A Fine-grained Access Control Model with Enhanced Flexibility and On-chain Policy Execution for IoT Systems

Hoang-Anh Pham[1], Ngoc Nhuan Do[2], Nguyen Huynh-Tuong[3]
Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam[1,2]
Vietnam National University Ho Chi Minh City (VNU-HCM), Linh Trung Ward, Ho Chi Minh City, Vietnam[1,2]
Industrial University of Ho Chi Minh City (IUH), 12 Nguyen Van Bao, Go Vap District, Ho Chi Minh City, Vietnam[3]

***Abstract*—Blockchain-based access control mechanisms have garnered significant attention in recent years due to their potential to address the security and privacy challenges in the Internet of Things (IoT) ecosystem. IoT devices generate massive amounts of data that are often transmitted to cloud-based servers for processing and storage. However, these devices are vulnerable to attacks and unauthorized access, which can lead to data breaches and privacy violations. Blockchain-based access control mechanisms can provide a secure and decentralized solution to these issues. This paper presents an improved Attribute-based Access Control (ABAC) approach with enhanced flexibility, which utilizes decentralized identity management on the Substrate Framework, codifies access control policies by Rust programming language, and executes access control policies on-chain. The proposed design ensures trust and security while enhancing flexibility compared to existing works. In addition, we implement a PoC to demonstrate the feasibility and investigate its effectiveness.**

***Keywords*—Attribute-based Access Control (ABAC); Internet of Things (IoT); blockchain; substrate framework**

I. Introduction

Access control is a security approach that governs who or what has access to and uses resources in a computing environment. The primary objective of access control is to reduce the hazards of unauthenticated system access while protecting personal information. Therefore, most computing applications require access control services to control and prohibit unauthorized access to system resources such as networks, devices, files, or sensitive data. Meanwhile, the number of connected IoT devices is rapidly increasing due to the maturation of connecting protocols for IoT (e.g., BLE, LoRa, NB-IoT, LTE, 5G, and 6G). In addition, the growth of Big Data and Artificial Intelligence also motivates data collection from the physical environment by adopting IoT infrastructures. However, this means that security in IoT becomes more critical because IoT systems can yield a lot of sensitive data [1][2][3][4]. For example, a faulty firmware of a camera vendor caused millions of camera devices of clients to be exposed publicly to the Internet, and malicious parties can exploit resources legitimately. Therefore, access control employment is an essential solution to improve the security of IoT systems.

Most conventional access controls for IoT are based on a centralized architecture with many limitations, such as single-point-of-failure, trusted third-party requirements, and low scalability [5][6][7]. Meanwhile, the maturity of Blockchain drives towards applying to numerous areas beyond cryptocurrencies to solve concerns of trust and security, such as digital certificate [8], smart factory [9], smart parking [10], healthcare [11], and traceability [12]. Additionally, there have been various research studies on the amalgamation of Blockchain to solve problems in existing IoT systems regarding scalability, interaction, security, privacy, and trust [13][14][15][16][17]. However, many aspects must be considered when applying Blockchain to conventional access control methods for IoT systems [18][19]. Due to heterogeneity and scenario variety in IoT, coarse-grained access control schemes, such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-based Access Control (RBAC), become cumbersome in administration. Nevertheless, fine-grained access control schemes, such as Attribute-based Access Control (ABAC), not only provide flexible administration but also guarantee security [20].

In the ABAC scheme, access control is evaluated by attributes (e.g., subject, object, and environmental attributes) and instructed by codifiable policies. Those features produce ABAC's advantages but make it challenging to design, especially on Blockchain platforms. An ABAC design should consider two main parts: attribute management and policy execution. Besides the scheme aspect, identity management is also crucial in access control. Thanks to the decentralized identity (DID) standard, participants own DIDs associated with their human-readable information on Blockchain, which can be resolved to DID Document (DDO). This standard also facilitates authentication (DID Auth), which is generally necessary before authorization or access validation. Nowadays, existing Blockchain-based ABAC solutions for IoT still have several limitations. They do not ensure both security and flexibility. Moreover, several solutions based on ancient Blockchain platforms, such as Bitcoin and Ethereum, need to be improved in terms of scalability for massive IoT systems.

In this paper, we propose an improved ABAC-based approach developed on the Substrate Blockchain Framework, which ensures trust and security while enhancing flexibility compared to existing ABAC-based works. The main contributions of this paper can be summarized as follows:

- Propose an alternative design of a blockchain-based access control approach that includes improved features compared to similar works.
- Present the implementation of a proof-of-concept in detail to demonstrate the feasibility and evaluate the

effectiveness of the proposed design.

The rest of this paper is organized as follows. Section II summarizes related works to clarifies the scope of our study. Then, Section III describes our proposed system. The implementation and evaluation are presented in Section IV. Finally, Section V provides concluding remarks and future works.

II. Related Works

A. Conventional Access Control Approaches

Discretionary Access Control (DAC) [21] is an identity-based access control model that gives users specific permissions to control their resources and data. Data owners can set access permissions for another user or group of users. These permissions are usually stored in an access control list. DAC is a simple and highly granular design because it allows users to freely configure access parameters on each data sample. However, it becomes a disadvantage and makes it for administrators more challenging to operate and maintain a more extensive system with a variety of users, data, and access configurations.

Mandatory Access Control (MAC) is a hierarchical model determined by the security level. In this model, each user is granted a security level, and each object is assigned a security label. A user can only access the according resource with a security label equal to or lower than that user's security level. In addition, this access control model also gives administrators complete control over access while users cannot configure their permissions. Therefore, the MAC model has a very high security. However, MAC-based systems can become quite unmanageable since administrators must configure permissions to all users and objects, leading to being overwhelmed if the system grows too fast.

Role-based Access Control (RBAC) [22] is a model based on user roles and responsibilities. Instead of granting access to users, RBAC gives access to roles that are then used to grant users the rights to access the system resources according to granted roles. RBAC has several variants, including flat RBAC, hierarchical RBAC, and constrained RBAC. The RBAC is suitable for small and medium systems because its static property is unsuitable for systems that grant access according to dynamic parameters.

Attribute-based Access Control (ABAC) [23] provides fine-grained and contextual access control capabilities based on attributes of users, system resources, and environment. The ABAC scheme allows administrators to define access control policies without prior knowledge of specific access objects. In addition, it can provide dynamic access control because it allows the use of environmental attributes, such as time, location, or IP address. Access decisions can be changed between access requests as attributes change. However, the ABAC scheme has low visibility (i.e., it is difficult to determine the privileges of a particular user) because it works based on attributes from many different sources. This also makes it challenging to identify security risks in the whole system.

Capability-based Access Control (CapBAC) [24] is an access control model based on the capability that is a token holding the privileges granted to users. When a user wants to perform any actions on system resources, he has to send a request with his token to the service provider to check the validity of the token before deciding to allow or deny the request without verifying the requester's privileges because the token was published by a trusted server. This procedure makes the system not needing to maintain the users' list or the access policies list at the access points. However, issuing tokens by authentication servers and validating tokens at service providers might consume time in case of token withdrawal.

Access control models can be selected and deployed according to specific applications and conditions. Historically, centralized systems were often chosen to implement access control systems or through a third-party service provider. However, centralized systems are often limited regarding system scalability and the problem of single-point-of-failure, i.e., the risk at the centralized entity. In the meantime, allowing third parties to manage important security information, such as access control, can lead to information leakage risks and loss of user privacy. In addition, the system administrator has full control over the system, including manipulation of system usage history, which reduces the reliability and transparency of the entire access control system. However, Blockchain can tackle these limitations with prominent characteristics such as immutability, stability, audibility, and reliability.

B. Blockchain-based Access Control Approaches

Applying Blockchain technology to access control management for IoT systems is a direction with many potential benefits. In [25], the authors proposed an access control solution by adopting smart contracts to define access control contracts (ACC). Each pair of a subject and an object in the system has one ACC that stores the subject's permissions with the object resources. This also means the number of ACCs will increase exponentially as the IoT system expands. Some other works [26][27] proposed blockchain-based authentication and authorization mechanisms for IoT with multiple domains and parties as access control solutions. Meanwhile, in [28], the authors proposed a capability-based access control on Blockchain, which applies the Decentralized Identifier standard to identify the parties involved in the system, including resource owners, resource access requests, and devices. However, this method does not have flexibility in access administration because it is a coarse-grained access control scheme that usually becomes cumbersome in access control management with large IoT systems, especially with many changes.

As mentioned above, fine-grained access control schemes are suitable for IoTs but more complex when developing on Blockchain. The six following works are the most closely related to ours. In [29], the authors proposed a fine-grained access control framework based on Hyperledger Fabric, called Fabric-IoT. Attribute fields for users, devices, and policies are pre-defined, and administrators specify their values. Meanwhile, values of environment attributes are dynamically detected at the request processing time. In this work, a policy definition is limited to only being set values of the pre-defined attribute fields. In addition, policy execution for the relationship between attributes and access decisions is hard-coded in a smart contract. A similar design was presented by Song et al. in [30], but user and device attribute fields in this

work do not need to be pre-defined. These two Blockchain-based approaches are generally simplified from the original ABAC scheme, making them less dynamic and flexible.

In [31], attributes and policies are stored on a Hyperledger Fabric blockchain and an InterPlanetary File System. A policy execution is performed at distributed nodes in a network, called off-chain execution, and its final result is reduced with the Practical Byzantine Fault Tolerance (PBFT) algorithm. This approach does not take advantage of Blockchain for policy execution. Meanwhile, Maesa at al. [32] proposed a manner to transpile ABAC policies with XACML language to smart contracts with Solidity language, which can be executed on EVM-integrated Blockchain. Similarly, in [33], the authors proposed a scheme to interpret ABAC policies with XAMCL language to Blockchain transactions in JSON and scripting-logic expression. Those can be executed as Bitcoin scripts with several extended commands dedicated to collecting attribute information in the ABAC scheme.

Besides policy execution, attribute management is crucial in ensuring the ABAC scheme's security. These above-mentioned works took attributes of subjects and objects provided by third parties or managed by an administrator. However, in [34], the authors proposed a model for endorsing subject attributes on Blockchain. An entity, so-called a trusted entity, can issue or revoke endorsements for other entities' attributes. An attribute's trust is a value accumulated from the trust levels of the entities endorsing it. Trust levels of entities are maintained by a reputation system deployed on Blockchain. Because of focusing on attribute endorsement, this work did not take policy execution in the scope of its study.

To the best of the authors' knowledge, existing ABAC designs for IoTs on Blockchain have yet to ensure trust, security, and flexibility simultaneously. Therefore, this paper proposes an improved ABAC design for IoTs, which utilizes decentralized identity management on Blockchain, applies attribute endorsement to ensure attribute trust and security, and leverage smart contract to codify policy for enhancing flexibility. We define four criteria to highlight the improvement of the proposed design in terms of flexibility compared to six related methods, as shown in Table I.

- C1: Attributes are modifiable.
- C2: Policies are codifiable.
- C3: Policies are executed on-chain.
- C4: Attribute values are endorsed.

TABLE I. COMPARISON OF RELATED WORKS AND OURS

Criteria	[29]	[30]	[31]	[32]	[33]	[34]	**Ours**
C1	No	N/A	Yes	N/A	Yes	Yes	Yes
C2	No	No	Yes	Yes	Yes	N/A	Yes
C3	No	Yes	Yes	Yes	Yes	No	Yes
C4	No	No	Yes	No	No	Yes	Yes

III. THE PROPOSED APPROACH

IoT infrastructures typically consist of numerous devices deployed distributedly in the physical world. These devices can be categorized into end-devices, gateway, and IoT devices. As the largest and most distributed part among others, end-devices should be optimized in cost and energy consumption, letting them be neglected with battery power for a long time. In addition, end-devices are usually constrained in computing and storing capability, so they can not efficiently perform heavy cryptographic techniques to consolidate security. Moreover, low-power networks (LPWN) of wireless end-devices also have low bandwidth. Therefore, security methods will be mainly deployed on gateways since they employ electric grid power, high bandwidth internet connection, and more powerful computing and storage capacity. Besides, IoT devices, such as surveillance cameras, robots, or smartwatches, which have more powerful hardware configurations, are also considered to accommodate self-serving cryptographic security techniques. In the proposed design, we choose gateways as end-points for access control service, restricting requests from outside to inside resources for empowering security and privacy.

As the core of the proposed design, critical data and execution of access control are carried out on Blockchain to empower security and trust. We develop the proposed method on Substrate Framework and customize the Blockchain system for the access control services with three major obligations. First, Blockchain is an underlying infrastructure for a decentralized identifying system, facilitating authentication. Second, it provides methods for participants to manage their access control attributes on Blockchain. Third, it provides a distributed computing environment to manage and execute access control policies.

The proposed access control system comprises users with different roles and permissions, who can be divided into three types: regular users, trusted users, and administrators. The regular users include requesters who request access to resources and owners who own shared resources. The trusted users can endorse attributes of regular users, specifically requesters or subjects. The administrators are a minority in the system and are responsible for governing access to IoT resources through policies. With a large or global IoT system, there can be many multiple domains in which several administrators can manage each domain. This access control system is not tied to a specific IoT domain; in other words, it also supports multiple IoT domains. Furthermore, owners are considered to have sovereignty over their own IoT devices, which they may control locally and physically.

A. Security Assumptions

A security system is usually designed and built based on specific assumptions. Our proposed system will be developed based on four security assumptions (SA).

- SA1: Regarding physical devices such as IoT devices and gateways, they are assumed to operate reliably to protect themselves. It is noted that a device cannot be secured with only software solutions if a device is physically attacked and manipulated. However, a malicious device will not harm other trusted devices.

- SA2: The connection from IoT devices to the Blockchain system is also considered secure, allowing transactions from devices to reach the Blockchain or events returned from the Blockchain to propagate smoothly.
- SA3: Like other Blockchain systems, participants are responsible for the confidentiality of their Blockchain accounts and other private keys associated with their decentralized identity. In addition, users with special roles, such as administrators, are assumed to be trusted in their authority.
- SA4: Actual deployment conditions might influence Blockchain network topology and the selection of consensus algorithms. However, a Blockchain system must be distributed, immutable, and transparent as inherent characteristics.

Among these four assumptions above, except for the last one (SA4), when other assumptions are violated, it just locally affects; for example, a compromised IoT device or a disclosed Blockchain account, the security problems will be only affected locally.

B. System Design

An access control system should have flexibility to be easily adopted for numerous scenarios. The flexibility also makes access management more convenient for administrators, particularly in granting or revoking permissions. Hence, we chose the ABAC scheme that a fine-grained access control scheme. In addition, to consider the flexibility of an access control system, we scope our study in two following use-cases.

- In smart agriculture, combining low-power wireless sensor networks with automatic control systems facilitates agriculture precision. Commonly, an agricultural product has to go through many stages before reaching its consumer, such as planting, harvesting, processing, transporting, and retailing, hence the need for traceability. Blockchain technology allows parties from those stages to participate in a system to share data in trust. Each farm or each factory can have numerous IoT devices, employees, several managers, and one possessor. To control access to IoT devices, a possessor can delegate to managers; in turn, they will manage access permissions for employees through ABAC policies. In this use case, managers could endorse employees' attributes or delegate to their assistants as trusted endorsers. Environment attributes like DateTime can grant temporary permissions for seasonal employees. Third parties, such as business partners and inspection centers, may also be granted appropriate permissions to access to monitor activity.
- Another use-case is to manage access controls for an IoT camera system on a campus. These cameras can be rented to students who want to conduct related experiments such as video streaming. A CapBAC-based Blockchain approach has been proposed in [28]. Owners control tenant access by issuing or revoking capability tokens, and each token issuing requires the participation of both the owner and the tenant. When the system scales up, managing issued and revoked tokens becomes cumbersome and inflexible. Meanwhile, the ABAC design can provide more efficient access management for that case. Surveillance cameras can be divided into groups for easy management by defining attributes in a large system. Owners can grant permissions to tenants by endorsing their corresponding attributes. Note that a pre-defined policy just links the camera group to the tenant group. Besides that, environment attributes can help rent out based on time conveniently.

Fig. 1 depicts the system architecture of the proposed method on Substrate Framework that supports DID management and smart contracts via DID and Contract Pallets, respectively. In addition, we design the ABAC scheme as an **ABAC Pallet** and integrate it into the Substrate Framework. A typical Substrate's Pallet has two main parts: **storage declaration** and external methods (so-called **extrinsic definition**). Extrinsic calls only accomplish each updation to pallet storage, and the change is also attached to Blockchain. As the core of the proposed design, the ABAC Pallet has three storage declarations for ABAC Attributes, Endorsements, and Policy Attachments, described as follows.

- The first one is to store attributes (**Attribute Storage**) of subjects and objects according to their decentralized identities. Each ABAC attribute is in the form of key-value pair. With self-sovereign design thinking, an owner of a decentralized identity also owns associated ABAC attributes on Blockchain.
- The second one is to store endorsements (**Endorsement Storage**), which trusted endorsers confirmed for reviewed attributes of subjects. The endorsements make trusty for the current values of the subject's attributes. The endorsers can specify a validity period for their endorsements.
- The third one is storing objects' attachments with ABAC policies (**Policy Storage**), which is in the form of a one-to-many relationship. In the proposed design, an ABAC policy is a deployed smart contract with a unique address on the blockchain system. For those policies to be valid to an object, attachments need to be committed by the corresponding owner or administrators. Each attachment also holds a reminiscent name and logs its author.

Based on the storage declarations above, necessary extrinsic are designed to control ABAC attributes, attribute endorsements, and policy attachments from outside Blockchain. To ensure the caller has authority with corresponding data on Blockchain, these extrinsic all require a decentralized identifier owned by the caller as the first parameter. Other parameters in a specific extrinsic are selected to fit multiple demands.

- There are two extrinsic for users to control attributes associated with their DIDs, including **setAttributes()** and **clearAttributes()**. Extrinsic setAttributes() allows users to create and modify one or more attributes on Blockchain. If an attribute with its key does not exist, a creation will occur, and vice versa, and an attribute-value update will be activated. Note that once the

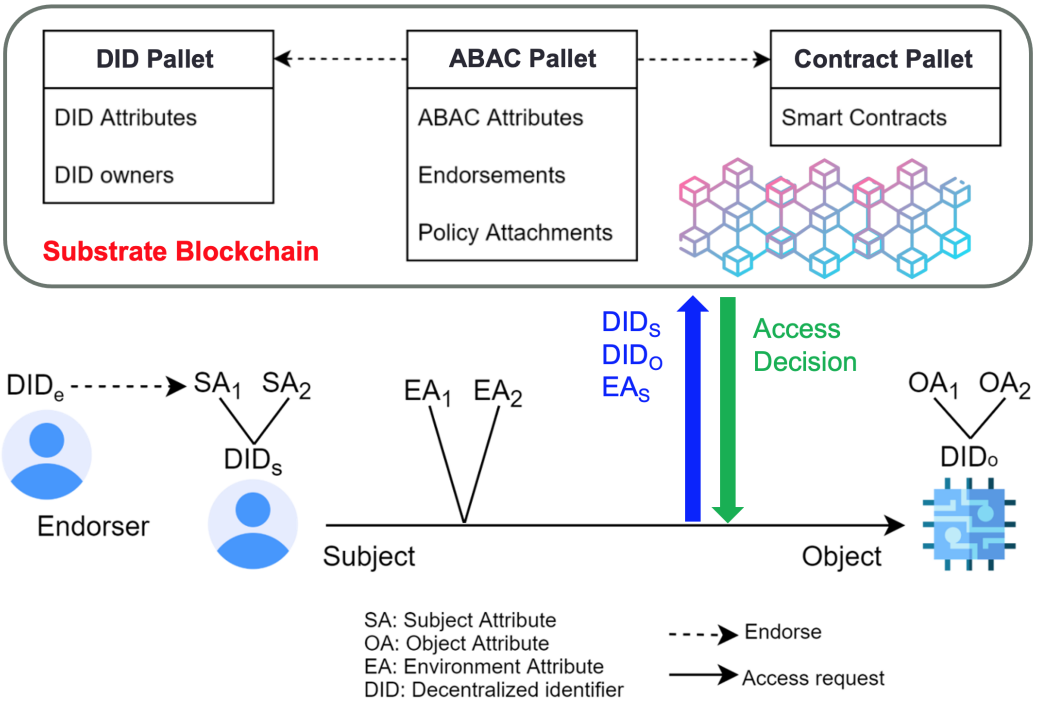

Fig. 1. System architecture.

value of an attribute is modified, all of its previous endorsements will no longer be valid, and then it will be deleted. Otherwise, extrinsic clearAttributes() is to delete the existing attributes of a decentralized identifier by specifying the corresponding attribute names.

- There are two extrinsic for trusted users to control endorsements of subject attributes, including **endorseAttributes()** and **unendorseAttributes()**. Endorsers can use endorseAttributes() to endorse multiple existing attributes of a subject with a specific validity duration. In the Substrate Framework, the value of that validity duration can be represented by a multiplier of the block finalization interval (e.g., 6 seconds by default). Otherwise, unendorseAttributes() allow endorsers to proactively delete their endorsements before expiring.
- There are two extrinsic for owners or administrators to manage policies of corresponding objects, including **attachPolicy()** and **detachPolicy()**. As mentioned above, a policy exists on Blockchain as a smart contract with a unique Blockchain address. Extrinsic attachPolicy() allows owners or administrators to connect an object and a policy. Furthermore, a reminiscent name for the attachment is also enabled, and the caller's DID is logged. Meanwhile, extrinsic detachPolicy() deletes the connection between an object and its policy.

C. Policy Execution Model

Access control policies define rules executed to make access decisions for requests. The policy execution should help flexibility in management but ensure security. In our design, the policy execution model coordinates on-chain and off-chain parts as described in Fig. 2. The off-chain part is performed outside the Blockchain system and is usually conducted on gateways. Policy Enforcement Point (PEP) is a middleware for receiving and handling access requests. When receiving an access request, PEP will forward that request to the Off-chain Context Handle (OFF-CH), and the Environment Attribute Detection (EAD) will derive related information as environment attributes. Subsequently, OFF-CH invokes on-chain policy execution, which contains the subject's DID, the object's DID, and environmental attributes. Once on-chain policy execution is completed, access decisions are returned to OFF-CH, which will forward them to the PEP. Based on those decisions, PEP enforces denying or allowing for the corresponding request.

The on-chain part is performed distributedly on Blockchain, where attributes of subjects and objects are stored and managed via distributed identities. Authority participants, including device owners or administrators, define policies of objects, which are also stored and managed on Blockchain. All on-chain data modifications are attached to the blocks. When the system receives an invocation of on-chain policy execution, the On-chain Context Handle (ON-CH) will forward it to Policy Decision Point (PDP) to execute corresponding policies. Note that which policies will be executed can be specified in the invocation or derived from the object's DID. During the execution, the ON-CH is responsible for fetching necessary attributes from the Subject/Object Attribute Authority to the PDP. Once completed, access decisions go from the PDP to the ON-CH and propagate to the OFF-CH through Blockchain events.

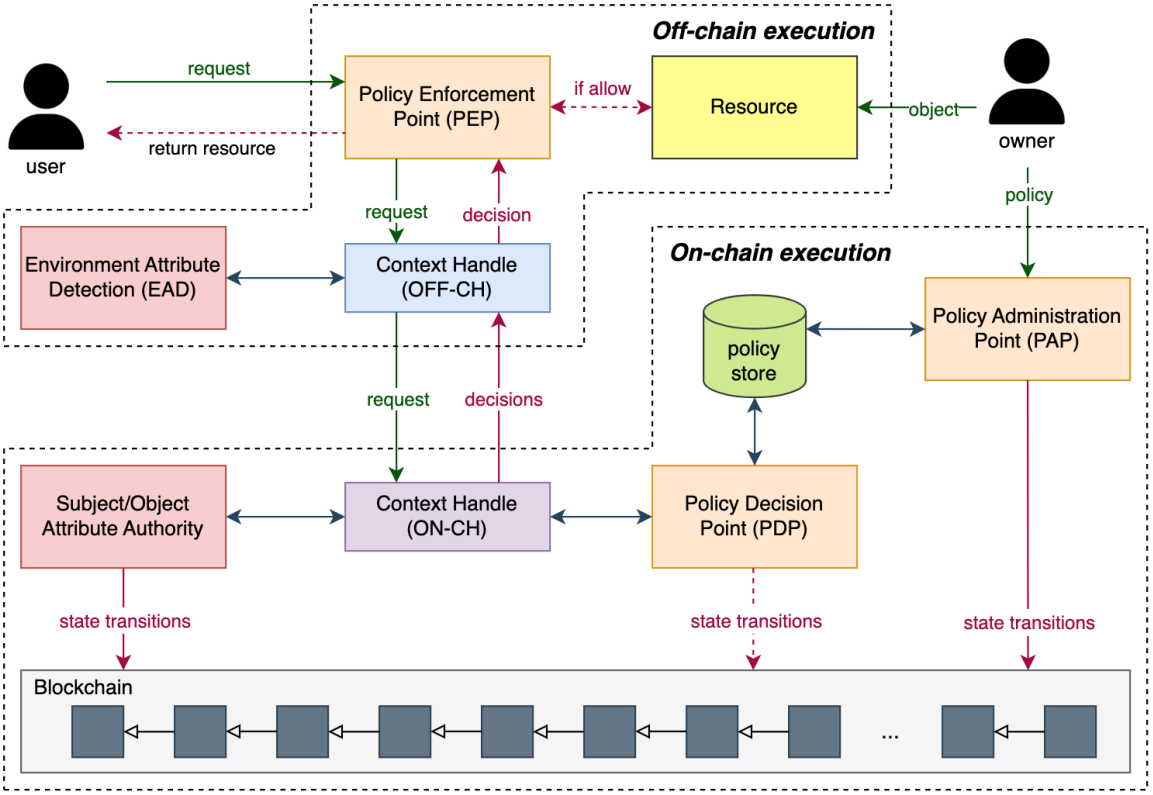

Fig. 2. Policy execution model.

A policy is a combination of programmable logic that defines the relationship among subject, object, environment attributes, and access decisions. In fact, policies can be expressed by procedural programming languages and deployed in appropriate computing environments. However, we leverage smart contracts in Substrate Framework to define policies.

IV. Implementation and Evaluation

A. Implementation

To investigate the proposed design, we implement and deploy a proof-of-concept (PoC) based on Substrate Blockchain Framework. Some abbreviations used to describe our algorithms are described in Table II. As presented in Section III-B, there are six main extrinsic for attributes management, attribute endorsement, and policy management in the proposed design. These six extrinsic belong to the ABAC Pallet (see Fig. 1) as interfaces to allow exteriors to manipulate their on-chain data. We implement these extrinsic as Algorithms 1, 2, and 3. Some remarks are summarized as follows:

- In Algorithm 1, if an attribute is modified or deleted, all its endorsements shall be dropped automatically, as in lines 11 and 23.
- In Algorithm 2, before creating an endorsement for an attribute, its existence should be verified as line 6, and its value is calculated as line 7. For example, if EV is 3600 and the block generation interval is six seconds, the corresponding endorsements will be valid for six hours.
- In Algorithm 3, the $checkPerm()$ is used to make sure those who utilize these functions are the object's owner or administrators delegated, and a reminiscent name is also possibly associated with each attachment for later use.

To express the access control policy by a smart contract for general purposes, several rules in development should be proposed to scope it for access validation. Firstly, it needs a list of trusted endorsers so that its administrator can decide who is trusted with them. Secondly, it must have at least one method for access validation. Thirdly, those methods should take necessary arguments (e.g., DID_s, DID_o, and EA), result in access decisions, and express policy logic in its body. Remarkably, there can be multiple policy logic expressions in one access validation method, and an access control template is proposed as Algorithm 4. A typical procedure should have five steps as follows.

- Step 1: Reading necessary attribute values.
- Step 2: Checking validity of the necessary attributes.
- Step 3: Evaluating defined policy logic.
- Step 4: Making a decision.
- Step 5: Saving the decision to the returning list

TABLE II. ABBREVIATIONS

Symbol	Description
DID	Distributed Identifier
DID_a	Administrator's DID
DID_e	Endorser's DID
DID_o	Object's DID
DID_s	Subject's DID
ATL	Attribute List
AS	Attribute Storage
AV	Attribute Value
ANL	Attribute Name List
ES	Endorsement Storage
EV	Endorsement Validity
PS	Policy Storage
$PAddr$	Policy Address
PRN	Policy's Reminiscent Name
EA	Environment Attribute
ADL	Access Decision List
ACC_s	The ith Blockchain Account
AEK	Authentication's Encrypted Key
APK_o	Authentication's Private Key of An Object
EAC	Encrypted Authentication Challenge
DDO	DID Document

In case of failing to read attribute values or invalid endorsement, the corresponding expression should be bypassed, and no decision should be considered denying access.

Moreover, to demonstrate the feasibility of the entire system based on the proposed design, we also develop an example script of the off-chain access control as Algorithm 5, which works as an Express.js server running on an access control object (e.g., the Gateway) and opens REST API for listening access control requests. This module can communicate with Blockchain via a Web Socket connection manipulated by the Polkadot.js API library to query on-chain storage or listen to Blockchain events. Some remarks are summarized as follows.

- Lines 1 and 2 are to establish a Web Socket connection to Blockchain and to initialize a DID Resolver, respectively. As a service, the infinite loop handles every access request when it comes. The corresponding handler will be triggered based on the incoming request type.
- Lines 8 to 13 are to perform an authentication process between the sender (i.e., a subject) and an object. When this process finishes, an authentication challenge will return to the subject.
- Lines 14 to 20 are to complete the DID Authentication process. The request must include an authentication response corresponding to the previous authentication challenge for the subject. If the response matches the challenge, an encrypted key will be returned to the subject as a successful authentication. Otherwise, an error will be returned as a failed authentication.

Algorithm 1 Attribute Management

Require: DID, $ATL[\,]$, $ANL[\,]$
1: *%Two extrinsics for users to control attributes associated with their DIDs*
2: **function** setAttributes($DID, ATL[\,]$)
3: **for** $attr$ **in** ATL **do**
4: $key \leftarrow < DID, attr.name >$
5: $value \leftarrow attr.value$
6: **if** AS.Exist(key) $== true$ **then**
7: AS.Insert(key, $value$)
8: **else**
9: AS.Mutate(key, $value$)
10: $keys \leftarrow < DID, attr.name, * >$
11: ES.RemoveAll($keys$)
12: **end if**
13: **end for**
14: **return** $Success$
15: **end function**
16:
17: **function** clearAttributes($DID, ANL[\,]$)
18: **for** $name$ **in** ANL **do**
19: $key \leftarrow < DID, name >$
20: **if** AS.Exist(key) $== true$ **then**
21: AS.Remove(key)
22: $keys \leftarrow < DID, name, * >$
23: ES.RemoveAll($keys$)
24: **end if**
25: **end for**
26: **return** $Success$
27: **end function**

Algorithm 2 Attribute Endorsement

Require: DID_e, DID_s, $ATL[\,]$, EV
1: *% Two extrinsics for trusted users to control endorsements of subject attributes*
2: **function** endorseAttributes(DID_e, DID_s, $ANL[\,]$, EV)
3: **for** $name$ **in** ANL **do**
4: $key_a \leftarrow < DID_s, name >$
5: $key_e \leftarrow < DID_s, DID_e, name >$
6: **if** AS.Exist(key_a) $== true$ **then**
7: $value \leftarrow EV + CurrentBlockNumber()$
8: ES.Insert($key_e, value$)
9: **end if**
10: **end for**
11: **return** $Success$
12: **end function**
13:
14: **function** unendorseAttributes(DID_e, DID_s, $ANL[\,]$)
15: **for** $name$ **in** ANL **do**
16: $key_e \leftarrow < DID_s, DID_e, name >$
17: **if** ES.Exist(key_e) $== true$ **then**
18: ES.Remove(key_e)
19: **end if**
20: **end for**
21: **return** $Success$
22: **end function**

- Lines 21 to 29 are to validate and then access a specific resource if authorized. The subject and policy validity

Algorithm 3 Policy Management

Require: DID_a, DID_o, $PAddr$, PRN
1: *% Two extrinsics for trusted users to manage access control policy*
2: **function** attachPolicy(DID_a, DID_o, $PAddr$, PRN)
3: **if** checkPerm(DID_a, DID_o) == false **then**
4: **return** $Fail$
5: **end if**
6: $key \leftarrow < DID_o, PAaddr >$
7: $value \leftarrow < DID_a, PRN >$
8: PS.Insert($key, value$)
9: **return** $Success$
10: **end function**
11:
12: **function** detachPolicy(DID_a, DID_o, $PAddr$)
13: **if** checkPerm(DID_a, DID_o) == false **then**
14: **return** $Fail$
15: **end if**
16: $key \leftarrow < DID_o, PAddr >$
17: PS.Remove(key)
18: **return** $Success$
19: **end function**
20:
21: *% A helper function to check permission for Policy Attachment*
22: **function** checkPerm(DID_a, DID_o)
23: $Admins \leftarrow DidPallet$.GetDelegates(
24: DID_o, "$PolicyAdmin$")
25: **if** $DID_a == DID_o$ **then** ▷ *self-management*
26: **return** $true$
27: **else if** $DID_a \in Admins$ **then**
28: **return** $true$
29: **end if**
30: **return** $false$
31: **end function**
32:

must be verified before making an on-chain access validation to get access decisions (ADL) returned from Blockchain. At line 27, $OnChainValidation()$ can collect environment attributes (EA) and encapsulate them into invocating on-chain policy execution.

The entire source code are freely shared at https://github.com/substrate-iot.

B. Deployment

We adopt containerization to simulate a Blockchain network as a runtime environment because it is flexible in establishing a more decentralized Blockchain. Currently, we have established a network consisting of 12 Blockchain nodes. Notably, Substrate Blockchain uses a Proof-of-Authority consensus algorithm that does not press on computing capability. Therefore, deploying multiple Blockchain nodes on low-cost desktops or personal computers is convenient for experiments while guaranteeing all Blockchain features.

The deployment process of our implementation on containers in Substrate Blockchain Framework is summarized in four main steps as follows.

Algorithm 4 The template to express the access control policy by smart contract

Input: DID_s, DID_o, $EA[\,]$
Output: $String[\,]$ (ADL: List of access decisions)
1: **function** validateAccess(DID_s, DID_o, $EA[\,]$)
2: Initialize ADL as an empty array of string
3:
4: *% Start of policy logic 1*
5: *. . . read attribute values . . .*
6: *. . . check validity of attributes . . .*
7: *. . . evaluate policy logic 1 . . .*
8: *. . . make a decision for evaluation 1 . . .*
9: ADL.Push(*the result for logic 1*)
10: *% End of policy logic 1*
11:
12: *% Start of policy logic 2*
13: . . .
14: . . . ADL.Push(*the result for logic 2*)
15: *% End of policy logic 2*
16: . . .
17: *% Start of policy logic n*
18: . . .
19: ADL.Push(*the result for logic n*)
20: *% End of policy logic n*
21: **return** ADL
22: **end function**

- Step 1: Create a docker image (DI) which includes the executable file (EF) of the Blockchain node by compiling the source code (SC).

$$SC \xrightarrow{compile} EF \xrightarrow{copy} DI$$

- Step 2: Generate Blockchain accounts (ACC) for every node in the networks.

$$EF \xrightarrow{generate} \{ACC_1, ACC_2, ..., ACC_{12}\}$$

- Step 3: Update the Default Chain Specification file (DCS), which would be loaded each the executable file boots up. A node must be aware of all other nodes and be updated with all other nodes' Blockchain addresses. Then, each particular node will have an Updated Chain Specification (UCS).

$$EF \xrightarrow{generate} DCS \xrightarrow{modify} UCS$$

- Step 4: Activate containers according to the number of Blockchain nodes from the docker image with the updated chain specification and their associated Blockchain account.

$$\{DI, UCS, ACC_i\} \xrightarrow{activate} BlockhainNode_i$$

C. Evaluation

We conduct experiments to investigate the timing problem, a common design metric in most IoT systems. It measures how long it takes to complete a transaction, especially when many users utilize the Blockchain simultaneously. If it is too long, it will negatively affect to real-time demand and scalability of the IoT system. Therefore, our experiments investigate the amount of time elapsed to thoroughly submit transactions

Algorithm 5 An off-chain access control JavaScript

Input: DID_o, APK_o

```
api ← PolkadotJS.ConnectTo(Blockchain)
DidResolver ← BuildDidResolver(api)
while true do
    < req, type, data > ← GetRequest()
    if req == null then
        continue
    end if
    if type == "AuthRequest" then
        DDO ← DidResolver(data.DID)
        publicKey ← GetAuthPublicKey(DDO)
        EAC ← GenerateEncryptedChallange(
                    publicKey)
        CacheChallange(data.DID, EAC)
        return Response(EAC)
    else if type == "AuthResponse" then
        authRe ← data.authReponse
        result ← VerifyAuthResponse(
                    data.DID, authRe)
        Ensure(result == true)
        encryptedKey ← GenerateEncryptedKey()
        CacheEncryptedKey(data.DID, encryptedKey)
        return Response(encryptedKey)
    else if type == "AccessResourceX" then
        encryptedKey ← data.encryptedKey
        PAddr ← data.policyAddress
        avRe ← VerifyAuthValidity(
                    data.DID, encryptedKey)
        pvRe ← VerifyPolicyValidity(
                    DID_o, PAddr)
        Ensure(avRe == true&&pvR == true)
        ADL ← OnChainValdation(
                    data.DID, PAddr)
        Ensure("allow_resource_X" ∈ ADL)
        return Response(current data of resouce X)
    end if
    return Response(Error)
end while
```

for seven main functions, including six extrinsic, as above-presented, and one for executing access policies. In each testing process, transactions have been submitted consecutively until a specific number of times is reached. The number of transactions for each process is increased by x10 times. Each testing process is performed five times to eliminate the effect of the natural unexpected random factor, and the final result is averaged. Fig. 3 summarises the experimental results, in which a logarithmic scale is utilized to present the value clearly due to the large differences.

Some findings are discussed based on the experimental results as follows:

- A single transaction's elapsed time, called response time, is roughly 1.0 to 1.5 on the logarithmic scale, equivalently from 10 to 30 milliseconds.
- When the number of transactions increases by x10 times, the elapsed times will also rise linearly by x10 times because it is equivalent to one unit on the logarithmic scale.
- At a specific number of consecutive transactions, the period of testing processes in different transaction types are almost the same, but one's validateAccess() is slightly greater than others. This is because invoking validateAccess() is a policy execution that leads to a smart contract execution behind the scenes, which is more complex than a normal extrinsic transaction.
- For application aspects, the average response time of 15 milliseconds is appropriate for the soft real-time IoT system. Besides, it can handle a vast number of transactions at a time, up to 10000, without any problems such as significant transaction failures or starvation. Therefore, the proposed design could meet the requirements of the IoT systems in terms of response time and accommodate more transactions compared to popular Blockchains like Bitcoin or Ethereum.

D. Security Analysis

The proposed design is trusted since it is inherently achieved by Blockchain compared to centralized systems. However, there is a trust-sharing model that is not mentioned before but exists implicitly in the proposed design. Device owners are considered the root of trust and can share it with other policy administrators through DID delegation. These administrators can manage ABAC policies of delegated devices or objects; in turn, they can specify who are trusted endorsers for ABAC policies. These trusted endorsers can endorse ABAC attributes of users or subjects to make them valid for access validation. The trust-sharing model makes access management more convenient but still maintains trust.

Additionally, the security of the proposed system is discussed on three typical security aspects of a software system: confidentiality, integrity, and availability (so-called CIA triad).

- **Confidentiality** is considered at IoT resources that should only be accessed by subject requests satisfying the requirements in the object's policies. For a malicious subject to access an IoT resource of an object, it has to modify its own ABAC attributes on blockchain to meet the object's policy. Nevertheless, such forged ABAC attributes have no meaning for access validation because they are not endorsed by one of the trusted endorsers specified in the policies. Furthermore, there are accident cases, for example, leakage of the blockchain account's private key. The previous trusted blockchain account and its DID become malicious. In such cases, endorsers can revoke their endorsements for that blockchain account. Similarly, device owners can also revoke delegations for policy administration in case administrators become malicious.
- **Integrity** is considered in on-chain data and policy execution. The data in the pallets' storages, such as ABAC attributes, attribute endorsement, ABAC policy attachment, etc., can only be modified by extrinsic signed by their owner's blockchain account. Regarding ABAC policies, once a smart contract is instantiated, the hash of both its source code and argument values passed into its constructor will be saved to the Pallet Contract's storage. This means that access validation

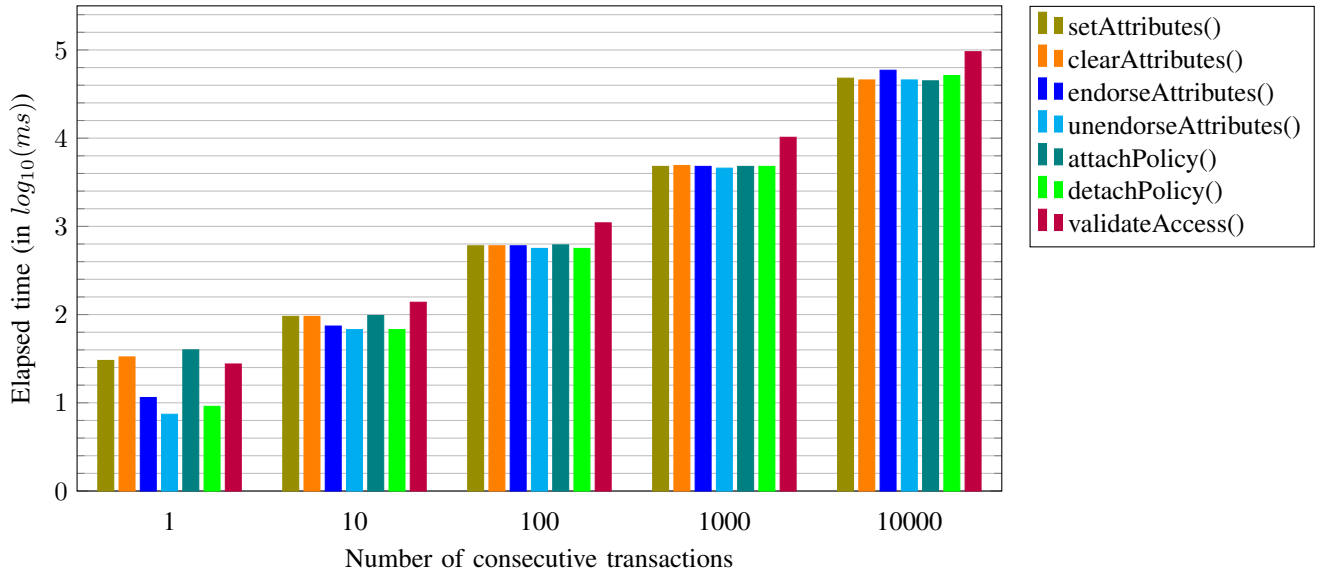

Fig. 3. Experimental results in terms of response time.

expressions of policy in the source code and trusted endorsers specified at the instantiating stage via the constructor are guaranteed to be consistent. Furthermore, policy execution can be distributedly conducted, and the result, which is access decisions, can be logged in the respective smart contract's storage. Generally, the integrity of policy execution is ensured by the smart contract. Moreover, in accident situations such as Blockchain account leakage, transactions originating from those leaked accounts are traceable because all transactions are kept in blocks.

- **Availability** is achieved by relying on a decentralized architecture with Blockchain technology. A decentralized system solves the single-point-of-failure problem in a centralized system; it can withstand a certain number of nodes failing simultaneously, and that number depends on the blockchain network topology and consensus algorithm. Each transaction has a certain fee in a blockchain system integrated with cryptocurrencies, so DoS or DDoS attacks by spamming transactions to disrupt the system are expensive for attackers.

As such, our proposed design achieves trust and security. However, these properties are also affected by the factors of a Blockchain system, such as network topology, consensus algorithm, and the number of honest nodes. Nonetheless, there is a horizontal scaling solution for the trust and security of a system by increasing the number of its Blockchain nodes. Meanwhile, consensus algorithm selection for a Blockchain is a trade-off between performance and security level, depending on assumptions of trust and security.

V. Conclusion

This paper presents details of a Blockchain-based ABAC model that includes two main parts: attributes management and policy execution. Each decentralized identifier (DID) can be associated with attributes to verify access permission, where attributes may need to be authenticated by trusted users (e.g., endorsers) via their DIDs. In our proposed model, the resource (e.g., IoT device or document) owner is considered the center, root-of-trust, who can delegate the authority to other administrators. Resource owners and administrators can also specify trusted endorsers for an access control policy for a particular object. Access control policies are codifiable by smart contracts, enhancing flexibility in access control management. Besides, policy validation is also executed on-chain, guaranteeing security in terms of the CIA triad. Therefore, our proposed design has demonstrated improved features compared to similar works based on four criteria, as shown in Table I. Furthermore, we also implemented the proposed model to investigate the performance regarding response time. The experimental results show that the proposed model could meet the soft real-time requirement for IoT systems.

In future work, intensive experiments be conducted to investigate other essential metrics, such as workload at the Blockchain node, storage cost, and transaction fee, before deploying it to practical applications.

Acknowledgment

The authors acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

References

[1] A. Riahi Sfar, E. Natalizio, Y. Challal, and Z. Chtourou, "A Roadmap for Security Challenges in the Internet of Things," *Digital Communications and Networks*, vol. 4, no. 2, pp. 118–137, 2018.

[2] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "IoT Privacy and Security: Challenges and Solutions," *Applied Sciences*, vol. 10, no. 12, 2020.

[3] I. Ben Dhaou, "A Secure IoT-enabled Sensor Node for Traffic Light Management and Level of Service Computation," in *Proceedings of the 18th International Multi-Conference on Systems, Signals and Devices (SSD)*, 2021, pp. 644–648.

[4] Rachit, , S. Bhatt, and P. Ragiri, "Security Trends in Internet of Things: a Survey," *SN Applied Sciences*, vol. 3, no. 121, 2021.

[5] H. A. Maw, H. Xiao, B. Christianson, and J. A. Malcolm, "A Survey of Access Control Models in Wireless Sensor Networks," *Journal of Sensor and Actuator Networks*, vol. 3, no. 2, pp. 150–180, 2014.

[6] F. Cai, N. Zhu, J. He, P. Mu, W. Li, and Y. Yu, "Survey of Access control Models and Technologies for Cloud Computing," *Cluster Computing*, vol. 22, pp. 6111–6122, 2019.

[7] J. Qiu, Z. Tian, C. Du, Q. Zuo, S. Su, and B. Fang, "A Survey on Access Control in the Age of Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.

[8] D.-H. Nguyen, D.-N. Nguyen-Duc, N. Huynh-Tuong, and H.-A. Pham, "CVSS: A Blockchainized Certificate Verifying Support System," in *Proceedings of the 9th International Symposium on Information and Communication Technology (SoICT)*. New York, NY, USA: Association for Computing Machinery, 2018.

[9] J. Wan, J. Li, M. Imran, D. Li, and F. e Amin, "A Blockchain-Based Solution for Enhancing Security and Privacy in Smart Factory," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3652–3660, 2019.

[10] W. A. Amiri, M. Baza, K. Banawan, M. Mahmoud, W. Alasmary, and K. Akkaya, "Towards Secure Smart Parking System Using Blockchain Technology," in *Proceedings of the 17th Annual Consumer Communications and Networking Conference (CCNC)*, 2020, pp. 1–2.

[11] K. Mohammad Hossein, M. E. Esmaeili, T. Dargahi, A. Khonsari, and M. Conti, "BCHealth: A Novel Blockchain-based Privacy-Preserving Architecture for IoT Healthcare Applications," *Computer Communications*, vol. 180, pp. 31–47, 2021.

[12] D.-H. *Nguyen, N. Huynh-Tuong, and H.-A. Pham, "A Blockchain-Based Framework for Developing Traceability Applications towards Sustainable Agriculture in Vietnam," *Security and Communication Networks*, vol. 2022, 2022.

[13] I. Riabi, H. K. B. Ayed, and L. A. Saidane, "A Survey on Blockchain based Access Control for Internet of Things," in *Proceedings of the 15th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2019, pp. 502–507.

[14] S. Rouhani and R. Deters, "Blockchain Based Access Control Systems: State of the Art and Challenges," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*. New York, NY, USA: Association for Computing Machinery, 2019, p. 423–428.

[15] T. Kumar, E. Harjula, M. Ejaz, A. Manzoor, P. Porambage, I. Ahmad, M. Liyanage, A. Braeken, and M. Ylianttila, "BlockEdge: Blockchain-Edge Framework for Industrial IoT Networks," *IEEE Access*, vol. 8, pp. 154 166–154 185, 2020.

[16] P. Patil, M. Sangeetha, and V. Bhaskar, "Blockchain for IoT Access Control, Security and Privacy: A Review," *Wireless Personal Communications*, vol. 117, pp. 1815–1834, 2021.

[17] H. A. Hussain, Z. Mansor, and Z. Shukur, "Comprehensive Survey and Research Directions on Blockchain IoT Access Control," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, 2021.

[18] B. Bhushan, P. Sinha, K. M. Sagayam, and A. J, "Untangling Blockchain technology: A survey on state of the art, security threats, privacy services, applications and future research directions," *Computers and Electrical Engineering*, vol. 90, p. 106897, 2021.

[19] S. Pal, A. Dorri, and R. Jurdak, "Blockchain for IoT access control: Recent trends and future research directions," *Journal of Network and Computer Applications*, vol. 203, p. 103371, 2022.

[20] S. Sun, S. Chen, R. Du, W. Li, and D. Qi, "Blockchain Based Fine-Grained and Scalable Access Control for IoT Security and Privacy," in *Proceedings of IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, 2019, pp. 598–603.

[21] J. Moffett, M. Sloman, and K. Twidle, "Specifying discretionary access control policy for distributed systems," *Computer Communications*, vol. 13, no. 9, pp. 571–580, 1990, network Management.

[22] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

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

[24] S. Gusmeroli, S. Piccione, and D. Rotondi, "IoT Access Control Issues: A Capability Based Approach," in *Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 787–792.

[25] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart Contract-Based Access Control for the Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2019.

[26] G. Ali, N. Ahmad, Y. Cao, S. Khan, H. Cruickshank, E. A. Qazi, and A. Ali, "xDBAuth: Blockchain Based Cross Domain Authentication and Authorization Framework for Internet of Things," *IEEE Access*, vol. 8, pp. 58 800–58 816, 2020.

[27] Y. E. Oktian and S.-G. Lee, "BorderChain: Blockchain-Based Access Control Framework for the Internet of Things Endpoint," *IEEE Access*, vol. 9, pp. 3592–3615, 2021.

[28] Y. Nakamura, Y. Zhang, M. Sasabe, and S. Kasahara, "Capability-based access control for the internet of things: An ethereum blockchain-based scheme," in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[29] H. Liu, D. Han, and D. Li, "Fabric-IoT: A Blockchain-Based Access Control System in IoT," *IEEE Access*, vol. 8, pp. 18 207–18 218, 2020.

[30] L. Song, Z. Zhu, M. Li, L. Ma, and X. Ju, "A Novel Access Control for Internet of Things Based on Blockchain Smart Contract," in *Proceedings of the 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 5, 2021, pp. 111–117.

[31] S. Sun, R. Du, S. Chen, and W. Li, "Blockchain-Based IoT Access Control System: Towards Security, Lightweight, and Cross-Domain," *IEEE Access*, vol. 9, pp. 36 868–36 878, 2021.

[32] D. Di Francesco Maesa, P. Mori, and L. Ricci, "A Blockchain based Approach for the Definition of Auditable Access Control Systems," *Computers and Security*, vol. 84, no. C, p. 93–119, 2019.

[33] E. Chen, Y. Zhu, Z. Zhou, S.-Y. Lee, W. E. Wong, and W. C.-C. Chu, "Policychain: A Decentralized Authorization Service With Script-Driven Policy on Blockchain for Internet of Things," *IEEE Internet of Things Journal*, vol. 9, no. 7, pp. 5391–5409, 2022.

[34] S. Dramé-Maigné, M. Laurent, and L. Castillo, "Distributed access control solution for the IoT based on multi-endorsed attributes and smart contracts," in *Proceedings of the 15th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2019, pp. 1582–1587.