Hindawi
Security and Communication Networks
Volume 2024, Article ID 9857812, 1 page
https://doi.org/10.1155/2024/9857812

Retraction

Retracted: Enabling Decentralized and Auditable Access Control for IoT through Blockchain and Smart Contracts

Security and Communication Networks

Received 8 January 2024; Accepted 8 January 2024; Published 9 January 2024

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] H. Truong, J. L. Hernández-Ramos, J. A. Martinez et al., "Enabling Decentralized and Auditable access Control for IoT through Blockchain and Smart Contracts," *Security and Communication Networks*, vol. 2022, Article ID 1828747, 14 pages, 2022.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 1828747, 14 pages
https://doi.org/10.1155/2022/1828747

Research Article

Enabling Decentralized and Auditable Access Control for IoT through Blockchain and Smart Contracts

Hien Truong,[1] José L. Hernández-Ramos,[2] Juan A. Martinez,[3] Jorge Bernal Bernabe,[4] Wenting Li,[5] Agustin Marin Frutos,[4] and Antonio Skarmeta[4]

[1]*Elisa Corporation, Helsinki, Finland*
[2]*European Commission, Joint Research Centre, Ispra 21027, Italy*
[3]*Odin Solutions, R&D Department, Murcia 30820, Spain*
[4]*University of Murcia, Department of Information and Communication Engineering, Murcia, Spain*
[5]*NEC Laboratories Europe, Heidelberg 69115, Germany*

Correspondence should be addressed to José L. Hernández-Ramos; jose-luis.hernandez-ramos@ec.europa.eu

Received 18 February 2022; Accepted 3 May 2022; Published 2 June 2022

Academic Editor: Bharat Bhushan

The increase in the interconnection of physical devices and the emergence of the 5G paradigm foster the generation and distribution of massive amounts of data. The complexity associated with the management of these data requires a suitable access control approach that empowers citizens to control how their data are shared, so potential privacy issues can be mitigated. While well-known access control models are widely used in web and cloud scenarios, the IoT ecosystem needs to address the requirements of lightness, decentralization, and scalability to control the access to data generated by a huge number of heterogeneous devices. This work proposes CapBlock, a design that integrates a capability-based access control model and blockchain technology for a fully distributed evaluation of authorization policies and generation of access credentials using smart contracts. CapBlock is intended to manage the access to information in federated IoT environments where data need to be managed through access control policies defined by different data providers. The feasibility of CapBlock has been successfully evaluated in the scope of the EU research project IoTCrawler, which aims at building a secure search engine for IoT data in large-scale scenarios.

1. Introduction

The Internet of Things (IoT) represents an ecosystem of interconnected computing devices equipped with a variety of sensors transmitting and receiving data over the Internet. The development of the IoT is continually growing at a rapid pace; indeed, the International Data Cooperation (IDC) estimated that there will be 41.6 billion IoT devices generating 79.4 zettabytes (ZB) of data in 2025 [1]. IoT devices are intended to continuously collect and distribute a huge amount of data over the network. This makes data management essential so that users have control over their data, as well as the ability to define access rights for their information disclosure to only whom they are willing to share with. Achieving this level of security and privacy is challenging in large IoT systems. Unauthorized access to IoT devices and their data could let attackers execute malicious commands and extract private sensitive data of devices' owners.

The definition of a suitable access control approach for IoT has attracted a significant interest in recent years [2, 3]. The main goal of the proposed efforts is to address the inherent requirements of IoT scenarios, in terms of scalability, lightness, and flexibility to be adopted in different use-cases with heterogeneous devices and networks. Furthermore, the complexity associated with the management of data provided by different users and devices in large-scale scenarios needs to properly avoid conflicting access requirements. For example, in the context of a *smart city* [4], several domains (interchangeably, data providers) should have full control to decide how their data are accessed and distributed. These domains create access control policies to

grant/deny the access on their data to different users. While domains have their own policies, the federated system composing such domains might need common policies in addition to domain local policies. The common policies define how access to data could be granted, and they are applied in the same way to every domain in the federated system. For example, given a policy "it is not legal to share or store European social user data to entities located outside Europe," which has been agreed by all domains, it should not be possible for a domain to issue a policy that is in conflict with the previous one. This security requirement for *managing federated policies* has not been addressed in current access control solutions.

Moreover, among the different access control models, including traditional and well-known approaches such as role-based access control (RBAC) [5] and attribute-based access control (ABAC) [6], the application of the capability-based access control (CapBAC) [7] approach into IoT scenarios has been strongly considered [8, 9]. The rationale behind the CapBAC model is to link a set of privileges (i.e., capabilities) to a certain user that is usually represented by an authorization token called a capability token where such capabilities are embedded by issuing an authorization token, called a capability token, which represents such capabilities. The realization of the CapBAC model in IoT has been proposed by adopting specific technologies and protocols so that it can be deployed even in scenarios with resource-constrained devices. In particular, based on the CapBAC foundations, the Distributed CapBAC (DCapBAC) approach was proposed as a lightweight and scalable approach for IoT, in which capabilities are linked to users through public key cryptography [10]. Therefore, each capability token is associated with the user's public key to avoid misuse or abuse of such credentials. Indeed, the format of DCapBAC tokens is similar to the approach of JSON Web Tokens (JWTs) [11] with an asymmetric proof-of-possession (PoP) key [12].

Even if CapBAC models (including the DCapBAC approach) have been strongly considered by the research community compared to other access control models, there are a few desired security properties that are not met yet. Firstly, most of the existing approaches require a trusted third party (TTP) for the generation and validation of capability tokens. Therefore, in case these components are compromised, adversaries could alter access control policies or the decision-making process itself to allow accesses to protected data or devices. Secondly, existing access control solutions have not addressed *auditability* and *verifiability*. This requirement is the key in contexts where access records should be stored in a trusted manner. For example, in the context of smart home with several smart devices continuously generating data, the home owner can delegate the access to such devices to a service provider, for example, a housekeeping service. Based on the data, the service provider offers certain services and charges some fees. Thus, access records are required to be protected from forgery and illegal modifications made by a semitrusted service provider. Thirdly, most of the existing CapBAC approaches do not support *token revocation*. Therefore, once an access token has been granted, the token can be used by the associated user throughout the tokens' validity period.

To cope with previous issues, blockchain technology has been widely applied to solve security challenges in IoT systems specifically for overcoming limitations of traditional access control approaches [13]. Blockchain is based on an immutable distributed ledger, where trust is distributed among network peers; thus, no single (centralized) third party is required. Blockchain's features such as decentralization, traceability, and tamper-proof nature can be leveraged by access control solutions. In particular, decentralization allows all members of the blockchain to participate in the process of adding and validating transactions/operations. Traceability allows all the peers that maintain the distributed ledger to validate data exchange. Each transaction is included in a block with a unique block address and timestamp. The blockchain is tamper-proof, and an adversary needs to obtain majority support of network members to change records in the blockchain (double-spending attack). Furthermore, adding new transactions to the blockchain requires a *consensus* mechanism, so that peers agree on the validity of such transactions. Moreover, a concept associated with blockchain is represented by *smart contracts*, which are pieces of code in the blockchain to be automatically executed when certain predefined conditions are satisfied.

The integration of blockchain with CapBAC models has been recently considered in the context of IoT scenarios [14–17]. However, these works still lack an integral architecture combining access control components and blockchain to manage the data access in IoT. In addition, most of these works do not provide details about the format of access control policies and capability tokens. Without such a specific design, a thorough overhead evaluation of a blockchain-based access control approach for IoT is not possible. To fill this gap, in this work, we introduce CapBlock as the first comprehensive design for a capability-based access control approach by using blockchain and smart contracts functionality. In particular, CapBlock integrates the eXtensible Access Control Markup Language (XACML) standard [18] for the definition of access control policies, and the DCapBAC model for the specification of capability tokens that was proposed by some of the authors of this work [19]. The functionality for evaluating XACML policies against access requests, as well as the generation of capability tokens, is implemented as the *policy smart contract* and the *capability smart contract*, and deployed over the blockchain. The blockchain stores all records for granted tokens, access history, and policy changes. This makes CapBlock auditable and verifiable. Furthermore, managing federated policies through smart contracts is a key feature of CapBlock. Any domain can add a new policy as long as it complies with existing common and previous local policies. The smart contracts for access control policies are automatically invoked and executed when a domain in the federated system adds a new policy. These smart contracts ensure integrity and conflict-free policies in the federated system. We have implemented CapBlock using the Hyperledger Fabric framework [20]. Based on this, we thoroughly evaluated the

performance with different settings of the scenario, considering several workload configurations. Our results show the feasibility of CapBlock in providing an auditable, verifiable, and federated access control approach for IoT scenarios. In addition, CapBlock has been validated in real-world scenarios in the scope of the EU H2020 IoTCrawler research project [21].

In summary, the contribution of our work is threefold:

(i) A blockchain-based access control modular architecture for IoT based on the DCapBAC model.

(ii) An implementation of the proposed architecture by using the XACML standard for the definition of access control policies and the Hyperledger Fabric for the blockchain network and handler service.

(iii) An exhaustive performance evaluation of the proposed solution to prove its feasibility in terms of network latency and throughput with different workload and blockchain conditions.

The remainder of this article is organized as follows. Section 2 describes some of the main blockchain-based access control approaches for IoT in the literature. Then, Section 3 presents the main building blocks of our work. Section 4 provides a detailed description of CapBlock, and a thorough performance evaluation is given in Section 5. Finally, Section 6 concludes the article with an outlook of our future work in this area.

2. Related Work

In recent years, the development of access control systems for IoT has attracted a significant interest to address the decentralization, lightness, and scalability requirements of such environments [8]. Indeed, while well-established access control technologies are already deployed today (e.g., OAuth [22]), controlling the access to data generated and shared by IoT devices requires the application of novel approaches to manage the access to a huge amount of heterogeneous devices. Towards this end, the use of blockchain has recently been proposed as an alternative to implement well-established access control models taking advantage of the benefits provided by this disruptive technology [13, 23].

Based on the use of blockchain, Ref. [24] integrates different smart contracts to create an access control and information sharing system that is implemented using the Ethereum platform. In addition to the access control itself, the authors present smart contracts to manage user authentication and potential misbehavior. The use of smart contracts is also considered by the FairAccess system [25] for the evaluation of access control policies, as well as the generation and verification of access tokens. This work is extended by Ref. [26], which provides details on access delegation and revocation, as well as a preliminary implementation. Unlike the previous approach, Ref. [27] describes a blockchain-based access control approach to IoT based on a single smart contract in order to simplify system management aspects. The proposal considers the use of constrained devices that cannot be part of the blockchain by defining interactions through lightweight communication technologies and protocols.

Although previous works propose the use of blockchain to create access control systems, they do not consider how the realization of existing access control models can take advantage of the properties of blockchain in terms of decentralization and auditability. Indeed, existing models such as role-based access control (RBAC) [5] or attribute-based access control (ABAC) [6] are already widely deployed in different scenarios, such as Cloud. For example, Ref. [28] uses smart contracts and the Ethereum platform to implement a RBAC authorization system with delegation functionality. However, the RBAC model has well-known issues (e.g., the well-known role explosion problem [29]) that limit its application in IoT scenarios. Therefore, many recent proposals are based on the use of the ABAC model, in which access permissions are associated with attributes of a subject. In this direction, Ref. [30] analyzes the deployment of an ABAC system for IoT using the blockchain implementation of Hyperledger Fabric. The authors discuss several performance aspects, such as latency and the block size of their blockchain system. Furthermore, Ref. [31] describes an ABAC system based on different types of policies that are defined using the XACML standard. However, implementation results are not provided. Also based on XACML, Ref. [32] implements a blockchain system where policy evaluation is carried out through smart contracts, which are additionally used to protect the system itself. In particular, the authors make use of the Solidity language [33] to implement XACML policies and evaluate the resulting system on an Ethereum implementation. Furthermore, Ref. [34] makes use of three smart contracts to implement the Fabric-IoT access control system that is implemented and evaluated on the Hyperledger Fabric blockchain.

In addition to the ABAC model, the capability-based access control (CapBAC) model [35] has been widely considered to be used in IoT scenarios. In CapBAC, access rights are directly associated with the subject, which simplifies authorization management functionality and provides a high level of flexibility [9]. These features are leveraged by Ref. [15], which describes a system for the management, delegation, and validation of access privileges that are designed as capability tokens. The authors provide an implementation on the Ethereum platform that is evaluated in terms of computation time and network latency. Also based on CapBAC, Ref. [16] integrates the use of decentralized identifiers (DID) [36] to manage the identity of IoT devices. The resulting system is partially evaluated on a local testbed. Additionally, Ref. [14] proposes the use of CapBAC to create access tokens for each access right, providing delegation and revocation capabilities. Furthermore, the authors evaluate the cost of their system on Ethereum and compare their results with the approach proposed by [15]. Additionally, while it is not bound to any specific model, Ref. [17] integrates ABAC and CapBAC concepts to build a blockchain-based access control system for IoT, but implementation details are not provided.

As previously described, the advantages of the CapBAC model have been leveraged by several proposals for the

design of a blockchain-based access control system for IoT. Indeed, our proposed system integrates the DCapBAC [19] model (which was proposed by some of the authors of this work) with the ABAC model by using the XACML standard. Unlike some of the described proposals (e.g., [15, 16]), our DCapBAC-based approach uses a well-defined format for capability tokens with a similar structure to the well-known JSON Web Token (JWT) standard [11]. This aspect allows CapBlock to be also deployed in scenarios with resource-constrained devices. Moreover, only Refs. [31, 32] consider the use of XACML policies in their blockchain-based access control approaches for IoT. While XACML eases the definition of authorization policies through a standard approach, tokens are still required in IoT scenarios to allow a scalable and accountable access control solution. In this direction, our approach makes use of blockchain for the evaluation of XACML policies, as well as for the generation and validation of DCapBAC tokens to create an auditable, distributed, and reliable access control system for federated IoT scenarios. Furthermore, unlike most of the approaches described, we integrate this system with an authentication approach based on the FI-WARE platform, which is an open-source platform for the development of interoperable and standard-based smart solutions. Additionally, unlike most of the described works, we evaluate the performance of CapBlock by using Hyperledger Fabric. Indeed, it should be noted that existing literature usually does not provide evaluation results to demonstrate the feasibility of the proposed solutions, or implementations are typically based on the Ethereum platform.

3. Background

3.1. IoTCrawler Framework. IoTCrawler [21] is an H2020 EU research project with the goal of providing a framework for integrating IoT resources coming from different and heterogeneous sources ranging from single IoT devices publicly available, to IoT platforms containing multiple information resources. From both data producers and consumers' point of view, IoTCrawler provides methods for discovering, crawling, indexing, and searching dynamic IoT information. As a key aspect of the project, security has been considered in this complex landscape where information coming from different domains requires also a flexible and dynamic solution for managing access control policies according to the specific requirements that producers would like to apply on their resources [21]. The IoTCrawler architecture consists of building blocks as presented in Figure 1.

In this architecture, the data provider is at the bottom of the diagram, whereas the data consumer is placed at the top of it. For each domain, IoTCrawler considers a *Crawling* component, which allows the integration of information sources. These sources must transform their current data representation into the data model defined by IoTCrawler based on NGSI-LD, which is a representation model standardized under the ETSI ISG CIM working group [37]. As shown, IoTCrawler considers a distributed scenario for the integration of different domains. The cornerstone piece of this diagram is represented by Metadata Repositories (MDRs), which act as data brokers to exchange contextual information by using NGSI-LD. Upon this intradomain layer, the interdomain layer comprises a federation of brokers (i.e., MDRs) to register the information of different domains so that it is the end-point for the other components defined in the upper layers. In particular, the metadata information is processed, enriched, and monitored so that it can eventually be offered to the end-users through the *Orchestrator* and *Search Enabler* components.

Security and privacy are addressed through a transversal layer comprising several technologies, which are grouped in different components, namely, *Identity Manager*, *Blockchain Handler*, *Authorization Enabler,* and *CP-ABE*. These technologies consider both intradomain and interdomain scenarios, so for this reason, IoTCrawler is making use of blockchain technology and the use of smart contracts so that access control policies could be defined and agreed by different domains. This approach is intended to provide a traceable and auditable access control solution for large-scale IoT scenarios as will be described in Section 4.

3.2. Distributed Capability-Based Access Control. In the scope of the IoTCrawler project, we use the capability-based access control model, which has been widely considered in IoT scenarios [9, 35], due to its flexibility and scalability. This model is based on linking a set of access privileges or capabilities to a certain entity, which is identified by a public key. In particular, the approach uses the distributed capability-based access control (DCapBAC) model proposed by Ref. [19], which makes use of access tokens with a similar semantics to the JSON Web Token (JWT) standard [11]. In DCapBAC, the *subject* field contains the public key of the entity associated with the access privileges included in the token. Thus, public key cryptography is used as a proof-of-possession mechanism for a certain token [12]. Furthermore, access privileges are specified using the Authorization Information Format (AIF) [38], which is being defined under the IETF ACE WG.

The DCapBAC model has been considered in the scope of several EU research projects, such as SMARTIE [39], together with an access control policy approach for the automated generation of DCapBAC tokens. Specifically, we use the XACML standard so that users can define access restrictions on their IoT devices using a well-known and widely deployed approach. Figure 2 shows an overview of the components involved in the DCapBAC model and the main interactions required to obtain access to a certain resource. The definition of these components follows the standard access control terminology provided by XACML and OAuth [22] standards. In particular, the *resource owner* is responsible for granting access to a protected resource, which is hosted on a certain *resource server*. To gain access to this resource, an entity acting as a *client* requests a DCapBAC token from the *authorization service* by specifying the target action and resource. In the DCapBAC model, the authorization service is composed of the *capability manager* entity, which is responsible for generating access tokens

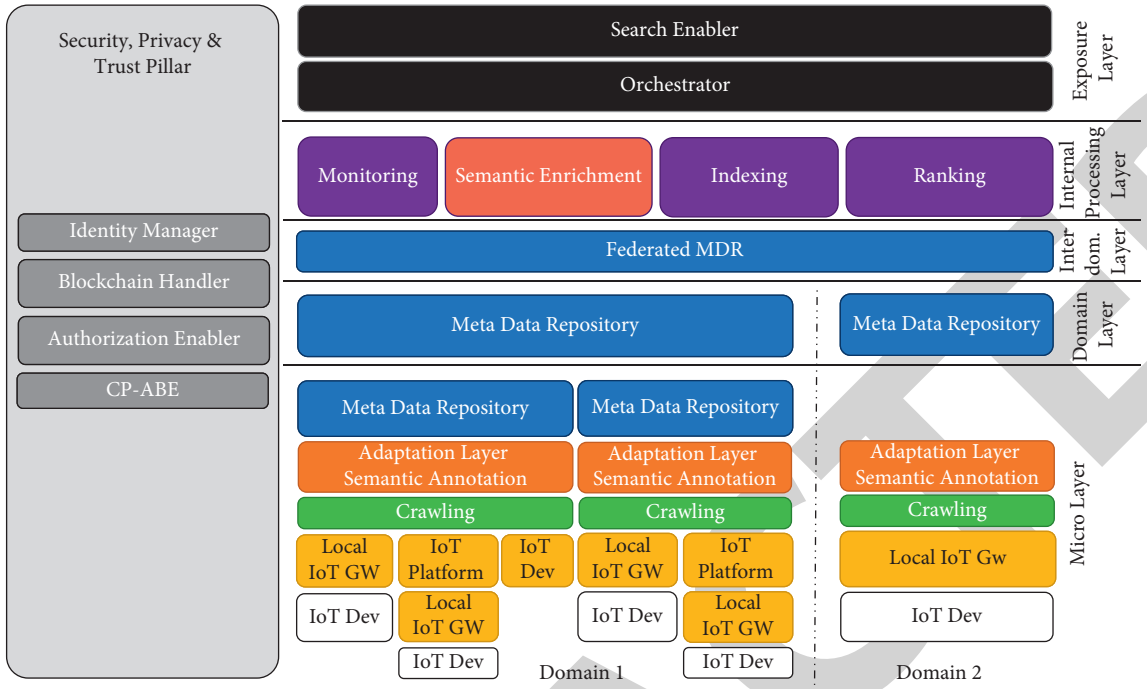

Figure 1: IoTCrawler architecture.

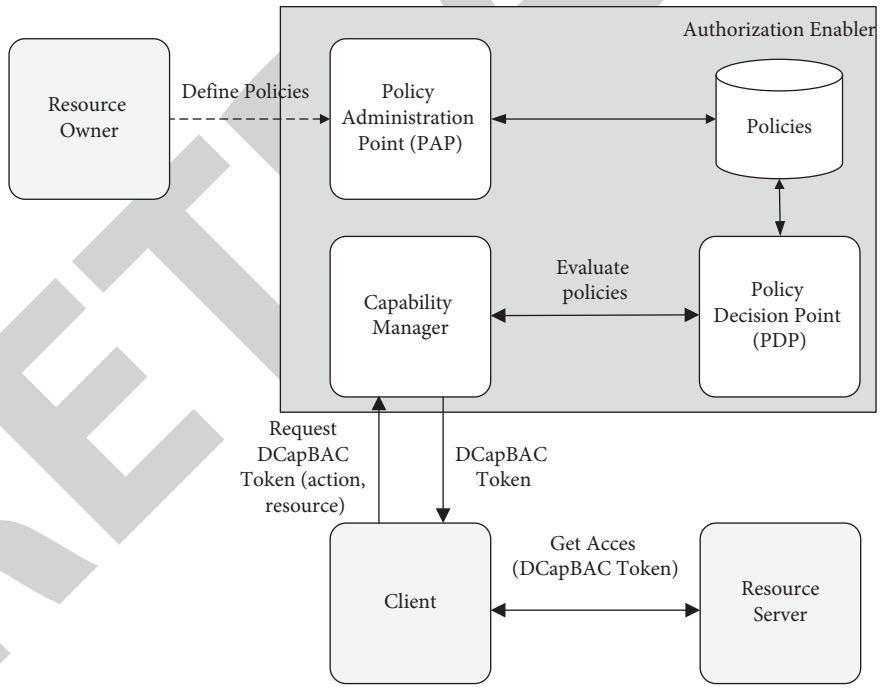

Figure 2: Overview of the DCapBAC model.

based on the authorization decision provided by XACML components. This service includes a Policy Administration Point (PAP) where the resource owner defines access control policies, and a Policy Decision Point (PDP), which evaluates these policies and provides an authorization decision to the capability manager. With this access token, the client can access the requested resource hosted on the resource server. It should be noted that the same token can be used during its validity period without the need to carry out a new authorization process.

DCapBAC has been used and tailored to be deployed in different IoT-enabled scenarios, such as smart buildings [39] and smart grid [40], and a more detailed description can be found in previous works such as Ref. [19]. However, unlike previous works, our approach is intended to leverage immutability, transparency, and traceability features of

blockchain to provide a reliable and accountable access control ecosystem for large-scale and highly distributed IoT scenarios.

3.3. Blockchain and Smart Contracts. Blockchain represents the main example of Distributed Ledger Technologies (DLTs). In blockchain, every transaction is recorded in the ledger in order of occurrence, and several transactions can be recorded in a block. These blocks are linked by including the cryptographic hash value of the previous block into the newly created block. This technique is used to prevent tampering of transactions without being detected. To agree on the validity of transactions, blockchain systems use a *consensus* protocol, such as proof of work (PoW), authority (PoA), or stake (PoS). Furthermore, blockchains are also classified into permissionless and permissioned systems. In a *permissionless* approach, any entity is enabled to perform transactions on the blockchain, while in a *permissioned* system, explicit authorization is required to carry out certain activities.

Associated with the development of blockchain implementations, the concept of *smart contract* has emerged as a key feature to implement the business logic agreed by the members of a certain blockchain system. A smart contract represents a piece of code, which is executed by all the nodes of the blockchain in an automated way. Blockchain members can add smart contracts to the blockchain by adding transactions; indeed, smart contracts are also included in blocks in the blockchain. Note that we focus on permissioned smart contracts, based on an underlying permissioned blockchain, where only authorized owners can update the contracts.

For the realization of a blockchain system, several implementations have been developed in recent years. Among the most popular initiatives, Hyperledger Fabric (or simply Fabric) [20] is an open-source blockchain platform managed by the Linux Foundation. In Fabric, the consensus mechanisms include three processes: endorsement, ordering, and validation. Fabric leverages channels to allow parallel and secure transactions. Transaction ordering is handled by a central *orderer,* which collects transactions submitted by *committers* and asks for votes from *endorsers*. Fabric uses a hybrid replication design, which incorporates primary-backup (passive) replication and active replication. This hybrid design makes Fabric a scalable permissioned blockchain. Chaincode in Fabric is the equivalent concept of smart contract. Because Fabric is a permissioned blockchain, participating peers need to prove their identity to the network. In Fabric, Membership Service Provider (MSP) running on the ordering service contains peer's public key, which is used to verify peer's signature on transactions. Without revealing member's private key, the network still can match the signature created by peer's private key with the public key stored by MSP. Moreover, there are several different implementations of consensus on the strict ordering of transactions between ordering nodes. From version v1.4.1, Fabric supports Raft, a crash fault-tolerant ordering service based on Raft protocol [41]. As will be described in Section 5, CapBlock uses Fabric with Raft consensus to implement its blockchain network.

4. CapBlock Architecture and Interactions

Based on the CapBlock building blocks previously described, this section describes the main components and interactions to realize the proposed approach.

4.1. Architecture Overview. The proposed architecture leverages the DCapBAC authorization model presented in Section 3 with the set of components aimed to ensure auditability, verifiability, and provenance of the access control solution. To this aim, the architecture relies on a blockchain to ensure the immutability and integrity of the security policies, while providing reliability in the federated and decentralized access control system.

Figure 3 shows the main components of the CapBlock architecture. Note that the proposed architecture represents an instantiation of some of the main security components of the IoTCrawler framework (see Figure 1). In particular, the *IoTCrawler Identity Manager* was instantiated through the *Identity Manager Service*, which is responsible for authenticating users and delivering authentication credentials that are required to access other services. Furthermore, it acts as an *attribute provider* storing identity attributes of end-users that are used during the authorization process. Moreover, the *IoTCrawler Blockchain Handler* is instantiated by the *Blockchain Handler Service*, which acts as a blockchain node and it is in charge of interfacing with the blockchain by requesting authorization tokens upon access control requests.

The *IoTCrawler Authorization Enabler* represents the core of the CapBlock approach. This component was instantiated by integrating different access control components based on the DCapBAC model (see Figure 3). In particular, the *Policy Administration Point* (PAP) is a tool used by the *resource owner* to define XACML policies that are stored in the *Policies* database. This component provides a GUI to facilitate the definition of authorization policies for end-users. Furthermore, the *Policy Enforcement Point* (PEP) is responsible for forwarding authorization requests to the Blockchain Handler Service, as well as validating authentication credentials when a client tries to get access to a *resource server*. Moreover, the functionality associated with the Policy Decision Point (PDP) and the capability manager entities has been embedded in the *policy* and *capability* smart contracts, respectively, which are deployed in the blockchain. The former makes use of the XACML policies defined by resource owners for the evaluation of access control requests that are performed by clients. The latter generates capability tokens according to the result of the previous evaluation process. This way, CapBlock provides a fully decentralized management for the evaluation of access control policies and the generation of authorization credentials. The functionality of both smart contracts is described in Section 4.

Furthermore, as shown in Figure 3, CapBlock interactions are divided into three main processes, namely, (a) *defining access control policies*, (b) *obtaining capability token,* and (c) *access with capability token*. The interactions of these

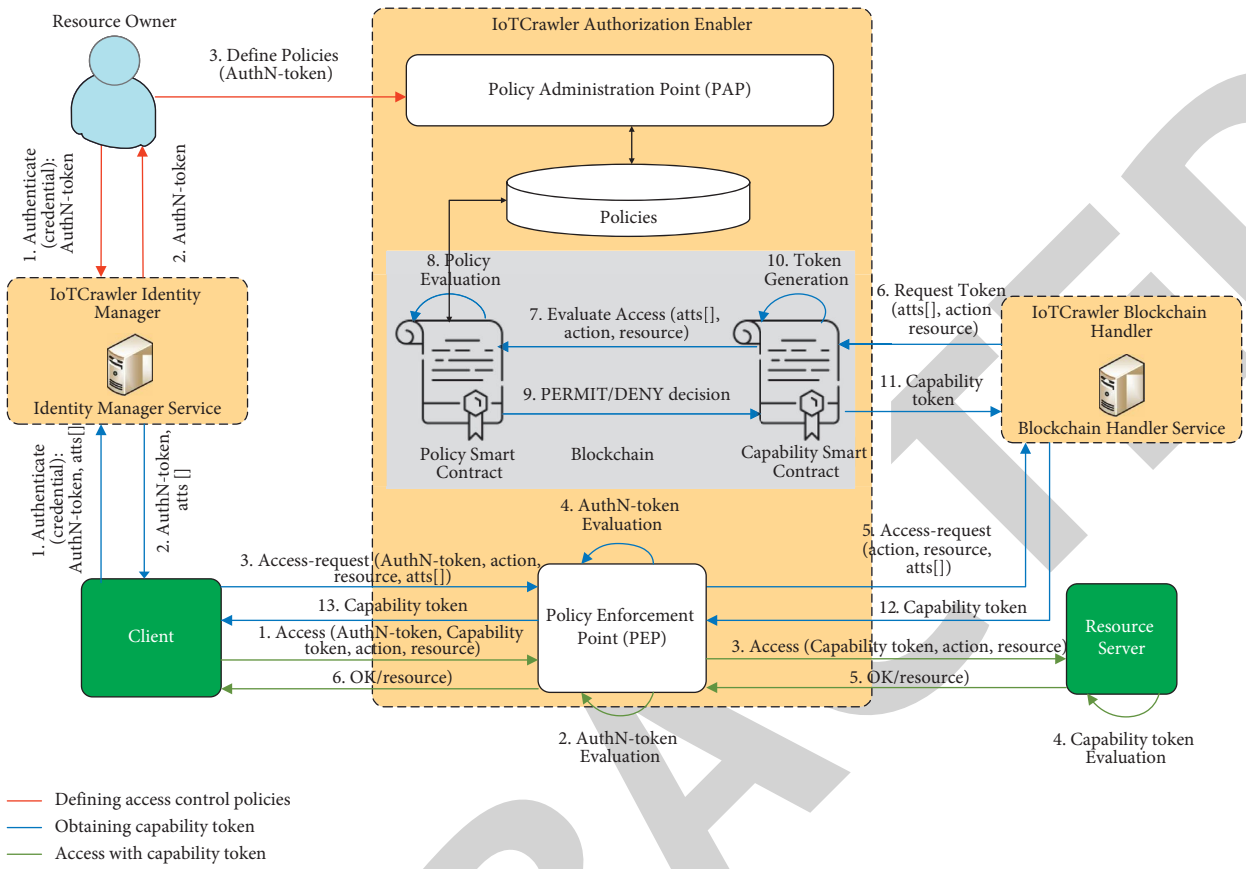

Figure 3: CapBlock architecture and workflow.

three main processes are depicted in three different colors (red, blue, and green, respectively), and they are described in Section 4.

4.2. CapBlock Workflow. As already mentioned, the interactions of CapBlock are divided into three main processes represented by different colors in Figure 3. During the *defining access control policies* process, the *resource owner* is authenticated in the federation through the *Identity Manager Service* (step 1). In particular, this service is implemented based on the identity management system provided by the EU FI-WARE platform called Keyrock [42]. As a result of this process, an authentication token is delivered to the resource owner (step 2) that is used to access the PAP to define new XACML policies (step 3). These policies are stored in the *Policies* database, and they will be accessible by the policy smart contract for the next phase.

In the *obtaining capability token* phase, a user or service (acting as a *Client*) is also authenticated by the *Identity Manager Service* (step 1), which delivers an authentication token and the set of identity attributes associated with the requesting user (step 2). These attributes are embedded in a certificate or a similar credential, which is signed by the service. Then, the client requests a capability token through the PEP (step 3) by indicating the requested action and resource, and including the token and attributes obtained in the previous step. The PEP validates the authentication token (step 4) and forwards the access request to the *Blockchain Handler Service* (step 5). This service serves as the interface between off-chain components and the blockchain by translating access control requests to blockchain transaction format in order to invoke the *capability* smart contract (step 6), which in turn invokes the *policy* smart contract (step 7). This smart contract is in charge of evaluating the XACML policies (step 8) defined in the previous phase by the resource owner. The decision generated by the policy smart contract is transferred to the capability smart contract (step 9); in case of a positive decision (i.e., PERMIT), a capability token is generated following the format of the DCapBAC approach [19] (step 10) and forwarded to the requesting client (steps 11–13).

After obtaining a capability token, the client is enabled to access the requested resource in the *Access with capability token* phase. For this purpose, the requesting client makes an access request to a certain *resource server* through the PEP by including the authentication token and capability token, as well as the requested action and resource (step 1). Like in the previous phase, the PEP is in charge of validating the authentication token (step 2) and forwards directly the request to the resource server (step 3). This entity validates the capability token considering the requested action or resource (step 4). While it is beyond the scope of this article, this process could require a mechanism by which the client demonstrates to be the entity associated with such capability

token integrate a proof-of-possession mechanism [12], so that the client demonstrates to have a cryptographic key associated with such token. Towards this end, the use of typical transport-layer security approaches, or emerging techniques such as the Ephemeral Diffie-Hellman over COSE (EDHOC) [43] could be considered. In case of a positive evaluation, the resource is provided to the requesting client (steps 5-6). Note that this phase is decoupled from the previous one; indeed, a client can reuse the token several times in different contexts to access different resources, until it is revoked or expired.

The authentication in the PEP is needed to check that nobody is impersonating the client using a stolen/compromised capability token. During the Access Stage, the PEP acts as a proxy that avoids clients to interact directly with the resource IoT server devices that might xnot be able to perform the proof-of-possession authentication continuously (e.g., constrained IoT devices). This approach enables different ways of authentication in the PEP Proxy that cannot be carried out directly by the resource server.

4.3. Policy and Capability Smart Contracts. Compared to the DCapBAC approach, the functionality of the PDP and capability manager components are shifted to the blockchain as smart contracts. In particular, CapBlock defines the *capability* smart contract, which is responsible for generating the verifiable capability tokens for requesting clients, as well as the *policy* smart contracts to evaluate access control policies. Such policies are used to make auditable authorization decisions based on client's attributes, requested action, and resource.

In our design, the Blockchain Handler Service invokes smart contracts to perform the functionality associated with the evaluation of authorization policies, and the generation of capability tokens. When smart contracts are executed, they generate transactions recorded on the ledger. Below, we describe the details of policy and capability smart contracts.

Algorithm 1 shows the functions included in the policy smart contract. *Register* function checks whether a policy is already in the blockchain (in fact, hash values of policies are stored in the blockchain). If the policy is new, it creates a transaction to register the policy and stores it in the blockchain's internal database. *Query* function retrieves a policy of a specific domain. *Evaluation* function takes an XACML request and domainID as input and applies a matching algorithm following XACML policy evaluation standard, for the request and the domain's policy to make access grant or deny decision.

Algorithm 2 shows the functions included in a capability contract. *Register* function checks and puts a new token to the blockchain. *Query* function retrieves a token and its status. *Generate* function takes attributes of user, device, and resource to create a new token. A capability token includes information of the issuer, the issuing date, user, action, resources, and signature of the issuer). Each token has a unique identity. In *generate* function, the time attribute is extracted from current system time. Signature function

```
contract policy:
register(domainID, XACMLfile):
// verify if the policy exists
if ! exist(hash(XACMLfile)) then
  put(domainID, hash(XACMLfile))
  store(XACMLfile)
  return true
else
  return false
end if
query(domainID):
  policy = get(domainID)
  if error != nil then
    return error
    //error occurred
  else
    return policy
    // hash value of policy returned
  end if
evaluate(domainID, XACMLrequest):
  policy = get(domainID)
  if error != nil then
    return error
    //error occurred
  else
    response = match(policy, XACMLrequest)
    return response
  end if
```

Algorithm 1: Policy smart contract.

(sig_func) is an algorithm to generate signature. The token is a JSON serialization of its attributes and a signature on those attributes. The token is digitally signed by the endorsing peer using its securely stored private key, which is passed as one of parameters of the smart contract call during its execution. The new token is set to be active by default. It should be noted that this function is called after using the function *Evaluation*, which is included in the policy contract previously described and refers to steps 8, 9, and 10 of Figure 3. Thus, in case the *Evaluation* function returns a "permit" decision, the *Generate* function is in charge of generating a new token. Moreover, *Revoke* function sets a token to be inactive when it receives a request to revoke a specific token.

5. Evaluation

5.1. Performance Analysis. The distributed ledger (blockchain in our current deployment) operates using Fabric framework with the Raft consensus protocol. With this framework, the most essential factor that affects overall performance of a blockchain network is the ordering service. We measured latency and throughput, as primary performance metric for blockchain, with various parameter settings of network sizes (number of ordering nodes) and block sizes (blocks committed to the blockchain of each transaction). Ordering latency is the time a transaction needs to wait for the ordering service until its order in a block is

```
contract capability:
register(token):
// verify if the policy exists
if ! exist(hash(token)) then
  token.status = active
  put(token)
  store(token)
  return true
else
  return false
end if
query(token):
  get(token)
  if error != nil then
    return error
  else
    status = token.status
    return status
  end if
generate(issuer, privkey, user, device, resource, action):
      time = get(current_system_time)
      id = get(random_ID)
      sig = sig_func(privkey, id, time, issuer, user,
      device, resource, action)
      token = JSON(id, time, issuer, user, device,
      resource, action, sig)
  register(token)
  return token
  revoke(token):
  get(token)
  if error != nil then
    token.status = inactive
    return true
  else
    return false
  end if
```

Algorithm 2: Capability smart contract.

assigned. Ordering throughput is the service's capacity to handle a certain number of transactions per second (tps).

To utilize all available resources at our lab environment for the experiments in this article even they do not have identical hardware configuration, we ran the benchmark experiment on 8 servers where ordering nodes are distributed in local network environment. Four of eight servers have 4 CPUs and 32 GB RAM; the remaining four servers have 6 CPUs and 128 GB RAM. All servers run Ubuntu 18.04 and Fabric version 1.4. All peers are given private keys and certificates to prove memberships when joining the permissioned blockchain network. The majority of the components were developed using Golang and Java as programming languages. We implemented smart contracts to run the various functions for policy management and capability manager as described in Section 4. We deployed these contracts in a Fabric network using Raft as the consensus algorithm.

Each server runs a docker container with Fabric images installed as node in the blockchain network. Before starting the network, it requires to generate new crypto materials. Fabric does provide samples for doing so such as cryptogen and configtxgen. The number of organizations, the number of orderers, and the number of channels are configured in a configuration file. To start the network, we write a shell script that starts docker containers on all servers, creates a channel, joins peers to the channel, then installs and instantiates the chaincodes, and finally starts the fabric-restapi on CLI containers. When the network is started, clients can query chaincodes or running through interfaces provided by fabric-restapi. In our evaluation, experiments are setup and run iteratively by various bash/shell scripts to reflect changes in the configuration file.

We varied sizes of our blockchain network with number of orderers ranging from 10 to 200. We evaluated the blockchain network performance using *throughput* and *latency* metrics. With each parameter setting, we run the experiment 10 times and compute average values.

Figures 4–7 show benchmark results of our blockchain network. We increased the number of clients, hence the number of transactions, and measured the throughput until

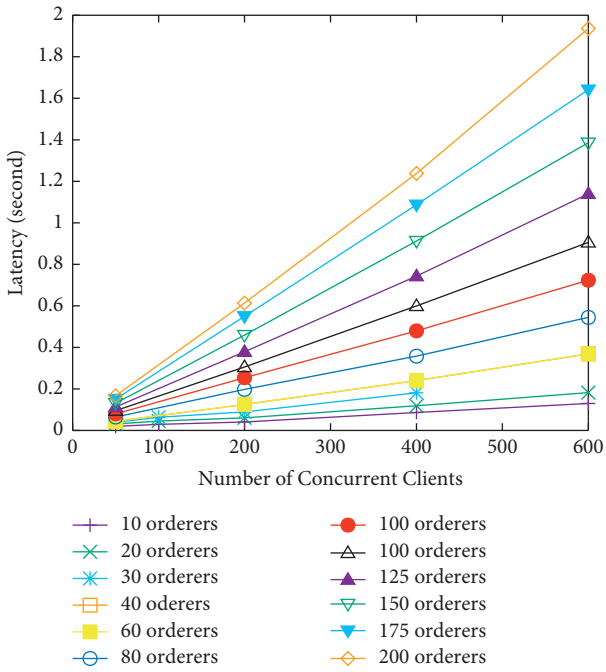

Figure 4: Latency performance with various network sizes and concurrent clients (batch size is fixed at 50 and payload size of a single transaction is fixed at 1024).

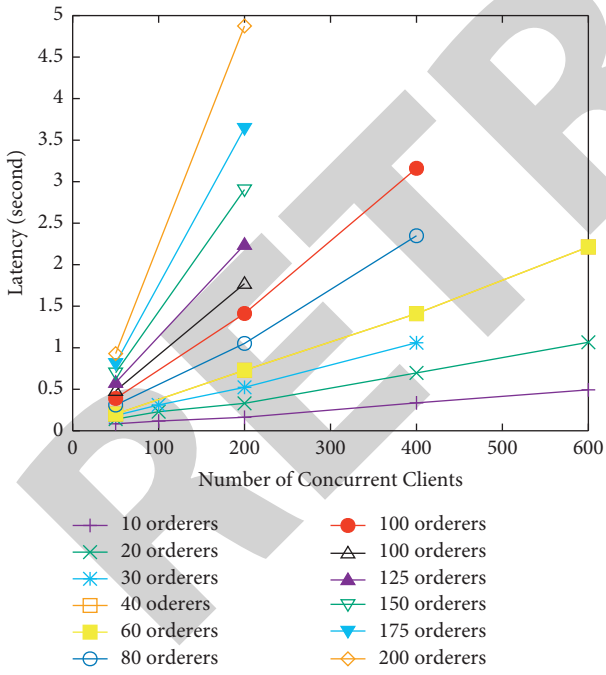

Figure 5: Latency performance with various network sizes and concurrent clients (batch size is fixed at 50 and payload size of a single transaction is fixed at 10240).

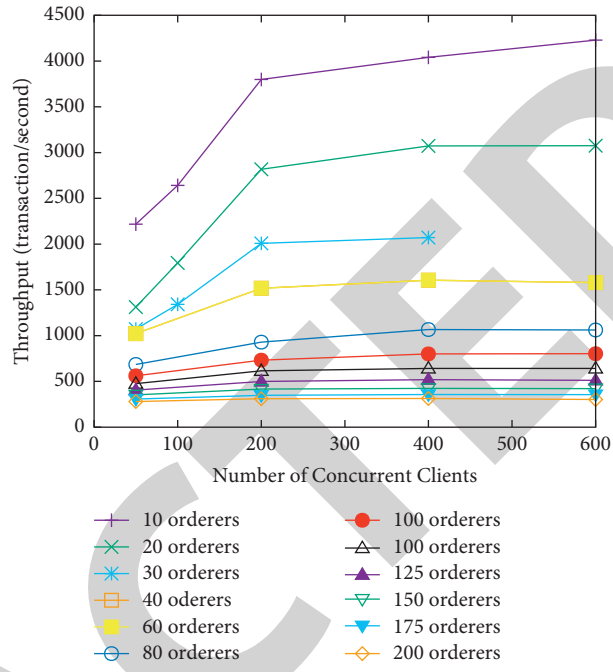

Figure 6: Throughput performance with various network sizes and concurrent clients (batch size is fixed at 50 and payload size of a single transaction is fixed at 1024).

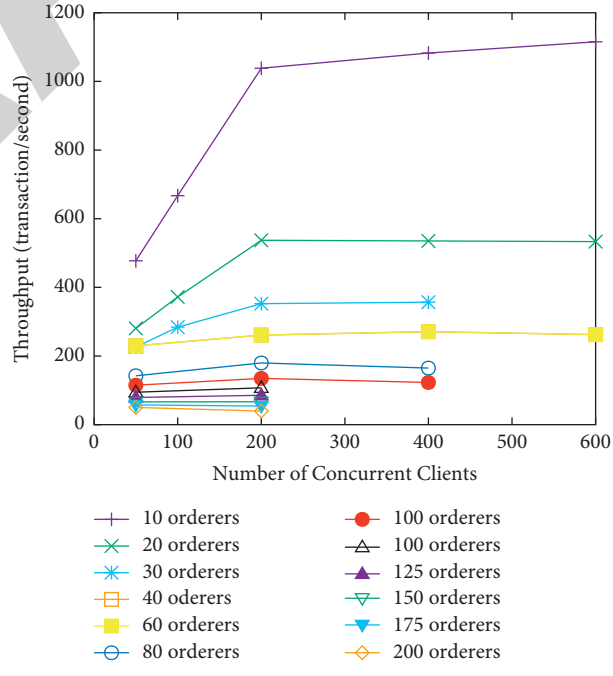

Figure 7: Throughput performance with various network sizes and concurrent clients (batch size is fixed at 50 and payload size of a single transaction is fixed at 10240). When the throughput saturated, we stopped the experiment.

it saturated. We show results of batch size of 50 transactions with batch timeout 3 seconds and the payload as size of single transaction of 1024 and 10240 bytes. When the number of orderers is small, latency is very low (with 10 orderers, less than 0.5 s for 600 concurrent clients). When the number of orderers increases, the latency also increases (with 200 ordering nodes, the latency at 200 concurrent clients is 5 s). The same pattern is observed for the throughput performance, which is shown in Figures 6 and 7.

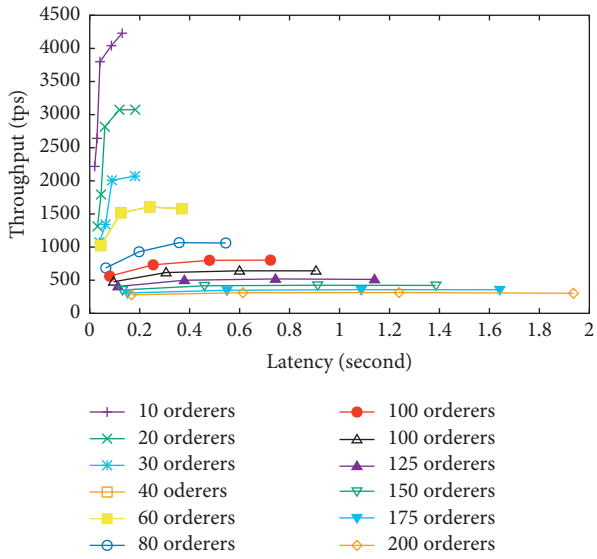

Figure 8: Latency and throughput performance (batch size is fixed at 50 and payload size of a single transaction is fixed at 1024).

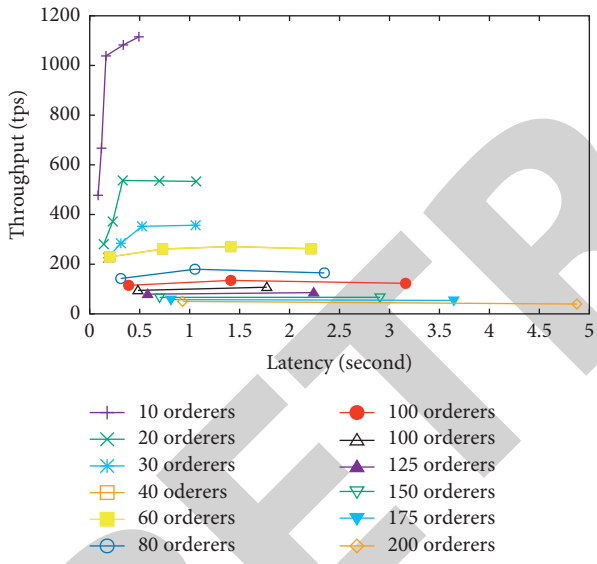

Figure 9: Latency and throughput performance (batch size is fixed at 50 and payload size of a single transaction is fixed at 10240).

Throughput drops when there are more orderers (with 200 nodes, the throughput at 600 concurrent clients goes below 500 transactions/second). That can be explained that the more the orderers, the more messages to send and thus the more messages to process. Figures 8 and 9 show the correlation of latency and throughput. With a small number of orderers, the throughput reaches a high value but quickly saturates (with 10 orderers, at the latency of 0.5, the throughput saturates at nearly 1200 tps).

This benchmark results show the trade-off of latency/throughput performance and network resilience against faulty. When the network size is larger, it is more resilient to tolerate faulty nodes; however, it comes with the cost of higher latency and lower throughput.

The overall performance of blockchain network needs to consider also the transaction execution time, smart contract invocation time, and transaction validation cost. These factors are very much application dependent.

To evaluate the performance of smart contracts, we chose a fixed setting of the blockchain network. We evaluate the time efficiency of CapBlock focusing on registering and evaluating policies, which includes achieving access token capacity. To focus on the smart contract performance, we used one ordering service and five blockchain peers. We used Apache benchmark tool to exhaust the blockchain handler and the blockchain network. Note that, when the blockchain network setting changes, for example, with more orderer nodes, the latency and throughput to update new states to the blockchain would vary. The execution of queries to respond to end-users (outside of the blockchain network) and the update a new state change to the blockchain could be done asynchronously.

Figure 10 shows the performance of requesting an access authorization and capability token. When the blockchain handler receives a request from user in XML format through Restful request, it parses the request and invokes the policy smart contract with evaluation function (see Algorithm 1) via CLI commands to execute the contract by peers in the blockchain network. The returned result is either a permit or a deny (including not applicable in case there is not a policy for such request). If the result is a permit, the blockchain handler invokes the capability smart contract to generate a token. The decision and the token pair are finally returned to the user in JSON format via Restful response. To assess the heaviest load, we sent the requests that satisfy the domain policy so that the capability smart contract will be always invoked and executed. The experiment results show that when there are 1000 of requests (of which there are always 10 concurrent requests), the time it takes from sending a request to receiving final decision and token is around 200 ms. When there are 1000 requests of which 100 requests are concurrently sent, it takes 4 s to process. This is comparable to the nonblockchain DCapBAC solution, which takes 180 ms–260 ms to complete 100 policy decision requests.

We measured the latency of registering policies to the blockchain network. Note that each domain stores only a valid policy at a time. The result shows that registering policies takes less than a second for the blockchain handler to send a policy and get confirmation (with 1000 requests, 10 concurrent ones sent, total registering time is of one second). Note that the blockchain network takes a while to record a transaction of policy registration to the network, but the end-users do not have to wait for this process; rather, the process is done asynchronously within the blockchain network.

Overall, our performance analysis shows that blockchain does not incur a significant overhead to handle access control functions while offering better security benefits.

5.2. Security Analysis. The security of an IoT system mainly depends on the access control mechanisms to manage the access to the information provided by IoT devices. Towards

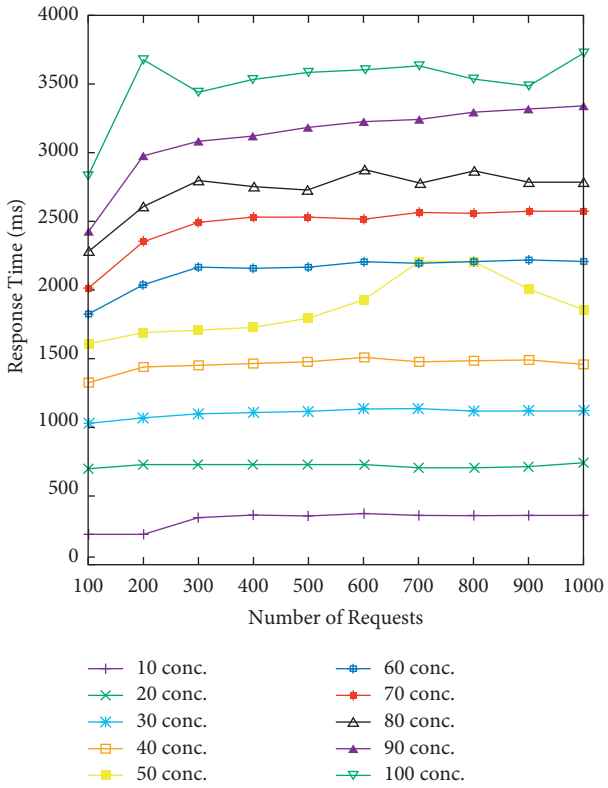

Figure 10: Time performance for evaluating a policy and retrieving capability token by varying number of concurrencies.

this end, the use of well-known authentication and authorization approaches must be in place. Authentication is employed to ensure users and entities are actually the ones they claim to be. After a successful authentication, authorization is used to assess whether such identity is allowed to perform a certain action over a specific resource or service. As previously described, our framework integrates authentication and authorization in a holistic architecture based on the IoTCrawler framework.

While it is not the focus of our work, authentication aspects are managed through the existing Keyrock service provided by the FI-WARE platform [44], which is based on well-known standard approaches, such as OAuth and the System for Cross-domain Identity Management (SCIM) [45]. In our architecture, authentication is managed by the IoTCrawler Identity Manager, which integrates such functionality by generating authentication tokens. It should be noted that a user could be tracked by using the same authentication token when requesting a capability token (step 3 in Figure 3). To mitigate such privacy concern, authentication tokens could be forced to be changed after each access, or privacy-preserving approaches could be considered so that users cannot be tracked. For example, in our previous work [42], we integrated anonymous credential systems by using advanced cryptographic algorithms in FI-WARE for such purpose. Moreover, in our architecture, authentication tokens are evaluated by the PEP. While this component is considered a separate component from the blockchain infrastructure, its functionality could be included in a smart contract, and it can be considered as future work based on the proposed system, so the advantages of blockchain can be also leveraged for authentication purposes. Moreover, it should be noted that the interactions for requesting and delivering authentication tokens are secured by well-known mechanisms, such as TLS.

Regarding authorization aspects, unlike the original DcapBAC model [19, 39] where main authorization components (i.e., PDP and capability manager) are fully trusted, in CapBlock, we consider an adversary who attempts to modify access control policies and fool the system to grant access tokens to malicious users. To mitigate such concern, CapBlock leverages the use of blockchain to embed the functionality of such components in smart contracts to avoid single point of failure entities in the proposed solution. Furthermore, it should be noted that a potential attacker will not be able to get a capability token unless it is properly authenticated through the IoTCrawler Identity Manager, as authentication is a step required for authorization purposes. A capability token is securely generated and signed only by permissioned organization of the blockchain network. The endorsing mechanism of the blockchain network ensures no adversary can generate, commit transactions, store, and distribute invalid tokens. Moreover, during the generation process of a new capability token, it should be noted that the token is delivered through the Blockchain Handler Service and the PEP, which could act as honest-but-curious entities to get access to such token. To cope with this aspect, the token could be encrypted by using the user's public key, which could be associated with the user's certificate considered for authentication purposes (step 1 in Figure 3).

Like in the case of authentication, the use of the same token for getting access to a specific resource could provoke privacy concerns for linkability reasons. A similar approach based on forcing to obtain capability tokens after each access could be considered, so that the PEP cannot link the access requests coming from the same user. Alternatively, the use of advanced privacy-preserving approaches can be leveraged to create unlinkable tokens, as some of the authors previously proposed [46].

Furthermore, the use of the same capability token for getting access to a specific resource could provoke privacy concerns in case users are identified through their public key in such token. Indeed, the access requests of a user could be linked even if such user is identified through a certain pseudonym in the capability token. Like in the case of authentication, a similar approach based on forcing to obtain capability tokens after each access could be considered, so that the PEP cannot link the access requests coming from the same user. In this case, each token would be associated with a different pseudonym. Alternatively, the use of advanced privacy-preserving approaches can be leveraged to create unlinkable tokens, as some of the authors previously proposed [42]. In particular, the use of Idemix [47] (as one of the main examples of anonymous credential systems) would enable the disclosure of certain information of the token (e.g., the access rights) or predicates, without the need

to disclose the user's identity. In addition, it should be noted that during the access to a certain resource server, the authentication token is validated by the PEP. Therefore, as already mentioned in Section 4, for end-to-end authentication between Client and Resource Server, the capability token could require a proof-of-possession key, so that the Client is able to demonstrate that it is the user associated with such capability token. Therefore, an attacker will not be able to use a capability token unless it has the cryptographic key associated with the token. Indeed, additional details about the security properties provided by DCapBAC can be found in Ref. [19] based on the AVISPA tool [48]. It should be noted that the implementation of a proof-of-possession mechanism between Client and Resource Server could be a complementary step to the authentication in the PEP, which could still be used to deny the access to unauthenticated requests, so that the access to IoT devices (acting as resource server) could be further protected.

6. Conclusion

We have presented CapBlock, the first solution to integrate DCapBAC access control into a blockchain and to leverage smart contract to handle access control decision. CapBlock achieved its goal by introducing a permissioned blockchain network based on Hyperledger Fabric framework and two smart contracts—policy contract and capability contract to implement various functions for policy and capability management. We have evaluated CapBlock in terms of throughput and latency. The experimental results show that the overhead is comparable with nonblockchain DCapBAC while offering more security and flexibility to access control in IoT search context. As the future direction, we plan to integrate other functionalities for access management in our blockchain infrastructure, such as the generation of authentication credentials or cryptographic keys (e.g., based on the CP-ABE) scheme to realize a decentralized IoT platform for secure data sharing.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This article received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement nos. 779852 and 830929.

References

[1] Idc, "Iot growth forecast by idc," 2019, https://www.idc.com/getdoc.jsp?containerId=prUS45213219.

[2] J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su, and B. Fang, "A survey on access control in the age of internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.

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

[4] J. L. Hernandez-Ramos, J. A. Martinez, V. Savarino et al., "Security and privacy in internet of things-enabled smart cities: challenges and future directions," *IEEE Security & Privacy*, vol. 19, 2020.

[5] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[6] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[7] L. Gong, "A secure identity-based capability system," in *Proceedings of the. 1989 IEEE Symposium on Security and Privacy*, pp. 56–63, Oakland, CA, USA, May 1989.

[8] A. Ouaddah, H. Mousannif, A. Abou Elkalam, and A. Ait Ouahman, "Access control in the internet of things: big challenges and new opportunities," *Computer Networks*, vol. 112, pp. 237–262, 2017.

[9] R. Xu, Y. Chen, E. Blasch, and G. Chen, "A federated capability-based access control mechanism for internet of things (iots)," in *Sensors and Systems for Space Applications XI*vol. 10641, Bellingham, International Society for Optics and Photonics, Article ID 106410U, 2018.

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

[11] M. Jones, J. Bradley, and N. sakimura, "Json web token (jwt)," *Tech. rep., RFC*, vol. 7519, May 2015.

[12] M. Jones, J. Bradley, and H. Tschofenig, *Proof-of-possession Key Semantics for Json Web Tokens (Jwts), RFC 7800, Standards Track*, IETF, Fremont, 2016.

[13] I. Butun and P. Österberg, "A review of distributed access control for blockchain systems towards securing the internet of things," *IEEE Access*, vol. 9, 2020.

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

[15] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: a blockchain-enabled decentralized capability-based access control for iots," in *Proceedings of the2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1027–1034, IEEE, Halifax, NS, Canada, August 2018.

[16] Y. Liu, Q. Lu, S. Chen et al., "Capability-based iot access control using blockchain," *Digital Communications and Networks*, vol. 7, 2020.

[17] O. J. A. Pinno, A. R. A. Gregio, and L. C. De Bona, "Controlchain: blockchain as a central enabler for access control authorizations in the iot," in *Proceedings of the GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, Singapore, December 2017.

[18] Oasis, "eXtensible access control Markup Language," 2013, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[19] J. L. Hernández-Ramos, A. J. Jara, L. Marín, and A. F. Skarmeta Gómez, "Dcapbac: embedding authorization logic into smart things through ecc optimizations," *International Journal of Computer Mathematics*, vol. 93, no. 2, pp. 345–366, 2016.

[20] E. Androulaki, A. Barger, V. Bortnikov et al., "Hyperledger fabric," in *Proceedings of the Thirteenth EuroSys Conference, EuroSys '18*, April 2018.

[21] T. Iggena, E. B. Bin Ilyas, M. Fischer et al., "IoTCrawler: challenges and solutions for searching the internet of things," *Sensors*, vol. 21, no. 5, p. 1559, 2021.

[22] D. Hardt, "The OAuth 2.0 authorization framework," Tech. rep., RFC 6749, Microsoft, Redmond, October (2012).

[23] A. I. Abdi, F. E. Eassa, K. Jambi, K. Almarhabi, and A. S. A.-M. Al-Ghamdi, "Blockchain platforms and access control classification for iot systems," *Symmetry*, vol. 12, no. 10, p. 1663, 2020.

[24] T. Sultana, A. Almogren, M. Akbar, M. Zuair, I. Ullah, and N. Javaid, "Data sharing system integrating access control mechanism using blockchain-based smart contracts for iot devices," *Applied Sciences*, vol. 10, no. 2, p. 488, 2020.

[25] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Advances in Intelligent Systems and Computing. Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 523–533, Springer, Cham, 2017.

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

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

[28] N. Tapas, G. Merlino, and F. Longo, "Blockchain-based iot-cloud authorization and delegation," in *Proceedings of the 2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 411–416, IEEE, Taormina, Italy, June 2018.

[29] A. Elliott and S. Knight, "Role explosion: acknowledging the problem," in *Proceedings of the 2010 International Conference on Software Engineering Research and Practice*, pp. 349–355, Las Vegas, NV, USA, July 2010.

[30] M. A. Islam and S. Madria, "A permissioned blockchain based access control system for iot," in *Proceedings of the2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 469–476, IEEE, Atlanta, GA, USA, July 2019.

[31] C. Dukkipati, Y. Zhang, and L. C. Cheng, "Decentralized, blockchain based access control framework for the heterogeneous internet of things," in *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pp. 61–69, Tempe AZ USA, March 2018.

[32] D. Di Francesco Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.

[33] C. Dannen, *Introducing Ethereum and Solidity*, Vol. 1, Springer, , New York, 2017.

[34] H. Liu, D. Han, and D. Li, "Fabric-iot: a blockchain-based access control system in iot," *IEEE Access*, vol. 8, pp. 18207–18218, 2020.

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

[36] D. Reed, M. Sporny, D. Longley et al., "Decentralized Identifiers (Dids) V1. 0," *Draft Community Group Report*, 2020.

[37] Etsi Gs Cim 009, "Context information management (CIM): ngsi-ld api," 2019, https://docbox.etsi.org/ISG/CIM/Open.

[38] C. Bormann, "An authorization information format (AIF) for ACE," 2019, https://tools.ietf.org/html/draft-bormann-core-ace-aif-06.

[39] J. L. Hernández-Ramos, M. V. Moreno, J. B. Bernabé, D. G. Carrillo, and A. F. Skarmeta, "Safir: secure access framework for iot-enabled services on smart buildings," *Journal of Computer and System Sciences*, vol. 81, no. 8, pp. 1452–1463, 2015.

[40] J. A. Martínez, J. L. Hernández-Ramos, V. Beltrán, A. Skarmeta, and P. M. Ruiz, "A user-centric internet of things platform to empower users for managing security and privacy concerns in the internet of energy," *International Journal of Distributed Sensor Networks*, vol. 13, no. 8, Article ID 1550147717727974, 2017.

[41] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," 2014, https://raft.github.io/raft.pdf.

[42] J. Bernal Bernabe, J. L. Hernandez-Ramos, and A. F. Skarmeta Gomez, "Holistic privacy-preserving identity management system for the internet of things," *Mobile Information Systems*, vol. 2017, Article ID 6384186, 20 pages, 2017.

[43] G. Selander, J. Mattsson, and F. Palombini, "Ephemeral diffie-hellman over COSE (EDHOC), IETF, ID draft-selander-lake-edhoc-05," 2021, https://tools.ietf.org/html/draft-ietf-lake-edhoc-05.

[44] A. Preventis, K. Stravoskoufos, S. Sotiriadis, and E. G. Petrakis, "Iot-a and fiware: bridging the barriers between the cloud and iot systems design and implementation," in *Proceedings of the 6th International Conference on Cloud Computing and Services Science CLOSER*, no. 2, pp. 146–153, Rome Italy, April 2016.

[45] K. Li, P. Hunt, B. Khasnabish, A. Nadalin, and Z. Zeltsan, *System for Cross-Domain Identity Management: Definitions, Overview, Concepts, and Requirements, RFC 7642, Standards Track*, IETF, Fremont, 2015.

[46] J. Hernández-Ramos, J. Bernabe, M. Moreno, and A. Skarmeta, "Preserving smart objects privacy through anonymous and accountable access control for a m2m-enabled internet of things," *Sensors*, vol. 15, no. 7, pp. 15611–15639, 2015.

[47] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 21–30, Washington, DC USA, November 2002.

[48] A. Armando, D. Basin, Y. Boichut et al., "The avispa tool for the automated validation of internet security protocols and applications," in *Proceedings of the International Conference on Computer Aided Verification*, pp. 281–285, Springer, Edinburgh Scotland UK, July 2005.