Efficient Attribute-Based Smart Contract Access Control Enhanced by Reputation Assessment

Yang Liu
Northeastern University
Qinhuangdao, China
liuyang@neui.net

Terry Guo
Tennessee Technological University
Cookeville, Tennessee, USA
nguo@tntech.edu

Zhe Chen
Northeastern University
Shenyang, China
chenzhe@mail.neu.edu.cn

Xueying Jiang
Northeastern University
Qinhuangdao, China
xueying.jiang@163.com

Abstract—Blockchain’s immutability can resist unauthorized changes of ledgers, thus it can be used as a trust enhancement mechanism to a shared system. Indeed, blockchain has been considered to solve the security and privacy issues of the Internet of Things (IoT). In this regard, most researchers currently focus on the realization of various access control models and architectures, and are working towards making full use of the blockchain to secure IoT systems. It is worth noting that there has been an increasingly heavy pressure on the blockchain storage caused by dealing with massive IoT data and handling malicious access behaviors in the system, and not many countermeasures have been seen to curb the increase. However, this problem has not been paid enough attention. In this paper, we implement an attribute-based access control scheme using smart contracts in Quorum blockchain. It provides basic access control functions and conserves storage by reducing the number of smart contracts. In addition, a reputation-based technique is introduced to cope with malicious behaviors. Certain illegal transactions can be blocked by the credit-assessment algorithm, which deters possibly malicious nodes and gives more chance to well-behaved nodes. The feasibility of our proposed scheme is demonstrated by doing experiment on a testbed and conducting a case study. Finally, the system performance is assessed based on experimental measurement.

Index Terms—Internet of Things (IoT), access control, blockchain, smart contract, Quorum, Attribute-Based Access Control (ABAC).

I. INTRODUCTION

There is no doubt that the Internet of Things (IoT) is pervading every aspect of our daily life. With the sharp increase in the number of smart devices, IoT has influenced many vertical domains: homes, transportation, health, buildings, cities, industries, and even our human bodies [1]. However, the resource and capability constraint of IoT devices and complex IoT network structures bring up significant security and privacy concerns. Adversaries can gain illegal access to the devices to get crucial data or take over the control of the devices to initiate malicious actions [2]. The consequence can be privacy leaking, system failure and even body injuries. Access control is an important line to guard the IoT systems against security and privacy threats. However, traditional access control schemes heavily rely on centralized authorities for access validation at risk of single point of failure, without users’ control over their own data [3]. It is essential and urgent to find effective ways to secure IoT access control.

A blockchain is a decentralized security framework with transparent, Byzantine fault-tolerant, immutable, and chronological ledgers maintained by distributed users. In recent years, it has received broad attention from both academia and industry because of its decentralized management mechanism, and many researchers have applied it to the field of access control to replace the centralized authorization entity with the trusted platform [4]. Dorri et al. proposed a blockchain-based access control architecture that consists of smart homes, an overlay network and cloud storage [5]. They added a policy header in the blockchain block to store access control policies and authorize devices. ControlChain is another architecture to provide access control in IoT [6]. It realizes all functions through the cooperation of four different blockchains. Djilali et al. used hierarchical architecture to alleviate the computation overhead in their scheme and developed a new distributed access control system for IoT using blockchain [7]. In particular, in a specific smart factory scenario, Wan et al. also used the blockchain to implement an access control system in hierarchical structure. They introduced the whitelist mechanism, asymmetric encryption mechanism and other methods to improve the security and privacy. [8]. All the above schemes focus on architecture design and are lack of expression capability to describe access rights in sufficient details in defining various functions. Fortunately, the blockchain smart contracts can solve this problem.

A smart contract is an agreement that can be self-executed without involving a third party. This key concept was first introduced by Ethereum in 2013 [9]. Most smart contracts use the Turing-complete language which can be used to implement complex logic and applications, such as various access control mechanisms and architectures. Because of this, one can see many recent research reports based on smart contracts to achieve access control in untrustworthy IoT environments. Novo et al. proposed a six-part architecture along with an access management system implemented by a single smart contract [10]. They focused on addressing scalability problem and only provided a few access management functions. Huh et al. proposed a simple contract system consisting of three individual contracts to track electricity usage in terms of meter value as well as policy values of air conditioner and lightbulb, respectively [11]. The smart contract expression capability can be fully utilized by describing various access control
models in every detail. Riabi et al. chose a model that is a combination of Capability-BAC (Capability based access control) and Identity-BAC and used the smart contract to store and manage an access control list (ACL) [12]. There are also several more-sophisticated contract systems proposed by researchers recently. The contract system proposed by Zhang et al. consists of multiple access control contracts (ACCs), one judge contract and one register contract to achieve distributed and trustworthy access control for IoT [13]; and it also uses ACL Wang et al. implemented a traditional ABAC (attribute-based access control) architecture by using smart contract [14]. Their scheme reduces storage occupancy but lacks dynamic access right validation compared to the work in [13]. MedRec is also a three-contract system being concerned with various problems of electronic medical records in practical scenarios [15]. These contract systems either aim to provide sufficient access control capability or focus on specific scenarios. Since little consideration is given to storage footprint, many proposed schemes can result in rapid growth in blockchain volume.

Although the blockchain is relatively secure, it still carries its own risks. There has been intensive research on improving blockchain security by considering mechanisms such as users' reputation. LVChain proposed by Yu et al. is a blockchain-based architecture for IoT access authorization and has some advantages enabled by its vote-based consensus algorithm [16]. Huang et al. presented a blockchain system with credit-based consensus mechanism for Industrial Internet of Things (IIoT) [17], where credit value is used in Proof of Work (PoW) mechanism to make adjustment between efficiency and security in consensus depending on whether or not a node is honest or malicious. However, these schemes are all designed at blockchain level (e.g., blockchain network, block header or consensus algorithm), instead of contract level. Still, research on detecting malicious behaviors in the blockchain access control systems is quite rare. Efforts on this particular matter are typically from the perspective of dynamic access control and use historical behavior and other contents to make decisions on granting new access. For example, Hwang et al. proposed a dynamic access control scheme to fit the dynamic environment of IoT [18]. In their scheme, dynamic policy creation upon receiving a data requesting has to be done manually by the manager. Wan et al. also presented the idea of dynamic access in their research [8]. Furthermore, in [13] the authors even proposed to use malicious behavior detection in their contract system.

To address the storage issues mentioned above and further improve malicious behavior detection and processing capability in the contract system, we design and prototype an access control system based on smart contract, and integrate a credit-based misbehavior detection method to better protect the IoT system against security and privacy threats. Our major contributions can be summarized into the following three aspects:

1) A new smart contract architecture for IoT access control is designed by leveraging the ABAC model. It can effectively reduce storage requirement and curb the rapid growth of blockchain volume. This work is an extension of smart contract-based framework proposed by Zhang [13]. Different from theirs, our proposal consists of multiple ACCs, one management contract, and one reputation contract. In particular, they use ACL to achieve access control and we choose ABAC model.

2) We have designed a reputation assessment mechanism and used it to discourage and deter malicious behaviors during access control process, which is in favor of well-behaved nodes and the overall security is enhanced. The reputation calculation is similar to that in [17], but we count the number of behaviors in the algorithm, instead of the time elapsed. In addition, we block certain number of requests to execute a penalty, while in [17] a penalty is related to the difficulty value of the PoW algorithm.

3) To validate our proposed scheme and assess its performance, we have designed and examined a prototype based Quorum blockchain, conducted a case study of a supply chain, and made experimental measurement for performance assessment.

The rest of this paper is organized as follows. Section II presents an overview of our proposed IoT system, the blockchain platform and access model we use in this paper. Section III describes the access control framework in details, including three types of smart contracts. In Section IV, we show the feasibility of our scheme using experiments and the case study. Section V gives performance assessment results, followed by conclusions in Section VI.

II. System Overview

A. Platform

Some researchers have already analyzed the blockchain platforms that meet the requirements for IIoT. According to [19], it is believed that the top factors for platform selection include protocol of block creations, consensus for block adding and smart contract support; and Hyperledger stands out if giving only four options: Corda, Hyperledger Fabric, Tendermint and Symbiont. In our project, we choose Quorum [20] because of the following three reasons. The first is that Quorum supports multiple consensuses without degrading performance. As a consortium blockchain, Quorum provides three consensuses (Raft, Istanbul Byzantine Fault Tolerant (IBFT) and Clique POA) and can process more transactions than others in a unit time, which are highly preferred for the IoT environment. We have noticed that Ethereum and PoW consensus is popular, but PoW consensus does not yield satisfactory performance for IoT applications. The second reason is that Quorum is fully based on the official Go implementation of Ethereum protocol, which makes it inheriting some advantages from Ethereum. The active Ethereum community is beneficial for solving technical problems and continuously applying new theories. Different from some other blockchain platforms, Ethereum has diverse architectures and can support a wide range of clients. It is possible to run the software on various IoT devices with different hardware architectures to serve
different types of clients. In addition, the deep integration with Swarm [21] enables off-chain storage of unimportant data. The last reason is privacy protection. In general, defining policies for access control on the blockchain is not wise, since the policies should not be seen by every participant [22]. However, in our opinion, the policies are necessary but should have auditable and immutable features. Although there have been a number of studies that use cryptography mechanisms or other approaches like multi-party computing to address privacy issues [23], Quorum offers a simpler approach—private transaction manager. It is able to keep transaction privacy between the involved participants. In this way, transaction and smart contract privacy can be preserved by preventing unrelated participants from accessing the transaction content.

B. Architecture

IBFT is selected as our consensus algorithm. There are two types of nodes with IBFT: validators and non-validators. Along with the two types of nodes, the system also contains numerous IoT devices (e.g., sensors and actuators). As illustrated in Fig.1, all of these components form three layers of the IoT system considered in this paper.

![Three-layer system architecture](image)

Fig. 1: Three-layer system architecture

The validators are devices (e.g., servers) that are powerful in terms of computation and storage, and responsible for maintaining the blockchain and reaching the final consensus state. The system can tolerate at most $F$ faulty nodes in an $N$-validator network, where $N = 3F + 1$, implying there should be at least four validators to tolerate one Byzantine fault [24]. The non-validators are normal nodes within the blockchain network, and they may be IoT gateways or user devices in our work. Each IoT gateway connects a cluster of third-layer IoT devices via wireless or wired connection. User devices (e.g., PCs, laptops) are used to connect and operate servers and IoT gateways. A large number of IoT devices are at the third layer, including 1) sensors for perceiving the environment and sending the acquired data to the storage devices, and 2) actuators acting according to their received control commands. The IoT gateways serve as agents for these IoT devices in the sense: 1) each gateway creates an independent blockchain account for each of its child devices at the third layer; all interactions like deploying smart contracts or calling a function in the contract would be executed through these accounts; and 2) they send requests from the blockchain to a device or return responses from the device to the chain through some middleware.

As shown in Fig.1, the blockchain network is overlaid on top of the IoT network, where the IoT devices are not part of the blockchain, and this is rational because the IoT network is typically resource-constrained. Indeed, most of such devices are difficult to run as Quorum clients, participate in consensus process and communicate timely. In our system, Quorum clients are running at all blockchain nodes but the IoT end devices.

C. Access Control Model

Quaddah et al. gave a review of access control in IoT and classified different solutions into four layers: objectives, models, architecture and mechanism [4]. From this point of view, the implementation work based on smart contract includes selecting models and making improvement at the architecture and mechanism layers. In typical IoT applications, each device may have some resources (e.g., data, storage space or others) that are needed by other devices. A device can act as a requester when it wishes to access the resources of other devices. To abstract the access control problem, we adopt the ABAC model [25] and define the following sets: objects $O$, subject set $S$, resource set $R_o$ and attribute set $A_s$ (e.g., device type or other customized attributes). Each object $o \in O$ has some resources $r_o \in R_o$, and each subject $s \in S$ has some device attributes $a_s \in A_s$. For each resource $r_o \in R_o$, we can create some policies to allow only the subjects with specified attributes $a_s$'s to access the resources. Whether the policies are predefined or not, all access requests can be checked automatically. For better implementation, we define device attributes in a Management Contract (MC) and set policies in the ACC of every device.

There are some other attribute-based access control schemes using blockchain for IoT. Ding et al. proposed a novel attribute-based access control scheme for IoT systems [26], but they mainly used blockchain technology to record the distribution of attributes defined by the central authority in advance. Their scheme is theoretically well defined and more like a cryptography protocol of attribute distribution. The authors’ focus is on the distributed and non-tampering features of the blockchain, instead of making full use of the potential of smart contracts. In contrast, [22] presents a practical application of ABAC model for the healthcare IoT environment. Although the proposal is also based on smart contract, the access control policy is not part of the smart contract. The policy is implemented and executed off-chain, with consideration of delay and privacy. We believe sufficient throughput can be achieved if Quorum or some other platforms (e.g. IOTA [27]) are employed. Securing the policy is also necessary, therefore we define policy in the smart contract rather than somewhere off-chain. The ABAC model can be well implemented by the smart contract, since it is able to describe every detail needed for adapting to the time-varying IoT environment.
D. Threat Models and Proposed Countermeasures

For the sake of precise and clear description of our scheme, we consider the following three threat models:

1) Threat model 1: A malicious node sends excessive number of requests within a given duration (counted in blockchain blocks) to increase its chance of successful access or cause network congestion intentionally.

2) Threat model 2: A malicious node violates normal policy items.

3) Threat model 3: A malicious node violates importance policy items, such as sending a request from an erroneous place or in a wrong time.

There are two kinds of contract cooperation in the system for detecting and treating these malicious behaviors. All malicious behaviors are detected inside the contract that makes the access decision, and then the access behavior information is submitted to the contract dedicated for reputation assessment to perform penalty or toleration by considering the requester’s historical behaviors.

E. Configurations

To apply the Quorum platform in our scheme, we need to make some addition and adjustment as follows.

1) All validators’ accounts are generated when the blockchain is created. If there is a new node that wants to be a validator, a voting process is initiated and a new validator is elected if majority of current validators vote “pass”. A voting-process is not necessary for accepting a non-validator node.

2) All devices in the system are differentiated by their blockchain accounts, rather than some associated unique identifiers.

3) In Quorum, any transaction that changes the state does not actually consume gas because all the spending will be returned to the associated node after execution. However, a device does need some balance in its account if it wants to send a transaction. A node creating a block does not get a reward in IBFT, and all the balance is allocated when we edit the genesis.json file before the geth init command is executed. Therefore, we make a design such that when a new node (validator or non-validator) joins the blockchain network, the specified account called bank must transfer a certain amount of ether (Ethereum currency unit) to the new node. When a new device is connected to a gateway, the gateway account must transfer some ether to the device account newly generated.

III. Access Control Framework

Our access control framework is built on smart contracts. As illustrated in Fig. 2, the system consists of three types of contracts (MC, RC and ACC). The functions can be divided into two categories: basic functions of access control such as attributes management, policies management, access request processing and malicious behavior detection. The first category of functions are accommodated by ACCs and MC, and they are responsible for managing attributes and policies, and making decisions upon receiving access requests. The main functions of the second category are supported by RC (Reputation Contract), and responsible for managing trustworthiness and issuing reward and penalty.

A. Access Control Contract

We create and deploy an ACC for each device in the system. An ACC is responsible for managing resources, environment attributes and policies, and processing access requests related to the device. When a new device is connected to a gateway, its account will be created automatically and it will receive some ether from the gateway. Then, the gateway will create an ACC for this device and deploy it in the blockchain using the device’s own account. The contract addresses of MC and RC need to be passed using constructor when the ACC is deployed. Note that there are only one MC and only one RC in the whole system, and their addresses can only be updated by the manager of the device. After being deployed, the ACC must be registered in MC. This process will be discussed below in Management Contract subsection.

There are three global environment attributes in our system, namely minInterval, threshold and algorithm. minInterval is the minimum allowable interval (in seconds) between two successive requests. threshold is a number of requests in a given period of time, such that exceeding this threshold is judged as a malicious behavior. algorithm is used to determine an outcome when policy items conflict, and the results are either “denyoverrides” or “allowoverrides”; the former means that as long as a single policy item is not satisfied, the access request is denied, while the latter is the opposite: as long as one policy item is satisfied, the user is allowed to access [14]. In addition, we define two requester-specific variables to record the subject’s state. ToLR is the time at which the last request occurs, and NoFR is the number of requests generated by a specific subject within a given period of time. All subjects have their own state variables.

The resource attributes of a device are also managed by its ACC. We use $AttrV = G(r_o, AttrN)$ to define them, where $G$ represents the mapping, and $AttrV$ and $AttrN$ are the value and name of the resource attribute, respectively. There is no limit on the number of resource attributes, and also a resource can have no attribute.
Defining and managing access control policies related to the device’s resources is the main function of ACC. For each action \( ac \) (e.g., read, write, etc.) associated with a resource \( r_o \), there can be a corresponding policy. A policy is built on basic policy items for the given attributes, and each policy item is defined using the following five terms:

- **attrOwner**: the owner of the attribute; it can be subject or object;
- **attrName**: the attribute name in current condition;
- **operator**: the operator between the attribute name and attribute value; it can be \( >, <, \) or \( = \);
- **attrValue**: the attribute value that needs to be satisfied.
- **importance**: the importance level of the condition, with 0 as its default corresponding to the least importance.

The logic value of each policy item is determined as follows. We get actual attribute according attrOwner and attrName, then compare it with attrValue based on the operator defined above, leading to an logic value which is used as the logic value of respective policy item. All policy item values are combined using the default logic operator \( \text{AND} \) to form the policy. There is no other logic operator like OR in the current version.

The most important function of ACC is decision making on accepting or rejecting an access request. The system receives two strings describing resource and action as input parameters. The address of a requester is used as the subject address automatically. Solidity (a Contract-Oriented Programming Language) does not provide a real timestamp, and the timestamp we use is provided by the block when it is collected. Therefore, there is still some security risk because a miner could make influence on the timestamp. However, in our scheme, the miners at the top layer do not involve specific access operations, so they could not make much influence. The major part of algorithm in pseudo code for access control decision making is given in Algorithm 1. In line 2-3, a penalty time (number of blocked access attempts) is given by MC. If the time point is in the future, the request will be rejected directly, and all the changes made inside the transaction will be reverted; here we use require (a keyword in Solidity) to judge on a condition. The portion of line 4-10 detects whether there are frequent requests. Then, in line 11-25, policies and attributes from MC and algorithm (one of the three global environment attributes) are used to determine whether the policy check is passed. behaviorID is used to mark the result type, and the final result is given in line 38-41 according it. If no policy related to the resource and action is defined, the final result will be NotDefine. Also, it is worth knowing that the result will be submitted to RC for further processing when it emits as an event.

We provide all the basic functions (Add, Delete, Get and Update) for management of policy and resource attributes. Besides, considering that each ACC represents a device and IoT devices are frequently added and removed, we implement deleteACC() function which performs the self-destruct operation. Note that only the manager of the device can add new

**Algorithm 1: Access control decision making**

**Input**: resource, action

**Output**: subject, result, behavior ID, block number

```python
subject ← requester address
/* Define global variables in there, include behaviorCheck, policyCheck, behavior ID, result[] and finalResult */
if mc.getTLFB(subject) ≥ block.number then
    transaction revert
if block.number – ToLR ≥ minInterval then
    NoFR++
if NoFR ≤ threshold then
    behaviorCheck ← true
else
    NoFR ← 0
behaviors[subject].ToLR ← block.timestamp
if The number of policies = 0 then
    return "NotDefine"
for each policy p[i] related to resource and action and !behaviorCheck do
    if p[i].attrOwner = "subject" then
        attrValue ← mc.getAttribute(subject, p[i].attrName)
    else
        attrValue ← resource.attrValue
    if attrValue and p[i].attrValue not satisfied p[i].operator then
        currentPolicy ← true
    if currentPolicy then
        result[1]++
        if p[i].importance != 0 then
            result[2]++
        else
            result[0]++
        if algorithm = "denyoverrides" and result[1] > 0 then
            policyCheck ← true
        if algorithm = "allowoverrides" and result[0] = 0 then
            policyCheck ← true
        if behaviorCheck and !policyCheck then
            behaviorID = 1
        if !behaviorCheck and policyCheck then
            if result[2] = 0 then
                behaviorID = 3
            else
                behaviorID = 2
```
Policy items, update or delete existing policies, and delete the ACC.

B. Management Contract

MC is deployed upon the blockchain is created. Its main role is to manage the information of RC and device (role as requester) attributes.

When a device joins the blockchain network, in addition to deploying the ACC associated with it, its attributes are also needed to be register in the MC. The eight fields of a lookup table we use for defining and searching these attributes are defined as:

- `isValued`: This field is used for repeatability check, if a device is registered, the value is true.
- `managerAddress`: For the gateway, it is the blockchain account address of the gateway; for a device, it is the blockchain account address of the gateway the device belongs to;
- `scAddress`: The smart contract address of the ACC associated with a device.
- `deviceID`: The UUID of device.
- `deviceType`: Device type, e.g., Loudness Sensor.
- `deviceRole`: Device role, e.g., validator, manager or device.
- `TIFB`: The last forbid block, 0 if unblocked.
- `customized`: Attributes that can be customized; the number of these attributes can be zero.

The first seven are fixed attributes, and the last one is a customizable attribute. All fixed attributes must be set when the device is registered, and the customizable attributes can be added afterward. An example of a lookup table is shown in Table I. A key field of the lookup table is the blockchain account address which is the unique identity of a device in the system.

The structure we use to manage RC includes three fields: `isValued`, creator and `scAddress`. The first one is used for repeatability check, the second one is the node account who has created and deployed the RC, and the last one is the address of the RC. In contrast to the management of device attributes that includes four basic operations (Register, Delete, Get and Update), the management of RC includes only three operations (Register, Get and Update). Deletion of RC is not allowed because it will cause a system crash.

C. Reputation Contract

Following the deployment of the MC, the RC is created, deployed, and registered in the MC. The main function of RC is to use the behavioral information submitted by ACCs to calculate the reputation level, so as to reward or punish the devices. We have designed an algorithm to calculate reputation based on all current and previous behaviors. The credit of device $i$ is defined as

$$C_i = \lambda_1 C^P_i - \lambda_2 C^N_i$$

which is composed of two parts: $C^P_i$, the positive part due to normal behaviors, and $C^N_i$, the negative part due to malicious behaviors; and they are weighted by $\lambda_1$ and $\lambda_2$, respectively.

The negative part of the credit value $C^N_i$ actually represents a penalty, and it is related to the number and type of malicious behaviors in the past, importance of the policy violated and so on. Taking into account implementation limitation, the penalty function is given by

$$C^N_i = \sum_{k=0}^{m_i-1} \frac{\alpha_k}{m_i - k}$$

where $m_i$ is the total number of malicious behaviors to be considered for device $i$, $k$ is the chronological index of a malicious behavior in the past, with $k=0$ for the earliest behavior, $\alpha_k$ is the penalty coefficient of the malicious behaviors $k$, representing a severity level within the range of $1 - 10$. Note that $\frac{1}{m_i-k}$ in the formula acts as a weighing factor that varies depending on when a malicious behavior appears, and the impact of each malicious behavior decreases gradually over time, but it will never disappear. The malicious behaviors under consideration fall into three types and they are assigned IDs 1, 2 and 3, respectively:

1) **High frequency request**: its behavior ID is 1.
2) **Policy check failed**: its behavior ID is 2.
3) **Importance check failed**: e.g., if the “importance” field of a policy item is not 0 (abnormal), this policy item check fails, corresponding to a more serious situation; its behavior ID is 3.

Any behavior cannot be judged as belonging to more than one type of malicious behavior at the same time. If there is ambiguity in categorizing a phenomenon, labelling it with the ID corresponding to a higher priority. For example, if a behavior can be classified as both the second or the third types, label the behavior with ID 3.

The positive part of the credit value $C^P_i$ can also be called a reward due to normal behaviors, and it is defined as

$$C^P_i = \min(C^P_{i,max}, \sum_{k=k_1}^{l_i} \omega_k)$$

where $C^P_{i,max}$ is a pre-defined upper limit on $C^P_i$ for preventing unlimited accumulation of rewards, $l_i$ is the total number of normal behaviors to be considered for device $i$, $k$ is the chronological index of a normal behavior in the past, $k_1$ is the index of the first normal behavior after the last penalty
was made, and $\omega_k$ is the weight on the normal behavior $k$. Currently only one type of normal behavior, i.e., authorized access is considered for testing, and its behavior ID is 0. Obviously, this list can be expanded to include more normal types.

Every behavior submitted automatically updates its respective behavior list, and then the value of penalty or reward is re-calculated to determine a new credit value. If this credit value is less than 0, the number of forbid blocks will be calculated, and the corresponding TLFB attribute of the device in MC will be updated; at the same time, $k_1$ will be updated. When $C_i^P$ is calculated next time, the normal behaviors before the index of $k_1$ will not be counted again. In contrast, malicious behavior records are never emptied, so every penalty calculation needs to count all previous malicious behaviors.

The penalty made by the system is to block device access requests, i.e., during a blocking period all access requests from that device will be denied. Note that the blocking period is not defined in actual timestamps, but a blocking time length measured in blocks, calculated in the following exponential function:

$$
forbid = \beta^{-C_i}, C_i < 0
$$

where $\beta$ (> 0) and takes 2 in our scheme. Since $C_i$ is always an integer due to practical restriction, $forbid = 2^{-C_i}$, ($C_i < 0$) is an integer as well. It needs to be pointed out that, rather than a duration, the penalty attribute used in MC and ACCs is the block number of the last forbid block, denoted by TLFB, calculated by adding $forbid$ to the block number of the block containing the access transaction.

There are some other issues that are worth mentioning. Firstly, the credit of a device should not be related to the activity level of the device. A device may not initiate a single request over a short period of time, but this situation does not necessarily correspond to a malicious behavior, and the device credit value should not be affected. Secondly, Solidity, the language used for Ethereum smart contract, does not support floating-point in performing definition and calculation, so we use a library that provides Quad-precision floating-point operations. Finally, access requested before the last forbid block will be reverted directly, as mentioned in section III-A.

Fig. 3 is an example of the whole behavior list. The list is organized in four parts: normal behavior list (LegalBehaviors), malicious behavior list (Misbehaviors), beginning index of LegalBehaviors (begin) and last forbid block (TLFB). The two most important parts, LegalBehaviors and Misbehaviors, have the same structure consisting of three parts (behavior ID, block number corresponding to the moment when the behavior appears, and current weight value). The behavior list can be found using subject address (as key).

We provide a reputation calculation algorithm shown in Algorithm 2. In line 3, the behaviors are added in behavior list. In line 4-13, credit is calculated according to formula (1). In line 14-19, TLFB, the calculated block number of the last forbid block is used to interact with MC. We still use Solidity events to return values, where subject is the requester address, behavior is the description of behavior passed in, bn is the block number when the behavior appears, credit and forbid are two values obtained during the algorithm runs. The function that implements this algorithm can only be called by ACC, which prevents interference made by outside users.

Only a single RC exists, so there is no self-destruct operation—even the owner of RC cannot delete the contract. There is another function named getLastError() that can be used to return the latest behavior. When we call this function, it will read the last item of the behavior list based on a behavior type number.

### IV. Function Validation

To demonstrate the feasibility of our solution, we have implemented a smart contract template applicable for three types of contracts. We will show the whole process of access control and give the results based on a supply chain use case.

#### A. Testbed Implementation

Our experimental testbed includes a laptop and two Raspberry Pi (3B and 3B+) modules, and the Quorum geth client is installed on such devices to emulate blockchain nodes. The details of these devices are shown in Table I. Four nodes as
Algorithm 2: reputation compute

\begin{algorithm}
\textbf{input :} subject, behaviorID, block number
\textbf{output :} subject, behavior, block number, credit, forbid

1. if requester is not ACC then
2. \hspace{1em} transaction revert
3. Add behavior to behavior list according to behaviorID
4. misLen ← the length of malicious behavior list
5. for i ← 0 to misLen do
6. \hspace{1em} $w_i \leftarrow \text{MisBehavior[i].currentWeight}$
7. \hspace{1em} $C^N \leftarrow C^N + \frac{w_i}{\text{misLen} - i}$
8. legLen ← the length of legal behavior list
9. for i ← begin to legLen do
10. \hspace{1em} $C^P \leftarrow \text{LegalBehavior[i].currentWeight}$
11. if $C^P > C^P_{\text{max}}$ then
12. \hspace{1em} $C^P \leftarrow C^P_{\text{max}}$
13. credit ← $\lambda_1 C^P - \lambda_2 C^N$
14. if block.number > TLFB and behaviorID \neq 0 and credit < 0 then
15. \hspace{1em} if LegLen > begin then
16. \hspace{2em} begin ← legLen - 1
17. \hspace{1em} forbid ← $\frac{-credit}{\text{forbid}}$
18. TLFB ← block.number + forbid
19. mc.updateEndBBN(subject,TLFB)
20. Trigger event isCalled(subject, behavior, bn, credit, forbid)
\end{algorithm}

the validators are running on the laptop, distinguished by their port numbers. One of the nodes, called Judger is responsible for deploying and managing the RC. In a real environment, the Judger should be an authority, such as a government agent or a trusted enterprise. Another one named Manager deploys and manages the MC. The Manager also acts as a bank and is used for transferring ether to a device when it joins the network for the first time. The Raspberry Pi modules act as the gateways (non-validators). In blockchain, a gateway represented by the first account of its geth clients. As an agent of its child devices, the gateway creates an account for each device connected to it.

Now, let us consider the access control issue between the two IoT devices on the pallet and truck, respectively. The device on the pallet is managed by gateway 1 (emulated by the Raspberry Pi 3B) and serves as a subject. It needs to send a request to verify whether the truck is a correct one. The device on the truck is managed by gateway 2 (emulated by the Raspberry Pi 3B+) and serves as an object. It is worth noting that for cost-effectiveness, we simply allocated two accounts on the testbed to represent the IoT devices on pallet and truck, instead of using two real devices physically. Of course, in a real-world supply chain, the process of data reading and operations requires underlying intermediate code (middleware) which is not implemented on our testbed.

To speed up our development work, we use Quorum Wizard command line tool to set up a small Quorum network. We utilize Remix to write and compile smart contracts, and use Truffle to test functions. Besides, we adopt the deploy & plugin in Remix to deploy contracts, and use web3.js to interact with the corresponding geth clients. We use cakeshop to visualize the blockchain and contract status. Finally, we create some JavaScript files for sending access requests and monitoring the results through a blockchain event.

B. Experiment and Testing Result

A supply chain is typically an untrustworthy environment, therefore it is an ideal use case to test our scheme. Specifically, let us consider a simple use case of supply chain: moving bananas on pallets by truck from a farm to a supermarket distribution center [23]. In this scenario, the pre-installed device on each pallet checks for whether it is loaded into the right truck, or unloaded at the correct warehouse at every stage of the journey. With the setting described above, we conduct an experiment to test the proposed scheme based on the supply chain use case. Ignoring some minor initialization process, below are the major experiment steps.

- Step 1: Quorum blockchain establishment and initialization.
- Step 2: The Manager deploys the MC, the Judger deploys the RC and registers it in the MC.
- Step 3: The Bank transfers 1 ether to the two gateway nodes respectively.
- Step 4: The two gateway nodes create accounts separately for the IoT devices, transfer 10^7 wei (the unit of Ethereum coin) to each device. Then, deploy ACC and register it in MC using the device accounts.
- Step 5: The manager of device on the truck (gateway 2) sets access policies in its ACC to allow the device on pallet to read the resource named ‘basicInformation’. The policy to be satisfied are shown in Table III.
- Step 6: The device on pallet (gateway 1) sends an access request.

We should add the attributes in Table III to the contract before sending a request. The access request and access monitor are implemented by using JavaScript, and the scripts are connected to Raspberry Pi through WebSocket rather than operating directly in the Raspberry Pi modules.

The parameters involved in the reputation function are defined as follows.

- In the penalty function, $\alpha_0 = 2, \alpha_1 = 3$ and $\alpha_2 = 5$, implying that the third type of malicious behavior should be given a higher weight;
- In the reward function, $\omega = 1$ and $C_i^{P_{\text{max}}} = 30$;
- In the process of calculating credit value, $\lambda_0 = 0.5, \lambda_1 = 0.5$.

In order to input these parameters, we adopt the method of decimal shift, since decimals are not allowed in Solidity. For example, to input 1.34, use integer numerator 134 and integer denominator 100, and then calculate the quotient from the two integers.
TABLE II: Hardware details

| Device            | CPU                     | Operating System | Memory | Hard Disk |
|-------------------|-------------------------|------------------|--------|-----------|
| HP OMEN laptop 15 | AMD Ryzen 7 4800H, 2.90 GHz | Ubuntu 20.04 LTS | 16 GB  | 512 GB    |
| Raspberry Pi 3 Model B Plus | quad-core ARM Cortex A53, 1.4 GHz | Raspbian (Buster) | 1 GB   | 32 GB (microSD card) |
| Raspberry Pi 3 Model B   | quad-core ARM Cortex A53, 1.2 GHz | Raspbian (Buster) | 1 GB   | 16 GB (microSD card) |

TABLE III: Policy Definition

| attrOwner | attrName  | operator | attrValue | importance |
|-----------|-----------|----------|-----------|------------|
| subject   | farm_name | =        | EarthDance| 0          |
| subject   | good_type | =        | banana    | 0          |
| object    | state     | =        | active    | 0          |

Fig. 4: One authorized access and three failed accesses from ACC Event.

Our source code for the three smart contracts, JavaScripts and experiment records is now available at Github. All smart contracts are audited through MythX tool to secure the contracts. A screenshot of requesting process is provided in Fig. 4, where both the access authorized and failed accesses are shown, and they all originate from the events triggered by the ACC.

V. PERFORMANCE ASSESSMENT

In this section, we present some performance assessment results based on experimental measurement. In particular, the effectiveness of our solution in suppressing the growth of the blockchain and the reputation mechanisms are demonstrated.

A. Gas Consumption

Gas consumption is such an important performance metric used in many solutions with Ethereum smart contracts. It is associated with real-world money and can accurately reflect the cost of each operation. Subject attribute management is handled by MC in our scheme and by subject contract (SC) in Wang’s scheme, and the comparison of MC and SC is shown by the first pair of bars in Fig. 5. Resource attribute management, policy management, access decision are handled by object contract, policy contract and ACC, respectively, in Wang’s scheme; while the three functions are all handled by just one ACC in our scheme. We compare the cumulative consumption of Wang’s three contracts and the consumption of our ACC, yielding the second pair of bars in Fig. 5. The statistical results shown here are based on transaction cost instead of execution cost.

Fig. 5: Gas consumption comparison

B. Access Time

Access time is another key performance metric, especially for some applications with highly real-time requirements. However, an access time actually depends on many factors, like hardware and network. Although the consensus choice and communication factor should have the greatest influence on the access time, the impact of inter-calling between the contracts cannot be ignored. Therefore, we measure the average access time of over 500 requests for the cases with and without reputation subsystem, and present comparison with Wang’s scheme in Table IV.

TABLE IV: Comparison of average access times (ms)

|                  | No reputation system | BBRAC      | Wang’s scheme |
|------------------|----------------------|------------|---------------|
| Average time     | 626.82               | 687.36     | 693.48        |
| Maximum time     | 990                  | 2710       | 1060          |
| Minimum time     | 570                  | 550        | 600           |

C. Storage Pressure

The speed at which the blockchain volume grows is more concerned with than many other issues. For blockchain solutions applied in IoT, massive devices often mean massive contracts, causing huge storage pressure on the blockchain. We have reduced the storage pressure in the following three
aspects at the contract level: access model selection, design of contract architecture, and the way of the function implementation. They are the key factors affecting a contract size.

In the following comparison, $n$ denotes the number of devices. In Zhang’s scheme [13], an ACC is deployed by a device (object device) who wants to control the access requests initiated by another device (subject device). The subject-object pair can agree on multiple access control methods, and each method is implemented by one ACC. With $n$ object devices (or subject devices), $n(n-1)$ subject-object pairs are generated. In other words, we need to deploy $n(n-1) \approx n^2$ contracts in the experiment. In Wang’s scheme [14], each policy contract is created by a user, i.e., one policy contract corresponds to one user. Assume each device belongs to a single user, so there are $n$ users and we need to deploy $n$ contracts. Based on contract deployment described above, we observe the actual blockchain volumes versus the number of devices (unit: KB) as shown in Fig. 6. The three schemes are Zhang’s that uses ACL to achieve access control, Wang’s and ours (BBRAC). Both of the latter two use ABAC model, and one can see that they outperform Zhang’s scheme in storage consumption. It is also observed that, as the number of devices increases, the storage occupancy of our scheme increases slightly faster than that of wang’s scheme, and this is mainly due to the inclusion of the reputation subsystem. We believe it is worth having such a subsystem at a small cost of additional storage occupancy.

![Fig. 6: Blockchain volume growth](image)

D. Reputation Subsystem

The reputation subsystem brings four mechanisms: reward, penalty, tolerance and alarm. Among them, alarm can be triggered by Solidity event if a threshold is defined in advance. The impact of other three mechanisms can be seen in Fig. 7 where the abscissa indicates the type of each access behavior, and each bar corresponds to current cumulative number of malicious behaviors. From a close watch at the plot, it can be seen that the initial legitimate accesses accumulate some reputation, then subsequent malicious behaviors lead to a decline in credit value, triggering penalty at certain point. In general, one can observe a few facts: 1) frequent malicious behaviors result in credit value decreasing, which is a desired outcome achieved through our design; 2) if repeatedly acting maliciously, the number of forbid blocks has an exponential growth trend; also the time that a device is forbid keeps growing, because the log of historical malicious behaviors is not cleared; and 3) when a malicious behavior occurs, if the device has accumulated a high credit value, the behavior could be tolerated. The observation 3 comes from our assumption that an accidental harmful behavior is usually due to an operational error and should be tolerable. To limit the tolerance of operational errors, we have intentionally set an upper limit on the credit value. The accumulation of legitimate access can increase the credit value, but it remains unchanged after reaching its upper limit.

![Fig. 7: Change of credit value and number of forbid blocks for a sequence of malicious and normal events.](image)

Finally, one should know that reward and penalty bring different results under two different situations, and both of these results are desirable. The first situation is that there are more requests to collect within a given duration (counted in blockchain blocks) than the blockchain can do. In this situation, the penalty mechanism can reduce the number of transactions generated by nodes that behavior maliciously, which, in consequence, gives more chance to well-behaved nodes. The second situation is that the blockchain is able to collect all requests generated within a given duration. In this case, the reduction of transactions due to the penalty mechanism helps suppress the growth rate of the blockchain.

We have conducted an experiment to test how much room (measured in transactions) can be saved for the well-behaved nodes. Access requests are sent by a node at a fixed time interval until the blockchain grows by 500 blocks, with the initial credit value being set to zero. Among the requests, the first fifty all exhibit malicious behaviors that fail to pass importance policy check. In a non-reputation system (without using the reputation algorithm), the number of requests sent over 500 blocks is 169, and all of these requests are collected successfully, regardless of their behavior types. In contrast, in a reputation system (enhanced by the reputation algorithm),
In this work, we focus on the access control in the IoT system and propose an improved smart contract-based framework. We design and implement the ABAC model using smart contracts and reduce the number of ACCs to ease the storage pressure on blockchain. A supply chain use case is considered and tested on a small testbed. The experimental results prove that execution times of some functions like attribute management are reduced. In particular, the growth rate of the blockchain volume is effectively curbed. The addition of the reputation subsystem tightens the security of access at the contract level. According to our experiment, the chance of successful collection of transactions from the well-behaved node is improved, thanks to the four mechanisms provided by the reputation subsystem.

A blockchain system may be regarded as a distributed database, but storing data on it is expensive. One natural solution used by many researchers is that only crucial data (attributes, policies or others) is stored on-chain and the rest of the data is stored off-chain. People have used cloud or distributed storage, like IPFS and Swarm as off-chain storage platforms, and these techniques can be employed in our proposed system. The usability of access control models is another topic to study, considering that the IoT environment is dynamic, and the attributes of devices can change over time. Compared with the traditional models developed over decades, such as CapBAC model that is also suitable for the IoT environment, and XACML language that is used to implement ABAC model, those based on smart contracts, including what we have proposed, are still in their infancy. The following are just a few that need to be researched in the future: automatic update of attributes, automatic discovery of attribute permission relationship, and improvement of smart contract implementation.

VI. CONCLUSION

REFERENCES

[1] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, “A survey on 5G networks for the internet of things: Communication technologies and challenges,” IEEE Access, vol. 6, pp. 3619–3647, 2018.
[2] M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” Future Generation Computer Systems, vol. 82, pp. 395–411, 2018.
[3] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, “Access control in the internet of things: Big challenges and new opportunities,” Computer Networks, vol. 112, pp. 237–262, 2017.
[4] J. Shi and L. Ru, “Survey of blockchain access control in internet of things,” Ruan Jian Xue Bao, no. 6, pp. 1632–1648, 2019.
[5] A. Dorri, S. S. Kanhere, R. Jurda, 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). IEEE, 2017, pp. 618–623.
[6] O. J. A. Pinno, A. R. A. Gregio, and L. C. E. De Bona, “Controlchain: Blockchain as a central enabler for access control authorizations in the IoT,” in GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–6.
[7] H. Benhadj Dijiali and D. Tandjaoui, “Efficient distributed authentication and access control system management for internet of things using blockchain,” in Mobile, Secure, and Programmable Networking. E. Renault, S. Boumerdassi, C. Leghrisi, and S. Bouzefrane, Eds. Cham: Springer International Publishing, 2019, pp. 51–60.
[8] J. Wan, J. Li, M. Imran, D. Li, and Fazal-e-Amin, “A blockchain-based solution for enhancing security and privacy in smart factory,” IEEE Transactions on Industrial Informatics, vol. 15, no. 6, pp. 3652–3660, 2019.
[9] Ethereum—a global, open-source platform for decentralized applications. Accessed: Apr. 3, 2020. [Online]. Available: https://ethereum.org/
[10] 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.
[11] S. Huh, S. Cho, and S. Kim, “Managing IoT devices using blockchain platform,” in 2017 19th International Conference on Advanced Communication Technology (WAICT), IEEE, 2017, pp. 464–467.
[12] I. Riabi, Y. Dhif, H. K. Ben Ayed, and K. Zaoutouiri, “A blockchain-based access control protocol for IoT,” in 2019 15th International Wireless Communications Mobile Computing Conference (IWCMC), 2019, pp. 2086–2091.
[13] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, “Smart contract-based access control for the Internet of Things,” IEEE Internet of Things Journal, vol. 6, no. 2, pp. 1594–1605, 2018.
[14] P. Wang, Y. Yue, W. Sun, and J. Liu, “An attribute-based distributed access control for blockchain-enabled IoT,” in 2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2019, pp. 1–6.
[15] A. Azaria, A. Ekbhaw, T. Vieira, and A. Lipman, “Medrec: Using blockchain for medical data access and permission management,” in 2016 2nd International Conference on Open and Big Data (OBD). IEEE, 2016, pp. 25–30.
[16] Y. Yu, S. Zhang, C. Chen, and X. Zhong, “A lightweight and vote-based blockchain framework for access control in the IoT,” in 2018 IEEE 4th International Conference on Computer and Communications (ICCC), 2018, pp. 870–874.
[17] J. Huang, L. Kong, G. Chen, M. Wu, X. Liu, and P. Zeng, “Towards secure industrial IoT: Blockchain system with credit-based consensus mechanism,” IEEE Transactions on Industrial Informatics, vol. 15, no. 6, pp. 3680–3689, 2019.
[18] D. Hwang, J. Choi, and K. Kim, “Dynamic access control scheme for IoT devices using blockchain,” in 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018, pp. 713–715.
[19] N. Teslya and I. Ryabchikov, “Blockchain-based platform architecture for industrial IoT,” in 2017 21st Conference of Open Innovations Association (FRUCT). IEEE, 2017, pp. 321–329.
[20] Quorum-enterprise ethereum client. Accessed: Apr. 3, 2020. [Online]. Available: https://docs.quorum.com/en/latest/
[21] K. R. Ozyilmaz and A. Yurdakul, “Designing a blockchain-based IoT with ethereum, swarm, and lora: the software solution to create high availability with minimal security risks,” IEEE Consumer Electronics Magazine, vol. 8, no. 2, pp. 28–34, 2019.
[22] S. Figueroa, J. Ahorga, and S. Arrizabalaga, “An attribute-based access control model in rfid systems based on blockchain decentralized applications for healthcare environments,” Computers, vol. 8, no. 3, p. 57, 2019.
[23] G. Zyskind, O. Nathan et al., “Decentralizing privacy: Using blockchain to protect personal data,” in 2015 IEEE Security and Privacy Workshops. IEEE, 2015, pp. 180–184.
[24] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” ACM, 1999.
[25] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, “Attribute-based access control,” Computer, vol. 48, no. 2, pp. 85–88, 2015.
[26] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, “A novel attribute-based access control scheme using blockchain for IoT,” IEEE Access, vol. 7, pp. 38431–38441, 2019.

[27] “IOTA,” accessed: June 22, 2021. [Online]. Available: https://www.iota.org/

[28] L. Evers and P. Havinga, “Supply chain management automation using wireless sensor networks,” in 2007 IEEE International Conference on Mobile Adhoc and Sensor Systems. IEEE, 2007-10, pp. 1–3.

[29] J. Benet, “IPFS-content addressed, versioned, p2p file system,” arXiv preprint arXiv:1407.3561, 2014.