SC-CAAC: A Smart-Contract-Based Context-Aware Access Control Scheme for Blockchain-Enabled IoT Systems

Mpyana Mwamba Merlec, *Member, IEEE*, and Hoh Peter In, *Member, IEEE*

***Abstract*—Integrating blockchain technology with the Internet of Things (IoT) facilitates seamless interaction between IoT devices and systems to securely share, access, and exchange data. However, ensuring adequate access control within blockchain-enabled IoT (BIoT) systems remains a significant challenge. It is often difficult to adapt existing access control mechanisms to the dynamic and context-dependent nature of IoT environments, necessitating a robust context-aware approach to ensure adequate security and the privacy of resources within BIoT systems. In this article, we propose a novel smart contract-enabled context-aware access control (SC-CAAC) scheme for BIoT systems. It utilizes context-aware access control models that consider contextual information, including user profile, purpose, date, time, location, resource, and operating environment specifications, to make access control decisions. Smart contracts dynamically enforce access control policies and manage access permissions, ensuring that sensitive data and resources are accessible only to authorized users. The proposed scheme leverages the immutability, transparency, and decentralization of a blockchain that is shared by multiple participants in a consortium network, removing the need for a central authority to record and audit access control policies and decisions and promoting accountability and trust. The implementation and evaluation of our proposed scheme using the Hyperledger Besu blockchain demonstrates its effectiveness and scalability in real-world scenarios.**

***Index Terms*—Blockchain, blockchain-based Internet of Things (BIoT), context-aware access control (CAAC), Internet of Things (IoT), smart contracts.**

I. Introduction

THE Internet of Things (IoT) has reshaped how we interact with the world around us. IoT systems connect various devices together to exchange data, providing new opportunities for applications in industrial automation, smart factories, smart homes, and smart cities [1]. However, these systems also introduce new challenges, particularly in terms of privacy and security [2], [3], [4], [5], [6]. In IoT systems, many of the devices may have limited computing power, memory, and/or battery life, complicating the application of traditional security mechanisms [4], [5], [6], [7]. IoT systems can also contain sensitive data that require protection against unauthorized access or modification [6], [7], [8]. IoT security, privacy, and trust issues are discussed in [2], which emphasizes the need for robust security and privacy mechanisms to protect sensitive data and ensure user trust. In addition, [3] explored the privacy and security challenges facing IoT and highlighted the importance of encryption, authentication, and access control mechanisms to mitigate security risks.

Blockchain technology has the potential to enhance the security and privacy of IoT systems [9], [10], [11], [12], [13]. A blockchain is a distributed ledger that enables multiple parties to share and verify data without the need for intermediaries [9], [10]. Transactions on a blockchain are secured through cryptography and, once recorded, they cannot be voluntarily modified or deleted [11]. This makes blockchain technology an attractive option for securing IoT systems, particularly those that involve multiple parties with conflicting interests [12], [13].

Access control mechanisms are an essential component of secure IoT systems [5], [6], [7], [8]. Access control involves granting or denying access to assets based on the identity and permissions of the requesting users or services. Access control policies may include rules, such as who can access a resource, when they can access it, and what actions they can perform on it [7], [14]. The integration of blockchain technology and IoT systems has the potential to offer a range of features for secure and decentralized data management [9], [10], [11], [12]. However, it remains challenging to implement effective access control within Blockchain-enabled IoT (BIoT) systems [6], [11], [12], [13], [14], [15]. Existing access control mechanisms, such as discretionary access control (DAC), role-based access control (RBAC), attribute-based access control (ABAC), and policy-based access control (PBAC) [5], [6], [7], [8], [13], [14], [15], [16], are often not suited to the dynamic and context-dependent nature of IoT environments, thus a robust, context-aware strategy for the protection of security and the privacy of resources within BIoT systems is required.

The centralized architecture of existing access control systems also imposes certain limitations in terms of security,

Manuscript received 25 January 2024; accepted 16 February 2024. Date of publication 29 February 2024; date of current version 23 May 2024. This work was supported in part by the Korea University Funding; in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP, High Assurance of Smart Contract for Secure Software Development Life Cycle) funded by the Ministry of Science and ICT (MSIT) of the Korean Government under Grant 2021-0-00177; in part by the Technology Incubator Program for the Startup (TIPS) funded by the Ministry of Small and Medium Enterprises, and Startups (MSS) of the Korean Government under Grant S3306708. *(Corresponding author: Hoh Peter In.)*

Mpyana Mwamba Merlec is with the Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea (e-mail: mlecjm@korea.ac.kr).

Hoh Peter In is with the Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea, and also with DAO Solution Inc., Seoul 06247, South Korea (e-mail: hoh_in@korea.ac.kr).

Digital Object Identifier 10.1109/JIOT.2024.3371504

scalability, and trust [14], [15]. In a centralized system, a single authority or entity controls access control policies and decisions, thus it represents a single point of failure (SPF) targeted by malicious attacks. In addition, scaling these systems to accommodate a large number of users or devices can be challenging due to the concentration of control [13]. By employing blockchain technology and smart contracts, access control systems for BIoT can benefit from enhanced security, decentralization, and resilience [13], [14], [15], [16], [17], [18], [19], [20]. Smart contracts are self-executing agreements encoded on a blockchain that automatically execute predefined actions when specific conditions are met [10], [17]. Smart contracts thus can play a crucial role in IoT access control systems by providing a decentralized and automated mechanism to manage permissions and enforce policies [18], [19].

To address the challenges associated with BIoT systems, this article makes the following contributions.

1) We propose a novel smart-contract-based context-aware access control (SC-CAAC) scheme for BIoT systems. The proposed scheme utilizes context-aware access control (CAAC) models that consider contextual information, including the user profile, purpose, date, time, location, resource, and operating environment specifications, to make access control decisions. The goal of this scheme is to enhance security and privacy and improve the granularity of access control by considering the specific context in which access requests are made.
2) We design and build smart contracts that are integrated with IoT decentralized applications (Dapps) to dynamically enforce CAAC policies and manage access permissions, ensuring that only authorized users can access sensitive data and resources within a BIoT system. Cryptographic techniques ensure data confidentiality and integrity, protecting sensitive information from unauthorized access or modification.
3) We propose a practical and comprehensive integrated blockchain and IoT framework with a layered architecture. Blockchain is employed as a distributed and decentralized approach for access control in an IoT system. This approach uses a distributed ledger that is shared among multiple participants in a consortium network, removing the need for a central authority. The decentralization, transparency, and immutability of the blockchain ensure that access control policies and decisions are immutably recorded and auditable by all participants, enhancing trust and accountability.
4) Our proposed SC-CAAC scheme is implemented using the Hyperledger Besu blockchain to verify its effectiveness and scalability in real-world scenarios. Several use cases are presented to demonstrate the applicability and adaptability of the proposed scheme in various BIoT system settings.

The remainder of this article is organized as follows. Section II presents the research background and related work, while Section III describes the proposed SC-CAAC scheme. Section IV describes the implementation and evaluation process, with the use cases and applications of the proposed scheme outlined in Section V. Section VI discusses the limitations and remaining challenges for this technology. Section VII concludes and suggests future research.

II. Background and Related Work

This section describes the background of the research and reviews the literature on blockchain technology and smart SC-CAAC.

A. Internet of Things

IoT is an evolving paradigm that facilitates the interconnection of physical devices, including vehicles, buildings, and appliances. These items are equipped with sensors, software, and network connectivity, enabling them to collect and exchange data [1], [2], [3]. IoT is a rapidly growing field that has the potential to transform many aspects of modern life, including healthcare, transportation, manufacturing, and agriculture [1], [9], [10]. IoT systems are characterized by a number of heterogeneous devices and frequent data transmission, which requires efficient processing, storage, and communication strategies [1], [5]. The architectural framework of IoT systems, which includes connected devices, data transmission protocols, cloud or edge computing platforms, gateways, data aggregation analytics, and security mechanisms to protect data and ensure privacy, has been detailed in [9], [10], [11], [12], and [13]. The continuing evolution of IoT offers numerous opportunities and challenges, particularly in terms of security, privacy, and scalability, thus requiring collaborative efforts among researchers, practitioners, and policymakers [1], [2], [3], [4], [5].

B. Access Control Models for IoT

Access control in IoT is needed to ensure that only authorized entities have the right to access, modify, or operate IoT devices, data, and services [5], [6], [7], [8]. However, the high heterogeneity of devices and diverse range of applications and operational contexts associated with IoT systems complicate the implementation of access control [6], [13], [14], prompting the search for novel approaches and techniques. The research reported in [5], [6], [7], [8], [13], [14], and [15] has explored new technological advancements, emphasizing scalable and flexible access control systems that can adapt to dynamic IoT environments.

As summarized in Table I, a number of traditional access control models, such as DAC, RBAC, ABAC, and PBAC, have been adapted for IoT security [5], [6], [7], [8], [13], [14], [15], [16]. DAC governs access to resources based on the discretion of the resource owner [5], [6], [7]. It is simple and flexible but can lack fine-grained control and struggles in dynamic IoT environments [6]. RBAC, which is known for its high performance, scalability, and straightforward administration, cannot fully address the contextual nuance of IoT systems [13], [14], [15], [16]. ABAC considers various attributes of users, resources, and the environment to make access decisions [14], [16] and thus provides fine-grained control, adaptability, and scalability. Similarly, the PBAC model

TABLE I
COMPARISON OF EXISTING ACCESS CONTROL MODELS

Property	DAC	RBAC	ABAC	PBAC	CAAC
Complexity	Low	Low	High	Moderate	High
Granularity	Low	Low	High	Moderate	High
Flexibility	High	Moderate	High	High	High
Administration	Simple	Simple	Complex	Complex	Complex
Performance	Low	High	High	High	High
Scalability	High	Low	High	High	High
Security Level	Low	High	High	High	High

uses high-level policies to make access control decisions, incorporating real-time contextual information [6], [14] to provide flexibility, adaptability, and fine-grained authorization. However, with the exception of DAC and RBAC, these options are characterized by complex policy management [7], [14], [15], [16], while defining and managing policies can be challenging, leading to potential policy conflicts.

An access control architecture for a secure IoT platform is proposed in [8], focusing on robust mechanisms to facilitate secure and authorized interactions. He et al. [15] reconsidered access control and authentication mechanisms, addressing challenges posed by smart home-based IoT devices and proposing new approaches to enhance security. These advancements highlight the importance of integrating contextual factors, such as the location, time, device type, and environmental conditions, into access control decisions to enable more dynamic and effective policies, thus enhancing the security and operational efficiency of IoT systems.

CAAC models [20], [21], [22], [23], [24], [25], which incorporate contextual information, such as the user identity, location, time, IP address, device security status, and resource sensitivity into access decisions, are increasingly useful in IoT settings. A comprehensive review of CAAC systems, particularly in cloud and fog networks, is provided in [20], highlighting the evolution of and challenges in this area. Context-based access control models that consider the environmental context of access requests have also been developed [21], emphasizing the importance of environmental factors in access decisions. In addition, integration of contextual information in cloud computing environments for adaptive policy enforcement is explored in [22].

Other research has focused on user context and capability-attribute-based approaches for IoT access control [23], [24], [25], assigning specific capabilities to entities to facilitate granular access control. Context-aware attribute-based models have been proposed in [23] and [24] to ensure the security of sensitive data such as electronic health records (EHRs) by taking into account critical incidents and context-specific factors. However, because these models have been proposed for centralized IoT systems, they are inherently susceptible to single points of failure and exhibit limitations in terms of security, privacy, and scalability.

C. Blockchain IoT and Smart Contracts

BIoT involves the integration of blockchain technology with IoT systems [9], [10], [11], [12], [13], [25]. BIoT has several use cases across various industries, including smart cities, smart farms and factories, logistics and supply chains, smart homes, smart grids, and appliances [10], [11], [12], [13], [14]. The use of blockchains can enable the secure and decentralized exchange of data between IoT devices to increase the security and privacy of IoT networks [12], [13], [14], [15]. In BIoT systems, IoT devices can transmit and store encrypted data on the blockchain, creating tamper-resistant records of shared transactions. Data confidentiality, integrity, and availability can be ensured by using decentralized persistent storage systems, such as the interplanetary file system (IPFS). Smart contracts, which practically originated with the Ethereum blockchain [26], are self-executing contracts in which the terms of agreements are directly built into the code. They can be used to automate on-chain access control decisions and provide tamper-proof audit trails [10], [17]. Smart contracts play a crucial role in BIoT systems.

Blockchains are mainly categorized based on the network type and access permissions [26], [27], [28]. *Public blockchain* networks, such as Bitcoin and Ethereum, are open to everyone without restrictions [26]. In contrast, *Private blockchain* networks, such as Hyperledger Fabric, restrict access to specific participants of an organization [27]. *Consortium blockchain* networks are governed by a group of organizations and allow a controlled set of participants [28], while *hybrid blockchain* networks combine both private and public features [28]. In addition, it permissionless blockchains allow anyone to participate, while *permissioned blockchains* require explicit permission. The choice of blockchain type depends on numerous factors, including access control needs, governance, performance, and scalability [9], [10].

D. Blockchain and Smart-Contract-Based Access Controls for IoT

Blockchain systems offer a decentralized and secure framework for access control in IoT [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42]. The inherent properties of these systems, including their immutability, transparency, and distributed consensus, can be used to enhance the security and efficiency of access control systems in IoT environments. Previous studies [29], [30], [31] have explored various blockchain-based solutions for IoT access control. Study of integrating a blockchain system with IoT is proposed in [32] for secure healthcare digital systems, using the transparency and immutability of blockchains to ensure data integrity and privacy in healthcare applications. A mutual authentication system using blockchain is presented in [34], with fine-grained access control features introduced to achieve secure Industry 4.0. In this framework, the blockchain is employed to improve the security and efficiency of access control in industrial environments. Smart contracts are used in [35] as a capability-based access control mechanism in decentralized IoT systems. In [36], the focus is on user capability-based access control strategies for situational awareness. A consortium capability access control approach using a blockchain system is proposed in [37] to facilitate secure and efficient resource sharing among IoT devices.

TABLE II
COMPARISON OF THE PROPOSED SCHEME WITH RELATED WORK

Reference	Domain	CAAC[1]	SC[2]	XACML	APMS[3]	APEA[4]	S & P[5]	Architecture	Blockchain	Network	Built	PE[7]
[5]	IoT	✗	✗	✓	✗	✗	✓/ ✓	Centralized	✗	Private	✓	✗
[13]	BIoT	✗	✓	✗	✗	✗	✓/ ✗	Decentralized	✓	Public	✓	✓
[15]	HIoT	✗	✗	✗	✗	✗	✓/ ✗	Centralized	✗	Private	✓	✓
[19]	EHR	✗	✓	✓	✗	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
[21]	Mobile	✓	✗	✗	✗	✗	✓/ ✓	Centralized	✗	Private	✗	✗
[22]	PaaS	✓	✗	✓	✓	✓	✓/ ✓	Centralized	✗	Private	✓	✓
[23]	EHR	✓	✗	✗	✗	✗	✓/ ✗	Centralized	✗	Private	✓	✓
[24]	EHR	✓	✗	✓	✗	✗	✓/ ✗	Centralized	✗	Private	✓	✓
[25]	BIoT	✗	✓	✗	✗	✗	✓/ ✗	Decentralized	✓	Public	✓	✓
[29]	BIoT	✗	✓	✗	✗	✗	✓/ ✓	Decentralized	✓	Private	✗	✗
[31]	Edge	✗	✓	✗	✗	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
[33]	IoMT	✗	✓	✗	✗	✓	✓/ ✗	Decentralized	✓	Public	✗	✗
[34]	Industry 4.0	✗	✓	✗	✗	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
[35]	BIoT	✗	✓	✗	✓	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
[36]	Space	✓	✓	✗	✓	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
[37]	BIoT	✗	✓	✗	✓	✓	✓/ ✓	Decentralized	✓	Consortium	✓	✓
[38]	General	✗	✓	✓	✓	✗	✓/ ✓	Decentralized	✓	Consortium	✓	✓
[39]	–	✓	✓	✗	✗	✓	✓/ ✗	Decentralized	✓	Public	✓	✓
[40]	BIoT	✓	✓	✗	✗	✗	✓/ ✗	Decentralized	✓	Hybrid	✓	✓
[41]	IoMT	✓	✓	✗	✗	✓	✓/ ✗	Decentralized	✓	Private	✓	✓
[42]	Supply chain	✗	✓	✗	✓	✗	✓/ ✓	Decentralized	✓	Private	✓	✓
[43]	SDN/NFV-6G	✓	✓	✗	✗	✓	✓/ ✓	Decentralized	✓	Private	✓	✓
This work	BIoT	✓	✓	✓	✓	✓	✓/ ✓	Decentralized	✓	Consortium	✓	✓

[1]Context-aware access control [2]Smart contract [3]Access Policy Modeling Schema [4]Access Policy Enforcement Algorithm [5]Security and Privacy [7]Performance evaluation

A smart-contract-enabled dynamic consent and fine-grain access control management approach using a blockchain system is proposed in [38] for personal data utilization that complies with the General Data Protection Regulation (GDPR). It allows individuals to maintain granular control over their data and make informed decisions regarding its usage. In [39], a dynamic context-aware RBAC mechanism using a blockchain system is proposed for decentralized IoT environments. Research in [40] introduces blockchain-based context-aware authorization management as a service for secure resource sharing in IoT. A blockchain-based context-aware CP-ABE encryption schema for the Internet of Medical Things (IoMT) is discussed in [41], combining a blockchain with context-aware attribute-based encryption for enhanced security. An access control framework for blockchain-based supply chain systems is presented in [42]. Finally, [43] proposes a context-aware authentication handover and secure network slicing approach using a DAG blockchain system for edge-assisted SDN/NFV-6G environments, ensuring secure handovers based on contextual information.

Table II compares our proposed SC-CAAC scheme for BIoT systems with related strategies. Our approach aims to address the limitations of previous methods and enhance the security and privacy of data in BIoT systems. CAAC policies and permissions are dynamically enforced using smart contracts, ensuring that only authorized users have access to assets within the BIoT environment. The decentralized nature of blockchain mitigates single points of failure, making it difficult for attackers to compromise the system. Furthermore, the use of consensus mechanisms in the blockchain system ensures tamper-proof policy enforcement, reducing the risk of unauthorized access.

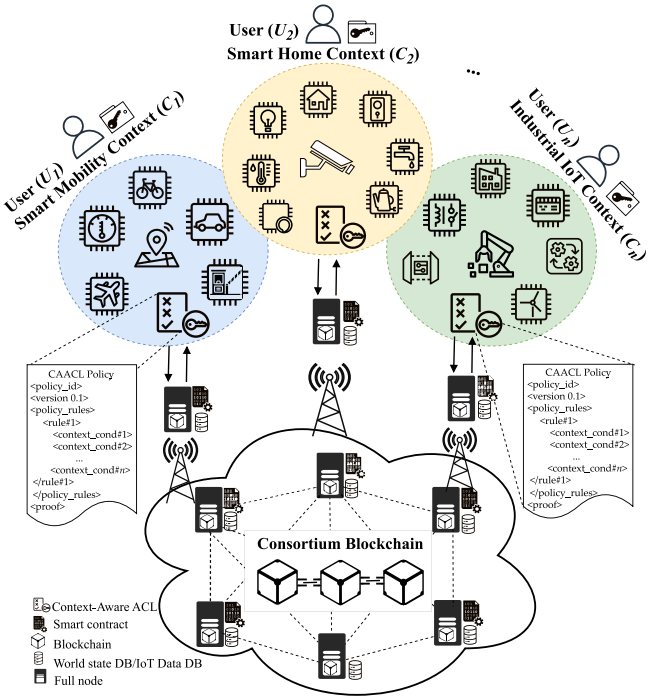

Fig. 1. Use cases of the proposed SC-CAAC scheme for BIoT systems. CAACL: Context-aware access control list.

III. SYSTEM DESIGN

Fig. 1 illustrates the proposed SC-CAAC scheme for BIoT systems. This scheme can be adapted to various use cases, including smart cities, smart mobility, smart homes, and Industrial IoT (IIoT). The use of smart contracts operating on IoT gateways or the nodes of the consortium blockchain network allows autonomous and reliable transactions without needing a central trusted authority. This approach uses a

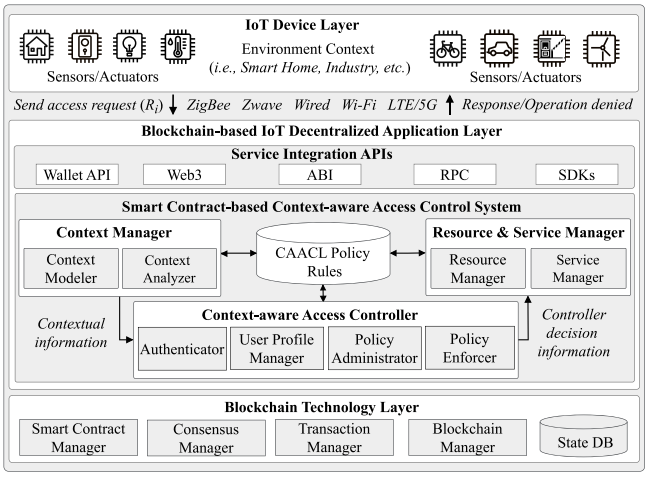

Fig. 2. Layered architecture framework of the proposed SC-CAAC scheme for BIoT systems. *ABI*: application binary interface; *API*: application interface, *DB*: database; *RPC*: remote procedure call; and *SDK*: software development kit.

CAAC model to make informed access control decisions. The model considers contextual information, such as the user profile, purpose, date, time, location, resource, and operating environment specifications to determine whether to grant or deny access to a resource. For example, in a smart home scenario, a smart lock can use context-aware smart contracts to assess access rights, allowing or denying entry based on predefined conditions. Similarly, a smart thermostat can be regulated using smart contracts to access online energy pricing data and adjust temperature settings during peak pricing periods to optimize energy usage. SC-CAAC can thus automate and decentralize access control for IoT devices in a BIoT system, enhancing its security, transparency, and ability to audit interactions and transactions.

A. System Architecture

Fig. 2 presents the layered framework architecture of our SC-CAAC scheme for BIoT systems. It consists of the following three layers.

1) *IoT Device Layer:* This layer comprises heterogeneous IoT devices, which encompass the physical devices and sensors that collect data from the physical environment. These devices can include various types of sensors, actuators, and embedded systems. The BIoT network ensures seamless connectivity and data exchange among IoT devices using various protocols, standards, and technologies, which enable reliable and efficient communication, such as Bluetooth, Wi-Fi, ZigBee, Zwave, and cellular networks (i.e., LTE or 5G). The identity management and certificate issuance for IoT devices can be managed by trusted entities within the BIoT consortium network. These entities, possibly IoT device manufacturers or network administrators, are responsible for registering devices and issuing certificates, ensuring a secure identity verification process.
2) *BIoT Dapp Layer:* This is a middleware layer that provides essential services and functionalities enabling interoperability, data processing, and integration with BIoT Dapps. It includes two main modules.
   a) *Service Integration Application Interfaces (APIs):* Services integration APIs play an essential role in facilitating interactions between IoT devices and the blockchain network. The integration APIs and protocols include Web3 and wallet APIs, application binary interfaces (ABIs), remote procedure calls (RPCs), and software development kits (SDKs). These enable secure authentication, data exchange, and execution of smart contract functions in BIoT systems. They allow IoT devices to operate in Dapps and benefit from the transparency, immutability, and security provided by the blockchain. The wallet APIs are used for managing cryptographic keys and digital wallets associated with IoT devices. It provides secure storage and generates signatures for transactions using private keys, allowing IoT devices to authenticate and interact securely with the blockchain network.
   b) *SC-CAAC System:* This consists of a number of core components. a) The *context manager* contains two subcomponents: i) the *context modeler*, which is used to define and represent the contextual information relevant to access control decisions by capturing data, such as the user profile, purpose, date, time, location, operating environment, and resource specifications and ii) the *context analyzer*, which processes the contextual data, applies analytics, and generates meaningful insights that are used by the context-aware access controller. b) The *CAACL policy rules database (DB)* serves as a repository for predefined CAACL policy rules, specifically designed to accommodate the contextual information captured. c) The *context-aware access controller* consists of several subcomponents that are responsible for checking and enforcing contextual constraints of access control policy rules. These subcomponents are listed as follows.
      i) The *authenticator* verifies the authenticity and identity of users or devices seeking access to the resources of the BIoT system.
      ii) The *user profile manager* manages user profiles and maintains relevant information, such as user roles, permissions, and associated contextual attributes.
      iii) The *policy administrator* handles the administration of the access control policies. It is used to define, update, and remove policies as required.
      iv) The *policy enforcer* evaluates the CAAC rules stored in the CAACL policy rules DB against the contextual constraints provided by the context manager and makes access control decisions, i.e., granting or denying access to resources and services within the system.

d) The *resource and service manager* oversees the management and provisioning of resources and services within the system. It consists of two subcomponents: i) the *resource manager* handles the allocation and management of resources (e.g., data, devices, and services) and it ensures that access to resources is regulated according to CAAC policies and ii) the *service manager*, which manages the availability and accessibility of services provided by the system and it coordinates the provisioning of services considering access control decisions made by the context-aware access controller.

3) *Blockchain Technology Layer:* This consists of several components working together to operate a blockchain network. These components include: a) the *smart contract manager*, which is responsible for managing and executing smart contracts within the blockchain network, facilitates the deployment, storage, and retrieval of smart contracts; b) the *consensus manager* is essential for maintaining the integrity and consensus of the state of the blockchain network; c) the *transaction manager* handles the processing and validation of transactions within the blockchain network and ensures that transactions are properly formatted, have valid signatures, and meet all of the criteria specified by the blockchain network; d) the *blockchain manager* is in charge of managing the blockchain's shared ledger; and e) the *state DB* stores and maintains the state of the blockchain network. It is updated and synchronized as new transactions are processed and blocks are added to the blockchain.

This architecture provides a structured approach for the design and implementation of efficient and robust CAAC schemes for BIoT systems. Each layer focuses on specific tasks and responsibilities, contributing to the overall functionality and efficiency of the IoT ecosystem.

B. CAAC Policy Management

CAAC policy management involves the defining, deploying, updating, and revoking/removing of policies. It also enforces access control policies by considering the contextual information of the system and users. CAAC policies are defined and managed by the BIoT system administrators (designated based on their role and authority within the consortium network) within the SC-CAAC.

1) CAACL Policy Definition: CAAC policy $\mathcal{P}_i$ can be defined as in

$$\mathcal{P}_i = \left\{d, \nu, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}\right\} \tag{1}$$

where d refers to the policy description, ν is the policy version, and $\mathcal{P}(\mathcal{R}_n)$ is the aggregation of defined policy context-aware rules, which is expressed in

$$\mathcal{P}(\mathcal{R}_n) = f_n\left(\mathcal{R}_j\right),\ j \in [1, n] \tag{2}$$

where $\mathcal{R}_j$ is a policy rule from predefined context-aware policy rule set $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots, \mathcal{R}_n\}$. Equation (3) is used to formally define $\mathcal{R}_j$

$$\mathcal{R}_j = \left\{\xi, \mathcal{U}, \bar{\mathcal{R}}, \mathcal{C}(\mathcal{R}_m), \mathcal{A}, p\right\} \tag{3}$$

$$\xi = \begin{cases} 1, & \text{enable} \\ 0, & \text{disable} \end{cases} \tag{4}$$

where ξ denotes the policy rule effect, which is a binary variable as defined in (4). $\mathcal{U}$ represents a set of authorized users defined as $\mathcal{U} = \{u_1, u_2, u_3, \ldots, u_x\}$. $\bar{\mathcal{R}}$ refers to a set of resources to which rule is applied, defined as $\bar{\mathcal{R}} = \{\bar{r}_1, \bar{r}_2, \bar{r}_3, \ldots, \bar{r}_y\}$. $\mathcal{A}$ is a set of authorized actions defined as $\mathcal{A} = \{a_1, a_2, a_3, \ldots, a_u\}$, and p represents the permission state defined in (5). Permission state p can be either it is 1 (*allow*) or 0 (*deny*).

$$p = \begin{cases} 1, & \text{allow} \\ 0, & \text{deny.} \end{cases} \tag{5}$$

Context-aware constraint rule $\mathcal{C}(\mathcal{R}_m)$ can be formally defined using

$$\mathcal{C}(\mathcal{R}_m) = f_m\left(\mathcal{C}_{c_k}\right),\ k \in [1, m] \tag{6}$$

where C_{c_k} is the kth of m defined context constraints of CAAC policy rule R_m, $f_m(\mathcal{C}_{c_k})$ is a function that aggregates context constraint-factor parameters and maps them onto the corresponding R_m, and $\bar{\mathcal{P}}$ denotes the policy creation proof document. This document includes type Π, created timestamp t, creator identifier U_i, verification method V, policy rule hash h, signature S, and current status s of the proof, as defined in

$$\bar{\mathcal{P}} = \{\Pi, t, U_i, V, h, S, s\}. \tag{7}$$

The extensible access control markup language (XACML) [44] is used to define models for the SC-CAAL policy rules for simplicity and standardization purposes. Code Listing 1 presents an SC-CAACL policy definition in JSON profile for XACML [38], [45]. It consists of three main sections.

1) The *policy header section* provides basic policy information, including the name, description, and version.
2) The *policy rules section* contains CAAC rules defined within the policy. This code snippet contains one rule, but several rules can be defined with an enabling effect. The policies rules section specifies the authorized users, resources, and various contextual constraints, such as user roles, dates and periods, weekdays, location ranges, places, devices, and authorized IP addresses. The allowed actions for this rule are also defined.
3) The *proof section* provides cryptographic proof of the policy's authenticity and integrity. It includes the signature, creation timestamp, creator identifier, verification method, policy rule hash, signature value, and status. This SC-CAACL policy schema allows granular access control policies to be enforced based on contextual factors.

Algorithm 1 describes the proposed SC-CAAC policy setup and registration process. The parameters received by the algorithm include the user account address U_a, session authentication status $\mathcal{A}_s$, and CAACL policy setup $\mathcal{P}_i = \{d, \nu, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}\}$. It begins by verifying whether the requesting user account is authorized and whether the session status is authenticated. It then parses and extracts all CAACL policy

```
{"policy_id":"ContextAwareAccessPolicy#01",
"policy_desc": "CAACL policy for BIoT",
"policy_version": "1.0",
"policy_rules": [
 {"rule_id":RL#001,
  "effect": "enable",
  "authorized_users":["U001","U002"],
  "resource":["R001","R002"],
  "context_constraints": {
    "user_role":["admin"],
    "date_period":{
    "start_date":"2024-06-01 T15:10:20Z",
    "end_date":"2025-05-31 T15:10:19Z"},
    "time_period":{"start_time":"01:00",
        "end_time":"23:59"},
   "weekdays":["Mon","Tue","Wed","Thu","Fri"],
   "location_range":{"latitude":40.7128,
       "longitude":-74.0060,"radius":50000},
   "place":["Office","Home","School"],
   "device":[{"id":"M24","type":"Mobile"}],
   "authorized_ip":["127.0.0.*"] },
  "action":["setDevice()","getIoTData()"],
  "permissions": "allow" }],
"proof":[
 {"type": "Ed25519Signature2022",
  "created": "2023-06-01 T15:05:20Z",
  "creator": "143Cda6...54f2bc2",
  "verificationMethod": "v2e...q6x/v#01",
  "policyRuleHash": "d4FxhTu...GMq6ZcB",
  "signatureValue": "b2GeZhk...aYq6ZbA",
  "status": "activated" }]
}
```

Code Listing 1. SC-CAAC policy JSON schema for XACML

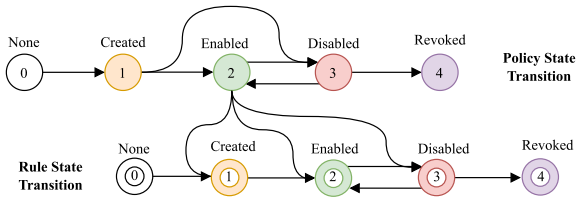

Fig. 3. SC-CAAC policy and policy rule state transition diagram: None (0), Created (1), Enabled (2), Disabled (3), and Revoked (4).

attributes and checks whether $\mathcal{P}_i$ is registered already to avoid duplication and whether the given proof is valid. It then maps all policy context constraint rules $\mathcal{R}_j$ of $\mathcal{P}_i$, collects timestamps t, and then pushes the policy set's instance to the blockchain. Transaction hash T_h and block number B_{no} are returned upon successful execution of the transaction. In the case of failure, all the policy states are reverted to their initial states.

2) SC-CAAC Policy and Rule State Transitions: Fig. 3 illustrates the state transitions for both SC-CAAC policy and the policy rule lifecycle. For the *Policy State Transition*, the process begins with the *Created* state (1), suggesting the initiation of a policy from *None* (0), indicating the absence or nonexistence of a policy. Following its creation, the policy can be *Enabled* (2), making it active and enforceable. However, there might be scenarios where it is necessary to temporarily deactivate the policy, leading the *Disabled* state (3). Finally, if the policy is no longer relevant or required, it transitions to the *Revoked* state (4), making it invalid. In parallel, the *Rule State Transition* presents a more nuanced picture. The *None* state (0) signifies the absence or noncreation of a rule. After it is formulated, the rule enters the *Created* state (1). Much like its policy counterpart, the rule can then be *Enabled* (2) to be actively applied or *Disabled* (3) for temporary deactivation. The final *Revoked* state (4) marks the rule's termination, and it is no longer valid. Notably, the diagram in Fig. 3 shows potential reversals between states, highlighting the dynamic nature of policy and rule management.

Algorithm 1 CAAC Policy Setup and Registration

Setting parameters: Contract address SC_a, admin address A_a, ACM
Input: Account address U_a, AuthStatus $\mathcal{A}_s$, Policy $\mathcal{P}_i = \{d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}\}$
Output: T_h, B_{no}

while $(U_a, \mathcal{A}_s, \mathcal{P}_i[d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}])$ **do**
  if $(U_a \neq NULL)$ **and** $(U_a \in U)$ **then**
    if $\mathcal{A}_s =$ **true then**
      Parse and extract all attributes of CAACL policy P_i
      if $(\mathcal{P}_i \notin \mathcal{P})$ **and** $(isValid(\mathcal{P}_i[\bar{\mathcal{P}}]) =$ **true**$)$ **then**
        for all $\mathcal{R}_j$ **in** $\mathcal{P}(\mathcal{R}_n)$ **do**
          Map all context constraints:
          $\mathcal{R}_j[\ ] \leftarrow \{\xi, \mathcal{U}, \bar{\mathcal{R}}, \mathcal{C}(\mathcal{R}_m), \mathcal{A}, p\}$
          return mapped CAACL Policy rules $\mathcal{P}(\mathcal{R}_n)$
        end for
        Map all $\mathcal{P}_i$ attributes $\mathcal{P}_i[d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}]$
        Collect a timestamp $t \leftarrow Datetime.now()$
        $sc.newPolicy.push(\mathcal{P}_i[d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}])$
        Compute the transaction: $T = [i, hash(\mathcal{P}_i), t)\ ]$
        Emit $sc.newPolicyAdded(\mathcal{P}_i, U_a, t)$
        return the transaction execution state (T_h, B_{no})
      else
        return "Policy proof $\mathcal{P}_i[\bar{\mathcal{P}}]$ is not valid."
      end if
    else
      return "User session is not authenticated: $\mathcal{A}_s =$ **false**"
    end if
  else
    return "User account address is not authorized: $U_a \notin U$"
  end if
end while

3) CAACL Policy Rule Checking and Enforcement: CAACL policy rule evaluation is a process in which it is determined whether requested access should be granted or denied based on the defined policies and the current context. Policy enforcement involves mechanisms to enforce access control policies and restrict unauthorized access attempts. These mechanisms may include authentication, authorization checks, encryption, and other security measures. Policy enforcement ensures that access control decisions are made correctly and consistently by computing all relevant CAACL policy rules from the access control matrix (ACM).

The ACM, which defines access permissions between specific subjects and objects, is defined in (8), where $\mathcal{Q}_i$ is the access request and $\mathcal{C}_{c_k(\Delta)}$ is the state of the context constraint condition. For example, let $\mathcal{C}_{c_k(\Delta)}$ be a context constraint condition for which the request location is within the authorized range, defined as a latitude (ℓ) of 40.7128, longitude (L) of -74.006, and a radius (r) of 500 000 m. Equations (9)–(11) [46] are used to determine if the request

location is within the authorized range

$$ACM = \begin{Bmatrix} \mathcal{Q}_i & \mathcal{C}_{c_1(\Delta)} & \mathcal{C}_{c_2(\Delta)} & \mathcal{C}_{c_3(\Delta)} & \cdots & \mathcal{C}_{c_m(\Delta)} \\ \mathcal{Q}_1 & 1 & 0 & 0 & \cdots & 0 \\ \mathcal{Q}_2 & 1 & 1 & 0 & \cdots & 1 \\ \mathcal{Q}_3 & 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{Q}_n & 1 & 0 & 1 & \cdots & 0 \end{Bmatrix} \quad (8)$$

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi 1.\cos\phi 2.\sin^2\left(\frac{\Delta\lambda}{2}\right). \quad (9)$$

Equation (9) calculates the value of a, which is an intermediate value used in the Haversine formula [46]. $\Delta\phi$ is the difference in latitude between two points, while $\phi 1$ and $\phi 2$ are the latitudes of these two points. $\Delta\lambda$ denotes the difference in longitude between the two points. Equation (10) calculates the value of c, which is the angular distance between the points on a sphere. *atan*2 is a two-argument arctangent function that is used to compute the inverse tangent of the quotient of its arguments. Finally, in (11), the distance is calculated by multiplying the angular distance c by the radius of the Earth R. Thus, the $\mathcal{C}_{c_k(\Delta)}$ constraint is satisfied if distance $\leq r$ for the request location is within the location range radius

$$c = 2.\text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \quad (10)$$

$$\text{distance} = R.c. \quad (11)$$

Predefined policy rules are enforced by computing $\mathcal{Q}_i(\Delta)$, which is the control decision for request access $\mathcal{Q}_i$. It is 1 (granted) if all of the condition state values $\mathcal{C}_{c_k(\Delta)}$ for context constraints in a row of the ACM are 1 (*True*), as expressed in

$$\mathcal{Q}_i(\Delta) = \begin{cases} 1, & \text{granted } \left(\forall\ \mathcal{C}_{c_k} = 1,\ k \in [1, m]\right) \\ 0, & \text{denied (otherwise).} \end{cases} \quad (12)$$

Algorithm 2 depicts the verification process for the CAACL policy rules. The received input parameters include user account address U_a, session authentication status $\mathcal{A}_s$, access request $\mathcal{Q}_i$, and relevant CAACL policy $\mathcal{P}_i$. The algorithm begins by verifying whether U_a is authorized and $\mathcal{A}_s$ is authenticated. It then checks whether $\mathcal{P}_i$ exists within the predefined CAACL policy set $\mathcal{P}$ and whether $\mathcal{P}_i[\bar{\mathcal{P}}]$ is valid. If all policy context constraint rules are verified, their verification states are updated in the *ACM* to 1 (*true*) or 0 (*false*). Finally, the updated *ACM* is returned as output, which is used by the policy enforcer for access control decision enforcement, as described in Algorithm 3.

Algorithm 3 receives input parameters that include user account address U_a, session authentication status $\mathcal{A}_s$, access request $\mathcal{Q}_i$, CAACL policy $\mathcal{P}_i$, and the relevant states in the *ACM*. It begins by checking whether U_a is authorized and $\mathcal{A}_s$ is authenticated. Next, it determines whether access request $\mathcal{Q}_i$ and the states of the relevant policy context constraint rules $\mathcal{Q}_i[\mathcal{C}_{c_k(\Delta)}]$ and $\mathcal{P}_i$ exist in the *ACM* and $\mathcal{P}$, respectively. The validity proof $\mathcal{P}_i[\bar{\mathcal{P}}]$ and the state values of all policy contextual constraint rules $\mathcal{Q}_i[\mathcal{C}_{c_k(\Delta)}]$ are then verified. If the proof is valid and all $\mathcal{Q}_i[\mathcal{C}_{c_k(\Delta)}]$ in the *ACM* are 1 (*true*), then the decision state $\mathcal{Q}_i(\Delta)$ of the access request is updated to 1 (*true*) and access is granted. Otherwise, it is updated to 0 (*false*) and access is denied.

Algorithm 2 CAAC List Policy Rules Verification

Setting parameters: Contract address SC_a, admin address A_a, CAACL Policy $\mathcal{P} = \{d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}\}$, *ACM*
Input: Account address U_a, AuthStatus $\mathcal{A}_s$, Access request $\mathcal{Q}_i$, Policy $\mathcal{P}_i$
Output: *ACM* updated

while $(U_a, \mathcal{Q}_i, \mathcal{P}_i, \mathcal{A}_s)$ **do**
  if $(U_a \in U)$ **and** $(\mathcal{A}_s =$ **true**$)$ **then**
    if $(\mathcal{P}_i \in \mathcal{P})$ **then**
      if $sc.isValid(\mathcal{P}_i[\bar{\mathcal{P}}] =$ **true**$)$ **then**
        for all $\mathcal{R}_j$ **in** $\mathcal{P}_i$ **do**
          for all $\mathcal{C}_{c_k}$ **in** $\mathcal{R}_j$ **do**
            if $(\mathcal{C}_{c_k} \neq NULL) \wedge (\mathcal{C}_{c_k} \stackrel{?}{=}$ **true**$)$ **then**
              update rule context constraint state: $\mathcal{C}_{c_k(\Delta)} \leftarrow 1$
            else
              update rule context constraint state: $\mathcal{C}_{c_k(\Delta)} \leftarrow 0$
            end if
            Save $\mathcal{C}_{c_k(\Delta)}$ in *ACM*: $ACM \leftarrow \mathcal{C}_{c_k(\Delta)}$
          end for
        end for
        return *ACM*
      else
        return "$\mathcal{P}_i[\bar{\mathcal{P}}]$ is not valid."
      end if
    else
      return "$\mathcal{P}_i$ does not exit."
    end if
  else
    return "U_a not authorized or session $\mathcal{A}_s$ not authenticated."
  end if
end while

IV. Implementation and Evaluation

This section discusses in detail how the proposed SC-CAAC scheme is implemented and evaluated.

A. Implementation Details

Table III summarizes the implementation and experimental setup. Hyperleder Besu [47] is adopted to build the prototype for our proposed scheme. Hyperledger Besu [47] is an open-source blockchain platform compatible with Ethereum. It supports several consensus protocols and provides a smart contract execution environment, monitoring and management tools. It also offers fine-grained access controls with data privacy options for permissioned blockchain networks. Using Hyperledger Besu [47], we establish a consortium blockchain network that enables multiple organizations to collaborate and participate in a permissioned BIoT network with enhanced privacy and security features. The Solidity [48] programming language is used to develop our smart contracts, which run on Ethereum Virtual Machines (EVMs) that are compatible with Hyperleder Besu. The architectural model of the implemented smart contracts is described in the Appendix.

The programming languages, libraries, APIs, and frameworks used to build our Dapps include Python, Flask,[1]

[1] https://github.com/pallets/flask/

Algorithm 3 CAAC Policy Rules Enforcement

Setting parameters: Contract address SC_a, admin address A_a, CAACL Policy $\mathcal{P} = \{d, v, \mathcal{P}(\mathcal{R}_n), \bar{\mathcal{P}}\}$, *ACM*
Input: Account address U_a, AuthStatus $\mathcal{A}_s$, Access request $\mathcal{Q}_i$, Policy $\mathcal{P}_i$ **Output:** $\mathcal{Q}_i(\Delta)$ Access Granted/Denied

while (U_a, $\mathcal{Q}_i$, $\mathcal{P}_i$, $\mathcal{A}_s$) **do**
  if ($U_a \in U$) **and** ($\mathcal{A}_s =$ **true**) **then**
    if ($\mathcal{Q}_i \in ACM$) **then**
      if ($\mathcal{P}_i \in \mathcal{P}$) **then**
        if $sc.isValid(\mathcal{P}_i[\bar{\mathcal{P}}] =$ **true**) **then**
          if $\mathcal{Q}_i[\mathcal{C}_{c_k(\Delta)}] = 1$ **for all** ($\mathcal{Q}_i[\mathcal{C}_{c_k(\Delta)}]$ **in** *ACM*) **then**
            update access control decision state: $\mathcal{Q}_i(\Delta) \leftarrow 1$
            save $\mathcal{Q}_i(\Delta)$ in *ACM*: $ACM \leftarrow \mathcal{Q}_i(\Delta)$
            return "Access granted: $\mathcal{Q}_i(\Delta)$"
          else
            update access control decision state: $\mathcal{Q}_i(\Delta) \leftarrow 0$
            save $\mathcal{Q}_i(\Delta)$ in *ACM*: $ACM \leftarrow \mathcal{Q}_i(\Delta)$
            return "Access denied: $\mathcal{Q}_i(\Delta)$"
          end if
        else
          return "$\mathcal{P}_i[\bar{\mathcal{P}}]$ is not valid."
        end if
      else
        return "$\mathcal{P}_i$ does not exit."
      end if
    else
      return "$\mathcal{Q}_i$ does not exist in *ACM*."
    end if
  else
    return "U_a not authorized or session $\mathcal{A}_s$ not authenticated."
  end if
end while

TABLE III
DETAILS OF THE IMPLEMENTATION AND EXPERIMENTAL SETUP

Consortium Blockchain Network	
Parameters	Values
Network	BIoTNet (Hyperledger Besu Consortium)
Chain ID	1337
Number of nodes	3 members, 4 validators, 1 rpcnode
Consensus protocol	IBFT 2.0
Epoch interval	30,000
Step period/Average block time (s)	5
Request timeout (s)	10
Total difficulty	0x1
Gas limit	16,234,336
Average gas used	1,181.19
Average block size (bytes)	845.2
Number of Tx per block	1
Smart contract language	Solidity
Solidity compiler version	v0.8.20
EVM Version	London
Private transaction manager	Tessera
Hardware, Dapp Development, Deployment, and Testing	
Parameters	Values
Computing Server (iMac Pro)	6-Core Intel i5 3.1 GHz, RAM: 128GB
IoT devices gateway	Pi 4 model B, 4GB RAM, OS: Raspbian
Digital wallet	MetaMask v10.23.3
Dapp language, frameworks, and APIs	Python, Flask, web3.py, JSON-RPC API
Average encryption time (ms)	25.4

TABLE IV
SPACE AND DEPLOYMENT COSTS OF THE CORE SMART CONTRACTS

No	Smart contract	Deployment Cost		
		Size (KB)	Gas Used †	ETH
(1)	CAAC_Policy_Mgr.sol	5.15	2,090,226	0.00209
(2)	ICAAC_Policy_Mgr.sol	1.05	–	–
(3)	CAAC_Rule_Mgr.sol	12.99	5,057,248	0.00506
(4)	ICAAC_Rule_Mgr.sol	1.78	–	–

† (Gwei)

ECDSA,[2] RESTful, and Web3.py.[3] Metamask wallet[4] is used to manage cryptographic keys and create and verify transaction signatures. RPCs are used to interact with the blockchain network nodes through the Web3.py API in the smart contracts. Tessera is used as a private transaction manager. Raspberry Pi [25] serves as a gateway, allowing low-resource IoT devices to run the smart contracts integrated within the Dapps to interact with the blockchain.

B. Performance Evaluation

The proposed SC-CAAC system is evaluated using the following performance metrics.

1) Space Complexity and Deployment Costs: This analysis examines the storage requirements and costs associated with smart contract deployment, noting the importance of striking a balance between resource usage and costs within a blockchain network. Table IV summarizes the space and deployment costs of the core SC-CAAC smart contracts. The *CAAC_Rule_Mgr.sol* smart contract is the costliest, using 12.99 kB and requiring 5 057 248 Gwei of gas, which is equivalent to 0.00506 ETH. *CAAC_Policy_Mgr.sol* is less costly, using 5.15 kB and 2 090 226 Gwei (0.00209 ETH) for deployment. The interfaces for these contracts, *ICAAC_Policy_Mgr.sol* and *ICAAC_Rule_Mgr.sol*, have sizes of 1.05 and 1.78 kB, respectively. Although they have important roles, they do not incur specific deployment costs due to their auxiliary nature. Interfaces can be used to define the methods and events available in smart contracts [48]. They are essential for interacting with deployed contracts from external scripts and applications. This cost evaluation provides insight into the resource allocation and efficiency of the core contracts in the SC-CAAC approach.

Table V summarizes the read and write time complexity and transaction execution gas costs for the main smart contract functions in SC-CAAC. The majority of the functions have a consistent time complexity of $O(1)$ for read operations. However, for rule write functions *setCAACRule, EnableRule, DisableRule, RevokeRule, authorizeUser*, and *removeUserAuthorization* have a slightly higher $O(2)$ complexity for these operations. For policy and contract setting write operations, functions *SetCAACPolicy, EnablePolicy, DisablePolicy, RevokePolicy, setCAACRule, EnableRule, DisableRule, RevokeRule, authorizeUser, removeUserAuthorization, DisableContract*, and *EnableContract* have a consistent $O(1)$ complexity. Of particular note, *getAllPolicies* and *getAllAuthorizedUsers* have linear time complexity $O(n)$ for read operations. This indicates that their execution time may increase proportionally with the amount of data processed.

[2]https://pypi.org/project/ecdsa/
[3]https://web3py.readthedocs.io/en/stable/
[4]https://metamask.io/

TABLE V
TIME COMPLEXITY AND TRANSACTION GAS COSTS OF KEY FUNCTIONS

No	Contract Functions	Read	Write	Gas Cost†	Cost in ETH
(1)	*SetCAACPolicy*	$O(1)$	$O(1)$	213,504	0.00021
(2)	*EnablePolicy*	$O(1)$	$O(1)$	45,548	0.00005
(3)	*DisablePolicy*	$O(1)$	$O(1)$	45,714	0.00005
(4)	*RevokePolicy*	$O(1)$	$O(1)$	45,943	0.00005
(5)	*GetPolicyInfo*	$O(1)$	–	–	–
(6)	*GetPolicyStatus*	$O(1)$	–	–	–
(7)	*GetPoliciesNumber*	$O(1)$	–	–	–
(8)	*getAllPolicies*	$O(n)$	–	–	–
(9)	*setCAACRule*	$O(2)$	$O(1)$	388,208	0.00034
(10)	*EnableRule*	$O(2)$	$O(1)$	63,957	0.00006
(11)	*DisableRule*	$O(2)$	$O(1)$	64,039	0.00006
(12)	*RevokeRule*	$O(2)$	$O(1)$	63,939	0.00006
(13)	*GetRuleInfo*	$O(1)$	–	–	–
(14)	*getRulePolicyID*	$O(1)$	–	–	–
(15)	*GetRuleStatus*	$O(1)$	–	–	–
(16)	*getRulesNumber*	$O(1)$	–	–	–
(17)	*authorizeUser*	$O(2)$	$O(1)$	94,344	0.00008
(18)	*removeUserAuthorization*	$O(2)$	$O(1)$	65,918	0.00005
(19)	*isUserAuthorized*	$O(1)$	–	–	–
(20)	*getAllAuthorizedUsers*	$O(n)$	–	–	–
(21)	*VerifyPolicyRule*	$O(1)$	–	–	–
(22)	*EnforcePolicyRule*	$O(1)$	–	–	–
(23)	*DisableContract*	$O(1)$	$O(1)$	25,977	0.00003
(24)	*EnableContract*	$O(1)$	$O(1)$	47,889	0.00005

† (Gwei)

Gas consumption quantifies the computational power and resources consumed when executing functions on an EVM-based blockchain network. For example, *setCAACRule*, with a gas consumption of 388 208 Gwei (equivalent to 0.00034 ETH), is relatively resource-intensive, illustrating its complexity in setting access control rules. Conversely, functions such as *EnablePolicy* and *DisablePolicy*, with gas costs of around 45 000 Gwei (0.00005 ETH), are more streamlined operations with modest resource consumption. Gas costs are also a direct indicator of transaction fees, and higher costs can hinder the widespread adoption of smart contracts, especially in public blockchain networks, if users consider them exorbitant. Unlike write operations, read operations do not incur any gas costs because they do not alter the ledger state.

2) Transaction Time Complexity and Latency Time: The time complexity and latency for transaction processing are measured to assess the efficiency and responsiveness of the system, which are vital for applications that prioritize the processing performance.

1) *CAACL On-Chain Policy and Rule Setting Operations:* Fig. 4 presents the latency times of various CAACL on-chain policy-setting operations. The overall policy setting registration latency is significantly affected by the transaction execution time, with an average of 3833 ms for *SetNewPolicy*, 4569 ms for *EnablePolicy*, 4679 ms for *DisablePolicy*, and 4659 ms for *RevokePolicy*. Although *EnablePolicy* has the highest latency at 4618 ms and *SetNewPolicy* the lowest at 3971 ms, the other operations—*DisablePolicy* and *RevokePolicy*—are similar at 4808 and 4807 ms, respectively. This consistency in the sending, signing, and preparation times across these operations highlights the robustness of our proposed system regardless of the specific policy transaction involved.

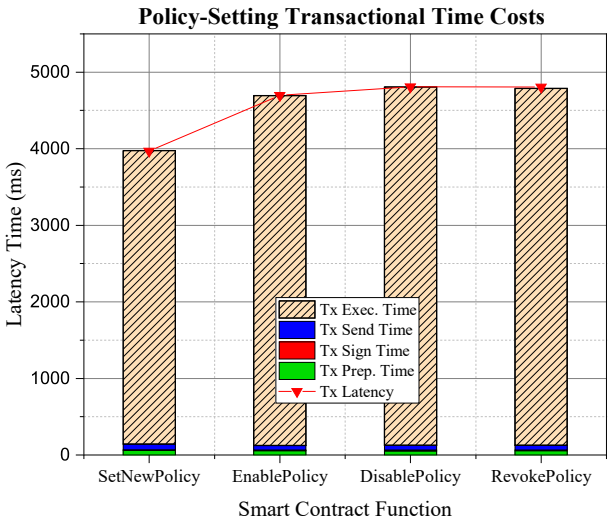

Fig. 4. Average latency time for SC-CAACL on-chain policy setting by function: 1) *SetNewPolicy*; 2) *EnabledPolicy*; 3) *DisablePolicy*; and 4) *RevokePolicy*.

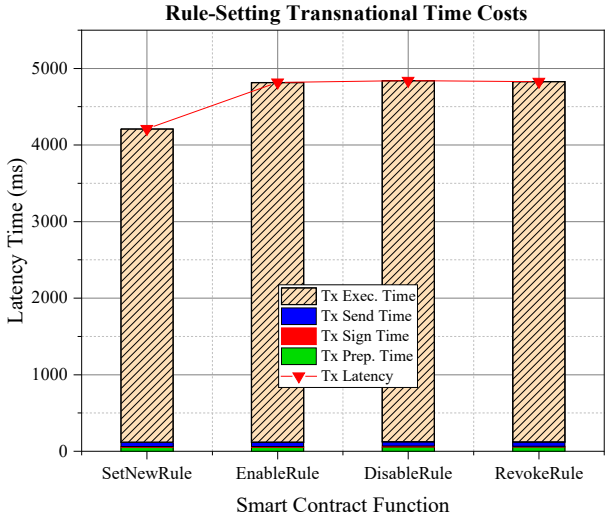

Fig. 5. Average latency time for SC-CAACL on-chain policy rule setting by function: 1) *SetNewRule*; 2) *EnabledRule*; 3) *DisableRule*; and 4) *RevokeRule*.

Fig. 5 presents the latency time for the on-chain SC-CAACL policy rule-setting transactions *SetNewRule, EnableRule, DisableRule*, and *RevokeRule* divided into individual actions. The execution of the transaction requires the longest time at 4091, 4698, 4716, and 4705 ms, respectively, while the auxiliary times for the other actions are consistent across operations (54–57 ms for sending, 10–12 ms for signing, and 53–56 ms for preparation). The overall latency is 4212, 4819, 4841, and 4829 ms for *SetNewRule, EnableRule, DisableRule*, and *RevokeRule*, respectively. These results represent a consistent system performance across various rule-setting operations, with the transaction execution time being the primary influencing factor.

2) *CAACL Policy and Rule On-Chain Query Operations:* Fig. 6 presents the latency time for various smart contract functions associated with CAACL policy queries.

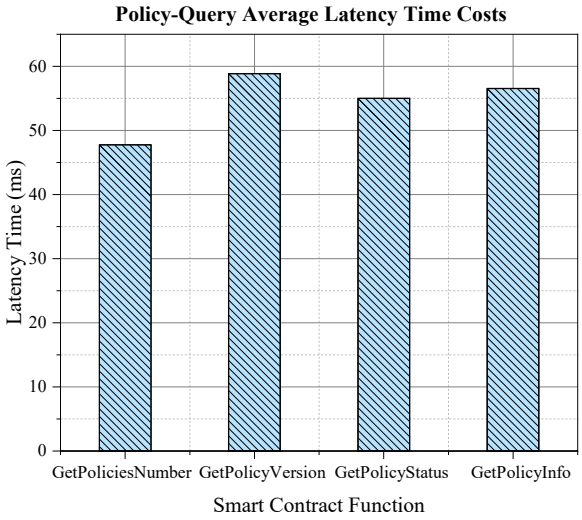

Fig. 6. Average latency time for SC-CAACL on-chain policy queries by function: 1) *GetPoliciesNumber*; 2) *GetPolicyVersion*; 3) *GetPolicyStatus*; and 4) *GetPolicyInfo*.

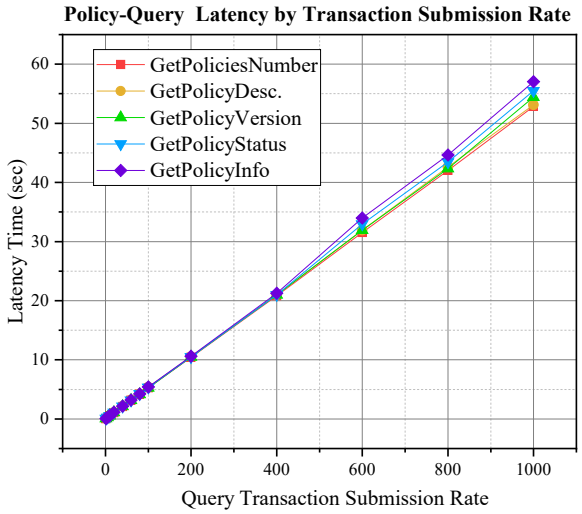

Fig. 7. Latency time for SC-CAACL on-chain policy queries by transaction submission rate: 1) *GetPoliciesNumber*; 2) *GetPolicyVersion*; 3) *GetPolicyStatus*; and 4) *GetPolicyInfo*.

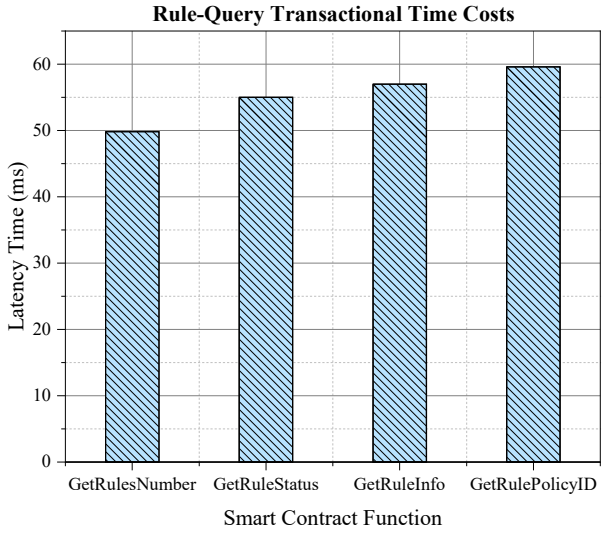

Fig. 8. Average latency time for SC-CAACL on-chain rule queries by function: 1) *GetRulesNumber*; 2) *GetRuleStatus*; 3) *GetRuleInfo*; and 4) *GetRulePolicyID*.

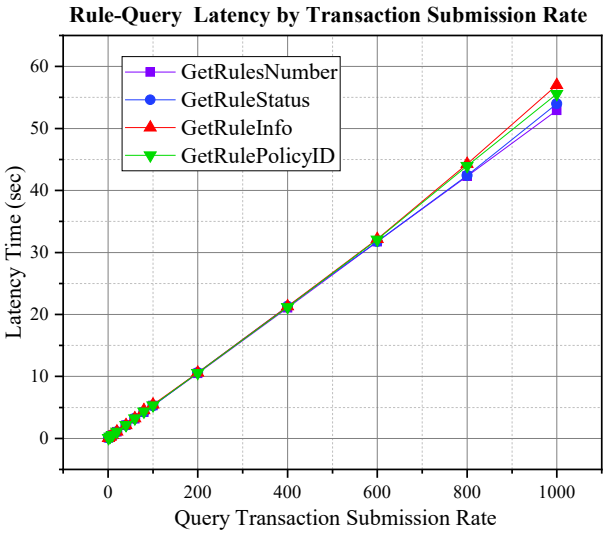

Fig. 9. Average latency time for SC-CAACL on-chain policy rule queries according to the transaction submission rate by function: 1) *GetRulesNumber*; 2) *GetRuleStatus*; 3) *GetRuleInfo*; and 4) *GetRulePolicyID*.

Of the assessed functions, *GetPoliciesNumber* has the lowest latency, with an average of 48 ms, while that of both *GetPolicyVersion* and *GetPolicyStatus* is 55 ms and that of *GetPolicyInfo* is 57 ms. This represents a relatively consistent response time across different policy-related queries. Fig. 7 shows the impact of the transaction submission load on the transaction latency for different CAACL policy-related query functions. As the transaction rate submission increases from 0 to 1000, the latency time for all functions increases linearly (*GetPoliciesNumber* from 2.10 to 52.82 s, *GetPolicyVersion* from 2.13 to 54.43 s, *GetPolicyStatus* from 2.13 to 55.49 s, and *GetPolicyInfo* from 2.16 to 57.03 s). The performance is thus consistent for all functions, with only minor variations, especially at higher transaction loads. Nevertheless, the higher latency with higher transaction loads for policy-related queries highlights the challenges facing scalability, meaning that optimization is required to improve this performance under high demands. Fig. 8 presents the transaction latency time for smart contract functions related to CAACL policy rule queries. *GetRulesNumber* has the lowest latency at 50 ms, increasing to 55 ms for *GetRuleStatus*, 57 ms for *GetRuleInfo*, and 60 ms for *GetRulePolicyID*. This indicates potential variation in the computational intensity and/or data access complexities of each function. Fig. 9 presents the change in the latency time in response to the transaction submission rates for rule-related queries. *GetRulesNumber, GetRuleStatus, GetRuleInfo*, and *GetRulePolicyID* all exhibit a direct and linear increase in latency as the

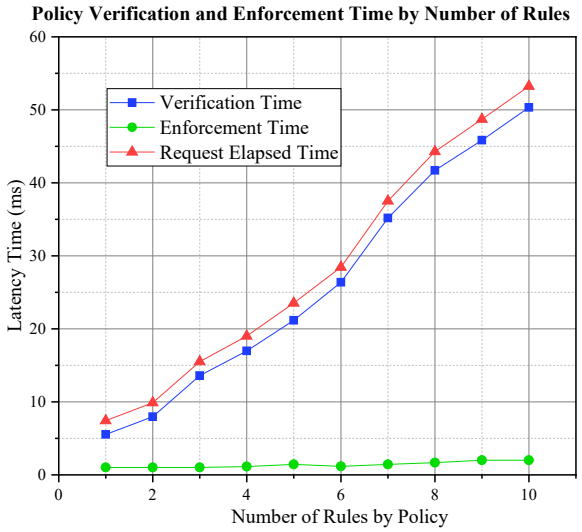

Fig. 10. Average latency time for SC-CAACL policy verification and enforcement according to the number of rules by policy.

submission rates increase. At a submission rate of 1000, the latency converges to around 55–60 s, while they remain slightly above 0 s at the lowest submission rate. This highlights the efficient query processing of the proposed system at low transactional loads, but scalability may be difficult at higher transaction loads. This indicates a need for further optimization to ensure robust performance.

3) *CAACL Policy Verification and Enforcement Times:* Evaluating the policy verification, enforcement, and overall elapsed time is crucial to understand the efficiency and scalability of the system. The policy verification time measures the time it takes to verify whether an access request satisfies predefined CAACL policy rules, while the enforcement time is the amount of time required to make an access decision. The overall elapsed time includes both verification and enforcement, as well as decryption and system I/O requests. The average policy verification, enforcement, and overall elapsed times for the CAACL system are 26.47, 1.39, and 28.76 ms, respectively. As shown in Fig. 10, the policy verification time increases with a higher number of rules (e.g., 5.53 ms for 1 rule to 50.33 ms for 10 rules). This nearly linear relationship illustrates the direct impact of rule complexity on the verification process. In contrast, the enforcement time remains consistent with an increase from 1.01 ms for 1 rule to 2.00 ms for 10 rules. This stability indicates the robustness of the enforcement mechanism. The overall elapsed time also increases with the number of rules in a pattern similar to the verification time. These results suggest that the verification time is a critical determinant of the overall policy processing latency.

3) Block-Related Metrics: Block-related parameters such as block size, propagation time, and processing efficiency are important in determining the overall performance and stability of a blockchain network. In the Hyperledger Besu blockchain network using the IBFT 2.0 consensus algorithm, block creation and validation follow a defined protocol tailored for enterprise environments. As summarized in Table III, blocks are proposed every 5 s, and the validator set is updated every 30 000 slots. A validator awaits a proposal or vote for 10 s before continuing. To simplify our analysis, each block carries one transaction with an average size of 845.21 bytes. This simplified transactional density allows for the more straightforward scalability analysis and prediction of network bandwidth requirements. However, this may differ from real-world applications depending on the use case, transactional demands, and network conditions. A block gas limit of 16 234 336 is set, and experiment results show that an average of 1182 gas was used per block.

V. Use Cases and Applications

This section presents a variety of use cases for the SC-CAAC scheme to highlight its broad applicability and effectiveness in various BIoT environments.

1) *Smart Home Automation:* SC-CAAC can be used to enhance the security and convenience of smart homes [12], [15], [25]. For example, access to home security systems can be controlled based on contextual data such as the homeowner's location or time of day. Smart contracts can automatically adjust permissions for devices such as smart locks or alarms based on these contexts.
2) *Smart Healthcare:* In smart healthcare [19], [23], [32], SC-CAAC can manage access to patient records and medical devices. Depending on the role of the user (doctor, nurse, or family member), location, and time, access to patient data can be controlled to ensure privacy and compliance with regulations such as HIPAA.[5]
3) *Supply Chain Management:* In blockchain-based supply chain systems [33], [42], SC-CAAC can be utilized to control access to the tracking of the movement of goods. Smart contracts can provide real-time, conditional access to data for different stakeholders (i.e., suppliers, transporters, and retailers) based on their role and the stage of the supply chain involved.
4) *IIoT:* In IIoT settings [34], [35], [36], SC-CAAC can ensure that only authorized users have access to the control of critical machinery. Access can be dynamically managed based on factors such as employee credentials, current machine status, and environmental conditions.
5) *Energy Systems:* In smart grids [25], [37], SC-CAAC can regulate access to energy usage data and control systems. This ensures data integrity and the secure operation of energy distribution, allowing for context-based access for maintenance, monitoring, or emergency responses.
6) *Smart Cities:* In smart city applications [40], [41], [42], [43], SC-CAAC can be used to manage access to various urban services such as public transportation, waste management, and city surveillance systems, adapting to different user roles and environmental contexts.

[5]Health Insurance Portability and Accountability Act.

These examples illustrate the versatility and practicality of our proposed scheme in real-world settings.

VI. Discussion

Though the proposed SC-CAAC system significantly improves the security and privacy of BIoT systems, it also has some limitations and challenges that need to be addressed.

1) *Complexity:* Designing and implementing CAAC policies within smart contracts can be complex and difficult. Contextual information in IoT environments, such as the user profile, device status, location, and time, is diverse and dynamic. Capturing and interpreting this information accurately within smart contracts can be challenging and thus requires robust modeling methods and efficient processing algorithms.
2) *Secure Smart Contract Development:* Developing smart contracts requires specific skills and care to prevent bugs and security vulnerabilities. Once deployed, smart contracts are difficult to modify, making it difficult to correct any mistakes or adapt to new requirements. Guidelines and tools for efficient and secure smart contract design and development are thus required.
3) *Scalability:* An increase in the number of IoT devices in a system can cause scalability issues. Because every transaction or access request has to be processed by smart contracts and recorded on the blockchain, it can lead to high latency, particularly when the network is large. As the number of IoT devices and transactions increases, the computational and storage requirements for executing smart contracts grow. Ensuring efficient and scalable execution of CAAC policies while maintaining the performance of the blockchain network is thus crucial.
4) *Performance and Efficiency:* Blockchains are computationally expensive and may cause delays in the processing of access requests, leading to slower response times. This can be a challenge in real-time systems where quick decisions are necessary. BIoT systems may have high transaction volumes, which may result in network congestion, high latency, and higher transaction costs.
5) *Computation and Energy Efficiency:* Public blockchains using PoW as their consensus mechanism are known for their high computing and energy requirements, raising sustainability concerns. For IoT devices with limited processing capabilities, this can result in significant computational overhead. This article focuses on the IBFT consensus algorithm, but we also intend to explore other consensus algorithms, such as Proof of Stake (PoS) and Proof of Authority (PoA) variants (e.g., Clique and QBFT) that are supported by Hyperledger Besu.
6) *Interoperability*: Manufacturers produce a wide variety of IoT devices with different standards, protocols, and data formats. Ensuring seamless integration and interoperability between heterogeneous IoT systems and various blockchain systems is important for widespread adoption and effective implementations.
7) *Privacy Concerns:* Though they can enhance data integrity and security, the transparency of blockchain systems can pose privacy challenges. Contextual information used for access control decisions can be sensitive and confidential. Protecting the privacy of this information, along with the associated access control policies and decision logs stored on the blockchain, requires robust security mechanisms and privacy-preserving techniques. Data obfuscation must occur before recording information on a blockchain.
8) *Legal and Regulatory Issues:* The use of smart contracts and blockchains also raises legal and regulatory questions. Regulations and legal frameworks for smart contract enforcement in BIoT systems are still evolving, leading to potential legal uncertainty.

VII. Conclusion and Future Work

This article proposes an SC-CAAC scheme for BIoT systems. The proposed scheme addresses the challenges of ensuring adequate access control in dynamic BIoT environments by considering contextual information in access control decisions. The decentralization, immutability, and transparency of a blockchain system provide a robust framework for the definition and enforcement of CAAC policies for access permission management in BIoT systems. CAAC policies are dynamically and securely enforced using smart contracts, ensuring that only authorized users can access sensitive data and resources. The proof of concept built on the Hyperledger Besu blockchain presented in this article validates the effectiveness and scalability of the proposed scheme in real-world scenarios. Furthermore, several use cases are described to illustrate the adaptability and applicability of the proposed scheme in various BIoT systems. Thus, this research contributes to the field by presenting a groundbreaking approach that employs smart contracts for CAAC in BIoT systems. This approach not only enhances the security and privacy of these systems but also fosters a higher degree of trust and accountability through decentralized access control mechanisms. Our research thus lays the groundwork for potential improvements in secure smart contract design for IoT, context-aware policy modeling, analysis, and enforcement optimization. It can also be used to devise scalability solutions for blockchains with multiple consensus protocols and to design privacy-preserving techniques and interoperability protocols. These advancements are required to maintain and improve the efficiency and effectiveness of SC-CAAC in evolving BIoT environments.

Appendix

Fig. 11 depicts the unified modeling language (UML) representation of the smart contract architecture implemented for our proposed SC-CAAC scheme. It includes two critical smart contracts, *CAAC_Police_Mgr* and *CAAC_Rule_Mgr*, which are vital to the architecture and operation of the SC-CAAC system. *CAAC_Police_Mgr* implements essential elements

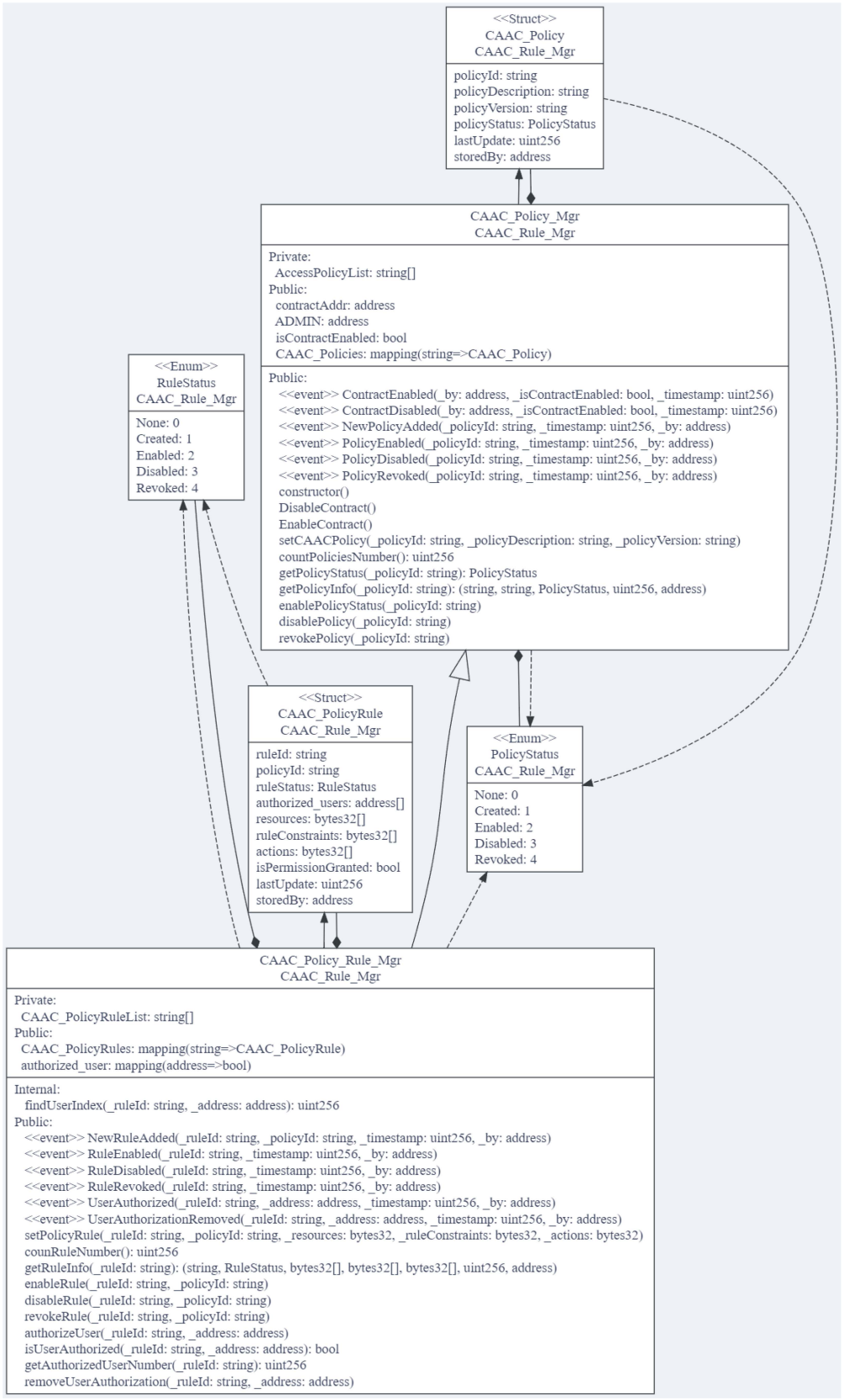

Fig. 11. SC-CAAC core smart contract architecture model representation in UML.

of the CAAC policies, such as policy identifiers, descriptions, versions, statuses, and update timestamps, along with the addresses modifying these policies. *CAAC_Police_Mgr* implements the rules linked to these policies, specifying rule identifiers, associated policies, operational states, and permission statuses. Key to this structure are functions and

events for policy and rule management, alongside *PolicyStatus* and *RuleStatus* enumeration to define their potential states. Based on its structure and functions, the proposed SC-CAAC system thus has the ability to manage access control in BIoT systems.

REFERENCES

[1] K. Rose, S. Eldridge, and L. Chapin, "The Internet of Things: An overview," in *Proc. Internet Soc. (ISOC)*, 2015, pp. 1–53.

[2] S. Sicari et al., "Security, privacy and trust in Internet of Things: The road ahead," *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015.

[3] C. Maple, "Security and privacy in the Internet of Things," *J. Cyber Policy*, vol. 2, no. 2, pp. 155–184, 2017.

[4] V. V. Jog and T. Senthil Murugan, "A critical analysis on the security architectures of Internet of Things: The road ahead," *J. Intell. Syst.*, vol. 27, no. 2, pp. 149–162, 2018.

[5] S. Gusmeroli, S. Piccione, and D. Rotondi, "IoT access control issues: A capability based approach," in *Proc. 6th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.*, 2012, pp. 787–792.

[6] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the Internet of Things: Big challenges and new opportunities," *Comput. Netw.*, vol. 112, pp. 237–262, Jan. 2017.

[7] R. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.

[8] P. Shantanu, M. Hitchens, and V. Varadharajan, "Towards a secure access control architecture for the Internet of Things," in *Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN)*, 2017, pp. 219–222.

[9] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8076–8094, Oct. 2019.

[10] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[11] L. Golightly, P. Modesti, R. Garcia, and V. Chang, "Securing distributed systems: A survey on access control techniques for cloud, blockchain, IoT and SDN," *Cyber Security Appl.*, vol. 1, Dec. 2023, Art. no. 100015.

[12] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2017, pp. 618–623.

[13] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[14] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in Internet-of-Things: A survey," *J. Netw. Comput. Appl.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[15] W. He et al., "Rethinking access control and authentication for the home Internet of Things (IoT)," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 255–272.

[16] E. Coyne, and T. R. Weil, "ABAC and RBAC: Scalable flexible and auditable access management," *IT Prof.*, vol. 15, no. 3, pp. 14–16, 2013.

[17] M. M. Merlec, Y. K. Lee, and H. P. In, "SmartBuilder: A block-based visual programming framework for smart contract development," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, 2021, pp. 90–94.

[18] R. Xu, Y. Chen, and E. Blasch, "Decentralized access control for IoT based on blockchain and smart contract," in *Modeling and Design of Secure Internet of Things*. Hoboken, NJ, USA: Wiley, 2020, pp. 505–528.

[19] M. Tuler De Oliveira, L. H. A. Reis, Y. Verginadis, D. M. F. Mattos, and S. D. Olabarriaga, "SmartAccess: Attribute-based access control system for medical records based on smart contracts," *IEEE Access*, vol. 10, pp. 117836–117854, 2022.

[20] A. S. M. Kayes et al., "A survey of context-aware access control mechanisms for cloud and fog networks: Taxonomy and open research issues," *Sensors*, vol. 20, no. 9, p. 2464, 2020.

[21] M. J. Covington and M. R. Sastry, "A contextual attribute-based access control model," in *Proc. OTM Confed. Int. Conf. Move Mean. Internet Syst.*, 2006, pp. 1996–2006.

[22] Y. Verginadis et al., "Context-aware policy enforcement for PaaS-enabled access control," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 276–291, Jan.–Mar. 2022.

[23] E. Psarra, Y. Verginadis, I. Patiniotakis, D. Apostolou, and G. Mentzas, "Accessing electronic health records in critical incidents using context-aware attribute-based access control," *Intell. Decis. Technol.*, vol. 15, no. 4, pp. 667–679, 2021.

[24] E. Psarra, D. Apostolou, Y. Verginadis, I. Patiniotakis, and G. Mentzas, "Context-based, predictive access control to electronic health records," *Electronics*, vol. 11, no. 19, p. 3040, 2022.

[25] L. Ngwira et al., "Towards context-aware smart contracts for blockchain IoT systems," in *Proc. Int. Conf. Inf. Commun. Technol. Conver. (ICTC)*, 2021, pp. 82–87.

[26] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum, Zug, Switzerland, Yellow Paper, 2014.

[27] E. Androulaki et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[28] M. M. Merlec, M. M. Islam, Y. K. Lee, and H. P. In, "A consortium blockchain-based secure and trusted electronic portfolio management scheme," *Sensors*, vol. 22, no. 3, p. 1271, Feb. 2022.

[29] S. Algarni et al., "Blockchain-based secured access control in an IoT system," *Appl. Sci.*, vol. 11, no. 4, p. 1772, 2021.

[30] S. Namane and I. B. Dhaou, "Blockchain-based access control techniques for IoT applications," *Electronics*, vol. 11, no. 14, p. 2225, 2022.

[31] I.-H. Chuang et al., "TIDES: A trust-aware IoT data economic system with blockchain-enabled multi-access edge computing," *IEEE Access*, vol. 8, pp. 85839–85855, 2020.

[32] N. Chendeb, N. Khaled, and N. Agoulmine, "Integrating blockchain with IoT for a secure healthcare digital system," in *Proc. 8th Int. Workshop Adv. ICT Infrastruct. Services*, 2020, pp. 1–8.

[33] S. Salonikias et al., "Blockchain-based access control in a globalized healthcare provisioning ecosystem," *Electronics*, vol. 11, no. 17, p. 2652, 2022.

[34] C. Lin, D. He, X. Huang, K. R. Choo, and A. V. Vasilakos, "BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0," *J. Netw. Comput. Appl.*, vol. 116, pp. 42–52, Aug. 2018.

[35] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Computers*, vol. 7, no. 3, p. 39, 2018.

[36] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Exploration of blockchain-enabled decentralized capability-based access control strategy for space situation awareness," *Opt. Eng.* vol. 58, no. 4, pp. 041609–041609, Feb. 2019.

[37] M. Amine Bouras, B. Xia, A. Omer Abuassba, H. Ning, and Q. Lu, "IoT-CCAC: A blockchain-based consortium capability access control approach for IoT," *PeerJ Comput. Sci.*, vol. 7, p. e455, Apr. 2021.

[38] M. M. Merlec, Y. K. Lee, S.-P. Hong, and H. P. In, "A smart contract-based dynamic consent management system for personal data usage under GDPR," *Sensors*, vol. 21, p. 7994, Nov. 2021.

[39] M. U. Rahman, B. Guidi, F. Baiardi, and L. Ricci, "Context-aware and dynamic role-based access control using blockchain" in *Advanced Information Networking and Applications*. Cham, Switzerland: Springer, pp. 1449–1460, 2020.

[40] T. Sylla et al., "Blockchain-based context-aware authorization management as a service in IoT," *Sensors*, vol. 21, no. 22, p. 7656, 2021.

[41] B. Annane, A. Alti, and A. Lakehal, "Blockchain based context-aware CP-ABE schema for Internet of Medical Things security," *Array*, vol. 14, Jul. 2022, Art. no. 100150.

[42] A. Sarfaraz, R. K. Chakrabortty, and D. L. Essam, "AccessChain: An access control framework to protect data access in blockchain enabled supply chain," *Future Gener. Comput. Syst.*, vol. 148, pp. 380–394, Nov. 2023.

[43] I. H. Abdulqadder and S. Zhou, "SliceBlock: Context-aware authentication handover and secure network slicing using DAG-blockchain in edge-assisted SDN/NFV-6G environment," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 18079–18097, Sep. 2022.

[44] OASIS Standard. "Extensible access control markup language (XACML) version 3.0." Jan. 22, 2013. Accessed: Dec. 28, 2023. [Online]. Available: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html

[45] OASIS Standard. "JSON Profile of XACML 3.0 Version 1.1." Jun. 20, 2019. Accessed: Dec. 28, 2023. [Online]. Available: https://docs.oasis-open.org/xacml/xacml-json-http/v1.1/xacml-json-http-v1.1.html

[46] C. F. F. Karney, "Algorithms for geodesics," *J. Geodesy*, vol. 87, no. 1, pp. 43–55, 2013.

[47] C. Fan, C. Lin, H. Khazaei, and P. Musilek, "Performance analysis of hyperledger Besu in private blockchain," in *Proc. IEEE DAPPS*, 2022, pp. 64–73.

[48] C. Dannen, "Solidity programming," in *Introducing Ethereum and Solidity*. Berkeley, CA, USA: Apress, 2017, pp. 69–88.

Mpyana Mwamba Merlec (Member, IEEE) received the B.Sc. degree in computer science and engineering from the Information Systems Engineering Department, University Protestant of Lubumbashi (UPL), Lubumbashi, D.R.Congo, in 2011, and the M.S.E. degree in computer science and radio communication engineering (with Academic Excellence Awards from the NIIED, Ministry of Education, Republic of Korea) and the Ph.D. degree in computer science and engineering (software) from Korea University, Seoul, South Korea, in 2017 and 2022, respectively.

He is currently a Research Professor with the Blockchain Research Institute, Korea University. His primary research interests include Web 3.0, distributed computing systems, distributed ledger technologies, distributed/decentralized applications, and the following topics: blockchain-enabled FinTech, e-commerce, supply chains, cryptocurrency algorithmic trading, cybersecurity, data privacy, GDPR compliance, machine learning, Internet of Things, and cyber–physical systems.

Hoh Peter In (Member, IEEE) received the B.Sc. degree in computer engineering and the M.Sc. degree in computer science from Korea University, Seoul, South Korea, in 1990 and 1992, respectively, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1998.

In 1999, he became an Assistant Professor with Texas A&M University at College Station, College Station, TX, USA. He joined the Department of Computer Science, Korea University as an Assistant Professor in 2003, where he is currently a Professor. He is a Founder and the Emeritus President of the Korean Society of Blockchain and the Director of the Blockchain Research Institute, Seoul. He is also the founder and CEO of DAO Solution, Inc., Seoul. He has published over 120 research papers. His main research interests are blockchain, smart contracts, and software engineering.

Prof. In received the ICRE 10-Year Most Influential Paper Award in 2006.