# Smart Contract-Based Access Control for the Internet of Things

Yuanyu Zhang, *Member, IEEE*, Shoji Kasahara, *Member, IEEE*, Yulong Shen, *Member, IEEE*, Xiaohong Jiang, *Senior Member, IEEE*, and Jianxiong Wan

***Abstract*—This paper investigates a critical access control issue in the Internet of Things (IoT). In particular, we propose a smart contract-based framework, which consists of multiple access control contracts (ACCs), one judge contract (JC), and one register contract (RC), to achieve distributed and trustworthy access control for IoT systems. Each ACC provides one access control method for a subject-object pair, and implements both static access right validation based on predefined policies and dynamic access right validation by checking the behavior of the subject. The JC implements a misbehavior-judging method to facilitate the dynamic validation of the ACCs by receiving misbehavior reports from the ACCs, judging the misbehavior and returning the corresponding penalty. The RC registers the information of the access control and misbehavior-judging methods as well as their smart contracts, and also provides functions (e.g., register, update, and delete) to manage these methods. To demonstrate the application of the framework, we provide a case study in an IoT system with one desktop computer, one laptop and two Raspberry Pi single-board computers, where the ACCs, JC, and RC are implemented based on the Ethereum smart contract platform to achieve the access control.**

***Index Terms*—Access control, blockchain, Internet of Things (IoT), smart contract.**

## I. Introduction

THANKS to the rapid advance of communication and networking technologies (e.g., Wi-Fi, ZigBee, and Bluetooth), a growing number of objects (e.g., sensors, actuators, and smart devices) are being connected to the Internet nowadays, leading to the concept of the Internet of Things (IoT) [1], [2]. The ubiquitous interconnection of physical objects significantly accelerates data collection, aggregation and sharing in the IoT, making the IoT one of the most fundamental architectures for various promising applications such as smart healthcare, intelligent transportation, home automation, etc. [3], [4]. However, such interconnection may also incur crucial security issues into IoT systems, because adversaries can intrude into the systems to gain illegal access to the provided resources (e.g., data, services, storage units, and computing units) by simply deploying their own or compromising existing IoT devices [5], [6]. Thus, access control, which aims to prevent the illegal resource access from unauthorized entities, has been regarded as an increasingly vital research issue in the IoT for both academia and industry [7]–[9].

Traditional IoT access control schemes are mainly built on top of the well-known access control models including the role-based access control model (RBAC) [10], the attribute-based access control model (ABAC) [11], and the capability-based access control model (CapBAC) [12]. In the RBAC-based schemes, the access control is based on the roles (e.g., administer and guest) of subjects (i.e., entities that access resources) within an organization. By associating the roles with access rights (e.g., read, write, and execute) and assigning the roles to the subjects, the RBAC-based schemes can establish a many-to-many relationship between the access rights and the subjects [13], [14]. The ABAC-based schemes implement the access control based on policies, which combine various types of attributes, such as subject attributes, object (i.e., the entity that holds resources) attributes and environment attributes, etc., to define a set of rules expressing under what conditions access rights can be granted to subjects [15], [16]. In the CapBAC-based schemes, access rights are granted to subjects based on the concept of capability, which is a transferable and unforgeable token of authority (e.g., a key and a ticket), and describes a set of access rights for each subject [17], [18].

It is notable that, in the above schemes, validating the access rights of subjects is usually conducted by a centralized entity, which turns out to be a single point of failure. To address this issue, distributed CapBAC models have been proposed recently [19], [20], where the access right validation is performed by the requested IoT objects themselves rather than a centralized entity. However, IoT objects are usually with low capability and thus may be easily compromised by adversaries, so they cannot be fully trusted as the access right validation entities. As a result, the distributed CapBAC models may fail to tackle the access control problem in untrustworthy

Manuscript received January 31, 2018; revised April 3, 2018; accepted June 11, 2018. Date of publication June 15, 2018; date of current version May 8, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1400700, in part by the Japan JSPS under Grant 15H04008, and in part by the China NSFC under Grant U1536202, Grant 61571352, and Grant 61502255. *(Corresponding author: Yulong Shen.)*

Y. Zhang and S. Kasahara are with the Graduate School of Science and Technology, Nara Institute of Science and Technology, Ikoma 630-0192, Japan (e-mail: yyzhang@is.naist.jp; kasahara@is.naist.jp).

Y. Shen is with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China (e-mail: ylshen@mail.xidian.edu.cn).

X. Jiang is with the School of Systems Information Science, Future University Hakodate, Hakodate 041-8655, Japan, and also with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China (e-mail: jiang@fun.ac.jp).

J. Wan is with the School of Data Science and Application, Inner Mongolia University of Technology, Hohhot 010051, China (e-mail: jxwan@imut.edu.cn).

Digital Object Identifier 10.1109/JIOT.2018.2847705

IoT environments. Thus, a crucial question arises: how can we achieve distributed and trustworthy access control in the IoT? The answer may lie in the emerging blockchain technology, the key enabler behind modern cryptocurrency systems like the Bitcoin [21] and Ethereum [22]. The blockchain is initially created as a distributed and immutable ledger of transactions for cryptocurrency systems. Thanks to the invention of smart contracts (executable codes that reside in the blockchain), the blockchain has now evolved into a promising platform for developing distributed and trustworthy applications, and has attracted considerable attentions from researchers in the IoT community [23], [24]. Therefore, this paper aims to apply the smart contract-enabled blockchain technology to achieve distributed and trustworthy access control for the IoT.

Some initial work has been done on the blockchain-based access control. Dorri *et al.* [25] considered the access control issue in an IoT network with service providers, cloud storage, user devices, and smart homes, each containing a miner and multiple IoT devices. Each home miner maintains a local private blockchain with a policy header storing access control policies to control all the access requests related to the home, i.e., internal, incoming, and outgoing requests. However, Nakamoto [26] eliminated the critical proof-of-work process in the blockchain technology, resulting in an untrustworthy access control scheme. Notice that the main purpose of the blockchain in [25] is to serve as a distributed and immutable storage for access control policies, whereas the computing capability of the blockchain was largely wasted. The idea of using the blockchain to only store access control policies has also been adopted in [27] and [28]. Recently, the computing capability of the blockchain has been exploited in [29] for access control, where the blockchain plays the role of a decentralized access control manager. The authors used access tokens to represent access rights and the tokens can be delivered from one peer to another through transactions. When delivering a token, the sender embeds access control policies into the locking scripts of the transaction output. The receiver of the token must unlock the locking scripts to prove the possession of the token (i.e., the access rights to a certain resource). Using this scheme, a peer can be granted access rights by receiving a token, grant access rights to another subject by delivering a token, and access an object by spending a token. Although using locking scripts for access control is an excellent idea, the computing capability of locking scripts is significantly limited. Different from [29], this paper utilizes smart contracts to provide a much higher computing capability for achieving various access control methods. Notice that the idea of using smart contracts for access control has been adopted in [30] and [31], where, different from this paper, the main purpose of the smart contracts is to manage data records.

To address the limitations of the above works, this paper proposes a smart contract-based access control framework, which consists of multiple access control contracts (ACCs), one judge contract (JC), and one register contract (RC), to achieve distributed and trustworthy access control for IoT systems. In the framework, each ACC provides one access control method for a subject-object pair, which implements both static access right validation based on predefined access control policies and dynamic access right validation by checking the behavior of the subject. The ACCs also provide functions for adding, updating, and deleting access control policies. Once called by a subject for access control, the ACC will be run and verified by most participants in the system, ensuring the trustworthiness of the access control. To facilitate the dynamic validation of the ACCs, the JC provides a misbehavior-judging method, which receives misbehavior reports about the subjects from the ACCs, judges the misbehavior and returns the corresponding penalty. To manage the access control and misbehavior-judging methods, the RC registers the information (e.g., name, subject, object, and smart contract) of the methods and also provides functions to register a new method and update or delete an existing method. To demonstrate the application of the framework, we provide a case study, in which we employ the Ethereum smart contract platform to implement the ACCs, JC, and RC for the access control in an IoT system with one desktop computer, one laptop, and two Raspberry Pi single-board computers.

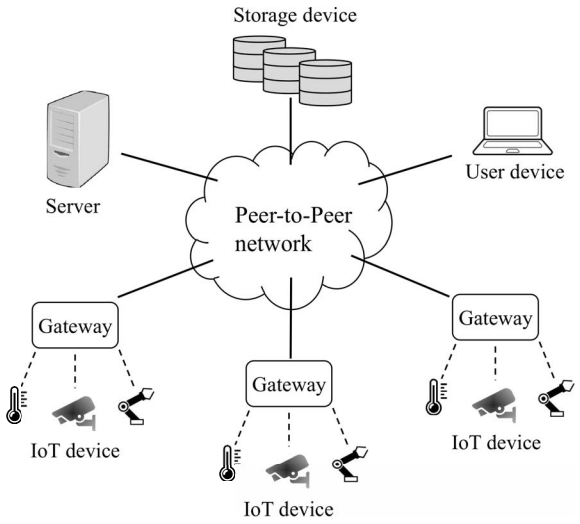



Fig. 1. Illustration of the considered IoT system.

The remainder of this paper is organized as follows. Section II presents the IoT system considered in this paper and Section III introduces the underlying smart contract platform for our access control framework. We introduce the distributed smart contract-based framework in Section IV and provide a case study for the proposed framework in Section V. Finally, Section VI concludes this paper.

## II. System and Security Model

### A. System Architecture

As illustrated in Fig. 1, the IoT system considered in this paper consists of a large number of servers, storage devices, IoT gateways, and user devices, which are connected together through a peer-to-peer (P2P) network. Also present in the system are numerous IoT devices (e.g., sensors and actuators), which are connected to the P2P network via the IoT gateways. The main roles of the peers are explained as follows.

1) *Server:* A server is a device or a cluster of devices that can interact with the IoT devices and storage devices

to provide a variety of services (e.g., smart home) for users. Interactions between the servers and other peers (e.g., IoT devices and storage devices) include collecting environmental data from the sensors, sending commands to the actuators to perform some operation, querying data from or storing data to the storage devices, etc.

2) *Storage Device:* A storage device can store data for other peers of the system, like the servers, sensors, and users. Various data can be stored on the storage devices, like the application data of the servers, environmental data gathered by the sensors, user profiles, etc.
3) *User Device:* A user device is a device (e.g., PCs, laptops, and smart phones) through which users can enjoy the services (e.g., checking the current temperature of his/her own house) provided by the servers and read data from or write data to the storage devices.
4) *IoT Gateway:* Each IoT gateway connects a cluster of IoT devices to the P2P network via short-range communication technologies like Bluetooth, Wi-Fi, and ZigBee, and serves as the service agent for these IoT devices at the same time.
5) *IoT Device:* The IoT devices in the system mainly include sensors, which can perceive environmental data (e.g., temperature) and send these data to the servers or storage devices for further use, and actuators, which can perform some operations (e.g., turning on the air conditioner) once receiving a command from users.

### B. Security Model

In typical IoT applications, each peer may have some resources (e.g., services, data, and storage space) that are needed by the other peers. Thus, access control must be implemented by all resource owners to prevent unauthorized use of their resources. For example, a server must be able to block the access requests from users who have not signed up, or the access requests from signed-up users for some services that they have not subscribed. To prevent illegal use of its storage space and data, a storage device must be able to restrict the access requests from unauthorized peers for querying data or storing data. An IoT device must be able to deny the unauthorized access requests for retrieving its data or controlling its actuators.

To abstract the access control problem in this system, we adopt the security model of access control matrix [12]. In this model, we define a set of subjects $S$, which are peers that wish to access the resources of other peers, and a set of objects $O$, which are peers that hold resources. Each object $o \in O$ has a set of resources $R_o$ (e.g., file and program) and each resource $r_o \in R_o$ is associated with a set of access rights $A_{r_o}$ (e.g., read, write and execute). For each subject $s$ and resource $r_o$, a mapping $\mathcal{G}(s, r_o) \subseteq A_{r_o}$ is defined to specify the access rights on $r_o$ that are granted to $s$. Notice that the access control matrix provides only an abstract characterization of the access control problem. To implement the access control matrix, we adopt the mechanism of access control list [12], where each entry specifies a subject, an object resource, an action of the subject performed on the resource and permission (e.g., allow and deny) on the action. The aim of this paper is to address the critical access control issue for the IoT system based on the above security model. In particular, we will propose an access control framework that exploits smart contracts to implement the access control lists.

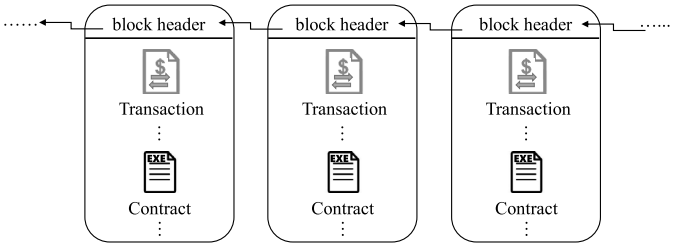



Fig. 2. Illustration of a blockchain.

## III. Smart Contract Platform

### A. Ethereum Platform

The proposed framework is based on the Ethereum smart contract platform [22], whose main elements are briefly introduced as follows. For a detailed introduction to the Ethereum platform, please refer to [22].

1) *Account/Address:* Ethereum has two types of accounts: a) externally controlled accounts (or simply accounts) and b) contract accounts (or simply smart contracts), both identified by a 20-byte address.
2) *Smart Contract:* A smart contract is a special account with associated code (i.e., its functions) and data (i.e., its state) [32]. The code is in an Ethereum-specific binary format (i.e., Ethereum Virtual Machine bytecode) and deployed by an account to a global database known as blockchain. A smart contract usually provides many functions or application binary interfaces (ABIs) that can be used to interact with it. These ABIs can be executed by sending a *transaction* from an account or a *message* from another contract.
3) *Transaction and Message:* A transaction is a data package signed and sent by an account to transfer some ether (Ethereum's native token) to another account or to execute the ABIs of a contract. A message is like a transaction, but it is sent by a contract instead of an account to run the associated ABIs of another contract.
4) *Blockchain:* The blockchain contains blocks of transactions and smart contracts with each block containing the hash of its previous block, as illustrated in Fig. 2. Every node connected to the network may have a local copy of the blockchain, and help maintain and update the blockchain by including new blocks.
5) *Mining:* Mining is a process that includes new blocks into the blockchain by nodes called miners. In one mining round, each miner constructs a block of newly generated transactions and contracts, and executes the proof-of-work consensus algorithm, where the miners repeatedly guesses random numbers to solve an extremely difficult cryptographic puzzle problem related to its block until one of them wins. The winning miner then broadcasts its block to the other nodes in the

network to validate the block. For the block validation, each node not only checks the formats of the transactions and contracts in the block, but also executes the ABIs called by these transactions in its local EVM. If the formats of the transactions and contracts as well as the results of the called ABIs are valid, the other nodes will include the new block into their local blockchains; otherwise, they will discard the block. Through mining, the whole system reaches a common tamper-resistant consensus on the blockchain and no participant can deceive the others by wrongly executing the ABIs, as long as it controls no more than half of the computing power of the system. This is the key to achieving trustworthy access control for IoT systems.

### B. System Configurations

To apply the Ethereum platform in our access control framework, we need to make the following basic configurations to the system.

1) Each peer must be associated with an Ethereum account, through which each peer can claim the deployment of a smart contract and identify itself during the access control.
2) The Ethereum client can be run at all peers in the system except for IoT devices, due to the limited energy and computing power of IoT devices. All clients are assumed synchronized on the same block. Using the client, each peer except for IoT devices can directly interact with the blockchain to deploy smart contracts and send transactions to run the ABIs of smart contracts. These peers can also function as miners to conduct the mining task for the system.
3) As IoT devices have no Ethereum clients, IoT gateways act as agents for their local IoT devices by storing their accounts. Using these accounts, IoT gateways deploy and execute smart contracts on behalf of its local IoT devices. We assume that gateways are physically accessible and thus unlikely to be compromised, so they can be trusted as the agents.

## IV. Access Control Framework

This section presents the smart contract-based distributed access control framework. We first introduce the system of smart contracts in the framework and then explain the main functions provided by the framework.

### A. Smart Contract System

As illustrated in Fig. 3, the proposed framework consists of multiple ACCs, each of which implements the access control for a pair of peers, one JC, which receives the misbehavior report of a peer from an ACC, judges the misbehavior and determines the corresponding penalty, and one RC, which stores the information of the JC and ACCs and provides functions to manage these contracts. Each of the contracts is introduced as follows.

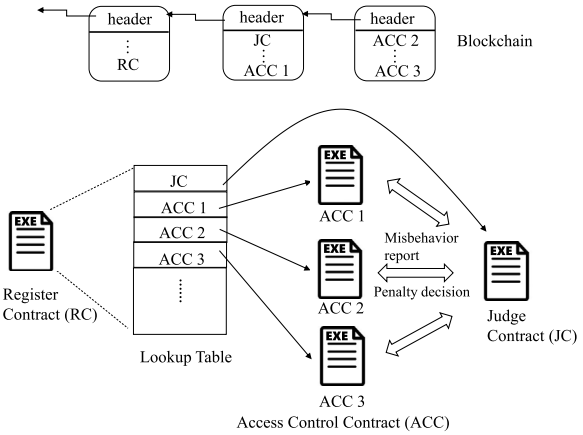



Fig. 3. Illustration of smart contract system.

TABLE I
Illustration of Policy List

| Resource | Action | Permission | ToLR |
|---|---|---|---|
| file A | read | allow | 2017-12-11 16:19 |
| file A | write | deny | 2017-12-12 20:34 |
| Program A | execute | deny | 2017-12-11 16:19 |
| ... | ... | ... | ... |

*1) Access Control Contract:* An ACC (e.g., ACC 1–ACC 3 in Fig. 3) is deployed by a peer (object) who wants to control the access requests from another peer (subject). We assume that the subject-object pair can agree on multiple access control methods, and each method is implemented by one ACC. As a result, one subject-object pair can be associated with multiple ACCs, but one ACC can be associated with one and only one subject-object pair. In this framework, to control the access requests from the subject, each ACC implements not only static access right validation by checking predefined policies but also dynamic validation by checking the behavior of the subject.

An example of the ACC is given as follows. In this example, to achieve the access control, the ACC maintains a policy list as illustrated in Table I, in which each row corresponds to the policy defined on a certain (resource, action) pair. The basic fields of each row are as follows.

- *Resource:* The resource for which the policy is defined, such as a file, a computing unit and a storage unit, etc.
- *Action:* The action that is performed on the resource, such as read, write, execute, etc.
- *Permission:* The static permission predefined on the action, such as allow, deny, etc.
- *Time of Last Request (ToLR):* The time of the last access request from the subject.

The *permission* field can be used for static validation and the ToLR can be used for dynamic validation, such as detecting the misbehavior that the subject sends access requests too frequently in a short period of time.

To record the misbehavior that the subject has exhibited on a certain resource as well as the corresponding penalty, the ACC also maintains a misbehavior list for each resource (as

TABLE II
ILLUSTRATION OF MISBEHAVIOR LIST FOR EACH RESOURCE

| Misbehavior | Time | Penalty |
|---|---|---|
| Sending access requests too frequently | 2017-12-11 16:19 | blocked for 2 hours |
| Sending access requests too frequently | 2017-12-12 20:34 | blocked for 4 hours |
| ... | ... | ... |

illustrated in Table II), where each row has the following basic fields.

- *Misbehavior:* The misbehavior of the subject on this resource, such as sending access requests too frequently in a short period of time, etc.
- *Time:* The time when the misbehavior is exhibited.
- *Penalty:* The penalty on the subject for its misbehavior, such as blocking its access requests for a certain period of time, etc.

The *misbehavior* field may also describe the details of the misbehavior to facilitate the misbehavior judging at the JC.

The ACC also provides the following main ABIs to manage the policies and implement the access control.

- *policyAdd():* This ABI receives the information of a new access control policy and adds the information to the policy list.
- *policyUpdate():* This ABI receives the information of a policy that needs to be updated and updates the policy.
- *policyDelete():* This ABI receives the identification information of a policy and deletes the policy.
- *accessControl():* This ABI receives the information required for access control and returns the access result and penalty. This ABI implements both static and dynamic validation. When the subject calls (by sending a transaction) this ABI to authorize its current access request, both static and dynamic validation processes will start to check the validity of the request. Once a possible misbehavior is detected, the ACC reports it to the JC by sending a *message* to execute the *misbehaviorJudge* ABI of the JC, receives a penalty decision on the misbehavior from the JC and takes countermeasures based on the penalty decision. The access request is authorized if and only if it successfully passes both static and dynamic validation processes.
- *setJC():* In order for the ACC to execute the ABI of the JC, the ACC needs to keep an instance of the JC, so this ABI is to receive the address of the JC and set the JC instance.
- *deleteACC():* This ABI performs the *selfdestruct* operation to remove the code and storage of the ACC from the blockchain [32], such that the ACC can no longer be available.

Notice that only the creator of the ACC can add a new policy, update or delete an existing policy, set the JC and delete the ACC. Thus, permission must be carefully considered in the implementation of the ABIs.

*2) Judge Contract:* The JC implements a misbehavior-judging method, which judges the misbehavior of the subject and determines the corresponding penalty, when receiving a potential misbehavior report from an ACC, as illustrated in Fig. 3. The penalty can be based on the misbehavior history of the subject, so the JC may need to keep a record of the misbehavior history of all subjects. After determining the penalty, the JC returns the decision to the ACC for further operation. Here, we give an example of the JC, which maintains a misbehavior list for each subject who has behaved abnormally, as illustrated in Fig. 4. The fields of each record include the following.

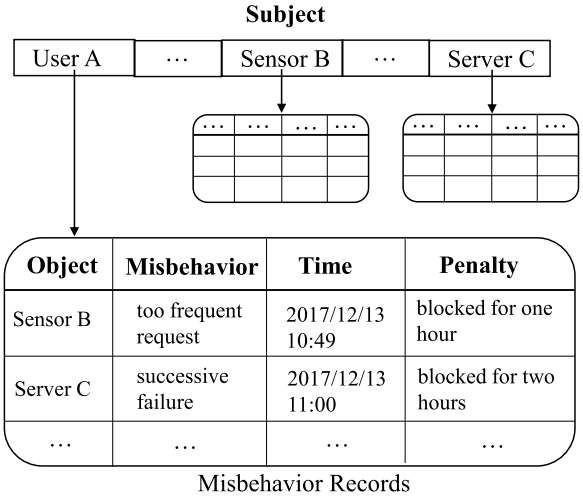



Fig. 4. Illustration of misbehavior records.

- *Object:* The peer who suffered from the misbehavior.
- *Misbehavior:* The details of the misbehavior.
- *Time:* The time when the misbehavior is exhibited.
- *Penalty:* The penalty imposed on the misbehavior.

The JC also provides the following main ABIs for judging misbehavior, determining the penalty, and managing the JC.

1) *misbehaviorJudge():* This ABI can be run by any ACC to report the misbehavior of a subject to the JC. After receiving the report, this ABI judges the misbehavior of the subject, determines the penalty on the subject based on the misbehavior history of the subject, and returns the penalty decision to the ACC that reported the misbehavior. This ABI also adds a new misbehavior record to the misbehavior list of the subject.
2) *deleteJC():* This ABI performs the *selfdestruct* operation to delete the JC.

*3) Register Contract:* The main role of the RC in the system is to manage the access control and misbehavior-judging methods. To achieve this goal, the RC maintains a lookup table, which registers the required information to find and execute all the methods. An example of the lookup table is given in Table III, in which each row contains the following information of a method.

- *MethodName:* The name of the method.
- *Subject:* The subject of the corresponding subject-object pair of the method.
- *Object:* The object of the corresponding subject-object pair of the method.
- *ScName:* The name of the corresponding smart contract implementing this method.

TABLE III
ILLUSTRATION OF LOOKUP TABLE

| MethodName | Subject | Object | ScName | Creator | ScAddress | ABI |
|---|---|---|---|---|---|---|
| Method 1 | Server A | Sensor B | ACC 1 | Sensor B | 0xca35b7d915458ef540ade6068dfe2f44e8fa733c | accessControl(),... |
| Method 2 | Server A | Sensor B | ACC 2 | Sensor B | 0xab072c469475346532bf47aea86df61761049565 | accessControl(),... |
| Method 3 | Sensor B | Server A | ACC 3 | Server A | 0xb51f6d86d4c998531056a501344060fbafc32a48 | accessControl(),... |
| JC | | | Judge | | 0x3f23c7b929cced4191ef6064ffcb33902ea1d92b | misbehaviorJudge()... |
| ... | ... | ... | ... | ... | ... | ... |

- *Creator:* The peer who created and deployed the contract.
- *ScAddress:* The address of the smart contract.
- *ABI:* The ABIs provided by the contract.

For the JC, the *subject* and *object* fields are left blank. In general, the object is the creator of the ACC as well as the creator of the access control method. Notice that for the case where the object is an IoT device, the creator is the local gateway, i.e., the agent for deploying contracts and sending transactions for the IoT device.

With the help of the lookup table, the RC provides the following main ABIs to mange these methods.

- *methodRegister():* This ABI receives the information of a new method and registers the information into the lookup table.
- *methodUpdate():* This ABI receives the information of an existing method that needs to be updated and update the information, especially the fields of ScAddress and ABI.
- *methodDelete():* This ABI receives the *MethodName* of a method and deletes the method from the lookup table.
- *getContract():* This ABI receives the *MethodName* of a method and returns the address and ABIs of the contract (i.e., the ACCs and JC) of the method.

Notice that only the creator of the method can register, update and delete the method.

### B. Main Functions of the Framework

With the help of the ACC, JC, and RC smart contracts, the framework can provide many functions to facilitate the access control of the IoT system. These functions mainly include registering, updating, and deleting an access control method; registering and updating the misbehavior-judging method; adding, updating, and deleting a policy of an ACC; and the access control for a subject-object pair. The process of each function is explained as follows.

*1) Registering New Access Control Method:* A subject-object pair can agree on a new access control method, which is registered by the creator (i.e., the object) of the method through the following steps.

- *Step 1:* Create (i.e., write and compile) an ACC for the new method.
- *Step 2:* Send a transaction to deploy the newly created ACC onto the blockchain.
- *Step 3:* Send a transaction to call the *methodRegister* ABI of the RC to register the required information of the new ACC in the lookup table of the RC.

Registering the misbehavior-judging method follows the same steps as above.

*2) Updating Existing Access Control Method:* A subject-object pair can agree on updating an existing access control method, which is conducted by the creator of the method through the following steps.

- *Step 1:* Create a new ACC, which is used to replace the old one.
- *Step 2:* Send a transaction to deploy the newly created ACC onto the blockchain.
- *Step 3:* Send a transaction to run the *methodUpdate* ABI of the RC to update the ACC-related fields of the method, such as the ScName, ScAddress, ABI, etc.
- *Step 4:* Send a transaction to run the *deleteACC* ABI of the old ACC to destruct it.

Updating the misbehavior-judging method follows the same steps as above.

*3) Deleting Existing Access Control Method:* A subject-object pair can agree on deleting an existing access control method, which is conducted by the creator of the method through the following steps.

- *Step 1:* Send a transaction to run the *methodDelete* ABI of the RC to delete the information of the existing method from the lookup table.
- *Step 2:* Send a transaction to run the *deleteACC* ABI of the ACC of the method.

*4) Adding, Updating, and Deleting Policy:* A subject-object can agree on adding an access control policy for a newly deployed resource, which is conducted by the creator of the method through sending a transaction to call the *policyAdd* ABI of the corresponding ACC. Similarly, the creator can send a transaction to call the *policyUpdate* (resp. *policyDelete*) ABI of the ACC to update (resp. delete) an existing policy of the access control method.

*5) Access Control:* The ACC for the access control of a subject-object pair can be called by either the subject or the object. We assume that both subject and object know the names of all the available methods for the access control between them. The illustration of the case where the ACC is called by the subject is given in Fig. 5(a), where a server (the subject) wants to access the resource of an IoT device (the object). To complete the access control, the following steps are executed.

- *Step 1:* The server calls the *getContract* ABI of the RC to retrieve the ACC [e.g., the ACC 2 in Fig. 5(a)] for the access control.
- *Step 2:* The RC returns the address and ABI of the ACC to the server.
- *Step 3:* The server sends a transaction, which contains the required information for access control, to call the

TABLE IV
SPECIFICATIONS OF DEVICES

| Device | CPU | Operating System | Memory | Hard Disk |
|---|---|---|---|---|
| Dell Inspiron 3650 | Intel Core i7-6700, 3.40GHz | Windows 10 Home (64 bit) | 16GB | 2TB |
| MacBook Pro | Intel Core i5, 2GHz | macOS Sierra (Version 10.12.6) | 8GB | 256GB |
| Raspberry Pi 3 Model B | quad-core ARM Cortex A53, 1.2GHz | Raspbian GNU/Linux 8 (jessie) | 1GB SDRAM | 16GB (microSD card) |

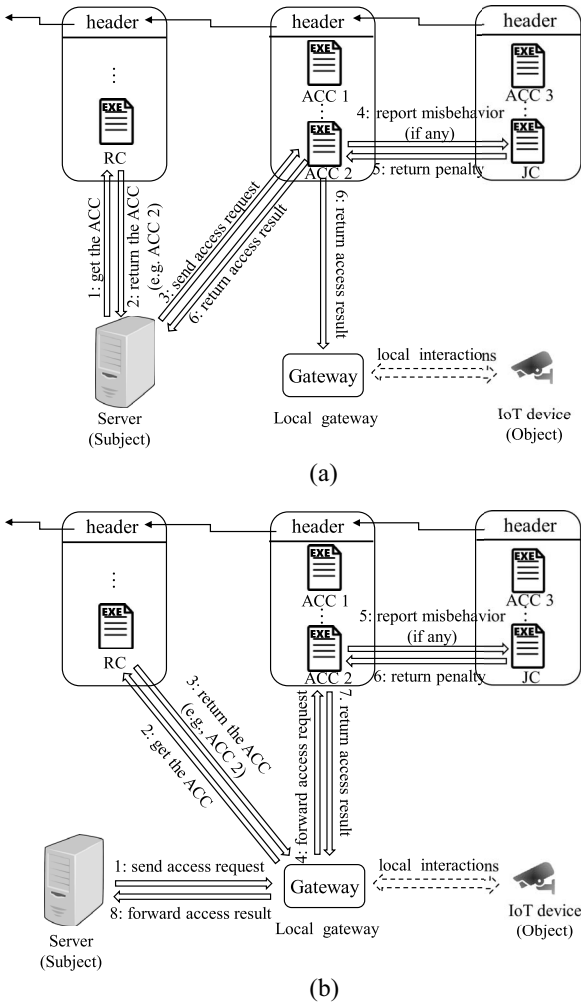



Fig. 5. Illustration of access control. ACC called by the (a) subject and (b) object.

*accessControl* ABI of the ACC. This transaction will be encapsulated in a new block and the *accessControl* ABI will not be executed until the new block is mined and included in the blockchain by some miner.

- *Step 4:* During the access control process, the ACC will send a *message* to call the *misbehaviorJudge* ABI of the JC, if some potential misbehavior of the subject is detected.
- *Step 5:* Once the *misbehaviorJudge* ABI completes judging the misbehavior and determining the penalty, it will return the penalty to the ACC.
- *Step 6:* Finally, the access result will be returned to both subject and object, after the access control process finishes.

Since all miners will reach a consensus on the result of the access control through mining, so no miners can tamper with the access control process. As the agent of the IoT device, the local gateway informs the IoT device the real-time status of the access control, such as the arrival of access requests and the access results, via secure local interactions. Fig. 5(b) illustrates the case where the ACC is called by the object. The main difference between the access control in Fig. 5(b) and that in Fig. 5(a) is that the access request of the subject (resp. the access result) in Fig. 5(b) is forwarded by the object rather than being directly sent to the *accessControl* ABI of the ACC (resp. the subject).

Fig. 6. Hardware used in the case study.

## V. CASE STUDY

This section provides a case study to demonstrate the application of the proposed framework for distributed access control in the IoT. We first introduce the hardware and software used in the study and then present how the access control is implemented based on the framework. Finally, we show some experiment results.

### A. Hardware and Software

We considered a case with one desktop computer (Dell Inspiron 3650), one laptop (MacBook Pro), and two single-board computers (Raspberry Pi 3 Model B), as shown in Fig. 6. The specifications of these devices are listed in Table IV. The desktop and laptop correspond to the user devices in the system and the single-board computers correspond to the local gateways. We considered the access control issue between the single-board computers, of which one serves as the subject (or the agent of the subject) and the other severs as the object (or the agent of the object).

On each device, a geth client [33] (a command line interface implemented in the Go language) was installed to transform the device into an Ethereum node. With the geth clients, we created an Ethereum account for each node and configured these nodes to form a private blockchain network (as illustrated in Fig. 7), where the desktop computer and the laptop play the roles of miners due to their relatively large computing

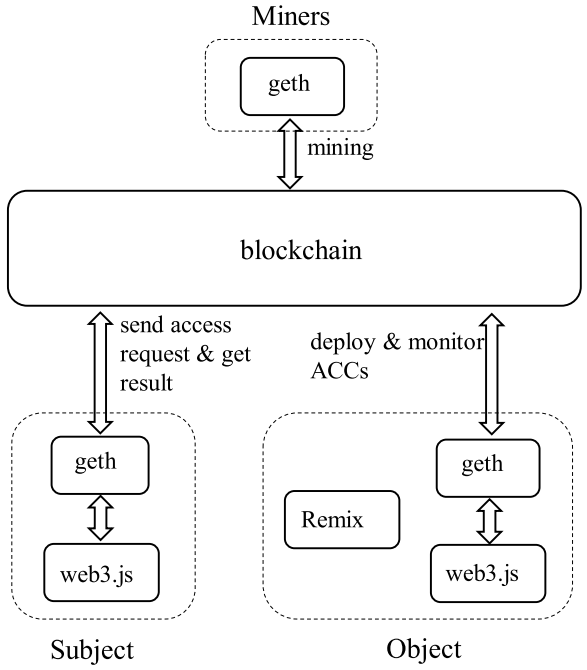



Fig. 7. Software used in the case study.

and storage capability, and the single-board computers function as lightweight Ethereum nodes that deploy ACCs and send transactions for access control.

For writing and compiling the ACC at the object side, we utilized the Remix integrated development environment (IDE) [34], which is a browser-based IDE for Solidity (i.e., the programming language for writing smart contracts) [35]. In addition, we adopted the web3.js [36] (i.e., the official Ethereum JavaScript API) at the object side to interact with the corresponding geth client through HTTP connections for deploying the compiled ACC and also monitoring the states of the ACC (i.e., the results of the access control). The web3.js was also installed at the subject side to interact with the geth for sending access requests to the ACC via transactions and also receiving the access control results from the ACC.

### B. Implementation

The implementation of the ACC, RC, and JC is based on the examples in Section IV-A.

*1) ACC:* In this implementation, we defined a simple misbehavior, which is sending access requests too frequently in a short period of time. To help characterize the misbehavior, we added the following fields to the rows (i.e., policies) in Table I.

- *minInterval:* The minimum allowable time interval between two successive requests. If the time interval between two successive requests is less than or equal to *minInterval*, the later request will be treated as a frequent request.
- *NoFR:* The number of frequent requests in a short time period.

**Algorithm 1** accessControl ABI

**Input:** *resource*, *action*, *time*
**Output:** *result*, *penalty*
*Require*: *policyCheck* ← false, *behaviorCheck* ← true, *penalty* ← 0, JC instance *judge*, policy list *policies*, *timeofUnblock* of *resource*.

```
1: if This request is from the subject then
2:   p ← policies[resource][action].
3:   if timeofUnblock ≤ time then
4:     if timeofUnblock > 0 then
5:       p.NoFR ← 0, p.ToLR ← 0, timeofUnblock ← 0.
6:     end if
7:     if p.policy = "allow" then
8:       policyCheck ← true.
9:     else
10:      policyCheck ← false.
11:    end if
12:    if time − p.ToLR ≤ p.minInterval then
13:      p.NoFR ← p.NoFR + 1.
14:      if p.NoFR ≥ p.threshold then
15:        Detect a misbehavior msb.
16:        behaviorCheck ← false.
17:        penalty ← judge.misbehaviorJudge(subject,
           msb).
18:        timeofUnblock ← time + penalty.
19:        Push msb into the misbehavior list of resource.
20:      end if
21:    else
22:      p.NoFR ← 0.
23:    end if
24:  end if
25:  p.ToLR ← time.
26: end if
27: result ← policyCheck and behaviorCheck.
28: Trigger event returnResult(result, penalty).
```

- *Threshold:* The threshold on the NoFR. If the NoFR is larger than or equal to the *threshold*, the ACC judges that a misbehavior occurs.

As the penalty for the misbehavior, the access requests from the subject will be blocked for a certain time period. We introduced a variable *timeOfUnblock* for each resource to represent the time until which requests are blocked, which is set to 0 when the requests are unblocked. The blocking period is thus quantified as the difference between the *timeOfUnblock* and the time when the misbehavior is detected. The blocking period can be a constant or a function of the misbehavior history of the subject. The subject can rejoin the access control after the blocking period expires. We used a struct to store the fields of a policy and applied a 2-D mapping from the fields of *resource* (primary key) and *action* (secondary key) to this struct to construct the policy list. The ACC also contains a JC instance, through which the *misbehaviorJudge* ABI of the JC can be run by the ACC. Based on the above fields and variables, we designed the *accessControl* ABI as in Algorithm 1, which receives the inputs of *resource*, *action*

**Algorithm 2** Access Request JavaScript

**Input:** *resource*, *action*, *time*
**Output:** *result*, *penalty*

```
1: Create a RC instance register.
2: Specify the access control method name method.
3: (addr, abi)← register.getContract(method).
4: Create an ACC instance acc with addr, abi.
5: Send a transaction containing parameters (resource,
   action, time) to the accessControl ABI of acc.
6: while ture do
7:    if Event returnResult() is captured then
8:       (result, penalty) ← returnResult().
9:       break.
10:   end if
11: end while
12: return result, penalty
```

**Algorithm 3** Access Monitor JavaScript

```
1: Create a RC instance register.
2: Specify the access control method name method.
3: (addr, abi)← register.getContract(method).
4: Create an ACC instance acc with addr, abi.
5: while ture do
6:    if Event returnResult() is captured then
7:       (result, penalty) ← returnResult().
8:       Display result, penalty.
9:    end if
10: end while
```

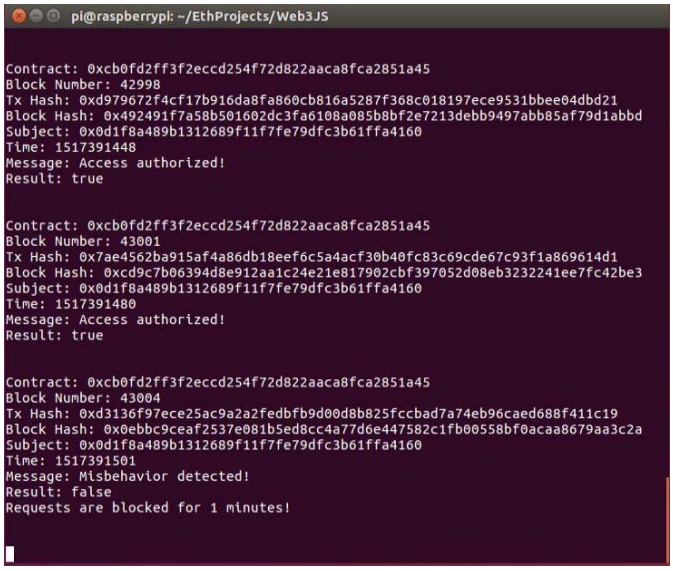
and *time* (i.e., when the request is sent), and returns the access *result* and *penalty*. The static validation is from lines 7–11 and the dynamic validation is from lines 12–23. The event *returnResult(result, penalty)* in line 28 is used to return the access result and penalty to both subjects and objects. For the detailed implementation of the ACC, please refer to [37].

*2) RC:* The key issue in the implementation of the RC is to construct the lookup table as shown in Table III. Like the construction of policy list for the ACC, we used a struct to store the information of each method and applied a mapping from the field of *MethodName* to this struct to construct the lookup table.

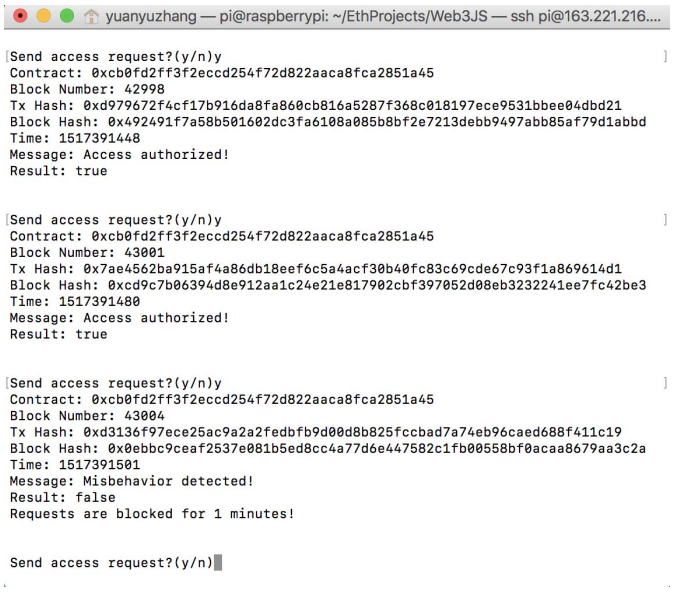
*3) JC:* In the implementation of the JC, we used a dynamic array to store the misbehavior records of a subject. We considered a simple misbehavior judging method, which treats all potential misbehavior received from the ACC as misbehavior. When receiving a misbehavior report of a subject from the ACC, the *misbehaviorReport* ABI pushes the misbehavior into the misbehavior record array of the subject and then uses the following function to determine the corresponding penalty:

$$\text{penalty} = (\text{base})^{\lfloor \ell/\text{interval} \rfloor} \tag{1}$$

where $\ell$ is the number of misbehavior records of the subject (i.e., the length of the misbehavior record array of the subject), and base and interval are parameters that determine how the penalty changes with $\ell$. Notice that base and interval are initialized when the JC is deployed.




(a)

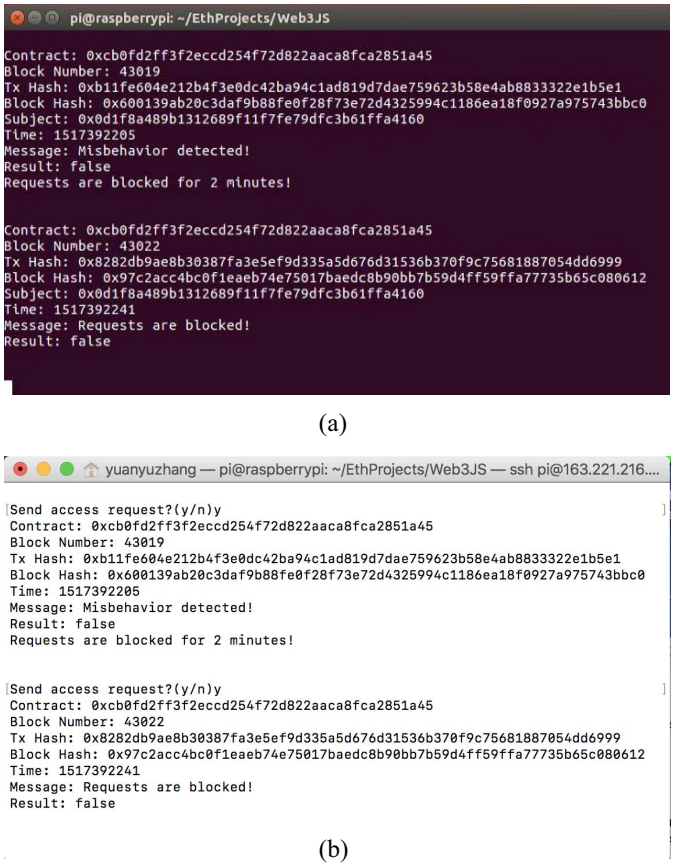



(b)

Fig. 8. Access results after misbehavior occurring once. Results at the (a) object and (b) subject.

*4) JavaScripts at the Subject and Object:* The access control in this paper is implemented based on the case in Fig. 5(a), where the ACC is called by the subject and the result is returned to both sides. To implement the access control, we created two JavaScripts (one at the subject and the other at the object) using the web3.js to interact with the JC and ACC. As shown in Algorithm 2, the JavaScript at the subject side first retrieves the address *addr* and ABI *abi* of the ACC from the RC (lines 1–3) and then sends a transaction that contains the access request information (*resource*, *action*, *time*) to run the *accessControl* ABI of the ACC for access control (lines 4 and 5). Finally, the JavaScript watches the event *returnResult()* returned from the *accessControl* ABI to retrieve the access result (lines 6–11).

The JavaScript at the object side is illustrated in Algorithm 3, which uses the same statements (lines 1–3) to

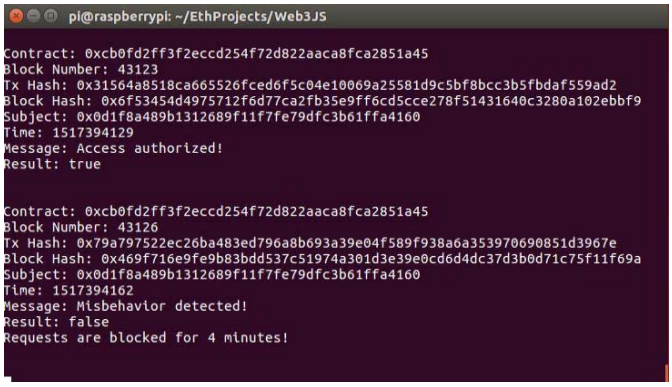



(a)




(b)

Fig. 9. Access results after misbehavior occurring for three times. Results at the (a) object and (b) subject.




(a)


(b)

Fig. 10. Access results after misbehavior occurring for six times. Results at the (a) object and (b) subject.

TABLE V
SYSTEM PARAMETERS

| Parameter | Value | Meaning |
|---|---|---|
| *minInterval* | 100 seconds | the minimum allowable time interval between two successive requests |
| *threshold* | 2 | the threshold on the *NoFR* |
| *base* | 2 | penalty-related parameter |
| *interval* | 3 | penalty-related parameter |

retrieve the address and ABI of the ACC from the RC and infinitely watches the *returnResult()* events from the ACC to know who wants to access which resource at what time, and what the corresponding result and penalty are (lines 4–10).

### C. Experiments

Our source code for the ACC, JC, RC, and JavaScripts of the case study is now available at [37]. Based on the code, the hardware and software, we conducted experiments to show the feasibility of the framework for distributed access control. We added a policy to the ACC and set the system parameters as in Table V.

The access results are summarized in Figs. 8–10, where the meaning of each item is illustrated in Table VI. Fig. 8 shows the access results displayed by the JavaScripts at the object [Fig. 8(a)] and subject [Fig. 8(b)], when the subject exhibited misbehavior for the first time. Figs. 9 and 10 show the access results, when the subject exhibited misbehavior for three times and six times, respectively. We can see that the request of the subject is blocked for 1, 2, and 4 min in Figs. 8–10, respectively, which is consistent with the penalty determining equation in (1).

TABLE VI
MEANING OF ITEMS IN FIGS. 8–10

| Item | Meaning |
|---|---|
| Contract | address of the ACC |
| Tx Hash | hash of the transaction containing access requests |
| Block Number | number of the block which the Tx resides in |
| Block Hash | hash of the block which the Tx resides in |
| Subject | address of the subject |
| Time | time (in seconds) when the access request is sent |
| Message | message showing the access result |
| Result | true if the authorization succeeds, otherwise false |

### D. Cost and Overhead

The Ethereum platform uses a unit called gas to measure how much work has been done to perform some task, for example, deploying or executing a smart contract. Gas has price and thus the number of gas consumed represents the capital cost for performing this task. In general, the more complex the task is, the more gas is required. In this case study, the numbers of gas required for deploying the ACC, RC, and JC are 2 543 479, 1 559 814, and 1 380 781, respectively, and that for executing the ACC for access control is 90 000.

In the case study, the average time required for deploying the ACC, JC, and RC is less than one minute and that for executing the ACC for access control is less than 30 s. Notice that the time required for deploying and running smart contracts depends on various factors, like the system computing power (or hashrate), the system networking architecture. Thus, in the real-world public Ethereum system, the time cost may differ significantly from that in this case study.

Notice that the main purpose of the case study is to demonstrate the possibility of our framework in achieving distributed access control for IoT systems. The overhead in such a simple scenario, like the CPU and network bandwidth occupation, may not be able to reflect the overhead of the framework in real-world IoT systems. Thus, our future work is to deploy this framework in our real-world IoT system in [38] and conduct extensive overhead tests to further demonstrate the feasibility of our framework.

## VI. Conclusion

This paper investigated the access control issue in the IoT, for which we proposed a smart contract-based framework to implement distributed and trustworthy access control. The framework includes multiple ACCs for access control of multiple subject-object pairs in the system, one JC for judging the misbehavior of the subjects during the access control, and one RC for managing the ACCs and JC. A case study was also provided for the access control in an IoT system with one desktop computer, one laptop, and two Raspberry Pi single-board computers. The case study demonstrated the feasibility of the proposed framework in achieving distributed and trustworthy access control for the IoT.

## References


[1] I. Yaqoob *et al.*, "Internet of Things architecture: Recent advances, taxonomy, requirements, and open challenges," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 10–16, Jun. 2017.

[2] M. R. Palattella *et al.*, "Internet of Things in the 5G era: Enablers, architecture, and business models," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 510–527, Mar. 2016.

[3] L. D. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[5] C. J. DáOrazio, K.-K. R. Choo, and L. T. Yang, "Data exfiltration from Internet of Things devices: IoS devices as case studies," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 524–535, Apr. 2017.

[6] E. Bertino and N. Islam, "Botnets and Internet of Things security," *Computer*, vol. 50, no. 2, pp. 76–79, Feb. 2017. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MC.2017.62

[7] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015.

[8] J. Singh, T. Pasquier, J. Bacon, H. Ko, and D. Eyers, "Twenty security considerations for cloud-supported Internet of Things," *IEEE Internet Things J.*, vol. 3, no. 3, pp. 269–284, Jun. 2016.

[9] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the Internet of Things: Big challenges and new opportunities," *Comput. Netw.*, vol. 112, pp. 237–262, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128616303735

[10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[11] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, Feb. 2015.

[12] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.

[13] A. Yavari, A. S. Panah, D. Georgakopoulos, P. P. Jayaraman, and R. V. Schyndel, "Scalable role-based data disclosure control for the Internet of Things," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2226–2233.

[14] Q. Liu, H. Zhang, J. Wan, and X. Chen, "An access control model for resource sharing based on the role-based access control intended for multi-domain manufacturing Internet of Things," *IEEE Access*, vol. 5, pp. 7001–7011, 2017.

[15] N. Ye, Y. Zhu, R.-C. Wang, R. Malekian, and L. Qiao-Min, "An efficient authentication and access control scheme for perception layer of Internet of Things," *Appl. Math. Inf. Sci.*, vol. 8, no. 4, p. 1617, 2014.

[16] S. Bhatt, F. Patwa, and R. Sandhu, "Access control model for AWS Internet of Things," in *Proc. Int. Conf. Netw. Syst. Security*, 2017, pp. 721–736.

[17] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the Internet of Things," *Math. Comput. Model.*, vol. 58, nos. 5–6, pp. 1189–1205, 2013.

[18] P. N. Mahalle, B. Anggorojati, N. R. Prasad, and R. Prasad, "Identity authentication and capability based access control (IACAC) for the Internet of Things," *J. Cyber Security Mobility*, vol. 1, no. 4, pp. 309–348, 2013.

[19] J. L. Hernández-Ramos, M. P. Pawlowski, A. J. Jara, A. F. Skarmeta, and L. Ladid, "Toward a lightweight authentication and authorization framework for smart objects," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 4, pp. 690–702, Apr. 2015.

[20] D. Hussein, E. Bertin, and V. Frey, "A community-driven access control approach in distributed IoT environments," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 146–153, Mar. 2017.

[21] *Bitcoin—Open Source P2P Money*. Accessed: Jan. 31, 2018. [Online]. Available: https://bitcoin.org/en/

[22] *An Introduction to Ethereum Platform*. Accessed: Jan. 31, 2018. [Online]. Available: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html

[23] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[24] N. Kshetri, "Can blockchain strengthen the Internet of Things?" *IT Prof.*, vol. 19, no. 4, pp. 68–72, Aug. 2017.

[25] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2017, pp. 618–623.

[26] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[27] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Security Privacy Workshops*, San Jose, CA, USA, May 2015, pp. 180–184.

[28] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proc. IFIP Int. Conf. Distrib. Appl. Interoperable Syst.*, 2017, pp. 206–220.

[29] A. Ouaddah, A. A. El Kalam, and A. A. Ouahman, "FairAccess: A new blockchain-based access control framework for the Internet of Things," *Security Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, 2016.

[30] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. IEEE Int. Conf. Open Big Data (OBD)*, 2016, pp. 25–30.

[31] A. Ramachandran and M. Kantarcioglu, "Using blockchain and smart contracts for secure data provenance management," *CoRR*, vol. abs/1709.10000, 2017. [Online]. Available: http://arxiv.org/abs/1709.10000

[32] *An Introduction to Ethereum Smart Contracts*. Accessed: Jan. 31, 2018. [Online]. Available: http://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html

[33] *Geth Client for Building Private Blockchain Networks*. Accessed: Jan. 31, 2018. [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/geth

[34] *Remix IDE for Ethereum Smart Contract Programming*. Accessed: Jan. 31, 2018. [Online]. Available: https://remix.ethereum.org/

[35] *Solidity—A Contract-Oriented, High-Level Language for Implementing Smart Contract*. Accessed: Jan. 31, 2018. [Online]. Available: https://solidity.readthedocs.io/en/develop/

[36] *Web3 Javascript API to Interact With Ethereum Nodes*. Accessed: Jan. 31, 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/JavaScript-API
[37] *Implement Access Control in a Simple IoT System Using Ethereum Smart Contrats*. Accessed: Jan. 31, 2018. [Online]. Available: http://mdlval.blogspot.jp/
[38] *Open IoT Platform in Xidian University*. Accessed: Apr. 3, 2018. [Online]. Available: http://222.25.188.1:50023/

**Yuanyu Zhang** (S'17–A'17–M'18) received the B.S. degree in software engineering and M.S. degree in computer science from Xidian University, Xi'an, China, in 2011 and 2014, respectively, and the Ph.D. degree from the School of Systems Information Science, Future University Hakodate, Hokkaido, Japan, in 2017.

He is currently an Assistant Professor with the Graduate School of Science and Technology, Nara Institute of Science and Technology, Ikoma, Japan. His current research interests include physical layer security, blockchain, Internet of Things security, and performance modeling and evaluation of wireless networks.

**Shoji Kasahara** (A'94–M'98) received the B.Eng., M.Eng., and Dr.Eng. degrees from Kyoto University, Kyoto, Japan, in 1989, 1991, and 1996, respectively.

He was an Assistant Professor with the Educational Center for Information Processing, Kyoto University, from 1993 to 1997, where he was also an Associate Professor with the Department of Systems Science, Graduate School of Informatics, from 2005 to 2012. He was a Visiting Scholar with the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, and the University of Waterloo, Waterloo, ON, Canada, in 1996. From 1997 to 2005, he was with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Japan, where he has been a Professor since 2012. His current research interests include stochastic modeling and analytics of large-scale complex systems based on computer/communication networks.

Dr. Kasahara is a member of the ORSJ, IPSJ, and ISCIE.

**Yulong Shen** (M'09) received the B.S. and M.S. degrees in computer science and Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2002, 2005, and 2008, respectively.

He is currently a Professor with the School of Computer Science and Technology, Xidian University, where he is also an Associate Director of the Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services Networks. His current research interests include wireless network security and cloud computing security.

Dr. Shen has also served on the Technical Program Committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.

**Xiaohong Jiang** (M'03–SM'09) received the B.S., M.S., and Ph.D. degrees from Xidian University, China, in 1989, 1992, and 1999, respectively.

He is currently a Full Professor with Future University Hakodate, Hakodate, Japan. Before joining Future University, he was an Associate Professor with Tohoku University, Sendai, Japan, from 2005 to 2010. He has authored or co-authored over 260 technical papers in premium international journals and conferences, which include over 50 papers published in top IEEE journals and conferences such as the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE INFOCOM. His current research interests include computer communications networks, mainly wireless networks and optical networks, network security, and routers/switches design.

Dr. Jiang was a recipient of the Best Paper Award of IEEE HPCC 2014, IEEE WCNC 2008 and 2012, IEEE ICC 2005-Optical Networking Symposium, and IEEE/IEICE HPSR 2002. He is a member of the ACM and IEICE.

**Jianxiong Wan** received the B.Sc. degree in computer science from Shannxi Normal University, Xi'an, China, in 2004, the M.Sc. degree in management science from the Beijing Information Technology Institute, Beijing, China, in 2009, and the Ph.D. degree in computer science from the University of Science and Technology Beijing, in 2013.

He was a Post-Doctoral Research Fellow with Massey University, Palmerston North, New Zealand, from 2016 to 2017, and a Visiting Scholar with the Nara Institute of Science and Technology, Ikoma, Japan, from 2017 to 2018. He joined the Inner Mongolia University of Technology, Hohhot, China, in 2013, where he is currently an Associate Professor with the School of Data Science and Application. His current research interests include energy-efficient computing and performance modeling of distributed systems with a special focus on cloud-scale systems.