computers

Article

BlendCAC: A Smart Contract Enabled Decentralized Capability-Based Access Control Mechanism for the IoT

Ronghua Xu [1], Yu Chen [1,*], Erik Blasch [2] and Genshe Chen [3]

1 Department of Electrical and Computer Engineering, Binghamton University, SUNY, Binghamotn, NY 13902, USA; rxu22@binghamton.edu

2 The U.S. Air Force Research Lab, Rome, NY 13441, USA; erik.blasch@gmail.com

3 Intelligent Fusion Technology, Inc., Germantown, MD 20876, USA; gchen@intfusiontech.com

* Correspondence: ychen@binghamton.edu; Tel.: +1-607-777-6133

Received: 2 May 2018; Accepted: 11 July 2018; Published: 13 July 2018

Abstract: While Internet of Things (IoT) technology has been widely recognized as an essential part of Smart Cities, it also brings new challenges in terms of privacy and security. Access control (AC) is among the top security concerns, which is critical in resource and information protection over IoT devices. Traditional access control approaches, like Access Control Lists (ACL), Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC), are not able to provide a scalable, manageable and efficient mechanism to meet the requirements of IoT systems. Another weakness in today's AC is the centralized authorization server, which can cause a performance bottleneck or be the single point of failure. Inspired by the smart contract on top of a blockchain protocol, this paper proposes BlendCAC, which is a decentralized, federated capability-based AC mechanism to enable effective protection for devices, services and information in large-scale IoT systems. A federated capability-based delegation model (FCDM) is introduced to support hierarchical and multi-hop delegation. The mechanism for delegate authorization and revocation is explored. A robust identity-based capability token management strategy is proposed, which takes advantage of the smart contract for registration, propagation, and revocation of the access authorization. A proof-of-concept prototype has been implemented on both resources-constrained devices (i.e., Raspberry PI nodes) and more powerful computing devices (i.e., laptops) and tested on a local private blockchain network. The experimental results demonstrate the feasibility of the BlendCAC to offer a decentralized, scalable, lightweight and fine-grained AC solution for IoT systems.

Keywords: decentralized access control; Internet of Things (IoT); blockchain protocol; smart contract; federated delegation; capability-based access control

1. Introduction

With the proliferation of the Internet of Things (IoT), a large number of physical devices are being connected to the Internet at an unprecedented scale. The prevalence of IoT devices has changed human activity by ubiquitously providing applications and services that have revolutionized transportation, healthcare, industrial automation, emergency response, and so on [1]. These capabilities offer both measurement data and information context for situation awareness (SAW) [2,3]. While benefiting from the large-scale applications like Smart Grid and Smart Cities, IoT systems also incur new concerns for security and privacy. With their increased popularity, connected smart IoT devices without sufficient security measures increase the risk of privacy breaches and various attacks. Security issues, such as privacy, authentication, access control, system configuration, information storage and management, are the main challenges that these IoT-based applications are facing [4].

Among the top security challenges in IoT environments, access authorization is critical in resource and information protection. Conventional access control approaches, like the Access Control List (ACL), Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC) have been widely used in information technology (IT) systems. However, they are not able to provide a manageable and efficient mechanism to meet the requirements raised by IoT networks due to the following features:

- *Scalability*: The fast growing number of devices and services also pose increasing management overload in access control systems that are based on ACL or RBAC models. Access control strategies are expected to be able to handle the scalability problem resulting from the distributed IoT networks.
- *Heterogeneity*: IoT systems normally integrate heterogeneous cyber physical objects with variant underlying technologies or in different application domains, and each domain or platform has its own specific requirements for identity authentication and authorization policy enforcement. Both RBAC and ABAC have been found to be inflexible in providing complex arrangements to support delegation and transitivity which are essential for efficient and effective intra-domain authorization and access control.
- *Spontaneity*: Traditional RBAC and ABAC systems envisage planned and long-lived patterns, while the IoT world is mainly characterized by short-lived, often casual and/or spontaneous interactions [5], in which an access control scheme is required to deal with dynamic challenges.
- *Interoperability*: IoT devices are usually resource-constrained and cannot support heavy computational and large storage-required applications. Smart devices connect to each other by low power and lossy networks. Consequently, the access control protocol should be lightweight and not impose significant overhead on devices and communication networks.

The extraordinary large number of devices with heterogeneity and dynamicity necessitate more scalable, flexible and lightweight access control mechanisms for IoT networks. In addition, the majority of the AC solutions rely on centralized authorities. Although the delegation mechanism helps migrate certain intelligence from the centralized cloud server to a near-site fog or edge network, the power of policy decision making and identity management is exclusively located in the cloud center. IoT networks need a new AC framework that provides a decentralized authentication and authorization scheme in trustless application network environments, such that intelligence can be diffused among large number of distributed edge devices.

While being well-known as the fundamental protocol of Bitcoin [6], the first digital currency, the blockchain protocol, has been recognized as having the potential to revolutionize the fundamentals of IT technology because of its many attractive features and characteristics, such as supporting decentralization and anonymity maintenance [7]. In this paper, a *BLockchain-ENabled, Decentralized, Federated, Capability-based Access Control* (BlendCAC) scheme is proposed to enhance the security of IoT devices. It provides a decentralized, scalable, fine-grained and lightweight AC solution to protect smart devices, services and information in IoT networks. An identity-based capability token management strategy is presented and the federated authorization delegation mechanism is illustrated. In addition, a capability-based access validation process is implemented on service providers that integrate SAW and customized contextualized conditions. The experimental results demonstrate the feasibility and effectiveness of the proposed BlendCAC scheme.

The major contributions of this work are as follows:

1. By leveraging the blockchain protocol, a decentralized, federate access control scheme is proposed, which is a scalable, fine-grained, and lightweight solution for today's IoT networks;
2. A complete architecture of a federated capability-based authorization system is designed, which includes delegation authority, capability management, and access right validation;
3. A capability-based federated delegation model is introduced, and the enforcement of polices is discussed in detail;

4. A concept-proof prototype based on smart contracts is implemented on resource-constrained edge devices and more powerful devices and is deployed on a local private blockchain network; and
5. A comprehensive experimental study is conducted that compares the proposed scheme with the well-known RBAC and ABAC models. The experimental results validate the feasibility of the BlendCAC approach in IoT environments without introducing significant overhead.

The remainder of this paper is organized as follows: Section 2 gives a brief review on state-of-the-art research in access control for IoT systems. Section 3 defines components of the federated delegation model including capability-based delegation and revocation. Then, Section 4 illustrates the details of the proposed BlendCAC system, and Section 5 explains the implementation of the proof-of-concept prototype. The experimental results and evaluation are discussed in Section 6. Finally, the summary, current limitations and on-going efforts are discussed in Section 7.

2. Background Knowledge and Related Work

2.1. Access Control Model in IoTs

Technology for the authentication and authorization of access to certain resources or services are among the main elements to protect the security and privacy for IoT devices [8]. As a fundamental mechanism to enable security in computer systems, AC is the process that decides who is authorized to have what communication rights on which objects with respect to some security models and policies [9]. An effective AC system is designed to satisfy the main security requirements, such as confidentiality, integrity and availability. However, recently raised security and privacy issues have pushed AC systems into the era of IoT to meet a higher standard with more design considerations such as high scalability, flexibility, lightweight and causality.

There are various AC methods and solutions with different objectives proposed to address IoT security challenges. The Role-Based Access Control (RBAC) model [10] provides a framework that specifies user access authorization to resources based on roles, and supports principles, such as least privilege, partition of administrative functions and separation of duties [11] in computer systems. However, a pure RBAC model presents a role explosion problem, which is inappropriate for the implementation of security policies that require the interpretation of complex and ambiguous IoT scenarios. The RBAC model implemented on IoTs adopts a Web of Things (WoTs) framework [12,13] to enforce AC policies on the smart objects via a web service application. An extended RBAC model is proposed that introduces context constraints which consider contextual awareness in AC decisions provided by the service on IoTs [14]. However, it is not able to address the key issues of implementing RBAC in distributed IoT networks, such as self-management to handle the explosion of roles and policies and the autonomy for the physical objects through device-to-device communication.

To address the weaknesses of RBAC model in a highly distributed network environment, an Attribute-based Access Control (ABAC) [15,16] was introduced in IoT networks to reduce the number of rules resulting from role explosion. In the ABAC, AC policies are defined through directly associating attributes with subjects. An efficient authentication and ABAC-based authorization scheme for the IoT perception layer has been proposed [17]. Based on user attribute certificates, access rights are granted by the AC authority to ensure fine-grained access control. However, specifying a consistent definition of the attributes within a domain or across different domains could significantly increase the effort and complexity of policy management as the number of devices grow, and hence, the attribute-based proposal is not suitable for large-scale distributed IoT networks.

Due to drawbacks that exist in traditional access control models, such as RBAC and ABAC, the requirements imposed by IoT scenarios cannot be satisfied. Given the many great advantages from an IoT perspective, such as scalability, flexibility, and he ability to be distributed and user-driven, IoT systems can support delegation and revocation [8]. Capability-based access control approaches have been considered a promising solution to IoT systems. The Access Control Matrix (ACM) model represents a good conceptualization of authorizations by providing a framework for describing

Discretionary Access Control (DAC) [11]. As two implementations of ACM, the Access Control List (ACL) and capability are widely used in authorization systems. In the ACL model, each object is associated with an access control list that saves the subjects and their access rights for the objects.

The ACL is a centralized approach that supports administrative activities with better traceability by implementing AC strategies on cloud servers [18]. However, as the number of subjects and resources increases, confused duty problems are identified in ACL and access rules become much more complex to manage. Due to the centralized management property, ACL cannot provide multiple levels of granularity, is not scalable and is vulnerable to a single point of failure. Meanwhile, in the capability-based access control (CapAC) model, each subject is associated with a capability list that represents its access rights to all concerned objects. The CapAC has been implemented in many large scale IoT-based projects, like IoT@Work [19].

Although capability-based methods have been used as a feature in many access control solutions for the IoT-based applications, the direct application of the original concept of capability-based access control model in a distributed networks environment has raised several issues, like capability propagation and revocation. To tackle these challenges, a Secure Identity-Based Capability (SICAP) System that enables the monitoring, mediating, and recording of capability propagations to enforce security policies as well as achieving rapid revocation capability by using an exception list was proposed [9]. The SICAP provides a prospective capability-based AC mechanism in distributed IoT-based networks. However, the centralized access control server (ACS) causes a performance bottleneck of the system, and the author did not provide a clear illustration of the security policy used in capability generation and propagation, nor was contextual information in making authorization decisions considered.

To enable contextual awareness in federated IoT devices, an authorization delegation method based on a Capability-based Context-Aware Access Control (CCAAC) model was proposed [20]. By introducing a delegation mechanism for the capability generation and propagation process, the CCAAC model has great advantages in terms of addressing scalability and heterogeneity issues in IoT networks. Given the requirement that prior knowledge of the trust relationship among domains in federated IoTs must be established, however, the proposed approach is not suitable universally for all IoT application scenarios. Inspired by the SUN DIGITAL ECOSYSTEM ENVIRONMENT project [21], a Capability-based Access Control (CapAC) model was proposed that adopted a centralized approach for managing access control policy [5]. However, the proposed CapAC scheme depends on a centralized authority and does not consider the lightweight requirement of smart IoT devices. Designing a lightweight access control architecture for an IoT system that is capable of handling security using a minimum number of policies and dynamic identity management, Shantanu Pal [22] proposed a hybrid model with attributes, capabilities and role-based access control. By applying attributes both in role-membership assignments and conditions in permissions, policy management was simplified to handle the scalability of IoT systems with a large number of things and users. However, the proposed access control architecture is not implemented and performance analysis based on the testbed was not discussed.

2.2. Decentralized Access Control Mechanism for IoTs

To address the limitations in centralized access control solutions, designing a decentralized access control mechanism remains a popular research topic. The authors in references [23,24] proposed a distributed Capability-based Access Control (DCapAC) model, which was directly deployed on resource-constrained devices. The DCapAC allows smart devices to autonomously make decisions on access rights based on an authorization policy, and it has advantages in scalability and interoperability. However, capability revocation management and delegation were not discussed, nor were granularity or context awareness considered. A number of solutions has been proposed on the databox [25] platform to address security and privacy issues in distributed systems like IoTs. As flexible, decentralized cryptographic authorization credentials for cloud services, Macaroons using nested and chained message authentication code (MAC) are proposed to address decentralized delegation problems in IoT systems [26]. Through cryptographic operations, new restrictions on an authorization certificates are hashed to hash-based MAC (HMAC) and chained in a nested fashion, Macaroons support the attenuation, delegation and contextual confinement for cloud-based applications. However, the feasibility of Macaroons for IoT systems has not been verified given the scalability issues resulting from symmetric-key distribution as well as the high computational cost ofpublic-key operations on resource constraint IoT devices.

Owing to its many attractive characteristics, blockchain technology has been investigated in regard to whether it can offer a decentralized AC scheme in trustless network environments. A blockchain-based AC was proposed for the publication of AC policy and to allow distributed transfer access rights among users on Bitcoin networks [27]. The proposal allows distributed auditability, preventing a third party from fraudulently denying the rights granted by an enforceable policy. However, the solution still relies on an external, centralized policy database to fetch access right given the links stored in the blockchain, and the experimental results were not provided. Based on blockchain technology, FairAccess was proposed to offer a fully decentralized pseudonymous and privacy-preserving authorization management framework for IoT devices [28]. In FairAccess, the AC policies are enclosed in new types of transactions that are used to grant, get, delegate and revoke access. However, the scripting language used in Bitcoin allows the users to transcode two types of AC policies so the proposed framework cannot support more complex and granular access control models.

A similar public ledger solution called the Google DeepMind verifiable ledger was proposed to carry out an audit of the trust, confidence and verifiability of data for health services [29]. The given assumption that a trust relationship has been established among institutions who can be relied on to verify the integrity of ledgers, the use of a consensus algorithm by the blockchain is not necessary in the Google DeepMind verifiable ledger, so that the computational wastefulness of the blockchain is avoided. However, the validation of a verifiable ledger by a reputable third-party authority is not suitable for trustless IoT network environments, and a platform is not implemented to verify the feasibility of the proposed scheme in IoT systems.

2.3. Blockchain and Smart Contract

The blockchain is the fundamental framework of Bitcoin [6], which was introduced by Nakamoto in 2008. The blockchain is the public ledger that allows the data be recorded, stored and updated distributively. Due to its nature, the blockchain is a decentralized architecture that does not rely on a centralized authority. The transactions are approved and recorded in blocks created by miners, and the blocks are appended to the blockchain in a chronological order. The blockchain uses a consensus mechanism to maintain the sanctity of the data recorded on the blocks. Thanks to the "trustless" proof mechanism enforced through mining tasks on miners across networks, users can trust the system of the public ledger stored worldwide on many different decentralized nodes maintained by "miner-accountants," as opposed to having to establish and maintain trust with the transaction counter-party or a third-party intermediary [30]. The blockchain is the ideal architecture to ensure distributed transactions between all participants in a trustless environment.

The blockchain has shown its success in the decentralization of currency and payments like Bitcoin. Currently, the design of programmable money and contracts which support a variety of customized transaction types has become a trend to extend blockchain applications beyond the cryptocurrency domain. The *smart contract*, which has emerged from the smart property, is a method of using a blockchain to achieve agreement among parties, as opposed to relying on third parties to maintain a trust relationship. By using cryptographic and other security mechanisms, smart contracts combine protocols with user interfaces to formalize and secure relationships over computer networks [31].

A smart contract is, essentially, a collection of pre-defined instructions and data that has been recorded at a specific address of a blockchain. Smart contracts and all transactions generated in the blockchain network are saved as Merkle trees in each block. A Merkle tree is a constructed bottom-to-up binary tree data structure. All transaction data are firstly hashed and saved as leaf nodes, and then the Merkle tree stores successive children node hashes from leaf-to-root upon any changes to the tree. Finally, the root hash value of the Merkle tree is generated and imprinted in the block to allow the auditability of large sets of data. Any attempt to change smart contract data in the Merkle tree will generate a different root hash value. After being validated by a consensus mechanism running on a large volume of miners in the blockchain network, the changed data can be accepted by whole network by generating a new block and appending to the blockchain. Public functions or application binary interfaces (ABIs) defined by a smart contract allow the user to interact with them, given the pre-defined business logic or contract agreement. Through encapsulating operational logic as a bytecode and performing Turing complete computation on distributed miners, a smart contract allows the user to transcode more complex business models as new types of transactions on a blockchain network. A smart contract provides a promising solution to allow the implementation of more flexible and fine-grained AC models on blockchain networks.

3. Federated Capability-Based Delegation and Revocation Model

In today's IoT based systems, data service and security enforcement are deployed on centralized cloud centers where abundant computing and storage resources are allocated. Such a centralized network architecture is not scalable for large-scale IoT networks, and management efforts for resource and security policy increase dramatically owning to their heterogeneity properties in highly decentralized IoT environments. Delegation enables an entity to give permission to let other entities function on its behalf by providing all or some of its rights. It is considered a useful and effective approach to improve the scalability of distributed systems and to decentralize access control tasks [32].

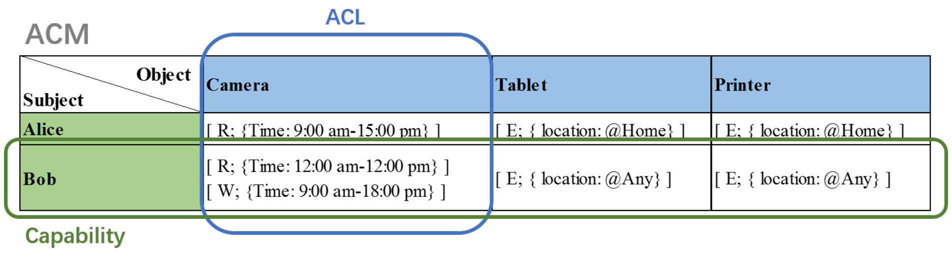
As an important factor for secure distributed systems, delegation has been recognized as one of the schemes to support access policy management in distributed computing environments [33]. Although a rule-based framework for role-based delegation and revocation was proposed [34], the proposal cannot support Capability-based Access Control systems. In this section we propose the *Federated Capability-based Delegation Model* (FCDM). This FCDM model supports capability-based hierarchy and multi-step delegation by introducing the delegation relationship.

3.1. Capability Access Control Model

Our Capability Access Control mode is composed of the following key components:

- The *subject* is a human being or device who applies for access right and interacts with the authorized service and resources on the service provider.
- An *object* is an entity who works as a service provider to offer services or resources for authorized subjects, such as a printer, NetCam or sensor.
- An *operation* is an action that can be carried out by authorized subjects to access service or resource on a target object, such as read, write or execute.
- A *constraint* is a set of attributes that defines an executable condition of an operation given a specific context or scenario, like time or location.
- *Permission* is a access right which defines a one-to-many relation assignment by combining operation and constraints.

The elements and relationships in the Capability Access Control model are depicted in Figure 1. The Access Control Matrix (ACM) includes sets of three basic elements: subject, *S*; object, *O*; and Permission, *P*. The Access Control List (ACL) and capability are two permission relationships in the ACM model. The ACL permission assignment is a many-to-one relationship between a subject and objects, which means that each object is associated with a set of Access Control Lists that save the subjects and their authorized permissions for the object. However, the capability model uses a subject-oriented permission assignment in which relationships between a subject and objects become one-to-many.



Subject \ Object	Camera	Tablet	Printer
Alice	[R; {Time: 9:00 am-15:00 pm}]	[E; { location: @Home}]	[E; { location: @Home}]
Bob	[R; {Time: 12:00 am-12:00 pm}] [W; {Time: 9:00 am-18:00 pm}]	[E; { location: @Any}]	[E; { location: @Any}]

Figure 1. Capability Access Control model.

The following is a list of definitions in the Capability Access Control model:

- S, O and P are sets of subjects, objects and permissions.
- ${}_{in}Cap_O \subseteq O \times P$ is the internal capability which defines a one-to-many relationship assignment between an object and permissions.
- ${}_{ext}Cap_{(S,O)} \subseteq S \times {}_{in}Cap_O$ is the external capability which specifies a one-to-many relationship assignment by associating a subject with a subset of internal capabilities.

The ${}_{in}Cap_O$ is defined by the object owner or policy authority to indicate all available service permissions supported by an object. The policy decision authority could generate ${}_{ext}Cap_{(S,O)}$ for a subject to access an object by authorizing permissions in ${}_{in}Cap_O$.

3.2. Federated Capability-Based Delegation Model

Delegation is an efficient mechanism to simplify access policy management by building a hierarchical relationship to reflect an organization's lines of authority and responsibility. Essentially a delegation hierarchy is a partial order relationship, $\preceq$. If x delegates the permissions to y, or y inherits the permissions of x, the delegation relation between x and y could be represented as a partial order, y$\preceq$x. A partial order is a reflexive, transitive and antisymmetric relationship.

To control the delegation propagation and simplify the federated delegation model, we specify that an entity can only be delegated once, so that any entity can only appear one time in the delegation tree. The requirement that a subject cannot be delegated any new permissions if the subject has already been assigned delegated permissions, the *delegation relationship (DR)*, in FCDM is defined as follows:

- $DR \subseteq {}_{ext}Cap \times {}_{ext}Cap$ is the one-to-many delegation relationship. A delegation relationship can be represented by $((S1,(O \times P1)),(S2,(O \times P2))) \in DR$, where P2$\preceq$P1. It indicates that subject S1 delegates subset of P1 to subject S2 as P2.

To confine the delegation relation propagation steps, it is necessary to set the delegation depth to define the maximum time that the delegation operation can be further performed. Two types of delegation, *single-step delegation* and *multi-step delegation*, are considered in FCDM. Single-step delegation prevents the delegated subject from further performing delegation, whereas multi-step delegation allows multiple delegate operations until ithe maximum delegation depth is reached. Thus, single-step delegation is considered to be a special case of multi-step delegation with the maximum delegation depth equal to one.

Multi-step delegation generates an ordered list of delegation relationships, called the *delegation path (DP)*. In general, a delegation path starts from an initial or root, ${}_{ext}Cap$, and is represented as the following notation:

$$DR_0 \rightarrow DR_1 \rightarrow \cdots \rightarrow DR_i \rightarrow \cdots \rightarrow DR_n$$

All delegation paths starting with the root ${}_{ext}Cap$ construct a hierarchical structure, the *delegation tree (DT)*. In the delegation tree, each node represents an ${}_{ext}Cap$ and each edge refers to a DR. The layer of ${}_{ext}Cap$ in the tree is defined as the delegation depth.

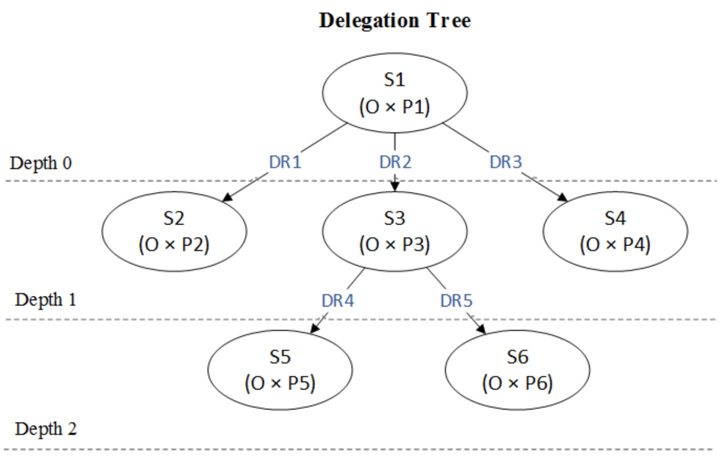
Given above discussions, the definitions and functions in FCDM are as follows:

- $DP \subseteq DR \times DR$ is an ordered list of delegation relationships indicating a delegation path.
- $DT \subseteq DR \times DR$ is a delegation relationship hierarchy representing a delegation tree.
- $N_{maxDepth}$ is a natural number representing the maximum delegation depth.
- $Ancestor$: ${}_{ext}Cap \rightarrow {}_{ext}Cap$ is a function that maps a ${}_{ext}Cap$ node to another parent ${}_{ext}Cap$ node in the delegation tree.
- $Path$: ${}_{ext}Cap \rightarrow DP$ is a function that maps an ${}_{ext}Cap$ node to a delegation path. Path(${}_{ext}Cap_i$) = $\{ DR_0 \rightarrow DR_1 \rightarrow \cdots \rightarrow DR_i \,|\, {}_{ext}Cap_i = Ancestor({}_{ext}Cap_{i-1}) \}$
- $DelegateDepth$: ${}_{ext}Cap \rightarrow N$ is a function that returns a delegation depth in delegation tree given ${}_{ext}Cap$ node, where N is a natural number set representing the delegation depth.

In order to illustrate the concepts of delegation path and delegation tree, a set of delegation relationship examples is listed:

$$\begin{aligned}
DR1 &: ((S1,(O \times P1)),(S2,(O \times P2))) \in DR \\
DR2 &: ((S1,(O \times P1)),(S3,(O \times P3))) \in DR \\
DR3 &: ((S1,(O \times P1)),(S4,(O \times P4))) \in DR \\
DR4 &: ((S3,(O \times P3)),(S5,(O \times P5))) \in DR \\
DR5 &: ((S3,(O \times P3)),(S6,(O \times P6))) \in DR
\end{aligned}$$

According to above delegation relationships, all delegation paths can be calculated by applying the $Path$ function and a delegation tree is built as shown in Figure 2. In the delegation tree, the parent node only delegates a subset of its permissions to a child, so that permission propagation over delegation path is essentially a partial order sequence. Take DP5 for example, permission delegation is represented as the partial order relationship $P6 \preceq P3 \preceq P1$.



Delegation Relation	Delegation Path
DR1	DP1: (S1, (O × P1)) → (S2, (O × P2))
DR2	DP2: (S1, (O × P1)) → (S3, (O × P3))
DR3	DP3: (S1, (O × P1)) → (S4, (O × P4))
DR4	DP4: (S1, (O × P1)) → (S3, (O × P3)) → (S5, (O × P5))
DR5	DP5: (S1, (O × P1)) → (S3, (O × P3)) → (S6, (O × P6))

Figure 2. An example of delegation paths and a delegation tree.

3.3. Capability-Based Delegation Authorization

Delegation authorization is mainly done to impose restrictions on which assigned access rights can be delegated to whom based on delegation authorization rules. In our proposed FCDM, the subject-to-subject delegation authorization relationship is defined as follows:

- ${}_{ext}Cap$, S, C, $N_{maxDepth}$ are sets of capability, subject and conditions for the authorization and maximum delegation depth, respectively.

- $can\_Delegate \subseteq {}_{ext}Cap \times S \times C \times N_{maxDepth}$.

The relationship $({}_{ext}Cap_{S1}, S2, C, N_{maxDepth}) \in can\_Delegate$ means that a subject, $S1$, who is a node in the delegation tree with capability ${}_{ext}Cap_{S1}$ can delegate its subset of permissions to subject $S2$ whose properties satisfy the conditions without exceeding the maximum delegation depth, $N_{maxDepth}$. For example, in Figure 2, $({}_{ext}Cap_{S4} \in {}_{ext}Cap\,) \wedge (({}_{ext}Cap_{S4}, S7, C, 1) \in can\_Delegate)$, where $N_{maxDepth} = 3$ and $C = \{{}_{ext}Cap_{S7} \notin {}_{ext}Cap\}$; thus $S4$ can delegate the subset of its permission $P4$ to new subject $S7$ as permission $P7$. As a result, the new delegation relationship, $DR6 : ((S4, (O \times P4)), (S7, (O \times P7))) \in DR$ and capability ${}_{ext}Cap_{S7} = (S7, (O \times P7)) \in {}_{ext}Cap$, is created and appended to DR3 as a leaf node of the delegation tree (DT).

3.4. Capability-Based Delegation Revocation

As an important process that accompanies the delegation mechanism, revocation refers to the process that nullifies the delegated permissions or attempts to roll back the state before permissions are delegated. The revocation approaches can be categorized into three dimensions [34]: *grant-dependency, propagation* and *dominance*. Our FCDM only considers two dimensions: *grant-dependency* and *propagation*.

Grant-dependency refers to the legitimacy of a subject who can take away assigned permissions from a delegated subject and has two types: *grant-dependent* and *grant-independent*. *Grant-dependent* revocation means that only the delegating subject (parent) can revoke the permissions from directly delegated subjects (children). *Grant-independent* revocation means any ancestor subject in the delegation path can revoke the delegated permissions from the offspring subjects.

Propagation specifies the extent of the revocation to subsequent delegated subjects. It can be categorized as *cascading* or *non-cascading*. *Cascading* revocation directly revokes delegated permissions from the subject as well as indirectly nullifying a set of subsequent propagated delegation relationships. In contrast, *non-cascading* revocation only takes away directly delegated permissions from child subjects.

To reduce the complexity in the revocation process, our FCDM enforces *grant-independent* and *cascading* rules in delegation revocation. Figure 3 is an example of grant-independent cascading revocation. Revocation authorization can be defined as follows:

- ${}_{ext}Cap$, S, $Ancestor()$ are sets of capability, subject and ancestor function, respectively.
- $can\_Revoke \subseteq {}_{ext}Cap \times S \times Ancestor(S)$.

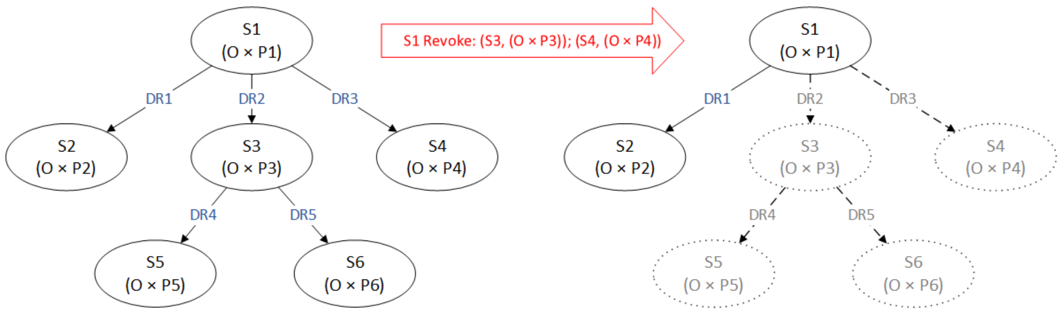


Figure 3. An example of grant-independent cascading revocation.

The relationship $({}_{ext}Cap_{S1}, Ancestor(S2)) \in can\_Revoke$ means that subject S1, who is the ancestor of subject S2, can revoke the delegated permissions of S2 as well as all indirect assigned permissions by S2 in a subsequent delegation relationship. As shown in Figure 3, owing to fact that both $S1 \in Ancestor(S3)$ and $S1 \in Ancestor(S4)$, revocation authorization satisfies the relationship $({}_{ext}Cap_{S1}, Ancestor(S3)) \vee ({}_{ext}Cap_{S1}, Ancestor(S4)) \in can\_Revoke$. As a result, the delegation relationships DR2 and DR3 are removed from the delegate tree and the delegated capabilities ${}_{ext}Cap_{S3}$ and ${}_{ext}Cap_{S4}$ are revoked. In addition, the subsequent relationships DR4 and DR5 assigned by S3 are also removed, and their associated capabilities ${}_{ext}Cap_{S5}$ and ${}_{ext}Cap_{S6}$ are revoked.

4. BlendCAC: A BLockchain-ENabled Decentralized Federated CapAC System

Most dominant IoT based systems, like Amazon Web Services (AWS) IoT, utilize a centralized cloud platform where abundant computing and storage resources are allocated to securely manage connected devices. As a result, all service access requests on devices need to be transmitted to remote servers for authentication and authorization. Such a centralized network architecture is not scalable for today's large IoT networks, and latencies are not tolerable in many mission-critical applications. To meet the requirements of real-time processing and instant decision making online, uninterrupted smart surveillance systems have been intensively studied and recently, the edge-fog-cloud computing paradigm has been leveraged [35–37]. Based on an automatic surveillance system architecture, the Federated Capability-based Access Control system (FedCAC) [38] was proposed to addresses scalability, granularity and dynamicity challenges in access control strategies for IoT devices. Through delegating part of the identify authentication and authorization task to the domain delegator, the workload of the centralized policy decision making center (PDC) is reduced. Migrating some processing validation tasks to local devices helps the FedCAC to be lighter and context-awareness enabled. Involving smart objects in the access right authorization process allows device-to-device communication, which implies better scalability and interoperability in an IoT network environment. However, a comprehensive capability-based delegation and revocation mechanism is not achieved. In addition, FedCAC is essentially still a centralized AC scheme, such that weaknesses include being the single-point of failure and performance bottleneck, are still not solved.

Inspired by the smart contract and blockchain technology, a decentralized federated Capability-based Access Control framework for IoTs, called BlendCAC, is proposed in this paper, and a prototype of proposal is implemented in a physical IoT network environment to verify the efficiency and effectiveness. The next subsection provides a comprehensive system design of the BlendCAC framework. Unlike the approaches discussed above, BlendCAC effectively provides AC strategies for IoTs with decentralization, scalability, granularity and dynamicity.

4.1. System Architecture of BlendCAC

Figure 4 illustrates the proposed BlendCAC system architecture, which is intended to function in a scenario including two isolated IoT-based service domains without pre-establishing a trust relationship. In our proposed BlendCAC framework, the cloud works as service provider to provide global profile data and security policy management, and the domain coordinator enforces delegated security policies to manage domain related devices and services. The FCDM model and AC policies are transcoded to the smart contract and deployed across the blockchain network, and identity authentication, authorization and access right verification are developed as service applications running on both cloud, coordinator and edge devices. The operation and communication modes are listed as follows:

1. *Registration*: All entities must create at least one main account defined by a pair of keys to join the blockchain network. Each account is uniquely indexed by its address that is derived from his/her own public key. This account address is ideal for identity authentication in the BlendCAC system given the assumption that the authentication process is ensured by a blockchain network. In our scenario, the identity authentication account addresses used are Virtual Identities (VIDs) which are recored in profile database, and identity management is deployed on two levels: the cloud and the coordinator. The cloud server maintains a global profile database, and the domain coordinator maintains a local profile database, and regular synchronization between the cloud server and domain coordinator ensures data consistency. New users can either send a registration request to the cloud or to the delegated coordinator. Once the identity information related to users or IoT devices is verified, the profile of each registered entity is created using his/her account address in the blockchain for the authentication process when an access right request happens. As a result, the domain coordinators are able to enforce delegated authorization policies and

perform decision-making to directly control their own devices or resources instead of depending on third parties.

2. *Smart Contract Deployment*: A smart contract that manages the federated delegation relationship and capability tokens must be developed and deployed on the blockchain network by the policy owner. In our framework, the cloud acts as the data and policy owner which can deploy smart contracts encapsulating delegation and CapAC tokens. After a smart contract has been deployed successfully on the blockchain network, it becomes visible to the whole network owing to the transparency and publicity properties of the blockchain, which means that all participants in the blockchain network can access transactions and smart contracts recorded in chain data. Thanks to cryptographic and security mechanisms provided by the blockchain network, the smart contract can secure any algorithmically specifiable protocols and relationships from malicious interference by third parties under trustless network environment. After synchronizing the blockchain data, all nodes can access all transactions and recent states of each smart contract by referring to local chain data. Each node interacts with the smart contract through the provided contract address and the Remote Procedure Call (RPC) interface.
3. *Federated Delegation*: The PDC service at the cloud server is responsible for delegation policy definition and access right authorization enforcement. To reduce the overhead of the centralized cloud server and meet the scalability and heterogeneity requirements in each IoT domain, the domain coordinator delegates part of the policy decision making tasks and carries out domain specified authorization rules based on domain-specified policies. After receiving a delegation request from a coordinator candidate and executing a policy decision making task to delegate permissions to the coordinator, the PDC service on the cloud launches a transaction to issue a delegation certificate to the smart contract. Finally, the federated delegation relationship is established between the cloud server and the coordinator, and profile and policy data synchronization between the cloud and coordinator is periodically carried out to ensure data consistency on both sides.
4. *Capability Authorization*: To successfully access services or resources at service providers, an entity initially sends an access right request to the domain coordinator to get a capability token. Given the registered entity information established in the profile database, a delegated policy decision making module running on the domain coordinator evaluates the access request by enforcing the delegated authorization policies. If the access request is granted, the domain coordinator issues the capability token encoding the access right and then launches a transaction to update the token data in the smart contract. After the transaction has been approved and recorded in a new block, the domain coordinator notifies the entity with a smart contract address for the querying token data. Otherwise, the access right request is rejected.
5. *Access Right Validation*: The authorization validation process is performed at the object that works as the local service provider after receiving a service request from the subject. Through regularly synchronizing the local chain data with the blockchain network, a service provider just simply checks the current state of the contract in the local chain to get a capability token associated with the entity's address. To determine the capability token validation and access authorization process result, if the access right policies and conditional constraints are satisfied, the service provider grants the access request and offers services to the requester. Otherwise, the service request is denied.

To enable a scalable, distributed and fine-grained access control solution to IoT networks, the proposed BlendCAC is focused on three issues: identity-based capability management, access right authorization and privilege mechanism delegation.

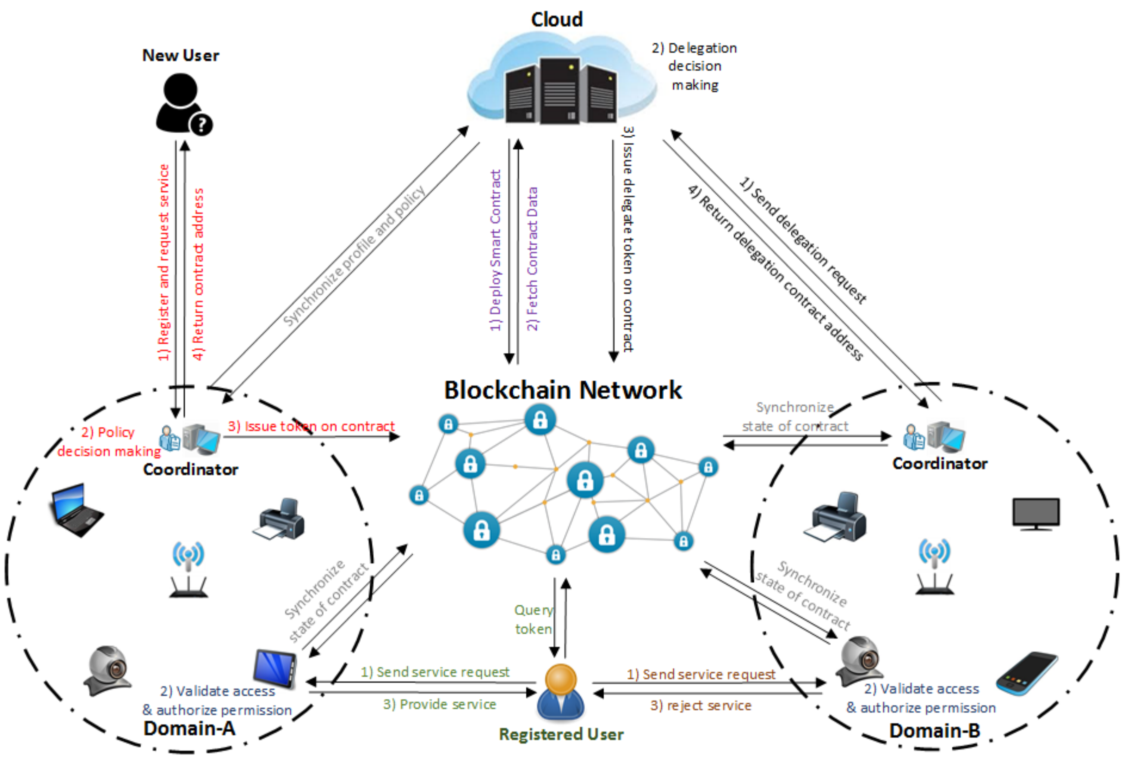

Cloud
2) Delegation decision making
New User
1) Register and request service
4) Return contract address
Synchronize profile and policy
1) Deploy Smart Contract
2) Fetch Contract Data
3) Issue delegate token on contract
1) Send delegation request
4) Return delegation contract address
Blockchain Network
2) Policy decision making
3) Issue token on contract
Coordinator
Synchronize state of contract
Coordinator
Query token
Synchronize state of contract
Synchronize state of contract
1) Send service request
3) Provide service
1) Send service request
3) reject service
2) Validate access & authorize permission
2) Validate access & authorize permission
Domain-A
Registered User
Domain-B

Figure 4. System architecture of BlendCAC.

4.2. Capability Token Structure

In the BlendCAC system, the entities are categorized as subjects and objects. *Subjects* are defined as entities who request services from the service providers, while *objects* are referred to entities who offer the resources or services. Entities could be either human beings or smart devices. In the profile database, all registered entities are associated with a globally unique Virtual Identity (VID), which is used as the prime key for identifying an entity's profile information. As each entity has at least one main account indexed by its address in the blockchain network, the blockchain account address is used to represent the VID for profiling register entities.

In general, the capability specifies which subject can access the resources of a target object by associating the subject, object, actions and condition constraints. The identity-based capability structure is defined as a hash table which is represented as

$$ICap = f(VID_S) \rightarrow \{VID_O, OP, C\}, \quad (1)$$

where the parameters are as follows:

- f: a one-way hash mapping function to establish relationship between the subject and authorized internal capability set;
- VID_S: the virtual ID of a subject that requests an access to a service or resource;
- VID_O: the virtual ID of an object that provides a service or resource;
- OP: a set of authorized operations, e.g., read, write and execute; and
- C: a set of context awareness information, such as time or location.

In the BlendCAC system, the elements in an OP set can be represented as actions, such as $\{read\}$, $\{write\}$, $\{read; write\}$, or $\{NULL\}$. If $OP = \{NULL\}$, the operation conducted on the resource is not allowed. C is defined as a context constraint set, like $C = \{C1, C2\}$ or $C = \{NULL\}$. If $C = \{NULL\}$, no context constraint is considered in the access right validation process.

4.3. Delegation Certificate Structure

The *Identity-Based Delegation Certificate* (IDC) is, essentially, a special capability token which specifies the delegation relationship. The structure of IDC is represented as

$$IDC = f(VID_S) \rightarrow \{VID_P, \{VID_C\}, D, W, DAR\}, \tag{2}$$

where the parameters are as follows:

- f: a one-way hash mapping function to establish the relationship between a subject and a delegated permission set;
- VID_S: the Virtual ID of a subject who is the owner of the delegation token;
- VID_P: the Virtual ID of a parent subject that delegates the token to VID_S;
- $\{VID_C\}$: a set of Virtual IDs of child subjects that records the delegated nodes;
- D: a natural number that indicates the current depth in the delegation tree;
- W: a natural number that defines the maximum delegation width to limit delegable child nodes in $\{VID_C\}$; and
- DAR: a set of delegated permissions for actions, e.g., authorize capability token.

In order to manage delegation relationships between $IDCs$, a hierarchical data structure, called the *Identity-based Delegation Tree* (IDT) is defined as

$$IDT = f(VID_S) \rightarrow \{MD, IDC\}, \tag{3}$$

where the parameters are as follows:

- f: a one-way hash mapping function to establish a relationship between a subject and a delegation tree;
- VID_S: the Virtual ID of a subject who is the owner of the delegation tree;
- MD: a natural number that defines the maximum delegation depth; and
- IDC: a delegation certificate that indicates the root node of the delegation tree.

4.4. Federated Delegation Mechanism

Through encapsulating a delegation certificate structure as a smart contract and deploying it on the blockchain network, the delegation mechanism can be enforced across different security domains in a federated network environment. In the BlendCAC system, the delegator, the delegation authority center (DAC) and the identity management are all implemented as service applications on the cloud server, while the delegatee is deployed on the coordinator in each network domain. The DAC is responsible for identity authentication and delegation authorization service. Prior to the delegation process, the delegator and delegatee should have finished the identity registration process. Figure 5 illustrates the delegation process in our proposed BlendCAC system. The involved work flow in delegation process is as follows:

- *Request Authentication*: The delegatee sends a delegation request to the delegator to ask for the IDC. On receiving the request from the delegator, the DAC verifies the identity of delegatee by referring to the identity management service. If the delegatee's identity is valid, the identity management service returns a Virtual ID (VID) of the delegatee to the DAC. Then, the DAC sends back the authentication result to the delegator. Otherwise, the DAC rejects the delegation request by returning a failure notification.
- *Delegation Authorization*: After receiving the verification result from the DAC, the delegator is capable of assigning delegable permissions to trusted delegatees given pre-defined delegation policies. In our proposal, the federated delegation mechanism is implemented by modifying the delegation certificate, IDC, to assign the delegated permissions or to change the delegation relationship. As the smart contract has received an updated DC transaction from the delegator,

it checks the delegator's IDC to validate the delegation right. If the delegation status can match the $can\_Delegate$ relationship, the delegator can delegate a subset of his/her delegated permissions to the delegatee entity by modifying delegatee's IDC and appending the delegatee's address to the delegator's child node set, $\{VID_C\}$, to set up the delegation relationship. Otherwise, the capability delegation request is rejected.

- *Delegation Revocation*: The delegation revocation considers two scenarios: delegated permission P revocation and delegation certificate IDC revocation. According to the revocation mechanism defined in FCDM, only entities who meet the can_Revoke relationship can carry out revocation operations over a smart contract. In the delegated permission revocation process, the delegator can nullify part of the assigned permissions by simply removing access right elements from DAR in the delegatee's IDC. In the case of IDC revocation, through $cascading$, all subsequent delegate relationships are removed from the DP. By starting from the delegator and destructing the delegatee's IDC, the delegator can tear down all delegation relationships that are associated with delegatee, including those $IDCs$ assigned by the delegatee.

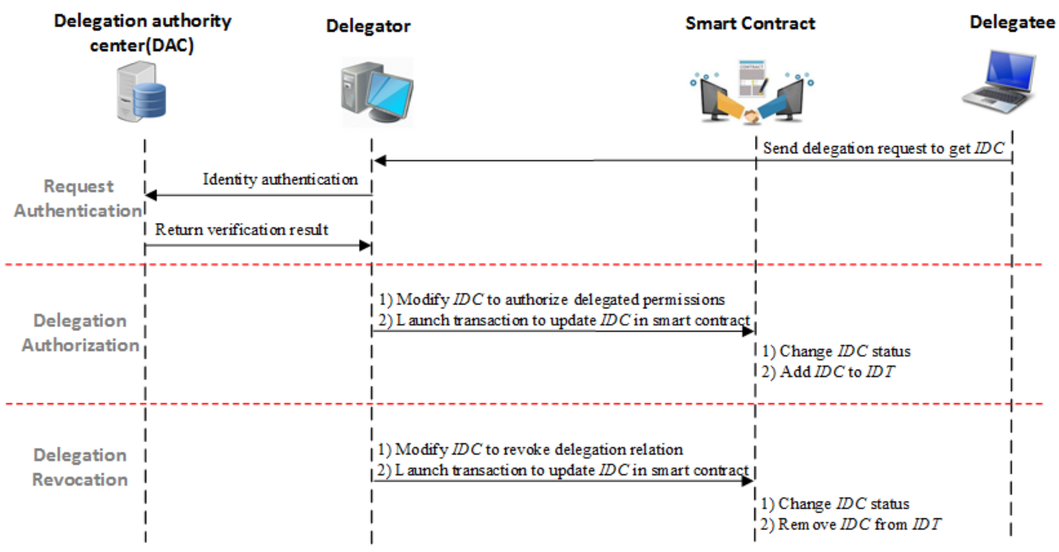

Delegation authority center(DAC)
Delegator
Smart Contract
Delegatee
Send delegation request to get IDC
Identity authentication
Request Authentication
Return verification result
Delegation Authorization
1) Modify IDC to authorize delegated permissions
2) Launch transaction to update IDC in smart contract
1) Change IDC status
2) Add IDC to IDT
Delegation Revocation
1) Modify IDC to revoke delegation relation
2) Launch transaction to update IDC in smart contract
1) Change IDC status
2) Remove IDC from IDT

Figure 5. Delegation process in the BlendCAC system.

4.5. Capability-Based Access Right Authorization

The capability token structure and the related operations are transcoded to a smart contract and deployed on the blockchain network, while the access right authorization is implemented as a policy-based decision making service running on the cloud or delegated domain coordinator. As shown in Figure 6, a comprehensive Capability-Based Access Right Authorization procedure consists of four steps: capability generation, access right validation, capability delegation and revocation.

1. *Capability Generation*: As one type of meta data that represents the access rights, the capability, $ICap$, can be generated by associating a VID with an AR; thus, the $ICap$ has the identified property to prevent forgery. After receiving an access request from a user, the domain coordinator generates a capability token based on the delegated access right authorization policy, and launches transactions to save a new token data to a smart contract. A large number of $ICap$'s are grouped into the capability pools on the smart contract which can be proofed and synchronized among the nodes across the blockchain network.
2. *Access Right Validation*: After receiving the service request from a subject, the service provider first fetches the capability token from the smart contract using the subject's address and then makes decisions on whether or not to grant an access to the service according to the local access control

policy. Implementing access right validation at the local service provider allows smart objects to be involved in the AC decision making tasks, which provides a flexible and fine-grained AC service in IoT networks.

3. *Capability Revocation*: The capability revocation considers two scenarios: partial access right revocation and $ICap$ revocation. In our proposal, only the entities with delegated capability management permissions are allowed to perform revocation operations on capability tokenized smart contracts. In the partial access right revocation process, the delegated entities can remove part of the entries from AR to revoke the selected access rights. In cases of $ICap$ revocation, through directly clearing the AR in $ICap$, the whole capability token becomes unavailable to all associated entities.

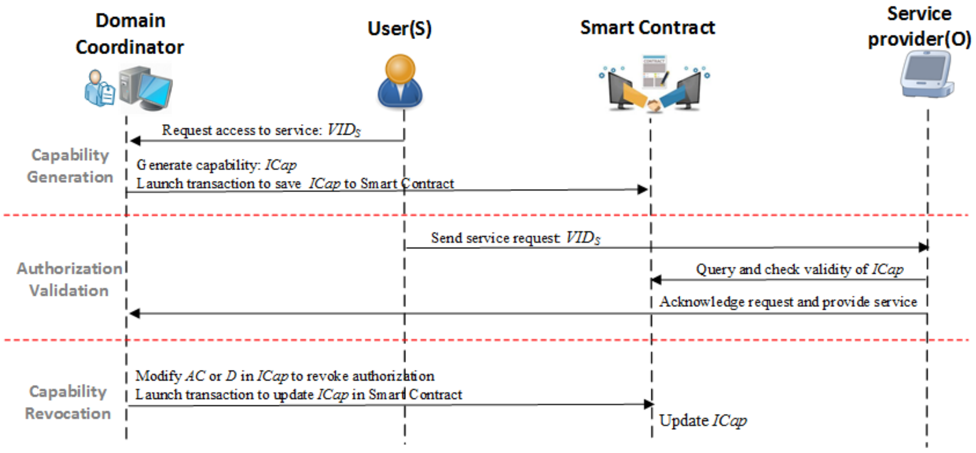

Domain Coordinator
User(S)
Smart Contract
Service provider(O)
Request access to service: VID_S
Capability Generation
Generate capability: $ICap$
Launch transaction to save $ICap$ to Smart Contract
Send service request: VID_S
Authorization Validation
Query and check validity of $ICap$
Acknowledge request and provide service
Modify AC or D in $ICap$ to revoke authorization
Launch transaction to update $ICap$ in Smart Contract
Capability Revocation
Update $ICap$

Figure 6. Flowchart of the Capability-based Access Right Authorization procedure.

5. Prototype Design

A proof of concept prototype system was implemented on a real private Ethereum blockchain network environment. Compared with other open blockchain platforms, like Bitcoin and Hyperledger, Ethereum has a more matured ecosystem and is designed to be more adaptable and flexible for the development of a smart contract and business logic [39].

5.1. Delegation Certificate and Capability Token Structure

The proposed BlendCAC model was transcoded to a smart contract using Solidity [40], which is a contract-oriented, high-level language for implementing smart contracts. With Truffle [41], which is a world class development environment testing framework and asset pipeline for Ethereum, contract source codes are compiled to Ethereum Virtual Machine (EVM) bytecode and migrated to the Ethereum blockchain network.

To implement a BlendCAC system on IoT devices without introducing significant overhead over network communication and computation, the delegation certificate and capability token data structure is represented in JSON [42] format. Compared to XML-based language for access control, like XACML and SAML, JSON is lightweight and suitable for constrained platforms.

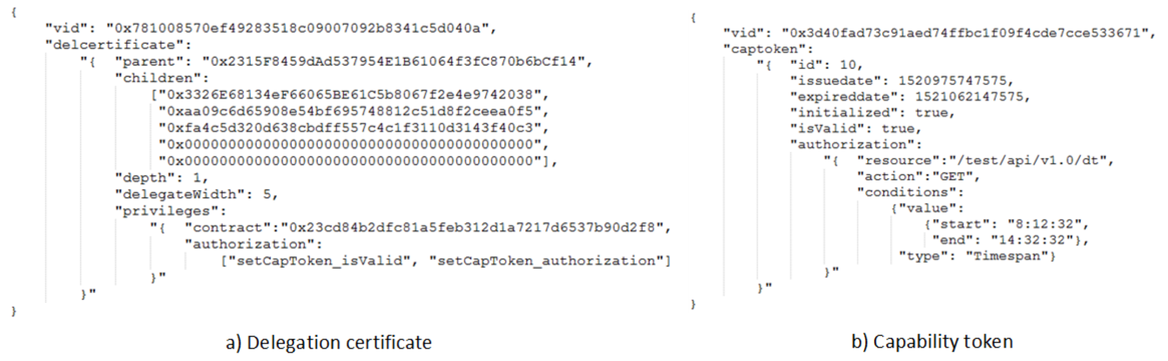
Figure 7a demonstrates a delegation certificate example, and the data fields in the data structure are described as follows:

- *vid* : a 20-byte value to represent the address of certificate owner in the blockchain network;
- *parent* : a 20-byte value to represent the address of the parent entity in the blockchain network;
- *children*: a queue to record all address of delegated entities;
- *depth*: a natural number to indicate the depth of the current delegation certificate in the delegation tree;

- *delegateWidth*: a natural number to constrain horizontal delegation times;
- *privileges*: a set of delegated access rights that the delegator has assigned to the delegatee, including
  - *contract*: a 20-byte value to indicate the address of the delegated smart contract; and
  - *authorization*: a set of delegated functions for which the operations are granted.

Figure 7b presents a capability token data example used in the AC system. A brief description of each field is provided as follows:

- *vid* : a 20-byte value to represent the address of the capability owner in the blockchain network;
- *id*: the auto-incremented prime key to identify a capability token;
- *initialized*: a bool flag used for checking the token's initialized status;
- *isValid*: a bool flag signifying the enabled status to show whether a token is valid or not;
- *issuedate*: to identify the date and time when the token was issued;
- *expireddate*: the date and time when the token becomes expired;
- *authorization*: a set of access right rules that the issuer has granted to the subject, including
  - *action*: to identify a specific granted operation over a resource;
  - *resource*: the resource in the service provider for which the operation is granted. In this case, the resource is defined as granted REST-ful API; and
  - *conditions*: a set of conditions which must be fulfilled locally on the service provider to grant the corresponding operation.



```
{
    "vid": "0x781008570ef49283518c09007092b8341c5d040a",
    "delcertificate":
        "{  "parent": "0x2315F8459dAd537954E1B61064f3fC870b6bCf14",
            "children":
                ["0x3326E68134eF66065BE61C5b8067f2e4e9742038",
                 "0xaa09c6d65908e54bf695748812c51d8f2ceea0f5",
                 "0xfa4c5d320d638cbdff557c4c1f3110d3143f40c3",
                 "0x0000000000000000000000000000000000000000",
                 "0x0000000000000000000000000000000000000000"],
            "depth": 1,
            "delegateWidth": 5,
            "privileges":
                "{  "contract":"0x23cd84b2dfc81a5feb312d1a7217d6537b90d2f8",
                    "authorization":
                        ["setCapToken_isValid", "setCapToken_authorization"]
                }"
        }"
}
```

a) Delegation certificate

```
{
    "vid": "0x3d40fad73c91aed74ffbc1f09f4cde7cce533671",
    "captoken":
        "{  "id": 10,
            "issuedate": 1520975747575,
            "expireddate": 1521062147575,
            "initialized": true,
            "isValid": true,
            "authorization":
                "{  "resource":"/test/api/v1.0/dt",
                    "action":"GET",
                    "conditions":
                        {"value":
                            {"start": "8:12:32",
                             "end": "14:32:32"},
                        "type": "Timespan"}
                }"
        }"
}
```

b) Capability token

Figure 7. Token data structure in BlendCAC.

After a smart contract has been successfully deployed on the blockchain network, all nodes in the network can interact with the smart contract using the address of the contract and the Application Binary Interface (ABI) definition, which describes the available functions of a contract.

5.2. Federated Delegation Policy Service

The federated delegation authorization and revocation mechanisms are implemented as service functions which are executed by the PDC application to enforce a delegation policy. Algorithm 1 illustrates the delegation authorization process. The *authorize_delegate*() function receives the inputs of *delegateeAddr*, *delegate_AR* and *deleWidth* and returns the delegate authorization result. The delegable validation is from Line 4 to Line 32, and delegation authorization is from Line 33 to Line 35. The description of the delegate revocation algorithm is illustrated in Algorithm 2. The delegate revocation can be carried out by two entities: the supervisor, which is the root node of delegation tree, or any ancestor in the delegate path. As illustrated from Line 3 to Line 8, the supervisor can revoke any assigned delegation certificate in the delegate tree. Otherwise, only the ancestors in the delegate path can revoke the delegatee's IDC, and the process is described from Line 10 to Line 16.

Algorithm 1 Authorize Delegation.

```
Require: $delegateeAddr, delegate\_AR, deleWidth$
 1: $BaseAddr = web3.eth.coinbase$
 2: $json\_delegator = get\_delegateToken(BaseAddr)$
 3: $json\_delegatee = get\_delegateToken(delegateeAddr)$
 4: if $delegateeAddr{==}json\_delegator['root\_node']$ then
 5:     $returnFalse$
 6: end if
 7: if $json\_delegatee['parent']! = address\_zero$ then
 8:     $returnFalse$
 9: end if
10: if $get\_childrenCount(json\_delegator['children']) >= json\_delegator['max\_width']$ then
11:     $returnFalse$
12: end if
13: if $json\_delegator['depth'] >= json\_delegator['max\_depth']$ then
14:     $returnFalse$
15: end if
16: if $BaseAddr == json\_delegator['root\_node']$ then
17:     for $ar$ in $delegate\_ar$ do
18:         if $ar$ in $full\_delegateAR$ then
19:             $authorize\_ar.append(ar)$
20:         end if
21:     end for
22: else
23:     $delgator\_ar = json\_delegator['privilege']$
24:     for $ar$ in $delgator\_ar$ do
25:         if $ar$ in $delgatorar$ then
26:             $authorize\_ar.append(ar)$
27:         end if
28:     end for
29: end if
30: if $delgator\_ar$ == Empty then
31:     $returnFalse$
32: end if
33: $addDelToken(delegateeAddr)$
34: $setDelegateWidth(delegateeAddr, deleWidth)$
35: $setPrivilege(delegateeAddr, authorize\_ar)$
36: $returnTrue$
```

Algorithm 2 Revoke Delegation.

```
Require: $delegateeAddr$
 1: $BaseAddr = web3.eth.coinbase$
 2: $json\_delegatee = get\_delegateToken(delegateeAddr)$
 3: if $BaseAddr == json\_delegator['root\_node']$ then
 4:     if $delegateeAddr == BaseAddr$ then
 5:         $returnFalse$
 6:     else
 7:         $revokeDelToken(delegateeAddr)$
 8:     end if
 9: else
10:     if $json_delegatee['parent'] == address\_zero$ then
11:         $returnFalse$
12:     end if
13:     if $(BaseAddr! = delegateeAddr)$ and $(isAncestor(BaseAddr, delegateeAddr) == False)$ then
14:         $returnFalse$
15:     end if
16:     $revokeDelToken(delegateeAddr)$
17: end if
18: $returnTrue$
```

5.3. Access Authorization Service

The access authorization and validation policy is enforced as a web service application based on the Flask framework [43] using Python. The Flask is a micro-framework for Python based on Werkzeug, Jinja 2 and good intentions. The lightweight and extensible micro-architectures make the Flask a preferable web solution for resource constrained IoT devices.

The web service application in BlendCAC system consists of two parts: the client and the server. The client performs an operation on a resource by sending a data request to the server, while the server provides the REST-ful API for the client to obtain data or perform an operation on a resource on the server side. A Capability-based Access Control scheme is enforced on the server side by performing access right validation on the service provider. The access right validation process is launched after a request containing the client's identity is received on server. Figure 8 shows a block diagram with the steps taken to process an authorization decision.

1. *Check cached token data*: After receiving a service request from a user, the service provider firstly checks whether or not the token data associated with the user's address exists in the local database. If the search for token data fails, the service provider can fetch the token data from the smart contract by calling an exposed contract method and saving the token data to the local database. Otherwise, the token data is directly reloaded from the local token database for further validation processes. The service provider regularly synchronizes the local database with the smart contract to ensure token data consistency.
2. *Verify token status*: As a capability token has been converted to JSON data, the first step of token validation is checking the current capability status of the token, such as the initialized, isValid, issuedate, and expireddate functions. If any status of a token is not valid, the authorization process stops and a deny access request is sent back to the subject.
3. *Check whether access is granted or not*: The service provider goes through all access rules in the access right set to guarantee that the request operation is permitted. The process checks whether or not the REST-ful method used by the requester matches the authorized action of current access rules and that the value of resource field is the same as the Request-Uniform Resource Identifiers (URI) option used by the requester. If the current access rule verification fails, the process skips to the next access rule for evaluation. If none of the access rules successfully pass the verification process, the authorization validation process stops and the access request is denied.
4. *Verify the conditions*: Even though the action on a target resource is permitted after the access validation, it is necessary to evaluate the context-awareness constraints on the local device by verifying whether or not the specified conditions in the token are satisfied. The condition verification process goes through all constraints in the condition set to find the matched ones. If no condition is fulfilled in the given local environment, the access right validation process stops and the access request is denied.

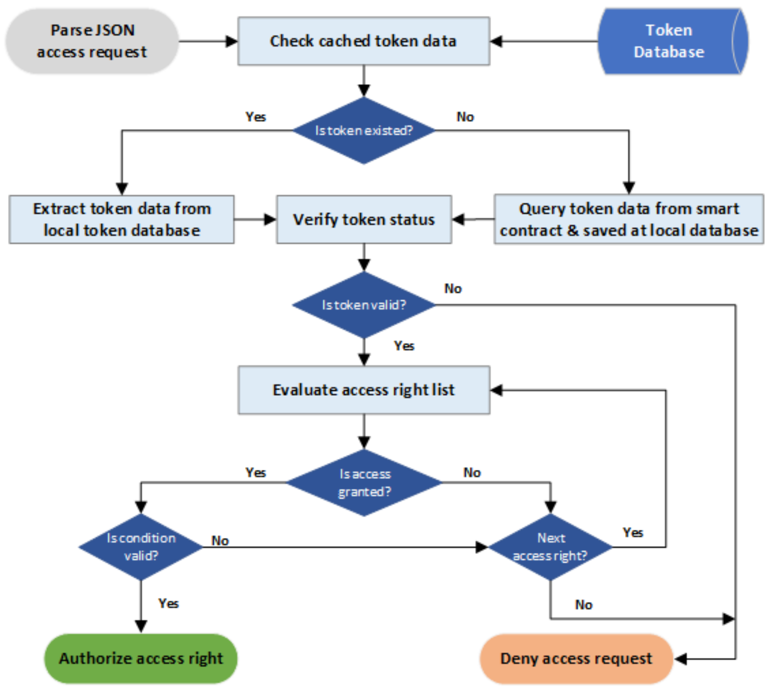


Figure 8. BlendCAC access authorization process.

6. Experiment and Evaluation

In order to evaluate the performance and the overhead of our BlendCAC scheme, two benchmark models, RBAC and ABAC, were also transcoded to separate smart contracts and enforced on the experimental web service system. All transcoded access control models had similar smart contract data structures, except for the type of authorization representation. In the RBAC-based smart contract, authorization was defined as the approach to bridge the relationship between the user and permission, and the ABAC-based smart contract used users' attributes as a representative format for authorization. Both the RBAC and ABAC need a local database, either to maintain the user–role–permission or to manage the attribute-permission policy for authorization validation process. The profiles and policy rule management were developed using an embedded SQL database engine, called SQLite [44]. The lower memory and computation cost make the SQLite an ideal database solution for resource constrained systems like Raspberry Pi. All documents and source code are available on the BlendCAC project repository [45].

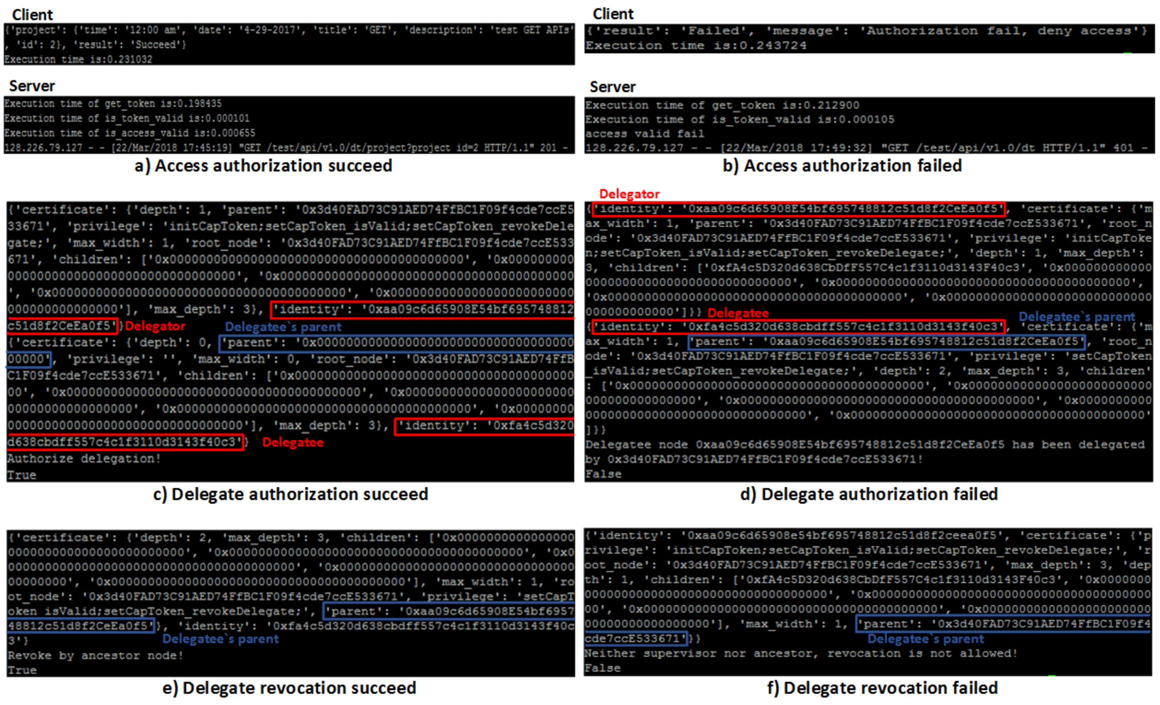
6.1. Testbed Setup

Mining tasks are performed on systems with stronger computing power, like a laptop or a desktop. In this experiment, two miners were deployed on a laptop, of which the configuration was as follows: the processor was 2.3 GHz Intel Core i7 (8 cores), the RAM memory was 16 GB and the operating system was Ubuntu 16.04. Another four miners were distributed to four desktops which were empowered with the Ubuntu 16.04 OS, 3 GHz Intel Core TM (2 cores) processor and 4 GB memory. In our system, the laptop acted as a cloud computing server, while all desktops worked as fog computing nodes to take the role of domain coordinators. Each miner used two CPU cores for mining. The edge computing nodes were two Raspberry PI 3 Model Bs with the following configuration: 1.2 GHz 64-bit quad-core ARMv8 CPU, 1GB LPDDR2-900 SDRAM memory and a Raspbian operation system based on the Linux kernel. Unfortunately, the Raspberry PI is not powerful enough to function as a miner, so all Raspberry Pi devices worked as nodes to join the private blockchain without mining. All devices used Go-Ethereum [46] as the client application to work on the blockchain network.

6.2. Experimental Results

To verify the effectiveness of the BlendCAC approach against unauthorized access requests, a service access experiment was carried out on a real network environment. In the test scenario, one Raspberry Pi 3 device worked as the client and another worked as the service provider. Given the access authorization process shown in Figure 9a,b, when any of the steps in the authorization procedure failed, the running process immediately aborts instead of continuing to step through all the authorization stages. As shown by Figure 9b, the server stopped the authorization process due to the failure to verify the granted actions or the conditional constraints specified in the access right list. Consequently, the client node received a deny access notification from the server and could not read the requested data. In contrast, Figure 9a presents a successful data request example, in which the whole authorization process was accomplished at the server side without any error, allowing the client to successfully retrieve the data from the service provider.

The delegate authorization and revocation results are shown in Figure 9c,f. Figure 9c shows that the delegator '0xaa09c6d65908e54bf695748812c51d8f2ceea0f5' successfully delegated a subset of its delegated permissions to the delegatee '0xfa4c5d320d638cbdff557c4c1f3110d3143f40c3' whose parent was empty. Figure 9d shows a failed delegation scenario caused by assigning permissions to a delegated entity. In the revocation process, only the supervisor or ancestor of the delegatee is allowed to call back the delegated permissions. Figure 9e shows that the delegatee's parent '0xaa09c6d65908e54bf695748812c51d8f2ceea0f5' was able to successfully revoke the delegation relationship. Otherwise, the delegation revocation request which was from neither the delegatee's parent '0x3d40fad73c91aed74ffbc1f09f4cde7cce533671' nor any ancestor in delegate path was denied and a failed result is shown in Figure 9f.



Client
```
{'project': {'time': '12:00 am', 'date': '4-29-2017', 'title': 'GET', 'description': 'test GET APIs'
, 'id': 2}, 'result': 'Succeed'}
Execution time is:0.231032
```
Server
```
Execution time of get_token is:0.198435
Execution time of is_token_valid is:0.000101
Execution time of is_access_valid is:0.000655
128.226.79.127 - - [22/Mar/2018 17:45:19] "GET /test/api/v1.0/dt/project?project id=2 HTTP/1.1" 201 -
```
a) Access authorization succeed

Client
```
{'result': 'Failed', 'message': 'Authorization fail, deny access'}
Execution time is:0.243724
```
Server
```
Execution time of get_token is:0.212900
Execution time of is_token_valid is:0.000105
access valid fail
128.226.79.127 - - [22/Mar/2018 17:49:32] "GET /test/api/v1.0/dt HTTP/1.1" 401 -
```
b) Access authorization failed

```
{'certificate': {'depth': 1, 'parent': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE5
33671', 'privilege': 'initCapToken;setCapToken_isValid;setCapToken_revokeDele
gate;', 'max_width': 1, 'root_node': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533
671', 'children': ['0x0000000000000000000000000000000000000000', '0x000000000
0000000000000000000000000000000', '0x000000000000000000000000000000000000000
', '0x0000000000000000000000000000000000000000', '0x0000000000000000000000000
000000000000000'], 'max_depth': 3}, 'identity': '0xaa09c6d65908E54bf695748812
c51d8f2CeEa0f5'}
{'certificate': {'depth': 0, 'parent': '0x00000000000000000000000000000000000
00000', 'privilege': '', 'max_width': 0, 'root_node': '0x3d40FAD73C91AED74FfB
C1F09f4cde7ccE533671', 'children': ['0x0000000000000000000000000000000000000
00', '0x0000000000000000000000000000000000000000', '0x00000000000000000000000
00000000000000000', '0x0000000000000000000000000000000000000000', '0x00000000
00000000000000000000000000000000'], 'max_depth': 3}, 'identity': '0xfa4c5d320
d638cbdff557c4c1f3110d3143f40c3'}
Authorize delegation!
True
```
c) Delegate authorization succeed

```
{'identity': '0xaa09c6d65908E54bf695748812c51d8f2CeEa0f5', 'certificate': {'m
ax_width': 1, 'parent': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671', 'root_n
ode': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671', 'privilege': 'initCapToke
n;setCapToken_isValid;setCapToken_revokeDelegate;', 'depth': 1, 'max_depth':
3, 'children': ['0xfA4c5D320d638CbDfF557C4c1f3110d3143F40c3', '0x00000000000
00000000000000000000000000000', '0x0000000000000000000000000000000000000000',
'0x0000000000000000000000000000000000000000', '0x0000000000000000000000000000
000000000000']}}
{'identity': '0xfa4c5d320d638cbdff557c4c1f3110d3143f40c3', 'certificate': {'m
ax_width': 1, 'parent': '0xaa09c6d65908E54bf695748812c51d8f2CeEa0f5', 'root_n
ode': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671', 'privilege': 'setCapToken
_isValid;setCapToken_revokeDelegate;', 'depth': 2, 'max_depth': 3, 'children'
: ['0x0000000000000000000000000000000000000000', '0x0000000000000000000000000
000000000000000', '0x0000000000000000000000000000000000000000', '0x0000000000
000000000000000000000000000000', '0x0000000000000000000000000000000000000000'
]}}
Delegatee node 0xaa09c6d65908E54bf695748812c51d8f2CeEa0f5 has been delegated
by 0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671!
False
```
d) Delegate authorization failed

```
{'certificate': {'depth': 2, 'max_depth': 3, 'children': ['0x0000000000000000
000000000000000000000000', '0x0000000000000000000000000000000000000000', '0x0
000000000000000000000000000000000000000', '0x00000000000000000000000000000000
00000000', '0x0000000000000000000000000000000000000000'], 'max_width': 1, 'ro
ot_node': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671', 'privilege': 'setCapT
oken_isValid;setCapToken_revokeDelegate;', 'parent': '0xaa09c6d65908E54bf6957
48812c51d8f2CeEa0f5'}, 'identity': '0xfa4c5d320d638cbdff557c4c1f3110d3143f40c
3'}
Revoke by ancestor node!
True
```
e) Delegate revocation succeed

```
{'identity': '0xaa09c6d65908e54bf695748812c51d8f2ceea0f5', 'certificate': {'p
rivilege': 'initCapToken;setCapToken_isValid;setCapToken_revokeDelegate;', 'r
oot_node': '0x3d40FAD73C91AED74FfBC1F09f4cde7ccE533671', 'max_depth': 3, 'dep
th': 1, 'children': ['0xfA4c5D320d638CbDfF557C4c1f3110d3143F40c3', '0x0000000
000000000000000000000000000000000', '0x00000000000000000000000000000000000000
00', '0x0000000000000000000000000000000000000000', '0x00000000000000000000000
00000000000000000'], 'max_width': 1, 'parent': '0x3d40FAD73C91AED74FfBC1F09f4
cde7ccE533671'}}
Neither supervisor nor ancestor, revocation is not allowed!
False
```
f) Delegate revocation failed

Figure 9. Experimental results of the BlendCAC system.

6.3. Performance Evaluation

In the test scenario, two Raspberry Pi 3 devices were adopted to play the roles of the client and the service provider respectively. To measure the general cost incurred by the proposed BlendCAC scheme both on the IoT devices' processing time and the network communication delay, 50 test runs were conducted based on the proposed test scenario, in which the client sent a data query request to the server for access permission. This test scenario was based on an assumption that the subject had a valid capability token when it performed the action. Therefore, all steps of authorization validation had to be processed on the server side so that the maximum latency value was computed.

6.3.1. Computational Overhead

According to the results shown in Figure 10, the average total delay time required by the BlendCAC operation of retrieving data from the client to server is 243 ms, which is almost the same as RBAC or ABAC. The total delay includes the round trip time (RTT), time for querying the capability data from the smart contract, time for parsing JSON data from the request, and time for access right validation. The token processing task is mainly responsible for fetching the token data from the smart contract and introduces the highest workload among the authorization operation stages. Owing to the fact that encapsulating the user–role relationship in the smart contract requires less data than the capability or attributes does, the RBAC incurred less computational cost than what BlendCAC and ABAC did in token processing stage. As the most computing intensive stage, the execution time of token processing is about 210 ms, which accounts for almost 86% of the entire process time.

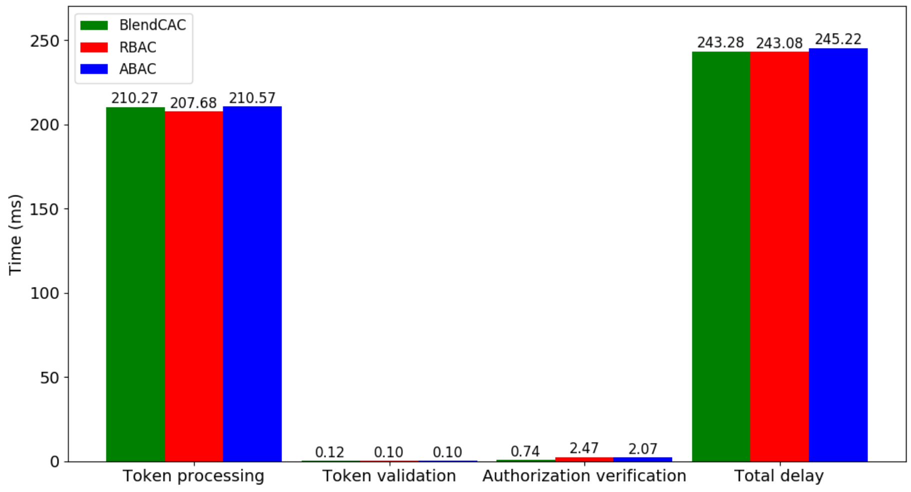


Figure 10. Computation time for each stage in BlendCAC.

The entire authorization process is divided into two steps—token validation and authorization verification—where the average time of the authorization process is about 0.86 ms (0.12 ms + 0.74 ms). Compared with the token validation process, in which only simply checks the token valid status, the authorization verification process requires more computational power to enforce the local access control policies. Although the similarity in token data structure allows all the three access control models to have almost the same time in the token validation stage, the BlendCAC outperforms the RBAC and ABAC in authorization verification. Since both the RBAC and ABAC need a database to either manage the user–role–permission relationship or maintain attribute-permission rules, this inevitably consumes time to search rules in the database. In our experimental study, the RBAC (2.47 ms) and ABAC (2.07 ms) had much higher processing times than BlendCAC (0.74 ms).

6.3.2. Communication Overhead

Owing to the high overhead introduced by querying token data from the smart contract in the token processing stage, a token data caching solution was introduced in the BlendCAC system to reduce network latency. When the client sends a service request to the server, the service side extracts cached token data from the local storage to validate authorization. The service providers regularly updates cached token data by checking the smart contract status. The token synchronization time is consistent with the block generation time, which is about 15 s in the Ethereum blockchain network.

Simulating a regular service request allowed us to measure how long it takes for the client to send a request and retrieve the data from the server.

Figure 11 shows the overall network latency incurred and compares the execution time of the BlendCAC with RBAC, ABAC and a benchmark without any access control enforcement. At the beginning, a long delay was observed in the first service request scenario, during which the service provider communicated with the smart contract and cached the token data. However, through processing the local cached token data for authorization validation, the network latency decreased quickly and became stable at a low level during the subsequent service requests. The benchmark without access control enforcement took an average of 31 ms for fetching requested data, whereas the BlendCAC consumed, on average, 36 ms. This means that the proposed BlendCAC scheme only introduces about 5 ms extra latency. The overhead in terms of delay is trivial. As shown by Figure 11, the BlendCAC also had a lower amount of latency than RBAC and ABAC in most periods of time. In addition, unlike RBAC and ABAC, which rely on the local policy database as an intermediate to valid access rights, encoding access rights directly in the capability token makes the BlendCAC more scalable and flexible in large-scale IoT networks.

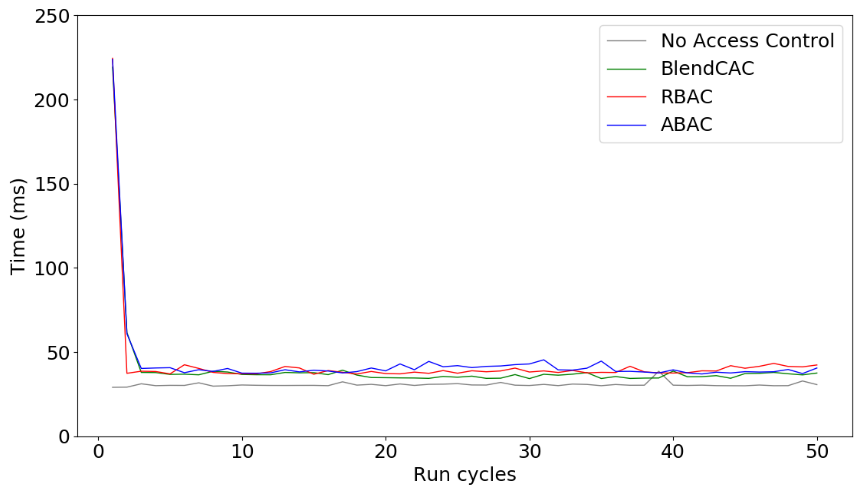


Figure 11. Network latency of BlendCAC.

6.3.3. Processing Overhead

The delegation certificate and capability token can be valid only if related transactions to the smart contract have been approved by miners and recorded to new blocks. The transaction rate is proportional to the block generation time, which refers to the time consumed by miners to verify new blocks. Figure 12 shows the impact of the number of miners on the blockchain network. In each scenario, sixty blocks were appended to the blockchain and the average block generation time was calculated. Initially, only two miners ran the consensus algorithm. As more miners performed the proof of work, the block generation time reduced and finally, became stable. As shown by Figure 12, the optimal number of miners to achieve the minimum block generation time is six in our private blockchain network. As a central part of Ethereum network, gas is used to pay for the computing resources consumed by miners. To evaluate process overhead resulting from the gas cost, 100 transactions that assign delegate certificate and capability were created on blockchain network, and the average gas cost for each transaction is 169,576.15 Wei (in Ethereum, 1 ether = 1.0×10^{18} Wei), which amounts to $1.018 given the current gas price in the public Ethereum market.

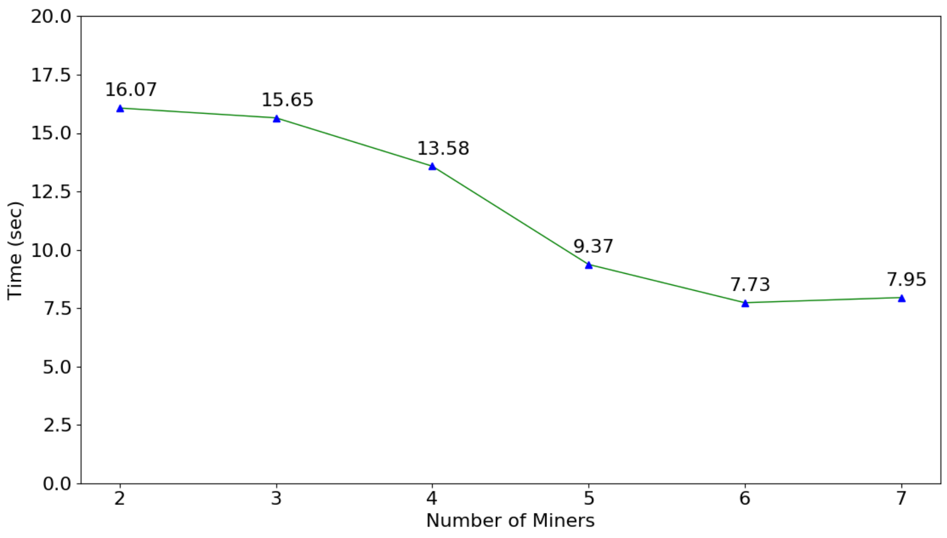


Figure 12. Time consumption of block generation.

As the experimental results show, the proposed BlendCAC scheme introduced a small amount of overhead, both at the network layer and the local device layer. To measure the general network latency of inter-domain communication, HTTP was executed on the same testbed to simulate a regular transaction, like connecting, sending a request and retrieving the reply. Compared with the calculated average network latency, which is about 300 ms, the trade-off in the proposed BlendCAC is acceptable for the network environments by only incurring 5 ms latency (no more than 2%). In addition, the test scenarios are based on Raspberry Pi devices, which belong to a type of simple board computer (SBC) with limited computation power and memory space. It is reasonable to expect a better performance when the BlendCAC scheme is implemented on more powerful smart devices, like smart phones. Although the synchronization of cached token data with the smart contract requires more computational resources, the transactions of querying smart contract status are regularly launched by the service providers in a separate service thread rather than being called in each service request, so that the network overhead over the service request communication is greatly reduced to improve the Quality of Service (QoS) requirement. Although the transaction rate, which is constricted by the block time, introduces latency into the delegation certificate and capability token generation and money is required to pay miners for transaction verification, BlendCAC still has good scalability in the distributed IoT network environment with minimal overheads and time consumption.

6.4. Discussion

The experimental results demonstrate that our proposed BlendCAC strategy is an effective and efficient method for protecting IoT devices from unauthorized access requests. Compared to the centralized AC model, our proposed scheme has the following advantages:

- *Load balance*: The BlendCAC framework takes advantage of a delegation mechanism to distribute the load of the centralized PDC server to separate local domain coordinators, such that the bottleneck effect of PDC is mitigated and the risk of malfunction resulting from centralized system is reduced. Even in the worst case when the PDC crashes for a short period of time, a large number of domain coordinators still work normally on behalf of the PDC to provide services;
- *Decentralized authorization*: By leveraging the blockchain technique, the proposed BlendCAC scheme allows users to control their devices and resources without depending on a third centralized authority to establish the trust relationship with unknown nodes; instead, it can define a domain-specific access authorization policy which is meaningful to distributive, scalable, heterogeneous and dynamic IoT-based applications;

- *Edge computing-driven intelligence*: Thanks to the federated delegation mechanism and blockchain technology, the BlendCAC framework provides a device-driven access control strategy that is suitable for the distributed nature of the IoT environment. Through transferring power and intelligence from the centralized cloud server to the edge of the network, the risk of performance bottleneck and single point of failure are mitigated, and smart things are capable of protecting their own resources and privacy by enforcing a user-defined security mechanism;
- *Fine granularity*: Enforcing access right validation to local service providers empowers smart devices to decide whether or not to grant access to certain services according to local environmental conditions. Fine-grained access control with a lease privilege access principle prevents privilege escalation, even if an attacker steals the capability token;
- *Lightweight*: Compared to XML-based language for access control, such as XACML, JSON is a lightweight technology that is suitable for resource-constrained platforms. Given the experimental results, our JSON based capability token structure introduces a small overhead on the general performance.

Although the proposed BlendCAC mechanism has demonstrated these attractive features, using a blockchain to enforce the AC policy in IoT systems also incurs new challenges in performance and security. The transaction rate is associated with the confirmation time of the blockchain data which depends on the block size and the time interval between the generation of new blocks. Thus, the latency for transaction validation may not be able to meet the requirement in real-time application scenarios. In addition, as the number of transactions increases, the blockchain becomes large. The continuously growing data introduces more overhead on the storage and computing resources of each client, especially for resource-constrained IoT devices. Furthermore, the blockchain is susceptible to majority attack (also known as 51% attacks), in which once an attacker takes over 51% computing power of the network by colluding selfish miners; they are able to control the blockchain and reverse the transactions. Finally, since the blockchain data is open to all nodes joined to the blockchain network, such a property of transparency inevitably brings privacy leakage concerns. More research effort is necessary to improve the trade-off when applying the BlendCAC in practical scenarios.

7. Conclusions

In this paper, we proposed a partially decentralized federated Capability-based Access Control framework that leverages smart contract and blockchain technology, called BlendCAC, to handle the challenges in access control strategies for IoT devices. A concept-proof prototype was built in a physical IoT network environment to verify the feasibility of the proposed BlendCAC. The FCDM model and CapAC policy were transcoded to smart contracts and work on the private Ethereum blockchain network. The desktops and laptops serve as miners to maintain the sanctity of transactions recorded on the blockchain, while Raspberry PI devices act as edge computing nodes to access and provide IoT-based services. Extensive experimental studies were conducted and the results are encouraging. It was validated that the BlendCAC scheme is able to efficiently and effectively enforce access control authorization and validation in a distributed and trustless IoT network. This work demonstrates that our proposed BlendCAC framework is a promising approach to provide a scalable, fine-grained and lightweight access control for IoT networks.

While the reported work has shown significant potential, there is still a long way towards achieving a complete decentralized security solution for IoT edge computing. Deeper insights are expected. Part of our on-going effort is focused on further exploration of blockchain-based access control schemes in real-world applications. Taking the smart surveillance system as a case study, the proposed BlendCAC will be extended to protect network cameras and motion sensors in the novel urban surveillance platform that we recently developed [36,47].

Author Contributions: Conceptualization, R.X. and Y.C.; Methodology, R.X.; Software, R.X.; Validation, R.X., Y.C., E.B. and G.C.; Formal Analysis, R.X.; Investigation, R.X.; Resources, R.X.; Data Curation, R.X.; Writing-Original Draft Preparation, R.X., Y.C.; Writing-Review & Editing, R.X., Y.C., E.B. and G.C.; Visualization, R.X.; Supervision, Y.C.; Project Administration, Y.C.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ABAC	Attribute-based Access Control
AC	Access Control
ACL	Access Control List
ACM	Access Control Matrix
BlendCAC	BLockchain-ENabled Decentralized Federated Capability-based Access Control
CAC/CapAC	Capability-based Access Control
FCDM	Federated Capability-based Delegation Model
IoT	Internet of Things
RBAC	Role-based Access Control
RTT	Round Trip Time
QoS	Quality of Service

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Snidaro, L.; Garcia-Herrera, J.; Llinas, J.; Blasch, E. *Context-Enhanced Information Fusion*; Springer: Berlin, Germany, 2016.
3. Blasch, E.; Kadar, I.; Grewe, L.L.; Brooks, R.; Yu, W.; Kwasinski, A.; Thomopoulos, S.; Salerno, J.; Qi, H. Panel summary of cyber-physical systems (CPS) and Internet of Things (IoT) opportunities with information fusion. In Proceedings of the International Society for Optics and Photonics, Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI, Anaheim, CA, USA, 11–12 April 2017; Volume 10200, p. 102000O.
4. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things security: A survey. *J. Netw. Comput. Appl.* **2017**, *88*, 10–28. [CrossRef]
5. Gusmeroli, S.; Piccione, S.; Rotondi, D. A capability-based security approach to manage access control in the internet of things. *Math. Comput. Model.* **2013**, *58*, 1189–1205. [CrossRef]
6. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 12 July 2018).
7. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. Blockchain technology: Beyond bitcoin. *Appl. Innov.* **2016**, *2*, 6–10.
8. Ouaddah, A.; Mousannif, H.; Elkalam, A.A.; Ouahman, A.A. Access control in the Internet of things: Big challenges and new opportunities. *Comput. Netw.* **2017**, *112*, 237–262. [CrossRef]
9. Gong, L. A secure identity-based capability system. In Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 1–3 May 1989; pp. 56–63.
10. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *Computer* **1996**, *29*, 38–47. [CrossRef]
11. Samarati, P.; de Vimercati, S.C. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 137–196.
12. De Souza, L.M.S.; Spiess, P.; Guinard, D.; Köhler, M.; Karnouskos, S.; Savio, D. Socrades: A web service based shop floor integration infrastructure. In *The Internet of Things*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 50–67.

13. Spiess, P.; Karnouskos, S.; Guinard, D.; Savio, D.; Baecker, O.; De Souza, L.M.S.; Trifa, V. SOA-based integration of the internet of things in enterprise services. In Proceedings of the 2009 IEEE International Conference on Web Services, Los Angeles, CA, USA, 6–10 July 2009; pp. 968–975.
14. Zhang, G.; Tian, J. An extended role based access control model for the Internet of Things. In Proceedings of the 2010 International Conference on Information Networking and Automation (ICINA), Kunming, China, 18–19 October 2010; Volume 1, pp. V1-319–V1-323.
15. Yuan, E.; Tong, J. Attributed based access control (ABAC) for web services. In Proceedings of the 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA, 11–15 July 2005.
16. Smari, W.W.; Clemente, P.; Lalande, J.F. An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system. *Future Gener. Comput. Syst.* **2014**, *31*, 147–168. [CrossRef]
17. Ye, N.; Zhu, Y.; Wang, R.C.; Malekian, R.; Qiao-min, L. An efficient authentication and access control scheme for perception layer of internet of things. *Appl. Math. Inf. Sci.* **2014**, *8*, 1617–1624. [CrossRef]
18. Liu, B.; Chen, Y.; Hadiks, A.; Blasch, E.; Aved, A.; Shen, D.; Chen, G. Information fusion in a cloud computing era: A systems-level perspective. *IEEE Aerosp. Electron. Syst. Mag.* **2014**, *29*, 16–24. [CrossRef]
19. Gusmeroli, S.; Piccione, S.; Rotondi, D. IoT@ Work automation middleware system design and architecture. In Proceedings of the 2012 IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), Krakow, Poland, 17–21 September 2012; pp. 1–8.
20. Anggorojati, B.; Mahalle, P.N.; Prasad, N.R.; Prasad, R. Capability-based access control delegation model on the federated IoT network. In Proceedings of the 2012 15th International Symposium on Wireless Personal Multimedia Communications (WPMC), Taipei, Taiwan, 24–27 September 2012; pp. 604–608.
21. Skinner, G.D. Cyber security management of access controls in digital ecosystems and distributed environments. In Proceedings of the 6th International Conference on Information Technology and Applications (ICITA 2009), Hanoi, Vietnam, 9–12 November 2009; pp. 77–82.
22. Pal, S.; Hitchens, M.; Varadharajan, V. Towards a Secure Access Control Architecture for the Internet of Things. In Proceedings of the 2017 IEEE 42nd Conference on Local Computer Networks (LCN), Singapore, 9–12 October 2017; pp. 219–222.
23. Hernández-Ramos, J.L.; Jara, A.J.; Marin, L.; Skarmeta, A.F. Distributed capability-based access control for the internet of things. *J. Int. Serv. Inf. Secur.* **2013**, *3*, 1–16.
24. Hernández-Ramos, J.L.; Jara, A.J.; Marín, L.; Skarmeta Gómez, A.F. DCapBAC: Embedding authorization logic into smart things through ECC optimizations. *Int. J. Comput. Math.* **2016**, *93*, 345–366. [CrossRef]
25. Databox Project. Available online: https://www.databoxproject.uk/publications/ (accessed on 12 July 2018).
26. Birgisson, A.; Politz, J.G.; Erlingsson, U.; Taly, A.; Vrable, M.; Lentczner, M. *Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud*; Network and Distributed System Security Symposium, Internet Society: Reston, VA, USA, 2014.
27. Maesa, D.D.F.; Mori, P.; Ricci, L. Blockchain based access control. In Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems, Neuchâtel, Switzerland, 19–22 June 2017; pp. 206–220.
28. Ouaddah, A.; Abou Elkalam, A.; Ait Ouahman, A. FairAccess: A new Blockchain-based access control framework for the Internet of Things. *Secur. Commun. Netw.* **2016**, *9*, 5943–5964. [CrossRef]
29. Trust, confidence and Verifiable Data Audit. Available online: https://deepmind.com/blog/trust-confidence-verifiable-data-audit/ (accessed on 12 July 2018).
30. Swan, M. *Blockchain: Blueprint for a New Economy*; O'Reilly Media, Inc.: Newton, MA, USA, 2015.
31. Szabo, N. Formalizing and securing relationships on public networks. *First Monday* **1997**, *2*, doi:10.5210/fm.v2i9.548. [CrossRef]
32. Gomi, H.; Hatakeyama, M.; Hosono, S.; Fujita, S. A delegation framework for federated identity management. In Proceedings of the 2005 workshop on Digital identity management, Fairfax, VA, USA, 11 November 2005; pp. 94–103.
33. Aura, T. Distributed access-rights management with delegation certificates. In *Secure Internet Programming*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 211–235.
34. Zhang, L.; Ahn, G.J.; Chu, B.T. A rule-based framework for role-based delegation and revocation. *ACM Trans. Inf. Syst. Secur.* **2003**, *6*, 404–441. [CrossRef]

35. Chen, N.; Chen, Y.; You, Y.; Ling, H.; Liang, P.; Zimmermann, R. Dynamic urban surveillance video stream processing using fog computing. In Proceedings of the 2016 IEEE Second International Conference on Multimedia Big Data (BigMM), Taipei, Taiwan, 20–22 April 2016; pp. 105–112.
36. Chen, N.; Chen, Y.; Blasch, E.; Ling, H.; You, Y.; Ye, X. Enabling Smart Urban Surveillance at The Edge. In Proceedings of the 2017 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 3–5 November 2017; pp. 109–119.
37. Chen, N.; Chen, Y.; Song, S.; Huang, C.T.; Ye, X. Smart Urban Surveillance Using Fog Computing. In Proceedings of the 2016 IEEE/ACM Symposium on Edge Computing (SEC), Washington, DC, USA, 27–28 October 2016; pp. 95–96.
38. Xu, R.; Chen, Y.; Blasch, E.; Chen, G. A Federated Capability-based Access Control Mechanism for Internet of Things (IoTs). In Proceedings of the Conference on Sensors and Systems for Space Applications, SPIE Defense & Commercial Sensing 2018 (DCS), Orlando, FL, USA, 17 April 2018.
39. Ethereum Homestead Documentation. Available online: http://www.ethdocs.org/en/latest/index.html (accessed on 12 July 2018).
40. Solidity. Available online: http://solidity.readthedocs.io/en/latest/ (accessed on 12 July 2018).
41. Truffle. Available online: http://truffleframework.com/docs/ (accessed on 12 July 2018).
42. Crockford, D. JavaScript Object Notation (JSON). Available online: https://tools.ietf.org/html/rfc4627 (accessed on 12 July 2018).
43. Flask: A Pyhon Microframework. Available online: http://flask.pocoo.org/ (accessed on 12 July 2018).
44. SQLite. Available online: https://www.sqlite.org/index.html (accessed on 12 July 2018).
45. BlednCAC Project. Available online: https://github.com/samuelxu999/Blockchain_dev/ (accessed on 12 July 2018).
46. Go-Ethereum. Available online: https://ethereum.github.io/go-ethereum/ (accessed on 12 July 2018).
47. Xu, R.; Nikouei, S.Y.; Chen, Y.; Song, S.; Polunchenko, A.; Deng, C.; Faughnan, T. Real-Time Human Object Tracking for Smart Surveillance at The Edge. In Proceedings of the IEEE International Conference on Communications, Selected Areas in Communications Symposium Smart Cities Track, Kansas City, MO, USA, 20–24 May 2018.