Using Ethereum Blockchain for Distributed Attribute-Based Access Control in the Internet of Things

Mirei Yutaka, Yuanyu Zhang, Masahiro Sasabe and Shoji Kasahara
Graduate School of Science and Technology, Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma, Nara 630-0192, Japan.
Email: yutaka.mirei.yj7@is.naist.jp, {yy90zhang, m-sasabe, kasahara}@ieee.org

***Abstract*—Access control has been recognized as a critical issue for preventing unauthorized access to the resources in Internet of Things (IoT) systems. This paper proposes an Attribute-Based Access Control (ABAC) framework for IoT systems by using the emerging Ethereum smart contract technology. The framework consists of one Policy Management Contract (PMC), one Subject Attribute Management Contract (SAMC), one Object Attribute Management Contract (OAMC) and one Access Control Contract (ACC). The PMC, SAMC and OAMC are responsible for storing and managing the ABAC policies, the attributes of subjects (i.e., entities accessing resources) and the attributes of objects (i.e., resources being accessed), respectively. When receiving access requests, the ACC retrieves the subject attributes and object attributes as well as the corresponding policy from the SAMC, OAMC and PMC to perform the access control. Combining the ABAC model and the blockchain technology, this framework is expected to achieve distributed, trustworthy and fine-grained access control for IoT systems. To show the feasibility of the proposed framework, we construct a local private Ethereum blockchain system to implement the four smart contracts and also conduct experiments to test the monetary and time cost.**

***Index Terms*—Ethereum Blockchain, Internet of Things (IoT), Attribute-Based Access Control (ABAC)**

I. Introduction

With the rapid development of the Internet technology, an increasing number of smart devices, like personal computers, smart phones and home appliances, are now being connected to the Internet, forming a huge Internet of Things [1]. Although these devices are making our life increasingly intelligent and convenient, their vulnerability to cyber attacks puts our property and lives in danger as well. An example of the cyber attacks is a malware called Mirai [2], which was reported in 2016. Mirai can illegally access and control a huge number of IoT devices to launch large-scale Distributed Denial-of-Service (DDoS) [3] attacks to servers. Another attack by illegally accessing and controlling a web camera was reported in [4], which significantly threatened the privacy of the users. The main cause of these attacks is the illegal access to IoT devices, through which adversaries can illegally access private information, control unit (e.g., car brake) and computer programs. Therefore, designing effective access control schemes that can prevent such illegal access in IoT systems has been regarded as a crucial research issue [5].

Most current IoT access control schemes are centralized, i.e., relying on a central server to control all the access requests in the system [6]–[8]. Although such schemes are easy to manage, the server itself becomes a single point of failure. This means that once the server is compromised by adversaries or destroyed by man-made or natural disasters, the data for access control (e.g., access rights assigned to entities, access records) may be falsified or lost. As a result, the access control scheme fails to function normally. Besides, modern IoT systems are distributed and large-scale in nature, which makes centralized access control schemes scale poorly when applied in these systems. To address the limitations of centralized schemes, distributed access control schemes, which rely on multiple nodes to control access requests in a distributed manner, have been proposed [9], [10]. Since multiple nodes participate in the access control process, all these nodes must maintain the same data for access control and agree on the same results. This usually requires distributed consensus mechanisms that are robust to solve the disputes among the participants, especially when malicious participants exist to deceive the others with falsified data and results.

The blockchain technology, the key enabler of cryptocurrency systems like the Bitcoin [11], has been recognized as a distributed digital ledger with robust consensus mechanisms. A blockchain consists of a sequence of blocks chained together and is shared by all the participants of the system. Each block contains the cryptographic hash of the previous block and a collection of transactions recording the remittance information (e.g., sender, receiver and amount). The process of calculating a valid hash value for a block is called mining and requires a huge amount of calculations. If the transactions in a certain block are altered, the hash values of the block and its subsequent blocks must be re-calculated. This is considered impossible in general and thus makes the blockchain resistant to tampering. Bitcoin-like blockchains store only static transactions and thus are considered as distributed databases in general. Recently, the Ethereum blockchain [12], which stores not only transactions but also executable programs called smart contracts on the blockchain, has attracted considerable attentions. A smart contract consists of variables as its states and functions called Application Binary Interfaces (ABIs) to view and change the states [13]. A transaction is required to

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

run an ABI for changing the states, and all miners receiving this transaction will execute the ABI as well to reach the same state. Therefore, the Ethereum's blockchain provides not only a distributed database but also a distributed and trustworthy computing platform.

Motivated by the appealing features of the blockchain, some recent efforts have been devoted to the design of blockchain-based access control schemes [14]–[20]. These schemes can be roughly classified into Bitcoin-based and Ethereum-based. Due to the limited storage capability of the Bitcoin blockchain, the Bitcoin-based schemes usually use the blockchains to store relatively simple access-related data, like the access control lists (ACL) in [14], the policies in [15] and the tokens in [16]. On the contrary, the Ethereum-based schemes can use powerful smart contracts to implement more complex and user-friendly access control. These schemes mainly employ the ACL access control model [17], Capability-Based Access Control (CapBAC) model [18] or Attribute-Based Access Control (ABAC) model [19], [20]. Compared with other models, the ABAC model combines the attributes of subjects (i.e., entities requesting resources), objects (i.e., resources being accessed), actions (e.g., read, write, execute) and context (e.g., IP address, time, location) to provide dynamic and fine-grained access control, and is considered as the *next-generation* authorization [21]. The authors in [19] proposed an smart contract-based ABAC scheme. In this scheme, ABAC policies are stored in external databases while their URLs are stored on the blockchain. When accessing an object, subjects send the URL of the related policy (i.e., the policy that is responsible for the access control of the subject-object pair) to a smart contract. The smart contract retrieves the policy from the external databases and performs the access control. Although storing only the URLs of the policies on the blockchain can reduce the storage overhead of the blockchain to some extent, the policies and attributes face the risk of being falsified, thus resulting in untrustworthy access control. Besides, the authors provided no implementations to verify the feasibility of the scheme.

To address the limitations of the scheme in [19], we propose a novel smart contract-based ABAC framework. The framework consists of one Policy Management Contract (PMC), one Subject Attribute Management Contract (SAMC), one Object Attribute Management Contract (OAMC) and one Access Control Contract (ACC). The PMC, SAMC and OAMC are responsible for storing and managing access control policies, the attributes of subjects and the attributes of objects, respectively. When receiving access requests from a subject, the ACC retrieves the corresponding policy, subject attributes and object attributes from the PMC, SAMC and OAMC respectively to perform the access control. To demonstrate the feasibility of the proposed framework, we construct a local private Ethereum blockchain system and implement the four smart contracts.

The remainder of the paper is organized as follows. Section II introduces the related work and Section III presents the proposed framework. We demonstrate the feasibility of the framework in Section IV by introducing the implementation details. Finally, we conclude this paper in Section V.

II. Related Work

The authors in [14] used a Bitcoin-like blockchain to propose an IoT access control scheme based on the ACL model in a smart home application. Each home has a local blockchain to maintain an ACL, where each entry specifies the allowed access rights of an internal or external subject to an internal object. In addition, a miner is placed in the home to receive access requests from the subjects and perform the access control based on the ACL. However, the access control inside each home is centralized due to the existence of the single miner. Besides, the miner does not perform the mining task, which makes it possible to tamper with the ACL and thus results in untrustworthy access control. In [15], an ABAC scheme was proposed based on the Bitcoin blockchain, where ABAC policies are stored in Bitcoin transactions and fetched by existing ABAC solutions to perform access control. The policy inside a transaction can be updated by appending a new transaction with update information to it or be deleted by simply spending the coins contained in it. A similar ABAC scheme based on Bitcoin was proposed in [22], where, different from [15], the policies are encrypted for privacy and security. In [16], the authors proposed a CapBAC scheme based on the Bitcoin blockchain, where the Bitcoin transactions are used to store capability tokens, each of which records the assigned access rights of a certain subject to one or more objects. Similar to the coin transfer, capability tokens can also be transferred from one subject to another through transactions. When accessing an object, the subject passes its token to the owner of the object to prove that it has the access rights.

In addition to the above Bitcoin-based access control schemes, there also exist schemes based on Ethereum smart contracts. For example, the authors in [17] proposed an ACL-based access control framework using Ethereum smart contracts, where one smart contract is deployed for each subject-object pair to store the ACL and implement the related access control. When a subject wants to access an object, it sends a transaction including the required access information to the corresponding smart contract. Once the smart contract is executed, the access control results will be returned to both the subject and object. In [18], a smart contract was deployed to store the capability tokens of subjects as well as the capability delegation information among the subjects. When receiving a capability token from a subject, the owner of the object can decide whether the subject has access rights by simply viewing the capability information stored in the smart contract. For dynamic and fine-grained access control, the authors in [19] proposed an ABAC scheme, which stores the URL links of ABAC policies on the blockchain and uses a smart contract to receive the URLs from the subjects and then perform the access control. However, the trustworthiness of the policies and the attributes cannot be guaranteed by storing them in external databases. Besides, the feasibility of the proposed scheme is not clear due to the lack of implementations. The authors in [20] also proposed an ABAC scheme based on Ethereum smart contracts, but the attributes of the objects were

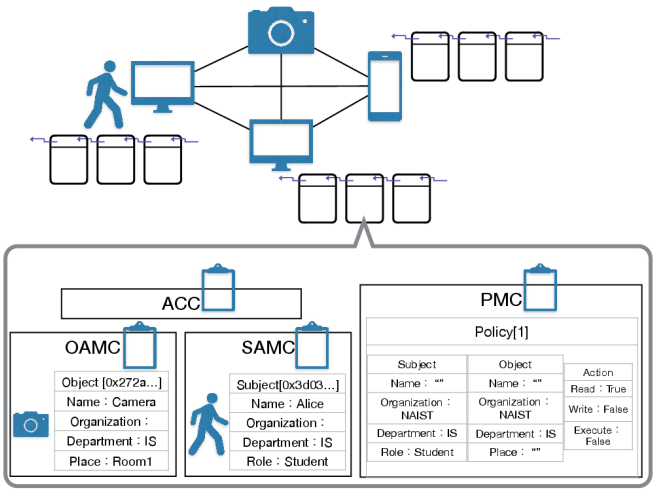

Fig. 1: The proposed ABAC framework.

not considered and only many-to-one access control could be realized.

III. Proposed ABAC Framework

A. Smart Contract System

As shown in Fig. 1, the proposed framework consists of four Ethereum smart contracts, namely the Subject Attribute Management Contract (SAMC), the Object Attribute Management Contract (OAMC), the Policy Management Contract (PMC) and the Access Control Contract (ACC). The SAMC, OAMC and PMC are responsible for storing and managing (e.g., updating, adding, deleting) the subject attributes, object attributes and policies, respectively. The ACC is responsible for the access control in the IoT system. Each smart contract is introduced as follows.

1) Subject Attribute Management Contract (SAMC): The SAMC is deployed on the blockchain to store and manage the attributes of the subjects in the IoT system. Only administrators of the subjects have the permissions to execute this smart contract. For example, if the subjects are citizens, the administrators can be the city office. If the subjects are IoT devices, the administrators can be the owners. Each subject has a unique identifier (i.e., ID) to represent itself in the system. This paper uses Ethereum account addresses (e.g., the 0x3d03... in Table I) as such ID information. In addition, each subject has multiple attributes associated with its ID. As illustrated in Table I, this paper considers the *Organization* (e.g., Nara Institute of Science and Technology, NAIST) and *Department* (e.g., Information Science, IS) to which the subject belongs as well as the *Role* (e.g., student, staff) of the subject as examples of the attributes. The SAMC also provides the ABIs of *subjectAdd()*, *subjectDelete()* to add/update and delete the attributes of subjects, respectively.

2) Object Attribute Management Contract (OAMC): Similar to the SAMC, the OAMC is executed by the object administrators to store and manage the attributes of the objects. Each object has multiple attributes, which are uniquely associated with its ID, i.e., its Ethereum account addresses (e.g., the 0x272a... in Table I). Table I shows some examples of the object attributes considered in this paper, including the *Organization* (e.g., NAIST) and *Department* (e.g., IS) to which the object belongs as well as the *Place* (e.g., Room1, Room2) where the object is placed. In addition to the attribute list, the OAMC also provides the ABIs of *objectAdd()* and *objectDelete()* to add/update and delete the attributes of the objects, respectively.

TABLE I: Examples of subject and object attributes

SubjectList[0x3d03...]		ObjectList[0x272a...]	
Name	"Alice"	Name	"Camera"
Organization	"NAIST"	Organization	"NAIST"
Department	"IS"	Department	"IS"
Laboratory	"LSM"	Laboratory	"LSM"
Role	"student"	Place	"Room1"
Others	""	Others	""

3) Policy Management Contract (PMC): The PMC is used to store and manage the ABAC policies defined in this paper and can only be executed by the policy administrators (e.g, the object owners). A policy is a statement that combines a set SA of subject attributes, a set OA of object attributes and a set A of actions to state that subjects with attributes in the SA can perform the actions in A on objects with attributes in the OA. Table II shows an example of the ABAC policy defined in this paper with $SA = \{Organization : NAIST, Department : IS, Laboratory : LSM\}$, $OA = \{Organization : NAIST, Department : IS, Laboratory : LSM\}$ and $A = \{Read, Write\}$. The policy states that any *student* belonging to the *LSM* laboratory of the *IS* department of the *NAIST* organization can *read* and *write* any object at any place of the same laboratory of the same department and organization. The PMC uses a list structure to store the policies and also provides the ABIs of *policyAdd()*, *policyDelete()*, *policyUpdate()* to add, delete and update the policies respectively. In addition, the PMC provides the ABIs of *findExactMatchPolicy()* and *findMatchPolicy()* for searching policies. These two types of policy search will be introduced in Section III-B2 in greater details.

TABLE II: Example of an ABAC policy.

Subject Attributes	Object Attributes	Action
Name:""	Name:""	Read:True
Organization:"NAIST"	Organization: "NAIST"	Write: True
Department:"IS"	Department: "IS"	Execute: False
Laboratory:"LSM"	Laboratory: "LSM"	
Role: "Student'	Place:""	

4) Access Control Contract (ACC): The ACC acts like the "brain" of the access control system to control the access requests from the subjects to the objects in the IoT system. The ACC can be executed by the subjects by sending transactions containing the required request information. When the ACC receives access requests from a subject, it will retrieve the attributes of the subject and object as well as the corresponding policy from the SAMC, the OAMC and PMC, respectively.

Using the attributes and policy, the ACC verifies the access right of the subject, determines the access results and returns the results to both the subject and object. A more detailed access control flow is described in Section III-B4.

B. Main Functions of the Framework

The proposed ABAC framework provides the following main functions.

1) Adding, Updating and Delete Attributes: As mentioned in Section III-A, the proposed framework provides the subject/object administrators with functions of managing the attribute information of their subjects/objects. For example, when adding/updating the attributes of a subject, the subject administrator can send the ID of the subject and the attributes to add/update via a transaction to the *subjectAdd()* ABI of the SAMC. The ABI will create a new entry in the subject list if no existing entry associated with the subject ID is found. Otherwise, the ABI will update the subject's attributes with the new ones. Similarly, to delete some attributes of a subject, the administrator sends to the *subjectDelete()* ABI the subject's ID and attributes to delete.

2) Searching Policies: Since the policies are stored in the structure of a list (i.e., array) in the PMC, policy search is required when deleting, updating and retrieving a certain policy. The framework provides two types of policy search, i.e., search by complete match and search by partial match, which are implemented by the ABIs *findExactMatchPolicy()* and *findMatchPolicy()*, respectively.

- Search by Complete Match: Search by complete match searches for the policy whose subject and object attributes match completely with the subject and object attributes provided by the caller. For example, when searching for the policy illustrated in Table II through the search by complete match, the caller needs to provide exactly the same attribute information as listed in Table II. This search is mainly used for deleting a policy.
- Search by Partial Match: Different from search by complete match, search by partial match returns all the policies whose subject and object attributes are contained by those provided by the caller. This means that the sets of attributes in the returned policies are subsets of those provided by the caller. Consider a case where the caller calls the search by partial match with a subject attribute set SA and an object attribute set OA. In this case, the search will return any policy with $SA' \subseteq SA$ and $OA' \subseteq OA$. This is because that any returned policy can handle the access request associated with the SA and OA. The search by partial match is mainly used for adding/updating policies and retrieving policies by the ACC. A list of indices of matched policies will be returned after the search.

3) Adding, Updating and Deleting Policies: Policy administrators can define policies by combining the attribute information of subjects, objects and actions. The more attributes a policy contains, the more fine-grained and flexible the access control it can achieve. To add a new policy, the policy administrator first needs to execute the *findMatchPolicy()* to search similar policies using the policy search by partial match. After receiving the similar policies from the *findMatchPolicy()* ABI, the administrator needs to make sure that no conflicts exist between the similar policies and the new policy to add. If conflicts exist, the administrator must resolve them. When all conflicts are resolved, the administrator then executes the *policyAdd()* ABI of the PMC to add the new policy to the policy list. Similarly, when updating a policy, the administrator also needs to execute the *findMatchPolicy()* to find the policy to update (i.e., the target policy) and other similar policies. After resolving conflicts (if any) between the similar policies and the new policy used for update, the administrator then executes the *policyUpdate()* ABI to update the target policy with the new one. Note that there are cases where there is no need to add new policies or update an policy, because other existing policies may cover the new ones. This can reduce the monetary cost and the storage overhead of the scheme. To delete a policy, the policy administrator first executes the policy search by complete match to find the index of the target policy. If the policy is found, the administrator then passes the index to the *policyDelete()* ABI, which then removes the policy from the policy list.

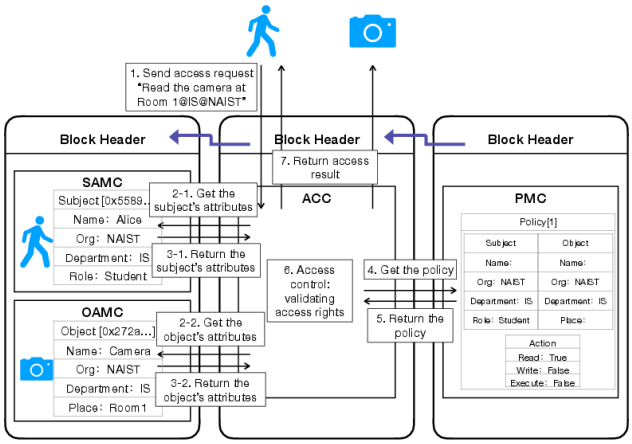

Fig. 2: Access control process.

4) Access Control: The core function provided by the framework is the access control. The process of the access control is illustrated in Fig. 2, which shows the case where a student Alice belonging to the IS department of the NAIST wants to access a camera located in Room 1 of the IS department of the NAIST. The detailed steps of the process are introduced as follows.

- Step 1: The subject Alice sends an access request transaction containing her ID, the ID of the object camera and the actions to perform to the ACC.
- Step 2: The ACC sends a message containing the IDs of the subject and object to the SAMC and OAMC to retrieve their attribute information.
- Step 3: The SAMC and OAMC return the attributes of

```
Contract:  0xd2b12e5B4D4536E9FD2a1b17b7Ef05A106cE5BEE
Block Number: 3320
Tx Hash: 0x96db405b0e91260e3cec3164d2cb8d0675d787256e9127ae59a8a96be601885a
Block Hash: 0x008201b601080148972638609 6e0dbd9a1763de7664ecf6481daa621006a394a
Subject:  0x3D038089541BacdA4321C42E660E834D7aA2CcFa
Massage:  Access authorized!
Result:  true
```

Fig. 3: Access result (Action: Read)

```
Contract:  0xd2b12e5B4D4536E9FD2a1b17b7Ef05A106cE5BEE
Block Number: 3278
Tx Hash: 0xdc81d9a419d453f3153b2348c46861a551414f1b2837f7d86d5c2ebd24f7a61a
Block Hash: 0xd4849117df55292a71434f0889d395a7a29103e82e221f9fcb84b1099d8cdeab
Subject:  0x3D038089541BacdA4321C42E660E834D7aA2CcFa
Massage:  Access Rquest Faild
Result:  false
```

Fig. 4: Access result (Action: Execute)

the subject and object to the ACC respectively.

- Step 4: The ACC sends a message containing the attributes of the subject and object to the PMC to query the related policies.
- Step 5: The PMC searches the related policies using the policy search by partial match, and returns the found policies to the ACC.
- Step 6: Based on the received policies, the ACC determines if the subject has rights to perform actions on the object.
- Step 7: The ACC returns the access results to both the subject and object.

Using the smart contract system, the access control becomes a distributed application, which is executed by the majority of the system nodes. In addition, the access history and results are also stored on the blockchain. Thus, even if some of the nodes are destroyed by disasters or compromised by adversaries, the access control framework can still work reliably. This achieves distributed and trustworthy access control for the IoT.

IV. Implementation

We constructed a private Ethereum blockchain network with three nodes on a computer server (Intel Xeon CPU E5-1620 3.60 GHz, 32 GB memory). We implemented the proposed framework on this private blockchain and conducted experiments to demonstrate the feasibility of the proposed framework. Specifically, we installed the geth client [23] on the server and used it to set up three Ethereum nodes to form the private blockchain network, as shown in Fig. 5. One of the three nodes serves as the miner of the blockchain network and the other two play the role of the subject and object, respectively. We also used the Remix [24], a browser-based integrated development environment (IDE), to edit and compile the smart contracts. The Remix IDE can be configured to connect to any of the nodes via an Remote Procedure Call (RPC) connection, so it can also be used to deploy the smart contracts for any node. To interact with the nodes, we also installed the web3.js package [25] and used it to create javaScript programs for sending transactions and viewing the access result at both the subject and object sides.

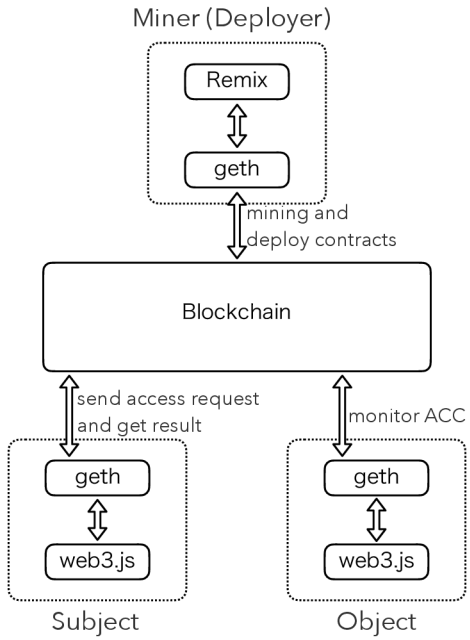

Fig. 5: Software used in the proposed framework.

A. Feasibility Validation

We considered that the subject and object have attributes as shown in Table I. We used the policy in Table II to control the access request from the subject to the object. From Tables I and II, we can see that the subject only has rights of *read* and *write* on the object. Fig. 3 and Fig. 4 depict the results when the subject sent a *read* request and an *execute* request, respectively. Both figures show the information of the address of the ACC, the transaction hash sent by the subject, the block hash where the access request transaction is stored, the address (i.e., ID) of the subject and the access results. The results in these two figures demonstrate the feasibility of the proposed ABAC framework.

B. Cost Evaluation

The users need to pay some money to deploy smart contracts on the blockchain and execute the ABIs of these contracts. Ethereum uses a unit called gas to measure the amount of operations needed to perform a task, e.g., deploying a smart contract or executing an ABI. In general, the more complex a task is, the more gas it consumes. Gas has price that varies with time. Thus, the money needed to pay for performing a task is the product of the amount of consumed gas and the gas price. Table III lists the money paid for some operations, like adding a policy/subject/object, deploying the ACC and executing the ACC. The money is calculated in US dollars (USD) based on the exchange rate between USD and Gas as of 2019/04/11 [26].

In the proposed scheme, the amount of gas required for one access control is $264,635$, which is about 0.141 USDs. We can see from the table that the proposed ABAC framework consumes more gas than the scheme in [17], but the monetary

TABLE III: Monetary cost for some operations

	Scheme in [17]		Propose Scheme	
	Gas	USD $	Gas	USD $
Adding a policy	128,777	0.068	363,964	0.193
Adding subject attributes	-	-	152,863	0.081
Adding object attributes	-	-	155.246	0.082
Deploying the ACC	1,706,290	0.906	1,301,972	0.691
Access control	75,771	0.04	264,635	0.141
Total	1,910,838	1.015	2,238,680	1.189

gap in USD between these two schemes is small. In [17], one ACC is deployed for only one subject-object pair (i.e., one-to-one access control), the monetary cost will increase linearly as the number of subject-object pairs of the system increases. However, the proposed framework achieves many-to-many access control between the subjects and objects, and thus does not needs to deploy new ACC when the the number of subject-object pairs increases. This will consume less money than the scheme in [17].

The average time for access control is about 10 seconds in the scheme of [17] and about 36 seconds in the proposed scheme. This gap is due to the relatively complex interactions between the ACC and other smart contracts for retrieving attributes and policies. Note that the execution time of the ABI varies depending on various factors such as the system's computing power, network architecture, timing of mining, etc., so the execution time may differ within the public Ethereum network.

V. Conclusion

This paper proposed an Attribute-Based Access Control (ABAC) framework for the IoT by using Ethereum smart contracts to manage ABAC policies, attributes of subjects and objects and perform access control. A local private Ethereum blockchain network was constructed to implement the proposed framework and evaluate its feasibility and cost in terms of money and time. The results showed that the framework is feasible to achieve distributed and fine-grained IoT access control but at some cost of money and time.

Acknowledgments

This work was supported in part by the JSPS KAKENHI (A) under Grant 19H01103, the JSPS KAKENHI (C) under Grant 19KT0045, the Telecommunications Advancement Foundation, the Support Center for Advanced Telecommunications Technology Research Foundation and the Nara Institute of Science and Technology (NAIST) Big Data Project.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] "Mirai botnet linked to dyn DNS DDoS attacks," available at https://www.flashpoint-intel.com/blog/cybercrime/mirai-botnet-linked-dyn-dns-ddos-attacks/.

[3] N. Long and R. Thomas, "Trends in denial of service attack technology," CERT Coordination Center, 2001.

[4] "Breached webcam and baby monitor site flagged by watchdogs," available at https://www.bbc.com/news/technology-30121159.

[5] 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.

[6] 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. of 2017 IEEE 37th International Conference on Distributed Computing Systems*, 2017, pp. 2226–2233.

[7] 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, no. 2, pp. 7001–7011, 2017.

[8] E. Yuan and J. Tong, "Attributed based access control (ABAC) for Web services," in *Proc. of IEEE International Conference on Web Services*, 2005, pp. 561–569.

[9] J. L. Hernandez-Ramos, A. J. Jara, L. Marin, and A. F. Skarmeta, "Distributed Capability-based Access Control for the Internet of Things," *Journal of Internet Services and Information Security*, vol. 3, no. 3/4, pp. 1–16, 2013.

[10] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "On the design of a decentralized and multi-authority access control scheme in federated and cloud-assisted cyber-physical systems," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5190–5204, 2018.

[11] "Bitcoin - open source p2p money," available at https://bitcoin.org/en/.

[12] "An introduction to Ethereum patform," available at http://ethdocs.org/en/latest/introduction/what-is-ethereum.html.

[13] "A next-generation smart contract and decentralized application platform," available at https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf.

[14] 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. of IEEE PerCom Workshops*, 2017, pp. 618–623.

[15] D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proc. of IFIP International Conference on Distributed Applications and Interoperable Systems*, 2017, pp. 206–220.

[16] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Fairaccess: A new blockchain-based access control framework for the Internet of things," *Security and Communication Networks*, vol. 9, no. 18, pp. 5943–5964, 2016.

[17] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of things," to appear in IEEE Internet of Things Journal.

[18] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Computers*, vol. 7, no. 3, pp. 39–65, 2018.

[19] C. Dukkipati, Y. Zhang, and L.C.Cheng, "Decentralized, Blockchain based access control framework for the heterogeneous Internet of things," in *Proc. of 3rd Workshop on Attribute Based Access Control*, 2018, pp. 61–69.

[20] G. Hao, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proc. of 2019 International Conference on Blockchain Technology*, 2019, pp. 6–11.

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

[22] Y. Zhu, Y. Qin, G. Gan, S. Yang, and W. C.-C. Chu, "TBAC: Transaction-based access control on blockchain for resource sharing with cryptographically decentralized authorization," in *Proc. of 2018 42nd IEEE International Conference on Computer Software & Applications*, 2018, pp. 535–544.

[23] "Geth client for building private blockchain networks." available at https://github.com/ethereum/go-ethereum/wiki/geth.

[24] "Remix ide for ethereum smart contract programming." available at https://remix.ethereum.org/.

[25] "Web3 javascript api to interact with ethereum nodes." available at https://github.com/ethereum/wiki/wiki/JavaScript-API.

[26] "ETH Gas Station," available at https://ethgasstation.info/index.php.