Hindawi
Scientific Programming
Volume 2020, Article ID 8832568, 13 pages
https://doi.org/10.1155/2020/8832568

Research Article

Trusted and Efficient Cross-Domain Access Control System Based on Blockchain

Shuang Sun,[1,2] Shudong Chen ,[1,2] and Rong Du[1,2]

[1]*Institute of Microelectronics of the Chinese Academy of Sciences, Beijing 100029, China*
[2]*School of Microelectronics, University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Shudong Chen; chenshudong@ime.ac.cn

Received 28 April 2020; Revised 24 June 2020; Accepted 17 July 2020; Published 10 August 2020

Academic Editor: Cristian Mateos

In a distributed system, cross-domain access control is an important mechanism to realize secure data sharing among multiple domains. Most of the existing cross-domain access control mechanisms are generally based on a single-server architecture, which has limitations in terms of security and reliability (the access decision may be incorrect) and completeness and confidentiality (the access records can be modified). Blockchain technology with decentralization, verifiability, and immutability properties can solve these problems. Motivated by these facts, in this article, we construct a trusted and efficient cross-domain access control system based on blockchain. Consequently, we integrate blockchain and role mapping technology to provide reliable and verifiable cross-domain access process. We use blockchain to record user roles, role mapping rules, access policies, and audit records, realizing user self-validation, and access nonreputation. Considering the low throughput of the blockchain, we design an efficient smart contract to make the access decision based on the access history of users. Finally, a performance evaluation of the system is presented to demonstrate the feasibility of the proposed system.

1. Introduction

With the development of cloud computing and the Internet of Things (IoT) technology, distributed systems and distributed storage have been widely used. In the distributed environment, devices and resources are scattered in different domains. There is an urgent need for resource sharing and device interoperability between domains [1–6]. For example, by opening government data, the government can help companies accurately predict corporate financial risks and avoid financial risks. Companies are promoting the development of local tourism by sharing traffic and travel data.

However, the security issues of the distributed environment have become increasingly prominent. According to the IoT security white paper 2018 [7], in March 2018, the Under Armour company exposed personal information for 150 million MyFitnessPal users in a data breach. Similarly, the high-risk vulnerability, identified as CVE-8-0171, allows an attacker bypassing authentication and execute any code to control the device completely, resulting in the configuration files being emptied. In April 2018, it was found that the WIFIKey-universal was involved in stealing personal information, invading others' WIFI networks, and sharing WIFI password.

Access control is a security mechanism that restricts access to critical or valuable resources such as data, services, computational systems, and storage space. Thus, a fine-grained, dynamic, and secure access control [8] isolates devices and resources between domains by restricting permissions for accessing subjects via access policies [9] and could provide security and privacy protection for a distributed system. However, domains have different security goals that result in heterogeneous access control mechanisms and access policies [10, 11]. Therefore, a number of cross-domain access control mechanisms have been proposed to realize secure resource sharing between heterogeneous domains [12–16], but almost all of them are based on a single-server architecture. Existing cross-domain access control mechanisms based on single-server architectures generally have the following limitations:

(1) Lack of security and reliability: a single-server cannot provide a trusted access decision, for a malicious or compromised server can easily prevent granted users from accessing resources or permit illegal users to access resources.

(2) Lack of completeness and confidentiality: users' historical access records stored by the server may be modified or deleted without the server's knowledge.

To solve these limitations, the role of blockchain technology has been highlighted in multiple-domain access control scenarios due to its properties such as decentralization, immutability, and verifiability [17]. These properties are beneficial in constructing a more secure, verifiable, and reliable access control systems.

To provide reliable and verifiable cross-domain access control processes, we integrate blockchain and role mapping access control. In our system, the user roles, role mapping rules, and access policies are publicly visible on the blockchain, allowing any users to verify whether their role satisfied the access policy at any time, which prevents a single-server from fraudulently denying the granted users or permitting the illegal users to access the resources. Considering the low throughput of the blockchain, we design an efficient smart contract which makes access decisions directly based on the access history. All requests and decisions will be recorded on the blockchain. Due to the immutability of blockchain, these requests and decisions are not easy to be tampered or deleted, providing reliable auditing.

2. Related Works

Due to the extensive use of the role-based access control (RBAC) mechanism [18–22], role mapping cross-domain access control is a typical method to realize interoperation between domains [23, 24].

With regard to secure interoperability in a multidomain environment, numerous literature studies have been proposed. Denker et al. [25] proposed a cross-domain access control mechanism based on PKI and RBAC. According to Grit, the PKI requires each domain has its own CA center to integrate the access control policies. Kapadia et al. [26] proposed an interdomain security interoperability model based on role mapping, which can create a flexible strategy for dynamic role mapping between the local domain and the external domain.

Upon the benefits of distributed access control, the authors in [27–33] proposed blockchain-based access control schemes for the IoT environment. Sukhodolskiy and Zapechnikov [34] proposed a blockchain-based access control system for cloud storage, which provides access control over the data stored in the cloud. Zhou et al. [35] proposed a blockchain-based access control scheme for smart grids. Wang et al. [36] design a blockchain-based framework for data sharing with fine-grained access control. Guo et al. [37] proposed a hybrid architecture to facilitate access control of EHR data by using both blockchain and edge node. Steichen et al. [38] use blockchain to realize decentralized access control for IPFS. Although these blockchain-based access control systems provide decentralized, reliable, and secure access control methods, they cannot be applied in a multiple-domain system with heterogeneous security and privacy requirements.

3. Preliminaries

In this section, we review some relevant background knowledge that will be used in this article.

3.1. Role-Based Access Control. As shown in Figure 1, the role-based access control (RBAC) model contains three basic elements: Users, Roles, and Permissions (PERM). It grants permissions to roles rather than users; thus, users obtain operation permissions via roles assignment [39, 40]. Systems create appropriate roles according to the system's tasks. After roles are created, systems grant their permission. Finally, users are assigned to roles based on the system's security requirement. Users and permissions are linked by roles; therefore, with the roles, users can operate (access or control) the objects (devices or resources).

3.2. Blockchain. Blockchain is more used as the underlying technology to provide decentralized, distributed anti-destruction, traceability, and tamper-resistant services, than a decentralized electronic currency trading system. Each node in the distributed network maintains the same ledger, so it does not need a third party to provide data service [41]. When a transaction is synchronized to the blockchain, it cannot be modified or deleted, so all transactions are traceable on the blockchain ledger.

Common blockchain platforms include Bitcoin [42], Ethereum [43], EOS [44], and Hyperledger Fabric [45], which can be divided into five layers: network layer, consensus layer, data layer, smart contract layer, and application layer. The structure of these blockchain platforms are shown in Table 1.

3.3. Cross-Domain Access Control Based on Role Mapping. Role mapping technology realizes cross-domain access by defining rules for mapping roles between two domains. As shown in Figure 2, role mapping rules can be expressed by a tuple $<D_1, r_1, D_2, r_2>$, where r_1 is the role in the domain D_1 and r_2 is the mapped role of r_1 in domain D_2. The user with the role r_1 in D_1 has the corresponding permissions of the role r_2 in domain D_2.

3.4. Identity-Based Signature. In our system, we use the Identity-based signature (IBS) to provide the identification of entities. IBS allows entity to generate a public key from a known identity (ID) such as email address, domain name, or a physical IP address. A trusted third party, called the Private Key Generator (PKG), generates the corresponding private keys for the ID. Kiltz et al [46] defined a set of four algorithms that form a complete IBS scheme:

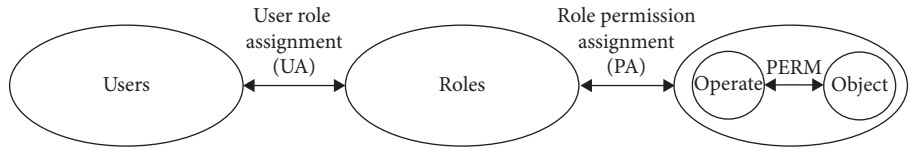

Figure 1: The RBAC model.

Table 1: Blockchain platform.

	Bitcoin	Ethereum	EOS	Hyperledger fabric
Application	Bitcoin	Dapp	EOS/Dapp	Hyper ledger
Smart contract	Script	Solidity	C++	Go/java
Data	File system	LevelDB	IPFS	File system/level DB
Consensus	POW	POW/POS	BFT-DPOS	PBFT/SBFT
Network	TCP-based P2P	TCP-based P2P	TCP-based P2P	HTTP/2-based P2P

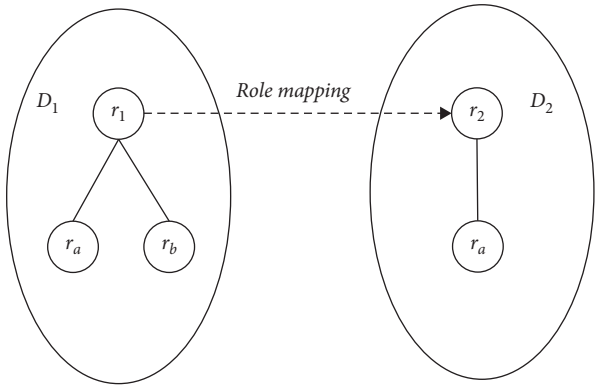

Figure 2: Role mapping.

Setup: this algorithm is run by the PKG one time for creating the whole IBS environment, which includes a master key pair (mpk, msk).

KeyDer: on input msk and identity (ID) of the entity, this algorithm is run by the PKG to generate the secret signing key Isk for the ID.

Sign: on input usk and a message M, this algorithm returns a signature σ of M.

Vf: on input mpk, ID, M, and signature σ, this algorithm returns 1 if σ is valid for ID and M, and returns 0 otherwise.

4. An Overview of the Proposed System

4.1. System Model. As shown in Figure 3, the system model mainly consists of the following three parts: domain organizations (DO), domain management servers (DMS), and the blockchain (BC).

Domain organization: DOs include users, devices, and a DMS. Users are cross-domain requesters. Devices are resource owners and can define access control policies to decide who can access their resources.

Domain management server: DMSs are responsible for formulating role mapping rules between DOs. They upload the role mapping rules, users' roles, and devices' access policies onto the blockchain. DMS is also responsible for providing cross-domain access services for users. In addition, all DMSs jointly maintain the blockchain.

Blockchain: the BC is used to supervise the DMSs. It records user roles, role mapping rules, access control policies, and the audit records which contain requests and the access decisions. To prevent malicious attackers, it uses a consensus algorithm to ensure the blockchain ledger is consistent.

Private key generator: PKG generates the secret signing key Isk for the entity (DMS, user, and device) with ID.

4.2. Secure Assumption. To ensure the security of our system, we make the following security assumptions.

Assumption 1. In our system, we assume that the DMSs cannot forge roles for users' illegal access.

Assumption 2. In our system, as a blockchain maintainer, the DMSs may become byzantine nodes because of various malicious attacks. Therefore, we assume that the number of byzantine nodes is no more than n out of $3n+1$ CBMs.

Assumption 3. In our system, we assumed that PKG can verify the authenticity of the entity's identity.

4.3. Security Goal

DMS attacks resistant: to ensure that the granted users can access resources, the system should be able to realize user self-validation.

Modification attacks resistant: the system should able to resist the attacker to modify the broadcasted request, request transactions, and replied data. In addition, the system should be able to resist the attacker to modify or delete the audit records.

Man-in-the-Middle attacks resistant: no one but the requester can read the returned data.

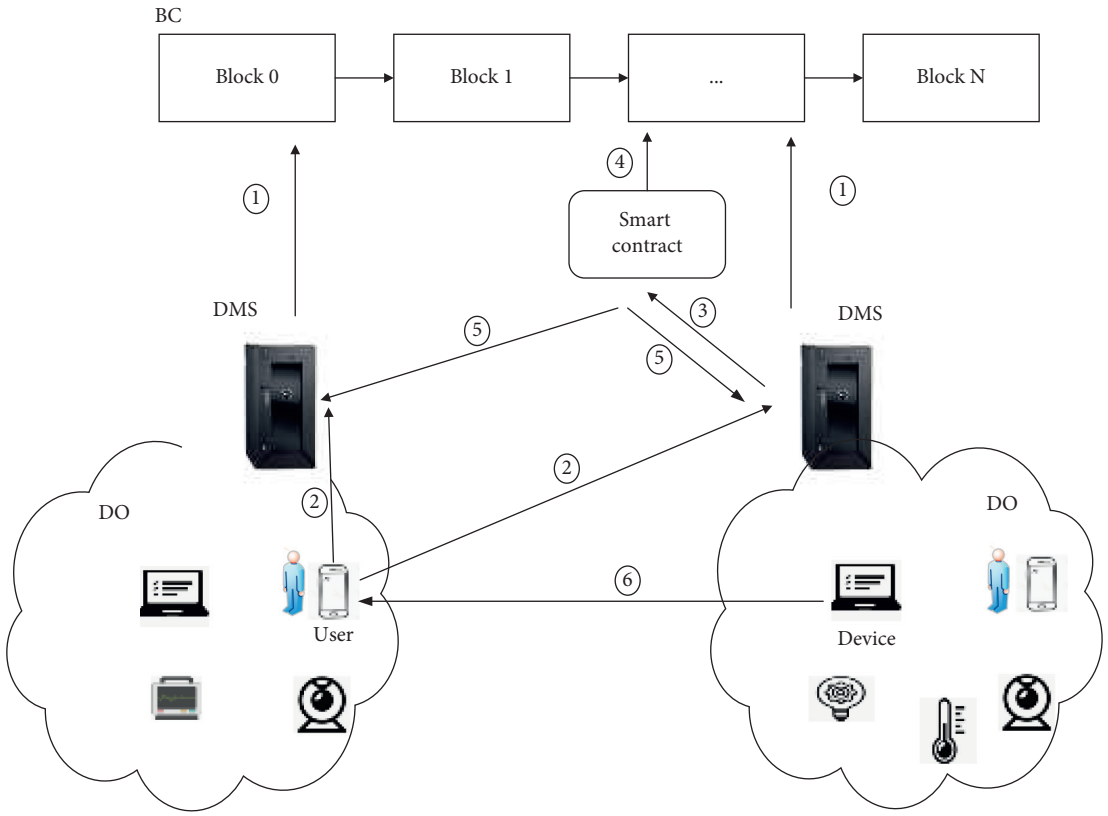

Figure 3: An overview of the proposed system. Step 1: DMSs upload the user roles, role map rules, and policies of devices on the blockchain. Step 2: users broadcast their cross-domain access requests. Steps 3 and 4: the blockchain records the audit records. Step 5: smart contract makes and returns the access decision. Step 6: devices return the resources to users.

5. System Design

This section presents the working of our system which is realized upon existing technologies, namely, blockchain and role mapping. Before presenting the system design, here, we first introduce the transaction types and the smart contract design in our system. We design the identity (ID) of the DO entity (DMS, user, and device) which contains the name of the entity and the DO information. Thus, anyone can determine the DO where the entity is located by its ID and then find the DMS's ID of the DO. For example, assuming DO_A is a DO, we design the DO user's ID as userName@DO_A, device's ID as devName@DO_A, and DMS's ID as DMS@DO_A. Therefore, anyone can determine an entity with entity@DO_A is in DO_A and find the DMS's ID (DMS@DO_A) of the DO_A.

5.1. Transaction. In a blockchain system, a transaction consists of two parts: the transaction header and the payload [47]. In our system, a request transaction consists of a tag, user's ID (uID), device's ID (devID), access control order, and signature, that is, $TX_{request}$ = tag || uID || devID || access_control_order || signature. Assuming that a user, Bob, wishes to read the camera's data in Alice's room, he needs to broadcast the request = 01 $||ID_{Bob}||$ ID_{ali_cam} $||r$ $||sig_{Isk_{Bob}}(01\|ID_{Bob}\|ID_{ali\_cam}\|r)$ to DMSs, where 01 means Bob has read the data of this camera before, ID_{Bob} is Bob's ID, ID_{ali_cam} is the ID of the Alice's camera, r is the access control order of "read", and sig denotes the signature activities using a digital signature scheme.

5.2. Smart Contract. The smart contract defines some executable logic. It will be installed in the blockchain, and it defined functions which provide interfaces invoked by a transaction.

In our system, in order to provide a trusted access control process, each DMS uses smart contract to upload their users' roles, role mapping rules between DOs, and devices' access policies onto the blockchain. In addition, we design a smart contract, accessDecision, to make access decisions based on the role and access policies recorded on the blockchain. Moreover, we use the smart contract to manage a user access history list (AHL) which records the permitted access requests of users. With this AHL, a user can easily prove he/she has access permission to the device. Thus, we design another smart contract, AHLaccessDecision, to make the access decision based on the AHL. AHLaccessDecision is more efficient than accessDecision. In order to provide reliable auditing, we adopt the smart contract to record the audit records that contain the requests from users and the access decision from accessDecison/AHLaccessDecision. The smart contract mainly provides the following eleven function interfaces:

uploadURole (uID, uRole, sig): this function is invoked by DMSs and used to upload the users' roles onto the blockchain. Each DMS invokes this function by sending a

```
(1) Input: uID, uRole, sig
(2) output: bool
(3) % invoke by DMSs to upload the users' role in the DO.
(4) % sig = signature_DMS_ISK (uID, uRole)
(5) if verify(DMS_ID, sig) = true && uID.DO = = DMS_ID.DO then
(6) {
(7)    if stub.GetState(uID) ! = null then % update the user's role
(8)       updateAHL(uID_update,uID);
(9)    stub.PutState(uID,uRole); % store key = uPK, value = rRole in blockchain
(10)   return 0;
(11) }
(12) else
(13)   return 1;
```

Algorithm 1: uploadURole.

```
(1) Input: Uid, rule, sig
(2) output: bool
(3) % invoke by DMS to upload the role map rule.
(4) % Uid is used to retrieve role mapping rules between DOs.
(5) % Uid = 'DMS_ID_sour-DMS_ID_dest'
(6) % rule = {"role1,role2", "role1, role3",...}
(7) % sig = signature_DMS_des_ISK (Uid, rule)
(8) DMS_ID_des = extract(Uid)
(9) if verify(DMS_ID_des, sig) == true then
(10)   stub.PutState(Uid, rule); % store key = Uid, value = rule in blockchain
(11)   return 0;
(12) else
(13)   return 1;
```

Algorithm 2: uploadRM.

```
(1) Input: devID, policy, sig
(2) output: bool
(3) % invoke by DMS to upload the device's access policies.
(4) % policy = {"role1, op1&op2&op3";...}
(5) % sig = signature_DMS_ISK (policy)
(6) if verify (DMS_ID, sig) = true && devID.DO = = DMS_ID.DO then
(7) {
(8)    if stub.GetState (devPK) ! = null then
(9)       updateAHL (policy_update, devPK);
(10)   stub.PutState (devPK, policy);
(11)   return 0;
(12) }
(13) else
(14)   return 1;
```

Algorithm 3: uploadPolicy.

user's ID (uID), role, and the signature signed by the DMS. The function verifies the signature and determines whether the user uploaded by the DMS is in its own DO. Then, the blockchain stores or updates the role and uses the uID as the key to retrieve the user's role. (Algorithms 1–12).

uploadRM (Uid, rule, sig): this function is invoked by DMSs and used to upload the role mapping rules between the DOs onto the blockchain. DMSs jointly formulated the role mapping rules between DOs. In addition, each DMS can only upload the rule sets that map external DO roles to its

```
(1) Input: uID
(2) output: string
(3) % obtain the user's role from blockchain.
(4) role = stub.GetState (uID)
(5) if role ! = null then
(6) return role;
(7) else
(8) return '9000';
```

Algorithm 4: getURole.

```
(1) Input: role, Uid
(2) output: string
(3)    % map a role to the DMS's role.
(4)    rule = Stub.GetState (Uid)
(5)    role' = match (role, rule)
(6)    if (role' ! = null) then
(7)       return role';
(8)    else
(9)       return '9001';
```

Algorithm 5: mapRole.

```
(1) Input: devID
(2) output: string
(3) % obtain the device's access policy.
(4)    policy = Stub.GetState (devID);
(5)    if (policy ! = null) then
(6)       return policy;
(7)    else
(8)       return '9002';
```

Algorithm 6: getPolicy.

local DO roles. DMSs construct the unique identifier (Uid), DMS_ID_sou || DMS_ID_des, as the key to retrieve the rule set, where DMS_ID_sou is the ID of the external DO's DMS and DMS_ID_des is the ID of the local DO's DMS. The DMS invokes this function by sending a Uid, the role mapping rules, and the signature signed by the DMS. This function extracts the DMS_ID_des from the Uid to verify the signature. Then, it stores or updates the role mapping rules and uses the Uid as the key to retrieve the role mapping roles.

uploadPolicy (devID, policy, sig): this function is invoked by DMSs and used to upload the devices' access policies onto the blockchain. Each DMS invokes this function by sending a device's ID (devID), the device's access policies, and the signature signed by the DMS. The function verifies the signature and determines whether the device uploaded by the DMS is in its own DO. Then, the blockchain stores or updates the access policies and uses the devID as the key to retrieve the device's access policies.

getURole (uID): Once a user's ID is received, this function returns the role of the user.

mapRole (role, Uid): once a role and the unique identifier of the rule mapping rules between two Dos are received, this function maps the role according to the role mapping rules.

getPolicy (devID): Once a device's public key is received, this function returns the access policies of the device.

AccessDecision (request, sig): this function is invoked by DMSs and used to make an access decision. Once an access request is received, this function gets the user's role via invoking getURole, maps the user's role via invoking mapRole, and gets the device's policy via invoking getPolicy. Then, it determines whether the mapping role is satisfied with the access policies of the device. Finally, it records the audit records on the blockchain via invoking recordAudit and adds the permitted access request to the access history list.

AHLaccessDecision (request, sig): this function is used to make an access decision based on AHL. Once an access request is received, this function retrieves the permitted access history list and makes the access decision based on the receiving result.

recordAudit (request, decision): This function is used to record the requests from users and the access decision from accessDecison/AHLaccessDecision on the blockchain.

addAHL(uID||devID||op): This function is used to add the permitted access requests to the AHL.

getAHL(): this function is used to get the AHL.

updateAHL (string, string): this function is used to delete the invalid permitted access request of the AHL when DMSs updating user roles or device access policies. If a user's role is updated, this function deletes all the list which contains the user's ID in the AHL. Similarly, if a device access policy is updated, this function deletes all the list which contains the device's ID in the AHL.

5.3. Construction of the System. There are four major phases of the system, including System Setup, Access Control, Data Transmission, and Handel Dispute. The key notations are listed in Table 2.

System setup: DMSs upload the users' roles, the role mapping rules, and access policies onto the blockchain via invoking uploadURole, uploadRM, and uploadPolicy.

Access control: when a user wishes to publish a cross-domain access request, he/she constructs the request from his/her requirement. For example, a user wants reading the data of a device which belongs to other DO; the request constructed by the user, in this case, is request = (tag, $uID, devID, r$, sig), where the sig = $\text{sig}_{uIsk}(\text{tag}, uID, devID, r)$ is signed by the user. Then, the user broadcasts the request in the DMSs network. If the tag is 00, the DMS invokes accessDecision. If the tag is 01, the DMS invokes AHLaccessDecision.

Data transmission: once the access decision is received, the device returns data to the user who is permitted to access the data resource. The device generates a session

```
(1) Input: request, Uid, sig
(2) output: string
(3) % invoke by DMS to make an access control.
(4) % sig = signature_uIsk (request)
(5) % request = 00|| uID || devID || op, op is an access control order.
(6) if verify (uID, sig) = true then
(7) role = getURole (uID)
(8) if role = = '9000' then
(9)    {
(10)   recordAudit (request, '9000');
(11)   return '9000';
(12)   }
(13)   DMS_ID_sou = find_DMS_ID (uID) % find DMS's ID whose DMS_ID_sou.DO = = uID.DO
(14)   DMS_ID_des = find_DMS_ID (devID) % find DMS's ID whose DMS_ID_des.DO = = devID.DO
(15)   role' = mapRole (role, DMS_ID_sou||DMS_ID_des);
(16)   if role' = = '9001' then
(17)   {
(18)   recordAudit (request, '9001');
(19)   return '9001';
(20)   }
(21)   policy = getPolicy (devID)
(22)   if policy = = '9002' then
(23)   {
(24)   recordAudit (request, '9002')
(25)   return '9002';
(26)   }
(27)   if ((role, devID, op) ∈ policy) then
(28)   recordAudit (request, permit);
(29)   addAHL (uID||devID||op);
(30)   return permit;
(31)   else
(32)      recordAudit (request, '9003');
(33)      return '9003';
(34) else
(35)   return 'sign_error';
```

Algorithm 7: accessDecision.

```
(1) Input: request, sig
(2) output: string
(3) % invoke by users to make an AHL based access control.
(4) % sig = signature_uIsk (request)
(5) % request = 01|| uID || devID || op, op is an access control order.
(6) if verify (uID, sig) = true then
(7)   ahList = getAHL();
(8)   if ahList.isNoexist (uID||devID||op) then
(9)      recordAudit (request, '9004');
(10)     return '9004';
(11)  else
(12)     recordAudit (request, permit);
(13) else
(14)  return 'sign_error';
```

Algorithm 8: AHLaccessDecision.

```
(1) Input: request, decision
(2) output: string
(3) % record audit on blockchain
(4) % decision = 9000/9001/9002/9003/9004/permit
(5) time = time.Now();
(6) stub.PutState (request||time, decision);
```

Algorithm 9: recordAudit.

```
(1) Input: uID||devID||op
(2) output: bool
(3) % add a permitted access request on the AHL.
(4)    ahList = getAHL();
(5)    if ahList.isNoexist (uID||devID||op) then
(6)       ahList.push (uID||devID||op);
(7)    return 0;
```

Algorithm 10: addAHL.

```
(1) Input: null
(2) output: string
(3) % get the AHL.
(4) return ahList;
```

Algorithm 11: getAHL.

key to encrypt the data $\text{Enc}_{\text{key}}(\text{data})$ and signs the hash of the data $\text{Sign}_{sk_d}(H(\text{data}))$, then returns $\{Enc_{key}(data), Enc_{pk_u}(key), Sign_{sk_d}(H(\text{data}))$ to the user.

Handle dispute: users can verify access decisions (9000/9001/9002/9003/9004). While the DMS can trace back to a user's access record to detect the user's abnormal access behavior. Moreover, the audit records can also give evidence to the AHL.

6. Security Analysis

In this section, we will explain how our proposed system meets all the security goals as mentioned in Section 5.3.

6.1. DMSs' Attack Resistant. We develop a blockchain to record the corresponding role mapping rules between DMSs and the access policies of the devices. Users can verify whether their roles satisfied the access policies. Therefore, our system can resist DMSs denying granted user access.

6.2. Modification Attack Resistant. The modified broadcasted transactions and replied data will be discovered and refused because of the signature and hash functions. Meanwhile, the audit records are stored on the blockchain, which inherits the solutions to resist modification attacks.

6.3. Man-in-the-Middle Attack Resistant. We use asymmetric encryption algorithm to encrypt the returned resources (e.g., data) and then use an asymmetric encryption algorithm and the requester's public key to encrypt the session key. Attacks cannot decrypt, so they cannot read resources.

7. Performance Evaluation

In this section, we give a performance evaluation of the system.

7.1. System Setting. The system's efficiency mainly depends on the blockchain platform and computing platform. For instance, in this paper, we use the Hyperledger Fabric v1.2.0 (HLF) to test the designed smart contract in our personal computer, where the system configuration is Ubuntu 18.04 LST OS (64 bits) with an Inter (R) core (TM) i3-2130 CPU@ 3.40GHz and 4-G RAM.

7.2. Blockchain Efficiency. Generally, we simulate a system with two DOs, each DO contains three users and three devices. Each user assigned a role, and each device formulates three access policies.

During the System Setup phase, DMSs upload the users' roles, role mapping rules, and devices' policies of the DOs onto the blockchain, via invoking uploadURole, uploadRM, and uploadPolicy. We obtained the approximate time cost of these smart contracts using shell script with 1000 running times. Table 3 shows the time cost for each smart contract during the system setup phase.

After the system setup, we tested the AHLaccessDecision and accessDecision and obtained the approximate time cost of these smart contracts using shell script with 1000 running times. Table 4 shows the time cost for each smart contract during the access control phase.

In our system, users can verify an access decision (9000/9001/9002/9003/9004) recorded on the blockchain. Concretely, a user gets the audit records form blockchain and then verifies the correctness of the access decision on their own devices. The time cost of the handle dispute phase is shown in Table 5.

To evaluate the scalability of our system, we also conduct extensive system performance evaluation by increasing the number of DO from 2 to 5 and increasing the number of users and devices inside the DO from 3 to 20, 50, and 80.

Figures 4–6 show the time cost for the system setup, access control, and handle dispute with different numbers of DOs whose users and devices number range from 3 to 20, 50, and 80.

```
(1) Input: string, string % userRole_update, policy_update %uID, devID
(2) output: bool
(3) ahList = getAHL();
(4) Case "userRole_update";
(5) for list in ahList
(6)       if list.uID = = uID then
(7)          ahList.DeleteRow (list);
(8) return 0;
(9) Case "policy_update";
(10)   for list in ahList
(11)   if list.devID = = devID then
(12)          ahList.DeleteRow (list);
(13) return 0;
```

Algorithm 12: updateAHL.

Table 2: Symbols of the system.

Symbol	Description
$\mathbf{H}(.)$	Collision resistant hash function
$\mathbf{Sign}(.)$	Signature algorithm
$\mathbf{Enc}_{\mathbf{key}}(.)$	Symmetric encryption algorithm
$\mathbf{Enc}_{\mathbf{pk}}(.)$	Asymmetric encryption algorithm
$\{\mathbf{pk}_{\mathbf{DMS}}, \mathbf{sk}_{\mathbf{DMS}}\}$	The device's DMS's key pair
$\{\mathbf{pk}_{\mathbf{u}}, \mathbf{sk}_{\mathbf{u}}\}$	The user's key pair
$\{\mathbf{pk}_{\mathbf{d}}, \mathbf{sk}_{\mathbf{d}}\}$	The device's key pair
data	Resource data requested by users

Table 3: System setup phase's time cost for each smart contract.

Smart contract (implement function)	Time cost (ms)
System setup	
uploadURole (user roles storage) (user roles update)	1.7786 2.3397
uploadRM (role mapping rules storage)	2.9921
uploadPolicy (access policies storage) (access policies update)	1.8215 3.0159
Total time	**6.5922**

Table 4: Access control phase's time cost for each smart contract.

Access control phase		
Smart contract (implement function)	Time cost (AHLaccessDecision)	Time cost (accessDecision)
getURole (get user's role)	0	0.087
mapRole (map user's role)	0	0.679
getPolicy (get device's policy)	0	0.023
getAHL (get permitted access request list)	0.0217	0
Access control (make access decision)	0.0132	0.0872
addAHL (add permitted access request)	0.0202	0.0202
recordAudit (store request and access decision)	0.2055	0.2055
Total time	**0.2606**	**1.1019**

Table 5: Handle dispute phase's time cost.

Handle dispute phase	
Function (verify error)	Time cost (ms)
Audit (9000)	0.134
Audit (9001)	0.476
Audit (9002)	0.0518
Audit (9003)	0.0326
Audit (9004)	0.0152

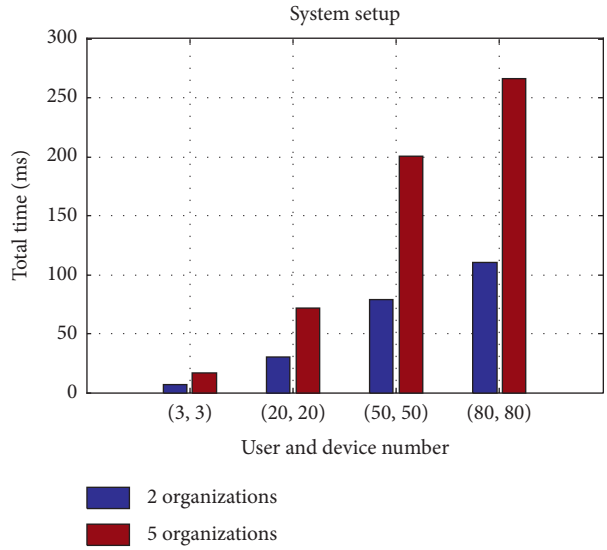

Figure 4: Time cost for system setup phase with different numbers of DOs and entities.

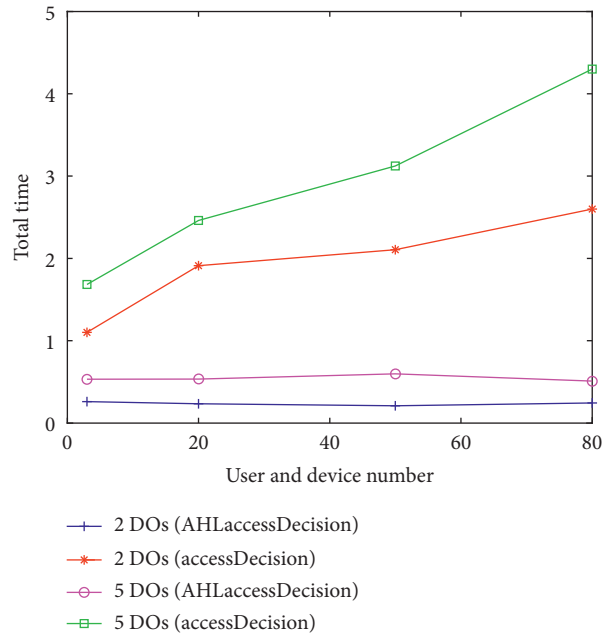

Figure 5: Time cost for access control phase with different numbers of DO entities.

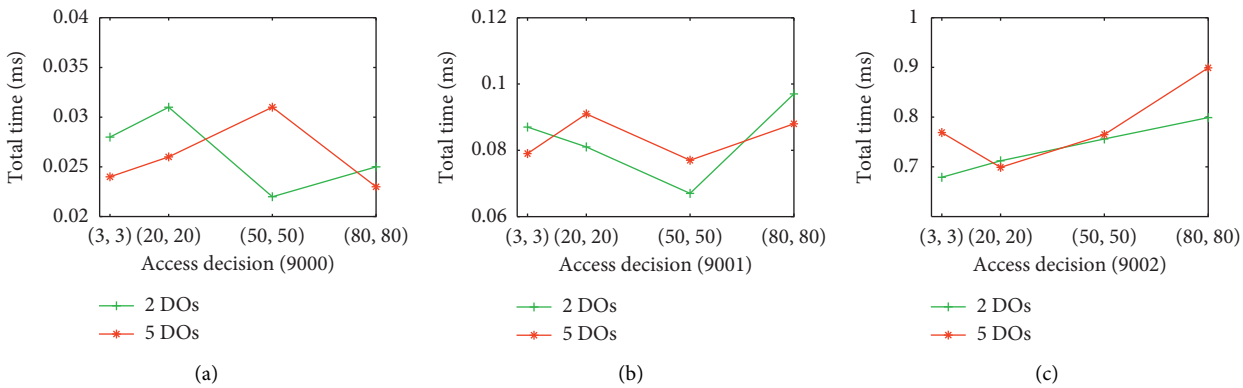

Figure 6: Continued.

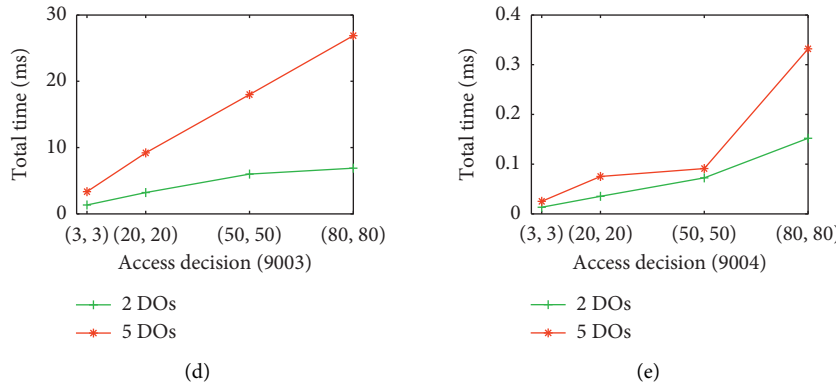

Figure 6: Time cost for the handle dispute phase of different numbers of DOs and entities.

8. Conclusions

Cross-domain access control and blockchain are two trending topics at the moment, and in this article, we proposed a cross-domain access control system based on blockchain. We demonstrated how blockchain can be utilized with role mapping technology to provide a user-certification and access nonreputation cross-domain access control. Specifically, the system uploads the users' roles, role mapping rules, and access policies onto the blockchain, realizing the access control process trustful. Considering the efficiency of the blockchain, we design an efficient smart contract to make the access decision based on the AHL. Finally, we record the user's access request and access decision onto the blockchain to realize the access nonreputation. The security analysis has proved our system to be secure in practical application, and the simulation experiments demonstrated that our scheme is feasible. In the future, we will research on how to solve the centralized limitations of IBS and enhance the distributed characteristics of the system.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant no. XDC02070600.

References

[1] B. Adriano, N. Yokoya, J. Xia, G. Baier, and S. Koshimura, "Cross-domain-classification of tsunami damage via data simulation and residual-network-derived features from multi-source images," in *Proceedings of the International Conference on Geoscience and Remote Sensing Symposium*, pp. 4947–4950, Yokohama, Japan, July 2019.

[2] J. Wu, M. Dong, K. Ota, M. Tariq, and L. Guo, "Cross-domain fine-granted data usage control service for industrial wireless sensor networks," *IEEE Access*, vol. 3, pp. 2939–2949, 2017.

[3] B. Jing, C. Lu, D. Wang, F. Zhuang, and C. Niu, "Cross-domain labeled LDA for cross-domain text classification," in *Proceedings of the International Conference on Data Mining (ICDM)*, pp. 187–196, Singapore, Singapore, December 2018.

[4] J. Shi and Q. Wang, "Cross-domain variational autoencoder for recommender systems," in *Proceedings of the International Advanced Infocomm Technology (ICAIT)*, pp. 67–72, Jinan, China, October 2019.

[5] A. S. Salehi, C. Rudolph, and M. Grobler, "A dynamic cross-domain access control model for collaborative healthcare application," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 643–648, Arlington, VA, USA, April 2019.

[6] K. Niu, Y. Huang, and L. Wang, "Fusing two directions in cross-domain adaption for real life person search by language," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 1815–1818, Seoul, Korea (South), October 2019.

[7] IoT security white paper 2018 [Online]. Available: https://www.huawei.com/minisite/iot/img/iot_security_white_paper_2018_v2_en.pdf.

[8] X. Yang and H. Wang, "A cross-domain access control model based on trust measurement," *Wuhan University Journal of Natural Sciences*, vol. 21, no. 1, pp. 21–28, 2016.

[9] M. L. Michelle, J. P. Arsenault, J. Bresee et al., "Access control for home data sharing: attitudes, needs and practices," in *Proceedings of the 28th Conference Human Factors in Computing Systems*, pp. 10–15, Atlanta,, GA, USA, April 2010.

[10] F. Cai, N. Zhu, J. He, P. Mu, W. Li, and Y. Yu, "Survey of access control models and technologies for cloud computing," *Cluster Computing*, vol. 22, no. S3, pp. 6111–6122, 2019.

[11] Z. Q. He, Y. L. Zhang, L. G. Zhang et al., "Access control architecture of service composition based on role mapping," *Computer Technology and Development*, vol. 25, no. 10, pp. 149–153, 2015.

[12] H. Zhang, J. Wang, and J. Chang, "A multi-level security access control framework for cross-domain networks," in *Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 316–319, Guangzhou, China, July 2017.

[13] Z. Xu, F. Zhang, W. Wang, H. Liu, and X. Kong, "Exploiting trust and usage context for cross-domain recommendation," *IEEE Access*, vol. 4, pp. 2398–2407, 2016.

[14] S. R. Chhetri, J. Wan, and M. A. A. Faruque, "Cross-domain security of cyber-physical systems," in *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 200–205, Chiba, Japan, January 2017.

[15] J. Du, C. Chen, J. Zhu, and X. Li, "Research on association securities in cross-domain interoperation model in pervasive computing," in *Proceedings of the Third International Conference on Pervasive Computing and Applications*, pp. 953–958, Alexandria, Egypt, October 2008.

[16] A. Liguori, F. Benedetto, G. Giunta, N. Kopal, and A. Wacker, "SoftGap: a multi independent levels of security cross-domain solution," in *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud*, pp. 754–759, Rome, Italy, August 2015.

[17] C. Lin, D. He, N. Kumar, X. Huang, P. Vijayakumar, and K.-K. R. Choo, "HomeChain: a blockchain-based secure mutual authentication system for smart homes," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 818–829, 2020.

[18] Q. Liu, H. Zhang, J. Wan, and X. Chen, "An access control model for resource sharing based on the role-based access control intended for multi-domain manufacturing internet of things," *IEEE Access*, vol. 5, pp. 7001–7011, 2017.

[19] R. Patel, U. Thakar, and V. Tewari, "A mechanism for operation level role based access control in web services," in *Proceedings of the 7th International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 384–388, Nagpur, India, November 2017.

[20] T. Phillips, X. Yu, B. Haakenson, and X. Zou, "Design and implementation of privacy-preserving, flexible and scalable role-based hierarchical access control," in *Proceedings of the 1st IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 46–55, Los Angeles, CA, USA, December 2019.

[21] H. Chen, C. Chang, and F. Leu, "Implement of agent with role-based hierarchy access control for secure grouping IoTs," in *Proceedings of the 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 120–125, Las Vegas, NV, USA, January2017.

[22] J. F. Chan, T. C. Yang, and H. T Liaw, "A cross cloud authorization mechanism using NFC and RBAC technology," in *Ubiquitous Computing Application and Wireless Sensor. Lecture Notes in Electrical Engineering*, J. Park, Y. Pan, HC. Chao, and G. Yi, Eds., vol. 331, Springer, Berlin, Germany, 2015.

[23] N. Ghosh, D. Chatterjee, and S. K. Ghosh, "An efficient heuristic-based role mapping framework for secure and fair collaboration in SaaS cloud," in *Proceedings of the International Conference on Cloud and Autonomic Computing*, pp. 227–236, London, UK, September 2014.

[24] L. Diao, H. Wang, S. Alsarra, I. Yen, and F. Bastani, "A smart role mapping recommendation system," in *Proceedings of the 43rd IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pp. 135–140, Milwaukee, WI, USA, 2019.

[25] G. Denker, J. Millen, and Y. Miyake, "Cross-domain access control via PKI," in *Proceedings of International Workshop on Policies for Distributed Systems and Networks*, pp. 202–205, Monterey, CA, USA, August 2002.

[26] A. Kapadia, J. Al-Muhtadi, R. H. Campbell et al., "Irbac 2000: secure interoperability using dynamic role translation," in *Proceedings of International Conference Internet Computing (IC)*, pp. 26–29, Las Vegas, NV, USA, June 2000.

[27] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

[28] A. Ouaddah, A. Ait Ouahman, and A. A. Ouahman, "Fair-access: FairAccess: a new blockchain-based access control framework for the Internet of Things," *Security and Communication Networks*, vol. 9, no. 18, pp. 5943–5964, 2016.

[29] O. Novo, "Blockchain meets iot: an architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[30] B. Dundua and M. Rukhaia, "Towards integrating attribute-based access control into ontologies," in *Proceedings of the 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON)*, pp. 1052–1056, Lviv, Ukraine, July 2019.

[31] V. C. Hu, D. F. Ferraiolo, R. Kuhn et al., *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, pp. 800–162, NIST Special Publication, Gaithersburg, MD, USA, 2014.

[32] M. A. Islam and S. Madria, "A permissioned blockchain based access control system for IoT," in *Proceedings of the IEEE International Conference on Blockchain (Blockchain)*, pp. 469–476, Atlanta, GA, USA, July 2019.

[33] D. R. Putra, B. Anggorojati, and A. P. Pratama Hartono, "Blockchain and smart-contract for scalable access control in internet of things," in *Proceedings of the International Conference on ICT for Smart Society (ICISS)*, pp. 1–5, Bandung, Indonesia, November 2019.

[34] I. Sukhodolskiy and S. Zapechnikov, "A blockchain-based access control system for cloud storage," in *Proceedings of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp. 1575–1578, Moscow, Russia, January 2018.

[35] Y. Zhou, Y. Guan, Z. Zhang, and F. Li, "A blockchain-based access control scheme for smart grids," in *Proceedings of the International Conference on Networking and Network Applications (NaNA)*, pp. 368–373, Daegu, Korea (South), October 2019.

[36] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[37] H. Guo, W. Li, M. Nejad, and C. Shen, "Access control for electronic health records with hybrid blockchain-edge architecture," in *Proceedings of the IEEE International Conference on Blockchain (Blockchain)*, pp. 44–51, Atlanta, GA, USA, July 2019.

[38] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, "Blockchain-based, decentralized access control for IPFS," in *Proceedings of the IEEE International Conference 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)*, pp. 1499–1506, Halifax, NS, Canada, July 2018.

[39] S. T. Chen, J. F. Xu, Y. X. Hang, and J. W. Li, "Role-based access control for memory security on Network-on-Chips," in *Proceedings of the 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1422–1424, Hangzhou, China, October 2016.

[40] R. Ghazal, A. K. Malik, B. Raza, A. R. Shahid, and H. Alquhayz, "Intelligent role-based access control model and framework using semantic business roles in multi-domain environments," *IEEE Access*, vol. 8, pp. 12253–12267, 2020.

[41] G. Qadeer, R. Galler, and B. Emmanuel, "Access controls for IoT networks," *SN Computer Science*, vol. 1, no. 1, pp. 1–13, 2020.
[42] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://bitcoin.org/bitcoin.pdf.
[43] Ethereum smart contract platform [Online]. Available: https://www.ethereum.org/.
[44] EOS.IO Technical White Paper v2 [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md.
[45] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, Chicago, Illinois, USA, July 2016.
[46] E. Kiltz and G. Neven, "Identity-based signatures," *Identity-Based Cryptography*, vol. 2, pp. 31–44, 2009.
[47] L. Zhou, L. Wang, Y. Sun, and P. Lv, "BeeKeeper: a blockchain-based IoT system with secure storage and homomorphic computation," *IEEE Access*, vol. 6, pp. 43472–43488, 2018.