Journal of Intelligent Computing and Networking
https://www.ffspub.com/index.php/jicn/index
ISSN 3079-9228 (print)
E-mail: jicn.office@ffspub.com

Article

A Low-Latency and Secure Data Sharing Method Based on Blockchain and Outsourced Attribute-Based Encryption

Wen Zhang[1], Li Duan[1], Yimeng Feng[2], Pengrui Chen[1], Lufeng Feng[1,†]
[1] *Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China*
[2] *School of Computing, Macquarie University, Sydney 2109, Australia*
[†] *E-mail: lufeng.feng@bjtu.edu.cn*

Received: October 29, 2025 / Revised: November 21, 2025 / Accepted: November 25, 2025 / Published online: November 27, 2025

Abstract: Current data sharing mechanisms face limitations in fine-grained access control, encryption overhead, terminal resource consumption, and result verifiability. These issues make them unsuitable for the low-latency and high-security demands of drone swarm collaboration in cloud-edge-end architectures. To address these challenges, this paper proposes a secure and low-latency data sharing method based on blockchain and outsourced attribute-based encryption. First, in the edge layer, a blockchain network is responsible for enforcing access control, where policy-matching smart contracts enforce fine-grained attribute-based access control. Second, encryption and decryption tasks are outsourced to the edge and cloud, effectively reducing the computational burden on terminal devices. Third, a consistency verification smart contract is introduced to validate outsourced results, ensuring data confidentiality and integrity. Experimental results show that the proposed method significantly lowers system latency and terminal overhead while maintaining strong security, making it suitable for edge-collaborative applications with strict real-time requirements.

Keywords: Blockchain; low-latency data sharing; outsourced attribute-based encryption; smart contract; edge computing security
https://doi.org/10.64509/jicn.12.44

1 Introduction

Drone swarms, as collaborative systems composed of multiple autonomous or semi-autonomous drones, are emerging as a core technology reshaping the execution of critical tasks. Their application domains have extended beyond traditional military reconnaissance to include public safety, precision agriculture, industrial inspection, and disaster response [1]. These scenarios are commonly characterized by task complexity, dynamic environments, and stringent real-time requirements. Rather than serving merely as isolated data collection platforms, drones within a swarm operate as integrated cooperative entities, exhibiting collective intelligence through tight coordination and real-time information exchange. The realization of swarm intelligence and collaborative behavior fundamentally relies on reliable and low-latency data exchange, both among individual drones and between the swarm and external command nodes [2]. Therefore, the efficiency of data sharing, especially with respect to latency, has become a critical concern in drone swarm missions.

Furthermore, drone systems are typically deployed in open and complex environments, which expose them to a diverse range of security threats. During transmission, data is highly vulnerable to eavesdropping, tampering, replay attacks, and unauthorized access. These threats can lead to the disclosure of sensitive mission-related information, such as reconnaissance objectives or navigation routes, and may compromise the integrity of mission execution. In more severe cases, adversaries may inject falsified data to gain unauthorized control over the swarm. Therefore, guaranteeing the confidentiality, integrity, and authenticity of shared data is a fundamental prerequisite for enabling drone swarms to operate reliably in mission-critical scenarios.

In response to the aforementioned threat landscape, the security architecture of drone systems must not only prevent external adversaries from eavesdropping on or manipulating communication channels, but also defend against

† Corresponding author: Lufeng Feng
* Academic Editor: Mengwei Xu
© 2025 The authors. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

potential threats posed by internal untrusted entities, such as compromised drones or malicious cloud servers, which may attempt to tamper with or fabricate mission-critical data. Under this adversarial model, conventional encryption mechanisms that focus solely on channel protection, such as Transport Layer Security (TLS) or Internet Protocol Security (IPSec) [3, 4], are insufficient to ensure end-to-end security across the entire data lifecycle. This limitation becomes particularly pronounced in environments characterized by resource-constrained nodes, unstable communication links, and stringent real-time mission requirements.

Recent research has explored the integration of authentication mechanisms, symmetric encryption, and identity management protocols to establish secure communication frameworks for drone networks [5]. However, these methods often struggle to balance lightweight design, low latency, and strong security guarantees when applied to scenarios characterized by highly dynamic network topologies, heterogeneous node collaboration, and resource-constrained environments. Such limitations hinder their practical feasibility and scalability in real-world deployments. Consequently, there is an urgent need for a lightweight security mechanism that operates effectively in untrusted network environments, supports fine-grained access control, and resists active attacks, in order to ensure both communication security and mission availability for drone swarms in critical applications.

Blockchain technology has attracted considerable attention for its decentralized architecture, tamper resistance, and traceability, particularly in ensuring identity trustworthiness and data integrity [6, 7]. However, existing studies primarily focus on scenarios involving static identity management or data provenance, while lacking systematic solutions for efficient and secure data sharing under conditions of resource constraints and dynamic network topologies. To address this gap, this paper proposes a low-latency and secure data sharing scheme based on blockchain and outsourced attribute-based encryption, aimed at enhancing both the efficiency and security of data exchange in edge environments. First, access control is delegated to the edge network, where fine-grained, attribute-based authorization is enforced through smart contracts. Second, an outsourced attribute-based encryption scheme is employed to offload encryption and decryption tasks to edge and cloud layers. The correctness of the outsourced results is then verified via a consistency validation contract, thereby enabling secure and efficient data sharing. The main contributions of this paper are as follows:

1) A low-latency and secure data sharing scheme tailored for drone-based cloud-edge-end collaborative scenarios is proposed.By integrating blockchain technology with outsourced attribute-based encryption, the scheme effectively addresses several limitations of traditional data sharing mechanisms, including coarse-grained access control, excessive computational burden on terminal devices, and insufficient security verification.
2) A policy-driven smart contract mechanism is designed to delegate access control authority to the edge-side blockchain network. This facilitates fine-grained, attribute-based access control, enhances the flexibility and scalability of access policy enforcement, and reduces dependence on centralized cloud-based management.
3) A consistency verification smart contract mechanism is developed to validate the correctness of outsourced encryption results executed at the edge or cloud layer. This design strengthens the trustworthiness and integrity of the data processing pipeline. Experimental evaluations confirm the effectiveness of the proposed method in reducing system latency and terminal resource consumption.

The remainder of this paper is organized as follows. Related work is reviewed in Section 2. In Section 3, we introduce the problem formulation. Specifically, it contains the system model, threat model, and design goals. Section 4 details the related methods and technologies. Section 5 provides the security analysis of the proposed method and theoretically demonstrates its effectiveness. The experimental analyses are provided in Section 6. Section 7 concludes this paper.

2 Related Work

In drone-enabled cloud–edge–end collaborative environments, the inherent characteristics of dynamic network topologies, limited device resources, and untrusted communication channels impose stringent demands on both the security and efficiency of data sharing. To address these challenges, existing research has primarily focused on ensuring secure data exchange through the implementation of access control mechanisms and the design of cryptographic algorithms.

2.1 Access Control Mechanisms in Drone Swarms

Implementing secure access control in drone swarms is critical to ensuring system reliability. The predominant models in current practice include Role-Based Access Control (RBAC) [8] and Attribute-Based Access Control (ABAC) [9]. RBAC assigns permissions based on predefined roles, offering simplicity in management but limited granularity. In contrast, ABAC makes dynamic access decisions based on entity attributes and policy rules, enabling fine-grained and flexible control at the cost of increased policy evaluation complexity. Jeong et al. [10] designed a multi-operator drone control system using the RBAC model, the administrator role is granted full system privileges, the pilot role is limited to flight control and status monitoring, and the camera operator role is restricted to controlling the gimbal and accessing video data. This role-based separation effectively prevents operational conflicts among multiple operators and enhances the efficiency and precision of task execution.

However, traditional RBAC models exhibit limited adaptability in dynamic environments, as they are unable to adjust permissions in response to real-time contextual conditions such as time, location, or device status. To address this limitation, Pang et al. [11] proposed a Location and Environment-Aware Attribute-Based Access Control (LE-ABAC) policy for securing drone flight control systems. This method combines internal drone attributes with external contextual information, such as geographic location and environmental conditions, to define fine-grained access rules, thereby enabling precise control over internal data interactions and mitigating risks such as malicious code injection and data

tampering in PX4-based flight controllers. Nonetheless, this work relies on a centralized Policy Decision Point (PDP) and attribute management service. If the central node is compromised, attackers may forge attributes or manipulate policies to bypass access control mechanisms. Therefore, although conventional RBAC and ABAC models improve the manageability and flexibility of access control in drone collaboration scenarios to a certain extent, their limited granularity and reliance on centralized infrastructure pose significant challenges. These limitations hinder their ability to fully support the distributed and autonomous coordination requirements of secure drone swarm operations.

2.2 Secure Data Sharing in Drone Cloud–Edge–End Collaboration

In the process of data sharing within drone-enabled cloud–edge–end collaborative systems, instructions and data are often transmitted over open communication channels, making them susceptible to malicious interception or forgery. During data exchange, edge collaboration nodes and cloud service nodes may also pose threats to the confidentiality and consistency of shared data. Attribute-Based Encryption (ABE) has emerged as a promising cryptographic method that enables fine-grained access control by specifying decryption policies based on user attributes [12, 13]. This level of granularity enhances the security of data sharing, particularly in the highly dynamic environments in which drones operate. However, conventional access control schemes typically impose significant computational overhead, rendering them unsuitable for resource-constrained terminal devices.

To mitigate this issue, encryption and decryption tasks can be partially outsourced to ground stations or edge computing nodes [14, 15]. For instance, Zhou et al. [16] proposed a blockchain-based secure and efficient data sharing model that leverages the immutability of blockchain and smart contracts to support identity authentication and access control, while outsourcing cryptographic operations to accelerate encryption and decryption. Nevertheless, the model does not address the verifiability the outsourced encryption process. To overcome these challenges, Ge et al. [14] proposed the verifiable and fair attribute-based proxy re-encryption (VF-ABPRE) scheme, a ciphertext-policy attribute-based proxy re-encryption mechanism that ensures verifiability and fairness. This work employs commitment techniques to allow recipients to verify the correctness of re-encryption, while preventing honest cloud servers from being falsely accused. Similarly, Sun et al. [15] introduced VF-PPBA, a verifiable and fair proxy broadcast re-encryption scheme supporting attribute-based data sharing among multiple recipients. By preserving structure and incorporating commitment mechanisms, the scheme enables users to verify the correctness of re-encrypted ciphertexts. However, both approaches suffer from high verification complexity and lack automated execution mechanisms, making it difficult to support efficient and scalable data validation workflows.

To address the challenges posed by the dynamic topology of drone networks, recent studies [17–19] have explored the use of blockchain technology and its embedded smart contracts to automate the enforcement of access control policies. These approaches enable decentralized access management and allow systems to adapt to rapid changes in network topology. By leveraging smart contracts, access rights can be granted or revoked automatically based on predefined conditions, thereby reducing dependence on centralized management entities and enabling real-time responsiveness to network state variations.In drone network scenarios, Feng et al. [19] developed a blockchain-based framework that leverages smart contracts and outsourced parallel computation to enable privacy-preserving data sharing without a trusted third party. Bera et al. [20] investigated the applicability and challenges of blockchain in 5G-enabled Internet of Things (IoT) environments and introduced a novel secure framework named BSD2C-IoD. This framework is designed to safeguard data exchanged among drone communication entities and to secure communication between drones and ground control. The study further proposes a consensus-based algorithm to facilitate lightweight verification and block addition. Mandal et al. [21] proposed CSUAC-IoT, a certificateless signcryption scheme with three-factor authentication for IoT systems. By integrating cryptographic techniques with biometric verification and mobile devices, the scheme ensures secure communication between IoT devices and users, and supports user authorization, authentication, and revocation. These studies collectively enhance the security and efficiency of complex network environments composed of drones, IoT devices, and edge computing nodes, demonstrating the pivotal role of blockchain in achieving these goals. Accordingly, the integration of smart contracts and attribute-based encryption enables fine-grained and secure data sharing without reliance on trusted third parties.

In summary, existing studies addressing data sharing security in drone-enabled cloud–edge–end collaborative scenarios face the following challenges: (1) The traditional access control mechanisms are prone to single points of failure or trust issues. (2) Current approaches often rely on computationally intensive encryption operations performed on terminal devices, resulting in low efficiency and high resource consumption. (3) There is a lack of verifiability mechanisms to ensure the correctness and trustworthiness of encryption results. To address these challenges, this paper proposes a blockchain-based smart contract for fine-grained access control, which ensures the immutability and transparency of access policies and enables precise data authorization and protection. Furthermore, an outsourced attribute-based encryption scheme is adopted to offload complex cryptographic operations to edge or cloud servers. A consistency verification mechanism, implemented via smart contracts, is introduced to validate outsourced results, thereby reducing the computational burden on drone terminals while ensuring the trustworthiness and integrity of shared data.

3 Problem Formulation

Under the cloud–edge–end collaborative architecture, traditional data sharing mechanisms struggle to meet the demands for efficient and secure data exchange due to centralized bottlenecks, poor adaptability to dynamic environments, and untrusted communication channels. These limitations are particularly pronounced in resource-constrained terminal scenarios

such as drones. To address these challenges, this paper proposes a low-latency and secure data sharing method based on blockchain and outsourced attribute-based encryption, which significantly enhances both the efficiency and security of data sharing in edge environments. This section presents the system model and threat model of the proposed method, and gives its corresponding security objectives.

3.1 System Model

The security assumptions of this paper consider the command center as a trusted authority within the system, equipped with comprehensive security protection mechanisms and access control policies. It is responsible for centralized scheduling, task issuance, and key management. Ground control stations (GCSs) are modeled as enhanced-trust, auditable edge nodes equipped with secure hardware execution and reliable communication links. They handle ciphertext transformation and policy evaluation without retaining long-term decryption keys, preventing them from becoming single points of failure even under partial compromise. In contrast, the cloud server is considered untrusted, as it may be subject to external attacks or internal unauthorized access, posing risks such as data leakage, tampering, or denial-of-service attacks. Similarly, drones are treated as untrusted terminals, given their deployment in open and complex environments where they are vulnerable to eavesdropping, tampering, replay attacks, and unauthorized access. As a result, their operations cannot be fully trusted. Therefore, throughout the processes of data collection, transmission, and processing, it is essential to design a lightweight security mechanism that supports end-side encryption and fine-grained access control while providing resilience against tampering by untrusted cloud servers. Ensuring the confidentiality, integrity, and availability of system-wide data under these conditions necessitates a rigorously designed and verifiable security framework.

The blockchain-based secure data sharing method for drone collaboration proposed in this paper is illustrated in the system model shown in Figure 1. The model involves five key entities: drones, ground control stations (GCS), blockchain network, cloud service (CS), and the command center. Drones are responsible for receiving commands, executing cooperative tasks, and collecting mission-related data, which are subsequently uploaded to the GCS. As edge computing nodes, the GCSs not only manage drone scheduling but also perform preliminary data processing and forwarding. CS provides auxiliary storage and computational capabilities. However, due to its uncontrolled operating environment, it is considered untrusted.

The blockchain functions as a distributed ledger system jointly maintained by multiple GCS nodes. Embedded smart contracts are employed to perform essential operations such as task validation and data access auditing. The command center, acting as a trusted authority, oversees identity registration, key distribution, and command dissemination for both drones and GCSs, thereby establishing a controllable root of trust at the system initialization phase.

Nevertheless, existing data sharing mechanisms exhibit notable limitations in terms of access control granularity, cryptographic computation overhead, terminal resource consumption, and result verifiability. These shortcomings make them inadequate for meeting the combined requirements of low latency and high security in drone swarm collaboration within cloud–edge–end architectures. To address these challenges, this paper proposes a low-latency and secure data sharing scheme that integrates blockchain with outsourced attribute-based encryption. By distributing access control enforcement and encryption verification tasks across edge and cloud layers, the proposed method significantly reduces the computational burden on terminal devices. The integration of this system model and methodology not only improves data sharing efficiency in edge-collaborative environments, but also reduces system latency and terminal energy consumption while preserving security, thereby offering strong potential for real-world deployment and broader application.

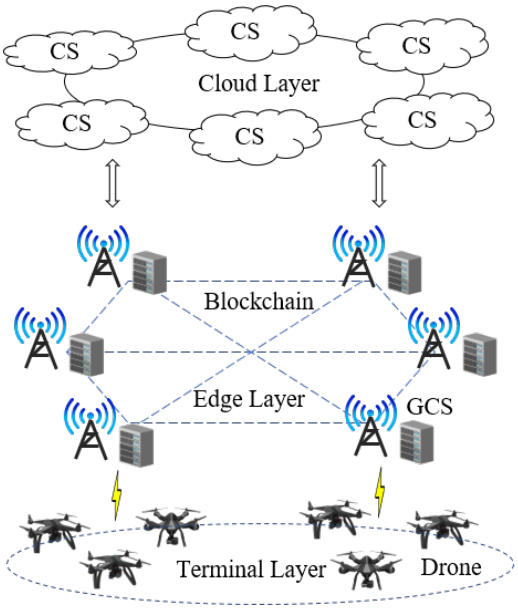

Figure 1: The model of data sharing scheme blockchain-based and attribute outsourcing encryption.

3.2 Threat Model

Due to the inherent characteristics of drone swarm networks and their communication channels, the security vulnerabilities and risks of privacy infringement may arise. The proposed scheme in this paper adopts the widely used Dolev–Yao (DY) threat model [22], which assumes that an adversary is capable of intercepting, modifying, and fabricating messages transmitted over insecure channels. Accordingly, the specific capabilities of the adversary in this model are defined as follows:

(1) During communication, a malicious drone may eavesdrop on transmitted messages and further manipulate the communication channel by modifying or injecting falsified messages.

(2) A malicious drone may collude with other drones to repeatedly request messages from the ground station, thereby disrupting the normal operation of the server through a denial-of-service attack.

(3) Drones and cloud servers may forge data or perform incorrect encryption and decryption operations. A drone may also

generate falsified decryption proofs in order to deny responsibility and shift blame to the cloud server for erroneous data storage.

3.3 Security Objectives

The primary objective of designing a data sharing scheme is to ensure data security and privacy while maintaining its availability. The data sharing scheme proposed in this paper is designed to achieve the following three security goals:

(1) Data Confidentiality

Data must be encrypted during both transmission and storage to prevent unauthorized access. In the data sharing process, open communication channels and insecure networks can lead to the leakage of sensitive information. To preserve data confidentiality, all data transmitted and stored must undergo encryption. Encryption ensures that even if data is intercepted during transmission or accessed illegally from storage devices, the content remains protected and cannot be easily deciphered.

(2) Data Integrity

Data integrity refers to the assurance that data has not been tampered with during storage or transmission. In general, hash functions can be employed to generate a unique hash value for the data. The sender computes the hash before transmission and sends it along with the data. Upon receipt, the receiver recomputes the hash and compares it with the original; if the two values match, the data is considered unaltered. Additionally, digital signatures offer a means of verifying both data integrity and authenticity. They not only detect unauthorized modifications but also confirm the origin of the data. However, since the proposed scheme involves outsourced attribute-based encryption, it requires not only verification of data integrity and consistency, but also validation of the correctness of outsourced encryption operations.

(3) Access Control

Strict access control mechanisms must be enforced throughout the data sharing process to ensure that only users with appropriate permissions are allowed to access or modify the data. Access control is a fundamental component of any data sharing scheme, serving to enforce permission boundaries and protect sensitive information. An effective access control strategy typically consists of two key steps: authentication and authorization. Authentication verifies the identity of the user, and once verified, authorization determines which specific resources the user is permitted to access.

4 Method

To enable secure end-to-end data sharing, this paper proposes a low-latency and secure data sharing scheme based on blockchain and outsourced attribute-based encryption. The proposed scheme first leverages blockchain technology to delegate access control permissions and policy matching to the edge layer, where dynamic and automated access control is enforced via smart contracts. Second, by employing an outsourced attribute-based encryption algorithm, computationally intensive encryption tasks are offloaded to edge and cloud layers, thereby significantly reducing the computational burden on terminal devices. Finally, the consistency and security of the entire data sharing process are verified through smart contracts, ensuring both the efficiency and reliability of data exchange.

This section provides a detailed description of the proposed data sharing method. The overall system workflow is illustrated in Figure 2. The symbols and their corresponding meanings used in this paper are summarized in Table 1. The protocol is designed as follows:

$Setup(\lambda, U) \rightarrow (pp, msk)$: The system setup algorithm executed by the command center takes as input the security parameter λ and the attribute universe U, and outputs the public parameters, including the public parameters for message-locked encryption pp, and the master secret key msk.

$AES.Enc(pp, m, K) \rightarrow C_m$: This process is a symmetric encryption procedure executed on the drone and the GCS. It takes as input the system public key pp, the message m, the symmetric key K, and outputs the ciphertext C_m.The symmetric key $K = H_1(R_{D1} \oplus R_{G1})$ is generated based on Physical Unclonable Function (PUF) technology [23]. Note that PUFs are not mandatory for the correctness of the protocol. They serve as an optional hardware-based enhancement for binding symmetric keys to device-specific physical characteristics, providing resistance against key extraction attacks. If a deployment does not support PUF hardware, conventional key-agreement mechanisms can be adopted without affecting the remaining components of the scheme.

$AES.Dec(pp, C_m, K) \rightarrow m$: This process is a symmetric decryption procedure executed on the drone and the ground control station (GCS). It takes as input the system public key pp, the ciphertext C_m, and the symmetric key K, and outputs the message m.

$MLE.Enc(pp, m, \sigma) \rightarrow \{R, r, C_{MLE}, C_{tag}\}$: This process is the Message-Locked Encryption (MLE) encryption procedure. It takes as input the system public key pp, the message m, the MLE encryption key σ, outputs the message authentication code R, the hash r of the combined message and code, and the locked ciphertext C_{MLE}, C_{tag}. The ciphertext tag C_{tag} ensures non-duplicate data storage on the cloud server without requiring decryption.

$KeyGen(msk, S) \rightarrow SK$: This process is also executed by the command center. It takes as input the master secret key msk and the attribute set S, and outputs the private key $SK = (z, TK)$.

$ABE.Enc(pp, (\mathbb{A}, \rho), C_{MLE}, \sigma, r) \rightarrow CT$: This process is an encryption procedure. It takes as input the system public key pp and the access control policy $(\mathbb{A}, \rho)$, treats the MLE encryption key σ as part of the secret in the linear secret sharing scheme, and combines it with the MLE ciphertext and encryption parameters. The final output is the complete ciphertext CT.

$ABE.UserEnc(pp, (\mathbb{A}, \rho), C_{MLE}, \sigma, r) \rightarrow C_{user}$: This algorithm is executed by the GCS. It performs partial encryption computation and takes as input the system public key pp, the access control policy $(\mathbb{A}, \rho)$, the MLE ciphertext, and the MLE encryption parameters. It outputs the partial ciphertext C_{user}.

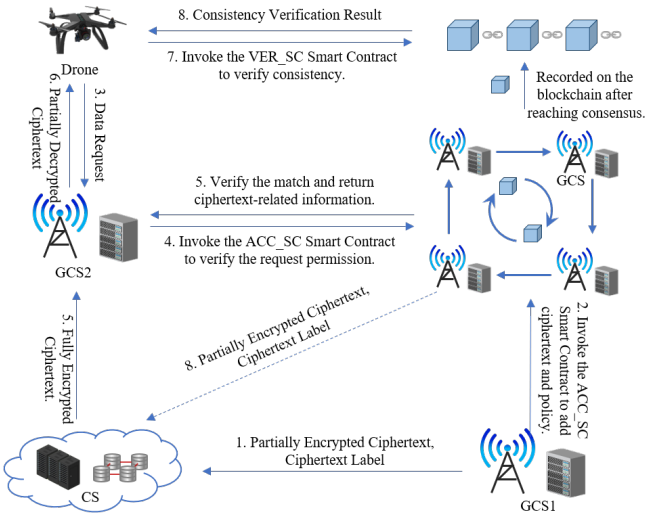

Figure 2: The flowchart of data sharing scheme blockchain-based and attribute outsourcing encryption, showing how drone, ground stations, the cloud, and the blockchain interact across data encryption, outsourced computation, access control, and final decryption. The figure highlights how computation is distributed among entities and how smart contracts enforce authorization and verify correctness.

Table 1: Notation table

Notation	Description
λ	Security parameters
U, S	Universal attribute set, Attribute subset
pp, msk	Public parameters, Master secret key
(e, G, G_T, G, p)	Bilinear mapping pair
$f_1, \ldots, f_U$	Attribute function
$\alpha, a, \phi, \varphi, Q$	Random number
H_1, H_2	Hash function
DID, GID	Unique real identity of drone and ground station
K	Symmetric key
$TS, \Delta T, TC$	Timestamp, Time interval, Current system time
m_1, m	Original message, Processed message
C_m	Ciphertext
σ, R, r	MLE encryption parameters
C_MLE	Ciphertext for locked encryption output
C_tag	Ciphertext tag
$SK, \{z, TK\}$	Private key, Private key components
$\mathbb{A}$	Access control policy
$(\mathbb{A}, \rho)$	Data access control structure
CT	Attribute-encrypted ciphertext
C_{user}	User-side encrypted ciphertext
C_{out}	Cloud-side encrypted ciphertext
D_{out}	Partially decrypted ciphertext
C'	Fully decrypted ciphertext
π	Message decryption proof
ACC_SC	Policy matching contract
VER_SC	Consistency verification contract

$ABE.OutEnc(pp, (\mathbb{A}, \rho)) \rightarrow C_{out}$: This algorithm is executed on the cloud server. It takes as input the system public key pp, the message m, and the access control policy $(\mathbb{A}, \rho)$, and outputs a partially encrypted ciphertext C_{out}.

$ABE.Dec(pp, (\mathbb{A}, \rho), SK, CT) \rightarrow C'$ consists of the following two partial decryption algorithms:

$ABE.OutDec(pp, (\mathbb{A}, \rho), TK, CT) \rightarrow D_{out}$: This algorithm is executed on the GCS and performs partial decryption. It takes as input the private key $TK = (K, L, \{K_x\}_{x \in S})$ and the ciphertext CT, as defined in Equation 1, and outputs a partially decrypted ciphertext D_{out}.

$$CT = ((\mathbb{A}, \rho), C_{MLE}, C_{tag}, C_1, \{C_{2,j}, C_{3,j}, C_{4,j}\}_{j \in [1,l]}) \quad (1)$$

$ABE.UserDec(pp, z, D_{out}) \rightarrow C'$: This is the decryption algorithm executed on the drone terminal, which runs after partial decryption. It takes as input the system public key pp, the private key z, and the partially decrypted ciphertext D_{out} from the cloud server, and outputs the fully decrypted ciphertext C'.

$MLE.Dec(pp, R, C_{MLE}, C') \rightarrow \{m', \pi\}$: This process is the MLE decryption procedure. It takes as input the system public key pp, the locked ciphertext C, the message authentication code R, and the fully decrypted ciphertext C'_{MLE}, and outputs the recovered plaintext m' from message-locked decryption and the corresponding decryption proof π.

Finally, two smart contracts are deployed on the blockchain. The first is the policy matching contract, denoted as ACC_SC, which verifies whether $V(S, (\mathbb{A}, \rho)) = 1$ holds; if so, access permission is granted. The second is the consistency verification contract, denoted as VER_{SC}, which checks whether the condition $C_{tag} = \phi \oplus \varphi \oplus H_2(m') \oplus H_2(R)$ is satisfied; if the condition holds, consistency verification is considered successful.

4.1 System Initialization Phase

The $\text{Setup}(\lambda, U) \rightarrow (pp, msk)$ algorithm, executed by the command center, takes as input the security parameter λ and the universal attribute set U, and returns the public parameters pp, including the public parameters for message-locked encryption and the master secret key msk. The command center generates a bilinear map tuple (e, G, G_T, G, p), and randomly selects values $(\alpha, a \in Z_p^*, g, f_1, \ldots, f_U, \phi, \varphi, Q \in G)$, along with two hash functions H_1 and H_2. The master secret key is defined as $msk = g^\alpha$, and the public parameters pp are constructed accordingly.

Each drone and ground control station is equipped with a PUF chip and assigned a unique identity, denoted as DID and GID, respectively. After pre-registration, the registration algorithm executed by the command center takes as input the master secret key msk and the user' s attribute subset S, and outputs the private key $SK = \{z, TK\}$.

$$pp = (e, G, G_T, g, f_1, \ldots, f_U, \phi, \varphi, g^a, e(g,g)^\alpha) \quad (2)$$

The drone Dr_1 first performs mutual authentication and key agreement with the GCS_1, resulting in the establishment of a symmetric key K and a secure communication channel. Afterward, Dr_1 encrypts the collected data using symmetric encryption and transmits it to GCS_1.

(1) According to the blockchain and PUF-based mutual authentication protocol proposed in [23], the validity of the

counterparty's identity can be verified. If the authentication is successful, both parties generate a session key K_1.
(2) The drone Dr_1 encrypts the current system time TS through $AES.Enc(K_1, DID\|TC)$ to request authentication from the GCS. The ground station first checks whether the current time TC satisfies the condition $TC - TS < \Delta T$. If the condition holds, the session key is considered valid and has not expired; otherwise, re-authentication is required.
(3) The drone uses the symmetric key K_1 to encrypt the collected data m_1 through symmetric encryption, resulting in the ciphertext C_{m_1}. The process is described in Equation 3.

$$C_{m_1} = AES.Enc(pp, (DID1||m_1), K_1) \tag{3}$$

4.2 Data Encryption and Upload Phase

In this phase, after decrypting the data, the GCS_1 processes and organizes the data, defines the data access control structure, and performs partial encryption. The partially encrypted data is then outsourced to the cloud server for final encryption and storage. The sequence diagram is shown in Figure 3.

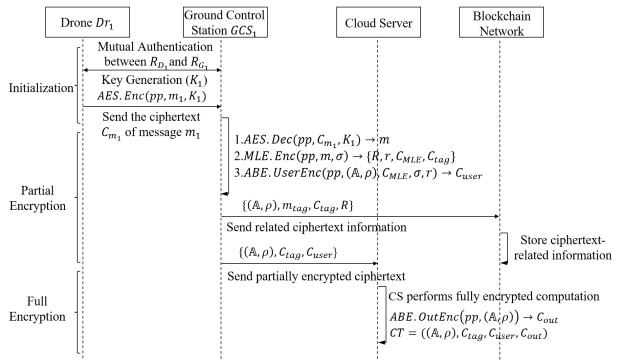

Figure 3: The sequence diagram of the encryption request phase of the data sharing scheme, where the drone transmits encrypted data to the ground station, and the ground station performs MLE locking and partial ABE encryption. The cloud completes the outsourced encryption, and the blockchain stores metadata necessary for subsequent access-control verification.

(1) GCS_1 decrypts the data using the key K_1 through $\mathrm{AES.Dec}(pp, C_{m1}, K_1)$. If the resulting data has the prefix DID_1, the original data m_1 is successfully recovered. The data is then processed and reorganized to form a new data item m, to which a new tag m_{tag} is assigned.
(2) First, the data m is encrypted using the message-locked encryption function $\mathrm{MLE.Enc}(pp, \sigma, m)$, which takes as input the system public key pp, the data m, and the MLE encryption key σ, and outputs the message authentication code R, the hash r of the combined message and code, and the locked ciphertext and tag C_{MLE} and C_{tag}. A random value $R \in \{0,1\}^k$ and a random $\sigma \in \mathbb{Z}_p$ are selected. The message-locked encryption computes $r = H_2(m\|R)$ and $C = (m\|R) \oplus H_1(e(g,g)^{\alpha r t})$, while the ciphertext tag is defined as $C_{tag} = \phi \oplus \varphi \oplus H_2(m') \oplus H_2(R)$. The ground control station stores $\{m_{tag}, C_{tag}, R, (\mathbb{A}, \rho)\}$ on the blockchain.
(3) The GCS sets an access structure $(\mathbb{A}, \rho)$, where $(\mathbb{A}, \rho)$ is an $l \times n$ matrix and ρ is a mapping that associates each row of $\mathbb{A}$ with an attribute in the universal set U. The partial attribute-based encryption function $\mathrm{ABE.UserEnc}(pp, (\mathbb{A}, \rho), C_{MLE}, r, C)$ is then invoked to perform partial encryption, resulting in the output of a partial ciphertext C_{user}. Based on the parameters of the MLE encryption, a random vector $\vec{\mu} = (\sigma, y_2, \ldots, y_n) \in Z_p^n$, rt is set to serve as the secret in a linear secret sharing scheme, and $y_2, \ldots, y_n$ are randomly chosen from Z_p. For each row $\mathbb{A}_j$ in $\mathbb{A}$, the set $\lambda = \{\lambda_j = \vec{\mu} \cdot \mathbb{A}_j, j \in [1,l]\}$ is computed. The resulting partial ciphertext is constructed as $C_{\mathrm{user}} = \{C_1 = g^{rt}, C_{2,j} = C_{MLE}, g^{a\lambda_j}, j \in [1,l]\}$.
(4) Upon receiving the partially encrypted data, the cloud server executes the outsourced encryption algorithm $\mathrm{ABE.OutEnc}(pp, (\mathbb{A}, \rho))$. This algorithm takes as input the system public key pp and the access control policy $(\mathbb{A}, \rho)$, and outputs a partially encrypted ciphertext C_{out}.

Random values $r_j \in Z_p, j \in [1,l]$ are selected, and the resulting ciphertext C_{out} is constructed as shown in Equation 4. By combining the MLE ciphertext with the partially encrypted ciphertext, the cloud server generates and stores the complete ciphertext CT, as defined in Equation 5.

$$C_{\mathrm{out}} = \left(C_{3,j} = f_{\rho(j)}^{-r_j}, \quad C_{4,j} = g^{r_j}, \forall j \in [1,l]\right) \tag{4}$$

$$\begin{aligned} CT &= ((\mathbb{A}, \rho), C_{tag}, C_{\mathrm{user}}, C_{\mathrm{out}}) \\ &= \left((\mathbb{A}, \rho), C_{MLE}, C_{tag}, C_1, \{C_{2,j}, C_{3,j}, C_{4,j}\}_{j \in [1,l]}\right) \end{aligned} \tag{5}$$

4.3 Data Request and Decryption Phase

In this phase, the drone Dr_2 sends a data request to the GCS_2, which then forwards the request to the cloud server. The cloud server invokes the access control smart contract on the blockchain to verify whether the attributes of Dr_2 satisfy the specified access structure. If the verification succeeds, the cloud server performs partial decryption and sends the partially decrypted data to GCS_2. The sequence diagram of this process is shown in Figure 4.
(1) The drone Dr_2 sends a request message $Request = \{DID2, S_{D2}, m_{tag}, TK_{D2}\}$ to the GCS_2, where DID_2 denotes the identity of the requesting drone, S_{D2} represents its attribute set, and m_{tag} is the tag of the requested data. The GCS invokes the blockchain-based access control smart contract ACC_SC which verifies whether the condition $V(S_{D2}, (\mathbb{A}, \rho)) = 1$ holds. If the verification is successful, the contract returns $\{V, C_{tag}\}$ to the cloud server and $\{V, C_{tag}, R\}$ to GCS_2, where V indicates that access has been granted.
(2) Upon receiving the broadcast message indicating the completion of the smart contract execution, the cloud server sends the corresponding ciphertext tag C_{tag} along with the ciphertext CT. The GCS then executes the algorithm $ABE.OutDec(pp, (\mathbb{A}, \rho), TK_{D2}, CT)$, where the input includes the private key $TK_{D2} = (K, L, \{K_x\}_{x \in S})$ and the ciphertext $CT = ((\mathbb{A}, \rho), C_{tag}, C_{user}, C_{out})$. The output is the partially decrypted ciphertext D_{out}, and this decryption algorithm is executed on the cloud server, as defined in Equation 6. Let $J = \{j : \rho(j) \in S\} \subset \{1, \ldots l\}$ exist an element $\theta_j \in Z_p^*$ such that $\sum_{j \in J} \theta_j \cdot \mathbb{A}_j = (1, 0, \ldots, 0)$.

$$D_{\text{out}} = e(g,g)^{r\sigma\alpha/z} = \frac{e(K,C_1)}{e\left(\prod_{j\in J} C_{2,j}^{\theta_j}\cdot C_{3.j}^{\theta_j}, L\right)\cdot \prod_{j\in J} e\left(C_{4,j}^{\theta_j}, K_{\rho(j)}\right)} \tag{6}$$

(3) After receiving the partially decrypted ciphertext from the cloud server, the drone Dr_2 invokes the $ABE.UserDec$ function. It takes as input the system public key pp, the private key z_{D2}, and the partially decrypted ciphertext D_{out}, and performs the computation as defined in Equation 7.

$$C' = H_1\left(D_{\text{out}}{}^{z_{D2}}\right) = H_1\left(e(g,g)^{\alpha r\sigma}\right) \tag{7}$$

(4) The drone then performs message-locked decryption using MLE. $\text{Dec}(pp, R, C_{MLE}, C')$. The input parameters include the system public key pp, the message m, and the access control policy $(\mathbb{A}, \rho)$. The outputs are the recovered plaintext m' from message-locked decryption and the corresponding proof π. The decryption is computed as $m'\|R = C \oplus C' = (m'\|R) \oplus H_1\left(e(g,g)^{\alpha r\sigma}\right) \oplus H_1\left(e(g,g)^{\alpha r\sigma}\right)$. By retrieving R from the blockchain, the drone obtains m', and subsequently computes the proof $\pi = \{H_2(m'), H_2(R)\}$.

(5) The drone Dr_2 sends the message proof π to the blockchain. The smart contract VER_SC on the blockchain verifies whether the condition $C_{tag} = \phi \oplus \varphi \oplus H_2(m') \oplus H_2(R)$ holds. If the verification succeeds, the contract outputs True; otherwise, it outputs False and triggers a penalty mechanism to punish the cloud server responsible for the outsourced encryption.

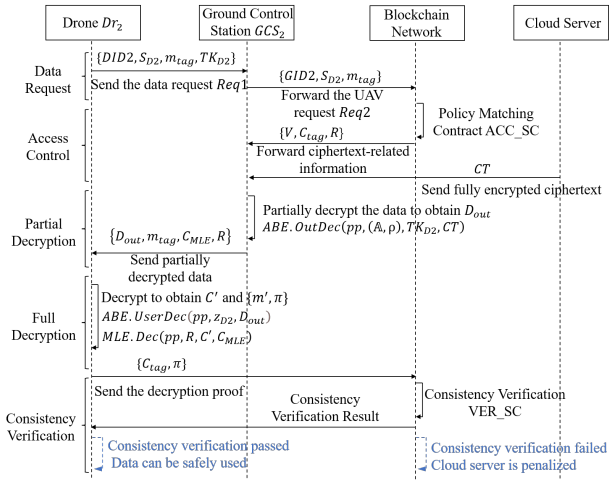

Figure 4: The sequence diagram of the decryption request phase in data sharing scheme, including access-policy verification through the blockchain and outsourced partial decryption performed by the cloud and ground station. The drone completes the final decryption and submits a proof to the blockchain for ciphertext-consistency verification.

4.4 Design of Smart Contracts

The design concept of the ACC_SC contract is to implement an attribute-based access control mechanism, while introducing a punishment mechanism to restrict access from malicious users. The ACC_SC contract initializes the number of users (userNum) and number of attributes (ArrNum), and maintains a punish mapping that records the punishment status of each user.

The ArrPolicyVerify function is used to verify user access requests, and its pseudocode is shown in Algorithm 1. The function receives the user attribute matrix ArrUser, the cloud attribute matrix PolCloud, as well as the user ID (userId) and address (userAdd) as parameters. First, the function records the user ID into the punish mapping and checks whether the user is in a punishment state. If the user is punished, the function returns a string indicating that the user has been punished. Otherwise, it calculates the access result based on the attribute matrices. If the calculation result is 0, it means that the user has the correct access permission, and the function returns a string indicating authorized access. Otherwise, it returns a string indicating that the user's access is denied.

The Punishment function is used to determine whether the user needs to be punished, and its pseudocode is shown in Algorithm 2. The function takes the user' s address (userAdd) as a parameter. First, it records the current time as nowTime and checks whether the user' s error time exceeds one minute. If it exceeds one minute, it means that the punishment period has ended; the function resets the error count to 1 and returns false, indicating that the user does not need to be punished. If it does not exceed one minute, the function checks whether the user' s error count exceeds three times.

Algorithm 1 Policy matching function (ACC_SC.ArrPolicyVerify)

Input: uint[][] ArrUser, uint[] PolCloud, uint userId, address userAdd
Output: string Access

```
function ARRPOLICYVERIFY(ArrUser, PolCloud, userId, userAdd)
    punish[userAdd].clo ← userId
    if punish[userAdd].falseTime≠0
        ∧PUNISHMENT(userAdd) then
        Access ← 'punish'
    else
        result ← 0
        for i ← 0 to resourceNum − 1 do
            result+ = PolCloud[userId] × ArrUser[i][userId]
        end for
        if result=0 then
            Access ← 'right'
        else
            punish[userAdd].falseTime ← block.timestamp
            Access ← 'wrong'
        end if
    end if
    return Access
end function
```

Algorithm 2 Punishment Function (ACC_SC.Punishment)

Input: address userAdd
Output: bool isPenalty

```
function PUNISHMENT(userAdd)
    if nowTime − punish[userAdd].falseTime > 1 minutes then
        punish[userAdd].falseTime ← nowTime
        punish[userAdd].falseNum ← 1
        isPenalty ← false
    else
        if punish[userAdd].falseNum > 3 then
            isPenalty ← true
        else
            punish[userAdd].falseNum ← punish[userAdd].falseNum + 1
            isPenalty ← false
        end if
    end if
    return isPenalty
end function
```

If it does, it means the user should be punished, and the function returns true. Otherwise, the function increments the error count and returns false, indicating that the user does not need to be punished.

5 Security Analysis

The attribute-based outsourced encryption and decryption protocol designed in this paper does not modify the underlying algorithm of attribute-based encryption. Based on outsourced decryption, the MLE algorithm is integrated into the attribute encryption process, assigning part of the encryption tasks to a trusted ground station while delegating the computation tasks unrelated to the secret values in attribute encryption to the cloud server. Therefore, to verify that the cloud server cannot obtain any information about the plaintext m in this system, a formal security proof is provided for the outsourced encryption and decryption part of the protocol. According to the security objectives of this paper, the proposed method is analyzed from the following three aspects to evaluate its security in the drone cloud–edge–end collaborative data sharing scenario:

(1) Data Confidentiality

In end-to-end data sharing, data confidentiality must be ensured to prevent sensitive information from being accessed by unauthorized third parties. In this scheme, the attribute-based outsourced encryption algorithm ensures that data remains encrypted throughout the sharing process. Before encryption, the attribute-based encryption algorithm assigns a series of attributes to the data, and only users who meet these attributes can decrypt it. Therefore, the attribute-based encryption approach guarantees that even if the data is intercepted during transmission, unauthorized users without the correct attributes cannot decrypt it, thereby effectively protecting the confidentiality of the data.

$MLE.Enc(pp, m, \sigma) \rightarrow \{R, r, C_{MLE}, C_{tag}\}$ is the MLE encryption process. The security of this algorithm depends on the security of the encryption key σ and the encryption algorithm itself. Since σ is randomly generated and unpredictable, C_{MLE} and C_{tag} do not reveal any information about m. In the outsourced encryption process, $ABE.OutEnc(pp, (\mathbb{A}, \rho)) \rightarrow C_{out}$, the cloud server does not directly access the message m or any information that could identify m. This function only receives the public parameters pp and the access control policy $(\mathbb{A}, \rho)$, and performs partial encryption computations, which themselves do not leak any information about m.

$ABE.UserDec(pp, z, D_{out}) \rightarrow C'$ is the terminal decryption process. The terminal possesses the private key z and the partially decrypted ciphertext D_{out} from the cloud server. These pieces of information are sufficient to complete the decryption process and recover the ciphertext C'. However, none of this information is included in the partial encryption or decryption performed on the cloud server, meaning that the cloud server cannot access these private keys or the fully decrypted ciphertext.

$MLE.Dec(pp, R, C_{MLE}, C') \rightarrow \{m', \pi\}$ is the MLE decryption process. Only users with the complete decryption information C' can perform this process to recover the original message m'. Since the cloud server does not have access to the ciphertext C'_{MLE} and the private key z, it cannot recover the plaintext m.

All sensitive information (such as m, the private key z, and the complete ciphertext C') remains confidential throughout the encryption and decryption processes and is neither stored nor transmitted on the cloud server.

(2) Data Integrity

To ensure data integrity during transmission, this paper uses a consistency verification smart contract for validation. The contract VER_SC verifies whether $C_{tag} = \phi \oplus \varphi \oplus H_2(m') \oplus H_2(R)$ holds, in order to determine whether the data has been correctly decrypted. The smart contract runs on the blockchain and does not store or transmit the actual message m, thereby preventing any leakage of plaintext information. Once the encrypted data is sent, the receiver can use the smart contract to verify whether the received data has been tampered with. The immutability of the blockchain ensures that once the data and its verification information are recorded, they cannot be modified or forged later. This mechanism guarantees that data remains in its original state during transmission between nodes. The correctness of outsourced encryption is verified through the consistency verification smart contract, which executes automatically without human intervention, ensuring the transparency of the verification process.

In practical deployments, edge nodes such as GCSs may be partially compromised. A malicious GCS could leak intermediate ciphertexts $\{C_{user}, C_{MLE}\}$, forge resource requests, or tamper with policy evaluation. However, it still cannot recover the plaintext m, as the ABE user key z is stored exclusively on UAVs and the verification tag R is protected by the blockchain ledger, making plaintext recovery require collusion across multiple domains. Moreover, any unauthorized modifications to access policies are verifiable on chain and subject to penalty contracts, thereby preserving authorization integrity even under insider edge attacks.

(3) Access Control

The proposed method in this paper delegates cloud-layer access control authority to a blockchain network composed of edge-layer nodes, implementing fine-grained access control through a policy-matching smart contract. The contract ACC_SC verifies whether $V(S, (\mathbb{A}, \rho)) = 1$ holds, to determine whether the user attributes match the access policy. The designed access permissions in this paper can be dynamically adjusted according to user attributes, ensuring that only users with the appropriate attributes can access specific data. The attribute-based access control mechanism enhances the flexibility and security of access control while reducing the risks of centralized management and single points of failure. The smart contract is used solely for verifying access policies and data consistency and does not store the original message m.

6 Experiments

To demonstrate the time consumption of the encryption and decryption algorithms, the smart contract execution time, concurrency performance, and gas consumption in the blockchain-based secure data access control method, a series of experiments were designed in this paper to evaluate its performance.

6.1 Experimental Setup

This paper conducts simulation tests on the application of the attribute-based outsourced encryption algorithm in data sharing scenarios involving terminal devices with limited resources, as this algorithm directly affects sharing efficiency. Meanwhile, the performance of the smart contracts in policy matching and data verification is evaluated to ensure the security and verifiability of data during transmission, as these operations impact the response speed of the access control system. To comprehensively verify the effectiveness of the above algorithmic modules in practical applications, this section performs detailed measurements of computational resource consumption for key components such as the attribute-based outsourced encryption algorithm and the deployment and invocation of smart contracts. The entire simulation process and performance analysis results are recorded in detail.

For the implementation of the outsourced attribute-based encryption scheme, Java was used to develop the ABE algorithm and the outsourced encryption and decryption components. Regarding the blockchain component, a multi-group, dual-node consortium blockchain system was built based on FISCO BCOS version 2.9.1. JavaSDK was employed to facilitate interactions with smart contracts, which were written in Solidity version 0.5.0. All experiments related to attribute-based encryption were conducted in the local environment, while blockchain-related experiments were deployed on the cloud server. The compilation of all components relied on the OpenJDK v14.0.21 compiler.

In the simulation of outsourced attribute-based encryption and decryption, this paper emulates terminal devices in a local environment and conducts tests on both conventional ABE and outsourced ABE under identical conditions. To better approximate real-world scenarios, the simulation considers the possible number of attributes held by drones and GCSs. A set of attributes is randomly generated and assigned to drones and ground stations. Encryption and decryption operations are then performed on these attributes under consistent testing conditions, with the corresponding computation time and communication overhead recorded.

For the smart contract simulation, the focus is on the number of concurrent access requests from drone nodes and the associated time consumption. To simulate concurrent access, a multithreaded approach is adopted in the cloud environment, where each thread represents a drone node accessing the smart contract. The number of simulated drones corresponds to the level of concurrency. By varying the number of concurrent threads, different levels of access concurrency are emulated. During the simulation, the time consumed by each access operation is recorded, and performance metrics such as average response time are calculated.

6.2 Computational Resource Overhead of Outsourced Attribute-Based Encryption

Outsourced attribute-based encryption is a critical cryptographic technique that enables fine-grained encryption and decryption of data, thereby enhancing information security. To support attribute-based encryption on resource-constrained devices, the process involves multiple stages, including key generation, partial encryption, full encryption, partial decryption, and complete decryption—each of which incurs varying levels of computational resource consumption. Therefore, measuring and analyzing the computational resource overhead associated with outsourced attribute-based encryption is essential for understanding its performance characteristics.

To evaluate the key generation efficiency of outsourced attribute-based encryption, this paper conducts a detailed experimental analysis of the KeyGen process, aiming to assess its computational resource consumption under varying user attribute set sizes. The experiment defines a universal attribute set U consisting of 100 elements and simulates a range of user scenarios with different attribute subsets S derived from this set. As illustrated in Figure 5, the horizontal axis represents the number of elements in the attribute subset S, increasing from 10 to the full set of 100. Correspondingly, the vertical axis quantifies the time required to execute the KeyGen operation as the size of S increases.

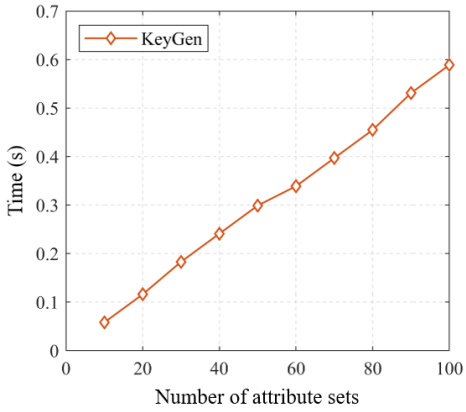

Figure 5: Time cost of attribute-based outsourcing encryption key generation.

Experimental results indicate that in outsourced attribute-based encryption, the computational resources consumed during the KeyGen process exhibit a clear linear increase as the number of attributes in the user set S grows. This behavior stems from the nature of the encryption mechanism, which requires a dedicated cryptographic computation for each attribute in the set. Each additional attribute necessitates an additional encryption operation. Thus, the size of the attribute set S directly determines the computational workload during key generation. While the time required for attribute-independent fixed operations remains relatively constant, the cumulative effect of increasing attributes leads to a linearly growing number of encryption computations, resulting in a linear increase in overall computation time. Therefore, the number of attributes held by a user is the primary factor influencing the efficiency of key generation in outsourced attribute-based encryption scenarios.

To further investigate the computational time costs associated with partial and full decryption operations in outsourced attribute-based encryption, the experiment constructs a universal attribute set U as the basis and generates a series of access trees T containing varying numbers of attributes. This setup is designed to simulate different levels of encryption and decryption complexity encountered in real-world applications. As illustrated in Figures 6 and 7, the horizontal

axis represents the number of attributes contained within each access tree T, while the vertical axis records the execution time required for the corresponding encryption and decryption tasks. These two figures clearly demonstrate the trend that, as the number of attributes in the access tree increases, the time required for encryption and decryption operations also changes accordingly.

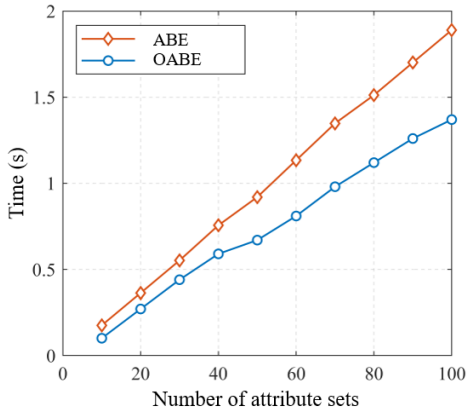

Figure 6: The time cost of attribute-based outsourcing encryption.

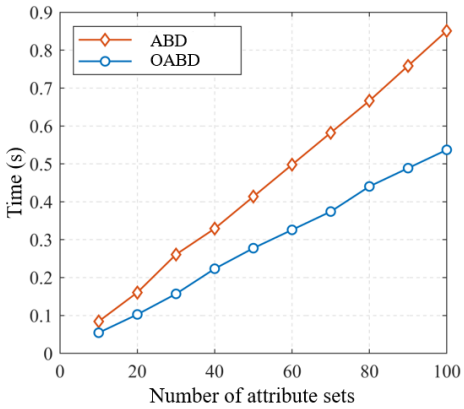

Figure 7: The time cost of attribute-based outsourcing decryption.

By comparing the two figures, it is evident that both attribute-based encryption and outsourced attribute-based encryption exhibit a synchronous increase in execution time as the number of processed attributes grows. Moreover, encryption operations consistently consume more time than decryption operations. This is primarily because the encryption process involves vector searches to determine valid combinations capable of reconstructing the secret value, whereas decryption merely requires using the corresponding private key as input for computation. As a result, the encryption phase is inherently more time-consuming.

Moreover, a comparison of the time savings achieved through outsourced encryption and outsourced decryption reveals that outsourcing encryption yields greater reductions in computational cost. Since decryption operations generally require less time than encryption, they are more suitable for resource-constrained devices. Additionally, in the outsourced attribute-based encryption scheme adopted in this study, MLE operations are incorporated to ensure verifiability, which further increases the time consumption of encryption on terminal devices. Therefore, in the data sharing scheme proposed in this paper, the encryption tasks originally performed on the terminal are offloaded to the edge-layer GCS, while decryption is retained on the drone side.

In summary, whether performing outsourced attribute-based encryption and decryption or conventional attribute-based operations, the consumption of computational resources is directly and closely related to the number of involved attributes. As the number of attributes increases, the computational overhead for encryption and decryption grows linearly. Furthermore, the analysis of time consumption across different components confirms the rationality and effectiveness of the proposed method's design.

6.3 Smart Contract Performance Evaluation

In the performance evaluation of smart contracts, this paper focuses on several key metrics, including concurrent execution time, and gas consumption. It should be noted that gas consumption reflects the computational complexity and storage overhead of on-chain operations, the measured gas values capture the actual execution cost of the smart-contract functions.To ensure the accuracy of the analysis, a series of controlled and repeated tests are conducted under consistent experimental conditions to obtain reliable results.

(1) Concurrent Execution Time

Concurrent time consumption refers to the amount of time required by the system to handle multiple concurrent tasks or requests. Concurrency involves competition and scheduling of system resources, as well as interactions among tasks or requests. Lower concurrent time consumption indicates that the system can manage concurrent tasks more efficiently, minimizing waiting times and resource contention among tasks.

In the testing of the *ACC_SC* contract, this paper first simulates transactions under varying loads to determine the maximum processing capacity of the smart contract. As shown in Figure 8, the time consumption of the contract initially increases linearly with the load. After reaching a certain threshold, the growth rate begins to level off slightly.

Based on the experimental results, it is observed that the *ACC_SC* contract accounts for the majority of invocation operations in the total latency, thus playing a significant role in the overall performance evaluation. To assess the efficiency of the proposed method, a comparative experiment is conducted against the SMACS approach introduced by Liu et al. [18], which implements fine-grained access control via smart contracts on the Ethereum platform—a mechanism similar to the one adopted in this study. For a fair comparison, the SMACS contract was reproduced and deployed on the FISCO BCOS blockchain platform under identical testing conditions. The results show that when the number of concurrent invocations is below 20, SMACS exhibits better performance. However, as the concurrency level exceeds 20, the proposed method outperforms SMACS. These findings indicate that the proposed method achieves superior performance under high-concurrency scenarios.

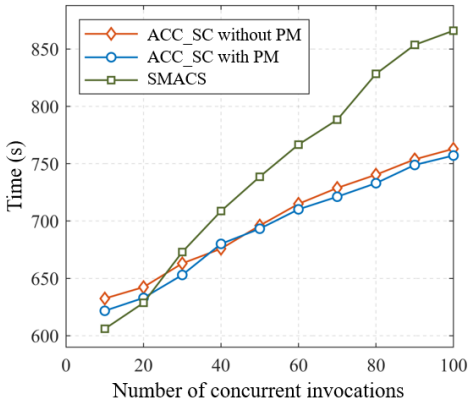

Figure 8: The time cost of ACC_SC with changes in concurrency.

Although SMACS offloads complex verification and management operations to off-chain infrastructure—implementing only token-based access control on-chain—token-based mechanisms can still introduce considerable overhead when the system experiences high concurrency. In application scenarios demanding high throughput, it is essential to optimize contract design to reduce execution time and minimize network load. Therefore, smart contracts should be designed to eliminate redundant operations as much as possible to ensure high throughput and efficient execution.

To evaluate the impact of increasing the number of attributes on the time consumption of smart contracts when processing 100 concurrent transactions, a series of simulation experiments were conducted. In these experiments, different sizes of attribute sets were employed, and the corresponding processing times were recorded. The experimental results are illustrated in Figure 9. According to the results, it can be observed that as the size of the attribute set increases, the processing time of the smart contract also exhibits a rising trend. This indicates that a greater number of attributes leads to increased execution time for the contract. Moreover, the growth in time consumption appears to be approximately linear—each additional attribute results in a nearly proportional increase in processing time.

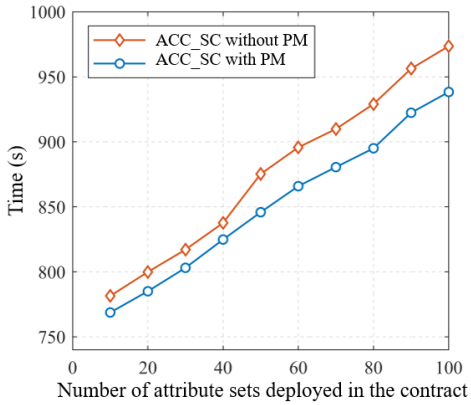

Figure 9: The time cost of ACC_SC with changes in the number of attribute sets.

These findings offer important insights for the design and optimization of smart contracts. In practical development, the number of attributes used within a contract should be carefully considered to avoid excessive processing delays. Developers are advised to balance actual functional requirements with performance constraints, and to control the number of attributes accordingly in order to ensure efficient contract execution.

Based on the above experimental results, the access control penalty mechanism proposed in this study can effectively reduce the time consumption of smart contracts under certain conditions. In particular, when the system experiences high levels of concurrent operations or involves a large number of attributes, the penalty mechanism significantly enhances the execution efficiency of smart contracts.

The results of the concurrent invocation test for the *VER_SC* contract are shown in Figure 10. As the system load increases, the overall concurrent time consumption of the *VER_SC* contract also increases, accompanied by noticeable fluctuations in latency. The *VER_SC* contract performs operations such as computing the bitwise XOR of input parameters and comparing the result with a given input to determine equality. During concurrent testing, multiple users simultaneously send requests to the contract and execute the same operations, which requires the contract to handle multiple concurrent requests in real time. Consequently, the contract's concurrent performance becomes a critical metric in evaluating its ability to manage high-concurrency scenarios.

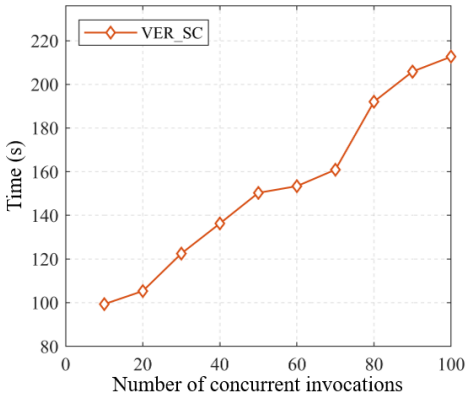

Figure 10: The time cost variation of VER_SC with concurrency changes.

The observed performance variability may stem from resource contention within the contract. When multiple requests access shared components of the contract simultaneously—such as reading or writing shared variables or accessing on-chain storage—resource contention can occur. Another contributing factor is the indeterminacy of the scheduling algorithm in the underlying system. The contract's concurrent performance is influenced by the blockchain platform's scheduling policies. Imbalances in load distribution or task prioritization may lead to uneven execution sequences and time slice allocation across requests, thereby resulting in fluctuating concurrency performance.

(2) Gas Consumption

Gas consumption is a key metric for evaluating the economic efficiency of smart contracts, representing the cost required to execute contract operations. On blockchain platforms such as Ethereum and FISCO BCOS, each operation incurs a specific Gas cost, meaning that Gas consumption directly affects the overall cost for users interacting with smart

contracts. In this paper, we conducted Gas consumption tests for various smart contract operations. Table 2 presents the corresponding function calls for each operation and the amount of Gas consumed.

Table 2: The gas consumption of smart contract functions

Operation	Function Call	Gas consumption
ACC_SC contract deployment	*ACC_SC*.Deploy	636425
View account status	*ACC_SC*.Punishment	31413
Attribute policy matching	*ACC_SC*.ArrPolicyVerify	92005
VER_SC contract deployment	*VER_SC*.Deploy	127609
Consistency verification	*VER_SC*.ConformVerify	37082

This paper conducted individual tests for each function within the smart contracts to quantify their average gas consumption. The experimental results were obtained using Remix, with Solidity version 0.5.0. The test results indicate that computationally intensive or storage-related operations significantly increase gas costs. For instance, operations such as contract deployment and attribute policy matching require modifications to the data stored on the blockchain, incurring both execution gas and storage gas costs. In this experiment, the deployment of the *ACC_SC* contract consumed 636,425 gas, while the deployment of the *VER_SC* contract consumed 127,609 gas. Contract deployment involves compiling and storing the contract code, thus requiring substantial computational resources and gas consumption.

Gas consumption varies across different operations. For example, the operation to view account status *ACC_SC.Punishment* consumed 31,413 gas, whereas the consistency verification operation $VER\_SC.ConformVerify$ consumed 37,082 gas. These differences arise from the varying computational complexity and data access demands of each operation. Notably, the attribute policy matching operation $ACC\_SC.ArrPolicyVerify$ incurred a relatively high gas cost of 92,005. This is because the operation involves validating and matching attribute policies, which entails complex computations and data comparisons, thereby consuming more computational resources and gas.

7 Conclusion

This paper addresses the problem of end-to-end secure data sharing in a drone-centric cloud–edge–device collaborative environment. By integrating blockchain technology with outsourced attribute-based encryption approach, an efficient and secure data sharing method tailored to resource-constrained environments is proposed. To tackle typical challenges such as limited computational capabilities of terminal devices, highly dynamic network topologies, and untrusted communication channels, the proposed method employs outsourced ABE to protect data confidentiality. The encryption transformation and decryption operations are offloaded to edge nodes and cloud servers, thereby effectively reducing the computational burden on terminals and minimizing system latency. Smart contracts are employed to dynamically enforce access control policies and to verify the consistency of outsourced encryption results, ensuring both flexibility and correctness in access control. Finally, the proposed method is theoretically analyzed and experimentally evaluated in terms of security and system performance. The results demonstrate that the proposed method not only guarantees data confidentiality and integrity but also achieves high computational efficiency and practical deployability.

Funding

This work is supported by the Open Research Projects of the Key Laboratory of Blockchain Technology and Data Security of the Ministry of Industry and Information Technology for the year 2025 under Grant KT20250009, 2025 Key Research Project of China Railway Information Technology Group Co., Ltd under Grant WJZG-CKY-2025014(2025N01).

Author Contributions

Wen Zhang completed the experimental design and wrote the paper. Pengrui Chen assisted in completing the experiments. Li Duan provided methodological ideas. Yimeng Feng and Lufeng Feng proposed changes to the paper as a whole. All authors have read and agreed to the published version of the manuscript.

Conflict of Interest

All the authors declare that they have no conflict of interest.

References

[1] Muchiri, G.N., Kimathi, S.: A review of applications and potential applications of UAV. In Proceedings of the 2016 Sustainable Research & Innovation (SRI) Conference, pp. 280-283 (2022)

[2] Poorvi, J., Kalita, A., Gurusamy, M.: Reliable and efficient data collection in uav based iot networks. IEEE Communications Surveys & Tutorials (2025). https://doi.org/10.1109/COMST.2025.3550274

[3] Kumar, J., Kumar, M., Pandey, D.K., Raj, R.: Encryption and authentication of data using the IPSEC protocol. In Proceedings of the Fourth International Conference on Microelectronics, Computing and Communication Systems, pp. 855-862 (2020). https://doi.org/10.1007/978-981-15-5546-6_71

[4] Reimers, E.: On the security of TLS and IPsec: Mitigation through physical constraints. Bachelor Thesis, Linköping University (2015)

[5] Xia, T., Wang, M., He, J., Lin, S., Shi, Y., Guo, L.: Research on identity authentication scheme for uav

communication network. Electronics **12**(13), 2917 (2023). https://doi.org/10.3390/electronics12132917

[6] Makhdoom, I., Abolhasan, M., Abbas, H., Ni, W.: Blockchain's adoption in IoT: The challenges, and a way forward. Journal of Network and Computer Applications **125**, 251-279 (2019). https://doi.org/10.1016/j.jnca.2018.10.019

[7] Jiang, Y., Ma, B., Wang, X., Yu, G., Yu, P., Wang, Z., Ni, W., Liu, R.P.: Blockchained federated learning for internet of things: A comprehensive survey. ACM Computing Surveys **56**(10), 1-37(2024). https://doi.org/10.1145/3659099

[8] Ramos, S., Cruz, T., Simões, P.: Security and safety of unmanned air vehicles: An overview. In ECCWS 2021 20th European Conference on Cyber Warfare and Security, pp. 357-368 (2021)

[9] Japp, W., Lee, V., Vagicherla, S.A., Rubio-Medrano, C.: Fly-ABAC: Attribute Based Access Control for the Navigation of Unmanned Aerial Vehicles. In 2024 IEEE International Conference on Big Data (BigData), pp. 7471-7476 (2024). https://doi.org/10.1109/BigData62323.2024.10825924

[10] Jeong, H.J., Ha, Y.G.: RBAC-Based UAV Control System for Multiple Operator Environments. In the proceedings of the 2012 International Conference on Advanced Software Engineering and Its Applications (ASEA 2012) and the 2012 International Conference on Disaster Recovery and Business Continuity (DRBC 2012), pp. 210-217 (2012). https://doi.org/10.1007/978-3-642-35267-6_27

[11] Pang, Y., Chen, Z.: Security Scheme of UAV Flight Control Based on Attribute-Based Access Control Policy. Computer Science **51**(4), 366–372 (2024). https://doi.org/10.11896/jsjkx.230200135

[12] Sahai, A., Waters, B.: Fuzzy identity-based encryption. In Annual international conference on the theory and applications of cryptographic techniques, pp. 457-473 (2005). https://doi.org/10.1007/11426639_27

[13] Liu, Q., Wang, Z., Yu, C., Wang, Z.: Efficient Attribute-Based Encryption Scheme from Lattices for Cloud Security. Netinfo Security, **23**(09), 25–36 (2023). https://doi.org/10.3969/j.issn.1671-1122.2023.09.003

[14] Ge, C., Susilo, W., Baek, J., Liu, Z., Xia, J., Fang, L.: A verifiable and fair attribute-based proxy re-encryption scheme for data sharing in clouds. IEEE Transactions on Dependable and Secure Computing **19**(5), 2907-2919 (2021). https://doi.org/10.1109/TDSC.2021.3076580

[15] Sun, J., Xu, G., Zhang, T., Yang, X., Alazab, M., Deng, R.H.: Verifiable, fair and privacy-preserving broadcast authorization for flexible data sharing in clouds. IEEE Transactions on Information Forensics and Security **18**, 683-698 (2022). https://doi.org/10.1109/TIFS.2022.3226577

[16] Zhou, Z., Wan, Y., Cui, Q., Yu, K., Mumtaz, S., Yang, C. N., Guizani, M.: Blockchain-based secure and efficient secret image sharing with outsourcing computation in wireless networks. IEEE Transactions on Wireless Communications **23**(1), 423-435 (2023). https://doi.org/10.1109/TWC.2023.3278108

[17] Saini, A., Zhu, Q., Singh, N., Xiang, Y., Gao, L., Zhang, Y.: A smart-contract-based access control framework for cloud smart healthcare system. IEEE Internet of Things Journal **8**(7), 5914-5925 (2021). https://doi.org/10.1109/JIOT.2020.3032997

[18] Liu, B., Sun, S., Szalachowski, P.: Smacs: smart contract access control service. In 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 221-232 (2020). https://doi.org/10.1109/DSN48063.2020.00039

[19] Feng, C., Yu, K., Bashir, A.K., Al-Otaibi, Y.D., Lu, Y., Chen, S.: Efficient and secure data sharing for 5G flying drones: A blockchain-enabled approach. IEEE Network **35**(1), 130-137 (2021). https://doi.org/10.1109/MNET.011.2000223

[20] Bera, B., Saha, S., Das, A.K., Kumar, N., Lorenz, P., Alazab, M.: Blockchain-envisioned secure data delivery and collection scheme for 5g-based iot-enabled internet of drones environment. IEEE Transactions on Vehicular Technology **69**(8), 9097-9111 (2020). https://doi.org/10.1109/TVT.2020.3000576

[21] Mandal, S., Bera, B., Sutrala, A.K., Das, A.K., Choo, K.R., Park, Y.: Certificateless-signcryption-based three-factor user access control scheme for IoT environment. IEEE Internet of Things Journal **7**(4), 3184-3197 (2020). https://doi.org/10.1109/JIOT.2020.2966242

[22] Mandinyenya, G., Malele, V.: Formal Verification of a Blockchain-Based Security Model for Personal Data Sharing Using the Dolev-Yao Model and ProVerif. International Journal of Advanced Computer Science & Applications, **16**(9), 444-456 (2025). https://dx.doi.org/10.14569/IJACSA.2025.0160942

[23] Zhang, J.: Research on Blockchain-Based Secure Data Sharing Scheme in Cloud-Edge-End Collaboration. Master's Thesis, Beijing Jiaotong University (2024)