TRABAC: A Tokenized Role-Attribute Based Access Control using Smart Contract for Supply Chain Applications

Aisyah Ismail, Qian Wu, Mark Toohey
Aglive Lab
P.O. Box 196
Geelong, Victoria, Australia
aisyah@aglive.com

Young Choon Lee
School of Computing
Macquarie University
Sydney, Australia

Zhongli Dong, Albert Y. Zomaya
School of Computer Science
The University of Sydney
Sydney, Australia
zhongli.dong@sydney.edu.au

2021 IEEE International Conference on Blockchain (Blockchain) | 978-1-6654-1760-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/BLOCKCHAIN53845.2021.00088

***Abstract*—The use of smart contracts for access control has shown to be promising as it ensures integrity and governs access to stored data, thanks to blockchain's immutability. While several recent studies have shown such usage, their applicability to supply chain applications remains limited due to less governance control capability and implementation complexity with smart contracts. This paper proposes the use of a tokenized role-attribute based access control (TRABAC) as a two-level access control for supply chain applications. In particular, TRABAC combines the simplicity of Role-Based Access Control (RBAC) and the flexibility and fine-grained capacity of Attribute-Based Access Control (ABAC). We consider these methods coupled with the use of Non-Fungible Token (NFT) as virtual assets in the supply chain. We also define four roles or parties that can have distinct and varied access rights. These roles are incorporated into TRABAC. The efficacy of TRABAC has been evaluated in five access control scenarios. Our experimental results show that TRABAC is capable of delegating access to four different supply chain roles. Importantly, TRABAC can effectively prevent unauthorized access requests by accounts that lack a valid Level 1 role or accounts that lack a valid token attribute or a tag in Level 2 of TRABAC.**

1. Introduction

A typical supply chain application involves multiple organizations or parties, from suppliers all the way along the chain until good reach consumers. The flow of data and information go back and forth between these organizations. Additionally, the supply chain involves many in-house departments with varying degrees of access or clearance for huge organizations. According to Zhou et al. [1], effective information sharing improves supply chain performance. Moreover, lack of end-to-end visibility in the supply chain may hinder the effectiveness of the supply chain [2]. Hence, accessing real-time information from every supply chain area is of the utmost importance.

Decentralized applications that use blockchain technology ensure data is stored in a reliable, trustworthy, and transparent manner. A leading blockchain for decentralized application is Ethereum. Ethereum was designed as a general-purpose programmable blockchain that uses smart contracts [3]. Smart contracts are a self-executing digital protocol that are designed to meet established conditions [4]. Many recognise the potential benefits of combining the use of smart contracts and blockchain technology to ensure the integrity of stored data. However, the question remains on how to best delegate access to the data stored on the blockchain?

Recent studies address the use of virtual tokens to validate and delegate access using smart contracts [5]–[7]. Two general categories of token are used to govern access control for virtual assets: fungible tokens and non-fungible token (NFT). A fungible token is exchangeable as each token's value remains the same. Fungible tokens are therefore similar to fiat currency, i.e. just as each note or coin holds a certain value, each fungible token can also be interchanged. The most widely used fungible token is the ERC-20[1] standard [8]. In contrast, each NFT can't be interchanged as it is unique and can only be used to record unique information in relation to a digital asset. The most common and popular NFT is the ERC-721 standard [9].

In this paper, we study the use of virtual asset tokens to achieve supply chain access control. TRABAC is a two-level tokenized role-attribute based access control method for supply chain applications. TRABAC combines the simplicity of Role-Based Access Control (RBAC) and the flexibility and fine-grained capacity of Attribute-Based Access Control (ABAC). It is coupled with the use of NFT that represent virtual assets in the supply chain.

Specific contributions of this paper are:

- We design a custom ERC-721 token to represent assets and attributes of both subjects and objects in supply chain applications.
- We devise a token child called "Activity" to record activities linked to the ERC-721 token.

1. ERC stands for Ethereum Request for Comment (ERC) with the following number (e.g. 20 or 721) being the proposal identifier. The token standard such as ERC-20 (fungible) and ERC-721 (non-fungible) define the set of rules and protocols for issuing Ethereum tokens.

978-1-6654-1760-0/21/$31.00 ©2021 IEEE
DOI 10.1109/Blockchain53845.2021.00088

- We define four roles to govern access to functions in the supply chain application.
- We build the TRABAC prototype as a proof of concept to show how the TRABAC model can be implemented using ERC-721. [2]

The rest of the paper is organized as follows. Section 2 covers background problems and existing solutions addressed in other research. In Section 3, we introduce the formal definition of the proposed access control model called TRABAC. In Section 4, we describe the prototype design and the experiment setup. We present experimental results in Section 5. Finally, we conclude the paper in Section 6.

2. Related Work

Access control is a system that delegates certain privileges to a subject over related objects. In the context of a supply chain application, the subjects are users from multiple organizations. The objects are protected data, information or documents stored in the system. There are several notable recent studies [10]–[12]. Section 2.1 examines commonly-used access control models and their applications. This is followed by a review of existing token-based access control schemes in Section 2.2.

2.1. Access Control Model

Nakamura et al. [10] described three commonly-used access control models: Role-Based Access Control (RBAC); Attribute-Based Access Control (ABAC); and Capability-Based Access Control (CapBAC). RBAC is coarse-grained and more suitable for a small organization or a closed environment. The use of RBAC is straightforward and easy to manage [12]. These constraints pose significant obstacles to the use of RBAC in a large-scale IoT network or application, particularly in a scenario where multiple domains interact without prior knowledge or trust between the domains [10], [11].

In contrast, the delegation of access in ABAC is based on a set of policies that support Boolean logic (e.g. IF-THEN). ABAC policies are based on the attributes of the subject (e.g. department, clearance level) and the attributes of an object (e.g. type, sensitivity) in addition to environmental conditions (e.g. time, location) [13]. Alternatively, the strength of CapBAC lies in the ability of the subject to delegate and revoke the access rights of other subjects, without the object owner's intervention, and using certificate-like capabilities [10], [11].

2.2. Token-Based Access Control

Blockchain tokens can either represent any real-life assets as a 'digital-twin' for digital assets or as a means of controlling admission to an application. We considered three recent studies that looked at token-based access control [5]–[7]. Guo et al. [6] used the ERC-20 token standard, while the other two papers used custom token designs [5], [7]. The general idea of token-based access control is that the smart contract will validate the token held by the subject when considering every request for access to an object. Access will be granted depending on the object access policies when compared to the token's metadata.

Guo et al. proposed the use of ERC-20 tokens to define attributes in ABAC [6]. Their implementation involves multiple authority nodes where each authority node validates and delegates one type of attribute or token. A subject may possess multiple attributes, ergo multiple tokens. Similarly, Almakhour et al. [5] also proposed a token-based access control with ABAC that requires a federated agreement among all of the involved parties before the smart contract is deployed. Access is thus granted by comparing the subject's token with the attributes that were pre-defined in the smart contract.

Liu C. et al. [7] proposed a fine-grained access control method that employed the 5W1H (who, what, where, why, when, and how) policies for IoT applications. The 5W1H policies are minted with their digital asset called Tokoin.

3. TRABAC: Tokenized Role-Attribute Based Access Control

This section sets out a formal definition of the proposed TRABAC model.

3.1. Level 1: Role-Based Access Control (RBAC)

Level 1 of the proposed work uses a traditional RBAC structure that involves administrators and users who hold a range of different roles. Sandhu et al. [14], defines the use of RBAC in Level 1 as:

- S, R, and P (subject, roles, and permission respectively);
- $SA \subseteq S \times R$, many-to-many subject-to-role assignment relationship;
- $PA \subseteq R \times P$, many-to-many role-to-permission assignment relationship.

Let R be the set of roles, P the set of permissions, and S the set of subjects. The subject-role assignment relationship is a Cartesian product of S and R, and permission-role assignment is a Cartesian product of R and P. Both of them define a many-to-many relationship between the elements of the sets. Hence, RBAC is defined as, $SA \subseteq S \times R$ and $PA \subseteq R \times P$.

3.2. Level 2: Tokenized Attribute-Based Access Control (ABAC)

The four main elements of ABAC are: subject; object; action; and context or environment. Furthermore, the assignment of attributes for subject and object gives ABAC a

2. https://gitlab.com/aglive-research/TRABAC

finer-granularity compared to other access control models. Assigned attributes, or attribute tags, can be adjusted to deliver more precise and targeted access delegation. The protected object and subject in the supply chain are asset. We tokenize supply chain by creating a single unified token called AgliveToken (AGL).

The basic token structure is:

$$AGL_{ID}[A] = \{tType, tTag, tMeta\} \quad (1)$$

where the parameters for AGL are describes as:

- ID, the identifier for associated token.
- A, the address of token holder or owner.
- $tType$, token type $= \{subject, object\}$.
- $tTag$, token attribute tag assigned to the token.
- $tMeta$, metadata attached to the token.

As shown in Equation 1, a single token AGL owned by A has its own ID with the description of three members. The three members in token AGL are the type of token ($tType$), the attribute tag of the token ($tTag$), and the metadata attached to the token ($tMeta$). The metadata can be in the form of a simple JSON file, image or link.

Once a block is added to the blockchain, that block of data is permanently stored and neither it nor its contents can be changed. This paper introduces a child to the token called Activity (AC) that tracks any changes to a token that had been added to the blockchain. It should be noted that the use of AC should only be used for token type objects.

AC is describes as:

$$AC[AGL_{ID}] = \{aType, aTag, aMeta\} \quad (2)$$

where the parameters are:

- AGL_{ID}, the AGL token ID attached to the Activity.
- $aType$, activity type.
- $aTag$, attribute tag assigned to the Activity.
- $aMeta$, metadata attached to the Activity.

Token AGL and AC have a one-to-many relationship where each token can have multiple ACs. Equation 2 describes how AC is related to the token by its identifier, AGL_{ID}. AC also includes a parameter $aType$ or activity type as a descriptive identifier. Similarly to token AGL, AC can have an attribute tag ($aTag$) and metadata ($aMeta$) attached to the AC.

4. Prototype Design

To test our proposed TRABAC model, we developed a proof of concept prototype using a simulated local Ethereum blockchain via Ganache, Truffle, and Solidity smart contract. We also used OpenZeppelin ERC-721 smart contract and its access control to speed up the process. OpenZeppelin contracts are stable and have gone through security audits by independent sources. Hence, it is safer to use an OpenZeppelin smart contract than to code a ERC-721 contract from scratch. This approach may help avoid or limit some security vulnerabilities.

The proposed work combines both RBAC by OpenZeppelin and the previously defined AGL token as a two-level access control. RBAC governs the first level by restricting access to functions in accordance with a users' role. The second level validates access to the requested object by authenticating the token attribute that is attached to either the token subject or the token object.

4.1. Ethereum Account

The experiment setup used a simulated Ethereum blockchain via Ganache CLI v6.12.1. We ran Ganache in a deterministic mode to ensure we get the same Ethereum addresses each time. By default, Ganache will produce ten Ethereum addresses that hold 100 ETH each. From hereinafter, the ten Ethereum addresses will referred as Address A, B, C, D, E, F, G, H, I, and J.

4.2. Roles in RBAC

There are three approaches to use OpenZeppelin RBAC: community; administered; and hierarchy [15]. We use the administered approach as it employs the traditional RBAC setup that includes both administrators and users. Also, multiple roles can be defined or added to the contract with a DEFAULT_ADMIN_ROLE for the contract owner. The supply chain workflow involves the interaction of subjects from different organizations that are involved at different sections of the supply chain. Consequently, this paper defines four roles: DEFAULT_ADMIN_ROLE, MODERATOR_ROLE, CUSTODIAN_ROLE, and USER_ROLE. Table 1 describes the use of each role to govern access control in a supply chain.

TABLE 1. PROPOSED RBAC ROLES IN TRABAC.

Role	Descriptions
DEFAULT_ADMIN_ROLE	The DEFAULT_ADMIN_ROLE is assigned to contract owner by default. It can grant and revoke permissions to any account.
MODERATOR_ROLE	The MODERATOR_ROLE can create and read token type subject.
CUSTODIAN_ROLE	The CUSTODIAN_ROLE can create and read token type object. The CUSTODIAN_ROLE can also create and read attached activities (AC) for token object.
USER_ROLE	The USER_ROLE can read information from token type object.

By default, the contract owner account will have the role of DEFAULT_ADMIN_ROLE (Address A). The contract owner will then delegate the role of MODERATOR_ROLE (Address B), CUSTODIAN_ROLE (Address C), and USER_ROLE to other accounts (Address D, E, F, G, H, I, and J). As a result, an account with MODERATOR_ROLE will have access to Level 2 of the proposed model to create a token type subject. In comparison, an

account with CUSTODIAN_ROLE can create a token type object. Finally, an account with USER_ROLE can only read information from a token type object.

Please note that these distinct roles did not have overlapping access rights. In particular, while DEFAULT_ADMIN_ROLE could assign any address to any role, the DEFAULT_ADMIN_ROLE itself cannot act in those roles (i.e. create and read tokens). This restriction was intentional as limiting the use of an account or address, reduces the chance of an account being compromised.

5. Experiment and Results

This section presents five access control scenarios in a supply chain. Section 5.1 demonstrates two scenarios with the creation of token subject and token transfer by the MODERATOR_ROLE. In Section 5.2, we simulate two scenarios where CUSTODIAN_ROLE creates a token object and also creates activities tied to the token object. Finally, we show token access requests from all ten address accounts in Section 5.3. Finally, we evaluate the cost of running the TRABAC smart contract in Section 5.4.

Figure 1 provided a guideline for all figures related to token creation, transfer, and access requests in this section. It shows the actors involved in TRABAC where Admin is an account with DEFAULT_ADMIN_ROLE while Moderator is an account with MODERATOR_ROLE. Custodian is an account with CUSTODIAN_ROLE, and User is an account with USER_ROLE. Additionally, Custodian and User may hold AGL tokens as shown in fourth and sixth actor from top left in Figure 1. Lastly, it also shows the relationship between an Activity (or AC) and an AGL token.

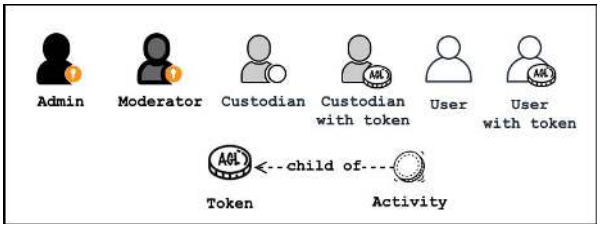

Figure 1. Actors and elements in TRABAC.

5.1. Token Creation and Token Transfer by MODERATOR_ROLE

In our supply chain simulation, Address A delegated the MODERATOR_ROLE to Address B. The purpose of the MODERATOR_ROLE is to create a token type subject with a specific $tTag$ and delegate the tokens to accounts with a CUSTODIAN_ROLE or USER_ROLE. Consequently, the token transfer involves the handover of ownership for any token (subject or object) between subjects in the system.

Accordingly, Address B exercised its MODERATOR_ROLE right by creating seven token with a variation of $tTag$ and delegated the tokens to CUSTODIAN_ROLE and USER_ROLE addresses. Figure 2 shows delegation of the seven tokens to Addresses C, D, E, F, G, H, and I.

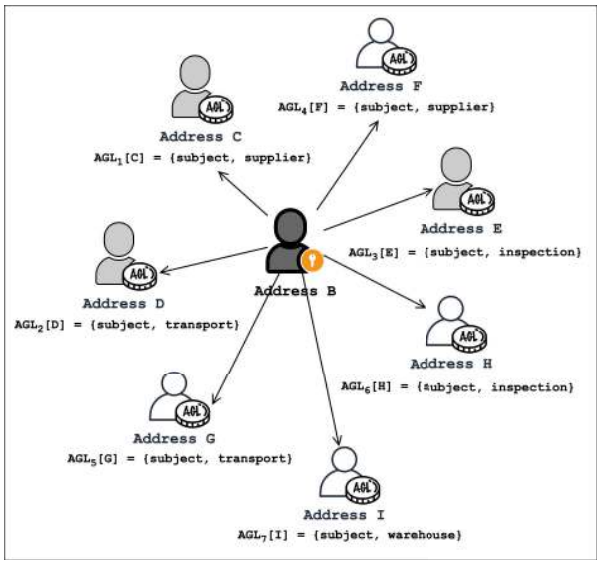

Figure 2. Delegation of token subject.

With TRABAC, Level 1 of the access control ensures only accounts with MODERATOR_ROLE are allowed to create and transfer the token subject ($tType = subject$). If another account (DEFAULT_ADMIN_ROLE, CUSTODIAN_ROLE or USER_ROLE) attempts to create a token with $tType = subject$, an error message will result.

5.2. Token and Activity Creation by CUSTODIAN_ROLE

As mentioned in Section 4.2, Address A granted three accounts with CUSTODIAN_ROLE (Addresses C, D, and E). Figure 2 shows how Address B with MODERATOR_ROLE granted Address C, D, and E with different $tTag$s. The three accounts are able to create token object and token child AC. However, they are limited to create token object and AC with their respective $tTag$ only. For example, Address C holds AGL token with $tTag = supplier$. As such, Address C can only create a token object and AC with $tTag$ or $aTag$ '$supplier$' only. Figure 3 illustrates the creation and access of token and activity by CUSTODIAN_ROLE and USER_ROLE in our supply chain simulation.

Figure 3 show how Address C can create token $AGL_8[C]$ with $tType = object$ and $tTag = supplier$. On the other hand, Address D which is in possession of token $AGL_2[D]$ can create token $AGL_9[D]$ with $tType = object$ and $tTag = transport$. Both Address C and D can only create a token object with the same $tTag$ token subject for any token in their possession.

Figure 3 also shows how Addresses C, D, and E can create AC for their respective $tTag$. Address C created $AC[AGL_8]$ which is a child of token $AGL_8[C]$ with

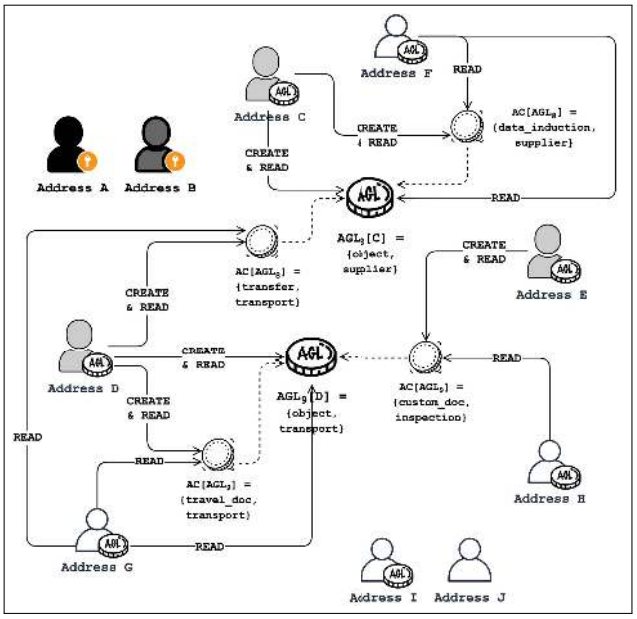

Figure 3. Creation and access of token and activity in TRABAC.

$aType = data\_induction$ and $aTag = supplier$. On the other hand, Address D created two ACs for token $AGL_8[C]$ and $AGL_9[C]$, respectively. The first AC is $AC[AGL_8]$ with $aType = transfer$ and $aTag = transport$ while the second AC is $AC[AGL_9]$ with $aType = travel\_doc$ and $aTag = transport$. Finally, Address E which holds $tTag = inspection$ created $AC[AGL_9]$. The $AC[AGL_9]$ is the child of token $AGL_9[D]$ with $aType = custom\_doc$ and $aTag = inspection$.

If other accounts that are without a CUSTODIAN_ROLE try to create a token object, they will receive an error message. Even accounts with DEFAULT_ADMIN_ROLE or MODERATOR_ROLE will not have the ability to create a token object. The create activity (AC) function, is only available to a CUSTODIAN_ROLE if it has a valid $tTag$ can create them.

5.3. Access Request by CUSTODIAN_ROLE and USER_ROLE

Similar to token creation, only accounts with a CUSTODIAN_ROLE and USER_ROLE will be able to request access to tokens or ACs. However, both roles are restricted to only access tokens or ACs that possess the same $tTag$ as illustrated in Figure 3. Both the DEFAULT_ADMIN_ROLE and the MODERATOR_ROLE are not able to access a token as they do not hold any tokens.

Figure 3 show how Address C and Address F will only be able to read token $AGL_8[C]$ and $AC[AGL_8]$ as both accounts have token with $tTag = supplier$. For accounts with $tTag = transport$, Address D and Address G can read token $AGL_9[D]$, $AC[AGL_8]$, and $AC[AGL_9]$. While Address E and Address H can only read $AC[AGL_9]$ because both accounts have $tTag = inspection$ in their possession.

On the other hand, Address I cannot read any of the tokens or AC even though the account in possession of AGL token. As shown in Figure 2, Address B delegate token AGL_7 with $tTag = warehouse$ to Address I. Because there is no token or AC with the same $tTag$, Address I will not be able to access any token object or AC in the experiment. For Address J, the account does not possess any AGL token. Thus, it will be automatically prevented from accessing any asset (token or AC).

5.4. Cost Evaluation

Each transaction on the Ethereum blockchain incur charges that are calculated using the Ethereum Gas unit. Each transaction that executes a smart contract has a fixed base cost. Additionally, the extra cost depends on the complexity of the smart contract and the amount of data stored in the blockchain.

Table 2 presents the average cost of functions in TRABAC during the course of the experiment as discussed in previous subsections. The amount of gas used for functions executions varies depending on the amount of bytecode to process. The more complex the input, the higher the gas used, which then leads to a higher execution cost.

TABLE 2. AVERAGE COST OF FUNCTIONS IN TRABAC.

Function	Gas Used
addModerator	90,589
addCustodian	80,567
addUser	79,317
createToken	304,497
transferToken	166,229
addActivity	247,81

Figure 4 shows the amount of gas used during the token creation and token transfer for all nine tokens. Obviously, the more gas used, the the higher the transaction cost. The creation of token AGL_1 is the most expensive. Token creation for subsequent six subject tokens used about the same amount of gas, but slightly lower than token AGL_1. Following this trend, token creation for all token objects used the same amount of gas and slightly lower than the creation of token subject.

The work by Nakamura et al. [10] shows similar findings with the pattern of gas consumed for token creation and where creation of the first token consumed more gas than the creation of the subsequent tokens. Moreover, they discovered the relationship between the amount of gas used and the length of input [10]. By the same notion, we list the actual gas consumed for each token creation in Table 3.

Based on Table 3, we found that creation of the first token subject cost an additional 60,000 gas unit compared to creation of other token subjects. Moreover, creation of token subjects cost an additional 95,775 gas units than the

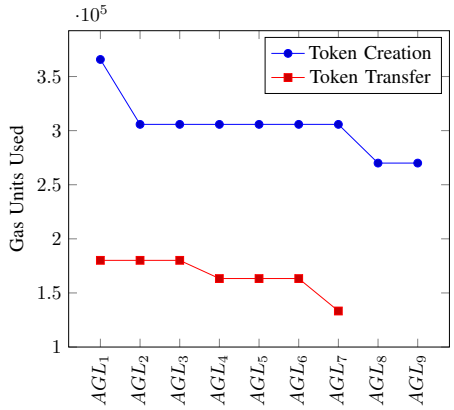

Figure 4. Cost for token creation and token transfer for all nine token.

creation of token objects. It is also worth noting that each character in the token tag costs 12 gas units. For example, creation of token AGL_3 cost an extra 12 gas units compared to the creation of token AGL_2.

TABLE 3. COST FOR CREATION OF TOKEN.

Token	Token Type	Token Tag	Gas Used
AGL_1	subject	supplier	365,770
AGL_2	subject	transport	305,782
AGL_3	subject	inspection	305,794
AGL_4	subject	supplier	305,770
AGL_5	subject	transport	305,782
AGL_6	subject	inspection	305,794
AGL_7	subject	warehouse	305,782
AGL_8	object	supplier	269,995
AGL_9	object	transport	270,007

Similarly, it appears that there is also a pattern in the amount of gas used during the token transfer (Figure 4). The first three token transfers used the same amount of gas, followed by another three token transfers that cost the same amount of gas - albeit slightly lower. There was no token transfer for token AGL_8 and AGL_9 during the experiment.

6. Conclusion

In this paper, we have addressed the problem of supply chain access control with the use of smart contracts. Our proposed access control model, TRABAC has demonstrated its effectiveness in controlling access between four different parties, and ensuring data integrity and governance. In particular, the two-level access control model of TRABAC is able to deliver flexible and fine-grained access control and consequently cater to the need of supply chain applications.

Our experimental results with a prototype of TRABAC confirm our claims. Other notable observations in this paper are:

- TRABAC is able to address the need of multi-organization with multi-admin in the supply chain application with the proposed four roles without overlapped privileges;
- Level 1 of TRABAC filters unwanted access to smart contract functions easily with RBAC; and
- ABAC on Level 2 of TRABAC provides fine-grained access control and limits access to only accounts that have the correct attributes.

As each transaction on Ethereum blockchain incurs costs, we plan to seek ways to reduce contract size for cost reduction. We also plan to investigate the ability to include an environmental condition as a token tag attribute.

References

[1] H. Zhou and W. Benton, "Supply chain practice and information sharing," *Journal of Operations Management*, vol. 25, pp. 1348–1365, 2007.

[2] M. Rogerson and G. Parry, "Blockchain: case studies in food supply chain visibility," *Supply Chain Management: An International Journal*, vol. ahead-of-print, 05 2020.

[3] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Yellow Paper*, June 2020. [Online serial]. Available: https://ethereum.github.io/yellowpaper/paper.pdf.

[4] N. Szabo, "Smart contracts," 1994. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[5] M. Almakhour, L. Sliman, A. E. Samhat, and W. Gaaloul, "Trustless blockchain-based access control in dynamic collaboration," in *Proceedings of the 1st International Conference on Big Data and Cyber-Security Intelligence (BDCSIntell 2018)*, vol. 2343 of *CEUR Workshop Proceedings*, pp. 27–33, CEUR-WS.org, 2018.

[6] H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," *Cryptography and Security (cs.CR)*, 2019.

[7] C. Liu, M. Xu, H. Guo, X. Cheng, Y. Xiao, D. Yu, B. Gong, A. Yerukhimovich, S. Wang, and W. Lv, "Tokoin: A coin-based accountable access control scheme for internet of things," 2020.

[8] F. Vogelsteller and V. Buterin, "Eip-20: Erc-20 token standard," *Ethereum Improvement Proposals,* no. 20, November 2015. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-20.

[9] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "Eip-721: Erc-721 non-fungible token standard," *Ethereum Improvement Proposals,* no. 721, January 2018. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-721.

[10] Y. Nakamura, Y. Zhang, M. Sasabe, and S. Kasahara, "Exploiting smart contracts for capability-based access control in the internet of things," *Sensors*, vol. 20(6), no. 1793, 2020.

[11] 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(3), no. 39, 2018.

[12] L. Song, M. Li, Z. Zhu, P. Yuan, and Y. He, "Attribute-based access control using smart contracts for the internet of things," in *Procedia Computer Science*, vol. 174, pp. 231–242, Elsevier B. V., 2020.

[13] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone., "Guide to attribute based access control (abac) definition and considerations," NIST Special Publication 800-162, National Institute of Standards and Technology, 2014.

[14] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE computer*, vol. 29, no. 2, pp. 38–47, 1995.

[15] A. C. Cañada, "How to use accesscontrol.sol," Aug 2020.