Computers, Materials & Continua
DOI: 10.32604/cmc.2022.023496
Article

Tech Science Press

A Traceable Capability-based Access Control for IoT

Chao Li[1], **Fan Li**[1,2], **Cheng Huang**[3], **Lihua Yin**[1,*], **Tianjie Luo**[1,2] **and Bin Wang**[4]

[1]Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, 510700, China
[2]PCL Research Center of Cyberspace Security, Peng Cheng Laboratory, Shenzhen, 518052, China
[3]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, N2L 3G1, Canada
[4]College of Electrical Engineering, Zhejiang University, Hangzhou, 310058, China
*Corresponding Author: Lihua Yin. Email: yinlh@gzhu.edu.cnc
Received: 10 September 2021; Accepted: 02 December 2021

Abstract: Delegation mechanism in Internet of Things (IoT) allows users to share some of their permissions with others. Cloud-based delegation solutions require that only the user who has registered in the cloud can be delegated permissions. It is not convenient when a permission is delegated to a large number of temporarily users. Therefore, some works like CapBAC delegate permissions locally in an offline way. However, this is difficult to revoke and modify the offline delegated permissions. In this work, we propose a traceable capability-based access control approach (TCAC) that can revoke and modify permissions by tracking the trajectories of permissions delegation. We define a time capability tree (TCT) that can automatically extract permissions trajectories, and we also design a new capability token to improve the permission verification, revocation and modification efficiency. The experiment results show that TCAC has less token verification and revocation/modification time than those of CapBAC and xDBAuth. TCAC can discover 73.3% unvisited users in the case of delegating and accessing randomly. This provides more information about the permissions delegation relationships, and opens up new possibilities to guarantee the global security in IoT delegation system. To the best of our knowledge, TCAC is the first work to capture the unvisited permissions.

Keywords: IoT access control; permission delegation; delegation trajectory; capability revocation; capability modification

1 Introduction

Internet of Things (IoT) has been discussed and deployed in various fields such as smart home, transportation, healthcare, and industrial automation [1–4]. This brings many security challenges [5–7]. The OWASP IoT project shows that the vulnerabilities of IoT devices are mainly focused on insecure access to IoT applications [8]. A lot of works focus on the IoT access control to avoid unauthorized access [9].

Delegation is an authorization mechanism in IoT system, which aims to make permissions transmission more convenient. It allows users to share permissions from their permission set to other users. For example, Alice delegates her car's smart key to Bob, and asks him to help retrieve her bag from her car. Bob can use the smart key to open the car door, but he cannot start the car. In this case, the permission set of Alice smart key is {*open car door*, *start the car, play music*, ...}. Alice only shares Bob with the {*open car door*} permission. Some products provide cloud-based delegation services to users to achieve permissions delegation, such as SmartThings [10], IFTTT [11] and GoogleHome [12]. However, the cloud-based solutions require users to register in the cloud. For example, if Alice wants to delegate the {*open car door*} permission to Bob, she must first ask Bob to register in the cloud. It is not convenient when there are a large number of temporarily delegatees who have not registered in the cloud. Since the capability-based access control (CapBAC) [13] issues and delegates permissions by capability tokens, the authorization process can be executed on smart devices locally [14]. It delegates the permissions in an offline way without registeration, and this makes permissions delegation user-kindly.

However, offline delegation in CapBAC are independently determined by users, which are more likely to be wrong or to conflict than those made by administrators. CapBAC allows users to delegate permissions according to their own security constraints. This leads no administer to know all the permission distribution, and brings difficulties in permissions revocation and modification. To solve the problem, CapBAC provides three functions to revoke delegated capabilities: *IdentifiedCapabilityOnly*, *AllCapabilities* and *DependentCapabilitiesOnly*. They are not flexible enough to deal with various changes in capability delegation relationships. For example, if an apartment is sold, and the new owner wants to retain the {*open smart lock*} permissions of all the tenants. Each tenant's {*open smart lock*} permission may be delegated to his family, which forms the delegation relationships. The old owner of the apartment will be deleted in the delegation trajectory while preserving the permissions delegation relationships of tenants. The new owner will be added in the delegation trajectory to replace him. The functions CapBAC provides can not address such problems.

To extend CapBAC, existing methods utilize a revocation/modification list to address delegation changes. However, these methods do not change the issued capability tokens which are still valid. These valid tokens bring more risks to the system. What's more, as soon as the revocation/modification list is maliciously tampered, any access in the system can not be trusted. In addition, the capability token used in the CapBAC contains the previous token. The nested tokens need to be decrypted iteratively to trace back the first delegator who is the capability owner, to verify the legitimacy of the capability token. This operation will be repeated every time when the token is used. We find that adding a tracking function can solve above problems. The permissions delegation trajectories can be used to simplify token verification, revocation and modification in IoT delegation system, which can solve these problems.

We propose a traceable capability-based access control approach (TCAC), which uses a new time capability tree (TCT) to track permissions delegation trajectories. It provides a concise legality verification, and a fine-grained permissions revocation and modification. It can also track permissions which have been delegated but not used, which obtains more delegation information and makes it possible to analyze the global permissions delegation relationships. This enhances the security of the IoT delegation system. Our contributions are as follows:

- We propose a trajectory extracting method to extract the trajectories of delegated permissions. By using the time capability tree with new capability token, all delegation trajectories will be obtained automatically.

- Based on the trajectory extracting method, we propose the traceable capability-based access control approach (TCAC). It supports fine-grained permissions delegation, revocation and modification. To the best of our knowledge, TCAC is the first work to capture unvisited permissions, which can track permissions delegated but not used.
- The experimental results show that TCAC has obvious advantage in token verification, token revocation/modification and delegation trajectory capture than those of CapBAC and xDBAuth. It can discover 73.3% of unvisited users when all users delegate and access randomly.

The rest of the paper is organized as follows. In Section 2, we describe related works. Section 3 presents the architecture of TCAC. The token verification, delegation, revocation and modification will be given in Section 4. Section 5 shows the experiments of TCAC. Section 6 summarizes our work.

2 Related Works

Access control methods prevent unauthorzied access from subjects to objects. In an IoT context, attribute based access control (ABAC) [15] is widely used, while it do not provide flexible and user-kindly permissions delegation features. CapBAC allows users to delegate permissions with its capability tokens. Recently, several works have further explored CapBAC. Anggorojati et al. [16] combine delegation mechanism with capability propagation to expand the flexibility of IoT access control. They propose capability-based context aware access control (CCAAC), which is used to adapt the scalability and heterogeneity of IoT networks. But the requirement for prior knowledge limits the use of CCAAC. Hernández-Ramos et al. [17] propose a distributed capability-based access control model (DCapAC) to expand the usage scenarios. It is directly deployed on resource-constrained devices in a distributed IoT system. However, the capability delegation and revocation are not described. Xu et al. [18] propose a federated capability-based access control (FedCAC) framework, which presents the whole token processing. FedCAC delegates part of the identity verification and authorization tasks to the domain delegator. However, the establishment of domain delegator is not suitable for the resource-constrained IoT environment. BlendCAC [19] is subsequently proposed as the follow-up work. It combines blockchain and CapBAC, and uses smart contracts to delegate capability tokens. But the token revocation and modification are not fine-grained enough. Pal et al. [20] use a hybrid method of attributes, roles, and functions to provide a policy-based access control architecture. The delegation of capabilities is described in detail in [21]. However, explicit revocation of tokens is not supported in its decentralized design, and the centralized token distribution limits its application.

To solve the capability revocation and modification problem, the traceability needs to be added in CapBAC, which has been widely used in the cloud of data sharing. For example, Shen et al. [22] propose a traceable group data sharing scheme, to support anonymous multiple users in public cloud. In the case of access control, Guo et al. [23] add tracking function in the process of data entryption to support a dynamic access control. Li et al. [24] propose TRAC to track private key in CP-ABE, to identify the malicious user leaking his private key. Yu et al. [25] utilize the blockchain to achieve a traceable data sharing in industrial IoT. The revocation list is used in their scheme. The tracking function in CapBAC has not been proposed.

In this paper, we propose a traceable capability-based access control approach (TCAC). Its traceability is guaranteed by the defined time capability tree and the new capability token. TCAC can monitor the permissions delegation automatically, achieving a concise token verification and flexible permissions revocation and modification. What's more, the ability to capture the unvisited permissions can get more information of the permissions delegation relationships, which is benefit to guarantee the system security.

3 TCAC Architecture

The architecture of TCAC is shown in Fig. 1. *Center Server* is the platform which issues the root capabilities. *User Group* is the users in the delegation system. *Resource Server* is the object which users want to access. The summary of abbreviations is as follows:

- *PIP*: Policy Information Point, which gets the information needed in access control decision-making;
- *PAP*: Policy Administration Point, which manages the access control policies;
- *PDP*: Policy Decision Point, which makes access decisions according to the access control policies and information;
- *PEP*: Policy Enforcement Point, which forms the access control request to PDP and translates the result of decision to other modules;
- *TI*: Token issuer, which issues the capability tokens;
- *TV*: Token Verification, which verifies the validity of capability tokens;
- *TR*: Token Reconstruction, which generates the new capability tokens;
- *DT*: Delegation Trajectory, which is the global capability delegation trajectory.

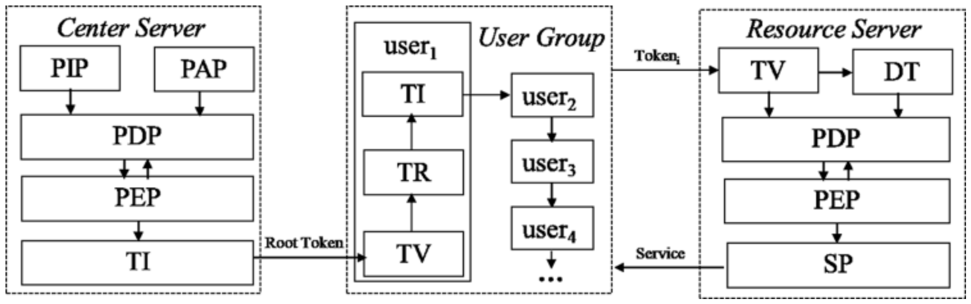

Figure 1: The architecture of TCAC

In the *Center Server*, after PDP makes access decisions with the information from PIP and access control policies from PAP, PEP translates the result to TI, which issues the root capability token to *user$_1$*. The validity of the root token can be verified by TV in *user$_1$*, then *user$_1$* writes the delegation information and his local time capability tree in the new capability token with TR. He can delegate the capability to all users in *User Group*. When a user accesses the resource, the *Resource Server* verifies the token by TV, extracts the local time capability tree, and combines it with other time capability trees to form a global tree as the permissions delegation trajectory, which is DT. It will be used for token verification, revocation and modification. If the PDP in *Resource Server* allows the user to access resource, the PEP translates the order to SP, which provides service to the user. Then a capability delegation is finished.

We introduce the TCAC architecture in this section, including the time capability tree, the capability token structure, and the authorization, encryption and verification methods.

3.1 Time Capability Tree

3.1.1 Local Time Capability Tree

In IoT delegation system, each user delegates his permissions to others. The one-to-multiple relationship is suitable to be saved by a tree, so we define a time capability tree to record the permissions delegation trajectory. The time capability tree is defined by a 5-tuple < *Resourse ID*, *Capability*, *User*

ID, *Effective Time*, *Access Records* >, which exists in each capability token of TCAC. As shown in Fig. 2, The *Resource ID* and The *Capability* in the *Title* record the user's permission to perform operations on resources, such as open the smart lock, close the car window, etc. In the *Tree*, *User ID* represents the unique identity of the user. *Effective Time* is the time interval when the capability is valid. *Access Record* is a series of timestamps that records the user's access time. Time capability tree exists in each capability token of TCAC. In TCAC, a permission delegation trajectory can be extracted from each capability token, and the global delegation trajectory can be obtained by combining all the time capability trees.

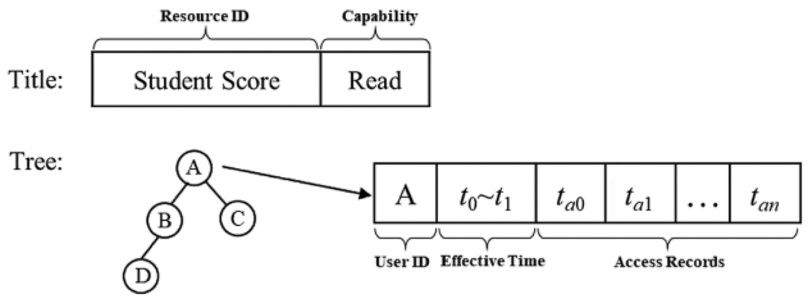

Figure 2: The structure of time capability tree

3.1.2 Combination of Global Time Capability Tree

Time capability tree exists in the token delegated to each user. Assuming that the time each user was given a token and each time capability tree was created are different, the delegators only get limited information about time capability tree. To obtain a globle time capability tree, all time capability trees received must be combined in the *Resource Server*. The combination of the time capability trees can be expressed as the following:

$$Tree_{Global} = Tree_1 \cup Tree_2 \cup \ldots \cup Tree_3 \tag{1}$$

where $Tree_{Global}$ is the global time capability tree combined by the *Resource Sever*, and $Tree_i$ is the local time capability tree exists in the capability token. Fig. 3 shows the combination of global time capability tree. Initially, there is no time capability tree on the *Resource Server* until an access request arrives. After receiving the first token from *user A*, the *Resource Server* constructs a root time capability tree. At this time, there is only one *user A* node in the global time capability tree. Then *users B*, *C*, and *D* access to the resource. They show their tokens to the *Resource Server*, and the local time capability trees are extracted. Since the time capability trees of *user C* and *user D* come from *user A* and *user B* which are the delegators, *user C* and *D* are unware of others capability delegation operations. These two local time capability trees are not complete. Only if the *Resource Server* conbains them can we get the global capability tree.

If a new user does not access the *Resource Server*, which is defined as the unvisited user. Then the *Resource Server* cannot discover the unvisited token, and the user will not exist in the global time capability tree. The unvisited users are the mainly barrier of monitoring the delegation relationships of IoT system. In TCAC, the local time capability tree of a permission always records delegation information of other users, through these users do not participate in that permission's delegation. The delegation information make TCAC able to capture the unvisited users by combain the global time capability tree. The capture efficiency of unvisited users will be discussed in Section 5.

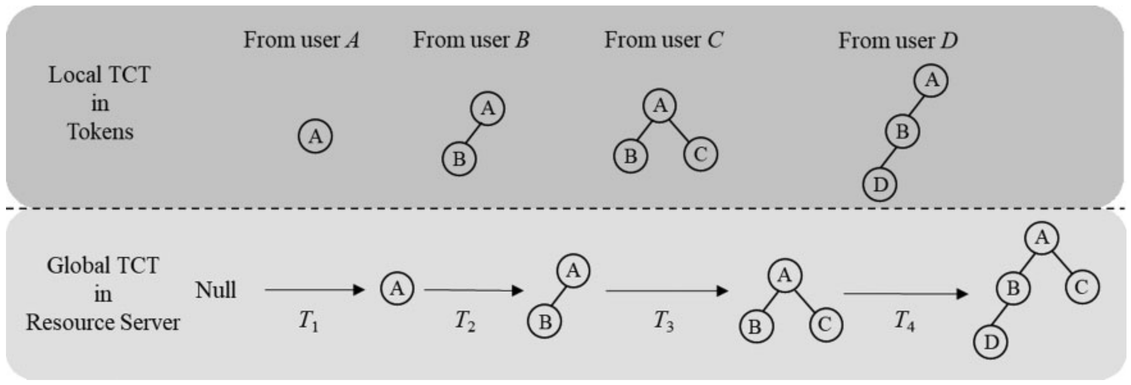

Figure 3: Combination of global time capability trees

3.2 TCT Capability Token

We define a new capability token for TCAC. When *user A* delegates capability to *user B*, the token $Cap_{A\to B}$ is defined as:

$$Cap_{A\to B} = \{ID_A, ID_B, PK_{pre}, Cap_{pre}, TCT_B, C, Signature_A\} \tag{2}$$

$$Signature_A = f(ID_A, ID_B, PK_{pre}, Cap_{pre}, TCT_B, C) \tag{3}$$

- ID_A: The unique identity of *user A*;
- ID_B: The unique identity of *user B*;
- PK_{pre}: The public key of the previous user which delegates the token to *user A*;
- Cap_{pre}: The Capability issued by the previous user which delegates the token to *user A*;
- TCT_B: The time capability tree constructed by *user A* including *node B*;
- C: The contextual information;
- $Signature_A$: The signature of *user A*;
- f: The one-way hash function;

In the TCAC token, ID_A and ID_B are used to identify the virtual/real identity of delegator and delegatee. PK_{pre} and Cap_{pre} provide information about the previous delegator of the token, which is used by the *Resource Server* in token verification. C allows users to add contextual information during delegation, such as the delegation time and user location [26]. $Signature_A$ is used to protect the token from being tampered. After the delegator verifies the delegatee identity and the public key, capability token will be signed and delegated.

4 Detailed System Description

The Fig. 4 shows the flowchart of authorization and request service in TCAC. The delegator receives an authorization request from the delegatee, which includes his ID and public key. The communication legitimacy is guaranteed by establishing a trusted channel. After legitimacy verification, the delegator generates the token, signs it and delegates it with the public key to delegatee. Delegatee uses the public key to verify the Signature, then obtains the required permission. We present the detail description of token verification, delegation, revocation and modification of TCAC in this section.

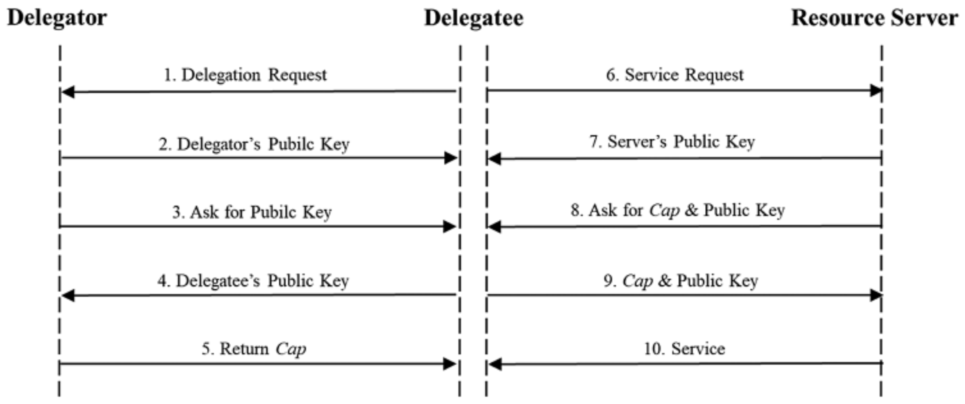

Figure 4: Flowchart of authorization and request service in TCAC

4.1 Token Verification

The token security of TCAC is guaranteed by the asymmetric encryption. Assume that Alice is the delegator, and Bob is the delegatee. The encryption is expressed as:

$$Token = (Cap_{A\to B}, PK_b) \tag{4}$$

where *Token* is the encrypted *token*, $Cap_{A\to B}$ is the content of the *token*, and PK_b is the public key of Bob. $Signature_A$ ensures that the token comes from Alice, and the content is not tampered. The asymmetric encryption enables only the delegatee knows the content of token. The two encryption methods work together to ensure the integrity and confidentiality of $Cap_{A\to B}$.

The token verification in TCAC supports common verification and quick verification.

(i) *Common Verification*

When *token* is successfully decrypted and the *Signature* is verified, the obtained capability *token* is considered to be valid. If a new user sends a capability request to the *Resource Server* for the first time, it needs to perform the iterative decryption on the *token* to verify the legitimacy. The decryption is expressed as:

$$Cap = (Token, SK) \tag{5}$$

where *SK* is the private key of the *Resource Server*. After *Cap* is decrypted, the obtained Cap_{pre} and PK_{pre} are used to continue the decryption. The process will be iterated until the root capability is verified, then the *token* is confirmed valid.

(ii) *Quick Verification*

When the user sends a capability request to the *Resource Server* again, the server only needs to obtain the *User ID* and the requested capability in the token, then search in the global time capability tree. If the *User ID* exists in the global time capability tree, and the request time is within the *Effective Time*, the request will be allowed. Then the access time will be recorded in the global time capability tree.

Algorithm 1: The common verfication and quick verification of TCAC

Input: $User_i$, $token_i$, TCT_{global}, *Effective Time*, *Expiration Time*

(Continued)

Algorithm 1: Continued

```
Output: $AlloworDenyaccess$
begin
verify the identity of the $User_i$;
decrpty the $token_i$;
if $User_i\ Node \in TCT_{global}$ & $User_i\ Access\ Time \in (Effective\ Time, Expiration\ Time)$ then
        $AlloworDenyaccess = Allow$;
else if $token_i$ is valid then
        $AlloworDenyaccess = Allow$;
        $TCT_{global} = TCT_{global} + User_i\ Node$;
else
        $AlloworDenyaccess = Deny$;
end if
end
```

The Algorithm 1 implements the common verfication and quick verification of TCAC, which solves the iterative decryption problem. It simplifies the token verification and reduces the need in special IoT devices, which makes TCAC more suitable for resource-constrained IoT environment [27].

4.2 Token Delegation

We present the time capability trees change in the TCAC's token delegation of three scenarios: all capability, partial capability, and constrained capability. All capability means the delegatation of all the delegator's permission set. Partial capability means delegating part of the delegator's permission set. Constrained capability means delegating a new constrained capability which is defined by the delegator. We use the first letter of each user's name to replace the node in the time capability tree, and set the *Effective Time* to the maximum $(t_i \sim t_n)$.

The delegation in TCAC is shown in Fig. 5. Alice is the root user with capabilities *read* and *write* to the resource *File*. We use **R** and **W** to represent the two capabilities.

1) At time t_0, after Alice gets the root token Ω_0, she obtains the capability to *read* and *write File*.
2) At time t_1, Alice generates a new token Ω_1, and delegates **R** and **W** to Bob by delegating them the token Ω_1. The time capability tree in Ω_1 is generated by Alice, which is the combination of the time capability tree known by Alice and the new node Bob.
3) At time t_2, Alice delegates **R** to Candy. All operations are the same, but the time capability tree has changed. The node Candy are added to the time capability tree, and only the **R** capability tree is saved in Ω_2. This helps to protect privacy while reducing the size of *token*.
4) At time t_3, Bob delegates **R** to David. Bob only adds a new node David on the **R** capability tree he has obtained, since the capability delegation of Alice is transparent to Bob.
5) At time t_4, Bob delegates the constrained capability of **W** to Edward, which is recorded as $\mathbf{W}^-$. $\mathbf{W}^-$ is the capability to rewrite part of *File*. $\mathbf{W}^-$ needs to be defined in the *Resource Server*, otherwise it will not be recognized. Bob adds Edward as a new node based on the **W** capability tree he has obtained, constructs the $\mathbf{W}^-$ capability tree, and adds it to the token Ω_4.

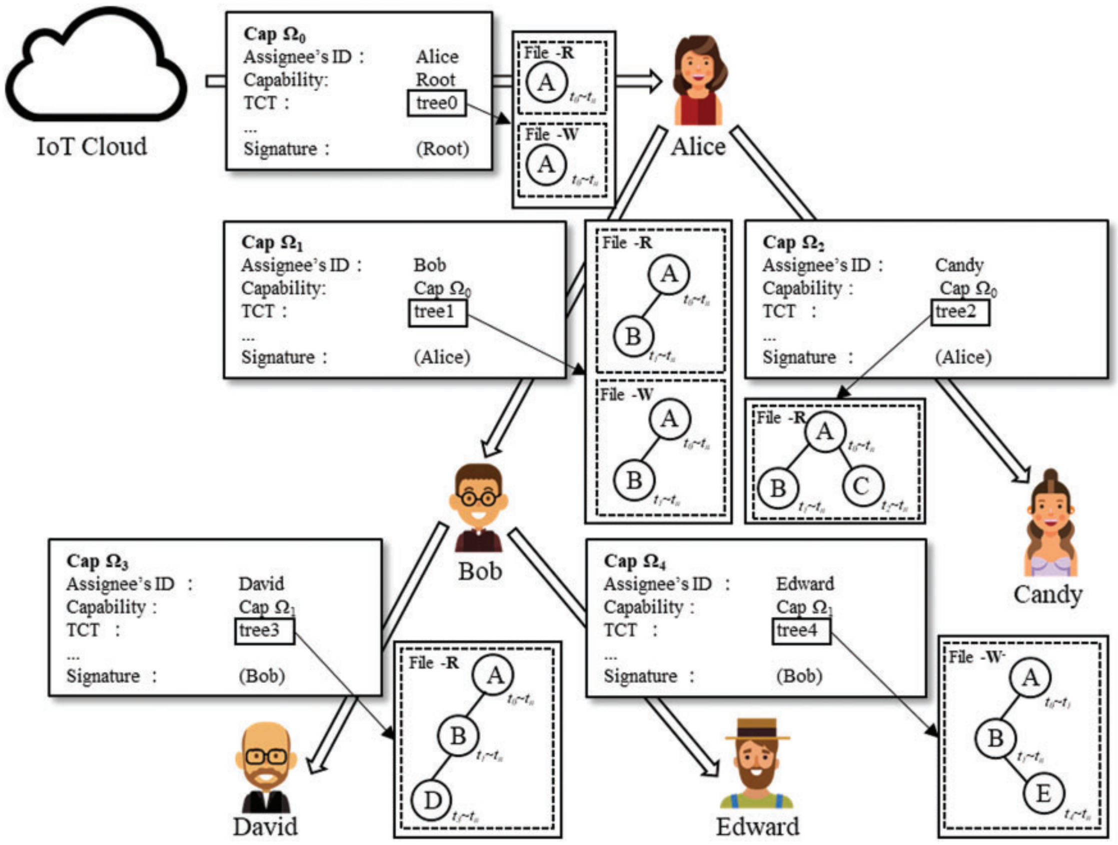

Figure 5: TCAC's capability delegation

Algorithm 2: The permissions delegation trajectory capture algorithm

```
Input: $User\ Group$, $User_i$, $token_i$, $TCT_{all}$, $TCT_{partial}$, $TCT_{constrained}$
Output: $Allowor Denyaccess$, $TCT_{global}$
begin
{the trajectory growth of a local $TCT$ in delegation}
{$Delegator$ constructs the new $TCT$ to delegate permission}
for each $User_i \in User\ Group$ do
    verify the identity of the $User_i$;
    decrpty the $token_i$;
    if the delegated capability is the all capability
        if $User_i\ Node \notin TCT_{all}$ & $token_i$ is valid then
            $TCT_{new} = TCT_{all} + User_i\ Node$;
            $AlloworDenyaccess = Allow$;
        end if
    else if the delegated capability is the partial capability
        if $User_i\ Node \notin TCT_{partial}$ & $token_i$ is valid then
            $TCT_{new} = TCT_{partial} + User_i\ Node$;
```

(Continued)

Algorithm 2: Continued

```
                $AlloworDenyaccess = Allow$;
            **end if**
        **else if** the delegated capability is the constrained capability
            **if** $User_i\ Node \notin TCT_{constrained}$ & $token_i$ is valid **then**
                $TCT_{new} = TCT_{constrained} + User_i\ Node$;
                $AlloworDenyaccess = Allow$;
            **end if**
        **else**
            $AlloworDenyaccess = Deny$;
        **end if**
        $TCT_{local} = TCT_{new}$
**end for**
$TCT_{global} = \sum TCT_{local}$;
**end**
```

The Algorithm 2 implements the permissions delegation trajectory capture algorithm of TCAC. In an IoT delegation system, the capability delegation and the service request are performed simultaneously. The *Resource Server* updates the global time capability tree according to the newly received *token*, then obtain a global delegation trajectory.

4.3 Token Revocation and Modification

In CapBAC, some operations can't be implemented, like to make new user replace an old one while ensuring the delegation relationships not changed. The time capability tree in TCAC provides fine-grained token revocation and modification.

4.3.1 Capability Revocation

Benefit from the global delegation trajectory, TCAC brings new possibilities to deal with more complex revocation requirements. Every access request in TCAC will add an access time to the *Access Records* in global time capability tree. In the token verification, it needs to be checked whether the access time is within the *Effective Time*. If not, the access is considered illegal and will not be authorized. If we want to revoke the capability that has been captured, we change the *Effective Time* in the global capability tree from [start time, end time] to [start time, current time], and delete all child nodes. After that, all the issued tokens will be considered as invalid tokens, because the *Effective Times* in their local time capability trees do not match the record in the global time capability tree. This change invalidates the capability of the user and all the related users. If we want to replace a user in the delegation relationships, we can change the global tree, and issue a new token to all the related users. Similar methods provide a great deal of flexibility for TCAC capability revocation.

4.3.2 Capability Modification

Compared with revocation, capability modification needs more flexibility of the access control system. Modifications are usually caused by caused improper capability definitions and improper delegations. In CapBAC, the improper capability definitions requires to correct of all the issued capability tokens, even though it is a minor change. But in TCAC, we only modify the semantics of the time capability tree *title* on the *Resource Server* to modify similar capabilities. This change not only avoids reissuing all the tokens, but also preserves the capability delegation relationships. The improper delegation requires to change issued capabilities. In TCAC, we can revoke the capability, then reissue the token when the user requests it again.

Algorithm 3: The token verification after permissions revocation and modification in TCAC

```
Input: $User_i$, $token_i$, $TCT_{global}$, $Effective\ Time$, $Expiration\ Time$
Output: $AlloworDenyaccess$, $TCT_{global}$
begin
{the change in a global $TCT$ of revocation and modification}
{after a permission is revoked or modified in the delegation system}
if $User_i\ Node \in TCT_{global}$ & $User_i\ Access\ Time \in (Effective\ Time, Expiration\ Time)$ then
        $AlloworDenyaccess = Allow$;
else if $User_i\ Node \in TCT_{global}$ & $User_i\ Access\ Time \notin (Effective\ Time, Expiration\ Time)$ then
        $AlloworDenyaccess = Deny$;
        $TCT_{global} = TCT_{global} - User_i\ Node$;
else if $token_i$ is valid then
        $AlloworDenyaccess = Allow$;
        $TCT_{global} = TCT_{global} + User_i\ Node$;
else
        $AlloworDenyaccess = Deny$;
end if
end
```

The Algorithm 3 presents the token verification after token revocation and modification in TCAC. It is worth notice that the revocation and modification methods in TCAC are not unique. Different methods can be customized for a delegation system by changing the token verification algorithm.

5 Experimental Evaluation

We evaluate the token verification time, communication time, revocation/modification time and delegation trajectory capture efficiency to verify the effectiveness of TCAC.

5.1 Experimental Implementation

The root token issuer and the *Resource Server* are deployed on a laptop with the following configuration: the CPU is 1.6 GHz Intel Core i5 (4 cores), the RAM memory is 8GB, and the operating system is Ubuntu 16.04. 10 Raspberry PI 4 Model B devices are used to represent 10 users. The configuration of the deivces is as follows: the CPU is 1.5 GHz 64-bit 4-core ARM Cortex-A72 architecture, the RAM is 4GB LPDDR4-3200, the operating system is Raspbian. The 2.4 GHz wireless hotspot is used for communication.

We generate delegation trajectories from the randomly selected devices, to represent the local time capability trees of users. Then we combain them to obtain global time capability trees with delegation depths from 1 to 10. We finally get 1000 global time capability trees, which consist of 10,000 users. An attribute-based access control is deployed in the root token issuer, and the tokens are delegated randomly between users. We use RSA in our experiments, which has been proved faster in token encryption and signature [28].

5.2 Token Verification Time

The token we use in token verification has only one authorized user every layer (for example, the 4-layer token contains only 4 authorized users). The CCapBAC represents CapBAC that uses a token cache which stores the visited capability tokens. Tab. 1 shows the results.

In Tab. 1, the verification time of the three methods are positively correlated with the delegation depth. CapBAC's verification time increases fast with the increase of the delegation depth. CCapBAC and TCAC keep a small growth rate in token verification, while CCapBAC is faster than TCAC before the delegation depth is less than 5. TCAC shows advantage in token verification after the delegation depth is more than 4, because it has a smaller growth rate than those of the other two methods. When the delegation depth is 7, the verification time of CapBAC exceeds 2000 ms, but TCAC still maintains a 43.8 ms verification time. It is caused by the iterative RSA decryption and signature verification in CapBAC. CCapBAC and TCAC only verify the signature and search in the token cache or the global time capability tree, so their verification efficiency is higher. But CCapBAC uses the nested tokens as the same of CapBAC. The increasing size of token makes the search speed increase faster than that of TCAC.

Table 1: The time of token verification with different delegation depths

Delegation depth		1	2	3	4	5	6	7
Verification time (ms)	CapBAC	41.176	98.917	195.681	363.037	659.619	1162.06	2071.46
	CCapBAC	31.221	33.124	33.893	40.312	44.267	50.318	55.571
	TCAC	40.253	41.372	41.981	42.562	43.021	43.398	43.820

5.3 Communication and Token Revocation/Modification Time

We implement experiments in communication and token revocation/modification time of CapBAC, TCAC and xDBAuth [29]. xDBAuth is another way to use capability-based method. It combines capability token with the blockchain to achieve cross-domain access control. The results are shown in Fig. 6.

In Fig. 6a, the communication time of CapBAC and TCAC are almost the same, and they increase with the increase of the token size. It shows that the communication time of TCAC is acceptable for the actual projects. The xDBAuth communication time is only related to the smart contract operations, so it is stable. Fig. 6b shows that the revocation/modification time of xDBAuth and TCAC are almost constant with the delegation depth increasing. xDBAuth takes more time in token revocation/modification, because the operations in the smart contract is more complex. CapBAC needs to constuct a new capability token, so the revocation/modification time in CapBAC is increasing with the delegation depth. The token revocation and modification in TCAC are implemented by modifying the global capability tree, so the revocation/modification time is always the smallest than those of CapBAC and xDBAuth.

5.4 Delegation Trajectory Capture Efficiency

The delegation trajectory capture efficiency of the 1000 global delegation trajectoris are recorded. The calculation is as follows:

$$\overline{P}_{\text{Capture}} = \frac{N_{\text{Captured}}}{N_{Total}} \tag{6}$$

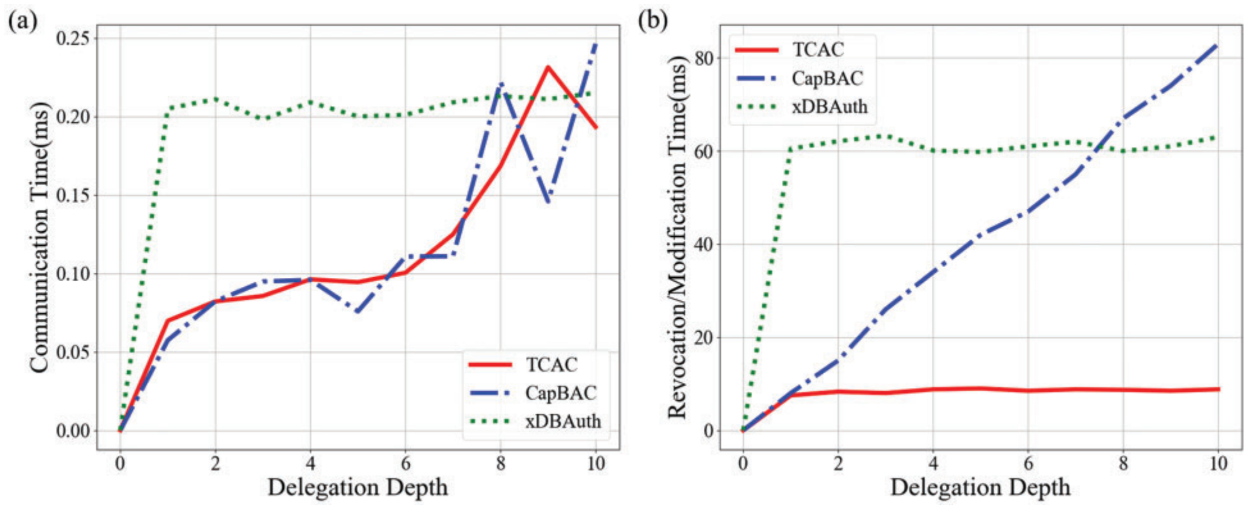

Figure 6: (a) Communication time and (b) revocation/modification time in different delegation depths

$$\overline{P}_{NV\text{Captured}} = \frac{N_{NV\text{Captured}}}{N_{NV}} = \frac{N_{\text{Captured}} - N_{V\text{Captured}}}{(1 - P_V) \cdot N_{Total}} \tag{7}$$

where V represents the visited users, NV represents the unvisited users, *Total* represents the total users, and *Captured* represents the captured users. We set the access probability for all the users. The average capture efficiency and the unvisited users capture efficiency in different access probabilities are shown in Fig. 7.

In Fig. 7a, the average capture efficiency of CapBAC is consistent with the users access probability, and the capture efficiency of TCAC is always higher than that of CapBAC. It shows that CapBAC only captures the visited users, and TCAC can discover the unvisited users. In Fig. 7b, the unvisited users capture efficiency of CapBAC is always 0, and TCAC obtains a increasing unvisited users capture efficiency with the increase of access probability. When there are 1000 unvisited users in the 10,000 users, TCAC captures 733 of them. It shows that benefit from the time capability tree, TCAC has a strong ability of capturing unvisited users in a delegation system.

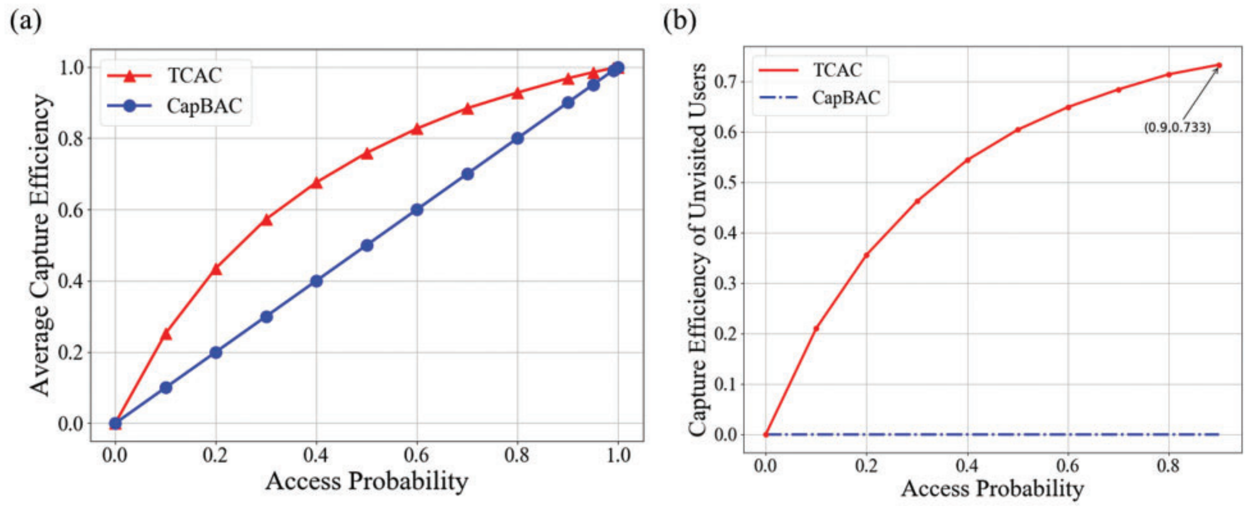

Figure 7: (a) The average capture efficiency and (b) the unvisited users capture efficiency in different access probabilities

6 Conclusion

In this paper, we propose a traceable capability-based access control approach TCAC. We design a time capability tree to extract capability delegation trajectories, which can monitor the permissions delegation automatically. We also present methods how to execute the capability verification, delegation, revocation and modification in TCAC. The experimental results show that TCAC has advantages of being flexible in token verification, revocation and modification. Furthermore, the delegation trajectory capture efficiency of TCAC is higher than that of CapBAC which can not capture the unvisited users. When 10% of users do not access the resource, TCAC discovers 73.3% of those unvisited users. TCAC's ability to capture unvisited users provides more information about the permissions delegation relationships. It makes the global delegation trajectory more complete, which brings new possibilities in global delegation analysis and enhances the security of the IoT delegation system. In conclusion, TCAC presents higher effectiveness in token verification, token revocation/modification, and delegation trajectory capture, with the acceptable communication time.

Funding Statement: This work supports in part by National Key R&D Program of China (No. 2018YFB2100400), National Science Foundation of China (No. 61872100), Industrial Internet Innovation and Development Project of China (2019), State Grid Corporation of China Co., Ltd. technology project (No. 5700-202019187A-0-0-00).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng *et al.,* "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 14, pp. 3690–3700, 2018.

[2] J. Park and S. Kim, “Noise cancellation based on voice activity detection using spectral variation for speech recognition in smart home devices,” *Intelligent Automation & Soft Computing*, vol. 26, no. 1, pp. 149–159, 2020.

[3] V. S. Naresh, S. S. Pericherla, P. Sita and S. Reddi, “Internet of things in healthcare: Architecture, applications, challenges, and solutions,” *Computer Systems Science and Engineering*, vol. 35, no. 6, pp. 411–421, 2020.

[4] D. Kim and S. Kim, “Network-aided intelligent traffic steering in 5g mobile networks,” *Computers, Materials & Continua*, vol. 65, no. 1, pp. 243–261, 2020.

[5] S. Rajendran and R. M. Lourde, “Security threats of embedded systems in IoT environment,” in *Inventive Communication and Computational Technologies*, Tamil Nadu, India, Springer, Singapore, pp. 745–754, 2019.

[6] S. T. Bakhsh, S. Alghamdi, R. A. Alsemmeari and S. R. Hassan, “An adaptive intrusion detection and prevention system for internet of things,” *International Journal of Distributed Sensor Networks*, vol. 15, no. 11, pp. 1550147719888109, 2019.

[7] B. Che, L. Liu and H. Zhang, “KNEMAG: Key node estimation mechanism based on attack graph for IOT security,” *Journal of Internet of Things*, vol. 2, no. 4, pp. 145–162, 2020.

[8] OWASP, “OWASP internet of things project,” 2018. [Online]. Available: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10.

[9] S. Ravidas, A. Lekidis, F. Paci and N. Zannone, “Access control in internet-of-things: A survey,” *Journal of Network and Computer Applications*, vol. 144, pp. 79–101, 2019.

[10] SmartTings, “One simple ho me system. A world of possibilities,” 2021. [Online]. Available: https://www.smartthings.com/.

[11] IFTTT, “IFTTT helps every thing work better together,” 2021. [Online]. Available: https://ifttt.com/.

[12] Google, “Google developers,” 2021. [Online]. Available: https://developers.google.com/.

[13] S. Gusmeroli, S. Piccione and D. Rotondi, “A Capability-based security approach to manage access control in the internet of things,” *Mathematical and Computer Modelling*, vol. 58, no. 5–6, pp. 1189–1205, 2013.

[14] S. Gusmeroli, S. Piccione and D. Rotondi, “IoT@ work automation middleware system design and architecture,” in *Proc. of 2012 IEEE 17th Int. Conf. on Emerging Technologies & Factory Automation (ETFA 2012)*, Krakow, Poland, pp. 1–8, 2012.

[15] W. W. Smari, P. Clemente and J. F. Lalande, “An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system,” *Future Generation Computer Systems*, vol. 31, pp. 147–168, 2014.

[16] B. Anggorojati, P. N. Mahalle, N. R. Prasad and R. Prasad, “Capability-based access control delegation model on the federated IoT network,” in *The 15th Int. Symp. on Wireless Personal Multimedia Communications*, Taipei, Taiwan, pp. 604–608, 2012.

[17] J. L. Hernández-Ramos, A. J. Jara, L. Marin and A. Skarmeta, “Distributed capability-based access control for the internet of things,” *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 3/4, pp. 1–16, 2013.

[18] R. Xu, Y. Chen, E. Blasch and G. Chen, “A federated capability-based access control mechanism for internet of things (IoTs),” in *Sensors and Systems for Space Applications XI. International Society for Optics and Photonics*, Orlando, Florida, United States, SPIE, vol. 10641, pp. 106410U, 2018.

[19] R. Xu, Y. Chen, E. Blasch and G. Chen, “Blendcac: A blockchain-enabled decentralized capability-based access control for iots,” in *2018 IEEE Int. Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, pp. 1027–1034, 2018.

[20] S. Pal, M. Hitchens, V. Varadharajan and T. Rabehaja, “Policy-based access control for constrained healthcare resources in the context of the internet of things,” *Journal of Network and Computer Applications*, vol. 139, pp. 57–74, 2019.

[21] T. Rabehaja, S. Pal and M. Hitchens, “Design and implementation of a secure and flexible access-right delegation for resource constrained environments,” *Future Generation Computer Systems*, vol. 99, pp. 593–608, 2019.
[22] J. Shen, T. Zhou, X. Chen, J. Li and S. Willy, “Anonymous and traceable group data sharing in cloud computing,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 912–925, 2017.
[23] L. Guo, X. Yang and W. C. Yau, “TABE-DAC: Efficient traceable attribute-based encryption scheme with dynamic access control based on blockchain,” *IEEE Access*, vol. 9, pp. 8479–8490, 2021.
[24] Q. Li, B. Xia, H. Huang, Y. Zhang and T. Zhang, “TRAC: Traceable and revocable access control scheme for mHealth in 5G-enabled IIoT,” *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 3437–3448, 2021.
[25] K. P. Yu, L. Tan, M. Aloqaily, H. Yang and Y. Jararweh, “Blockchain-enhanced data sharing with traceable and direct revocation in IIoT,” *IEEE Transactions on Industrial Informatics*, vol. 17, pp. 7669–7678, 2021.
[26] Z. Li, J. Zhang, K. Zhang and Z. Li, “Visual tracking with weighted adaptive local sparse appearance model via spatio-temporal context learning,” *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4478–4489, 2018.
[27] A. Bader, H. ElSawy, M. Gharbieh, M. -S. Alouini, A. Adinoyi *et al.,* “First mile challenges for large-scale IoT,” *IEEE Communications Magazine*, vol. 55, no. 3, pp. 138–144, 2017.
[28] Q. Zhou, M. Elbadry, F. Ye and Y. Yang, “Heracles: Scalable, fine-grained access control for internet-of-things in enterprise environments,” in *IEEE INFOCOM 2018-IEEE Conf. on Computer Communications*, Honolulu, HI, USA, pp. 1772–1780, 2018.
[29] G. Ali, N. Ahmad, Y. Cao Y. S. Khan, H. Cruickshank *et al.,* “xDBAuth: Blockchain based cross domain authentication and authorization framework for internet of things,” *IEEE Access*, vol. 8, pp. 58800–58816, 2020.