Capability-Based Access Control for the Internet of Things: An Ethereum Blockchain-Based Scheme

Yuta Nakamura, 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: nakamura.yuta.ns1@is.naist.jp, {yy90zhang, m-sasabe, kasahara}@ieee.org

***Abstract*—The large-scale and trustless nature of the Internet of Things (IoT) calls for distributed and trustworthy access control schemes to prevent unauthorized resource access. This paper proposes a Capability-Based Access Control (CapBAC) scheme by applying the emerging Ethereum blockchain technology. This scheme uses Ethereum smart contracts, i.e., executable codes residing in the blockchain, to store and manage the capability tokens, i.e., special data structures that maintain the allowed actions of a user (i.e., subject) on a certain resource (i.e., object). To provide more fine-grained access control and more flexible token management, this scheme defines capability tokens in units of actions, i.e., by dividing a conventional capability token containing multiple actions into multiple ones with each being associated with a certain action. In addition, this scheme uses a delegation graph instead of the delegation tree in existing smart contract-based CapBAC schemes to store the token delegation relationship among the subjects. By storing the tokens and the delegation graph in smart contracts, this scheme allows object owners to verify the ownership and validity of the capability tokens of the subjects. To demonstrate the feasibility of the scheme, we constructed a local Ethereum blockchain network and conducted extensive experiments.**

***Index Terms*—Ethereum Blockchain, Internet of Things, Capability-Based Access Control (CapBAC).**

I. Introduction

Thanks to the rapid maturation and commercialization of the Internet of Things (IoT), the number of devices connected to the Internet is increasing at an unprecedented speed. It was reported that over 30 billion IoT devices (e.g., appliances, wearables and industrial equipment) will be deployed to form a extremely huge IoT network by 2020 [1]. Despite the convenience and intelligence brought by these devices to our life, they are vulnerable to unauthorized access by malicious users, posing significant threats to our personal and property safety [2]. For example, malicious users may know the current situations or the contents of private conversations inside a home by illegally accessing some appliances [3]–[5]. In addition, malicious users may also be able to gain illegal access to the control unit (e.g. brake, accelerator) of a self-driving car to cause severe accidents intentionally [6]. Therefore, access control, which prevents unauthorized access by explicitly or implicitly specifying who (subjects) can access what resources (i.e., objects) under what conditions, has been a crucial research issue [7]–[9].

Conventional access control schemes are mainly centralized [10], [11], i.e., relying on a central server for all the access control-related processing including access right assignment, management (e.g., update, revocation) and verification. Although such schemes are easier to manage, the server becomes a single point of failure and may destroy the access control system once it suffers from man-made/natural disasters or is compromised by adversaries. Besides, the large-scale and distributed nature of IoT systems makes it difficult to control the resource access requests by centralized schemes. Distributed access control schemes are expected to address the above limitations of the centralized ones. In distributed access control schemes, the access control-related processing is conducted by the majority of the nodes instead of a single server. All these nodes must reach a consensus on the assigned rights, access policies and verification results to ensure robust and trustworthy access control that is resistant to the tampering of malicious users. This is why there is an increasing interest in applying the emerging blockchain technology to achieve distributed and trustworthy access control.

Blockchain was originally invented as a distributed and tamper-resistant ledger to store financial transfer data (i.e., transactions) of cryptocurrency systems, like the Bitcoin [12]. The most appealing feature of the blockchain is its ability to reach consensuses on its states (e.g., transaction history and balances) among its participants by using cryptographic hash functions, even in the presence of attackers. In addition to transactions, current blockchains, like the Ethereum [13], can also store executable programs called smart contracts on the blockchain to provide a distributed storage and computing platform. A smart contract usually consists of some variables as its state and several functions called Application Binary Interfaces (ABIs) to view and change the states. To execute an ABI to change the state, a transaction must be sent to the smart contract. Once this transaction is mined and included in the blockchain, the ABI will be executed by the majority of the participants to reach a consensus on the latest state.

The goal of this paper is to implement distributed and trustworthy access control for the IoT using Ethereum smart contracts. In particular, we focus on the Capability-Based Access Control (CapBAC) model, because this model can ensure that each subject uses the least amount of privilege (i.e.,

TABLE I
SYMBOLS IN ICAP AND IDC TOKENS.

Variable	meaning
O	The associated object.
VID_S	Identifier (ID) of the subject S.
OP	A set of authorized actions (e.g., read, write, execute).
C	Context information (e.g., time, place, IP address).
VID_P	ID of the parent subject that delegated the authorized actions to VID_s.
$\{VID_C\}$	ID of descendant subjects that the VID_s delegates part or all of the authorized actions to.
Dep	Depth of the IDC in the delegation tree.

access rights) necessary to finish its job (i.e., the principle of least privilege) [14]. In addition, the CapBAC model allows subjects to delegate access rights from one to another for flexible and spontaneous access control. Recently, some initial attempts have been made to implement access control using the blockchain technology [15]–[21]. Among these schemes, the Blockchain-ENabled Decentralized Capability-based Access Control (BlendCAC) scheme in [21] is mostly related to this paper. We will introduce the BlendCAC scheme including its main idea and limitations as well as our contributions in Section II. For the introduction of other schemes, please refer to the related work in Section V. The remainder of this paper is organized as follows. Section III introduces the proposed CapBAC scheme and Section IV presents the implementation details of the proposed scheme. Finally, we conclude this paper in Section VI.

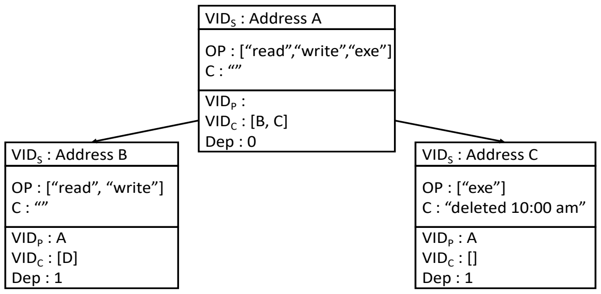
II. BLENDCAC SCHEME

To manage the authorized actions (i.e., access rights) of the subjects for each object, the BlendCAC scheme defines two types of tokens, i.e., Identity-based Capability (ICap) and Identity-based Delegation Certificate (IDC). An ICap token records the authorized actions (e.g., read, write, execute) of a subject and an IDC token records the delegation relationships of the authorized actions among the subjects. The following expressions illustrate the data structures of an ICap token and an IDC token of a certain subject S, respectively.

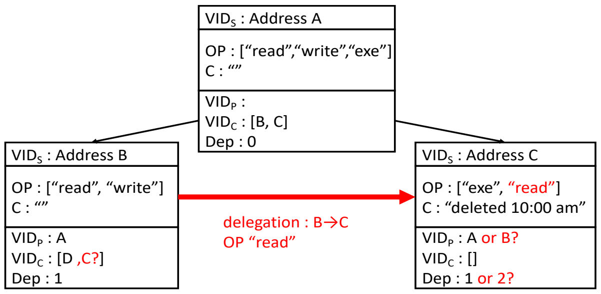
$$ICap_O[VID_S] = \{OP, C\}, \quad (1)$$
$$IDC_O[VID_S] = \{VID_P, \{VID_C\}, Dep\}, \quad (2)$$

where the meanings of the symbols are described in Table I.

Using these tokens, the BlendCAC scheme manages the capabilities of subjects and their delegation relationships for each object by a delegation tree. Fig. 1 shows an example of the delegation tree with three subjects A, B and C. This tree shows that subject A, the owner of the object, delegates its *read* and *write* rights to subject B and *exe* (i.e., execute) right to subject C. The parent subjects of B and C are set as A due to the delegation. Consider now the case where B needs to delegate its *read* right to C. In this case, should A or B be the parent subject of C? We can see that neither A nor B as the parent subject cannot record all the delegation information completely. A similar problem arises to the *Dep*



(a) Before delegation.



(b) After delegation.

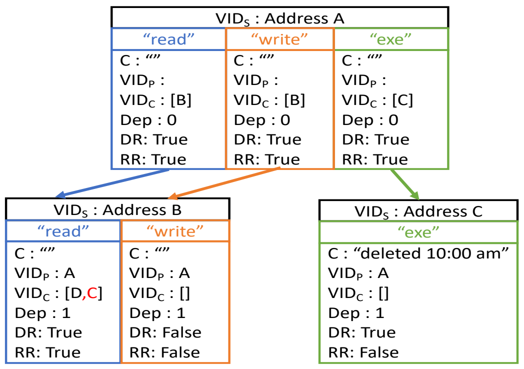
Fig. 1. Delegation tree in the BlendCAC scheme.

information. As a result, a subject cannot obtain rights from more than one subject due to the contradiction/ambiguity about the delegation information. In addition, to complete a delegation, the related ICap and IDC tokens must be updated synchronously. However, this requirement cannot always been satisfied in the blockchain system, due to the difference of the times when the two transactions for updating the tokens are included into the blockchain.

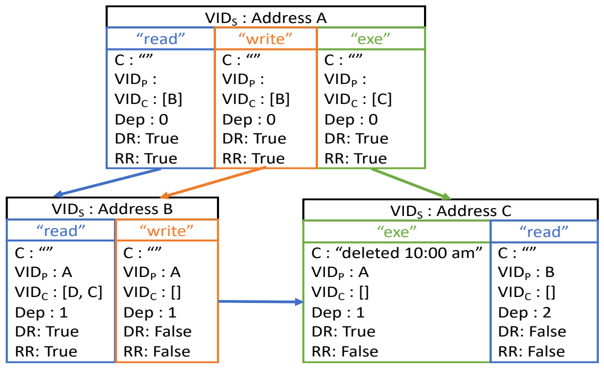
To address the above two main limitations of the BlendCAC scheme, we propose a novel smart contract-based CapBAC scheme with more fine-grained capability management and more flexible capability delegation. More specifically, we first define the capability tokens *in units of authorized actions*, i.e., in the manner of one token per action instead of one token per subject as in the BlendCAC scheme. Second, we use *one type of token* to summarize the information of capabilities and delegation relationship so as to update these information simultaneously. Finally, we manage the delegation relationship of the subjects by a *delegation graph* instead of the delegation tree in the BlendCAC scheme to enable more flexible capability delegation. Compared with the BlendCAC scheme, the proposed scheme also provides the functionality of adding new authorized actions.

III. PROPOSED CAPBAC SCHEME

This section introduces the proposed CapBAC scheme including the structure of the capability token, the delegation graph and the main functions.



(a) Before delegation.



(b) After delegation.

Fig. 2. Delegation graph of the proposed CapBAC scheme.

A. Capability Token Structure and Delegation Graph

The idea of constructing capability tokens is to split the capability tokens of the BlendCAC scheme into multiple ones based on the authorized actions with each being associated with an action. Thus, each token can be uniquely identified by the ID of the subject VID_S and an action OP, as shown in the following expression.

$$CAPS_O[VID_S][OP]{=}\{C, VID_P, \{VID_C\}, Dep, DR, RR\}, \tag{3}$$

where the meaning of O, C, VID_P, $\{VID_C\}$ and Dep is described in Table I.

Note that this paper uses the Ethereum account addresses as the ID information of both subjects and objects. The field DR indicates whether the owner of the token (i.e., VID_S) can further delegate it to other subjects. Similarly, the field RR indicates whether the subject VID_S can revoke the delegated tokens from the descendant subjects in $\{VID_C\}$. This structure allows each subject to own multiple tokens and to flexibly delegate authorized actions to and from multiple subjects. In addition, we can use only one type of token for each object to manage the capabilities of the subjects, and construct a delegation graph for managing the delegation relationships.

Fig. 2 illustrates a simple example of the delegation graph for an object O with three subjects A, B and C. Subject A, the owner of the object, has three tokens with authorized actions *read*, *write* and *exe*, and delegates the *read* and *write* tokens (resp. *exe* token) to subject B (resp. C). Again, we consider the case where B needs to delegate its *read* token to C. Because each token is independent of the others, the delegation causes no contradiction or ambiguity about the delegation information. After the delegation, C is appended to the set of descendant subjects (i.e., $\{VID_C\}$) of the *read* action of B, and B becomes the parent subject of C in terms of the *read* action. Accordingly, the depth of the delegated *read* token of C is increased by 1 compared with that of B. To manage the tokens and delegation graph, we deploy a smart contract on the Ethereum blockchain, the main functions of which are described in the following subsection.

B. Main Functions

The proposed CapBAC scheme provides the following main functions including token creation, token delegation, token revocation and token verification. Each function is introduced as follows.

1) Token Creation: Different objects require different sets of authorized actions. When the current set of actions is not enough to support new applications, some new actions may be needed. In this case, the function of token creation can be used. The smart contract provides a *createAction()* ABI for this function. Only the owner of the objects has permissions to execute this ABI. When executing this ABI, the owner needs to send a transaction containing the information defined in (3).

2) Token Delegation: Token delegation is a fundamental and critical function of CapBAC schemes to support flexible and spontaneous access. Subjects can gain access rights through the tokens delegated by other subjects without the intervention of the owner, improving the scalability of the access control scheme. The smart contract provides a *delegation()* ABI to enable the token delegation. Only the owner of the token can execute this ABI by sending a transaction with the required information.

3) Token Revocation: When the delegator (i.e., the subject that delegates the token) of a token decides that the current owner has no access permissions any more, it can revoke the token to avoid token abuse. The smart contract provides two ABIs, i.e., *singleRevocation()* and *allChildrenRevocation()*, to support the token revocation. The *singleRevocation()* ABI revokes the tokens from the children of the delegator, while the *allChildrenRevocation()* ABI revokes the tokens from all the descendants.

4) Token Verification: When accessing an object, a subject needs to hand the related token to the object's owner, which then performs the token verification to confirm that the subject has the required access rights. The smart contract provides an *accessRequest()* ABI for the token verification. Any subject can execute this ABI by offering the required information like the subject's ID and the action to perform via a transaction. The transaction will be mined, included into a block and

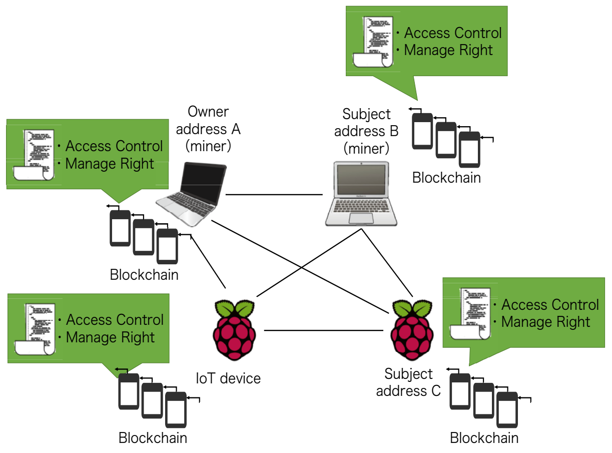


Fig. 3. Experiment environment.

broadcasted to most of the nodes in the system. During this process, each node that receives this transaction will execute the ABI to confirm whether the subject has the required access rights. This ensures that no nodes can deceive others with wrong processing results, achieving robust and trustworthy access control. After the verification of the token, the results will be returned to both the subject and the object.

IV. Implementation

In this section, we implement the capability, delegation graph and functions introduced in Section III to demonstrate the feasibility of the proposed CapBAC scheme.

A. Ethereum Private Network

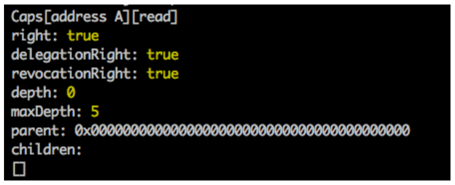
As shown in Fig. 3, we built a private Ethereum blockchain network using one Macbook Pro (CPU: 3.1 GHz Intel Core i5, Memory: 8 GB), one Macbook Air (CPU: 1.8 GHz Intel Core i5, Memory: 8 GB) and two Raspberry Pis (CPU: 1.4 GHz ARM Cortex-A, Memory: 1 GB). One Pi works as the object and the other devices work as the subjects. Besides, the Macbook Pro also plays the role of the owner entity of the object. To form a private Ethereum blockchain network, each device maintains a local copy of the blockchain and interacts with the blockchain (e.g., send transaction, obtain access result) through a javaScript program based on the web3.js package [22]. The Macbook Pro and Macbook Air serve as miners in this private Ethereum blockchain network. We created Ethereum accounts $addressA$, $addressB$ and $addressC$ for the Macbook Pro, Macbook Air and the subject Raspberry Pi, respectively, the information of which is summarized in Table II.

B. Token Creation

At the beginning of the experiment, the owner entity with address $addressA$ registered a smart contract to store and manage the capability tokens and the delegation graph. The owner entity then created new tokens by executing the *createAction()* ABI. Fig. 4(a) shows the information returned by

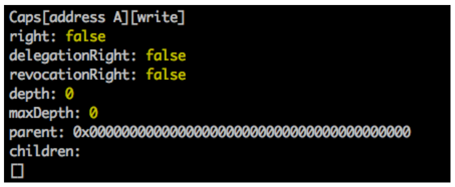
TABLE II
Ethereum addresses of the subjects.

Variable	Address
$addressA$	$0x9bE252cf45F6daa4680edeC081d7A1Bc1a92Cd6f$
$addressB$	$0xF59c4bf63FEB4ce4df4cD0E5facAE2eA95448e85$
$addressC$	$0x28bBa96539A24a98b3e0e3d00F4C02e201c3b080$



```
Caps[address A][read]
right: true
delegationRight: true
revocationRight: true
depth: 0
maxDepth: 5
parent: 0x0000000000000000000000000000000000000000
children:
[]
```

(a) Token information for a newly-created *read* token.



```
Caps[address A][write]
right: false
delegationRight: false
revocationRight: false
depth: 0
maxDepth: 0
parent: 0x0000000000000000000000000000000000000000
children:
[]
```

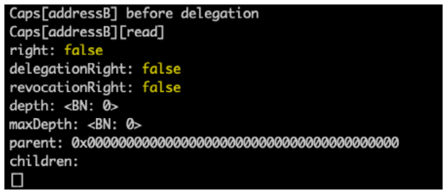
(b) Token information for a non-existent *write* token.

Fig. 4. Token creation by the owner entity $addressA$.

calling the *getCap()* ABI via the javaScript program after the owner entity created a new *read* token for the subject $addressA$ (i.e., the owner entity). The token states that the subject $addressA$ has *read* right to the object. It can also delegate this token to other objects and revoke the token from its descendants. At this point, the token has depth $\mathrm{depth} = 0$ and maximum depth $\mathrm{maxDepth} = 5$, which means that the token can be further delegated to at most five generations. Also, we can see that the token has no parents and no children. Fig. 4(b) shows the result of calling the *getCap()* ABI to query a token (i.e., the *write* token) that does not exist or has not been created on the blockchain. We can see that all fields are set as "empty," since the javaScript program cannot fetch any information from the blockchain.

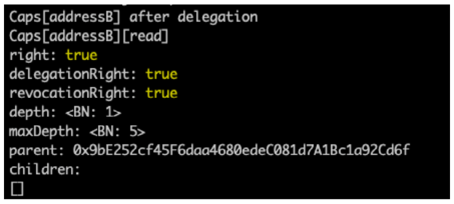
C. Token Delegation

After creating new tokens, the owner entity $addressA$ then delegated the tokens to other subjects. Fig. 5(a) and Fig. 5(b) show the token information of the subject $subjectB$ before and after the owner entity delegates the *read* token to it, respectively. We can see that the *right* field of the *read* token changes from *false* to *true* after the delegation, which means that the delegation successfully delivers the *read* right to the subject $addressB$. In addition, the *depth*, *maxDepth* and *parent* fields are changed accordingly. Note that the fields of "*delegationRight: true*" and "*revocationRight: true*" indicate that the subject $addressB$ can delegate the token to other subjects and revoke the token when necessary.



```
Caps[addressB] before delegation
Caps[addressB][read]
right: false
delegationRight: false
revocationRight: false
depth: <BN: 0>
maxDepth: <BN: 0>
parent: 0x0000000000000000000000000000000000000000
children:
□
```

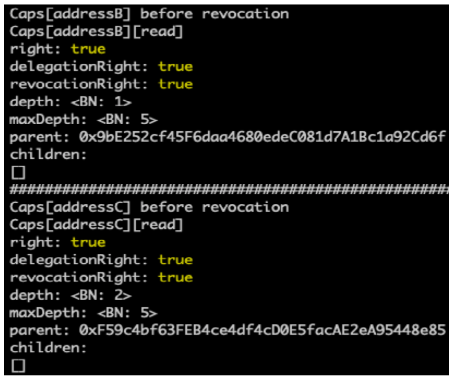
(a) Token information of $addressB$ before delegation.



```
Caps[addressB] after delegation
Caps[addressB][read]
right: true
delegationRight: true
revocationRight: true
depth: <BN: 1>
maxDepth: <BN: 5>
parent: 0x9bE252cf45F6daa4680edeC081d7A1Bc1a92Cd6f
children:
□
```

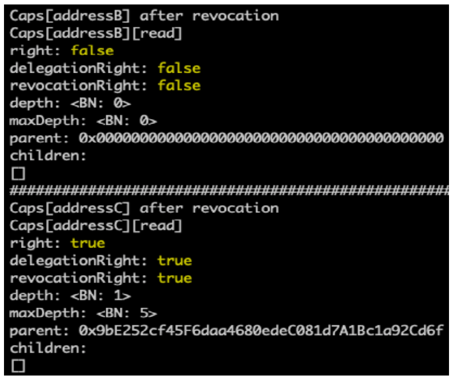
(b) Token information of $addressB$ after delegation.

Fig. 5. Delegation of *read* token from subjects $addressA$ to $addressB$.



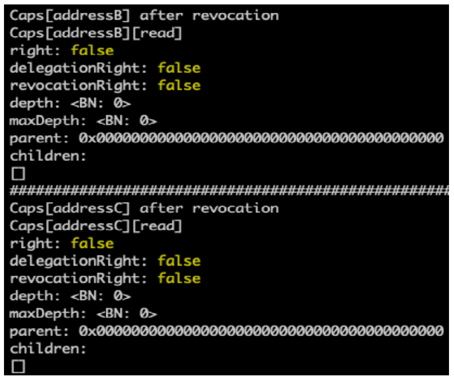
```
Caps[addressB] before revocation
Caps[addressB][read]
right: true
delegationRight: true
revocationRight: true
depth: <BN: 1>
maxDepth: <BN: 5>
parent: 0x9bE252cf45F6daa4680edeC081d7A1Bc1a92Cd6f
children:
□
################################################
Caps[addressC] before revocation
Caps[addressC][read]
right: true
delegationRight: true
revocationRight: true
depth: <BN: 2>
maxDepth: <BN: 5>
parent: 0xF59c4bf63FEB4ce4df4cD0E5facAE2eA95448e85
children:
□
```

(a) Token information of $addressB$ and $addressC$ before revocation.



```
Caps[addressB] after revocation
Caps[addressB][read]
right: false
delegationRight: false
revocationRight: false
depth: <BN: 0>
maxDepth: <BN: 0>
parent: 0x0000000000000000000000000000000000000000
children:
□
################################################
Caps[addressC] after revocation
Caps[addressC][read]
right: true
delegationRight: true
revocationRight: true
depth: <BN: 1>
maxDepth: <BN: 5>
parent: 0x9bE252cf45F6daa4680edeC081d7A1Bc1a92Cd6f
children:
□
```

(b) Token information of $addressB$ and $addressC$ after *singleRevocation()*.



```
Caps[addressB] after revocation
Caps[addressB][read]
right: false
delegationRight: false
revocationRight: false
depth: <BN: 0>
maxDepth: <BN: 0>
parent: 0x0000000000000000000000000000000000000000
children:
□
################################################
Caps[addressC] after revocation
Caps[addressC][read]
right: false
delegationRight: false
revocationRight: false
depth: <BN: 0>
maxDepth: <BN: 0>
parent: 0x0000000000000000000000000000000000000000
children:
□
```

(c) Token information of $addressB$ and $addressC$ after *allChildrenRevocation()*.

Fig. 6. Revocation of *read* token.

D. Token Revocation

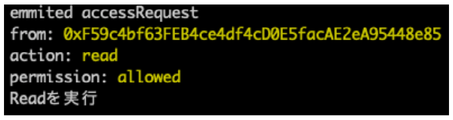
Suppose that the subject $addressB$ further delegated the *read* token to another subject $addressC$. At this point, the token information of the subjects $addressB$ and $addressC$ is shown in Fig. 6(a). When the token delegator (i.e., subject $addressA$) decides to revoke the token from the subject $addressB$ but allow the subject $addressC$ to keep the token, the delegator executes the *singleRevocation()* ABI. Fig. 6(b) shows the token information after the execution of the ABI, where only the token information of subject $addressB$ is deleted. Note that the parent of subject $addressC$ is changed from $addressB$ to $addressA$, and thus the depth is also decreased by one. On the other hand, if the delegator wants to revoke the tokens from both subjects $addressB$ and $addressC$, the delegator executes the *allChildrenRevocation()* ABI. Fig. 6(c) shows the token information after the execution of the ABI, where the token information of both subjects is deleted.

E. Token Verification

After receiving the *read* token, the subject $addressB$ can pass the token to the smart contract to verify that it has the *read* right when it wants to read the object. Fig. 7(a) depicts the result when the subject $addressB$ sent the read request. The result shows that the subject $addressB$ is allowed to read the object. For comparison, Fig. 7(b) depicts a rejected execution request when the subject $addressB$ does not own the corresponding token.

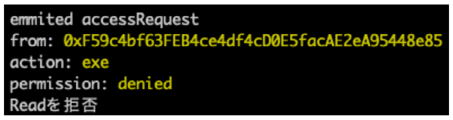
V. Related Work

In [15], a Bitcoin-like blockchain was implemented to achieve access control in a smart home application based on the Access Control List (ACL) model. The authors deployed a local blockchain in each home to store the ACL that controls the access requests from inside and outside of the home. Since the blockchain is maintained only by a single miner and the critical mining process is eliminated, the access control in each home becomes centralized and untrustworthy. The authors in [16] used the Bitcoin transactions to store access policies for an existing Attribute-Based Access Control (ABAC) scheme. In the ABAC model, each policy combines the attributes of subjects, objects, actions and context to provide dynamic and fine-grained access control. When receiving an access request, the ABAC scheme retrieves the related policies from the blockchain to perform the access control. Similar to [16], the Bitcoin transactions were used to store the tokens of the



```
emmited accessRequest
from: 0xF59c4bf63FEB4ce4df4cD0E5facAE2eA95448e85
action: read
permission: allowed
Readを実行
```

(a) Read request.



```
emmited accessRequest
from: 0xF59c4bf63FEB4ce4df4cD0E5facAE2eA95448e85
action: exe
permission: denied
Readを拒否
```

(b) Execution request.

Fig. 7. Access result when subject $addressB$ sends requests.

CapBAC model. By sending transactions among the subjects, the capability tokens can be delegated from one subject to another. When accessing an object, the subject passes its own capability token to the object owner, who then performs the access control by checking the validity of the token.

Recently, the Ethereum smart contract-based access control schemes have attracted considerable attentions. In [18], an ACL-based IoT access control framework was designed using multiple smart contracts. Each contract stores an ACL and the corresponding access control ABI for one subject-object pair. The authors also provided implementations to demonstrate the feasibility of the framework.In [19], a smart contract was deployed to maintain the roles assigned to each user in a Role-Based Access Control (RBAC) model, such that any service provider can verify the users' ownership of roles when providing services. An ABAC scheme was proposed in [20], which stores the URL links of policies on the blockchain and also deploys a smart contract for access control. When accessing an object, a subject sends the link of the related policy to the smart contract, which then retrieves the policy from external databases to achieve the access control. However, adversaries may be able to tamper with the polices without changing the URL links, resulting in untrustworthy access control.

VI. Conclusion

This paper proposed a Capability-Based Access Control (CapBAC) scheme by using Ethereum smart contracts to store and manage the capability tokens. Compared with the existing BLockchain-ENabled Decentralized Capability-based Access Control (BlendCAC) scheme, this scheme is expected to achieve more fine-grained access control and more flexible token management by defining capability tokens in units of actions and using a delegation graph to store the token delegation relationship among the subjects. Experiments based on a local Ethereum blockchain were conducted and the results demonstrated the feasibility of the scheme.

Acknowledgement

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] "Intel iot gateway." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/gateway-solutions-iot-brief.pdf
[2] "Mirai Botnet Linked to Dyn DNS DDoS Attacks." [Online]. Available: https://www.flashpoint-intel.com/ja/blog/cybercrime/mirai-botnet-linked-dyn-dns-ddos-attacks/
[3] U. Blase, Jung Jaeyeon, and S. Schechter, "The Current State of Access Control for Smart Devices in Homes," in *Workshop on Home Usable Privacy and Security*, 2013.
[4] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, and R. Boreli, "An experimental study of security and privacy risks with emerging household appliances," in *2014 IEEE Conference on Communications and Network Security*, 2014, pp. 79–84.
[5] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-Level Security and Privacy Control for Smart-Home IoT Devices," in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications*. IEEE, 2015, pp. 163–167.
[6] R. Coppola, M. Morisio, and P. Torino, "Connected Car: Technologies, Issues, Future Trends," *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–36, 2016.
[7] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-porisini, "Security, privacy and trust in Internet of Things : The road ahead," *COMPUTER NETWORKS*, vol. 76, pp. 146–164, 2015.
[8] J. Singh, T. Pasquier, J. Bacon, H. Ko, and D. Eyers, "Twenty Security Considerations for Cloud-Supported Internet of Things," *IEEE Internet of Things*, vol. 3, no. 3, pp. 269–284, 2016.
[9] A. Ouaddah, H. Mousannif, A. Abou, and A. Ait, "Access control in the Internet of Things: Big challenges and new opportunities," *Computer Networks*, vol. 112, pp. 237–262, 2017.
[10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," vol. 29p, no. 2, pp. 38–47, 1996.
[11] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, February 2015.
[12] "Bitcoin - open source p2p money." [Online]. Available: https://bitcoin.org/en/
[13] J. L. Vitalik Buterin, Gavin Wood, "Ethereum homestead documentation," https://github.com/ethereum/wiki/wiki/White-Paper.
[14] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
[15] 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*, January 2017, pp. 618–623.
[16] D. D. M. Francesco, P. Mori, and L. Ricci, "Blockchain based access control," *Proc. of IFIP International Conference on Distributed Applications and Interoperable Systems*, pp. 206–220, May 2017.
[17] 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, pp. 5943–5964, February 2017.
[18] 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*.
[19] J. P. Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12 240–12 251, March 2018.
[20] C. Dukkipati, Y. Zhang, and L. C. Cheng, "Decentralized, blockchain based access control framework for the heterogeneous Internet of Things," *Proc. of 3rd Workshop on Attribute Based Access Control*, pp. 61–69, March 2018.
[21] 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, July 2018.
[22] "Web3 javascript api to interact with ethereum nodes." available at https://github.com/ethereum/wiki/wiki/JavaScript-API.