to add token
        AuthList authList = AuthList(_authList);
        authList.addTokenData(_file,_secKey,_response);
    }
    // ===== Finished Properties =====
```

Figure 5.7: Abstract of Policy Contract

# Chapter 6

# Experimental Results

Proposed access control method has been implemented successfully. In this chapter, computational overhead in different operations, generated blocks in different transactions and its usage example is discussed.

## 6.1 Computational Overhead

Table-6.1 shows the time taken in different events at SE side and table-6.2 shows the time taken at requester side. Time taken in these events includes the time of transaction submission and mining by miners.

Table 6.1: Time taken at SE side

| ***S.No.*** | ***Event Name*** | ***Time in seconds*** |
|---|---|---|
| 1 | Access Check Event | 0.065 |
| 2 | Token Create Event | 35.512 |
| 3 | Token Verification Event | 35.437 |

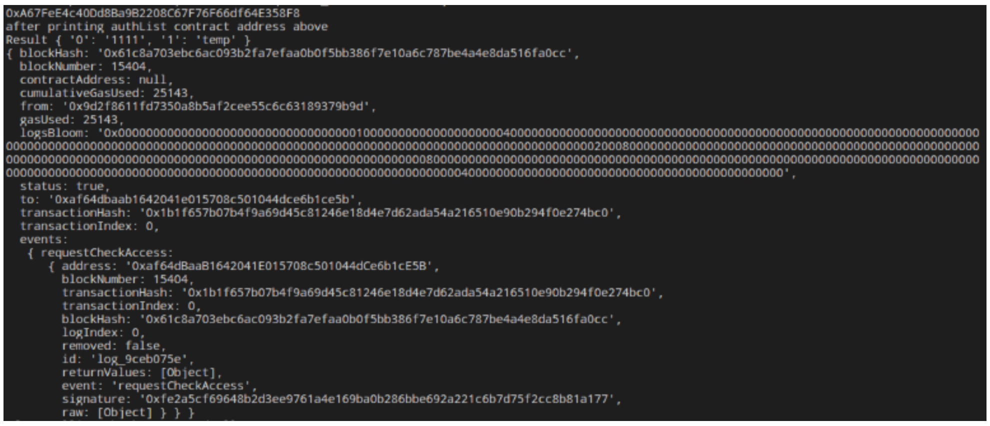
Table 6.2: Time taken at RE side

| ***S.No.*** | ***Event Name*** | ***Time in seconds*** |
|---|---|---|
| 1 | Access Policy Check Event | 23.084 |
| 2 | Request Device Event | 17.422 |
| 3 | Device Access Event | 0.022 |

## 6.2 Different Transactions

### 6.2.1 At Requester Side

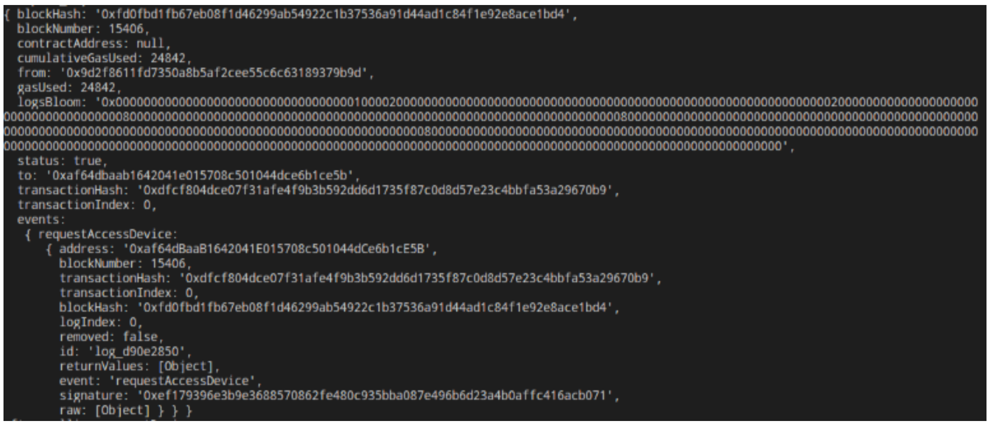
Below figures show blocks of different transactions done in the whole process. Figure-6.1 shows the CheckAccess transaction block. It shows that event requestCheckAccess is triggered at SE side. In this block, the field ’to’ represents the address of smart contract.



```
0xA67FeE4c40Dd8Ba9B2208C67F76F66df64E358F8
after printing authList contract address above
Result { '0': '1111', '1': 'temp' }
{ blockHash: '0x61c8a703ebc6ac093b2fa7efaa0b0f5bb386f7e10a6c787be4a4e8da516fa0cc',
  blockNumber: 15404,
  contractAddress: null,
  cumulativeGasUsed: 25143,
  from: '0x9d2f8611fd7350a8b5af2cee55c6c63189379b9d',
  gasUsed: 25143,
  logsBloom: '0x00000000000000000000000000000000000100000000000000000004000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000020080000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000',
  status: true,
  to: '0xaf64dbaab1642041e015708c501044dce6b1ce5b',
  transactionHash: '0x1b1f657b07b4f9a69d45c81246e18d4e7d62ada54a216510e90b294f0e274bc0',
  transactionIndex: 0,
  events:
   { requestCheckAccess:
      { address: '0xaf64dBaaB1642041E015708c501044dCe6b1cE5B',
        blockNumber: 15404,
        transactionHash: '0x1b1f657b07b4f9a69d45c81246e18d4e7d62ada54a216510e90b294f0e274bc0',
        transactionIndex: 0,
        blockHash: '0x61c8a703ebc6ac093b2fa7efaa0b0f5bb386f7e10a6c787be4a4e8da516fa0cc',
        logIndex: 0,
        removed: false,
        id: 'log_9ceb075e',
        returnValues: [Object],
        event: 'requestCheckAccess',
        signature: '0xfe2a5cf69648b2d3ee9761a4e169ba0b286bbe692a221c6b7d75f2cc8b81a177',
        raw: [Object] } } }
```

Figure 6.1: CheckAccess Block

Figure-6.2 shows the request device transaction block. In this block, requestAccessDevice event is triggered at SE side.

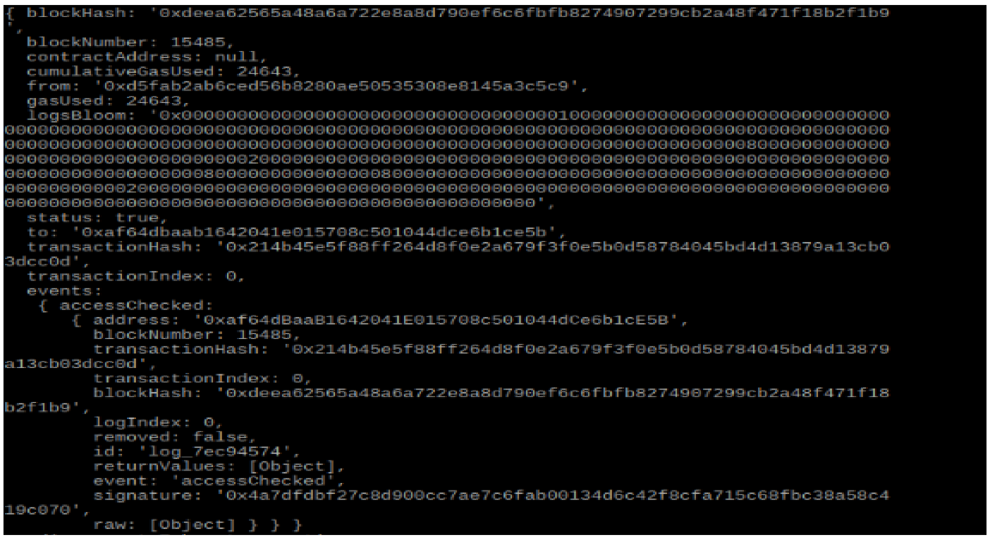


```
{ blockHash: '0xfd0fbd1fb67eb08f1d46299ab54922c1b37536a91d44ad1c84f1e92e8ace1bd4',
  blockNumber: 15406,
  contractAddress: null,
  cumulativeGasUsed: 24842,
  from: '0x9d2f8611fd7350a8b5af2cee55c6c63189379b9d',
  gasUsed: 24842,
  logsBloom: '0x00000000000000000000000000000000000100020000000000000000000000000000000000000000000000000000000000000000200000000000000000000
0000000000000000008000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000008000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
  status: true,
  to: '0xaf64dbaab1642041e015708c501044dce6b1ce5b',
  transactionHash: '0xdfcf804dce07f31afe4f9b3b592dd6d1735f87c0d8d57e23c4bbfa53a29670b9',
  transactionIndex: 0,
  events:
   { requestAccessDevice:
      { address: '0xaf64dBaaB1642041E015708c501044dCe6b1cE5B',
        blockNumber: 15406,
        transactionHash: '0xdfcf804dce07f31afe4f9b3b592dd6d1735f87c0d8d57e23c4bbfa53a29670b9',
        transactionIndex: 0,
        blockHash: '0xfd0fbd1fb67eb08f1d46299ab54922c1b37536a91d44ad1c84f1e92e8ace1bd4',
        logIndex: 0,
        removed: false,
        id: 'log_d90e2850',
        returnValues: [Object],
        event: 'requestAccessDevice',
        signature: '0xef179396e3b9e3688570862fe480c935bba087e496b6d23a4b0affc416acb071',
        raw: [Object] } } }
```

Figure 6.2: Request Device Block

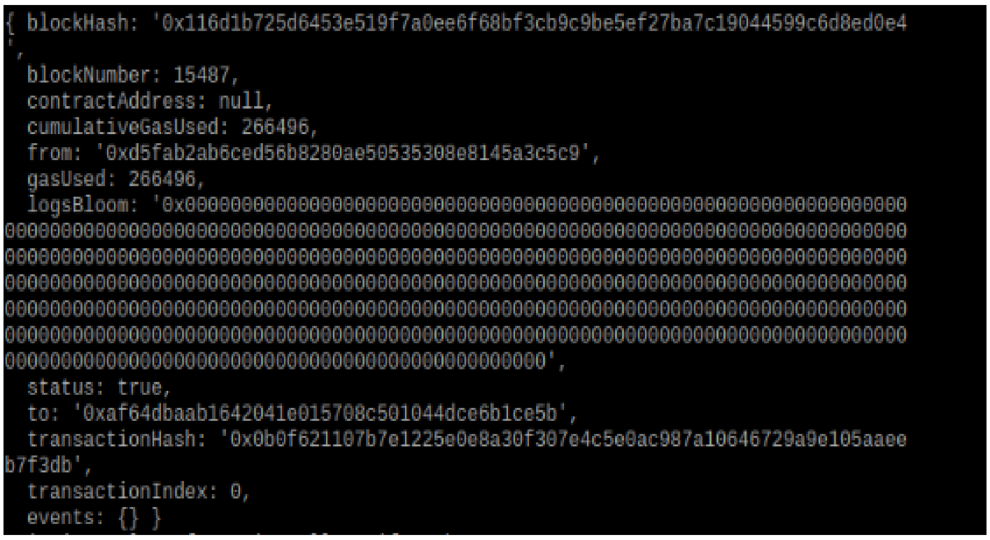
### 6.2.2 At IoT Side

Figure-6.3 shows policy check block at SE side. It triggeres an event accessChecked on RE side. Figure-6.4 presents the token creation block. Figure-6.5 presents the verify token block.



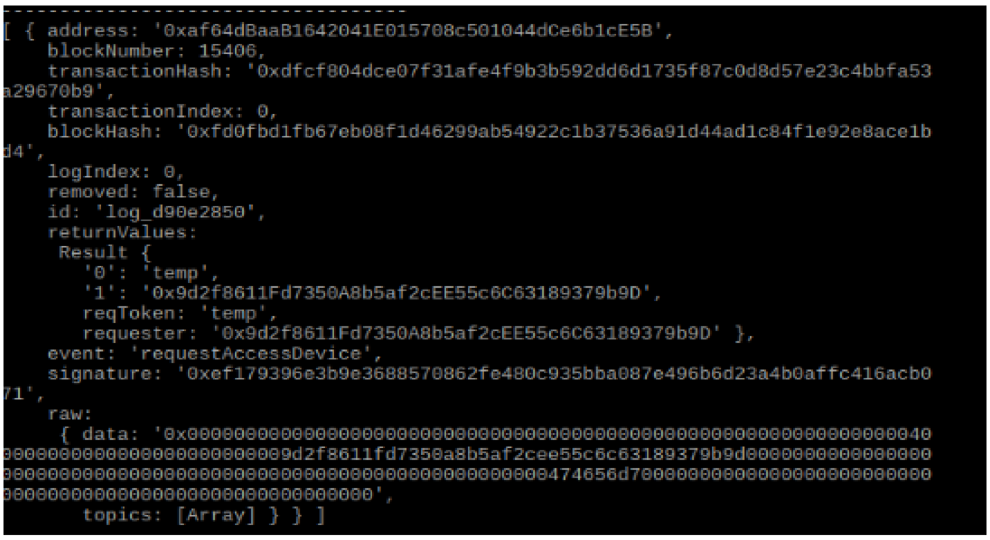
```
{ blockHash: '0xdeea62565a48a6a722e8a8d790ef6c6fbfb8274907299cb2a48f471f18b2f1b9
',
  blockNumber: 15485,
  contractAddress: null,
  cumulativeGasUsed: 24643,
  from: '0xd5fab2ab6ced56b8280ae50535308e8145a3c5c9',
  gasUsed: 24643,
  logsBloom: '0x0000000000000000000000000000000000100000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000800000000
00000000000000000000002000000000000000000000000000000000000000000000000000000000
00000000000000000800000000000000008000000000000000000000000000000000000000000000
00000000000200000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000',
  status: true,
  to: '0xaf64dbaab1642041e015708c501044dce6b1ce5b',
  transactionHash: '0x214b45e5f88ff264d8f0e2a679f3f0e5b0d58784045bd4d13879a13cb0
3dcc0d',
  transactionIndex: 0,
  events:
   { accessChecked:
      { address: '0xaf64dBaaB1642041E015708c501044dCe6b1cE5B',
        blockNumber: 15485,
        transactionHash: '0x214b45e5f88ff264d8f0e2a679f3f0e5b0d58784045bd4d13879
a13cb03dcc0d',
        transactionIndex: 0,
        blockHash: '0xdeea62565a48a6a722e8a8d790ef6c6fbfb8274907299cb2a48f471f18
b2f1b9',
        logIndex: 0,
        removed: false,
        id: 'log_7ec94574',
        returnValues: [Object],
        event: 'accessChecked',
        signature: '0x4a7dfdbf27c8d900cc7ae7c6fab00134d6c42f8cfa715c68fbc38a58c4
19c070',
        raw: [Object] } } }
```

Figure 6.3: Policy Check Block



```
{ blockHash: '0x116d1b725d6453e519f7a0ee6f68bf3cb9c9be5ef27ba7c19044599c6d8ed0e4
',
  blockNumber: 15487,
  contractAddress: null,
  cumulativeGasUsed: 266496,
  from: '0xd5fab2ab6ced56b8280ae50535308e8145a3c5c9',
  gasUsed: 266496,
  logsBloom: '0x0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000',
  status: true,
  to: '0xaf64dbaab1642041e015708c501044dce6b1ce5b',
  transactionHash: '0x0b0f621107b7e1225e0e8a30f307e4c5e0ac987a10646729a9e105aaee
b7f3db',
  transactionIndex: 0,
  events: {} }
```

Figure 6.4: Token Creation Block



```
----------------------------------
[ { address: '0xaf64dBaaB1642041E015708c501044dCe6b1cE5B',
    blockNumber: 15406,
    transactionHash: '0xdfcf804dce07f31afe4f9b3b592dd6d1735f87c0d8d57e23c4bbfa53
a29670b9',
    transactionIndex: 0,
    blockHash: '0xfd0fbd1fb67eb08f1d46299ab54922c1b37536a91d44ad1c84f1e92e8ace1b
d4',
    logIndex: 0,
    removed: false,
    id: 'log_d90e2850',
    returnValues:
     Result {
       '0': 'temp',
       '1': '0x9d2f8611Fd7350A8b5af2cEE55c6C63189379b9D',
       reqToken: 'temp',
       requester: '0x9d2f8611Fd7350A8b5af2cEE55c6C63189379b9D' },
    event: 'requestAccessDevice',
    signature: '0xef179396e3b9e3688570862fe480c935bba087e496b6d23a4b0affc416acb0
71',
    raw:
     { data: '0x0000000000000000000000000000000000000000000000000000000000000040
0000000000000000000000009d2f8611fd7350a8b5af2cee55c6c63189379b9d000000000000000
00000000000000000000000000000000000000000000000474656d700000000000000000000000
0000000000000000000000000000000000',
       topics: [Array] } } ]
```

Figure 6.5: Verify Token Block

## 6.3 Case Study

The proposed access control system can be applied in cases where there are different entities want to access some resources of IoT. These resources can be sensor readings. Suppose IoT device is reading temperature and motion readings of an industrial device X which has been planted in a remote area. Access policy for motion reading is different from access policy of temperature readings. Device X's motion and temperature settings will be controlled remotely by some workers at different posts. A worker with appropriate attributes will be granted access to read temperature or motion readings. Token will be given to appropriate workers to access the data. The decision to make changes in device settings according to temperature readings or motion readings will be taken after accessing these data. Figure-6.6 shows such a setting.

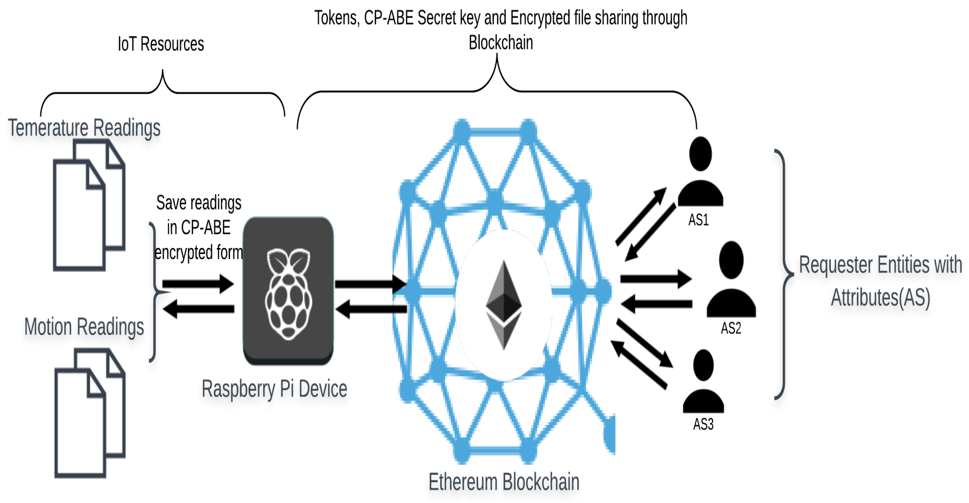

Tokens, CP-ABE Secret key and Encrypted file sharing through Blockchain
IoT Resources
Temerature Readings
Save readings in CP-ABE encrypted form
Motion Readings
Raspberry Pi Device
Ethereum Blockchain
AS1
AS2
AS3
Requester Entities with Attributes(AS)

Figure 6.6: Accessing sensitive data through Blockchain

Such access control system can be used in healthcare systems. In healthcare, it can be used for secure patient monitoring or to improve clinical efficiency. Doctors use IoT sensors to track patient visits and read EMRs. Some wearable sensors are also used. Examples of wearable sensors are- biometric sensor, brain sensors etc. Some sensors are used to monitor environment like home/consumer monitoring, infant monitoring and sleep monitoring etc[37]. Only doctors or nurses with right attributes should be able to access such sensitive IoT information.

# Chapter 7

# Conclusion

Providing fine-grained access control and user privacy in IoT devices has become a necessity. Ouaddah et. al.[1] proposed an idea to use Ethereum blockchain and smart contracts for such purpose. Considering the idea proposed by Ouaddah et. al.[1] and collaborating it with a CP-ABE a new system is made to provide such fine-grained access control. Any type of interaction between IoT devices and users are done through Ethereum blockchain and smart contracts. Access is token-based. All data is encrypted with CP-ABE. Legitimate users will be able to access data provided by IoT devices using valid tokens. Requesters are given user credentials to access the encrypted data provided by IoT devices. Smart contracts will save tokens and provide it to users on demand. Implementation is done on Raspberry Pi2 device and Linux16.04 OS-based Dell PCs with acceptable timings.

# Bibliography

[1] Ouaddah Aafaf, Anas Abou Elkalam, and A. A. I. T. Ouahman. "Harnessing the power of blockchain technology to solve IoT security & privacy issues." Second Int. Conf. Internet Things, Data Cloud Comput.(ICC 2017). Cambridge City, United Kingdom: ACMâĂŘInternational Conference Proceedings Series (ICPS), 2017.

[2] NEST, "Nest Thermostat," 2015. [Online]. Available: https://nest.com/uk/thermostat/meet-nest-thermostat/.

[3] Cisco, âĂIJCisco Visual Networking Index Predicts Near-Tripling of IP Traffic by 2020,âĂİ Newsroom.Cisco.Com, 2016.

[4] Ouaddah, Aafaf, Anas Abou Elkalam, and Abdellah Ait Ouahman. "Towards a novel privacy-preserving access control model based on blockchain technology in IoT." Europe and MENA Cooperation Advances in Information and Communication Technologies. Springer, Cham, 2017. 523-533.

[5] Ouaddah, Aafaf, Anas Abou Elkalam, and Abdellah Ait Ouahman. "FairAccess: a new BlockchainâĂŘbased access control framework for the Internet of Things." Security and Communication Networks 9.18 (2016): 5943-5964.

[6] Pinno, Otto Julio Ahlert, Andre Ricardo Abed Gregio, and Luis CE De Bona. "ControlChain: Blockchain as a Central Enabler for Access Control Authorizations in the IoT." GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017.

[7] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).

[8] Alharby, Maher, and Aad van Moorsel. "Blockchain-based Smart Contracts: A Systematic Mapping Study." arXiv preprint arXiv:1710.06372 (2017).

[9] EU funded Horizon 2020 Program. biotope. Available at http://www.biotope-project.eu/.

[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in Proceedings of the IEEE Symposium on Security and Privacy (SP'07), pp. 321-334, May 2007.

[11] K. Emura, A. Miyaji, K. Omote, A. Nomura, and M. Soshi, "A ciphertext-policy attribute-based encryption scheme with constant ciphertext length," International Journal of Applied Cryptography, vol. 2, no. 1, pp. 46-59, 2010.

[12] Puthal, D., Malik, N., Mohanty, S. P., Kougianos, E., and Yang, C. (2018). The blockchain as a decentralized security framework. IEEE Consum. Electron. Mag., 7(2), 18-21.

[13] Chalaemwongwan, Nutthakorn, and Werasak Kurutach. "State of the art and challenges facing consensus protocols on blockchain." Information Networking (ICOIN), 2018 International Conference on. IEEE, 2018.

[14] Buterin, Vitalik, and Virgil Griffith. "Casper the Friendly Finality Gadget." arXiv preprint arXiv:1710.09437 (2017).

[15] Kikitamara, Sesaria, M. C. J. D. van Eekelen, and Dipl Ing Jan-Peter Doomernik. "Digital Identity Management on Blockchain for Open Model Energy System." MS Thesis, Radbound University(2017).

[16] Buterin, Vitalik. On Public and Private Blockchain. Aug. 2015. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[17] Multichain. URL: https://www.multichain.com/

[18] Blockchain Technologies for Business. URL: https://www.hyperledger.org.

[19] Blockchains & Distributed Ledger Technologies. URL: https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/

[20] Access Control for Internet of Things. URL: https://www.intopalo.com/blog/2015-05-25-access-control-for-internet-of-things/

[21] Dan Boneh and Matthew K. Franklin. "Identity-based encryption from the Weil pairing". SIAM J. Comput., 32(3):586-615, 2003.

[22] Xavier Boyen and Brent Waters. "Anonymous hierarchical identity-based encryption (without random oracles)". In CRYPTO, pages 290-307, 2006.

[23] Ran Canetti, Shai Halevi, and Jonathan Katz. "Chosen-ciphertext security from identity-based encryption". In EUROCRYPT,pages 207-222,2004.

[24] C. Lee, P. Chung, and M. Hwang, "A survey on attribute-based encryption schemes of access control in cloud environments", International Journal of Network Security, vol-15, no. 4, pp. 231-240, 2013.

[25] Lewko, Allison, and Brent Waters. "Decentralizing attribute-based encryption." Annual international conference on the theory and applications of cryptographic techniques. Springer, Berlin, Heidelberg, 2011.

[26] Szabo, N. (1994). Smart contracts. Unpublished manuscript.

[27] Smart Contract Application Examples and Use Cases. URL: https://www.draglet.com/blockchain-services/smart-contracts/use-cases/

[28] Zhang, Yuanyu, et al. "Smart Contract-Based Access Control for the Internet of Things." arXiv preprint arXiv:1802.04410 (2018).

[29] Christidis, Konstantinos, and Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things." Ieee Access 4 (2016): 2292-2303.

[30] Gupta, Yash, et al. "The applicability of blockchain in the Internet of Things." Communication Systems & Networks (COMSNETS), 2018 10th International Conference on. IEEE, 2018.

[31] Maesa, Damiano Di Francesco, Paolo Mori, and Laura Ricci. "Blockchain based access control." IFIP International Conference on Distributed Applications and Interoperable Systems. Springer, Cham, 2017.

[32] Seitz, Ludwig, GÃűran Selander, and Christian Gehrmann. "Authorization framework for the internet-of-things." World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a. IEEE, 2013.

[33] Azaria, Asaph, et al. "Medrec: Using blockchain for medical data access and permission management." Open and Big Data (OBD), International Conference on. IEEE, 2016.

[34] Fujimura, Shigeru, et al. "BRIGHT: A concept for a decentralized rights management system based on blockchain." Consumer Electronics-Berlin (ICCE-Berlin), 2015 IEEE 5th International Conference on. IEEE, 2015.

[35] Zyskind, Guy, and Oz Nathan. "Decentralizing privacy: Using blockchain to protect personal data." Security and Privacy Workshops (SPW), 2015 IEEE. IEEE, 2015.

[36] CP-ABE Toolkit. URL: http://acsc.cs.utexas.edu/cpabe/

[37] IoT sensors. URL: https://www.cbinsights.com/research/iot-healthcare-market-map-company-list/

[38] Lucidchart to create figures. URL: https://www.lucidchart.com/

[39] Sandhu, Ravi. "Engineering authority and trust in cyberspace: The OM-AM and RBAC way." Proceedings of the fifth ACM workshop on Role-based access control. ACM, 2000.

[40] Bitcoin Vs Ethereum. URL: https://learn.onemonth.com/bitcoin-vs-ethereum/

[41] Sethia, Divyashikha, Huzur Saran, and Daya Gupta. "CP-ABE for Selective Access with Scalable Revocation: A Case Study for Mobile-based Healthfolder."International Journal of Network Security, Vol.20, No.4, PP.689-701, July 2018.

[42] Ambrosin, Moreno, Mauro Conti, and Tooska Dargahi. "On the feasibility of attribute-based encryption on smartphone devices." Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems. ACM, 2015.

[43] Sethia, Divyashikha, Suraj Singh, and Vaibhav Singhal. "ABE Based Raspberry Pi Secure Health Sensor (SHS)." Advances in Ubiquitous Networking 2. Springer, Singapore, 2017. 599-610.

[44] L. Touati and Y. Challal, "Efficient CP-ABE Attribute/Key Management for IoT Applications," 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool, 2015, pp. 343-350.

[45] Top 10-Differences between Bitcoin and Ethereum. URL: https://medium.com/blockmatics-blog/top-10-differences-between-bitcoin-and-ethereum-d2d3dd62101

[46] Jemel, Mayssa, and Ahmed Serhrouchni. "Decentralized access control mechanism with temporal dimension based on blockchain." 2017 IEEE 14th International Conference on e-Business Engineering (ICEBE). IEEE, 2017.

[47] Yuan, Chao, et al. "Blockchain with Accountable CP-ABE: How to Effectively Protect the Electronic Documents." Parallel and Distributed Systems (ICPADS), 2017 IEEE 23rd International Conference on. IEEE, 2017.