A Permissioned Blockchain based Access Control System for IOT

MD Azharul Islam
Department of Computer Science
Missouri University of Science and Technology, USA
mdazharul.islam@mst.edu

Sanjay K. Madria
Department of Computer Science
Missouri University of Science and Technology, USA
madrias@mst.edu

***Abstract*—IoT devices produce a lot of valuable and sensitive data that is often shared with external parties to provide different kinds of useful services. Traditional IoT access control systems are centralized and do not include all the stakeholders in the access control decision making process. To fill this gap, we propose a permissioned blockchain based access control system for IoT where a different phase of access control like creating access policy and making the access control decision happens based on the consensus of all the stakeholders. To be more specific, we design and implement Attribute Based Access Control (ABAC) in a permissioned blockchain called Hyperledger Fabric and leverage its smartcontract and distributed consensus to enable a distributed access control for IoT. The effectiveness of our proposed system is demonstrated by the performance evaluation result in an IoT testbed.**

***Index Terms*—IOT, Blockchain, Access control**

I. Introduction

As IoT devices are becoming more popular, security and privacy of the heterogeneous data produced by these devices have become more important than ever before. This is because the data produced by IoT devices can contain extremely private information like audio and video clips from smart surveillance systems, medical information from fitness devices, location and activity pattern or even daily schedule of individuals in the house hold. Often times, IoT devices are not utilized to their full potential unless this data is shared with different service providers. For example, data from fitness devices may need to be shared with the physician and the hospital, temperature sensor data may need to be shared with the emergency department and service providers like Amazon and Google can collect user data through smart home devices like Echo, Google home etc. to ensure better quality of service. While sharing the IoT device data with other parties, there are two fundamental questions that need to be asked: 1) **Who** is accessing the shared data? and 2) **What** data is accessed? Answer to the first question determines whether the data falls into the hands of the wrong parties. On the other hand, the second question is to find out whether IoT data requester is collecting anything without the data owner's consent.

Currently, how data requester collects user data from IoT devices lacks transparency and even doubtful in some cases. This is because the owner has no role in the access control of how the data will be shared with the data consumer. Although in some cases, requesters provide the owner with some kind of agreement policy that the owner has to agree on to enjoy the intended service. This leaves the data owner with no other choice but to trust the data consumer blindly. These agreement policies are often very high level and obscure. Moreover, there is no way for the data owner to verify whether the requester is complying with the agreement and not collecting anything more than what was agreed upon. On top of that, it is hard to tell if different service providers implement their security mechanisms properly. This gives the malicious parties an opportunity to get access to the user's confidential and sensitive IoT device data by exploiting any security backdoor that may exist.

The above mentioned problems of IoT data security mainly stem from the fact that different parties involved in the IoT ecosystem are under different administrative entity and there may be a lack of trust between them. Using traditional approaches like [1], it is impossible to ensure the active participation of all mutually untrusted parties in different aspects of the IoT access control mechanism, like fixing the access policy and making the access control decision. Blockchain offers a great platform to build distributed applications for mutually untrusted parties by eliminating the need of a trusted central authority. At a high level, blockchain is a distributed immutable ledger maintained by a network of peers where all the peers in the network at any given point of time agrees on a single identical version of the ledger through some consensus protocol.

The blockchain empowering the cryptocurrencies like bitcoin [2] and Ethereum [3] are called public or permissionless blockchain as no permission is needed for a peer to participate in the blockchain. While public blockchain is well-suited for cryptocurrency, it has a scalability issue that limits the number of transactions the network can process referred to as blockchain bloat [4]. For example, bitcoin can process only a maximum of seven transactions per minute. This is due to the fact that the block creation frequency (1 block per 10 minutes) and size (1MB) is limited [2]. The security of the public blockchain relies on the proof of work (PoW) where all the peers in the network validate all the transactions and try to solve a computationally intensive cryptographic puzzle. The hardness of the puzzle is set so that a new block is created every 10 minutes. Due to the network latency, there exist multiple forks of the blockchain and it can take up to six hours to eventually reach a consensus. That is why transaction wait time is very high in pubic blockchain (sometimes up to six hours). Though, consensus protocols like proof of stake (PoS) are there, the transaction wait time is still high in public blockchain [5]. However, in the private or permissioned blockchain, the transactions are much faster. This is because it does not rely on PoW or PoS.

This research has been partially funded by NSF grant CNS 1460697.

978-1-7281-4693-5/19/$31.00 ©2019 IEEE
DOI 10.1109/Blockchain.2019.00071

Rather, it incorporates much faster consensus protocols like Byzantine Fault Tolerance (BFT) and yet provides a way to secure the transactions among a group of participants with verified identities who have a common goal but do not fully trust each other [5]. As a result, it is a better fit for IoT access control. This inspires us to propose an access control system for IoT which is based on permissioned blockchain. By utilizing the smartcontract, we implement attribute based access control (ABAC) in the blockchain which is a great fit for IoT than other access control mechanisms such as role based access control or identity based access control. This is because IoT ecosystem consists of a huge number of IoT devices that varies in functionalities, characteristics and capabilities. Only ABAC can offer expressive fine-grained access control in such a diverse environment. We implement our access control system in Hyperledger Fabric [5] which is an open source implementation of a permissioned blockchain, and evaluate its effectiveness in an IoT testbed. Our results also demonstrates the practicality of our proposed system. The most closely related work to ours is [6] which is a physical access control management system based on permissioned blockchain. However, it is not specifically intended for IoT and it can not therefore enforce fine-grained access control like ours as role based access control was used. IoT access control systems proposed in [7]–[10] are also related to ours. Since they are based on public blockchain, it is needless to say that all of them inherit the the existing limitations of the public blockchain. Besides, access control mechanisms proposed in these works are not as sophisticated as ABAC.

Our contributions can be summerized as follows:

- To the best of our knowledge, we are the first to propose a private blockchain based IoT access control system based on attribute based access control. Since our scheme is based on private blockchain, the access requests are resolved much faster than that of a public blockchain.
- We report a full implementation of our proposed system in a permissioned blockchain platform called Hyperledger Fabric and prove its practicality by evaluating its performance in an IoT testbed environment.

II. Related work

Besides cryptocurrency, the adoption of blockchain is also noticeable in the IoT spectrum. For example, an access control mechanism based on bitcoin was proposed in [4], [7]. Since, bitcoin does not support smartcontract, the proposed access control mechanism is very basic and does not offer fine-grained access control for heterogeneous IoT devices. [8] proposed an architecture for scalable access management of IoT devices based on Ethereum smartcontract where blockchain is run in the IoT devices. It was implemented in a small scale local Ethereum test network. However, it is not clear how it is going to work in the original pubic Ethereum network. Dorri et al. proposed a solution in [9]–[11] where an overlay network is formed by the IoT nodes. Multiple overlay nodes form a cluster with an elected cluster head (CH) for each cluster. The CHs maintains a newly proposed pubic blockchain. It has a new transaction format and does not have smartcontract which makes implementation of sophisticated access control policy really hard. An access control management system based on private blockchain has been proposed in [6] although it was not directly intended for IoT. Besides, role-based access control was considered in this work which is not a good fit for IoT access control involving a large scale of heterogeneous devices with lack of standardization.

III. Background

In this section, we discuss the necessary background of for our proposed system. First, we discuss how resource is managed inside an IoT network. Then, we discuss different components of Hyperledger Fabric.

A. Resource Management in an IoT Network

Many standards in line with the IEEE 802.15.4 radio [12] were defined by the IETF for IoT networks. IoT-Auth [1] adopted many of these well defined protocols to manage resource within an IoT network. For this purpose, the *gateway* maintains three data structures: *Routing Table, Resource Table*, and *Data Table*. There are following steps involved:

1) Device Discovery: The first step is to do the device discovery. It is done according to the Routing Protocol for Low Power and Lossy Networks (RPL) protocol. More specifically, a network coordinator or *sink node* initiates this process by periodically sending in the IoT network the Destination Oriented Directed Acyclic Graph (DODAG) Information Object (DIO) message. A newly joining device replies with the Destination Advertisement Object (DAO) message. The *sink node* forwards this message to the *gateway*. The *gateway* stores the identifier of all the active IoT end nodes along with their communication path in the *Routing Table*.

2) Resource Discovery: The *gateway* starts the process by sending a GET message to the well-known URI of the IoT device called *./well-known/core*. The IoT device sends a response message in the Constrained RESTful Environments (CoRE) Link Format. The response message includes information such as resource type *(rt)*, interface description *(rt)*, and maximum size estimate *(sz)*. These information are processed and stored in the *Resource Directory*.

3) Data collection: The *gateway* collects data from the intended IoT device as new data is available. Data is stored in the *Data Table* along with its identity information.

B. Hyperledger Fabric

Hyperledger Fabric is an opensource implementation of a permissioned blockchain [5]. It offers a modular architecture allowing different kinds of pluggable functionalities by leveraging well known and proven technologies. One of the most powerful aspect of Hyperledger Fabric is that it gives a platform to run smartcontracts on the blockchain. Smartcontracts are special kinds of programs that can be written using traditional programming language such as GO to perform different kinds of operations on the underlying blockchain. We discuss the core building blocks of Hyperledger Fabric as follows:

1) Peer, Organization, and Client: In Hyperledger Fabric, peers are the nodes that host the blockchain and runs the smartcontract. There could be two basic types of peers based on the role it takes up: *validating peer* and *non-validating peer*. A *validating peer* runs the consensus protocol, executes and validates transactions, and maintains the blockchain. A *non-validating peer* acts as a proxy to connect the external applications/clients to the *validating peers*. Some peers can take up a special role of endorsing transactions referred to as *endorsing peers*. Peers are part of a conceptual entity called the organization. Each peer is part of some organization and multiple organizations collectively maintain the blockchain. On the other hand, clients are the entities that submit transaction requests to the blockchain. They are normally third-party applications written by the provided SDK.

2) Membership Service Provider (MSP): The permission to participate in the blockchain is handled by the MSP. For example, every peer needs to collect *enrollment certificate* and *transaction certificate* from the designated certificate authority (CA) of the MSP to connect to the network and submit transactions, respectively. Each organization can have a separate MSP that independently operates its own membership service.

3) Transaction Endorsement: Hyperledger fabric does not rely on *proof-of-work* or *proof-of-stake* to maintain the immutability of the blockchain or to prevent double spending. Rather, it relies on the *endorsement policy* that states which peers need to endorse a transaction to be considered as a valid one. An *endorsement policy* is written using *endorsement policy syntax*. For example, endorsement policy *OR('Org1.peer1', AND('Org2.peer2', 'Org3.peer3'))* states that transaction needs to be endorsed by either peer1 of Org1 or by both peer2 of Org2 and peer3 of Org3.

4) Ordering Service: The ordering service accepts endorsed transactions from the client, orders them according to the plugged-in consensus protocol, and delivers them to the designated peers to be written in the blockchain. It guarantees the proper ordering of the transactions that ensures the consensus.

5) Chaincode: Chaincode is similar to smartcontract in the context of Hyperledger Fabric. These are piece of programs written in traditional programming language such as Go, java, and node.js and can manipulate the blockchan. In this paper, we will use the term smart contract and chaincode interchangeably.

6) Blockchain Data Structures: Blockchain in Hyperledger Fabric incorporating two different kinds of data structures: *state* and *ledger*. *State* stores the latest state of the blockchain by modeling it as a key-value storage (KVS). It is maintained and hosted by the peers and can be manipulated from the chaincode, triggered by transactions. On the other hand, *ledger* stores the verifiable history of all the unsuccessful attempts and successful change made in the *state* as a totally ordered *hashchain* of blocks of transactions.

IV. Proposed System Architecture

We discuss our system architecture in this section. Our proposed architecture is depicted in Fig.1. In the following subsections, we first discuss the main actors of our system. Then, we discuss the main components followed by how the constrained IoT resource is accessed by a requester who is external to the IoT network.

A. Actors

There are mainly two types of actors in our system as discussed below:

1) Resource Provider/Owner: This actor is basically the owner of the IoT equipped smart home, smart office, school etc. where variety of IoT devices produce different kinds of data. The data could range from environment sensing data (such as temperature, pressure, humidity, luminosity, etc.) to healthcare data generated by wearable devices or even image, audio and video data generated by surveillance systems.

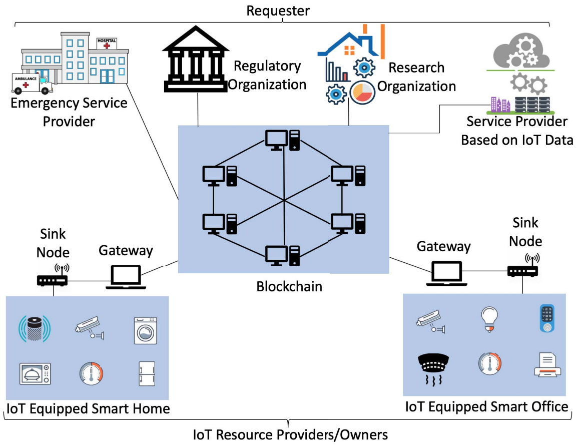

Fig. 1: System Architecture

2) Requester: Any party that accesses data generated by IoT devices is a requester. Normally, a party who relies on the IoT data to provide different kinds of service is considered as data requester. For instance, google, amazon provides services like music streaming, voice search result etc. based on the data provided by google home, amazon echo. Emergency service providers such as hospitals, fire service etc. are also requesters since emergency services may rely on the IoT device such as elderly monitoring device, or different healthcare devices for data. Different research organizations may also rely on user IoT data to conduct scientific research and survey. Finally, regulatory organizations are also considered as requester since they may need to access IoT data for auditing purpose. These requesters are generally external to the IoT local network and access IoT data by through access control mechanism imposed by blockchain.

B. Components

The main components of our system are the local IoT networks and the blockchain. A brief discussion of these components are given below:

1) Local IoT Network: Each local IoT network is composed of one or more IoT devices, a sink node and a gateway. The sink node works like a network coordinator for all the IoT devices and is connected to the gateway. The gateway acts as an interface to the external world to access any resource within the local IoT network. A gateway can have its own public IP address or may be connected to the cloud that provides it with a public interface so that resource requesters can access IoT data from outside the local IoT network. It manages all the information available within the IoT network. For this purpose, it maintains three data structures named as *Routing Table, Resource Dictionary,* and *Data Table*. According to our architecture, there can be many local IoT networks as such, each representative of an IoT equipped smart home, office or school etc.

2) Blockchain: The blockchain in our architecture is a permissioned one, implemented in Hyperledger Fabric. All the attributes and ABAC policies upon validation are stored in the

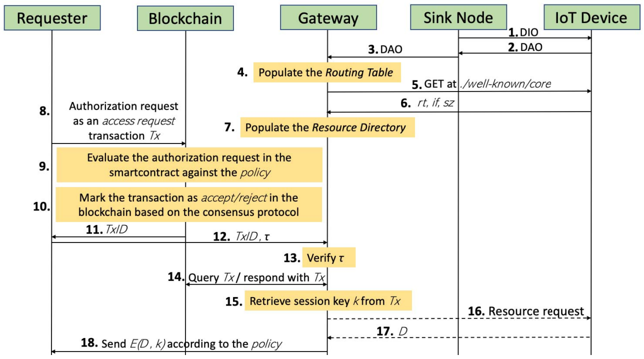
Requester
Blockchain
Gateway
Sink Node
IoT Device
1. DIO
2. DAO
3. DAO
4. Populate the Routing Table
5. GET at ./well-known/core
6. rt, if, sz
7. Populate the Resource Directory
8. Authorization request as an access request transaction Tx
9. Evaluate the authorization request in the smartcontract against the policy
10. Mark the transaction as accept/reject in the blockchain based on the consensus protocol
11. TxID
12. TxID, τ
13. Verify τ
14. Query Tx / respond with Tx
15. Retrieve session key k from Tx
16. Resource request
17. D
18. Send E(D, k) according to the policy

Fig. 2: Resource Access

blockchain. It also works as a policy enforcement point for any access request to a particular IoT resource. In brief, the blockchain provides an unified ABAC platform for the entire IoT ecosystem.

C. Resource Access Process by the Requester

The sequence diagram presented in Fig. 2 shows how the IoT resource is accessed by the requester. Steps 1, 2, 3, and 4 is for the *Device Discovery* and populating the *Routing Table*. Steps 5, 6, and 7 are to complete the *Resource Discovery* process and populate the *Resource Table* as described in Sec. III-A. At this point, a requester who is external to the IoT local network, can request for IoT resource. First, the requester sends the authorization request as an *access request* transaction Tx (step 8). Let PK_{Ow} and DK_{Ow} be the public and private key of the IoT resource provider/owner respectively; *Enc* and *Dec* be the public key encryption and decryption algorithm respectively; SK_{Re} and VK_{Re} be the requester's signing and verification key respectively; *Sig* and *Ver* be the DSA (Digital Signature Algorithm) signing and verification algorithm respectively, and $H(.)$ be a secure hash function. In the authorization request, the requester sends the following information:

$$Req = (Resource\ Location,\ policyID,\ C),\ \sigma \tag{1}$$

Resource Location is Eqn. 1 is the IP address or public interface of the *gateway* of the intended IoT network, *PolicyID* is the ID of the ABAC *policy* that determines the access control rule of the resource. $C = Enc(k,\ PK_{Ow})$, where k is a session key. Signature σ is computed as $\sigma = Sig(H(Req),\ SK_{Re})$.

In step 9, each *endorsing peer* in the blockchain first checks the validity of the request by checking the following equality:

$$H(Req) == Ver(\sigma,\ VK_{Re}) \tag{2}$$

Then, the *policy* corresponding to the *PolicyID* is retrieved from the blockchain. Each *endorsing peer* evaluates the request in the smartcontract against the *policy*. In step 10, the transaction is marked as *accept / reject* based on the consensus protocol and committed in the blockchain. The following information is saved in the transaction:

$$TxID, Tx = (Req,\ \sigma,\ Decision) \tag{3}$$

Here, *TxID* is a unique ID for *Tx*, and *Decision* can have as a value either *accept* or *reject*. Then, the requester gets the *TxID* from the blockchain (step 11). Later, the requester sends $TxID$, $\tau = Sig(H(TxID),\ SK_{Re})$ to the *gateway* indicating its intention to access the IoT resource (step 12). In step 13, the *gateway* checks the authenticity of the request by verifying the requester's signature as in $H(TxID) == Ver(\tau,\ VK_{Re})$. Later in step 14, the *gateway* queries the blockchain for transaction Tx with *TxID* and gets the *Tx* as response. Then (in step 15), upon confirming if the transaction was marked as *accept*, the *gateway* retrieves the session key k by decrypting C with its decryption key as $k = Dec(C,\ DK_{Ow})$. The *gateway* checks if the resource is sent to the proper requester by verifying σ using Eqn. 2 with the same verification key used to verify τ. If the requested resource according to the *policy* is available in the *Data Table*, the *gateway* sends to the requester the data D right away by symmetrically encrypting it with k as in $E(D,k)$ (step 18). Otherwise, the *gateway* polls the resource from the designated IoT device(s). That is why steps 16 and 17 are shown in dotted arrows.

V. Details of our ABAC model

In this section, we discuss our attribute based access control (ABAC) model in detail. This includes how we model attributes and policies and how those policies are evaluated against attributes.

A. Modeling Attributes

Attribute is a core component of our ABAC system. We model attribute as a multi-component data structure. Each attribute is composed of three components: name, value, and type. Name is simply a string. On the other hand, value can be any data type such as string, numeric, date etc. The general representation of an attribute is as follows:

$$att_i^t = (name,\ t,\ val) \tag{4}$$

name, t, and *val* stands for the name, type, and value of the attribute, respectively. We use the dot (.) operator to access an assigned member of an attribute, i.e. $att_i^t.val$ is used to access the assigned values of att_i^t. The set of all possible values of an attribute att_i^t is denoted as $Val(att_i^t) = \{val_{i,j} : 1 \leq j \leq M_i\}$. M_i is the total number of possible values of att_i. In our system, we consider the following four types of attributes:

1) Subject Attribute: This type of attribute is used to completely describe the requester or the resource owner. Examples of subject attribute can be the organization, title, rank etc. Let att_i^{Sub} and N_{Sub} be a subject attribute and the total number of subject attributes in the system, respectively. Then, the set of all subject attributes are expressed as $Attr(Sub) = \{att_i^{Sub} : 1 \leq i \leq N_{Sub}\}$.

2) Resource Attribute: Anything that can describe the IoT resource properly are considered as resource attributes and is denoted as att_i^{Res}. For example, the IoT resource name, type, identifier etc. are resource attributes. If there are N_{Res} resource attributes in the system, then all resource attributes are represented as $Attr(Res) = \{att_i^{Res} : 1 \leq i \leq N_{Res}\}$.

3) Environment Attribute: Time, location etc. related attributes are considered as environment attributes and are represented as att_i^{Env}. We assume that there are N_{Env} environment attribute in total and express them as $Attr(Env) = \{att_i^{Env} : 1 \leq i \leq N_{Env}\}$.

4) Action Attribute: Any type of action the requester or resource owner is allowed to perform falls into this category. Example includes attributes such as read, write, delete, update etc. Action attributes are represented as att_i^{Act}. Let *Attr(Act)* be the set of all N_{Act} possible action attributes in the system which is expressed as $Attr(Act) = \{att_i^{Act} : 1 \leq i \leq N_{Act}\}$.

B. Modeling Policy

A policy is expressed as a boolean expression that defines the access rule to a resource in terms of attributes and their corresponding values. In our ABAC model, policies are very expressive as both attribute values and attributes themselves are boolean expressions. A complete policy consists of *attribute value expression* and *attribute expression* as discussed below:

1) Attribute Value Expression: Allowed values of an attribute att_i^t in a policy are expressed as the following boolean expression:

$$E_{att_i^t} := Exp\ (\mathcal{E}[,\mathcal{E}]) \tag{5}$$

In Eqn. 5, *Exp* is either *AND* or *OR* joining two boolean expressions and $\mathcal{E}$ is either an attribute value $val_{i,j} \in Val(att_i^t)$ or a recursive call to *Exp*.

2) Attribute Expression: For each attribute type *t*, we use a separate policy $\mathbb{P}_t$. Each such policy is a boolean expression composed of *t* type attributes as in:

$$\mathbb{P}_t := Exp\ (\mathbb{E}[,\mathbb{E}]) \tag{6}$$

In the above equation, *E* is either an attribute value expression $E_{att_i^t}$ or a recursive call to *Exp*. Finally, the combined policy is written as a conjunction of all four types of policies as in Eqn. 7:

$$\mathbb{P} = \mathbb{P}_{Sub}\ AND\ \mathbb{P}_{Res}\ AND\ \mathbb{P}_{Env}\ AND\ \mathbb{P}_{Act} \tag{7}$$

C. Policy Evaluation

Granting an access request for a resource is determined by evaluating a policy against a set of subject, object, environment and action attributes. A complete policy $\mathbb{P}$ contains four *attribute expressions* and each attribute in an *attribute expression* contains an *attribute value expression*. So, evaluation of $\mathbb{P}$ can be broken down into following two parts:

1) Evaluation of Attribute Value Expression: If $\mathcal{S} \subset Val(att_i^t)$ is a set of assigned values to a particular attribute att_i^t, and $\mathcal{I}$ is the minimum number of values required to satisfy the boolean expression in $E_{att_i^t}$, then we say that att_i^t satisfies $E_{att_i^t}$ if $\mathcal{I} \subset \mathcal{S}$. It is expressed by the following notation:

$$E_{att_i^t} \vdash att_i^t = \begin{cases} true, & \text{if } \mathcal{I} \subset \mathcal{S}, \\ false, & \text{otherwise.} \end{cases}$$

2) Evaluation of Attribute Expression: Let $\mathbb{I}_t$ be the minimum number of required attributes to satisfy the boolean expression in $\mathbb{P}_t$, and $E_{att_i^t} \vdash att_i^t = true$ for $\forall att_i^t \in \mathbb{I}_t$ where $E_{att_i^t} \in \mathbb{P}_t$. Then we say that attribute set $\mathbb{A}_t \subset Att(t)$ satisfies $\mathbb{P}_t$ if $\mathbb{I}_t \subset \mathbb{A}_t$, and it is represented by the following notation:

$$\mathbb{P}_t \vdash \mathbb{A}_t = \begin{cases} true, & \text{if } \mathbb{I}_t \subset \mathbb{A}_t, \\ false, & \text{otherwise.} \end{cases}$$

Finally, the satisfaction of a complete policy $\mathbb{P}$ by an attribute set $\mathbb{A} = \{\mathbb{A}_{Sub}, \mathbb{A}_{Res}, \mathbb{A}_{Env}, \mathbb{A}_{Act}\}$ is represented by the following notation:

$$\mathbb{P} \vdash \mathbb{A} = \begin{cases} false, & \text{if } \exists \mathbb{A}_t \in \mathbb{A} : \mathbb{P}_t \vdash \mathbb{A}_t = false, \\ true, & \text{otherwise.} \end{cases}$$

VI. ABAC IMPLEMENTATION IN HYPERLEDGER FABRIC

In this section, we discuss how our ABAC model discussed in the previous section is implemented in Hyperledger Fabric. The first step towards our ABAC implementation is to form the blockchain network. After that, attribute creation and assignment, policy creation and resource access request are done by sending transactions to the blockchain by the requester. These steps are discussed in the following subsections.

A. Forming the Blockchain Network

The permission to join the blockchain is managed by the MSP (Managed Service Provider) of some high-level organizations. For our ABAC implementation in the Hyperledger Fabric, we assume that there exist multiple high level organizations. For example, an organization may represent all the regulatory institutions, companies like google, amazon who provide IoT based services can have their own authoritative organizations in the blockchain, and research institutions can have an authoritative organization as well. Finally, the data owners must be part of some organization also. For example, a smart city can play the role of an organization for all the smart home owners of that city. Each organization may have one or more running peers. It is worth noting that the data owners need to have their own running peers to be able to directly take part in the access control decision making process of their data. There will be some dedicated nodes that will perform the ordering service. Peers from different organizations along with the ordering service collectively form and run the blockchain.

B. Setting up the Endorsement Policy

The consensus in hyperledger fabric largely depends on the endorsement policy. This is because it dictates who need to endorse a particular transaction to be considered by the validating peers as a valid one. A data owner fixes the endorsement policy by creating a configuration transaction in the blockchain. The endorsement policy along with the identity of all the endorsing peers are embedded in this transaction. An endorsement policy creates a logical channel between the endorsing peers and the ordering service. In order to be committed in the blockchain, transactions submitted to a channel need to be endorsed by the channel's endorsing peers according to the endorsement policy. Different data owners will have different endorsement policies. Hence, many channels as such will exist in the Hyperledger Fabric.

C. Attribute Management

For an ABAC system to function properly, attributes should be created with name, type and a set of allowable values. After creating attributes, they need to be assigned to different entities. Subject attributes are created and assigned to the specific subject by an attribute authority through an administration point. There are multiple such authorities, each with authority over different set of subject attributes. In our blockchain based

ABAC model, the MSP of each organization plays the role of this attribute authority. Resource attributes on the other hand are created and assigned to the specific resource by the owner of that resource. Environment and action attributes are system wide common, and must be created by the regulatory organizations. We use general terms attribute creator and issuer to refer to the entity responsible for attribute management. In practice, they are the same entity. Attribute creation and assignment is handled by a smart contract function named *AttributeMgr* and details are discussed below.

1) Attribute Creation: No attribute can be used in our system without registering it in the blockchain. To register an attribute, the creator first sends a transaction request in the bockchain. Within the transaction, the complete attribute structure as in Eqn. 4 is embedded. *AttributeMgr* parses the attribute from the transaction, checks the semantics, and converts it into a json object. Besides name, type and value, some additional fields such as creatorID, organization name (orgName) are also added in the json object. Upon endorsement, ordering and verification phase, the json object is written in the key-value storage of Hyperledger Fabric called the *state*. One critical issue of ABAC system is the conflict resolution of attributes. Conflict during the attribute creation occurs when two different attributes with the same name are created by two different creators. For example, *manager* attribute of org A is different from the *manager* attribute of org B. The system should allow both org A and org B to create *manager* attribute. At the same time, the system should know the difference between them. *AttributeMgr* resolves this issue prior to writing the attribute in the *state* by creating unique IDs for each attribute as follows:

$$ID_{att_i^t} = H\left(orgName \mid\mid \text{creatorID} \mid\mid att_i^t.name \mid\mid att_i^t.t\right)$$

$ID_{att_i^t}$ is used as the key for the attribute when stored in the *state*. Two attributes with the same key cannot be stored in the *state*. This allows the creation of attributes with same name by two different creators while restricting same creator from creating duplicate attributes.

2) Attribute Assignment: After attribute creation, the issuer has to assign the attribute along with the appropriate set of values to the proper entities. It is important to take enough security measures so that attributes cannot be altered or tampered with to maliciously satisfy an access policy. To accomplish this, we cryptographically bound attributes to the entities. During attribute assignment, the attribute issuer would add the attributes to the `attributes` field of an X.509 attribute certificate (AC) according to the IETF standard [13]. The issuer sends this certificate by embedding it in a blockchain transaction. *AttributeMgr* verifies the certificate and converts it in a json object to store it in the *state*. Finally, it is stored in the *state* after endorsement, ordering and verification phase.

D. ABAC Policy Management

In our blockchain based access control system, access to the restricted IoT resource is controlled by the ABAC access policy. Both the IoT resource owner and requester first agrees on a policy which is expressed according to our policy model as discussed in Sec. V-B. The policy is sent in a blockchain transaction. The transaction has to be endorsed by both the resource owner and the requester. It is then written in the *state* as a key-value pair with key being *policyID = H($\mathbb{P}$)*, and the value being the policy itself.

Meta Policy: Some policies may require meta policy. It determines things like who can modify or delete the actual policy $\mathbb{P}$, the validity period of $\mathbb{P}$ etc. Meta policy is denoted by $\mathbb{MP}$ and modeled in the similar fashion as the original policy except the resource attribute expression $\mathbb{P}_{Res}$ points to $\mathbb{P}$. Meta policy is included in the blockchain transaction when the policy is created. Default meta policy applies to the policies that do not have any meta policy. According to the default policy, only the resource owner can modify or delete the policy. For the security purpose, meta policy is assumed to be immutable.

A smartcontract function named *PolicyMgr* is responsible for policy management tasks such as checking the semantics while policy creation and modifying the policy according to the meta policy.

E. Policy Evaluation

The policy evaluation logic is implemented in a smart contract function called *ACDecMaker* (access control decision maker). The endorsing peers responsible for endorsing resource access request transaction *Tx* (Eqn. 3) invoke this smartcontract. *ACDecMaker* takes as input *Req = (Resource Location, policyID, C)*$,\sigma$ and checks if the access policy $\mathbb{P}$ corresponding to the *policyID* is satisfied by the relevant attribute set $\mathbb{A}$. The algorithm for *ACDecMaker* is shown in Alg. 1.

Algorithm 1

```
Procedure: ACDecMaker ( Req, $\sigma$)
get policy $\mathbb{P}$ from the state database for key Req.policyID
get the relevant attribute set $\mathbb{A} = \{\mathbb{A}_{Sub}, \mathbb{A}_{Res}, \mathbb{A}_{Env}, \mathbb{A}_{Act}\}$
    from the state database
if H(Req) != Ver($\sigma$, $VK_{Re}$) then
    return (TxID, Tx = (Req, $\sigma$, reject))
end if
for each $(\mathbb{A}_t, \mathbb{P}_t)$ in $(\mathbb{A}, \mathbb{P})$ do
    if $\mathbb{P}_t \vdash \mathbb{A}_t$ == true then
        continue
    else
        return (TxID, Tx = (Req, $\sigma$, reject))
    end if
end for
return (TxID, Tx = (Req, $\sigma$, accept))
```

VII. Experiment

We did a full implementation of our blockchain based access control system in order to demonstrate the practicality of our solution. In the following section, we first provide details of our implementation, i.e. blockchain and IoT network testbed implementation. Then, we evaluate the performance of our system by experiments.

A. The IoT Network

We have developed an IoT testbed in our lab using MEMSICs TelosB Mote TPR2420CA devices [14]. TPR2420CA bundles many essential elements required to perform IoT based lab studies such as an integrated temperature, light

TABLE I: Endorsement policy for IoT resource access control of different IoT groups

ID	Endorsement Policy	Intended IoT group
1	Any peer from OrgA, OrgB or OrgC	Group 1
2	At least one peer from each Org	Group 2
3	All peers from OrgA, OrgB, and OrgC	Group 3

and humidity sensor, an IEEE 802.15.4 radio with integrated antenna etc. In our testbed, we divide the sensors into three groups: group 1, 2, and 3. Each group has four TPR2420CA devices- three of them act as IoT end device and one serves as the purpose of a sink node. All three sink nodes are connected to a PC through a USB hub and directly communicate with the PC through USB port. For each group, the PC runs a separate gateway program. The gateway program is written in java and uses SQLite for storing *Routing Table, Resource Dictionary,* and *Data Table*. Each group along with the gateway forms an individual IoT network. The device discovery and the resource discovery within the IoT network is done according to the IEEE 802.15.4 standard as discussed in Sec. III-A. Each group of our testbed represents an IoT equipped smart home, office or school etc. in real life.

B. The Blockchain Network

Our blockchain network has been implemented in Hyperledger Fabric v1.3 [15]. We have considered three different organizations: OrgA, OrgB, and OrgC for our Hyperledger Fabric setup. We assume that all IoT resource owners belong to OrgA and each of them has a running peer under OrgA (OrgA.peer1-3). OrgB is the representative organization for all resource requesters and there are five running peers (OrgB.peer1-5) under it. Finally, we assume that OrgC represents all regulatory organizations and there are three running peers (OrgC.peer1-3) under it. All peers of OrgA are hosted in a desktop with Intel® Core i5-2400@3.1 GHz × 4 processor and 8 GB RAM running Ubuntu 16.04. A desktop with Xeon(R) ES-1620v2@3.7 GHz × 8 processor and 16 GB RAM running Ubuntu server 16.04.3 was used to host all the OrgB peers. Peers of OrgC run in a desktop which has the similar configuration as that of OrgA. As the ordering service, we have two orderer nodes backed by a kafka-zookeeper cluster. The ordering service runs in a separate desktop having the same configuration as that of OrgA.

We have written a java web based client application using the fabric client SDK to interact with the blockchain. This client application provides a simple interface to create attributes, assign attributes to a particular entity and create ABAC policy targeting a specific resource.

During the bootstrap phase, the MSP of each organization provides the participants with necessary crypto materials, i.e. certificates, signature keys, and encryption keys. Then, three different endorsement policies are created for three different IoT groups as in Table I. After setting up the endorsement policies, our smartcontract with three main functions, i.e. *AttributeMgr, PolicyMrg,* and *ACDecMaker* is installed in each peer. Then, the MSP of each organization creates subject attributes and registers in the blockchain. Environment and action attributes are created by the MSP of OrgC. Finally, the resource attributes for each IoT group is created by their respective owner using the client application. After attribute creation, attribute is assigned to different entities. In our experimental setup, we create 20 subject, 20 resource, 10 environment and 5 action attributes, each having 5 values. Five different ABAC policies were created with the number of attributes required to be satisfied ranging from 10 to 50.

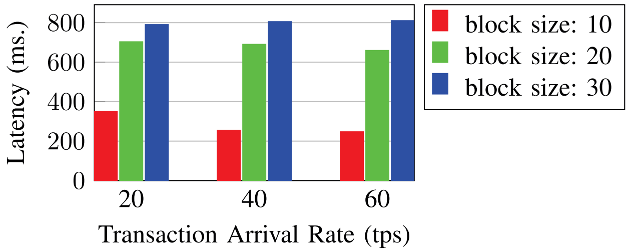

Fig. 3: Attribute Creation

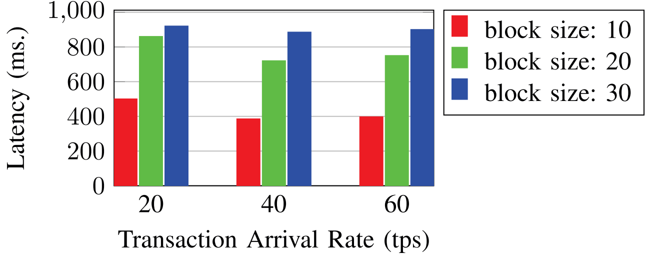

Fig. 4: Attribute Assignment

C. Workloads and Experimental results

We show the performance of our scheme in terms of how fast different ABAC actions can be performed. All the results presented are averaged over five runs. Three of the most important fabric configurable parameters are the *stateDB*, endorsement policy, and block size. Between the two choices of GoLevelDB and CouchDB, we choose *stateDB* as it was shown in [16] that GoLevelDB has better throughput and faster read/write. A new block is created when either the number of pending transactions since the last written block reaches the block size or timeout happens. We set the timeout to be 1 second. Then, we examine the latency for attribute creation, attribute assignment and policy creation operations for block size values 10, 20, and 30 with three different transaction arrival rates (20, 40 and 60 transactions per second). It is noticeable from Fig. 3, 4, and 5 that the latency increases with the increase in block size. For example, when the transaction arrival rate for attribute creation (Fig. 3) is at 40, an increase in the block size from 10 to 30 increases the latency by 3-fold from 255 ms. to 805 ms. This is because with larger block size, a pending transaction has to wait a little longer at the orderer queue causing delay in the transaction writing rate in the blockchain on average. Between attribute creation, attribute assignment, and policy creation, we observe that attribute assignment takes longer on average compared to the other two operations. The reason for this is that *AttributeMgr* in this case verifies the `signatureVale` in X.509 attribute certificate which is an expensive cryptographic operation. Across the board, we notice the lowest latency when the transaction arrival rate is at 40 tps and the block size is 10. Note that, we use the first endorsement policy in Table I for the three experiments discussed above.

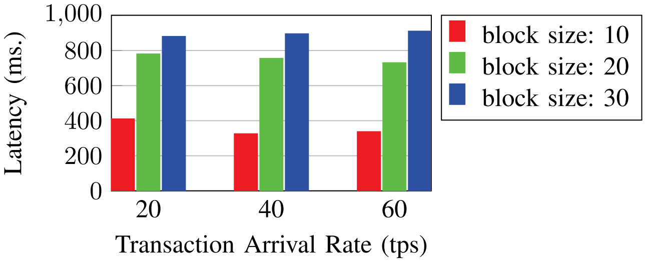

Fig. 5: Policy Creation

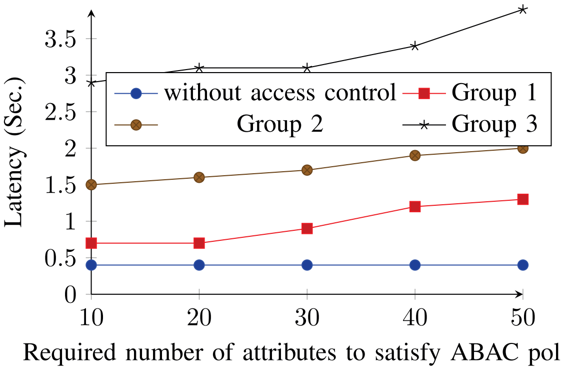

Fig. 6: Latency of serving IoT resource access request

For the next experiment (Fig. 6), we set the block size to be 10 and transaction arrival rate at 40 tps which was found to be the optimum for our Hyperledger Fabric setup. Besides, hash function *H*, public key encryption (*Enc, Dec*), and digital signature (*Sig, Ver*) were implemented using SHA-256, RSA-1024, and DSA with 1024 bits key size, respectively. With these parameters fixed, we measure the latency of serving IoT resource access request for different types of ABAC policies in different IoT groups configured with different endorsement policies as stated in Table I. As a baseline comparison, the latency of serving IoT resource is shown when no access control mechanism is in place. We observe that latency is the lowest (.4 sec.) when there is no access control mechanism in place. On the other hand, group 3 has much higher latency compared to group 1 and 2. The reason is that the endorsement policy of group 3 requires all 11 peers to endorse a transaction while group 1 and 2 requires only 1 and 3, respectively. The latency also increases with the increase in required number of attributes to satisfy ABAC policy. This is because ABAC policy evaluation algorithm is NP-complete and therefore, the complexity increases with the number of attributes in the policy.

Public blockchain based IoT access control schemes are still limited by the very high transaction latency. For example, it may take several hours before a bitcoin transaction is committed in the blockchain. So, it is not suitable for any IoT access control scenario requiring low transaction latency. For instance, medical emergency service requires quick access to the wearable device data of elderly people. Our permissioned blockchain based scheme can serve IoT resource access request much faster (around 4 sec.) by utilizing the low transaction latency in Hyperledger Fabric.

VIII. Conclusion and Future Work

Any public blockchain based IoT access control system inherits the shortcomings of a public blockchain. On the other hand, permissioned blockchain can overcome these limitations with a very small number of peers whose identities are verified but are not necessarily trusted to each other. This motivates us to design and implement a permissioned blockchain based access control system for IoT in Hyperledger Fabric. By using attribute based access control (ABAC) as our access control model, we can provide fine-grained access control while IoT devices share resource (data) with external parties. By running experiments in an IoT testbed, we have fine-tuned our blockchain network for access control by finding the optimum parameter values for our network (block timeout = 1s, block size = 20 at 40 transactions per second). Using the optimum parameter values, we show that our access control system can serve access request of IoT resources much faster than public blockchain. The ABAC policy evaluation algorithm we have used is NP-complete. Thus, in our future work, we plan to reduce the latency even more by optimizing this algorithm.

References

[1] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "Oauth-iot: An access control framework for the internet of things based on open standards," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 676–681, IEEE, 2017.

[2] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[4] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, pp. 5943–5964, 2016.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, D. De Caro, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, p. 30, ACM, 2018.

[6] S. Rouhani, V. Pourheidari, and R. Deters, "Physical access control management system based on permissioned blockchain," *arXiv preprint arXiv:1901.09873*, 2019.

[7] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 523–533, Springer, 2017.

[8] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[9] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 618–623, IEEE, 2017.

[10] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *The second international conference on Internet-of-Things design and implementation*, pp. 173–178, ACM, 2017.

[11] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: challenges and solutions," *arXiv preprint arXiv:1608.05187*, 2016.

[12] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[13] S. Farrell, R. Housley, and S. Turner, "An internet attribute certificate profile for authorization," tech. rep., 2010.

[14] "Telosb motes, 2017." http://www.memsic.com/info/aceinna-landing.cfm?nu=/wireless-sensor-networks.

[15] "Hyperledger fabric v1.3." https://hyperledger-fabric.readthedocs.io/en/release-1.3/whatsnew.html.

[16] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 264–276, IEEE, 2018.