Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 6926408, 25 pages
https://doi.org/10.1155/2022/6926408

Research Article

An Attribute-Based Access Control Model for Internet of Things Using Hyperledger Fabric Blockchain

Elham A. Shammar[1], **Ammar T. Zahary**[1,2] **and Asma A. Al-Shargabi**[2,3]

[1]*Department of Information Technology, Faculty of Computer and Information Technology (FCIT), Sana'a University, Sana'a 1247, Yemen*
[2]*Department of Computer Science, Faculty of Computing and IT, University of Science and Technology, Sana'a 15201, Yemen*
[3]*Department of Information Technology, Collage of Computer, Qassim University, Buraydah 51452, Saudi Arabia*

Correspondence should be addressed to Elham A. Shammar; eshammar@gmail.com

Received 20 January 2022; Revised 19 June 2022; Accepted 21 June 2022; Published 15 July 2022

Academic Editor: Michele Girolami

The Internet of Things (IoT) is emerging from its infancy and establishing itself as a component of the future Internet. However, the ability to manage a huge number of IoT devices is one of the IoT technical challenges. To implement access control, traditional schemes typically rely on a trusted central organization. Traditional centralized access control systems of the IoT lead to the shortcomings of a single point of failure, low overall system efficiency, and ethical and privacy issues. To overcome such challenges, an attribute-based access control model using Hyperledger Fabric blockchain (ABAC-HLFBC) is proposed in this paper. By adopting ABAC, it is no longer to create access control lists (ACLs) or assign roles to all system users. Instead, ABAC grants access based on the attributes presented by the target. No one is permitted access unless he/she possesses sufficient attributes that correspond to the access policy. The Hyperledger Fabric Raft consensus mechanism has been used to verify the transaction on the proposed model because it has a demanding feature for faster and less complicated consensus in comparison to the Kafka ordering service. To evaluate the proposed model, it has been tested against the most recent previous work, called fabric-iot model. The proposed model has been tested and evaluated in two parts. The first part tests the cost time using a client test program written in Golang. The second part tests the latency and throughput using the Hyperledger Caliper benchmark tool. Results show that the proposed model efficiently outperforms the previous work in terms of the performance metrics mentioned above.

1. Introduction

As the IoT gains popularity, the security and privacy of the heterogeneous data generated by IoT devices become more crucial than ever. This significance stems from the fact that the data generated by IoT devices may contain highly sensitive information such as video and audio from smart surveillance systems, location and activity patterns, medical data from fitness equipment, or even the daily schedule of family members at home. To reap the full potential benefits of IoT devices, their generated data needs to be shared with different service providers. For example, the data generated from fitness devices may need to be shared with the hospital and the physician, whereas the temperature sensor data may need to be shared with emergency departments and service providers such as Google and Amazon that can collect users' data in smart homes such as Echo or Google Home, to ensure a better quality of service [1, 2].

Currently, data requesters' methods for collecting user data from IoT devices are ambiguous, and in some cases, they are doubtful since the data owners cannot control how their data is shared with the data consumers. In some cases, the data consumers introduce some form of agreement policy that the data owner must agree to enjoy the intended service. However, the data owner is forced to blindly trust the data consumer. This agreement policy is mostly high-level and ambiguous. Moreover, the data owner has no way of ensuring that the data requester is adhering to

the terms of the agreement and not collecting more than what was agreed upon. This allows malicious parties to access users' sensitive and confidential data in IoT devices through backdoors installed previously. Therefore, when sharing IoT device data with third parties, we must be sure who is gaining access to the shared data to determine whether or not the data ends up in the hands of the wrong people. We also need to define what information is accessed to determine whether the IoT data requester is collecting anything without the data owner's permission [1].

More than 150,000 IoT devices were compromised, and the investigation determined that access control was the primary reason for the security breach [3]. Access control can be used to detect unauthorized access and control people's entrance to restricted areas [4]. It is defined as a method of granting or denying access privileges to the user based on whether or not he/she is viewed as a potential threat [5, 6]. A complete access control solution includes three components: authentication, authorization, and auditing. The true identity of a subject is determined by authentication. The subject's authorization determines whether or not he/she has the authority to perform operations on the object. Auditing (or accountability) enables subsequent analysis of the system's activities. The three components play critical roles in system security, but authorization deserves special attention because it is responsible for enforcing access rules. Nevertheless, the adoption of improper access control systems will cause significant privacy and economic harm to individuals and enterprises [3].

Traditional access control technologies are addressed through centralized approaches such as discretionary access control (DAC), identity-based access control (IBAC), and mandatory access control (MAC). DAC and IBAC technologies depend on creating access control lists (ACLs) for everyone on the system, which would be nearly impossible on the IoT system, whereas MAC is usually enforced by the central administrator, which has the disadvantage of a single point of failure [7]. In addition, centralized approaches have the shortcomings of limited scalability and low throughput and reliability. Furthermore, they may be dependent on a third party and have limited availability, which may result in a performance bottleneck [8, 9].

Some access authorization systems make use of three well-known traditional architectures: XACML, UMA, and OAuth. Even though these architectures have been used in different fields, none of them addresses the issues caused by the centralization of the architecture components. Moreover, none of these architectures provide critical IoT access control characteristics such as scalability, user transparency, and resilience to intermittent wireless communications. Thus, despite the significant efforts to bring appropriate access control to the IoT, traditional architectures have design barriers that prevent them from achieving complete success [3].

Because IoT devices have limited CPU capability, memory, and battery life, a lightweight access control solution with low latency is required [6]. In addition to limited performance, IoT devices are likely to be mobile, and they may belong to different organizations or users, making it difficult for centralized access control to meet the access control requirements of the IoT environment [8]. Such flaws of centralized systems can be overcome with blockchain technology [10], which provides an excellent platform for developing distributed applications for mutually untrustworthy parties because it eliminates the need for a trusted central authority [1].

In comparison to capability-based access control (CBAC) and role-based access control (RBAC), Attribute-based access control (ABAC) is one of the most suitable decentralized models for IoT scenarios, with high scalability, dynamicity, and flexibility. ABAC governs access between the subjects to the objects based on the attributes of operations, entries, and associated environments [8]. It can also provide fine-grained and expressive access control. Therefore, ABAC is better suited to IoT than other access control mechanisms [1]. However, recent research that focuses on access control schemes based on ABAC is based on centralized models to improve security and privacy [2].

IoT network requires a distributed and lightweight secure access control that uses permissioned blockchain such as Hyperledger Fabric blockchain. However, in fabric-iot [8], the authors did not incorporate ABAC of Hyperledger Fabric that supports ABAC for chaincode. They also did not include the endorsement policy in the configurations of the Hyperledger Fabric network. In addition, in fabric-iot, a single orderer node is launched using Apache Kafka and ZooKeeper ensembles. Kafka ordering service is a third-party service, and it is not fully decentralized. Kafka ordering service also is hard to configure and manage, and it does not use a byzantine fault-tolerant (BFT) consensus mechanism.

This motivates us to propose ABAC-HLFBC, a distributed and lightweight secure access control model for IoT based on ABAC and Hyperledger Fabric blockchain. By leveraging Hyperledger Fabric's low transaction latency, the proposed ABAC-HLFBC model can serve IoT resource access requests much faster (around 4 seconds) [1]. We implemented the ABAC on the blockchain by utilizing smart contracts. By using ABAC, we can provide fine-grained and expressive access control. ABAC is one of the most suitable decentralized models for IoT scenarios, with high scalability, dynamicity, and flexibility. ABAC first extracts the attributes of the subject (user), object (resource), permission, and environment; then flexibly combines these attributes' relationships; and finally transforms permission management into attribute management, resulting in a dynamic and fine-grained access management method [8]. To verify the transaction in our design, we chose the Raft ordering service cluster because of its features of a faster and less complicated consensus. The Raft consensus mechanism can dynamically select orderer nodes to achieve reliable and fast consensus. Using the Raft ordering service, it is possible to achieve the root of trust model within the Hyperledger Fabric network [11]. Our contribution can be summarized as follows:

(1) We develop an IoT access control model that is based on ABAC and Hyperledger Fabric (ABAC-HLFBC) to overcome traditional centralized access control issues

(2) We define a device resource sharing model based on real-world data production from IoT devices. To simplify the device's sharing mode and storage structure, the ABAC-HLFBC model uses the URL as the device's data resource [8]

(3) We implement three smart contracts on Hyperledger Fabric: device contract (DC), policy contract (PC), and access contract (AC). DC includes methods for storing the URL of device-generated resource data and querying it. The PC can be used by administrators to manage the ABAC policies. AC is the main program used to implement an access control method for normal users [8]

(4) In Hyperledger Fabric, ABAC is used for restricting access to the specific user having the necessary attributes in their certificate. In the proposed, model, we Add the "cid" library to the implementation, which supports ABAC for chaincode. "cid" includes functions that allow interaction with asset attributes, such as "GetId," "GetAttributeValue," and "GetX509Certificate"

(5) In the proposed model, we add the endorsement policies to the configurations, which has a large impact on the Hyperledger fabric's consensus

(6) To verify the model's performance, a proof of concept implementation has been provided to compare the cost time, latency, and throughput of the proposed model with the benchmark (the fabric-iot model in [8])

The rest of this paper is organized as follows: Section 2 summarizes the related works that concern IoT-based blockchain access control. Section 3 provides a brief overview of blockchain, Hyperledger Fabric, the Hyperledger Fabric ordering service, and ABAC. The proposed model is described in Section 4. Section 5 introduces the experiment configurations and settings. The results study and the evaluation of the proposed model are introduced in Section 6. We conclude the paper in Section 7.

2. Related Work

Because of the rapid growth of the IoT, distributed access control systems are required. Access control will benefit from blockchain technology features such as decentralization, non-tampering, and data encryption. Blockchain technology, in version 3.0, introduced the concept of smart contracts, which serve as its core technology and create a reliable and safe operating environment for applications while also giving the blockchain more powerful functions. As a result, many researchers have proposed different IoT access control methods that combine existing access control methods with blockchain and smart contracts [8].

FairAccess, a decentralized authorization management framework based on blockchain, was introduced by Ouaddah et al. in [12]. FairAccess is a bitcoin-based access control mechanism that makes use of new types of transactions to grant or revoke user access. The blockchain is used in this framework to distribute access tokens. However, FairAccess access control mechanism is very basic and does not provide fine-grained access control for heterogeneous IoT devices. It may also cause delays due to the owner's communication since tokenization is used for authorization. The authors also did not test the security and performance of their architecture. ControlChain, a blockchain-based architecture for IoT access authorization, was presented by Pinno et al. [3]. Their architecture supports offline operation because wireless media is the dominant media type in IoT communication, which is known to be unstable and may result in intermittent connections. Unlike FairAccess [12], each identity in this model could be linked to any other identity. However, the authors did not test the security and performance of their architecture, and the scheme's efficiency was not demonstrated in their work. Furthermore, IoT devices must store a portion of the blockchain in their architecture.

Novo [13] presented a new architecture for managing IoT devices. The proposed architecture incorporates a decentralized access control system that is linked to geographically dispersed sensor networks. The IoT devices in the proposed architecture are not part of the blockchain network, making it easier to integrate existing IoT devices into the proposed system. However, the use of Ethereum blockchain technology may limit the proposed solution. Moreover, the transactions initiated by the manager nodes may be subject to lengthy delay. Although this solution gains scalability by distributing query permissions through management hubs, it may face security threats if the manager is malicious. Ayoade et al. [14] proposed IOTSMARTCONTRACT, a decentralized data management system for IoT devices that stores the audit trail of data access on the blockchain and enforces all data access permissions using smart contracts. The authors used Intel SGX as a component of the trusted execution environment (TEE), which ensures the security and privacy of the application's sensitive parts (code and data). However, the use of the Ethereum blockchain may limit the proposed solution.

Ourad et al. [10] proposed a blockchain-based solution to authenticate users and allow secure access to IoT devices. The proposed approach uses the blockchain to manage data access to IoT data in a decentralized manner. The raw encrypted data is stored in an SGX-enabled platform, ensuring that all data processing and storage occurs in a secure environment. The authors' strategy thwarted crafted attacks attempting to hijack legitimate sessions and brute force credentials. However, the use of Ethereum blockchain technology may limit their proposed solution. Riabi et al. [6] proposed a trustworthy and distributed access control solution based on the blockchain. The authors chose a model that combines the CBAC and IBAC and uses tokens that allow the requester to access a resource. They also used an ACL, an IBAC-provided mechanism for storing a list of subjects and their access permissions. The resource owner will distribute the ACLs to the blockchain by registering the ACLs in a smart contract. However, due to a large number of unknown identities, it is nearly impossible to create an ACL for everyone in the IoT system.

Ding et al. [7] proposed a novel attribute-based access control scheme for IoT systems that greatly simplifies access management. The authors used blockchain technology to record the distribution of attributes. It is no longer necessary to create ACLs or assign roles to all system users. No one is permitted access unless he/she possesses sufficient attributes that correspond to the access policy. The access control process has also been optimized to meet the high efficiency and lightweight calculation requirements of IoT devices. However, there is a storage overhead because additional data, such as global parameters, session keys, and access policies, must be stored locally on each device. Moreover, the session key uses the AES-128 algorithm to ensure the confidentiality and authenticity of the communication between the involved parties; thus, there is a computational overhead as the number of attributes increases. Islam and Madria [1] proposed a permissioned blockchain-based IoT access control system in which each phase of access control, such as creating access policies and making access control decisions, is based on the agreement of all stakeholders. They designed and built ABAC on Hyperledger Fabric, leveraging its smart contracts and distributed consensus to enable IoT distributed access control. Nevertheless, the latency in their system increases when the number of peers endorsing a transaction increases. Because the ABAC policy evaluation algorithm is NP-complete, the complexity grows when the number of attributes in the policy increases. The latency also increases as the number of attributes required to satisfy ABAC policy grows.

Figueroa et al. [2] presented an ABAC model for RFID systems based on chaincode. The Hyperledger Fabric blockchain serves as the basis for their ABAC system. Their ABAC model employs smart contracts and blockchain technology as adaptable infrastructures to represent the trust and endorsement relationships required in the ABAC model to ensure the security of RFID systems. However, to build a secure supply chain management system, the authors must create a strong mutual authentication RFID protocol that works in tandem with their blockchain system. Zhang et al. [5] proposed a collaborative ABAC scheme based on blockchain. A digital account for each device is created using the blockchain, which records the attributes and access policies used for authorization and securely forwards access information. A verifiable collaboration mechanism is intended to meet the needs of controlled access authorization in an emergency while also detecting malicious behavior and limiting extra authorization for a specific group. However, there is a storage overhead since each device must store a configuration file containing an ID, an IP address, a groupId, a pair of ECC keys, and an access policy. As a result, when the number of requesters increases, the storage overhead also increases. The authors also compared the computational overhead of their proposed scheme to [7], where their scheme is more computationally efficient than [7] because they reduced the number of ECDSA signing and verifying operations. Nonetheless, the computational overhead with the collaboration scheme is significantly higher than with no collaboration scheme.

Sultana et al. [15] proposed a blockchain-based access control and data-sharing system. The goals of the proposed system are to achieve trustworthiness, authentication, and authorization for data sharing in IoT networks. To provide efficient access control management, smart contracts such as register contract (RC), access control contract (ACC), and judge contract (JC) are used. When the misbehavior is detected, a penalty is assigned to that subject. Furthermore, if the subject does not engage in any misbehavior, the privileges are set, and access is granted. Mbarek et al. [16] proposed a blockchain-based access control (BAC) mechanism for the IoT in smart home systems. The incorporation of blockchain into IoT networks via agent-embedded systems distinguishes the BAC solution, in which the authors modeled the agent-based policy for blockchain management and designed the algorithms for smart home access control.

Algarni et al. [9] proposed a novel architecture based on a multi-agent system that employs a private distributed blockchain, resulting in a light architecture and a decentralized access control system for the IoT. However, the authors did not execute the proposed architecture to test its security. Bouras et al. [17] proposed IoT-CCAC, an IoT consortium capability-based access control model. The authors prioritized data exchange and interoperability by organizing access control data into assets, services, and profiles to make the solution granular and flexible enough to be more suitable for IoT rapid growth and scalability. However, the authors must conduct additional research on the security and privacy of blockchain-based access control in IoT networks and application scenarios.

Ren et al. [18] proposed the SIApps' ledger (SILedger), an open, trusted, and decentralized access control mechanism based on blockchain and attribute-based encryption (ABE). It supports effective SIApp authorization in untrusted and heterogeneous SDN-IoT control domains. They redesigned blockchain transaction, token initialization, token encryption, and token update schemes specifically for SIApps to achieve cross-domain, flexible, and fine-grained permission management. Iftekhar et al. [19] demonstrated access control and established a root of trust for IoT devices using the Hyperledger Fabric blockchain and IoT devices. An ABAC mechanism was built using Hyperledger Fabric to gain access to the IoT device. Using Hyperledger Fabric source code, the authors created executable binaries and Docker images for ARM64. However, the system must evaluate various configurations and integration models to assess the performance of hardware capabilities, power consumption, processing and memory constraints, scalability, network latency, and various network consensus protocols.

Liu et al. [8] proposed an access control system for the IoT called fabric-iot, which is based on the Hyperledger Fabric blockchain framework and ABAC. The system includes three types of smart contracts: device contract (DC), policy contract (PC), and access contract (AC). Fabric-iot, in conjunction with blockchain technology and ABAC, can provide a decentralized, dynamic, and fine-grained access control management system in the IoT. However, to coordinate distributed communications within the blockchain network in fabric-iot, a Kafka ordering service with a single

broker is launched using Apache Kafka ZooKeeper ensembles. Kafka is a third-party service with a slower consensus speed than a Raft. Moreover, in fabric-iot, there is no endorsement policy, which has a large impact on the Hyperledger Fabric's consensus. The endorsement policy specifies who must endorse a specific transaction for it to be considered valid by the validating peers.

The contributions of the previously discussed papers are summarized in Table 1.

To prevent unauthorized users from accessing data resources, a decentralized fine-grained attribute-based access control via smart contracts is required [20]. A comparison between the access control articles that incorporate ABAC and Hyperledger Fabric, which resemble the proposed ABAC-HLFBC model, is presented in Table 2. In [5, 7], as mentioned previously, there is a storage overhead because there is additional data that needs to be stored locally for each device. The Hyperledger Fabric's ABAC and endorsement policies have not been included in [5, 7, 8]. In addition, the scheme in [5] used a Solo consensus algorithm with one orderer node, which has been deprecated and may be removed entirely in a future release of Hyperledger Fabric. In [1], two orderer nodes backed by a Kafka-ZooKeeper cluster are used. Article [1] also did not incorporate ABAC of Hyperledger Fabric. One orderer organization with a Solo ordering service is used in [19]. As mentioned previously, Fabric-iot [8] uses Apache Kafka-ZooKeeper with one broker. The additional administrative overhead of managing a Kafka cluster may be undesirable to many users.

In the proposed ABAC-HLFBC model, there is no storage overhead because the certificates will be issued by the Hyperledger Fabric certificate authority (CA), which means that there is no need to store additional data in IoT devices. In Hyperledger Fabric, ABAC is used for restricting access to the specific user having the necessary attributes in their certificate. Therefore, in the proposed model, the "cid" library that supports ABAC for chaincode has been added and used. The endorsement policy has a large impact on the Hyperledger fabric's consensus. This is because it specifies who must endorse a specific transaction for it to be considered valid by the validating peers. Therefore, we have added the endorsement policy to the proposed model. In addition, a Raft ordering service with five orderers is used. A Raft-based ordering service is easier to set up and manage than a Kafka-based ordering service, and its design allows different organizations to contribute nodes to a distributed ordering service. Raft contains all of its components within the ordering node; hence, we only need to configure the orderer image. Raft, unlike Kafka, attempts to develop a byzantine fault-tolerant BFT ordering service. It also offers a more efficient, fast, and reliable consensus mechanism with higher throughput than Kafka.

3. Preliminary

This section first introduces the blockchain, Hyperledger, Hyperledger Fabric, Hyperledger Fabric components, and the Hyperledger ordering mechanism. Then, it describes the ABAC and compares it to the CBAC and RBAC.

3.1. Overview of Blockchain. In 2008, Satoshi Nakamoto introduced the blockchain to serve as the public ledger for the Bitcoin cryptocurrency [21]. The term "blockchain" refers to a type of distributed ledger technology in which transactions are recorded using a hash, which is an immutable cryptographic signature. It is extremely difficult to significantly modify, hack, or cheat the blockchain system [22]. Because of its unique characteristics, such as decentralization, immutability, anonymity, security, and auditability, blockchain has gotten a lot of attention in recent years. As the blockchain evolves, a wide range of nonmonetary applications has been explored [21, 23]. Blockchain is expected to have a significant impact on a variety of industries, including finance and real estate, government administration, energy, and transportation. Blockchain technology is gaining popularity, with applications in a variety of fields such as supply chains, healthcare, IoT, sharing economies, and vehicular edge computing [24].

Blockchain can be used to overcome the viral spread of false/fake information with malicious intent. In IoT, storing IoT data on the blockchain adds another layer of security to help prevent malicious attacks [25, 26]. Anyone who attempts to overwrite existing data records is prevented by blockchain encryption. The combination of blockchain and IoT enables secure machine-to-machine transactions while also improving security and decreasing inefficiencies [19]. However, the limited transaction processing power is a hindrance, especially when compared to proven alternatives such as distributed database systems [24].

Even though the public blockchain is well-suited for cryptocurrency, it suffers from a scalability issue known as "blockchain bloat," which limits the number of transactions the network can process. Due to the limitations of block size (1 MB) and block creation frequency (1 block every 10 minutes), Bitcoin, for example, can only process 7 transactions per minute. Private/permissioned blockchain transactions, on the other hand, are much faster because they do not rely on Proof of work (PoW) or Proof of stake (PoS). Instead, private/permissioned blockchain uses much faster consensus protocols, such as byzantine fault tolerance (BFT) [1]. As opposed to permissionless blockchains, the use of permissioned blockchains allows only authorized devices to become part of the IoT network. This prevents nodes from joining the blockchain network and writing to the ledger directly. Permissioned blockchain does not require a large amount of computation to reach a consensus; therefore, it does not take a long time or use a lot of energy [27–29].

Furthermore, permissioned blockchains have lower network latency because of the measures that public blockchains must employ to motivate their peers. Moreover, there are fewer trust issues, which means that fewer security measures are required between nodes. This results in a more responsive network, which may be desirable for IoT implementation. Another benefit of the permissioned blockchain implementation is that the system data is entirely contained within the network, giving the network owners complete control over their personal data [30, 31]. As a result, for IoT access control, a permissioned blockchain

Table 1: IoT access control using blockchain.

Paper	Contributions	Implementation	Blockchain platform	Smart contract support	Configuring IoT as a blockchain node	Support ABAC	Metrics
[12]	FairAccess, a decentralized authorization management framework	Yes	Bitcoin	Yes	Yes	No	—
[3]	ControlChain, a blockchain-based architecture for IoT access authorizations	No	—	No	Yes	No	(i) Scalability (ii) Fault-tolerant (iii) Low object overhead (iv) Compatibility
[13]	A new architecture for arbitrating permissions and roles in IoT	Yes	Ethereum (private)	Yes	No	No	(i) Throughput (ii) Timeout
[14]	IOTSMARTCONTRACT, A decentralized system of data management for IoT devices	Yes	Ethereum	Yes	No	No	(i) Gas utilization for write operation on smart contract (ii) Throughput (iii) Avg seal and unseal time
[10]	A blockchain-based solution to authenticate users and allow secure access to IoT devices	Yes	Ethereum	Yes	No	No	(i) Availability (ii) Scalability (iii) Decentralization (iv) Tamper proof
[6]	A trustworthy and distributed access control solution based on a blockchain mechanism	Yes	Ethereum	Yes	Yes	No	—
[7]	A novel attribute-based access control scheme for IoT systems	Yes	Hyperledger Fabric	Yes	—	Yes	(i) Storage overhead (ii) Computation overhead
[1]	A permissioned blockchain-based access control system for IoT	Yes	Hyperledger Fabric	Yes	No	Yes	(i) Latency of transaction arrival rate (tps)
[2]	An attribute-based access control in RFID systems	Yes	Hyperledger Fabric	Yes	No	Yes	—
[5]	An attribute-based access control scheme	Yes	Hyperledger Fabric	Yes	No	Yes	(i) Storage overhead (ii) Computational overhead (iii) Time consumption
[15]	A blockchain-based access control and data sharing system	Yes	Ethereum	Yes	No	No	(i) Transaction cost (ii) Execution cost (iii) Smart contracts cost
[16]	A blockchain-based access control (BAC) for smart home	Yes	Hyperledger Fabric	Yes	No	No	(i) Service execution time
[9]	A novel architecture based on a multi-agent system	No	Private blockchain	No	—	No	—
[17]	IoT-CCAC, IoT consortium capability-based access control model	Yes	BigchainDB	No	—	No CBAC	(i) Authentication and creation time of transactions

Table 1: Continued.

Paper	Contributions	Implementation	Blockchain platform	Smart contract support	Configuring IoT as a blockchain node	Support ABAC	Metrics
[18]	SILedger, access control for SDN-IoT applications (SIApps)	Yes	—	No	—	Yes ABAC model is given by NIST	(i) Authorization time (ii) Call latency
[19]	Hyperledger Fabric access control system	Yes	Hyperledger fabric	Yes	Yes (in raspberry pi)	Yes	(i) Latency (ii) Consensus speed (iii) Throughput (iv) Resource cost (v) Communication cost
[8] Benchmark	Fabric-iot, a blockchain-based access control system in IoT	Yes	Hyperledger Fabric	Yes	No	Yes	(i) Cost time (ii) Consensus speed
The proposed model	ABAC-HLFBC access control model	Yes	Hyperledger Fabric	Yes	No	Yes	(i) Cost time (ii) Latency (iii) Throughput

Table 2: Access control research incorporating ABAC and HLF.

Paper	The use of ABAC of HLF	Ordering service type	HLF endorsement policy
[7]	No	Not mentioned	No
[1]	No	Two orderer nodes backed by a Kafka-ZooKeeper cluster	Yes
[2]	Yes	Not mentioned	Yes
[5]	No	One Orderer organization with a Solo ordering service	No
[19]	Yes	One Orderer organization with a Solo ordering service	Yes
[8] Benchmark	No	One orderer backed by apache Kafka-ZooKeeper	No
The proposed ABAC-HLFBC model	Yes	Raft ordering service with five orderer nodes	Yes

Table 3: Blockchain platform comparison [33].

Features	Bitcoin	Ethereum	Hyperledger fabric
Block size	1 MB	Ethereum's block size varies	Hyperledger's block size varies
Permissioned?	No	Yes	Yes
Permissionless?	Yes	Yes	No
Validation time	10 minutes	15-20 sec	Less than Ethereum
Consensus mechanism	PoW	PoW and PoS (Casper)	PBFT
Scalability	No	No	Yes
Throughput Transaction per second (TPS)	7 TPS	20 TPS	Between 3,000 and 20,000 TPS
Vulnerability to attacks	51% attack	51% attack	>1/3 faulty nodes and DoS attack
Cryptocurrency	Bitcoin	Ether	No native cryptocurrency
Smart contract	No	Yes (written in solidity)	Yes (written in go, Java, Node.js)
Data confidentiality	Yes	No	Yes
User authentication	No	Digital signature	Based on enrolment certificates
Energy and computational cost	High	High	Low

implementation is a better fit. As shown in the comparison in Table 3, the Hyperledger Fabric is more suitable for IoT than Bitcoin and Ethereum. This motivates us to use Hyperledger Fabric to create a permissioned blockchain-based access control model for the IoT. Because our scheme is built on a permissioned blockchain, access requests are processed much faster than on a permissionless blockchain [32].

3.2. Hyperledger. Hyperledger [34] is a Linux Foundation project that creates and promotes a wide range of business blockchain technologies, such as distributed ledger frameworks, smart contract engines, utility libraries, client libraries, graphical interfaces, and sample applications [35]. Hyperledger's goal is to build a scalable blockchain that will allow organizations to conduct business with anyone without the need for mutual trust. By incorporating new processes into traditional blockchain features for more accurate identity verification, Hyperledger aspires to go where the blockchain has not yet arrived [36]. Some of the frameworks that Hyperledger provides are Hyperledger Fabric (contributed by IBM) [34, 37], Hyperledger Sawtooth [38], Hyperledger Iroha [39], Hyperledger Burrow, and Hyperledger Indy [35]. Hyperledger also contains open-source tools such as Hyperledger Composer [40], Hyperledger Caliper [41], Hyperledger Explorer [42], Hyperledger Grid, Hyperledger Cello, Hyperledger URSA, and Hyperledger Quilt/Interledger.js.

3.3. Hyperledger Fabric (HLF). The Hyperledger Fabric (contributed by IBM) was introduced in July 2017 [32]. HLF, formerly an IBM project known as Open BC, is the first project to be incubated at Hyperledger. It is an open-source permissioned blockchain, wherein only authorized nodes can participate in the blockchain. Hyperledger Fabric also allows the development of smart contracts, called "chaincodes" in this context. Chaincode is executed when transactions are submitted and validated. Hyperledger Fabric is a highly adaptable blockchain technology implementation. It can run smart contracts written in any programming language. The Hyperledger Fabric platform allows for the management of multiple networks capable of supporting a wide range of assets, agreements, and transactions between different groups of member nodes. Fabric is written in Golang, uses gRPC APIs, and supports Java Chaincode and Node.js client SDK [11]. Hyperledger Fabric introduces a novel framework that separates the execution phase from

the consensus phase and implements policy-based endorsements [43]. The official documentation of Hyperledger Fabric is available online on [44].

In contrast to Bitcoin and Ethereum, HLF does not use cryptocurrency [23, 27]. It is intended for developing applications or solutions with a modular architecture, allowing components, such as consensus and membership services, to be plug-and-play [35]. Hyperledger Fabric also meets IoT requirements for low electric power consumption, no mining, no PoW, and fast transactions. According to the Hyperledger Fabric website, it can process between 3,000 and 20,000 TPS, as opposed to Bitcoin, which can handle around 7 TPS, whereas Ethereum (the PoW version) executes around 20 TPS. In addition, Hyperledger Fabric is resilient to the 51% attack because there is no mining involved. Moreover, Hyperledger uses practical byzantine fault tolerant (PBFT) as a consensus mechanism, which does not require a lot of electricity or CPU power and eliminates the need for miners [32].

However, in Hyperledger Fabric, all participants have known identities. This can be viewed as a disadvantage in systems where anonymity is one of the requirements. On the other hand, knowing the identities of participating parties can be considered a good feature in blockchains where we want to know all the parties that are participating. Hyperledger Fabric is still a young project, released in July 2017. This can be an issue because newer projects are often less tested and more prone to bugs [32].

3.4. Hyperledger Fabric Components. The Hyperledger Fabric blockchain organizes a collection of nodes into organizations, resulting in a network that interfaces with external applications. Organizations (ORG) in HLF are regarded as members of the blockchain network. Each organization is identified by its membership service provider (MSP), which governs how new members receive digital signatures and are verified. Organizations, like in the real world, can be as large as needed. A consortium is formed by a group of organizations, but these organizations must be non-orderer organizations in the blockchain network [11].

Docker containers are used to run all the HLF components. To ensure the application's security, the container provides a sandbox environment that separates the application program from the physical resources and isolates the containers from each other. In HLF, all nodes must be authorized before joining the blockchain network [8]. Figure 1 shows the components of the Hyperledger Fabric network, which would be summarized in the following subsections.

3.4.1. Nodes. Nodes within the HLF network are managed by the membership service provider (MSP), which acts as the identity manager, providing valid digital signatures [44]. The nodes within the HLF network, as shown in Figure 1, can be categorized into three groups:

(i) Clients: The clients can be web/mobile applications, command-line interfaces (CLIs), or software development kits that interact with the HLF network by submitting transaction requests for execution and requesting transaction ordering. The operations of the clients come in two categories: the management category, which is primarily used to manage nodes, such as starting, stopping, and configuring nodes, and the chaincode category, which is primarily used for chaincode life cycle management, such as installation, instantiation, execution, and upgrading [8, 11]

(ii) Peers are the basic building blocks of any fabric network. Peers store the blockchain ledger and validate transactions before committing them to the ledger. Peers run the smart contracts which contain the business logic that is used to manage the assets on the blockchain ledger [44]. Invoking chaincodes allows applications to connect to peers and query or update ledgers. Peers are classified into two types: endorsers and committers. The endorser node is in charge of validating, simulating and endorsing transactions. The endorser node must hold a chaincode. The committer node is in charge of verifying transaction legitimacy as well as updating the blockchain and ledger status. The committer node may hold a chaincode [8]

(iii) Ordering service or orderer: The orderer acts as the network's communication backbone. The orderer establishes consensus in Fabric and is responsible for keeping the transaction's order [19]. The ordering service can be used with pluggable implemented actions like Apache Kafka or Fabric Raft

3.4.2. Channel. Data privacy and confidentiality are important requirements for most enterprise applications. HLF creates channels, which serve as a private layer of communication between network members. Channels are only accessible to organizations that have been invited to them and are invisible to other network members. Every channel has its own blockchain ledger. Organizations invited to the channel collaborate with their peers to validate and store transactions in the channel ledger [44]. Channels allow nodes in the network to communicate with one another privately and securely, ensuring nodes can send messages and blocks without outside interference. To increase confidentiality, a single network could have multiple channels between different organizations and peers [11]. As a result, HLF is a multichannel, multiledger, and multiblockchain system [8].

3.4.3. Ledger and World State. A ledger is a chain of blocks containing immutable transaction records. The peer nodes within the HLF network agree on the ledger state, and this ledger is distributed across the participating nodes. In HLF blockchain, the ledger comprises two related components: a world state and a blockchain. The world state contains the ledger's current value state, which means that the values can change as frequently as the ledger is updated. The world state is accessible as a database, with efficient storage and retrieval methods. There are two options for storing the world state: GoLevelDB and CouchDB [44]. In this work,

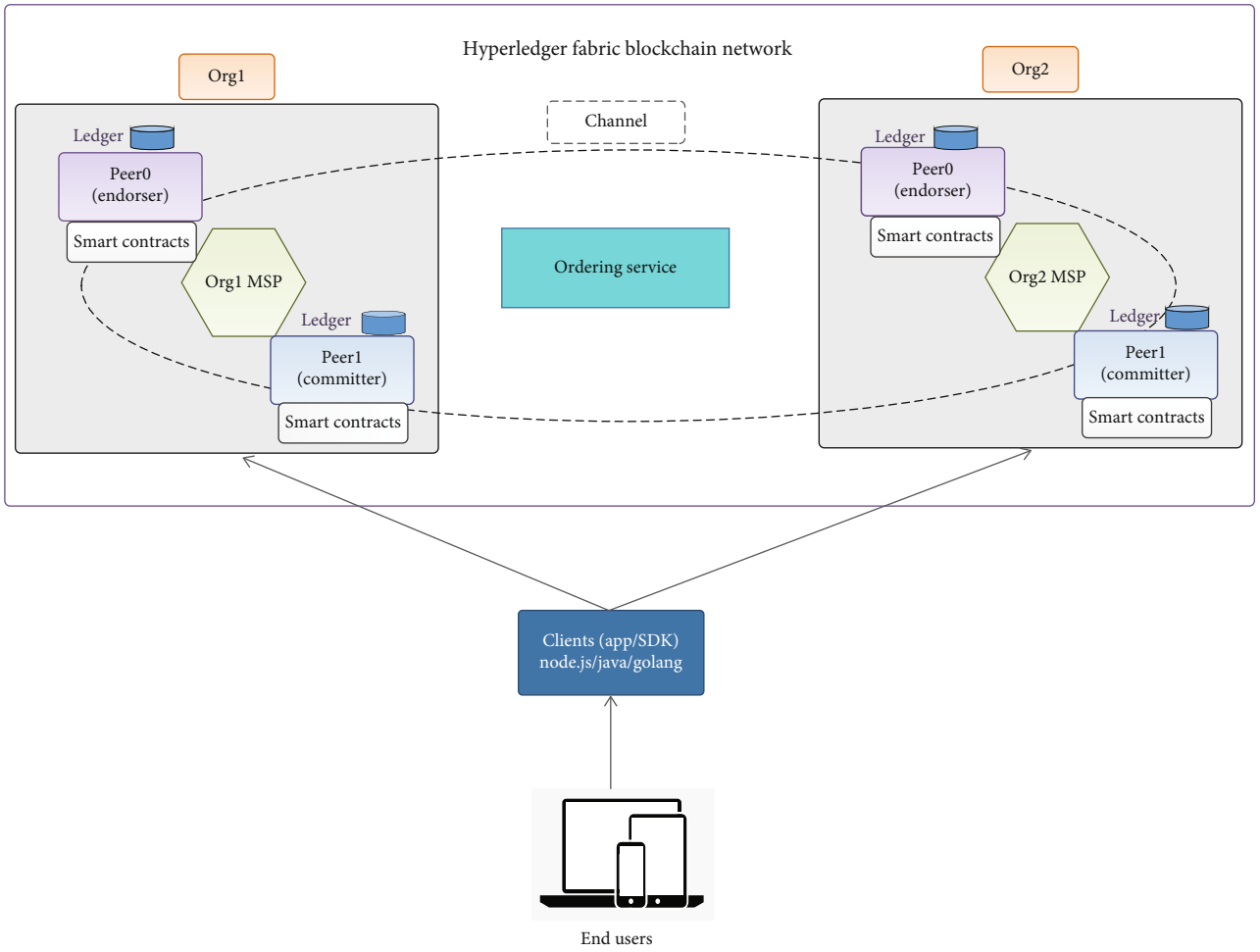

Figure 1: Hyperledger Fabric blockchain network.

we chose CouchDB as the state database (SDB). The blockchain is a chained structure of transaction blocks. The genesis block is the first block in a blockchain and is always the starting point of the ledger containing configuration transactions [44].

3.4.4. Smart Contracts/Chaincode. The Hyperledger Fabric makes use of smart contracts, and they are usually referred to as "chaincode," which is a collection of similar smart contracts. When a chaincode is deployed on a network, those contracts become accessible as applications. In HLF, smart contracts can be written in Go, JavaScript (node.js), and eventually other programming languages such as Java that implement a prescribed interface [44]. In this work, the smart contracts are written in Golang.

3.4.5. Membership Service Provider (MSP). In the HLF network, there are various entities, such as peers, orderers, applications, and administrators, that must have a valid network identity. Each HLF user has an X.509 certificate, which serves as a valid identity for each entity. These certificates are critical in determining who has access to which network resources. A certificate is only as trustworthy as the authority that issued it. The rules that determine which entities have valid identities are defined by the MSP [45].

The certificate authority (CA) issues certificates to all entities. All blockchain nodes must have a valid digital identity, which can be distributed by one or more CAs. The CA keeps both its private key and the entities' private keys private and makes its certificate, which includes its public key, publicly available. Each entity's certificate is required to be verified by the CA that signed the certificate with its private key [45].

3.5. Ordering Algorithms/Consensus Mechanisms. Ordering is defined as the act of structuring received transactions and creating a block out of those received transactions. Many existing blockchains, such as Ethereum and Bitcoin, take a probabilistic approach, in which any peer has an equal chance of adding data to the network. Each peer competes to find a hash collision by attempting various nonce values to obtain the desired hash. The node that receives the matched hash broadcasts its proposed block to all other peers along with the nonce. Although finding the hash requires a significant amount of computing power, verifying it is almost instantaneous. Once the peer has verified the authenticity of the block, it is added to their local ledger. However, this approach has many drawbacks such as power consumption and the possibility that more than one peer

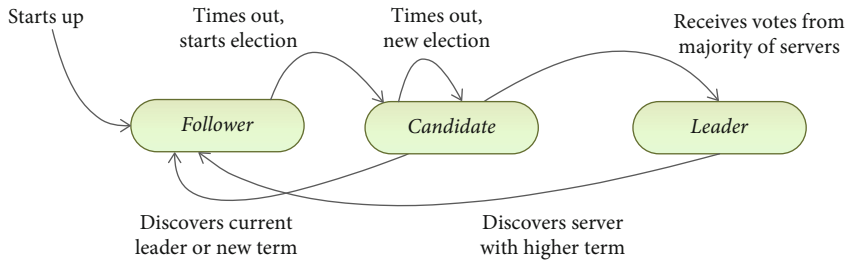

Figure 2: Raft process of election of a follower to candidate when no communication is received, then elected leader [47].

will find the hash solution at the same time; hence, we will have multiple branches in the blockchain ledger, which will result in forking [45].

Hyperledger, on the other hand, takes a deterministic approach. In Hyperledger, nodes that run the chaincode (smart contract) are known as endorsers, while nodes that create the block from the transactions are known as orderers. We can have a single node as the orderer or a group of nodes that form the ordering service. In Hyperledger consensus, any block generated by the orderer is considered final and valid, resulting in its deterministic nature. This determinism aids us in preventing ledger forking or branching, which gives Hyperledger a substantial performance benefit. Moreover, the separation of the endorser nodes (performing the execution of chaincode) and the orderer nodes (doing the ordering of the transactions) provide much better scalability and performance advantages over other blockchains [45].

In HLF, we have three types of implementations of the ordering service: Solo, Kafka, and Raft. In the Solo configuration, we have just one ordering node, which makes the network lose its advantage of being fault-tolerant. The Solo ordering service implementation is only used for test purposes. It is deprecated and may be completely removed in the future release of HLF. Existing Solo users should migrate to a single-node Raft network for equivalent functions. Kafka is a multi-node ordering service implementation provided for Hyperledger. Kafka follows a leader-follower configuration. It uses the Apache Zookeeper ensemble for managing various nodes. The additional administrative overhead of managing a Kafka cluster may be intimidating or undesirable to many users. Raft is the latest implementation provided in the Hyperledger Fabric from version 1.4.1. Raft is a crash fault-tolerant (CFT) ordering service built on the etcd Raft protocol implementation [44, 45].

Raft operates on a "leader and follower" model, with a leader node elected per channel and the followers replicating its decisions. Each node in the Raft consensus model is either a leader, a follower, or a candidate. The nodes' initial state is a follower, but if the follower does not receive the leader's heartbeat message during the election period, it changes to a candidate state. Normally, the candidate is used to elect the leader via a remote procedure call (RPC) request, and there is only one leader [46]. Once a leader node is chosen, all the other nodes follow it. When the leader goes down, a new leader is randomly chosen, and all the nodes then follow the newly elected leader, which makes the Raft ordering service CFT [44, 45]. The state transition of the Raft ordering service is depicted in Figure 2. In our design, we used the Raft ordering service to verify the transaction because it has the features of a faster and less complicated consensus.

The Raft ordering service is easier to set up and manage than the Kafka-based ordering service, and its design allows multiple organizations to contribute to a distributed ordering service [44, 45]. Apache's Kafka, which is also a CFT ordering service, is a third-party service. Raft, on the other hand, contains all of its components within the ordering node, so we only need to configure the orderer image. Raft, unlike Kafka, attempts to develop a Byzantine fault-tolerant BFT ordering service. Using the Raft ordering service, it is possible to achieve the root of trust models within the HLF network [11]. The Kafka-based ordering service is not fully decentralized, whereas a raft-based ordering service gets you one step closer to having a decentralized network, and you will eventually be able to have a fully decentralized Fabric network when the BFT ordering service does become available [44].

3.6. Attributed-Based Access Control (ABAC). Applying an optimal access control model to billions of IoT devices is difficult. Despite the extensive research, authentication and authorization issues are still in their early stages in the IoT environment. Numerous efforts have been made to adapt traditional access control models to meet new IoT security requirements. Some of the most commonly used access control mechanisms in IT infrastructure are capability-based access control (CBAC), role-based access control (RBAC), and attribute-based access control (ABAC) [9].

The CBAC methods rely heavily on centralized solutions, which can lead to a variety of issues [17]. OAuth-IoT [48] is a capability-based access control model that implements the OAuth protocol in the IoT domain. Despite its widespread success, CBAC introduces several challenges in the propagation and revocation of access rights [9]. Role-based access control (RBAC) is a popular method for limiting authorized users' access privileges [7]. However, because of the highly dynamic environment and a large number of IoT users, the role explosion problem of a pure RBAC model makes it unsuitable for IoT scenarios [6]. In IoT, the RBAC model was extended to a new model known as context-based access control [12]. Nevertheless, RBAC is unsuitable for IoT access control, which involves a large number of heterogeneous devices and a lack of standardization [1]. Furthermore, RBAC does not readily support

multifactor decisions (e.g., specialized training and decisions dependent on physical location) [2].

Attribute-based access control (ABAC) is one of the most suitable decentralized models for large-scale, flexible, and dynamic IoT scenarios. When used in IoT environments, ABAC can provide fine-grained and flexible control over each device's access requests [5]. ABAC is a method of making access control decisions without the subject's prior knowledge of the object or the object owner's knowledge of the subject [2]. To address the issue of role explosion in RBAC, the ABAC model directly associates attributes with subjects, and the user attribute certificate is used to grant access rights [9]. When compared to RBAC, attribute-based access control (ABAC) is scalable and more flexible, and it may provide fine-grained access control [7].

ABAC considers the attributes of the subject, object, permission, and environment to grant access. Based on the presence or absence of the required attributes in the request, it determines whether or not to grant access to the requester. To determine whether the requester has sufficient access privileges, a target may also use these attributes to express specified access policies. Because subject and object attributes are defined separately, ABAC can effectively separate access control and policy management [5, 8]. Moreover, by relying on the concepts of subject and object attributes that are consistently defined across organizations, ABAC eliminates the need for explicit authorization to be directly assigned to the individual subject before requesting to perform the operation on the object [2].

A detailed comparison of the three access control methods is summarized in Table 4.

4. The Proposed Model (ABAC-HLFBC)

Access control mechanisms for ABAC must be able to verify subject attributes, verify access control policies (rules), verify object attributes, and verify environmental conditions for a subject to be able to execute a policy on the object (e.g., allow or deny access). Almost always, the subject is assumed to be human. The subject could also be a non-person entity (NPE), such as a self-contained service or application. For scalability, the policy can be altered based on the current situation by adding or removing attributes. Furthermore, entity attributes can be defined from various perspectives to achieve fine-grained access control [2].

Attributes are considered the heart of ABAC. The attributes can be defined as a set of four elements: $A \in \{S, O, P, E\}$ [2], where:

(i) $\boldsymbol{A}$ is the attribute that has a key-value pair structure, $A = \{name : value\}$

(ii) $\boldsymbol{S}$ is the subject's attribute. It identifies the identity and characteristics of the entity that launches the access request, such as a person's ID, name, position, and age

(iii) $\boldsymbol{O}$ defines the object's attribute, which is the attribute of the accessed resource, such as service IP address, network protocol, and resource type

(iv) $\boldsymbol{P}$ defines the permission attribute. It represents the permissioned operation the subject can perform on the object, such as reading, writing, and executing

(v) $\boldsymbol{E}$ is the environment's attribute, which means the environment information at the time the access request is generated, such as the location and time [2]

Based on [8], a device resource model in this paper is defined as $\{Device\} \longrightarrow \{resource\} \longrightarrow \{url\}$. The access policy defines the subject's (user's) access rights to the object (resource). Instead of requesting resources directly from the device, the users get the resource data via a URL from the blockchain system after checking permission. The process is according to the following steps [8] as shown in Figure 3:

(1) The resource is published on the Internet by the device that generates the URL for that resource

(2) The URL of that resource is stored on the blockchain

(3) The users request authorization from the blockchain system based on attributes

(4) The blockchain distributes URLs to authorized users

(5) Using the URL, the users pull or download resource data from the Internet

The device access control policy model, when integrated with the ABAC model and the characteristics of data generated by IoT devices, can be defined as the following [8]:

(i)

$$P = \{AS, AO, AP, AE\}, \tag{1}$$

(ii)

$$AS = \{userId, role, group\}, \tag{2}$$

(iii)

$$AO = \{deviceid, MAC\}, \tag{3}$$

(iv)

$$AP = \begin{cases} 1, allow \\ 0, deny \end{cases}, \tag{4}$$

(v)

$$AE = \{createTime, endTime, allowedIP\}. \tag{5}$$

Table 4: Comparison between CBAC, RBAC, and ABAC mechanisms [17].

Access control approach	CBAC	RBAC	ABAC
Description	It employs a communicable and unforgeable authority token. The token refers to an object as well as a set of access rights associated with it	Predefined roles with a specific set of privileges are used. You must assign a role to the object to grant access	Policies are defined using a set of attributes selected from the user, subject, resource, and environmental attributes
Scalability	Scalability is enabled by using tokens only (token management is simpler and more efficient), but this can be a problem for complex systems where a user may handle tens of tokens, each of which represents an access right	Not scalable because it is impossible to predefine roles for billions of devices, which will result in numerous errors when assigning roles to rapidly changing devices	The access policies are defined on an attribute, which allows them to scale because, in a complex system or nested policies, the more granular your system is, the more efficient it is to handle billions of devices
Dynamicity	Moderate (the token needs to be changed every time the policy is changed)	Low (a role is not dynamic because it is predefined, and changing a role will have an impact on all associated devices)	High (a set of conditions defines access policies, which makes them dynamic and resistant to change)
Heterogeneity	High	Moderate	High
Granularity	Moderate	Low	High
Flexibility	High	Moderate	High
Lightweight	High	Moderate	Moderate

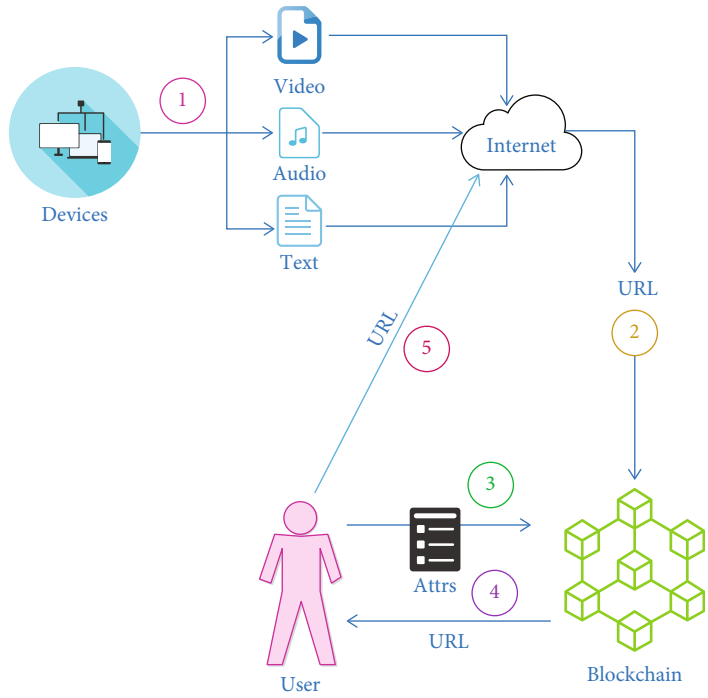

Figure 3: Connections between users and resources [8].

(i) ***P*** (policy) represents an attributed access control policy. There are four elements in this set: *AS*, *AO*, *AP*, and *AE*

(ii) ***AS*** (attribute of subject) defines the attributes of a subject (user) that include the userID, which uniquely identifies each user, user role, and user group

(iii) ***AO*** (attribute of object) defines the attributes of the object (resource), which include a MAC address and device ID. A unique identifier is used as a device attribute in this model. It is assumed that each device in the system has a single function and that each MAC or ID can only be linked to one resource URL at a time

(iv) ***AP*** (attribute of permission) specifies whether the user has access to resources. The value 1 represents "allow," while the value 0 represents "deny." When the AP is first configured, the default value is 1 (allow). Admins can revoke access authorization based on the circumstances by setting the value of AP to 0

(v) ***AE*** (attribute of environment) represents the characteristics of the environment that are required for access control. Attributes in AE include time, end time, and allowed IP. The term "time" refers to the period during which the policy is developed. The term "end time" refers to the policy's expiration date. When the current time is later than the end time, the policy is null and void. The goal of allowed IP is to prevent IP addresses from accessing the system from outside the network segment

Based on [8], ABACR (Attribute-based access control request) is defined as $ABACR = \{AS \wedge AO \wedge AP \wedge AE\}$, which is a collection of the four attributes listed above. It represents the AP of the AS on the AO of the AE.

ABACP (Attribute-based access control policy) is defined as $ABACP = \{AS \wedge \text{or} \vee AO \wedge \text{or} \vee AP \wedge \text{or} \vee AE\}$, which represents the subject's access control rules to the object. It expresses the required attribute set for accessing protected resources.

4.1. The Proposed Model Architecture. The proposed model is a blockchain-based access control system for the IoT, which is based on the benchmark [8], divided into four components: users (regular users or admins), blockchain, smart gateway, and devices, as illustrated in Figure 4 [8].

4.1.1. Users. Users in this system are categorized into two types: administrators and regular users. The admin user is in charge of managing the blockchain system and the smart gateway program. To gain access to the blockchain system, the administrator must provide a certificate. The regular user, the owner of the device, obtains the resource URL by sending attributed-based authorization requests to the blockchain [8].

4.1.2. Blockchain. The blockchain is the heart of the proposed model. Before joining the blockchain system, all nodes must obtain certificate authority (CA) authentication. The blockchain is built on Hyperledger Fabric, which uses smart contracts to control access. The blockchain exposes an API

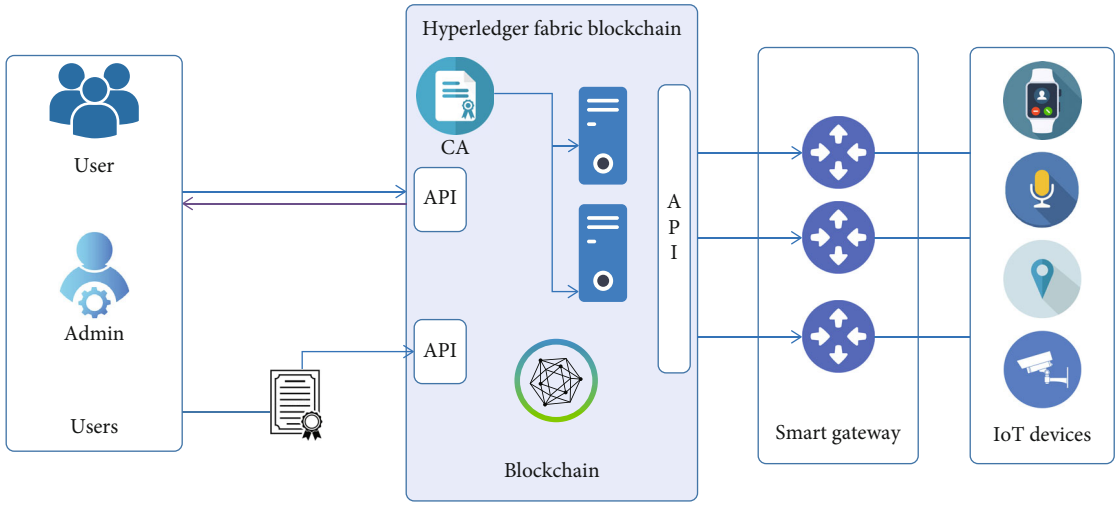

Figure 4: Proposed model architecture [8].

to users and smart gateways. It primarily performs three functions, which are as follows [8]:

(1) Storing the devices' URL resource data

(2) The management of attribute-based user rights

(3) User authentication for resource access

Unlike [8], which uses the Kafka ordering service with one orderer, the proposed blockchain follows the Raft ordering service with five orders, as shown in Figure 5 which shows the network topology of the prototype network.

4.1.3. Smart Gateway. The smart gateway is used as a connection between the devices and the blockchain system. The smart gateway can receive a message from a device and send the URL in that message to the blockchain, which can avoid the strain on the blockchain system caused by direct device access [8].

4.1.4. IoT Devices. IoT devices generally lack strong computing capabilities, adequate storage, and a long-lasting battery. As a result, it is impossible to deploy IoT devices directly as blockchain peer nodes [8]. A copy of the blockchain must be stored on each node in a blockchain network. Nevertheless, the blockchain can grow to be quite large and will continue to grow in size over time. Because of their limitations, most IoT devices will be unable to store blockchain data [13]. As a result, in the proposed model, IoT devices will not be part of the blockchain. The proposed design of IoT devices is independent of the blockchain network's consensus process, which significantly reduces the overall computation and communication overhead.

4.2. The Proposed Model Configurations

4.2.1. Smart Contract Configurations. The smart contract is the essence of the access control implementation. Based on [8], there are three types of smart contracts in this model: policy contract (PC), device contract (DC), and access contract (AC). The policy contract (PC) includes methods for running ABACP such as AddPolicy (), UpdatePolicy (), DeletePolicy (), and QueryPolicy (). The device contract (DC) is primarily responsible for storing the device's resource URL in SDB. There are two input parameters for DC *{DeviceId: URL}*. The access contract (AC) has the CheckAccess() function, which is the main function to accomplish access control management [8].

(1) Policy Contract (PC). It includes the following methods for running ABACP.

(i) AddPolicy (): PC calls AddPolicy () to add ABACP to the SDB, at that point all-action records are written to the blockchain

(ii) UpdatePolicy(): In some cases, the administrator must modify ABACP. The function UpdatePolicy() implements the interface for updating SDB, and the updating operation record is also written to the blockchain. UpdatePolicy() is similar to AddPolicy() in that it also calls the application interface's put method to overwrite the old value

(iii) DeletePolicy(): ABACP has an expiration date and can be canceled by the administrator. There are two scenarios in which deletion operation occurs. One instance occurs when the administrator actively deletes a policy by invoking this function. In the other instance, if the attribute is expired when the "end-Time" is reached, it will invoke DeletePolicy() function in the PC to delete the associated policy

(iv) QueryPolicy(): It implements the database querying interface, which includes a function for obtaining ABACP for other chaincodes. CouchDB was chosen as the SDB. Despite being a key-value document

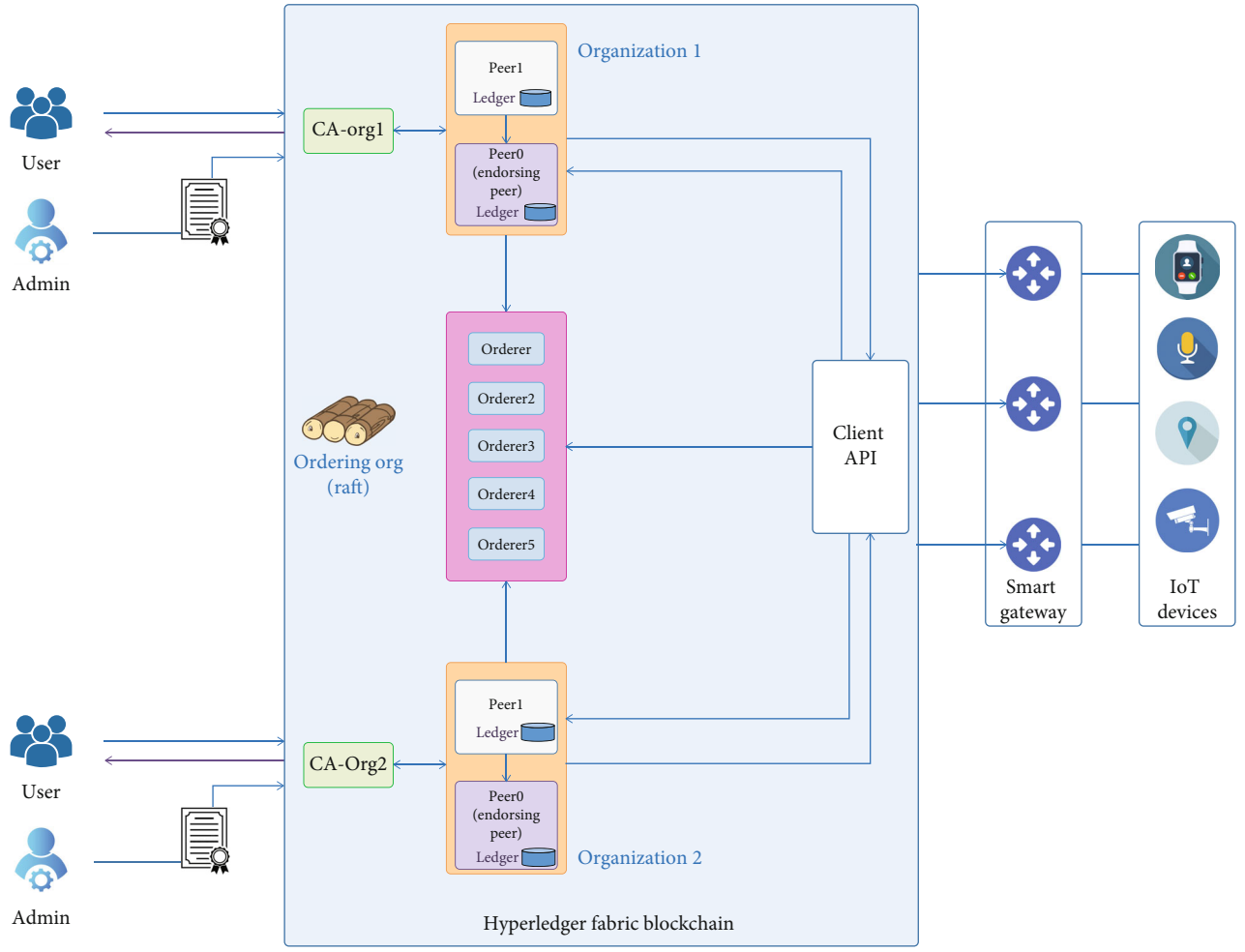

Figure 5: The proposed model (ABAC-HLFBC) with raft orderers.

database, CouchDB, like MongoDB, supports complex queries. In that case, QueryPolicy() allows us to query ABACP by AS or AO

(2) Device Contract (DC). DC is primarily responsible for storing the device's resource URL in SDB. There are two input parameters for DC *{DeviceId: URL}*. The following are the available functions:

(i) AddURL(): It stores DeviceId as a key and URL as a value in SDB

(ii) GetURL(): This method retrieves the corresponding URL value from SDB based on the DeviceId

(3) Access Contract (AC).

(i) CheckAccess(): It is the core function to achieve access control management. AC checks to see if the user's ABACR matches the ABAC policy. The requested data, like PC, is signed by the user's private key, and then AC verifies the signature to confirm the user's identity using the user's public key

ABAC is used in Hyperledger Fabric to restrict access to a specific user who has the required attributes in their certificate [49]. Hence, on the three smart contracts, we have added a function in the proposed model, which is part of the "github.com/hyperledger/fabric/core/chaincode/lib/cid" library, that implements the ABAC of HLF for restricting access to the specific user having the necessary attributes in their certificate. We have used *GetAttributeValue* to get the X509 certificate of the client, or nil if the client's identity was not based on the X509 certificate. We provide the attributes, which is an array of key-value pair objects when registering users with fabric-ca-client. The following steps are summarizing the added function:

(1) An admin (registrar) is enrolled in the CA. Then, the admin receives the signing key and certificate

(2) The admin then registers the client into the CA with proper information. The CA returns with a secret

```
Policies:
      Readers:
        Type: Signature
        Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
      Writers:
        Type: Signature
        Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
      Admins:
        Type: Signature
        Rule: "OR('Org1MSP.admin')"
      Endorsement:
         Type: Signature
         Rule: "OR('Org1MSP.peer')"
```

Figure 6: Org1 policies [13].

```
Policies:
      Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
      Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
      Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"
      # BlockValidation specifies what signatures must be included in the block
      # from the orderer for the peer to validate it.
      BlockValidation:
        Type: ImplicitMeta
        Rule: "ANY Writers"
```

Figure 7: Orderer organization policy [13].

(3) This secret is then used to enroll the client in the CA. The result is the signing key and certificate for that client

(4) Create a client identity instance

(5) Get the MSPID from the invoker as a string

(6) Get a specific attribute value as a string

(7) Verify a specific attribute through an assertion

4.2.2. Policy Configuration and Implementation. When accessing or updating network and channel settings, the Hyperledger Fabric policies specify the rules that must be followed. These policies specify who can and cannot modify which elements or configurations, as well as how these changes will be carried out. The initial set of policies is defined by the consortium as the entity in charge of providing a fine-grained access control foundation over the various components of the blockchain network. These policies are stored in the *configtx.yaml* file and are included in the genesis block [19]. The policies in the fabric network are organized in a hierarchy. Each policy is given its own section. Figure 6 depicts our Org1 policies (Org2 also has the same policies). Readers is the policy name, and Signature is the policy type. The Readers policy rule is "OR," which means that anyone from Org1MSP.admin, Org1MSP.peer, or Org1MSP.client can read the channel. Our Orderer organization policies are depicted in Figure 7.

4.3. Performance Metrics. Table 5 shows the metrics used to measure and compare the performance of the proposed model and fabric-iot.

5. Execution Environment

The experiments were carried out on a PC with an Intel i7-5600U at 2.60GHz and 16 GB of RAM. It was implemented on Ubuntu, installed on a virtual machine with 8 GB of RAM. In our experiment, we used a Nodejs-SDK to deploy a blockchain client, and the client connecting to the peer nodes was used to invoke the chaincode. The chaincodes have been implemented in Golang and are deployed on all peers. The test codes were written in Golang. We also tested the model using the Hyperledger Caliper tool. The software that has been used is listed in Table 6. We built our experiment according to the fabric-iot [8] experiment. The source code of the fabric-iot project is open-source on GitHub [52].

Table 5: Performance metrics.

Metric		Definition	Equation
Cost time		The cost time measure counts the processing time that the blockchain takes with different numbers of concurrent requests [8]	*Cost time = end time – Start time*
Latency	Read latency	The time between submitting a read request and receiving a response [50]. This metric is used in query operations	*Read latency = time when* the *response received – Submit time*
	Transaction latency	The time it takes for the effect of a transaction to be usable across the network. This measurement includes the time from the point that the transaction is submitted to the point that the result is widely available in the network [50]	*Transaction latency* = (*confirmation time* × *network threshold*) – *Submit time*
Throughput	Read throughput	The number of reading operations completed in a given time period [50] which is expressed in reads per second (rps). Read throughput is not used as a primary performance metric for the blockchain [51]. This metric is used in query operations	*Read throughput = Total read operations/total time* in *seconds*
	Transaction throughput	Transaction throughput is the rate at which the blockchain commits valid transactions in a given time period [51], which is commonly expressed in transactions per second (TPS), but can also be expressed in minutes (TPM) or hours (TPH) [50]. Transaction throughput is measured not at a single node but across the entire network [51]	*Transaction throughput = Total committed transactions/total time* in *seconds*

Table 6: Experiment software configuration.

Software	Version
Ubuntu	20.04.1-desktop-amd64
Hyperledger Fabric	1.4.3
Docker	20.10.8
Docker compose	1.13.0
Node	16.6.2
NPM	7.20.3
Golang	go1.13.8 linux/amd64
git	2.33.0
Python	2.7.18
Hyperledger Caliper	0.3.0

Table 7: Proposed model containers.

Type of the node	Number
Orderer nodes	5 (raft)
Peer nodes	4 (two peers for each organization)
CouchDB nodes	4 (one for each peer node)
CA node	2 (one CA for each organization)
cli fabric tool	1
Device contract	4 (one in each peer)
Policy contract	4 (one in each peer)
Access control contract	4 (one in each peer)

5.1. Use of Dockers and Network Architecture. Figure 5, shown previously, depicts the topology of the Hyperledger Fabric network that has been implemented. There are two organizations defined (Org1 and Org2), each with two peer nodes. Peer0, the endorsing peer in each organization, serves as the anchor node, bridging communications between organizations and Peer1 which is the committing peer. Two certificate authorities (CAs) work as gateways to authenticate the parties joining the blockchain network.

We use Docker containers to configure various entities in the blockchain network. For our blockchain network, we have container images for the following entities:

(i) Orderer containers

(ii) Peer containers

(iii) Fabric CA containers

(iv) CouchDB containers

(v) cli container to access all the peers in the network

```
elhamshammar@ubuntu:~/Desktop/fabric-iot-master/network$ docker ps -a --format="table {{.Names}}\t{{.Image}}\t{{.Status}}"
NAMES                       IMAGE                                  STATUS
cli                         hyperledger/fabric-tools:latest        Up 12 minutes
peer0.org1.fabric-iot.edu   hyperledger/fabric-peer:latest         Up 12 minutes
peer1.org1.fabric-iot.edu   hyperledger/fabric-peer:latest         Up 12 minutes
peer1.org2.fabric-iot.edu   hyperledger/fabric-peer:latest         Up 12 minutes
peer0.org2.fabric-iot.edu   hyperledger/fabric-peer:latest         Up 12 minutes
orderer4.fabric-iot.edu     hyperledger/fabric-orderer:latest      Up 12 minutes
couchdb1                    hyperledger/fabric-couchdb             Up 12 minutes
couchdb3                    hyperledger/fabric-couchdb             Up 12 minutes
orderer3.fabric-iot.edu     hyperledger/fabric-orderer:latest      Up 12 minutes
ca_peerOrg2                 hyperledger/fabric-ca:latest           Up 12 minutes
ca_peerOrg1                 hyperledger/fabric-ca:latest           Up 12 minutes
couchdb0                    hyperledger/fabric-couchdb             Up 12 minutes
orderer.fabric-iot.edu      hyperledger/fabric-orderer:latest      Up 12 minutes
couchdb2                    hyperledger/fabric-couchdb             Up 12 minutes
orderer5.fabric-iot.edu     hyperledger/fabric-orderer:latest      Up 12 minutes
orderer2.fabric-iot.edu     hyperledger/fabric-orderer:latest      Up 12 minutes
elhamshammar@ubuntu:~/Desktop/fabric-iot-master/network$
```

Figure 8: Docker container images.

```
elhamshammar@ubuntu:~$  docker ps -a --format="table {{.Names}}\t{{.Image}}\t{{.Status}}"
NAMES                                IMAGE                                                                                                STATUS
dev-peer1.org2.fabric-iot.edu-dc-1.0   dev-peer1.org2.fabric-iot.edu-dc-1.0-141cf18778cdc663614ca843ca9efc11154295cc6d8c1ac8118bed4dea03d318   Up 8 minutes
dev-peer0.org2.fabric-iot.edu-dc-1.0   dev-peer0.org2.fabric-iot.edu-dc-1.0-2d2a86eabf0111ec43332ff0ed0476c8a1c38cc0bb5d8ea40184b8ad3e2d0a87   Up 8 minutes
dev-peer1.org1.fabric-iot.edu-dc-1.0   dev-peer1.org1.fabric-iot.edu-dc-1.0-379929899865d062d50679a0aa931f1ac434ca0b65530277e14417feb71796c1   Up 8 minutes
dev-peer0.org1.fabric-iot.edu-dc-1.0   dev-peer0.org1.fabric-iot.edu-dc-1.0-db399fac0dec9de6d317c5b86f5810451beeb5f28abe1be98814273b130b7741   Up 8 minutes
dev-peer1.org2.fabric-iot.edu-pc-1.0   dev-peer1.org2.fabric-iot.edu-pc-1.0-740d3c4c9eeb49907ecb1c2c66a003628d90b97d5f3537e57a5fb3651a184450   Up 9 minutes
dev-peer1.org1.fabric-iot.edu-pc-1.0   dev-peer1.org1.fabric-iot.edu-pc-1.0-f522331bf6784cfd02949e3904ab20b24e5102a67f918b51edb11e942284482d   Up 9 minutes
dev-peer0.org2.fabric-iot.edu-pc-1.0   dev-peer0.org2.fabric-iot.edu-pc-1.0-e0d09f6b54e529a8a7eb360dd3aec96e19dbaa31e67e8a9e2ffe3fff724b7000   Up 9 minutes
dev-peer0.org1.fabric-iot.edu-pc-1.0   dev-peer0.org1.fabric-iot.edu-pc-1.0-d54b6d4d48db0470fc650c016b355929d048df74cd45c329364466589d868df5   Up 9 minutes
dev-peer1.org1.fabric-iot.edu-ac-1.0   dev-peer1.org1.fabric-iot.edu-ac-1.0-94e2c7b16c37b881da1a18ab774a438e931a35221f74fb7ab1d1fcce157a055e   Up 9 minutes
dev-peer0.org2.fabric-iot.edu-ac-1.0   dev-peer0.org2.fabric-iot.edu-ac-1.0-4f17d3fa78c7cab00728b99899f97ad3d6a34b02aa48b4dfbba9f25c42608567   Up 9 minutes
dev-peer1.org2.fabric-iot.edu-ac-1.0   dev-peer1.org2.fabric-iot.edu-ac-1.0-59fa85591072fe7be08791da848b613e5efa0ddde675193cfbcbe3f405d5a5f5   Up 9 minutes
dev-peer0.org1.fabric-iot.edu-ac-1.0   dev-peer0.org1.fabric-iot.edu-ac-1.0-0685b04405bc6a824a6bdb49968d8d465adf1b113fcef514ebe48958d4798aea   Up 10 minutes
```

Figure 9: Smart contract containers.

All these containers are part of a single Docker network. The chaincodes are also executed in separate containers.

The blockchain network was built with Hyperledger Fabric v1.4.3. The prototype network is made up of five order nodes, two CA nodes, four peer nodes, the cli fabric tool, and one channel. The chaincodes are installed and instantiated in all peer nodes. Table 7 shows all the containers in this model.

5.2. Raft Orderer. We made modifications to the settings in the *configtx.yaml* file which will be embedded in the genesis block for the channels. This file contains the settings of the etcdraft and the consenters of five orderers. The orderers' configuration is written in *Docker-compose-etcdraft.yaml* as in [53]. In the compose files, we added the containers for the five orderers.

5.3. Initialization of the Proposed Model and Processes Sequence. Step 1: Generating the root certificates and secret key pairs for the nodes (peer, orderer, etc.) using the Hyperledger *cryptogen* tool

Step 2: Creating a genesis block using *configtxgen* tool. The genesis block is used to bundle transactions that include node and channel configuration. When the model is operational, the genesis block is written into the blockchain, ensuring that the identity information of all the nodes in the system cannot be changed. The images of the remaining nodes will then be started in accordance with the Docker-compose initialization configuration. When all containers have been successfully completed, the peer nodes are added to the channel

Step 3: Creating the Docker images. The list of containers of the proposed model is shown in Figure 8 (the orderer containers are surrounded by the red rectangle)

Step 4: Creating the channel and joining all peers to the channel. We have to mention that there is only one channel (iot-channel) in the proposed model

Step 5: Installing and instantiating smart contracts. Figure 9 shows the 12 containers after installing and instantiating the three smart contracts

The whole system processes sequence is shown in Figure 10 [8].

5.4. Fabric Client. In the Hyperledger Fabric platform, there are several methods for invoking chaincode. Outsiders can invoke chaincode on the HLF platform using a client or SDK (Java, Golang, or Node). The proposed model employs a client written by the node SDK to invoke chaincode. As shown in Figure 11, the first file, "test.js," is used as a "gateway" that receives access requests to execute smart contract functions. The second file, "invoke.js," allows generating transactions when interacting with the chaincode through the "invoke" function. "enrollAdmin.js" and "registerUser.js" are used to register users.

6. Results Study and Evaluation

In this section, a comparison is made to evaluate the performance of the proposed model and the fabric-iot model [8]. Section 6.1 evaluates the performance of the two models in terms of cost time, whereas Section 6.2 evaluates the latency

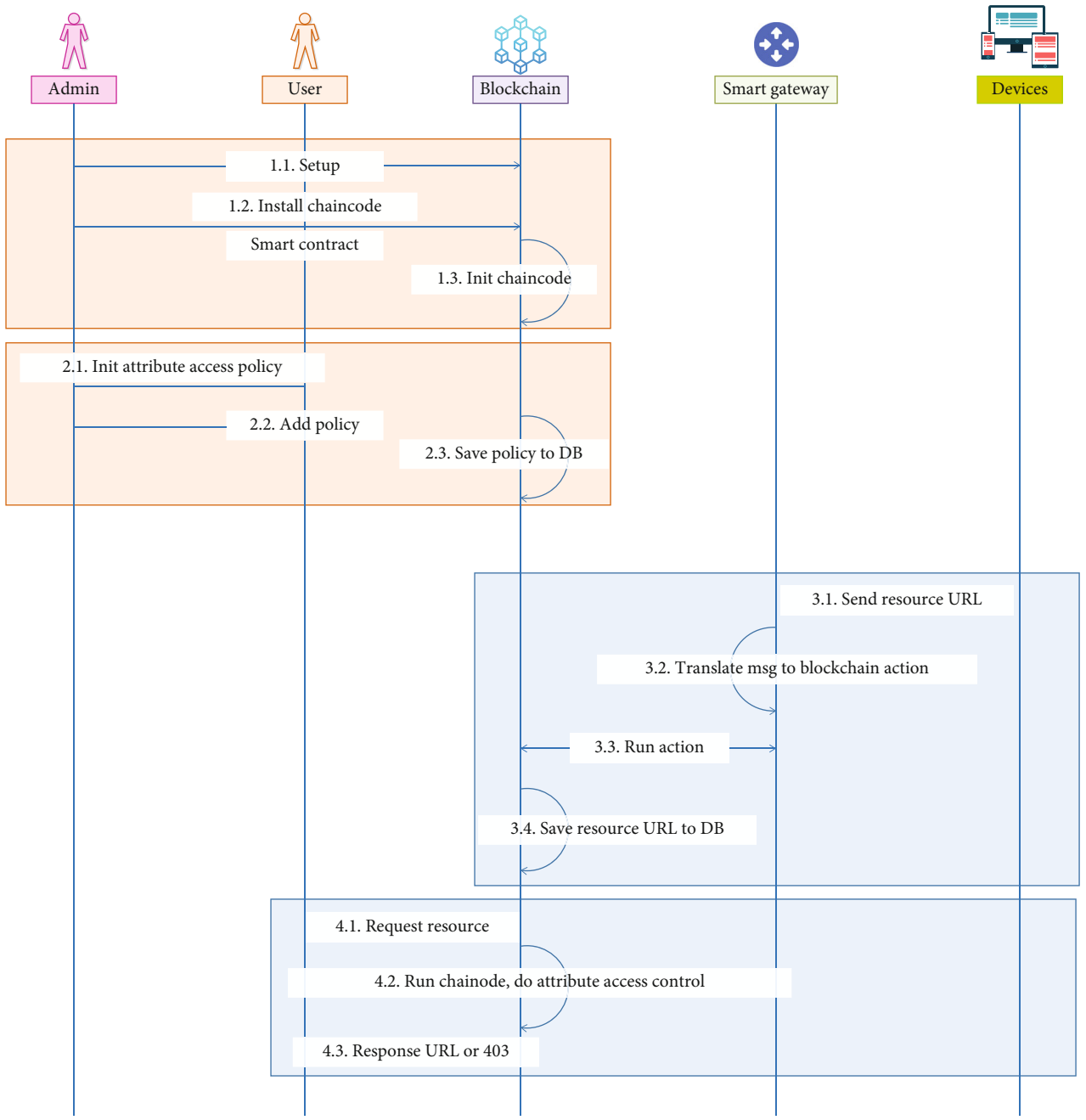

Figure 10: The proposed model process sequence [8].

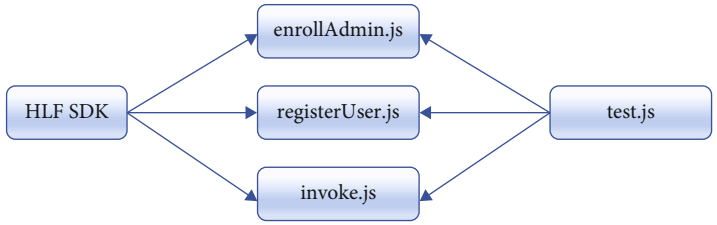

Figure 11: Fabric client.

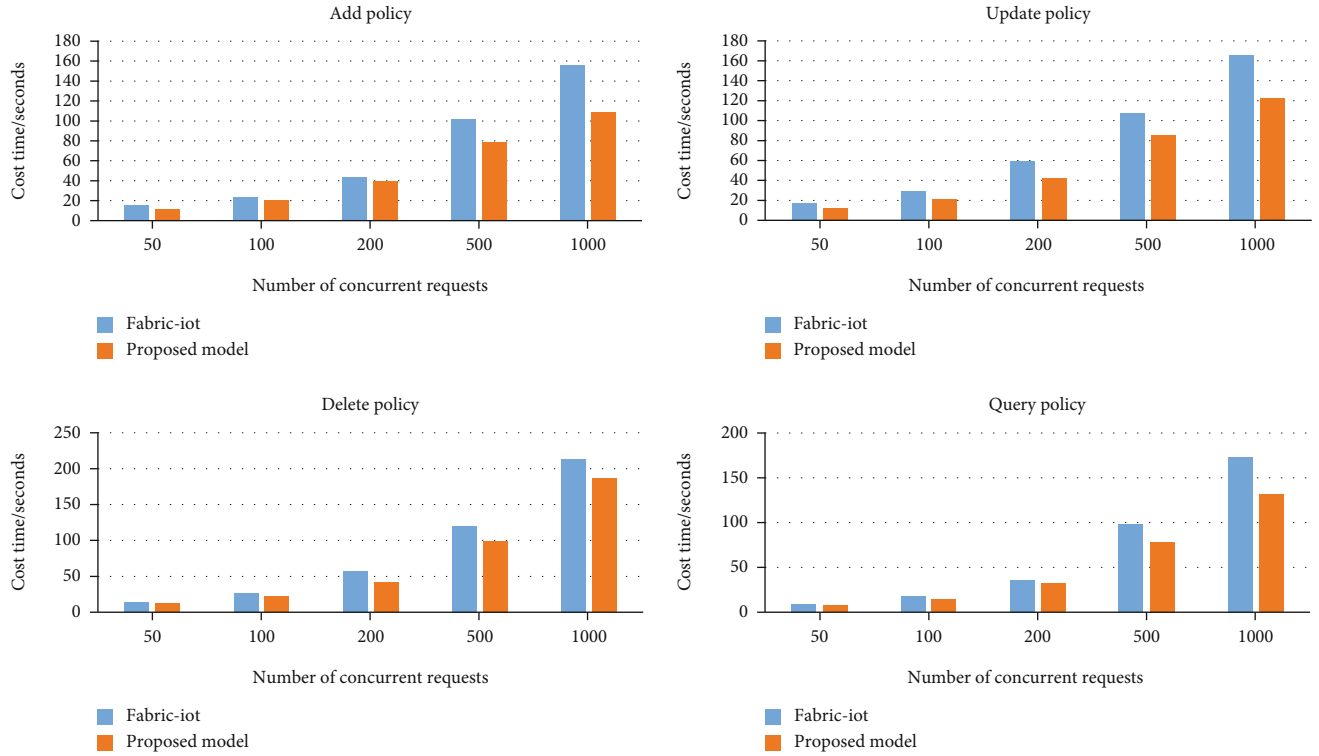

Figure 12: PC cost time.

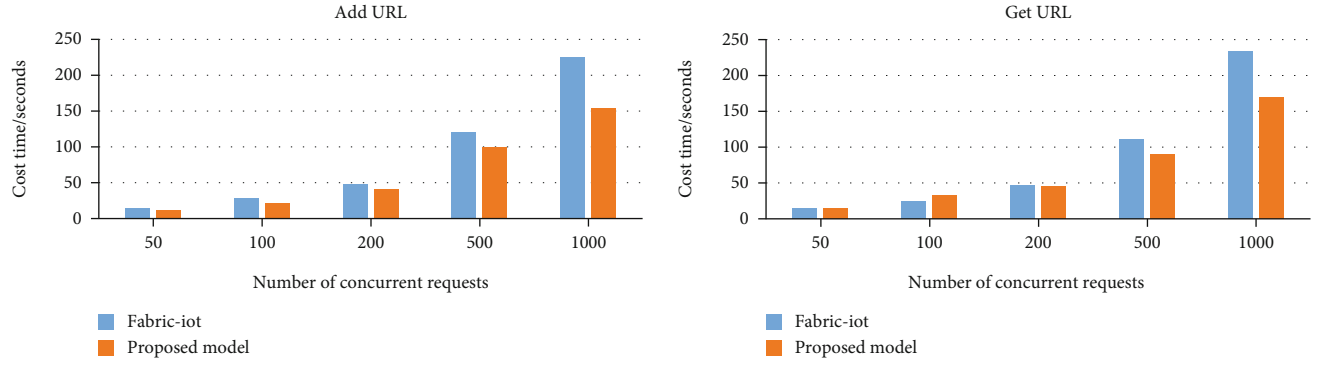

Figure 13: DC cost time.

and throughput using the Hyperledger Caliper benchmark tool. The general discussion is introduced in Section 6.3.

6.1. Cost Time Evaluation. For testing the cost time, the test program is written in Golang to send the requests to the Nodejs API to call the smart contract functions. The number of concurrent requests is set to 50, 100, 200, 500, and 1000, respectively. The consumed time to complete each request is recorded. Since the response time results are different for each request operation, we sent each request 10 times, and then we took the average of the response time of each request. The statistical results for each operation are shown in Figures 12, 13, and 14.

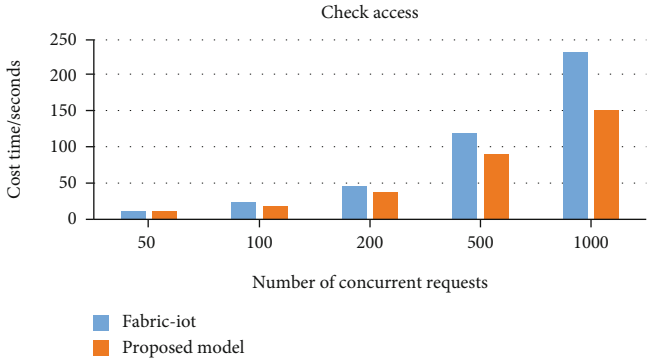

Figure 14: AC cost time.

HYPERLEDGER CALIPER

Caliper report

Basic information

DLT: fabric
Name: simple
Description: This is an example benchmark for caliper, to test the backend DLT's performance with simple account opening & querying transactions
Benchmark Rounds: 13
Details

Benchmark results

Summary
AddPolicy

System under test

Version: 1.4.3
Size: 2 Orgs with 2 Peer
Orderer: Raft
Distribution: Single Host
StateDB: CouchDB
Details

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
AddPolicy	10	0	5.6	2.18	0.68	1.58	4.0
AddPolicy	20	0	5.3	2.07	0.20	1.13	4.9
AddPolicy	30	0	5.2	2.27	0.20	1.20	5.0
AddPolicy	40	0	5.1	2.17	0.17	1.15	5.0
AddPolicy	50	0	5.1	2.08	0.19	1.13	5.0
AddPolicy	60	0	5.1	2.09	0.17	1.13	5.0
AddPolicy	70	0	5.1	2.10	0.16	1.11	5.0
AddPolicy	80	0	5.1	2.60	0.15	1.27	5.0
AddPolicy	90	0	5.1	2.12	0.16	1.15	5.0
AddPolicy	100	0	5.1	2.35	0.16	1.18	5.0
AddPolicy	200	0	5.0	2.98	0.17	1.26	5.0
AddPolicy	500	0	5.0	2.77	0.16	1.22	5.0
AddPolicy	1000	0	5.0	2.48	0.14	1.14	5.0

Figure 15: Caliper report of AddPolicy.

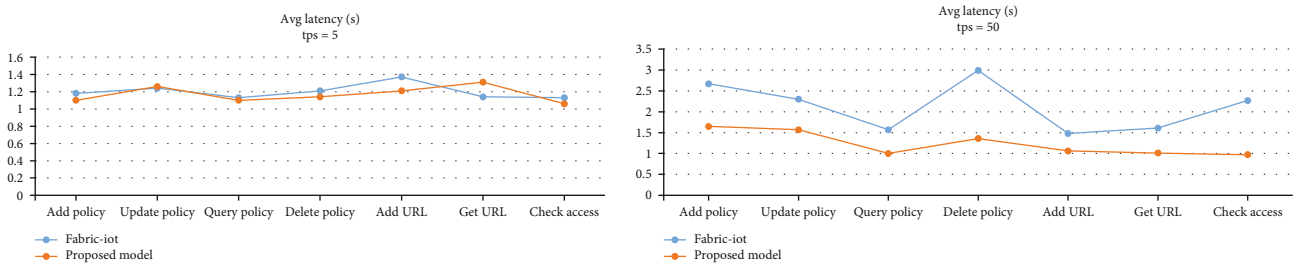

Figure 16: Avg latency (s).

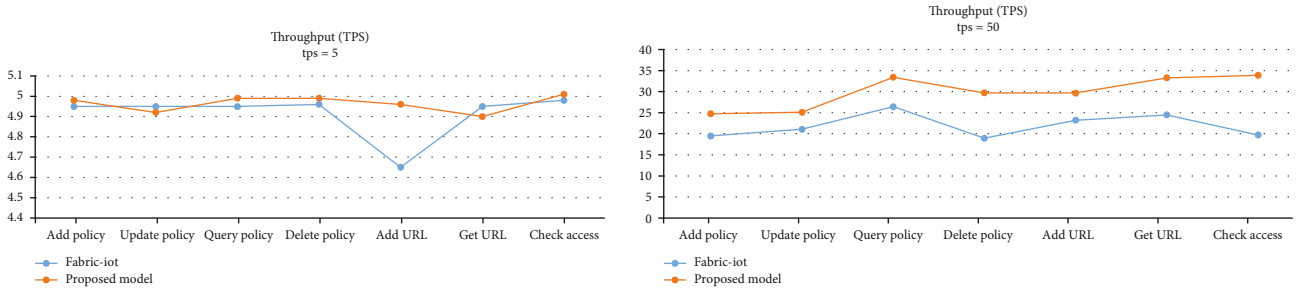

Figure 17: Throughput (TPS).

As shown in these figures, there is an overall improvement in the cost time in all smart contract operations in comparison with the benchmark (fabric-iot).

6.2. Latency and Throughput Evaluation. We used the Hyperledger Caliper benchmark tool to test the latency and throughput of the proposed model. To get rid of the consistency problems, the Hyperledger Caliper docker image has been used. Caliper is published as the Hyperledger/caliper Docker image, providing a single point of usage for every supported adapter. The image is built upon the node:10.16-alpine image to keep the image size as low as possible. We used a containerized caliper available on GitHub [54]. Figure 15 shows a sample of the caliper report of the AddPolicy function.

As shown in Figure 15, we have tested the fabric-iot and the proposed model by changing the number of transactions (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 500, and1000); then, we took the average of latency and throughput. We have tested the two models in two scenarios. In the first one, we set TPS = 5, and in the second, we set TPS = 50. The latency for both models is shown in Figure 16.

As shown on the left side of Figure 16, in the first scenario, when TPS = 5, the AddPolicy, DeletePolicy, AddURL, and CheckAccess operations of the proposed model have lower latency than the fabric-iot model. However, in UpdatePolicy and QueryPolicy operations, both models do not have much difference in latency. In GetURL, the proposed model has higher latency. In the second scenario, when TPS = 50 (right side), the proposed model has better results where the latency of the proposed model is lower than fabric-iot. We have to mention that the unstable results of the first scenario, when TPS = 5, are due to the small numbers of TPS in the Hyperledger Caliper testing scenarios.

Figure 17 shows the presentation of the caliper throughput reports of smart contract operations. As in latency, in

the first scenario (left side), some of the results show no improvement in the throughput. Most of the results have a throughput number that ranges between 4.6 and 5, which is close to the TPS number (TPS = 5). However, in the second scenario (right side), when TPS = 50, the proposed model shows better throughput than fabric-iot.

6.3. General Discussion. As shown by the results in Sections 6.1 and 6.2, the ABAC-HLFBC proposed model outperforms the fabric-iot model [8] in terms of cost time, latency, and throughput.

As shown in Figures 12–14, the cost time of the proposed model and the benchmark (fabric-iot) is approximately the same when the number of concurrent requests is small (for example, 50). Nevertheless, the proposed model outperforms the benchmark when the number of transactions increases. Moreover, the proposed model outperforms the fabric-iot when the number of TPS in the Hyperledger Caliper testing scenario is high (TPS = 50), as shown in Figures 16 and 17. This improvement in the results is due to the faster and less complicated consensus and higher throughput of the Raft ordering service. However, when the number of TPS = 5, the latency and throughput results are unstable, and there is no significant improvement in the proposed model. We have to mention that the unstable results of the first scenario, when TPS = 5, are due to the small number of TPS in the Hyperledger Caliper testing scenarios.

There are some limitations to this work that should be discussed in this subsection. The fact that all of the benchmark experiments are performed on a single-host virtual machine is one of the study's limitations. In a production environment, however, these nodes can be distributed across multiple locations. Furthermore, in order to simplify implementation, we only consider software-based configurable network components.

7. Conclusions and Future Work

This work proposed a blockchain-based access control model (ABAC-HLFBC) that employs the blockchain as the trusted center of the access control model and implements the access control policy through the use of smart contracts. The data stored on the blockchain is trustworthy and credible because of its tamper-proof and no single point of failure features. The proposed model is fully decentralized (no third-party required), user-friendly, user-transparent, fault-tolerant, scalable, and compatible with a wide range of IoT access control models.

The proposed ABAC-HLFBC model utilizes smart contracts to achieve a flexible, scalable, and fine-grained access control process based on the ABAC method. Hyperledger Fabric is used as the blockchain platform for executing smart contracts, making the system more user-friendly while fully utilizing the blockchain's characteristics. Without a doubt, Hyperledger Fabric is one of the most promising distributed ledger platforms. Whether in academia or industry, extensive analysis of the performance of this well-known platform has already been provided. Instead of the Kafka ordering service, which is used in fabric-iot [8], the proposed model used a Raft ordering service. Raft is a crash-fault-tolerant (CFT) ordering service based on the Raft protocol implementation. Raft, unlike Kafka, attempts to develop a byzantine fault-tolerant BFT ordering service. The proposed ABAC-HLFBC model incorporated ABAC of Hyperledger Fabric that supports ABAC for chaincode and included the endorsement policy in the configurations. The proposed model proof of concept has been tested, and the experimental results showed that the proposed model has better cost time, latency, and throughput than fabric-iot. This improvement in the results is due to the faster and less complicated consensus and higher throughput of the Raft ordering service.

As a vital conclusion, the blockchain faces several critical challenges while providing IoT data security. An analysis of the main challenges of the blockchain and IoT integration should be conducted for successful integration. Recently, there has been a significant amount of industry investment and interest from academia to solve major research challenges in blockchain technology, which can be summarized as follows: (1) Although blockchain-based access control has emerged as a promising security technology, it still suffers from high latency in the consensus process and poor adaptability to dynamic changes in the network environment. (2) Blockchain is not designed to store large amounts of data, which typically necessitates the proper integration of on-chain and off-chain databases to handle specific tasks [17]. (3) Performance and scalability have always been major issues with blockchain technology. Regardless of recent improvements in transaction execution and validation performance by introducing lighter consensus mechanisms and more efficient transaction schemes as in Hyperledger Fabric, the performance and scalability of blockchain-based access control solutions cannot compete with current centralized solutions [17]. (4) Even if blockchain can guarantee the immutability of data in the chain and identify transformations, data that enters the blockchain corrupted remains corrupted. Hence, it is necessary to check the data before entering the blockchain. (5) Some IoT devices may be found in public places. How can blockchain be used to ensure the security and privacy of the data stored on the IoT devices that are physically under the control of an adversary?

For future work, first, we are planning to implement the model in more than two organizations and more than one channel and assign the access control attribute according to the organization. Second, we are planning to test the security of the system against access control attacks such as forgery attacks, injection attacks, and man-in-the-middle attacks. Third, for a better test of the proposed model, we are planning to use IoT physical devices to test the reliability and throughput of the system. Finally, physical infrastructure can have an impact on blockchain performance; thus, in future work, we intend to implement the proposed model on platforms on more than one PC.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. A. Islam and S. Madria, "A permissioned blockchain based access control system for IOT," in *2019 IEEE international conference on Blockchain (Blockchain)*, pp. 469–476, Atlanta, GA, USA, 2019.

[2] S. Figueroa, J. Añorga, S. Arrizabalaga, I. Irigoyen, and M. Monterde, "An attribute-based access control using chaincode in RFID systems," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, Canary Islands, Spain, 2019.

[3] 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 *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, Singapore, 2017.

[4] E. A. Shammar and A. T. Zahary, "The Internet of Things (IoT): a survey of techniques, operating systems, and trends," *Library Hi Tech*, vol. 38, no. 1, pp. 5–66, 2019.

[5] Y. Zhang, B. Li, B. Liu, J. Wu, Y. Wang, and X. Yang, "An attribute-based collaborative access control scheme using blockchain for IoT devices," *Electronics*, vol. 9, no. 2, p. 285, 2020.

[6] I. Riabi, Y. Dhif, H. K. B. Ayed, and K. Zaatouri, "A blockchain based access control for IoT," in *2019 15th International Wireless Communications Mobile Computing Conference (IWCMC)*, pp. 2086–2091, Tangier, Morocco, 2019.

[7] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, "A novel attribute-based access control scheme using blockchain for IoT," *IEEE Access*, vol. 7, pp. 38431–38441, 2019.

[8] 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.

[9] S. Algarni, F. Eassa, K. Almarhabi et al., "Blockchain-based secured access control in an IoT system," *Applied Sciences*, vol. 11, no. 4, p. 1772, 2021.

[10] A. Z. Ourad, B. Belgacem, and K. Salah, "Using blockchain for IOT access control and authentication management," in *International Conference on Internet of Things*, vol. 10972, pp. 150–164, Springer, Cham, 2018.

[11] E.-E. Chinyati, *Securing Internet of Things (IoT) Devices Using Hyperledger Fabric (Blockchain Technology), [M.S. thesis]*, Department of Computer Science, University of Westminster, London, UK, 2020.

[12] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "FairAccess: a new blockchain-based access control framework for the internet of things," *Security and communication networks*, vol. 9, no. 18, p. 5964, 2016.

[13] 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.

[14] G. Ayoade, V. Karande, L. Khan, and K. Hamlen, "Decentralized IoT data management using blockchain and trusted execution environment," in *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pp. 15–22, Salt Lake City, UT, USA, 2018.

[15] 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.

[16] B. Mbarek, M. Ge, and T. Pitner, "Blockchain-based access control for IoT in smart home systems," in *Database and Expert Systems Applications*, pp. 17–32, Springer International Publishing, Cham, 2020.

[17] M. A. Bouras, B. Xia, A. O. Abuassba, H. Ning, and Q. Lu, "IoT-CCAC: a blockchain-based consortium capability access control approach for IoT," *PeerJ Computer Science*, vol. 7, article e455, 2021.

[18] W. Ren, Y. Sun, H. Luo, and M. Guizani, "SILedger: a blockchain and ABE-based access control for applications in SDN-IoT networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4406–4419, 2021.

[19] A. Iftekhar, X. Cui, Q. Tao, and C. Zheng, "Hyperledger fabric access control system for Internet of Things layer in blockchain-based applications," *Entropy*, vol. 23, no. 8, p. 1054, 2021.

[20] X. Zhao, S. Wang, Y. Zhang, and Y. Wang, "Attribute-based access control scheme for data sharing on hyperledger fabric," *Journal of Information Security and Applications*, vol. 67, article 103182, 2022.

[21] L. Hang, B. Kim, K. Kim, and D. Kim, "A permissioned blockchain-based clinical trial service platform to improve trial data transparency," *BioMed Research International*, vol. 2021, Article ID 5554487, 22 pages, 2021.

[22] A. Iftekhar and X. Cui, "Anti-tamper protection for Internet of Things system using Hyperledger Fabric blockchain technology," 2021, June 2022 http://arxiv.org/abs/2109.07074.

[23] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "LSB: a lightweight scalable blockchain for IoT security and anonymity," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 180–197, 2019.

[24] L. Hang, B. Kim, and D. Kim, "A transaction traffic control approach based on fuzzy logic to improve Hyperledger Fabric performance," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 2032165, 19 pages, 2022.

[25] A. D. Dwivedi, R. Singh, S. Dhall, G. Srivastava, and S. K. Pal, "Tracing the source of fake news using a scalable blockchain distributed network," in *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 38–43, Delhi, India, 2020.

[26] S. Dhall, A. D. Dwivedi, S. K. Pal, and G. Srivastava, "Blockchain-based framework for reducing fake or vicious news spread on social media/messaging platforms," *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 21, no. 1, pp. 1–33, 2022.

[27] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of trust: a decentralized blockchain-based authentication system for IoT," *Computers & Security*, vol. 78, pp. 126–142, 2018.

[28] S. Asiri and A. Miri, "A sybil resistant IoT trust model using blockchains," in *2018 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. 1017–1026, Halifax, NS, Canada, 2018.

[29] B. W. Nyamtiga, J. C. S. Sicato, S. Rathore, Y. Sung, and J. H. Park, "Blockchain-based secure storage management with edge computing for IoT," *Electronics*, vol. 8, no. 8, p. 828, 2019.

[30] D. Puthal, S. P. Mohanty, V. P. Yanambaka, and E. Kougianos, "PoAh: a novel consensus algorithm for fast scalable private blockchain for large-scale IoT frameworks," 2020, arXiv preprint arXiv:2001.07297.

[31] D. Puthal, S. P. Mohanty, P. Nanda, E. Kougianos, and G. Das, "Proof-of-authentication for scalable blockchain in resource-constrained distributed systems," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–5, Las Vegas, NV, USA, 2019.

[32] S. Knezevic, *A blockchain approach for negotiating trust in IoT, [M.S. thesis]*, Engineering and Science, Florida Institute of Technology, Melbourne, Florida, 2020, June, 2020 https://repository.lib.fit.edu/handle/11141/3060.

[33] E. A. Shammar, A. T. Zahary, and A. A. Al-Shargabi, "A survey of IoT and Blockchain integration: security perspective," *IEEE Access*, vol. 9, pp. 156114–156150, 2021.

[34] "Hyperledger–open source blockchain technologies," June, 2020 https://www.hyperledger.org/.

[35] R. Boncea, I. Petre, and V. Vevera, "Building trust among things in omniscient internet using blockchain technology," *Romanian Cyber Security Journal*, vol. 1, no. 1, pp. 17–24, 2019.

[36] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: a systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, 2018.

[37] "Hyperledger Fabric–Hyperledger," June, 2020 https://www.hyperledger.org/use/fabric.

[38] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (PoET)," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, vol. 10616, pp. 282–297, Springer, Cham, 2017.

[39] C. Saraf and S. Sabadra, "Blockchain platforms: a compendium," in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pp. 1–6, Bangkok, Thailand, 2018.

[40] "Introduction|Hyperledger composer," June, 2021 https://hyperledger.github.io/composer/latest/introduction/introduction.html.

[41] Hyperledger Caliper, "Hyperledger caliper," September, 2021 https://hyperledger.github.io/caliper/.

[42] "Hyperledger explorer documentation—Hyperledger explorer documentation," September, 2021 https://blockchain-explorer.readthedocs.io/en/main/.

[43] J. Ali, T. Ali, S. Musa, and A. Zahrani, "Towards secure IoT communication with smart contracts in a blockchain infrastructure," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, 2018.

[44] "A blockchain platform for the enterprise — hyperledger-fabricdocs master documentation," September, 2021 https://hyperledger-fabric.readthedocs.io/en/release-2.2/.

[45] A. David, *Managing IOT Data on Hyperledger Blockchain, [M.S. thesis]*, Department of Computer Science, University of Nevada, Las Vegas, 2019.

[46] Y. Feng, W. Zhang, X. Luo, and B. Zhang, "A consortium blockchain-based access control framework with dynamic orderer node selection for 5G-enabled industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2840–2848, 2022.

[47] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, pp. 305–320, USA, 2014.

[48] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "OAuth-IoT: an access control framework for the Internet of Things based on open standards," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 676–681, Heraklion, Greece, 2017.

[49] P. Adhav, "Attribute-based access control (ABAC) in Hyperledger fabric," *Coinmonks*, vol. 25, 2020, June, 2022 https://medium.com/coinmonks/attribute-based-access-control-abac-in-hyperledger-fabric-1eb81330f67a.

[50] Hyperledger Performance and Scale Working Group, "Hyperledger blockchain performance metrics white paper, Hyperledger," October, 2021 https://www.hyperledger.org/learn/publications/blockchain-performance-metrics.

[51] L. Hang and D.-H. Kim, "Optimal blockchain network construction methodology based on analysis of configurable components for enhancing Hyperledger Fabric performance," *Blockchain: Research and Applications*, vol. 2, no. 1, article 100009, 2021.

[52] Newham, "Fabric-iot is a blockchain based decentralized access control system in IoT," 2021, November, 2021 https://github.com/newham/fabric-iot/blob/bc73b8eed37b5967072943752237b85a38c4617e/README.en.md.

[53] IBM, "Build and run a smart contract on a Hyperledger Fabric network with the Raft ordering service," International Business Machines, 2021, November, 2021 https://github.com/IBM/raft-fabric-sample.

[54] adhavpavan, "ContainerisingCaliperForFabricBenchmark," 2021, November, 2021 https://github.com/adhavpavan/ContainerisingCaliperForFabricBenchmark.