A domain-based access control mechanism using smart contracts for the internet of things

Kobaruto Sato1, a) and Masaki Bandai1
1 Department of Information and Communication Sciences, Sophia University
7-1 Kioicho, Chiyoda-ku, 102-8554, Japan
a) k.sato@bandailab.jp

Abstract: This paper addresses the scalability in the cost of smart contracts, which prevent illegal resource access (control access) in the Internet of Things. In previous studies, smart contracts have been used to avoid having a single point of failure, while maintaining tamper resistance. However, a high cost (called gas) is necessary when the number of subject-object pairs becomes large. In order to reduce cost, this paper proposes an access control method using smart contracts with a hierarchical name space. The proposed method scales well in terms of cost-decisive factors, namely, the complexity of functions and the number of smart contracts, leading to scalability in cost.

Keywords: internet of things, smart contracts, access control

References

[1] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, “Smart contract-based access control for the internet of things,” IEEE Internet Things J., vol. 6, no. 2, pp. 1594–1605, 2019. DOI: 10.1109/jiot.2018.2847705
[2] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, “A novel attribute-based access control scheme using blockchain for IoT,” IEEE Access, vol. 7, pp. 38431–38441, 2019. DOI: 10.1109/access.2019.2905846
[3] O. Aafaf, E. Anas, and O. Abdellah, “Fairaccess: a new blockchain-based access control framework for the Internet of Things: fairaccess: a new access control framework for IoT,” Security and Communication Networks, vol. 9, no. 18, pp. 5943–5964, Feb. 2017. DOI: 10.1002/sec.1748
[4] M. Yutaka, Y. Zhang, M. Sasabe, and S. Kasahara, “Distributed and finegrained attribute-based access control for the internet of things,” IEICE Technical Report, vol. 118, no. 465, NS2018-251, pp. 343–348, March 2019 (in Japanese).
[5] M. Yutaka, Y. Zhang, M. Sasabe, and S. Kasahara, “Monetary cost evaluation for IoT attribute-based access control framework implemented on Ethereum blockchain,” IEICE Technical Report, vol. 119, no. 460, NS2019-190, pp. 77–82, March 2020 (in Japanese).
[6] S. Tanzeela, A. Ahmad, A. Mariam, Z. Mansour, U. Ibrar, and J. Nadeem, “Data sharing system integrating access control mechanism using blockchain-based smart contracts for IoT devices,” Applied Sciences, vol. 10, no. 2, p. 488, Jan. 2020. DOI: 10.3390/app10020488
1 Introduction

A growing number of devices are openly connected online, forming the Internet of Things (IoT). IoT devices contain data including confidential or private information under constrained environments. Preventing illegal resource access (control access) is a major issue [1, 2]. Most access control solutions provide the ability for centralized authorities [3]; however, these systems are still exposed to a single point of failure. To avoid having this single point of failure, smart contracts can be used [1, 4, 5, 6]. A smart contract is a piece of Ethereum Virtual Machine code that may be associated with an account (identified by 160-bit address) [7]. Every function call message (transaction) to the address of the smart contract is broadcast, executed and verified by all network participants (miners) based on a consensus protocol. As a reward to miners for consuming their computational power and storage, a gas cost [7] is imposed on the account of the transaction sender. According to the definition of gas [7], the `store` and `create` operations, which store a new value and create a smart contract, respectively, are generally expensive. Other single operations are cheaper, but become expensive if iterated many times.

Reducing the gas cost for IoT access control is an important issue [5, 6]. In the system of [1], a `create` operation is required for every subject-object pair. Also, in the systems of [1, 6], `store` operations to set policies are required for every subject-object pair. Then, the system of [4] provides the ability to store a policy for many-to-many pairs, where subjects and objects are given attributions and then controlled by the attributions. Compared with the system of [1], the gas cost to store policies in the system [4] is reduced when a large average number of pairs are handled by each policy [5]. However, the gas cost to conform a permission increases linearly with the number of policies, while the counterpart of the system of [1] is constant. This is because the system of [4] iterates over all policies to find the policy associated with the subject and object. This search is necessary when conforming to a permission, and causes the gas cost to be high when the number of policies becomes large.

In this paper, in order to reduce the gas cost, we propose an access control system using smart contracts with a hierarchical name space as attribution. By using the hierarchy, it is no longer necessary to iterate over all policies to find an associated policy. The method scales well in terms of cost-decisive factors, namely, the complexity of transactions and the number of smart contracts, leading to scalability in terms of cost.

2 Proposed method

The flow of access control using smart contracts is shown in Figure 1. Suppose that a subject (userA) wants to read the resource (file1) of an object (dev1). The subject sends a transaction to a smart contract (Access Control Contract). Then, the subject
Fig. 1. Access control with smart contracts in the proposed method

and the object that has the resource know the permission through the transaction output. Based on the permission, the object responds or ignores the request.

The proposed method consists of the following five contracts:

- **Subject contract** - manages the names of subjects (request senders)
- **Object contract** - manages objects (service senders)
- **Policy contract** - manages policies
- **Domain contract** - manages domain owners
- **Access control contract** - manages the overall flow

Here, a domain refers to a set of subjects and the files that objects possess. In this design, an object itself creates a domain.

### 2.1 Subject, object, and domain contracts

All subjects, objects, and domains are given their own *hierarchical name*. A *hierarchical name* is defined as a concatenated string that consists of three parts: its own name, the separator "." and the hierarchical name of the domain it belongs to (except for the root domain, which has an empty string as its name). Based on the hierarchical name, a name space is considered, as illustrated in Figure 1. A subject contract has a hash table (map), named `nameOf`, which maps subject addresses to hierarchical names. An object contract has a map, named `nameOf`, which maps object addresses to hierarchical names. A domain contract has a map, named `ownerOf`, which maps hierarchical names to the addresses of domain owners.

### 2.2 Policy contract

A policy contract has a two-dimensional map, named `policy`, which is described as follows:

- The first key is a subject set name.
  
  (e.g., student1.sophia, .sophia)
• The second key is a resource set name.
  (e.g., file1.dev1.ics.sophia, .dev1.ics.sophia, .ics.sophia, .sophia)
• The mapped value is a permission, which is a 4-bit value for all-forbidden, executing, writing, and reading.

Here, a subject set name can be a domain name or a subject name, referring to a set of subjects. A resource set name can be a domain name or a file name before the separator "." and the name of domain the file belongs to, referring to a set of files. As a rule, the first character of a set name is the separator "." if it is a domain name.

2.3 Access control contract

An access control contract manages the overall flow, calling other contracts as shown in Figure 1. This contract has two main transactions: setPolicy and permission as shown in Algorithm 1.

Algorithm 1 Access Control Contract

1: Contract AccessControlContract:
2:   Transaction setPolicy(src, domainName, permission, fileName=""):
3:     if DomainContract.ownerOf[domainName].equal?(TX_SENDER)
4:       dst = fileName + "." + domainName
5:       PolicyContract.set(src, dst, permission)
6:       return accepted
7:     end_if
8:     return denied
9:
10:  Transaction permission(subjectAddr, objectAddr, fileName):
11:    sbj_name = SubjectContract.nameOf[subjectAddr]
12:    sbj_parts = sbj_name.split("")
13:    res_name = fileName + "." + ObjectContract.nameOf[objectAddr]
14:    res_parts = res_name.split("")
15:    permission = 0
16:    dst = ""
17:     for each of res_parts from last with ri: index, part_ri : res_parts[ri]
18:       dst = joinPart(dst, part_ri, ri)
19:       src = ""
20:         for each of sbj_parts from last with si: index, part_si : sbj_parts[ri]
21:           src = joinPart(src, part_si, si)
22:           permission = PolicyContract.get(src, dst) only if policy is defined
23:         end_for
24:     end_for
25:    return permission
26:
27:    Function joinPart(org, part_i, i):
28:      return part_i + org only if i = 0
29:      return "." + part_i + org
2.3.1 setPolicy transaction
The transaction has an average time complexity of $O(1)$. The functionality of the transaction is simply to set an entry to the map $policy$ in the policy contract (line 5). Prior to this, the map $ownerOf$ in the domain contract is referred to prevent illegal settings (line 3). In general, the map can be set or read with an average complexity of $O(1)$. The address of the transaction sender can be referred to from the inside of the smart contract, as per TX_SENDER (line 3).

2.3.2 Permission transaction
This transaction uses a policy to control access, and has an average time complexity of $O(H^2)$, where $H$ is the average number of layers in the name. This transaction finds an associated policy by enumerating all possible patterns in the subject set and resource set (lines 17 to 24). If multiple associated policies exist, the primary policy that is used in Algorithm 1 is that which has the smallest subject set among the policies that have the smallest resource set.

3 Evaluation
The proposed method is compared to the related methods of [1, 4, 6] in Table I(a). We evaluated the following cost-decisive factors:

**Time complexity** - how many iterations are necessary, on average

**Storage complexity** - how many store operations [7] are necessary, on average

**Number of smart contracts** - the number of required create operations [7]

We assumed that one associated policy should exist for each of the subject-resource pairs. The definitions of the variables in Table I(b) are as follows:

Table I. Comparison of cost decisive factors

(a) Complexity for setting policies

|        | time complexity | storage complexity |
|--------|-----------------|--------------------|
| [1]    | $O(X \cdot R)$  | $O(X \cdot R)$     |
| [4]    | $O((X \cdot R)^2)$ | $O(X \cdot R)$     |
| [6]    | $O(X \cdot R)$  | $O(X \cdot R)$     |
| Proposed | $O(\frac{X \cdot R}{A})$ | $O(\frac{X \cdot R}{A})$ |

(b) Complexity for conforming a permission

|        | time complexity | storage complexity |
|--------|-----------------|--------------------|
| [1]    | $O(1)$          | $O(1)$             |
| [4]    | $O(\frac{X \cdot R}{A})$ | $O(1)$             |
| [6]    | $O(1)$          | $O(1)$             |
| Proposed | $O(H^2)$        | $O(1)$             |

(c) Number of smart contracts

|        | number of smart contracts |
|--------|---------------------------|
| [1]    | $2 + X \cdot Y$          |
| [4]    | 4                         |
| [6]    | 3                         |
| Proposed | 5                        |
$X$ - number of subjects  
$Y$ - number of objects  
$R$ - number of files  
$A$ - average number of pairs that are handled together by one policy  
$H$ - average number of layers in name; e.g., there are three layers in ”a.e.com”.

$X \cdot R$ becomes potentially large as an increasing number of subjects and objects are connected in the IoT. $A$ is also potentially large if many subjects and objects belong to an organization. $H$ is supposed to be much smaller than $X \cdot R$.

The complexity for setting policies is compared in Table I(c). The systems of [1, 6] set policies with an average storage complexity of $O(X \cdot R)$. This is because one policy is set for each of the $X \cdot R$ pairs. On the other hand, the system of [4] and the proposed system aggregate $A$ policies to an average of one policy, dividing the storage complexity by $A$, which leads to fewer store operations (one of the most expensive operations).

The complexity for conforming a policy is compared in Table I(b). The time complexity of the systems of [1, 6] is $O(1)$. This is because the permission can be easily mapped from the subject-resource pair, for policies that differ from pair to pair. An iterative search is necessary to find a policy with $O(XY/A)$ (time) for the system of [4] and $O(H^2)$ (time) for the proposed system. Based on the assumption that $H^2$ is acceptable compared with $(X/R)$, the proposed method works well, regardless of how many $X$, $Y$, and $R$ there are.

The number of smart contracts is compared in Table I(c). Although the number of smart contracts is larger than the systems of [4, 6], similar to the systems of [4, 6], it is a constant.

4 Conclusion

This paper proposed an access control system using smart contracts with a hierarchical name space as attribution in order to reduce gas cost in terms of the time complexity of transactions, while reducing the storage complexity of transactions and the number of smart contracts. In conclusion, the proposed method scales well in terms of cost-decisive factors, namely, the time and storage complexity of transactions and the number of smart contracts, which leads to scalability in terms of cost.