The following publication D. Li et al., "Hyper-IIoT: A Smart Contract-Inspired Access Control Scheme for Resource-Constrained Industrial Internet of Things," in IEEE Transactions on Sustainable Computing, vol. 10, no. 5, pp. 820-829, Sept.-Oct. 2025 is available at https://doi.org/10.1109/TSUSC.2025.3542466.

Hyper-IIoT: A Smart Contract-inspired Access Control Scheme for Resource-constrained Industrial Internet of Things

Dun Li , Hongzhi Li , Noel Crespi , Roberto Minerva , Ming Li , Wei Liang , Kuan-Ching Li

***Abstract*—In recent years, the refinements in industrial processes and the increasing complexity of managing privacy-sensitive data in Industrial Internet of Things (IIoT) devices have highlighted the need for secure, robust, and adaptive data management solutions. In this work, we propose a smart contract-assisted access control scheme for IIoT, which employs the Attribute-Based Access Control (ABAC) model to set access permissions for different industrial components. Besides, we defined a storage model and data format for private data by designing and deploying smart contracts to manage system operations and access policies. In addition, the bloom filter component is deployed to optimize the efficiency of contract management and system performance. Experimental results show that in real-world simulations, Hyper-IIoT demonstrates excellent contract execution time control, stable system throughput, and rapid consensus processes that are capable of handling high throughput and efficient consensus in distributed systems, even in large-scale request scenarios.**

***Index Terms*—Blockchain, IIoT, Access Control, Smart Contracts**

I. Introduction

INDUSTRIAL Internet of Things (IIoT) integrates wireless sensor networks, communication protocols, and internet infrastructure with the monitoring, analyzing, and managing industrial operations [1], [2]. By building distributed networks through wired and wireless connections, IIoT devices engage with external environments and generate diverse data [3]–[5]. The evolution from centralized to decentralized systems improves scalability, personalization, and product quality [6]. In Industry 4.0, IIoT integrates smart devices with sensors, which support mobile computing and continuous communication across industrial sectors [7], [8]. By autonomous monitoring, collecting, analyzing, and optimizing industrial production environments, IIoT systems significantly enhance operational efficiency [9], [10]. Platforms such as Siemens MindSphere and GE Predix optimize data collection, storage, and analysis in smart manufacturing, effectively eliminating data silos and fostering integrated device ecosystems [11], [12].

However, existing IIoT frameworks normally rely on trusted third parties or assume mutual trust across different domains, which is challenging to fully implement in practical industrial environments [13], [14]. Maintaining data confidentiality in a wide range of industrial manufacturing systems is critical, as any data breach presents a significant threat to the system security [15]–[17]. In addition, the diverse array of device types in current IIoT systems further complicates the mutual utilization of generated data [18]. Therefore, centralized storage systems are fundamentally inadequate for effective data management, failing to guarantee data authenticity, integrity, and validity [19]–[21]. To overcome these limitations, decentralized management and collaborative data maintenance become essential to ensure safer and more efficient data processing in IIoT environments.

Furthermore, with the expansion of industrial operations and the increasing interconnection of devices, traditional IIoT architectures encounter growing pressure to support large-scale networks, with the increasing complexity of data types and communication protocols [22]. Current research on IIoT trust mechanisms focuses on addressing the practical needs of application environments, especially considering the limited and diverse computing and storage capabilities of IIoT devices [23], [24]. Therefore, more customized and practical solutions are required when implementing these mechanisms to overcome the associated challenges. This paper aims to answer the following research questions: i) How can high computational complexity in managing data access control and ensuring data security be effectively mitigated in resource-constrained IIoT environments? ii) What customized and practical solutions can be developed to implement trust mechanisms in IIoT applications, considering the limited and diverse computing and storage capabilities of IIoT devices? iii) How can emerging technologies like AI and machine learning be integrated into IIoT architectures to enhance robustness without exacerbating computational complexities?

To address these challenges, smart contracts enabled by

Dun Li is with the Department of Industrial Engineering, Tsinghua University, 100084, China, and Samovar, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France. E-mail:lidunshmtu@outlook.com

Hongzhi Li is with the College of Big Data and Artificial Intelligence, Chizhou University, Chizhou 247100, China. Email: pickpickup@sohu.com

Noel Crespi is with Samovar, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France. Email: noel.crespi@mines-telecom.fr

Roberto Minerva is with Samovar, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France. Email: roberto.minerva@telecom-sudparis.eu

Ming Li is with the Research Institute for Advanced Manufacturing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong, 999077, China. Email: ming.li@polyu.edu.hk

Wei Liang is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Sanya Research Institute, Hunan University of Science and Technology, and the Hunan Key Laboratory for Service Computing and Novel Software Technology, China. E-mail: wliang@hnust.edu.cn

Kuan-Ching Li is with the Dept. of Computer Science and Information Engineering, Providence University, 43301, Taiwan. E-mail: kuancli@pu.edu.tw

Corresponding Authors: Kuan-Ching Li, Noel Crespi, Hongzhi Li

blockchain technology have provided a robust solution, significantly improving security, trust, and data integrity within the IIoT framework [25]. Smart contracts integrate with the distributed architecture of IIoT, managing data transactions based on predefined rules while providing instant, automated responses to security threats [26]. By embedding operational rules into the blockchain, smart contracts make the interactions of the IIoT network transparent, autonomous, and immutable [27], [28]. Moreover, smart contracts can adapt to the diverse technical requirements of IIoT devices, enabling precise access control and secure data sharing, and adaptability is especially crucial in complex IIoT environments with unsynchronized device operations, as it reduces reliance on central systems and mitigates the risk of data breaches [29], [30].

A. Related work

Smart contracts provide a crucial security mechanism and reliable operating environment for IIoT data access control, with attribute-based access control (ABAC) models commonly integrated to implement precise control in asynchronous multi-stakeholder environments, protecting IIoT resources and sensitive information [31]–[33]. For instance, Yang *et al.* [34] developed a secure data-sharing protocol using blockchain technology, enhancing single critical exposure protection, enabling efficient keyword searches, and supporting dynamic user and key management. Then, Liu *et al.* [35] developed an IoT access control system based on Hyperledger Fabric but lacks comprehensive auditing capabilities for access records. To address the limitations of existing models, especially the latency issues in smart contract operations, researchers are exploring new enhancements to blockchain-based IIoT system access control modules. Wu *et al.* [36] presented MapChain-D, a framework that integrates distributed hash tables with Ethereum storage, explicitly catering to environments with significant latency and bandwidth constraints and thus serving resource-limited IIoT devices effectively. Similarly, the combination of multiple blockchain distributed keys and fog computing, as proposed in [37], shows the potential to reduce various link operation times and improve user privacy and security.

Although smart contracts have made progress in IIoT, challenges still exist due to the high computational complexity of device interactions. Existing research has not fully addressed the burden, hindering scalable access control and data security. Therefore, it is critical to reduce computational overhead while ensuring strong access control.

B. Our contributions

Based on the above-mentioned analysis, this work introduces Hyper-IIoT, a data access control framework for IIoT utilizing the Hyperledger Fabric platform. Hyper-IIoT employs an ABAC model to provide a flexible, dynamic, and fine-grained approach to data access control. The integration of smart contracts with Bloom filters in formulating access control policies reduces computational overhead and promotes adaptive governance over private data access protocols, effectively addressing the complexities of securing private data in the IIoT domain. Compared to existing approaches, the proposed method reduces the computational overhead associated with access control and data querying in IIoT environments. Deploying Bloom filters enables fast query response times, while smart contracts ensure consistent and transparent enforcement of access control policies. Specifically, the contributions of this work include the following:

- This paper introduces a secure private data storage structure for IIoT that addresses the limitations of centralized storage systems by enabling decentralized management and collaborative data maintenance.
- Innovative smart contracts are developed to build complex data architectures, enforce access controls, and orchestrate systemic processes, thereby accelerating data querying and access while improving system functionality and performance.
- An auxiliary architecture integrating Bloom filters with ABAC is designed, significantly reducing system overhead and enhancing data management efficiency in IIoT.
- Simulations and experiments demonstrate the effectiveness and operational efficiency of the proposed Hyper-IIoT framework within the IIoT ecosystem, highlighting the advantages of the Kafka consensus mechanism in achieving rapid consensus and high throughput compared to traditional Proof of Work (PoW) systems.

The remainder of this paper is organized as follows. Section II introduces the problem definition. Section III presents the architectural framework of Hyper-IIoT. An in-depth presentation of the performance evaluation is provided in Section IV. Finally, conclusions and future directions are given in Section V.

II. Problem definition

This section introduces the problem definition, including the interplay among models outlined in Subsection II-A, the data structure described in Subsection II-B, and interconnectivity in Subsection II-C.

A. Definition of the structural relationship between users and private data resource stores

The storage protocol proposed in this work reduces the load on the blockchain network by hosting public data on cloud servers. This arrangement allows users to access the data independently through the server address. In contrast, private data containing sensitive information is securely maintained within the blockchain network. Data owners should implement access control policies for private data and manage private data by auditing access records. Fig 1 presents the interaction among users, the IIoT platform, cloud, database, admin, and blockchain for managing private data, showing the specific data flow steps from data ownership, storage, and access requests to blockchain verification.

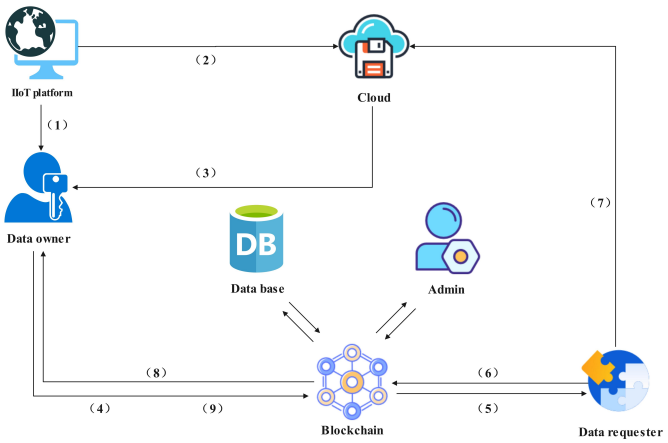


Fig. 1: Structural relationship of Hyper-IIoT

B. Data structure definition

In this subsection, we present a reliable mechanism for protecting private data, including the data format, access control policies, and documentation methods.

1) PersistentData: This model defines private data in a uniform format, as shown in Eq 1:

$$PD = \{persistentId, persistentData, persistentUrl\} \quad (1)$$

where PD represents the persistent data set of private data in the IIoT application scenario, $persistentId$ represents the unique primary critical identification of persistent data resources, and $persistentData$ represents the persistent format of private data. $persistentURL$ Indicates the URL where public data can be accessed on the cloud server.

2) AccessPolicy: The access control strategy of the Hyper-IIoT model is the core to achieving fine-grained access control. The data structure is mainly based on the ABAC model, and its structure is shown in Eq 2 - 5:

$$PL = \{AUser, APD, AAuth, AEnu\} \quad (2)$$

$$AUser = \{userId, role, PKUser\} \quad (3)$$

$$APD = \{dataId, signer, signData, dataKey, dataUrl\} \quad (4)$$

$$AEnu = \{createTime, endTime, IP, signPKuser\} \quad (5)$$

where PL signifies the access policy related to the attributes such as user, data, environment, and permission. $AAuth$ represents the permission attributes, delineating access privileges for private and public data endpoints within the IIoT domain. It comprises attributes such as $authUrl$ and $authData$, which correspond to the access control attributes for public data URLs and private data, focusing on validating the access permissions for private data. On the other hand, $AEnu$ signifies the set of environmental properties, delineating the contexts under which a data requester may access private environments. Environment properties encompass attributes like $createTime$ (policy creation time), $endTime$ (policy expiration time), ip (requesting node's IP address), and $signPKuser$ (data owner's signature on the policy to prevent tampering or forgery).

3) Record: To ensure the traceability of the Hyper-IIoT model, the system defines the access record as four data structures: $RECORD$, $REQUEST$, $RESPONSE$, and HIS, as shown in Eq 6 - 9.

$$RECORD = \{REQUEST, RESPONSE, HISTORY\} \quad (6)$$

$$REQUEST = \{AUser, APD\} \quad (7)$$

$$RESPONSE = \{policyID, owner, requestID, status, endTime, timestamp\} \quad (8)$$

$$HIS = \{resourceId, requestor, version\} \quad (9)$$

$RECORD$ encompasses privacy data access records for auditing, comprising $REQUEST$ (data request record), $RESPONSE$ (data owner's response record), and $HISTORY$ (data set access history). $REQUEST$ contains attributes about the requester and data set, while $RESPONSE$ includes details like $policyId$ (index for accessing blockchain's control policy), $owner$ (data owner), $requestID$ (index for request record), and $timestamp$ (data owner's response time). $HISTORY$ serves as a ledger for data set interactions, autonomously documenting pertinent details of requesters within the blockchain network. This registration occurs in conjunction with the access instance, precisely when a data collection, identified by $resourceId$, is successfully acquired by a requester. By efficiently structuring access records, Hyper-IIoT ensures minimal computational overhead during audit processes.

C. Model interrelation

The Hyper-IIoT model encapsulates a triad of interactive processes: Policy-to-Data, Data-to-Log, and Log-to-Policy, as shown in Fig 2.

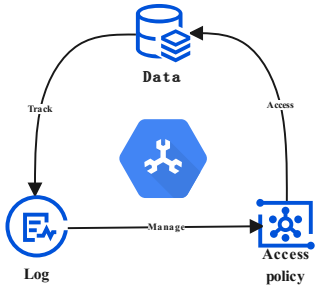


Fig. 2: Model relationship of Hyper-IIoT

In this model, the access control model ensures auditability by recording data requester IDs, creating request and response records, and validating legitimacy based on access, request,

and response records stored in the blockchain network. Direct correspondence between access logs and confidential data is established, simplifying the auditing process. The model uses ABAC to achieve fine-grained and context-aware access management. The basic concepts of the ABAC model are as follows: Subject (S) represents the entity requesting access, Object (O) is the resource being accessed, Policy (P) defines the rules that govern the access, and Environment (E) encompasses contextual information influencing the access decision. This structured approach enhances the security and efficiency of the access control mechanism within the IIoT framework.

III. System Model and Design

The smart contract and Bloom filter-based access control system proposed in this paper offers a reliable private data protection scheme. This section outlines the architectural design and details of Hyper-IIoT.

A. System architecture

Hyper-IIoT is deployed on the consortium chain, and the main design components include Users, Certification Authority (CA), Administrator (Admin), and Blockchain Network (BN), which provides user access modules that form the underlying structure. The system, integrating a smart contract-based access control model enhanced with a Bloom filter, is illustrated in Fig 3.

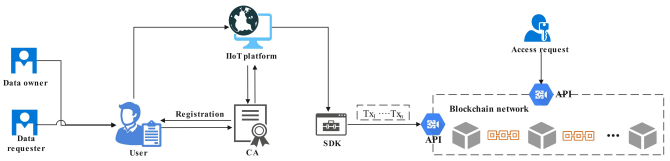


Fig. 3: Structural Design of Hyper-IIoT

1) IIoT: The IIoT device management platform processes various datasets produced by IIoT devices, including video, audio, and text content. The data generated during this operation is classified into private and public categories based on the system's predefined isolation criteria, which are transmitted to the cloud server for storage.

2) User: The system encompasses data owners who upload, manage access, and respond to data requests and data requesters requiring permission for access. Data owners can audit their data resources via access control policies, enhancing data security and traceability.

3) CA: This component issues certificates to uniquely identify new nodes, requiring each user to possess a CA certificate for various procedures like connecting to and authenticating with the blockchain network.

4) Admin: Responsible for managing the essential capabilities, including network startup and smart contract installation rights, while overseeing access control through CA authentication, access policy formulation, user data access rights assignment, and maintaining policies, enabling Hyper-IIoT to authorize users for data-related activities.

5) BN: The critical platform of the system, which consists of data storage and identity permission authentication, manages the storage of private data, access control, user authentication, and auditing, requiring authorization from the CA for users and Admin for data access, whereas data circulation and retention on the blockchain ledger are governed by smart contracts, culminating in a comprehensive data exchange ecosystem.

B. Smart contract design

Smart contracts are key components for implementing the logic and access control of Fab-IIoT. To ensure system efficiency while maximizing data privacy and integrity, this paper designs the following four types of smart contracts from the perspectives of user management, data management, and access control.

1) User management contracts (UMC): This contract is responsible for the management of user names and serial numbers, including the $CreateUser()$, $QueryUser()$, and $CheckUser()$. Fig 4 shows a simple example of a user management contract creating a name and serial number.

```
{
"userName": "user1",
"userNumber": "1a462ae9c1350c24ca8e389090946c7ed517667231c7b6ae22d0fed39ca55dd1",
}
```

Fig. 4: User data example

$CreateUser()$ method: The initial step for user participation involves user registration, where $CreateUser()$ generates a unique user identification, $UserNumber$, based on the $UserName$ field.

$QueryUser()$ method: Facilitates user information retrieval using $UserName$ and $UserNumber$, including a check for existing users during registration.

$CheckUser()$ method: Before seeking authorization via the access policy, users must undergo authentication, which verifies the user's signature $SignID$, public key Pub_k, and private key Pri_k.

2) Access control contracts (ACC): This function checks if the request to access the data is aligned with the preset access control policy. It includes key functions $Auth()$, $GetAttrs()$, $CheckPolicy()$, and $CheckAccess()$.

$Auth()$ method: Uses Admin-distributed public keys for request encryption, facilitating user request authentication and identity verification.

$GetAttrs()$ method: This module processes the attributes in received requests, focusing on subject and object characteristics, denoted as $\{S, O\}$. In addition, it configures environmental attributes E, forming an integrated set of attributes $\{S.O.E\}$.

$CheckAccess()$ method: This method validates access requests by checking if they align with established access policies, involving retrieval of attributes using $GetAttrs()$, querying the corresponding policy with $QueryPolicy()$, and verifying attribute satisfaction, with an invalid request result if no supporting policy is found or if attributes E and A fail to satisfy the policy as Alg 1 shows. The deployment interface of ACC is shown in Fig 5.

```
root@jtli-ubuntu:/home/gopath/src/github.com/Hyper-IIoT/client/nodejs# node ./invoke.js acc CheckAccess 'ABACR{Su, Ou, Eu}'
Wallet path: /home/gopath/src/github.com/Hyper-IIoT/client/nodejs/wallet
acc CheckAccess ABACR{Su, Ou, Eu}
Transaction has been submit, result is: OK
```

Fig. 5: The interface of $CheckAccess()$ method installation

Algorithm 1 The $CheckAccess()$ method checks the access request

Require: $ABAC_{Request}$
Ensure: $TrueorError$
1: $\{Su, Ou, Eu\} \leftarrow GetAtrrs(ABAC_{Request})$;
2: //Retrieve attribute info from the access control request based on attributes
3: $PolicySet = \{Policy_1, \ldots, Policy_n\} \leftarrow QueryPolicy(Su, Ou)$;
4: //Get attribute information based on attribute-based access control request
5: **if** PolicySet is Empty **then**
6: **return** $Denial()$
7: **end if**
8: **for** each $Policy$ in $PolicySet$ **do**
9: //Check each policy for compliance with visitor criteria
10: $\{Ap, \ldots, Pp, Ep\} \leftarrow Policy$
11: **if** $Value(Pp) == 1$ and $Eu \cap Ep$ is Not Empty **then**
12: **return** $Approval$
13: **end if**
14: **end for**
15: **return** $Denial()$

3) Private data control contract (PCC): This contract is used to manage both private and public data. It enables the data owner to amalgamate private and public data into singular resources, facilitating uploads and downloads. The contract encompasses methods as follows:

$AddData()$ method: Enables the data owner to combine private and public data into a single data resource. Once combined, the consolidated data can subsequently be transmitted to the permissioned blockchain network.

$GetData()$ method: This function permits data owners to access the data resources via a controlled policy mechanism, which coordinates with the blockchain's indexing system for retrieval.

4) Access policy management contracts (PMC): This paper defines the following four ways of operating $ABACPolicy$.

$AddPolicy()$ method: It involves the CA executing the $CheckPolicy()$ module to ensure policy validity before addition. Only policies that pass this check are recorded in the SDB and blockchain, as detailed in Alg 2.

$DeletePolicy()$ method: To remove expired policies, the administrator can explicitly invoke this function, or it can be executed automatically when the $CheckAccess()$ module runs.

$UpdatePolicy()$ method: Invoked by the administrator to modify $ABACPolicy$, and it can be done by either deleting the policy and writing the modified record to SDB and the blockchain or by adding the modified policy after updating.

$QueryPolicy()$ method: Allows administrators to retrieve $ABACPolicy$ details by attributes $ABACPolicy\_S$ or $ABACPolicy\_O$, as all policies are stored in CouchDB.

Algorithm 2 The AddPolicy() method customizes access policies

Require: $ABACP$
Ensure: $TrueorError$
1: @implement SmartContract Interface;
2: //Implementation of smart contract interface
3: $APIstubChaincodeStub \leftarrow Invoke()$;
4: **if** $err! = Null$ **then**
5: **return** $Error(BadPolicy)$;
6: **end if**
7: //Generate a unique index of the access policy based on the host and guest attributes
8: //Store the strategy in the blockchain
9: **if** $err! = Null$ **then**
10: **return** OK
11: **end if**
12: **return** OK;

C. Policy query optimization based on bloom filter

In IIoT systems, efficient data management is critical due to limited resources. Traditional filtering techniques, like hash tables, skip lists, and tree structures, face challenges in memory usage and query speed.

Hash Tables: Although hash tables offer fast lookups, they require significant memory and can experience performance issues due to collisions. In contrast, Bloom filters are highly memory-efficient and provide fast membership checks with minimal overhead.

Skip Lists: Require complex maintenance and higher memory usage. Bloom filters offer faster queries and use less memory without maintenance overhead.

Tree-Based Structures: These structures have higher latency due to disk or memory I/O. Bloom filters offer constant-time query responses with no need for reorganization, making them ideal for real-time IIoT applications.

The bloom filter, denoted as BF, is a spatially efficient, probabilistic data structure tailored for approximate set membership verification, as explored by Waikhom *et al.* [38]. BF offers memory efficiency, making them ideal for resource-constrained IIoT systems, while ensuring fast querying for low-latency data processing, with an acceptable trade-off in accuracy through a low false positive rate for many IIoT applications. Comprised of an array of t bits, $BF = (b_1, \ldots, b_t)$, all initialized to zero. For a set of x elements $A = \{\alpha_1, \alpha_2, \ldots, \alpha_x\}$ and p unique hash functions $\{hash_1, hash_2, \ldots, hash_p\}$ that map uniformly to the range $[1, n]$, the insertion process of an element $\alpha_i \in A$ involves calculating p hash values $\{hash_1(\alpha_i), \ldots, hash_p(\alpha_i)\}$ and setting the corresponding indices in BF to one ($BF[hash_j(\alpha_i)] = 1$ for $1 \leq j \leq p$, $1 \leq i \leq x$). The BF can produce two types of indications: certainty of the data's presence or a probability of its presence,

with the latter known as a false positive, which erroneously indicates the presence of data not inserted. A response of $BF[hash_j(\alpha_i)] = 0$ conclusively signifies that the element α_i does not exist in BF.

BF is utilized to enhance query efficiency within smart contracts, specifically to test the membership of variables within queries initiated by an Admin, thereby reducing the computational demands and the consequent system response delays. The indices generated by the Bloom filter, denoted as $Index$, are used to quickly check the presence of attributes in the policy sets. Assuming that the attribute sets of $ABACPolicy\_S$ and $ABACPolicy\_O$ with x elements are $\{k_0^s : value_0^s, \ldots, k_i^s : value_i^s, \ldots, k_{x-1}^s : value_{x-1}^s\} \in ABACPolicy\_S$ and $\{k_0^o : value_0^o, \ldots, k_i^O : value_i^O, \ldots, k_{x-1}^O : value_{x-1}^O\} \in ABACPolicy\_O$ respectively, the query procedure can be defined as follows:

Step 1: Build a bloom filter structure BF_A consisting of a t-bit array (all default values are 0) and p independent hash functions.

Step 2: Enter the defined set of smart contract attributes $\{ABACPolicy\_S, ABACPolicy\_O\}$, update the array $BF_A = (b_1, \ldots, b_n)$ with a value of 1, and its corresponding mapping process is $BF[hash(ABACPolicy\_S, ABACPolicy\_O)] = Index^{\{ABACPolicy\_S, ABACPolicy\_O\}}$.

Step 3: Input the attribute set $\{ABACPolicy\_S^{'}, ABACPolicy\_O^{'}\}$ of the query contract, $Hash_A$ it, and output the result set $Index^{S^{'},O^{'}}$.

Step 4: The result set $Index^{\{ABACPolicy\_S^{'}, ABACPolicy\_O^{'}\}} = \{I_0^{'}, I_1^{'}, \ldots, I_p^{'}\}$ is iterated successively to determine the $b_j = BF(I_j)$ value of the element $I_j^{'} \in Index^{\{ABACPolicy\_S^{'}, ABACPolicy\_O^{'}\}}$ in the BF_A index position obtained previously. If $b_j = 0$, it is determined that the attribute set $\{ABACPolicy\_S^{'}, ABACPolicy\_O^{'}\}$ of the contract in this query does not exist in the contract attribute set $\{ABACPolicy\_S, ABACPolicy\_O\}$ defined by the system, and the query process ends. If the value of $Index^{\{ABACPolicy\_S^{'}, ABACPolicy\_O^{'}}$ in the corresponding index position of BF_A is 1, then $\{k_j^{s^{'}} : value_j^s, k_j^{O^{'}} : value_j^{O^{'}}\} \in \{ABACPolicy\_S, ABACPolicy\_0\}$ is indicated.

Step 5: Loop through Step 3 and Step 4. If all elements$\{ABACPolicy_S^{'}, ABACPolicy_O^{'}\}$ and $\{k_0^{s^{'}} : value_0^{s^{'}}, \ldots, k_i^{s^{'}} : value_i^{s^{'}}, \ldots, k_{x-1}^{s^{'}} : value_{x-1}^{s^{'}}\} \in ABACPolicy\_S^{'}$ in $\{k_0^o : value_0^o, \ldots, k_i^O : value_i^O, \ldots, k_{x-1}^O : value_{x-1}^O\} \in ABACPolicy\_O^{'}$ exist in $\{ABACPolicy\_S, ABACPolicy\_O\}$, the query policy matches, and the query ends.

The above steps reduce the computational overhead of queries by quickly excluding non-matching elements. While the ABAC model supports fine-grained, context-aware access control, it can be computationally demanding. By integrating Bloom filters into access policy management, Hyper-IIoT lowers policy evaluation complexity and enhances access control efficiency in resource-constrained environments. Consequently, the computational complexity for querying the target contract is reduced from $O(t^2)$ to $O(p * t)$.

D. System workflow

As shown in Fig 6, the blockchain-based access control policy model mainly consists of five parts, and this part will explain the workflow of each part in detail.

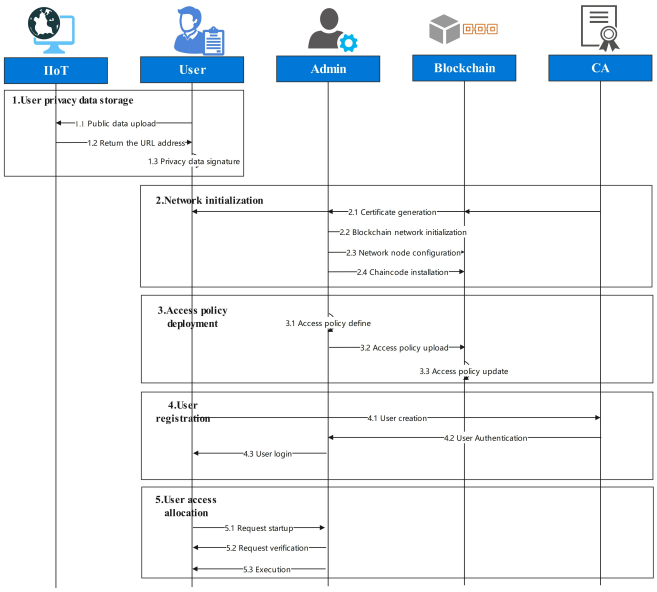


Fig. 6: Workflow for Hyper-IIoT solutions

1) Data storage: Data owners must first upload their private data resources and formulate relevant access control policies to complete the storage process of private data in the blockchain network. Suppose that there is a user $user1$ with $priData$, $pubData$, and public key $PKowner$, so the specific process of storing private data is as follows:

Step 1: The user transfers public data to the cloud server.

Step 2: The cloud server provides URL data in response.

Step 3: The user identifies and prepares private data for secure storage.

Step 4: The user must hash the private data and obtain the result sign_data.

2) Blockchain network initialization: After configuring the experimental environment, the system must be started first. The initial setup of the blockchain network mainly includes generating certificates for each part of the blockchain, creating and connecting nodes between the network, and installing chain codes and policies on each node. The specific implementation process is executed in the form of a script, and the steps are as follows:

Step 1: The certificate is generated. The certificates, public and private keys of User, Orderer, Peer, and Admin are uniformly generated by the CA, as shown in Eq 10:

$$\begin{cases} User \rightarrow CA \rightarrow \{Ct_{User}, pubk_{User}, prik_{User}\} \\ Orderer \rightarrow CA \rightarrow \{Ct_{Orderer}, pubk_{Orderer}, prik_{Orderer}\} \\ Peer \rightarrow CA \rightarrow \{Ct_{Peer}, pubk_{Peer}, prik_{Peer}\} \\ Admin \rightarrow CA \rightarrow \{Ct_{Admin}, pubk_{Admin}, prik_{Admin}\} \end{cases} \tag{10}$$

Step 2: Creation and connection of blockchain network nodes. Each user in the network constitutes a node. In a stand-alone environment, creating each node is equivalent to creating a docker container, ensuring each node has the same system environment. In a multi-machine environment, each computer is comparable to a node and does not need to have the same environment, it only needs to have the same client to achieve communication between nodes. Employing the Fabric-provided configtxgen tool $ctgen$ is essential for constructing a blockchain network, with the process detailed in Eq 11.

$$Fabric \overset{ctgen}{\longrightarrow} Block_1 \overset{kafka}{\longrightarrow} \{Block_1, Block_2, \ldots, Block_n\} \quad (11)$$

Step 3: Configure network nodes. The configuration information for nodes and channels is used to set up the network, while transaction details are processed as shown in Eq 12.

$$\{Ct_{Orderer}, Cf_{Orderer}, Ct_{Peer}, Cf_{Peer}\} \overset{SDK}{\rightarrow} \{T_{x_1}, \ldots, T_{x_n}\} \quad (12)$$

Step 4: Chaincode installation. chaincode installation is performed by the administrator according to Eq 13. Installing a chaincode on a blockchain network node can facilitate the authentication and endorsement of transactions on that node.

$$Chaincode \rightarrow Admin \overset{SDK}{\rightarrow} Peer \quad (13)$$

3) Access policy deployment: The system customizes access policies to ensure users' access rights to relevant data. The deployment procedure is as follows.

Step 1: Access policy definition. Admin provides management functions for attribute-based access control policies developed based on business rules, as shown in Eq 14:

$$\{S, O, P, E\} \rightarrow Admin \rightarrow \{ABACPolicy\} \quad (14)$$

Step 2: Access policy upload. Once the access policy is defined, it will be broadcast to the blockchain network in the steps indicated by Eq. 15.

$$\{ABACPolicy\} \rightarrow Admin \rightarrow \{Block_1, Block_2, \ldots, Block_n\} \quad (15)$$

Step 3: Access policy updates. The blockchain network does this by executing chaincode, updating the Ledger state, and saving the policy to the database, as shown in Eq 16:

$$\{ABACPolicy\} \rightarrow \{Ledger, LevelDB\} \quad (16)$$

4) User registration: Registering users involves recording the identity details of the data owners and requesters on the blockchain. The steps for registration are outlined as follows.

Step 1: User creation. This step first needs to check whether $UserName$ and $UserNumber$ already exist in the CA. If the member is new, the CA proceeds with registering the $User$, leading to the creation of the fields $\{SignID, Pubk, Prik, Ct\}$, as shown in Eq 17:

$$User \rightarrow CA \rightarrow \{SignID, Pubk, Prik, Ct\} \quad (17)$$

$UserName$ and $UserNumber$ are then stored in the database, as shown in Eq 18:

$$\{UserName, UserNumber\} \rightarrow \{Ledger, LevelDB\} \quad (18)$$

Step 2: User authentication. After registration, if the user wants access to create, query, and update the corresponding data, they need to obtain authentication by calling the $checkUser()$ method and then invoke the pre-set access policy to obtain authorization as shown in Eq 19.

$$User \rightarrow CA \overset{checkUser()}{\longrightarrow} Admin \rightarrow Policy_{User} \quad (19)$$

Step 3: User login. Upon user authentication, the server initially verifies the existence of the member. Once confirmed, the member's public and private keys, Pub_k and Pri_k, facilitate access to the Fabric blockchain network. Subsequently, the member's data retrieval is executed through the $queryUser()$ function. A successful data query triggers the server to dispatch the retrieved information and deliver the login outcome to the member.

5) User access allocation: User access authorization is the core of access control. Authorization must ensure sharing and dynamics while ensuring the integrity and synchronization of financial project data. User access authorization requires a strict definition of the policy based on actual service logic. The assignment steps are as follows.

Step 1: Request activation. The user initiates an access request to the blockchain, verified upon arrival at Admin as shown in Eq 20.

$$User \overset{Request}{\longrightarrow} Admin \quad (20)$$

Step 2: Request validation. After receiving an access request from the user, Admin calls the $GetAtrrs()$ method to get the user properties and then the $CheckAccess()$ method for validation as shown in Eq 21.

$$\begin{aligned} &Admin \overset{GetAttrs()}{\longrightarrow} User\{S, O\} \longrightarrow User\{S, O, E\} \\ &Admin \overset{Checkaccess()}{\longrightarrow} User\{S, O, E\} \longrightarrow User\{S, O, E, P\} \end{aligned} \quad (21)$$

Step 3: Execution. After the administrator verifies the user's access policy, the system processes the user's request per the established access policy. If the return result is 1, the access is passed. If the return result is 0, the access is denied, as shown in Eq 22.

$$Admin \overset{ABACPolicy}{\rightarrow} User\{S, O, E, P\} \begin{cases} 1, & Pass \\ 0, & No \end{cases} \quad (22)$$

IV. Performance evaluation and analysis

This section describes the experimental process for evaluating functionality and presents the performance and comparative results of the proposed Hyper-IIoT. Section IV-A details the experimental setup and the parameters configured for the evaluation, while Section IV-B delves into the analysis of the experimental data and the consequent performance implications.

A. System environment and configuration

The experimental evaluation in this study is executed in a stand-alone environment and configured as detailed subsequently.

1) Network structure: The proposed network topology contains nine types of nodes. This setup comprises four database nodes ($Fabric - couchdb$), two certificate authority nodes ($Fabric - ca$), four peer nodes ($Fabric - peer$), one order node ($fabric-orderer$), and one client node ($fabric-tools$). Additionally, there are four nodes each for user-managed contracts (UMC and AMC), private data control contracts (PCC), and access control policies (PMC), as shown in Tab I. We emulate a distributed multi-machine environment utilizing three Raspberry Pi microcomputers, depicted in Fig 7.

TABLE I: Symbols used.

Node	Implication	Number
UMC	The user manages the contract nodes	4
AMC	The user manages the contract nodes	4
PCC	Private data control contract nodes	4
PMC	Access control policy nodes	4

(a) (b)

Fig. 7: Raspberry Pi microcontroller devices

2) Chaincode deployment: Within the Hyper-IIoT framework, chaincode defines properties, manages access control, and orchestrates system operations. This encompasses the processes of installation, instantiation, and subsequent upgrades.

Step 1: Installation. Chaincode installation is initiated after completing blockchain network initialization, performed through Hyperledger client nodes, and installed into each peer node.

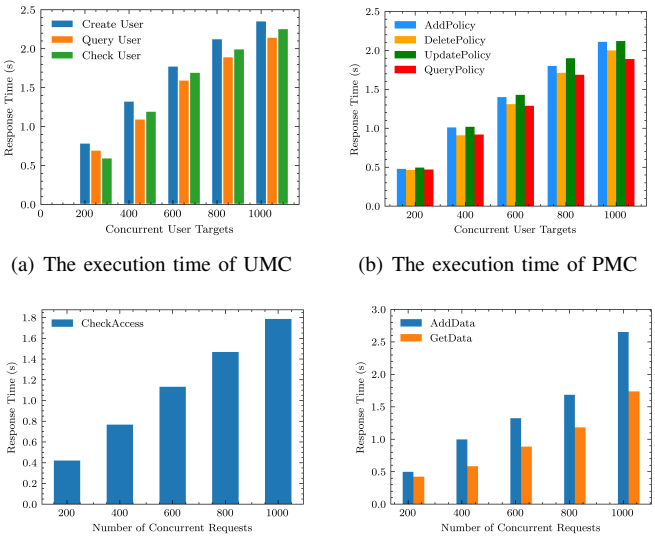
Step 2: Instantiation. The instantiated chain code is specified after it has been installed on any peer node

Step 3: Upgrade. Before updating the chain code, it is imperative to install the new version; the update will take effect exclusively on peer nodes that have the revised chain code in place.

B. Performance test and experimental results

To evaluate Hyper-IIoT's performance, this study conducts three simulation experiments to assess system concurrency, transaction time, and throughput under real-world conditions, demonstrating its ability to maintain high throughput and effective consensus in a distributed system even under large-scale request scenarios. To verify the practical applicability of Hyper-IIoT, we used multi-machine experiments to conduct simulations and case studies reflecting real industrial scenarios.

1) Contract execution time: Rooted in the operational logic, this research initially assesses the execution time for user management, policy management, and access control contracts, as depicted in Fig 8. The findings reveal a consistent and gradual rise in the contract execution time corresponding to increased block size, suggesting a well-regulated and systematic growth pattern. It is evident that the $add()$ and $update()$ methods require more time to execute compared to the $query()$ and $delete()$ methods, primarily due to the more significant number of data writes they involve. However, this difference in execution time is sufficiently minor to be effectively managed.



(a) The execution time of UMC (b) The execution time of PMC

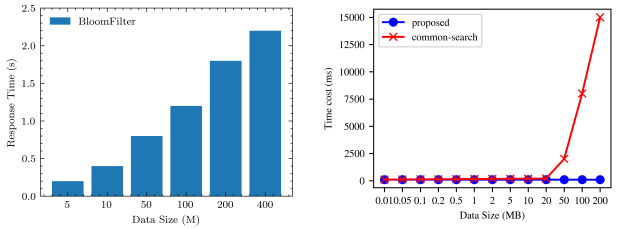
(c) The execution time of ACC (d) The execution time of PCC

Fig. 8: The execution time of smart contracts

Furthermore, Hyper-IIoT integrates a bloom filter-enhanced API into the smart contract in Hyperledger Fabric. Fig 9(a) presents the efficiency of bloom filters in processing queries across varying file sizes. The data indicate a systematic increase in processing time correlating with heightened concurrency, underscoring the bloom filter's compatibility with scalable data storage and query scenarios. For query precision, the bloom filter's byte size and hash function count are finely tuned by the anticipated data dimensions. Subsequently, the MD5 hash is stored as a key-value pair within the state database, prompting an update to the ledger. A comparative analysis between the bloom filter-enabled and traditional search methodologies underpin the augmented efficacy of the bloom filter approach as data volume escalates, as illustrated in Fig 9(b).

The integration of Bloom filters substantially reduces the computational complexity of access control and data querying operations. Traditional search methods typically exhibit linear or even quadratic time complexity with respect to the number of policies or data attributes, which can lead to significant performance degradation as the system scales. In contrast, Bloom filters enable constant-time complexity for membership checks, thereby enhancing query performance and alleviating the computational burden on the blockchain network. The reduction in computational overhead is critical for maintaining

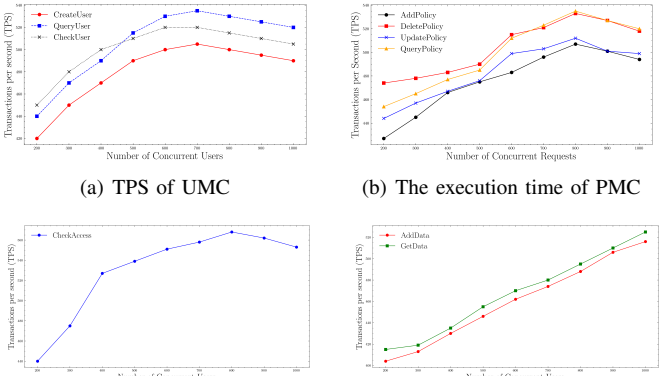
low latency and high throughput in resource-constrained IIoT environments.



(a) Query time consumption of bloom filters (b) Comparing the query time between the bloom filter-assisted scheme and the regular scheme

Fig. 9: The execution time of bloom-filter assisted scheme

2) TPS of smart contracts: This section evaluates the Transaction Per Second (TPS) of Hyper-IIoT in simulated scenarios with different concurrency levels. The system's throughput under the Kafka consensus protocol shows unparalleled reliability, as seen in Fig 10. Notably, the administrative contract, which handles addition and modification functions, exhibits a marginally reduced throughput compared to the contract responsible for querying and validation due to the inherent read and write demands. Nonetheless, the Hyper-IIoT system sustains a robust TPS and stable performance even amidst stringent operational demands.

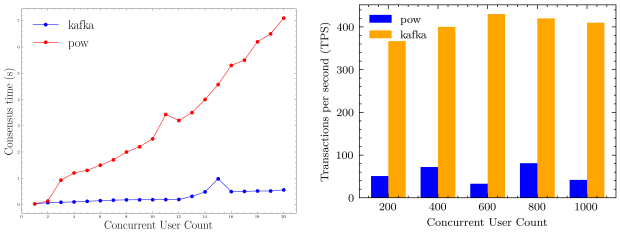


(a) TPS of UMC (b) The execution time of PMC

(c) The execution time of ACC (d) The execution time of PCC

Fig. 10: Smart contract throughput

3) Consensus Time Contrast: We compared the duration and throughput of the consensus mechanisms of Kafka and Proof of Work (PoW) across networks ranging from 10 to 100 nodes. Fig 11(a) shows the time consumption required to reach consensus using the proposed Hyper-IIoT system with Kafka compared to the traditional PoW mechanism. Kafka's ability to achieve consensus significantly faster than PoW, especially as the number of nodes grows, highlights its suitability for high-throughput, low-latency IIoT applications by dramatically reducing consensus time. In contrast, Fig 11(b) demonstrates Kafka's notable advantage over PoW in terms of throughput. Due to the computationally intensive nature, PoW's TPS decreases sharply as the number of nodes increases. Kafka maintains a stable and high TPS rate, even as network size scales up to 100 nodes, and meets the high-throughput requirements for efficient data access control and management in real-world IIoT applications.



(a) The Consensus time comparison between Kafka and POW (b) Consensus time comparison

Fig. 11: The TPS comparison between Kafka and POW

V. Conclusions and Future Work

This paper proposes an IIoT data management framework based on smart contracts and bloom filters to achieve secure data storage and access control. The proposed Hyper-IIoT enhances accurate data governance and access control by maintaining secure storage, enabling efficient data access, and ensuring traceability of sensitive data, effectively addressing evolving data formats and increasing privacy demands. The access policy in the system is programmable and offers adaptability to diverse access requisites. Through the implementation of ABAC, Hyper-IIoT has achieved fine-grained access control, and the bloom filter deployment improves system performance. Experimental results show that Hyper-IIoT exhibits well-controlled contract execution times, stable throughput, and efficient consensus in a distributed environment and highlight the advantages of the Kafka consensus mechanism in achieving rapid consensus and high throughput, meeting the requirements in real-world applications.

In future directions, this work can be explored in the following aspects.

1) Future work will involve investigating and implementing a variety of access control policies, such as Role-Based Access Control (RBAC) and Context-Aware Access Control (CAAC), alongside advanced security mechanisms like anomaly detection and intrusion prevention systems.
2) To integrate Hyper-IIoT with cutting-edge technologies such as 5G connectivity and edge computing frameworks for information security management.
3) Future research will focus on designing and implementing novel consensus algorithms specifically for IIoT environments, reducing latency and maximizing throughput, thereby improving the responsiveness and consensus efficiency of Hyper-IIoT in large-scale distributed networks.
4) To reduce the incidence of false positives inherent in Bloom filters, future work will explore the development of more accurate probabilistic data structures, such as Counting Bloom Filters or Cuckoo Filters.

REFERENCES

[1] J. Long, W. Liang, K.-C. Li, Y. Wei, and M. D. Marino, "A regularized cross-layer ladder network for intrusion detection in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1747–1755, 2022.

[2] S. Zhang, B. Hu, W. Liang, K.-C. Li, and A.-S. K. Pathan, "A trajectory privacy-preserving scheme based on transition matrix and caching for iiot," *IEEE Internet of Things Journal*, 2023.

[3] A. Mahmood, L. Beltramelli, S. F. Abedin, S. Zeb, N. I. Mowla, S. A. Hassan, E. Sisinni, and M. Gidlund, "Industrial iot in 5g-and-beyond networks: Vision, architecture, and design trends," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4122–4137, 2021.

[4] S. Zhang, B. Hu, W. Liang, K.-C. Li, and B. B. Gupta, "A caching-based dual k-anonymous location privacy-preserving scheme for edge computing," *IEEE Internet of Things Journal*, 2023.

[5] S. Zhang, Z. Yan, W. Liang, K.-C. Li, and C. Dobre, "Baka: Biometric authentication and key agreement scheme based on fuzzy extractor for wireless body area networks," *IEEE Internet of Things Journal*, 2023.

[6] D. Li, D. Han, T.-H. Weng, Z. Zheng, H. Li, H. Liu, A. Castiglione, and K.-C. Li, "Blockchain for federated learning toward secure distributed machine learning systems: a systemic survey," *Soft Computing*, vol. 26, no. 9, pp. 4423–4440, 2022.

[7] Z. Xu, W. Liang, K.-C. Li, J. Xu, A. Y. Zomaya, and J. Zhang, "A time-sensitive token-based anonymous authentication and dynamic group key agreement scheme for industry 5.0," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7118–7127, 2021.

[8] W. Liang, Y. Yang, C. Yang, Y. Hu, S. Xie, K.-C. Li, and J. Cao, "Pdpchain: A consortium blockchain-based privacy protection scheme for personal data," *IEEE Transactions on Reliability*, 2022.

[9] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International journal of production research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[10] Y. Li, W. Liang, K. Xie, D. Zhang, S. Xie, and K. Li, "Lightnestle: quick and accurate neural sequential tensor completion via meta learning," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[11] K. Smolander, P. Grünbacher, S. Hyrynsalmi, and S. Jansen, "Proceedings of the 2nd acm sigsoft international workshop on software-intensive business: Start-ups, platforms, and ecosystems," in *2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-ups, Platforms, and Ecosystems*. ACM, 2019.

[12] M. Salhaoui, A. Guerrero-González, M. Arioua, F. J. Ortiz, A. El Oualkadi, and C. L. Torregrosa, "Smart industrial iot monitoring and control system based on uav and cloud computing applied to a concrete plant," *Sensors*, vol. 19, no. 15, p. 3316, 2019.

[13] B. Chen, S. Qiao, J. Zhao, D. Liu, X. Shi, M. Lyu, H. Chen, H. Lu, and Y. Zhai, "A security awareness and protection system for 5g smart healthcare based on zero-trust architecture," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 248–10 263, 2020.

[14] D. Li, D. Han, Z. Zheng, T.-H. Weng, H. Li, H. Liu, A. Castiglione, and K.-C. Li, "Moocschain: A blockchain-based secure storage and sharing scheme for moocs learning," *Computer Standards & Interfaces*, vol. 81, p. 103597, 2022.

[15] M. Liu, F. R. Yu, Y. Teng, V. C. Leung, and M. Song, "Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3559–3570, 2019.

[16] N. Hu, D. Zhang, K. Xie, W. Liang, C. Diao, and K.-C. Li, "Multi-range bidirectional mask graph convolution based gru networks for traffic prediction," *Journal of Systems Architecture*, vol. 133, p. 102775, 2022.

[17] J. Li, D. Han, Z. Wu, J. Wang, K.-C. Li, and A. Castiglione, "A novel system for medical equipment supply chain traceability based on alliance chain and attribute and role access control," *Future Generation Computer Systems*, vol. 142, pp. 195–211, May 2023.

[18] J. Wan, J. Li, M. Imran, D. Li *et al.*, "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.

[19] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart cyber-physical systems," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 2, pp. 766–775, 2018.

[20] J. Cai, W. Liang, X. Li, K. Li, Z. Gui, and M. K. Khan, "Gtxchain: A secure iot smart blockchain architecture based on graph neural network," *IEEE Internet of Things Journal*, 2023.

[21] D. Li, N. Crespi, R. Minerva, W. Liang, K.-C. Li, and J. Kołodziej, "Dps-iiot: Non-interactive zero-knowledge poof-inspired access control towards information-centric industrial internet of things," *Computer Communications*, p. 108065, 2025.

[22] R. Verma, N. Dhanda, and V. Nagar, "Towards a secured iot communication: A blockchain implementation through apis," in *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security: IC4S 2021*. Springer, 2022, pp. 681–692.

[23] B. Pourghebleh, K. Wakil, and N. J. Navimipour, "A comprehensive study on the trust management techniques in the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9326–9337, 2019.

[24] W. Liang, Y. Li, K. Xie, D. Zhang, K.-C. Li, A. Souri, and K. Li, "Spatial-temporal aware inductive graph neural network for c-its data recovery," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[25] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[26] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: Attacks and protections," *IEEE Access*, vol. 8, pp. 24 416–24 427, 2020.

[27] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2018, pp. 1–4.

[28] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic review of security vulnerabilities in ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022.

[29] R. Saha, G. Kumar, M. Conti, T. Devgun, T.-h. Kim, M. Alazab, and R. Thomas, "Dhacs: Smart contract-based decentralized hybrid access control for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3452–3461, 2021.

[30] W. Wang, H. Huang, Z. Yin, T. R. Gadekallu, M. Alazab, and C. Su, "Smart contract token-based privacy-preserving access control system for industrial internet of things," *Digital Communications and Networks*, vol. 9, no. 2, pp. 337–346, 2023.

[31] S. Zhou, K. Li, L. Xiao, J. Cai, W. Liang, and A. Castiglione, "A systematic review of consensus mechanisms in blockchain," *Mathematics*, vol. 11, no. 10, p. 2248, 2023.

[32] H. Li and D. Han, "A novel time-aware hybrid recommendation scheme combining user feedback and collaborative filtering," *Mobile Information Systems*, vol. 2020, pp. 1–16, 2020.

[33] W. Liang, M. Tang, J. Long, X. Peng, J. Xu, and K.-C. Li, "A secure fabric blockchain-based data transmission technique for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3582–3592, 2019.

[34] N. Yang and C. Tang, "Blockchain-assisted secure data sharing protocol with a dynamic multi-user keyword search in iiot," *IEEE Internet of Things Journal*, 2023.

[35] 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.

[36] T. Wu, G. Jourjon, K. Thilakarathna, and P. L. Yeoh, "Mapchain-d: A distributed blockchain for iiot data storage and communications," *IEEE Transactions on Industrial Informatics*, 2023.

[37] M. Ma, G. Shi, and F. Li, "Privacy-oriented blockchain-based distributed key management architecture for hierarchical access control in the iot scenario," *IEEE access*, vol. 7, pp. 34 045–34 059, 2019.

[38] L. Waikhom, S. Nayak, and R. Patgiri, "A survey on bloom filter for multiple sets," in *Modeling, Simulation and Optimization: Proceedings of CoMSO 2020*. Springer, 2021, pp. 775–789.