Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Hidden-Policy Attribute-Based Access Control for IoT Using Blockchain with Replay-Resistant Verification

[1]Saiyed Alwaz Hussain
[1]Integrated M.Tech Student
[12]School of Computing Science Engineering and Artificial Intelligence (SCAI)
[12]VIT Bhopal University, Sehore, Madhya Pradesh, India

[2]Sreevani Maddukuri
[2]Associate Professor Senior Grade 2
[12]School of Computing Science Engineering and Artificial Intelligence (SCAI)
[12]VIT Bhopal University, Sehore, Madhya Pradesh, India

***Abstract*— The rapid growth of Internet of Things (IoT) systems has increased the need for secure, fine-grained, and privacy-preserving access control. Existing blockchain-based attribute access control schemes improve decentralization and auditability, but may still expose sensitive policy information when attribute names or policy structures are stored publicly on-chain. They may also rely on off-chain trusted entities for challenge-response verification, which can weaken replay protection if those entities are compromised. This paper proposes a privacy-preserving blockchain-based IoT access control framework that extends ciphertext-policy attribute-based encryption (CP-ABE) with hidden on-chain policy metadata and smart-contract-managed replay-resistant verification. The proposed scheme blinds both attribute names and attribute values before blockchain storage, reducing policy leakage in healthcare, smart home, and industrial IoT environments. It introduces an on-chain nonce mechanism in which nonce generation, binding, verification, and consumption are enforced by a Solidity smart contract. Encrypted IoT data is stored off-chain using IPFS, while only blinded metadata and content identifiers are recorded on-chain. Hardhat-based gas analysis evaluates deployment feasibility across multiple IoT scaling scenarios. Overall, the framework improves policy confidentiality, replay resistance, auditability, and practical blockchain deployment awareness.**

Keywords*— *Access control, attribute-based encryption, blockchain, ciphertext-policy attribute-based encryption, gas cost analysis, IoT security, policy hiding, replay resistance, smart contract.

I. INTRODUCTION

The rapid growth of the Internet of Things (IoT) has enabled large-scale data collection through smart sensors, wearable devices, healthcare systems, smart home appliances, and industrial control environments [1], [2], [3]. These devices generate sensitive data such as location information, health parameters, activity records, environmental readings, and usage patterns. Since many IoT devices have limited computation and storage capacity, collected data is commonly outsourced to cloud servers or decentralized storage systems. However, outsourcing sensitive IoT data introduces security and privacy risks, especially when storage providers cannot be fully trusted.

Access control is a key mechanism for protecting outsourced IoT data. Traditional identity-based or role-based models may not provide enough flexibility for dynamic IoT environments because access decisions often depend on multiple attributes such as role, department, device type, or clearance level. Attribute-based encryption (ABE) was introduced to support encryption based on descriptive attributes rather than fixed identities [4], and was later developed for fine-grained encrypted access control [5]. Ciphertext-policy attribute-based encryption (CP-ABE) allows data owners to define access policies over user attributes [6], [7]. A ciphertext can be decrypted only when the user's attributes satisfy the policy, making CP-ABE suitable for one-to-many encrypted data sharing in IoT systems.

Blockchain technology has also been applied to access control to reduce dependence on centralized authorities [8], [9], [2]. Its decentralization, immutability, transparency, and traceability make it useful for storing access-control metadata and recording access transactions in a tamper-resistant manner. Smart contracts can further automate access-control operations, making blockchain suitable for auditable IoT access control.

Yang et al. [1] proposed a blockchain-based attribute access control scheme for IoT data protection. Their scheme combines CP-ABE, LSSS-based expressive policies, blockchain traceability, and challenge-response verification. However, several limitations remain. First, its policy hiding is partial: attribute values are hashed, but attribute names mapped by ρ remain visible on-chain. This may reveal whether access depends on role, department, clearance level, or medical specialization. Second, challenge-response verification depends on an off-chain Attribute Management Node (AMN), so replay handling is not fully enforced by blockchain state. Third, the evaluation mainly reports Java-based algorithm execution time and does not include Solidity deployment or gas-cost analysis. Finally, ciphertext storage relies on a cloud server, which partially weakens the decentralization objective.

Lai et al. [10], [11] provide important cryptographic foundations for policy-hiding CP-ABE. Lai et al. [10] introduced expressive CP-ABE with partially hidden access structures, while Lai et al. [11] studied ciphertext-policy hiding CP-ABE in a full-security setting. However, these works mainly focus on cryptographic construction and do not address blockchain integration, smart-contract verification, IoT deployment, decentralized ciphertext storage, or gas-cost evaluation. Therefore, system-level mechanisms are required for practical blockchain-based IoT access control.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

To address these gaps, this paper proposes a privacy-preserving blockchain-based IoT access control framework with hidden on-chain policy metadata and replay-resistant verification. The proposed scheme hides both attribute names and attribute values before blockchain storage. For each policy attribute, a fresh random salt is generated, and a blinded representation is computed. Since salts are stored off-chain in encrypted form and are not revealed on-chain, public blockchain observers cannot directly infer attribute values or attribute categories used in the access policy.

The proposed framework also replaces AMN-dependent replay handling with a smart-contract-managed nonce verification process. A Solidity smart contract generates a fresh nonce for each access request, binds it to the requester and resource identifier, stores its state on-chain, and marks it as consumed after verification. Since each nonce can be used only once, replayed responses are rejected by contract logic. Encrypted IoT data is stored off-chain using IPFS, while only the content identifier and blinded access metadata are stored on-chain.

The main contributions are: (1) hiding attribute names and values before blockchain storage; (2) smart-contract-based nonce verification for replay-resistant access control without off-chain AMN nonce management; (3) IPFS-based ciphertext storage where only the content identifier is recorded on-chain; and (4) gas-cost evaluation of Solidity smart contract operations across multiple IoT scaling scenarios.

II. RELATED WORK

A. Attribute-Based Encryption and Policy Hiding

Attribute-based encryption (ABE) is widely used for fine-grained access control in distributed systems. ABE was first introduced to support encryption based on descriptive attributes rather than fixed identities [4], and was later extended for fine-grained encrypted access control [5]. In ciphertext-policy attribute-based encryption (CP-ABE), the access policy is associated with the ciphertext, while user secret keys are associated with attribute sets [6]. A user can decrypt the ciphertext only when the user's attributes satisfy the access policy. Waters [7] further developed expressive CP-ABE using LSSS-based access structures, making CP-ABE suitable for IoT data sharing where access may depend on roles, departments, device permissions, or security levels.

However, conventional CP-ABE schemes may reveal the access policy together with the ciphertext. In sensitive environments, the policy itself can disclose private information even when the data remains encrypted. Lai et al. [10] proposed expressive CP-ABE with partially hidden access structures, where sensitive attribute values are hidden while other policy information remains visible. Lai et al. [11] studied ciphertext-policy hiding CP-ABE in a full-security setting. These works provide important cryptographic foundations, but they do not address blockchain-based access control, smart contract verification, IoT deployment, decentralized ciphertext storage, or gas-cost evaluation.

B. Blockchain-Based IoT Access Control

Since the introduction of blockchain through Bitcoin [8], blockchain has been adopted in access control systems to reduce dependence on centralized authorities. Its decentralization, immutability, transparency, and traceability make it useful for recording access policies, user requests, and authorization events. Smart contracts can further automate access-control operations. Cruz et al. [9] proposed a role-based access control model using smart contracts, while Novo [2] proposed a blockchain-based access management architecture for IoT environments.

Despite these advantages, blockchain introduces privacy challenges because on-chain data is visible to network participants. If access policies or attribute information are stored directly on-chain, observers may infer sensitive information from policy metadata. Therefore, a blockchain-based IoT access control scheme must protect both encrypted data and on-chain access-control metadata. In addition, blockchain deployment introduces gas costs, which directly affect practical feasibility. Many existing works discuss blockchain-based access control conceptually but do not evaluate the cost of smart contract operations.

C. Blockchain and CP-ABE-Based IoT Security

Several works have combined blockchain, ABE, and privacy-preserving access control. Zhang et al. [3] studied efficient policy-hiding attribute-based access control for smart health systems, while Gao et al. [12] proposed TrustAccess, a blockchain-based ciphertext-policy and attribute-hiding access control scheme. These works show the importance of policy privacy and fine-grained access control in sensitive data-sharing environments.

Yang et al. [1] proposed a blockchain-based attribute access control scheme for IoT data protection. Their scheme combines CP-ABE, LSSS-based expressive policies, blockchain traceability, and challenge-response verification for attribute ownership. Compared with centralized access control, their approach improves decentralization and supports fine-grained data access. However, several practical and privacy-related gaps remain. First, policy hiding is partial because attribute values are hidden using hash values, while attribute names mapped through ρremain visible on-chain. This may reveal the structure of the access policy. Second, the challenge-response mechanism depends on an off-chain Attribute Management Node (AMN), which introduces an external trusted component. Third, the evaluation is based on Java implementation time and does not include Solidity smart contract deployment, gas-cost measurement, or Layer 2 cost comparison. Finally, ciphertext is stored on a cloud server, which keeps part of the architecture dependent on centralized storage.

To address these gaps, the proposed scheme extends the blockchain-based CP-ABE model by introducing hidden on-chain policy metadata, smart-contract-managed nonce verification, IPFS-based decentralized ciphertext storage, and gas-cost analysis. Table I summarizes the comparison between related schemes and the proposed framework.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

TABLE I. COMPARISON OF RELATED SCHEMES

Scheme	CP-ABE	Policy Hide	Attribute Name + Value Hidden On-Chain	Block chain	Replay Prot.	Gas Analysis
Lai et al. [10], [11]	Yes	Yes	No	No	No	No
Zhang et al. [3]	Yes	Yes	No	No	No	No
Gao et al. [12]	Yes	Yes	No	Yes	No	No
Yang et al. [1]	Yes	Partial	No	Yes	Off-chain	No
Proposed	Yes	Yes	Yes	Yes	On-chain	Yes

III. BACKGROUND AND PRELIMINARIES

This section summarizes the main concepts used in the proposed framework: CP-ABE, LSSS access structures, policy hiding, blockchain smart contracts, nonce-based replay prevention, and IPFS-based off-chain storage.

A. Ciphertext-Policy Attribute-Based Encryption

Ciphertext-policy attribute-based encryption (CP-ABE) is a public-key encryption technique for fine-grained access control. In CP-ABE, the ciphertext is associated with an access policy, while the user secret key is associated with a set of attributes [6]. A user can decrypt the ciphertext only if the attribute set satisfies the policy defined by the data owner. A CP-ABE scheme generally consists of four algorithms: $Setup(\lambda)$ generates the public key PK and master secret key MSK ; $Encrypt(PK, M, (A, \rho))$ encrypts message M under policy (A, ρ) and outputs ciphertext CT ; $KeyGen(PK, MSK, S)$ generates secret key SK_S for attribute set S; and $Decrypt(PK, SK_S, CT)$ recovers M only if S satisfies the policy. This notation follows prior CP-ABE works [6], [7], [10], [11].

B. LSSS Access Structure

A linear secret sharing scheme (LSSS) is commonly used in expressive CP-ABE to represent flexible access policies [7]. In an LSSS-based policy, (A, ρ) denotes the access structure, where A is an access matrix and ρ maps each row of A to an attribute. A user with attribute set S satisfies the policy if the corresponding rows of A can reconstruct the secret used during encryption. Yang et al. [1] use LSSS-based policies, and the proposed framework keeps this notation while changing how policy attributes are represented on-chain.

C. Policy Hiding

Policy hiding is important because the access policy itself may reveal sensitive information. In partial policy hiding, only part of the policy is hidden. Yang et al. [1] hide attribute values using hash values, but attribute names mapped by ρ remain visible on-chain. In the proposed scheme, both attribute names and attribute values are blinded before blockchain storage. For an attribute name $attr\_name_i$, the blinded form is computed as $blindName_i = H(attr\_name_i \parallel r_i)$, where H is a cryptographic hash function, r_i is a fresh random salt, and $\parallel$ denotes concatenation. A similar process is applied to attribute values. Since salts are not published on-chain, observers cannot directly recover plaintext policy attributes.

D. Blockchain, Smart Contracts, and Gas

A blockchain is a decentralized and tamper-resistant ledger. Since Bitcoin [8], blockchain has been used for decentralized trust and auditability. Smart contracts are blockchain programs that execute automatically when predefined conditions are met. Smart-contract-based access control has been studied for role-based access control and IoT access management [9], [2]. In the proposed framework, the smart contract registers blinded metadata, manages access requests, generates nonce values, verifies request freshness, and records access events. Since each on-chain operation consumes gas, gas-cost analysis is necessary to evaluate deployment feasibility.

E. Nonce-Based Replay Prevention and IPFS Storage

A replay attack occurs when an attacker reuses a previously valid authorization response. A nonce is a fresh one-time value used to prevent such reuse. In the proposed framework, the smart contract generates a nonce for each access request, stores it as pending, and marks it as consumed after the first verification attempt.

Storing large IoT ciphertexts directly on-chain is expensive. Therefore, encrypted IoT data is stored in IPFS or IPFS-compatible content-addressed storage. IPFS returns a content identifier CID , while the blockchain stores only the CID , blinded policy metadata, nonce states, and access logs.

IV. THREAT MODEL AND SECURITY GOALS

This section defines the system entities, adversary capabilities, and security goals considered in the proposed blockchain-based IoT access control framework.

A. System Entities

The proposed framework consists of five main entities. IoT devices, such as sensors, wearable devices, smart home appliances, and industrial monitoring devices, generate raw data. These devices may have limited computation, storage, and communication capabilities.

The data owner or IoT gateway defines access policies for IoT data. In practical deployment, the gateway may act on behalf of resource-constrained IoT devices by collecting data, encrypting it using CP-ABE, storing the ciphertext in off-chain storage, and registering blinded policy metadata on the blockchain.

The data user requests access to encrypted IoT data. The user has an attribute set S and a corresponding secret key SK_S. The user can decrypt the ciphertext only if S satisfies the access policy defined by the data owner, following the standard CP-ABE model [6], [7].

The attribute authority issues attribute-based secret keys to legitimate users according to the underlying CP-ABE construction [6], [7]. Unlike Yang et al. [1], the proposed framework does not delegate nonce generation and replay prevention to an off-chain AMN. Instead, nonce generation and consumption are enforced by the smart contract.

The blockchain and smart contract store blinded policy metadata, IPFS content identifiers, nonce states, and access logs. The smart contract handles policy registration, nonce generation, request verification, nonce consumption, and access-event recording. Blockchain and smart-contract-based access control support decentralization, transparency, and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

automated authorization [8], [9], [2]. Encrypted IoT data is stored in IPFS or an IPFS-compatible content-addressed storage system, while only the ciphertext identifier is stored on-chain.

B. *Adversary Model*

The adversary is assumed to be probabilistic polynomial-time and may observe all public blockchain records. Since blockchain data is transparent, the adversary can inspect registered metadata, transaction history, IPFS content identifiers, nonce states, and access logs. The adversary may also access encrypted ciphertext stored off-chain.

The adversary may attempt to infer sensitive access-policy information from on-chain metadata, recover attribute names or values from blinded policy components, replay a previously valid access response, impersonate another user, collude with unauthorized users, tamper with off-chain ciphertext, or analyze transaction records to infer access behavior. However, the adversary is not assumed to break standard cryptographic assumptions. In particular, the adversary cannot invert secure hash functions, forge valid signatures, recover encrypted salt vectors without the required key, or break the underlying CP-ABE scheme. The adversary also cannot modify finalized blockchain records or change smart contract state without valid blockchain transactions. Therefore, data confidentiality depends on the CP-ABE and policy-hiding CP-ABE foundations [10], [11], [6], [7].

C. *Security Goals*

The proposed framework aims to satisfy the following security goals.

SG1: Policy Confidentiality: The access policy should not reveal sensitive information to public blockchain observers. The proposed framework hides both attribute names and attribute values before storing policy metadata on-chain.

SG2: Attribute Privacy: User and policy attributes should not be exposed directly on the blockchain. Attribute-related information should appear only as blinded or hashed values, while salt values remain encrypted and off-chain.

SG3: Fine-Grained Access Control: Only users whose attribute set Ssatisfies the access policy (A, ρ)should be able to decrypt ciphertext CT. Unauthorized users should not obtain plaintext even if they access the ciphertext from IPFS.

SG4: Collusion Resistance: Multiple unauthorized users should not be able to combine their keys or attributes to decrypt data that none of them can access individually. This property is inherited from the underlying CP-ABE construction [10], [11].

SG5: Replay Resistance: An attacker should not be able to reuse an old proof or authorization response. Each access request must be linked to a fresh nonce, and each nonce must be consumed after one verification attempt.

SG6: Impersonation Resistance: An attacker should not be able to submit a valid response on behalf of another user without the required cryptographic credentials. The verification process binds the response to the requester address, resource identifier, and nonce.

SG7: Auditability and Traceability: Policy registration, nonce requests, verification attempts, and access events should be recorded on-chain, enabling transparent auditing and non-repudiation [8], [9], [2].

These goals guide the design of the proposed framework described in the next section.

V. PROPOSED SCHEME

The scheme extends the blockchain-based CP-ABE model of Yang et al. [1] by introducing hidden on-chain policy metadata, smart-contract-managed replay-resistant verification, and decentralized ciphertext storage using IPFS. CP-ABE enforces fine-grained data confidentiality, while the blockchain layer handles metadata registration, nonce management, replay prevention, and audit logging.

A. *System Architecture*

The proposed framework consists of IoT devices, an IoT gateway or data owner, an attribute authority, IPFS-based off-chain storage, a blockchain smart contract, and data users. IoT devices generate raw data such as health, environmental, device activity, or industrial monitoring records. Since IoT devices are usually resource-constrained, the gateway performs heavier operations such as data aggregation, encryption, policy blinding, and blockchain interaction. The overall architecture is shown in Fig. 1.

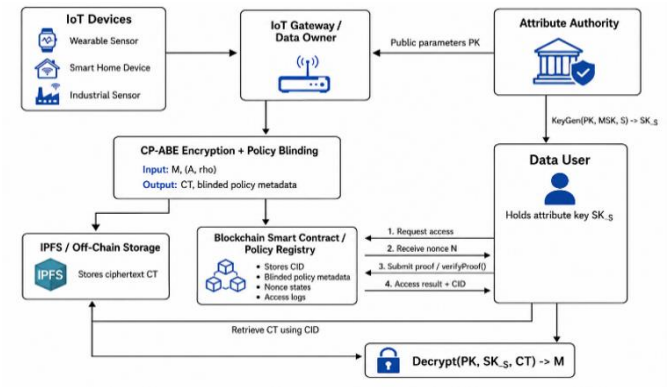

Fig. 1. . Proposed system architecture.

The data owner defines an access policy for each IoT data object using the LSSS notation (A, ρ), where $A$is the access matrix and ρmaps each row to a policy attribute. The IoT data is encrypted using CP-ABE under this policy, and the ciphertext $CT$is uploaded to IPFS or IPFS-compatible storage. Instead of storing ciphertext directly on-chain, the smart contract stores only the resource identifier, content identifier CID, blinded policy metadata, nonce states, and access-control records.

A data user requests access through the smart contract. The contract generates a fresh nonce and records its status on-chain. The user submits a response linked to the nonce, resource identifier, and requester address. If the response is valid and the nonce has not been consumed, the contract records the access attempt and allows retrieval of the corresponding CID. Actual decryption of CTremains controlled by CP-ABE: the user can recover plaintext Monly if SK_Ssatisfies the access policy.

B. *Full Policy Hiding Mechanism*

Yang et al. [1] use an access policy of the form (A, ρ, T_H), where $A$is the LSSS matrix, ρmaps matrix rows to attribute names, and T_H stores hashed attribute values. Although this hides attribute values, the attribute names represented by ρremain visible on-chain. Therefore, an observer may still infer the structure of the policy.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

To reduce this leakage, the proposed scheme hides both attribute names and attribute values before policy metadata is registered on-chain. For each policy attribute, the data owner generates a fresh random 256-bit salt r_i . The blinded representations are computed as:

$$blindName_i = H(attr\_name_i \parallel r_i)$$

$$blindValue_i = H(attr\_value_i \parallel r_i)$$

where H is a cryptographic hash function and $\parallel$ denotes concatenation. The pair $(blindName_i, blindValue_i)$ is stored on-chain instead of plaintext attribute information. Since r_i is generated freshly for each encryption, the same attribute used in different policies produces different blinded outputs.

The salt vector is represented as $R = \{r_1, r_2, \dots, r_l\}$, where l is the number of policy rows. The salt vector is not stored publicly on-chain; instead, it is stored off-chain after being encrypted under the data owner's key. The modified hidden policy metadata is represented as $P_{hidden} = (A, \rho', B)$, where A is the LSSS matrix, ρ' maps each row to a blinded attribute identifier, and B is the set of blinded attribute components. This preserves LSSS compatibility while preventing direct exposure of policy attributes on-chain. The scheme hides plaintext attribute names and values from public blockchain metadata, although policy size or matrix dimensions may still reveal limited structural information.

C. Smart Contract-Based Nonce Verification

Yang et al. [1] use an off-chain Attribute Management Node (AMN) for challenge-response verification. In contrast, the proposed scheme moves nonce generation, nonce-state management, and nonce consumption into a Solidity smart contract, making replay prevention enforceable on-chain. The nonce verification flow is illustrated in Fig. 2.

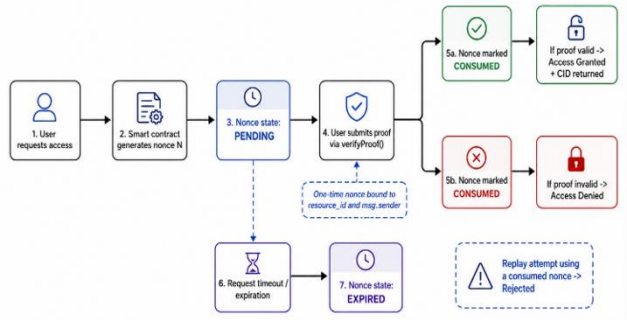

Fig. 2. On-Chain Nonce Verification

For each access request, the smart contract generates a fresh nonce:

$$N = keccak256(block.prevrandao \parallel resource\_id \parallel msg.sender \parallel timestamp)$$

where $resource\_id$ identifies the protected IoT data, $msg.sender$ is the blockchain address of the requester, and $timestamp$ represents the current block timestamp. The nonce is stored with status $PENDING$. The contract maintains three nonce states: $PENDING$, $CONSUMED$, and $EXPIRED$.

When the user submits a response through $verifyProof()$, the contract checks whether the nonce exists, whether its state is $PENDING$, and whether it is linked to the same requester and resource. After the first verification attempt, the nonce is marked as $CONSUMED$, regardless of whether the response is valid or invalid. This prevents the same nonce from being reused.

The smart contract does not perform full CP-ABE decryption on-chain because pairing-based cryptographic operations are expensive for Ethereum-compatible smart contracts. Instead, it enforces request freshness, response binding, replay rejection, and access logging. Final plaintext confidentiality remains enforced by the CP-ABE layer.

D. Access Control Protocol

The access control protocol consists of the following phases. First, the attribute authority initializes the CP-ABE system by running $(PK, MSK) \leftarrow Setup(\lambda)$, where PK is the public key and MSK is the master secret key. The public parameters are shared, while MSK is kept secret.

Second, for a legitimate data user with attribute set S, the authority generates $SK_S \leftarrow KeyGen(PK, MSK, S)$. The user stores SK_S locally and uses it during decryption. Third, the data owner or gateway defines an access policy (A, ρ), blinds each policy attribute name and value, and encrypts the IoT data using $CT \leftarrow Encrypt(PK, M, (A, \rho))$. Fourth, the ciphertext is uploaded to IPFS or IPFS-compatible storage using $CID = IPFS.store(CT)$.

Fifth, the data owner registers the protected resource on the smart contract by submitting the resource identifier, CID, and blinded policy metadata. Plain attribute names, values, and salts are never stored on-chain. Sixth, a data user requests access for a specific $resource\_id$. The smart contract generates a fresh nonce N, stores it as $PENDING$, and binds it to the requester and resource identifier.

Seventh, the user submits a response linked to N , $resource\_id$, and the requester address. The contract checks the nonce state and response binding, then marks the nonce as $CONSUMED$. Successful and failed attempts are recorded through blockchain events, and the nonce cannot be reused. Finally, after successful verification, the user retrieves the ciphertext from IPFS using the CID and runs $M \leftarrow Decrypt(PK, SK_S, CT)$. If S satisfies (A, ρ) , the plaintext is recovered; otherwise, decryption fails.

E. Informal Security Discussion

The proposed scheme improves policy confidentiality by replacing visible attribute names and values with blinded representations. Since each blinding operation uses a fresh random salt, the same attribute does not produce the same on-chain value across different encryptions. This reduces policy leakage and limits pattern analysis by blockchain observers. Attribute privacy is improved because plaintext attribute names, values, and salts are not stored on-chain.

Replay resistance is achieved through smart-contract-managed nonce states. Each nonce is linked to a requester and resource identifier and is consumed after the first verification attempt. Therefore, a previously captured response cannot be reused successfully. Impersonation resistance is supported by binding the response to the requester address, resource identifier, and nonce. Even if an attacker obtains the CID , plaintext cannot be recovered without a valid SK_S satisfying the CP-ABE access policy.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Auditability is achieved because policy registrations, nonce requests, verification attempts, and access events are recorded on-chain. Overall, the proposed scheme extends Yang et al. [1] by replacing partial policy hiding with hidden attribute-name and attribute-value metadata, replacing AMN-dependent replay handling with on-chain nonce management, and replacing centralized ciphertext storage with IPFS-based decentralized storage.

VI. IMPLEMENTATION

This section describes the implementation approach used to evaluate the proposed blockchain-based IoT access control framework. The implementation is divided into two parts: on-chain smart contract operations and off-chain cryptographic/storage operations. The blockchain layer handles policy registration, nonce management, verification logging, and access traceability, while CP-ABE encryption and decryption are performed off-chain.

A. Development Environment

The smart contract component is implemented using Solidity 0.8.x. Initial contract development and functional testing are performed using Remix IDE. For automated testing and gas measurement, Hardhat with ethers.js is used. The Hardhat local test environment simulates 10, 50, and 100 protected IoT resources and records gas consumption for major smart contract operations. The contract is also compatible with Ethereum-compatible test networks such as Sepolia.

The off-chain cryptographic component is simulated using Python 3.10. CP-ABE operations follow the standard ciphertext-policy attribute-based encryption model [6], [7]. IoT data is generated through software simulation instead of physical IoT hardware. For ciphertext storage, the implementation uses an IPFS-compatible local content-addressed storage model, where encrypted data is stored off-chain and represented by a content identifier CID.

B. Smart Contract Implementation

The smart contract is implemented as PolicyRegistry.sol. Its main purpose is to store blinded policy metadata, manage protected resource records, generate and consume nonce values, and record access events. The contract does not perform CP-ABE encryption or decryption on-chain because pairing-based cryptographic operations are computationally expensive for Ethereum-compatible smart contracts.

The main smart contract functions are as follows. The registerPolicy() function registers a protected IoT resource by storing the resource identifier, IPFS content identifier, and blinded policy metadata. Plain attribute names, attribute values, and salt values are not stored on-chain. The requestNonce() function generates a fresh nonce for a user requesting access to a specific resource. The nonce is linked to the requester address and resource identifier, and its status is stored as PENDING. The verifyProof() function checks the submitted response against the stored nonce and resource identifier. After the first verification attempt, the nonce is marked as CONSUMED to prevent replay. Successful and failed verification attempts are recorded through blockchain events. The getResourceCID() function returns the IPFS content identifier of the protected ciphertext after successful access verification.

This design keeps the blockchain layer lightweight. The blockchain is responsible for nonce freshness, response binding, replay rejection, and audit logging, while the final cryptographic access-control decision is enforced by CP-ABE during off-chain decryption.

C. Off-Chain CP-ABE and Storage

The CP-ABE component is executed off-chain by the data owner, gateway, and data user. Simulated IoT devices generate sample data records such as sensor readings, health parameters, or monitoring values. The data owner defines an access policy using the LSSS structure (A, ρ). The plaintext IoT data M is encrypted using the public key PK and access policy to generate ciphertext CT.

Before policy metadata is stored on-chain, attribute names and values are blinded using fresh random salts. Only the blinded policy components are sent to the smart contract. The salt vector is encrypted and stored off-chain, and it is never published on the blockchain. After encryption, CT is stored in IPFS or local content-addressed storage. The storage system returns a content identifier CID, which is registered on-chain. During access, an authorized user obtains CID, retrieves CT, and runs CP-ABE decryption using SK_S. If the user's attribute set S satisfies the access policy, plaintext M is recovered.

D. Device Scaling Scenarios and Assumptions

To evaluate scalability, the implementation considers three scenarios: 10, 50, and 100 simulated IoT resources. For each scenario, the system measures the gas cost of contract deployment, policy registration, nonce request, proof verification, replay rejection, and event-based access logging. These scenarios are used to analyze how the framework behaves as the number of protected resources increases.

The implementation is based on software simulation rather than physical IoT hardware. This is sufficient for evaluating the access-control workflow, smart contract behavior, and gas-cost overhead. The implementation does not claim to provide a production-ready IoT deployment. Instead, it demonstrates the feasibility of the proposed architecture and provides an experimental basis for analyzing blockchain deployment cost, replay-resistant verification, and privacy-preserving policy registration. Actual performance may vary depending on device capabilities, blockchain network conditions, IPFS availability, and key-management infrastructure.

VII. EVALUATION

This section presents the evaluation methodology and results for the proposed framework. The evaluation focuses on blockchain gas cost, deployment feasibility, and security comparison with reference schemes. Since CP-ABE encryption and decryption are performed off-chain, the blockchain evaluation is limited to smart contract operations such as policy registration, nonce request, proof verification, replay rejection, and event-based access logging.

A. Evaluation Setup

The smart contract is implemented using Solidity 0.8.x and evaluated using Hardhat with ethers.js. Initial functional testing is performed in Remix IDE, while automated gas measurement is performed in the Hardhat local test environment. The evaluation considers three simulated IoT scaling scenarios: 10, 50, and 100 protected IoT resources. Each resource is assumed to represent encrypted IoT data stored off-chain, while the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

blockchain stores only blinded policy metadata, content identifiers, nonce states, and access logs.

The evaluated operations include contract deployment, policy registration, nonce request, proof verification with access logging, and replay attempt rejection. Gas consumption is recorded for each operation to analyze whether the proposed on-chain policy registration and nonce verification process is practical for IoT access control.

B. Gas Cost Analysis

Table II shows the gas cost of the main smart contract operations. Contract deployment required 1,917,118 gas and remained constant because the same contract was deployed in each experiment. Policy registration consumed the highest runtime gas because it stores the resource identifier, IPFS content identifier, and blinded policy metadata on-chain. The average policy registration cost increased from 229,936 gas in the 10-resource scenario to 252,602 gas in the 100-resource scenario.

Nonce request required approximately 121,000–123,000 gas, while proof verification with access logging required about 69,700 gas. Replay attempts using a previously consumed nonce were rejected with about 31,700 gas, confirming that nonce reuse is blocked through smart contract state management. Since CP-ABE operations are off-chain, gas consumption is limited to metadata management, nonce handling, and audit logging rather than expensive pairing-based cryptographic computation.

TABLE II. GAS COST OF SMART CONTRACT OPERATIONS

Operation	*10 Devices*	*50 Devices*	*100 Devices*
Contract deployment	1,917,118	1,917,118	1,917,118
Avg. policy registration	229,936	248,064	252,602
Avg. nonce request	123,363	121,994	121,822
Avg. proof verification and access logging	69,712	69,712	69,711
Replay attempt rejection	31,762	31,750	31,762

C. Blockchain Cost Estimation

To estimate economic feasibility, gas usage is converted into transaction cost. When the gas price is expressed in gwei, the estimated cost is calculated as:

$$Cost_{token} = GasUsed \times GasPrice \times 10^{-9}$$

$$Cost_{USD} = Cost_{token} \times TokenPrice$$

Table III presents indicative transaction costs using the 100-resource average gas values. The assumptions are: Ethereum mainnet at 20 gwei and ETH at $3,500; Polygon at 30 gwei and token price $0.40; and Optimism and Base using a simplified execution-only estimate of 0.001 gwei and ETH at $3,500. These estimates show that Ethereum mainnet is significantly more expensive than low-cost Ethereum-compatible networks. Therefore, frequent IoT access control operations are more practical on Polygon, Optimism, Base, or similar Layer 2 networks. The values are indicative and may vary with gas price, token price, congestion, and Layer 2 data availability fees.

TABLE III. ESTIMATED DEPLOYMENT COST ACROSS BLOCKCHAIN NETWORKS

Network	*Deployment Cost*	*Policy Registration*	*Nonce Request*	*Proof Verification*
Ethereum Mainnet	$134.20	$17.68	$8.53	$4.88
Polygon	$0.0230	$0.0030	$0.0015	$0.0008
Optimism	$0.0067	$0.0009	$0.0004	$0.0002
Base	$0.0067	$0.0009	$0.0004	$0.0002

D. Security and Functional Comparison

Table IV compares the proposed framework with Lai et al. [10], [11] and Yang et al. [1]. Lai et al. provide strong cryptographic foundations for CP-ABE and policy hiding, but they do not address blockchain traceability, decentralized storage, smart contract nonce verification, or gas-cost analysis. Yang et al. combine CP-ABE and blockchain for IoT data protection, but their on-chain policy hiding is partial because attribute names remain visible. Their challenge-response process also depends on an off-chain AMN, and their evaluation does not include Solidity gas measurement.

The proposed framework addresses these gaps by hiding both attribute names and values in on-chain metadata, enforcing nonce consumption through smart contract state, storing ciphertext off-chain using IPFS, and evaluating gas cost across multiple scaling scenarios. The comparison does not claim a new CP-ABE cryptographic proof; instead, the contribution is a system-level improvement over blockchain-based CP-ABE deployment.

TABLE IV. SECURITY AND FUNCTIONAL COMPARISON

Property	*Lai et al. [10], [11]*	*Yang et al. [1]*	*Proposed Scheme*
Fine-grained access control	Yes	Yes	Yes
LSSS-based expressiveness	Yes	Yes	Yes
Policy hiding	Yes	Partial	Attribute name + value hidden on-chain
Attribute name hiding	No	No	Yes
Blockchain traceability	No	Yes	Yes
Decentralized ciphertext storage	No	No	Yes
Replay resistance	No	Off-chain	On-chain
Smart contract verification	No	No	Yes
Gas cost analysis	No	No	Yes

E. Discussion of Results

The results show that operations involving blockchain storage consume more gas than operations that update smaller state variables. Policy registration is the most expensive runtime operation because it stores metadata for each protected resource. Proof verification is cheaper because it mainly checks nonce state, verifies request binding, updates access status, and emits events.

Using IPFS reduces the amount of data stored directly on-chain. This is important because storing full IoT ciphertexts on Ethereum-compatible blockchains would be costly and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

inefficient. By storing only the CID, blinded metadata, and access records on-chain, the framework reduces storage overhead while preserving traceability.

Overall, the evaluation indicates that the proposed scheme improves policy privacy and replay resistance while keeping blockchain operations limited to metadata registration, nonce management, and audit logging.

VIII. DISCUSSION

This section discusses the practical relevance, advantages, limitations, and future extensions of the proposed blockchain-based IoT access control framework.

A. Deployment Scenarios

The proposed framework is suitable for IoT environments where data sensitivity and access-policy privacy are important. In smart healthcare, wearable devices and medical sensors may collect heart rate, blood pressure, glucose level, and patient activity data. In such systems, not only the data but also the access policy may reveal sensitive information, such as medical role, department, or treatment context [3]. The framework can also be applied to smart home systems, where access to camera feeds, door locks, energy records, and device logs must be restricted to authorized users. In industrial, campus, and enterprise IoT systems, access policies may reveal operational roles, machine categories, laboratory permissions, or internal monitoring zones. Hiding policy metadata and recording access events on-chain can improve privacy and auditability in these settings.

B. Advantages

The proposed framework improves blockchain-based CP-ABE access control in four main ways. First, it hides both attribute names and attribute values before storing policy metadata on-chain, reducing leakage compared with partial policy hiding in Yang et al. [1]. Second, replay resistance is strengthened by moving nonce generation and nonce consumption into the smart contract. Since each nonce is stored and consumed on-chain, an old response cannot be reused in a later transaction. Third, IPFS-based off-chain storage avoids storing large ciphertexts directly on the blockchain and reduces dependence on a centralized cloud-only model. Fourth, the framework includes gas-cost analysis, which is important for judging deployment feasibility on Ethereum-compatible networks.

C. Limitations

The proposed framework has some limitations. First, it does not introduce a new formal CP-ABE security proof. The encryption-layer security is inherited from existing CP-ABE and policy-hiding CP-ABE constructions [10], [11], [6], [7], while the proposed contribution is mainly system-level. Second, CP-ABE operations may be expensive for highly constrained IoT devices, so the framework assumes that encryption and decryption are handled by an IoT gateway or edge device. Third, the implementation uses software simulation and Hardhat-based testing rather than physical IoT hardware. Real deployments may introduce device failures, network delay, key-distribution complexity, and storage availability issues. Fourth, gas cost depends on blockchain network conditions, so Layer 2 networks may be more suitable than Ethereum mainnet for frequent access transactions. Finally, salt and key management must be handled carefully because loss of encrypted salt data may affect policy management or recovery.

D. Future Work

Future work can test the framework on real IoT hardware such as Raspberry Pi or ESP32 devices, optimize the smart contract to reduce gas cost, and evaluate deployment on Layer 2 networks such as Polygon, Optimism, or Base. A formal security proof can also be developed for the policy-hiding extension and smart-contract-based verification model. Future versions may integrate decentralized identity, decentralized key management, or zero-knowledge proofs to reduce dependence on a single authority and further improve privacy-preserving authorization.

IX. CONCLUSION

This paper proposed a privacy-preserving blockchain-based access control framework for IoT systems using ciphertext-policy attribute-based encryption. The proposed scheme addresses key limitations in existing blockchain-based CP-ABE access control models, particularly the partial on-chain policy hiding, off-chain replay handling, centralized ciphertext storage, and lack of blockchain deployment cost analysis in Yang et al. [1].

The framework improves policy confidentiality by hiding both attribute names and attribute values before storing policy metadata on-chain. It also introduces a smart-contract-based nonce verification mechanism in which nonce generation, binding, and consumption are enforced directly by blockchain logic. This provides replay-resistant access verification without relying on an off-chain Attribute Management Node for nonce management. In addition, encrypted IoT data is stored using IPFS-based off-chain storage, while only the content identifier and blinded metadata are recorded on-chain.

The evaluation includes Hardhat-based gas measurement for major smart contract operations across different IoT scaling scenarios. Overall, the proposed scheme improves policy privacy, replay resistance, auditability, and deployment awareness for blockchain-based IoT access control. Future work may include real IoT hardware testing, formal security proof development, gas optimization, and Layer 2 deployment.

REFERENCES

[1] Z. Yang, X. Chen, Y. He, L. Liu, Y. Che, X. Wang, K. Xiao, and G. Xu, “An attribute-based access control scheme using blockchain technology for IoT data protection,” High-Confidence Computing, vol. 4, no. 2, Art. no. 100199, 2024, doi: 10.1016/j.hcc.2024.100199.

[2] O. Novo, “Blockchain meets IoT: An architecture for scalable access management in IoT,” IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1184–1195, 2018.

[3] Y. Zhang, D. Zheng, and R. H. Deng, “Security and privacy in smart health: Efficient policy-hiding attribute-based access control,” IEEE Internet of Things Journal, vol. 5, no. 3, pp. 2130–2145, 2018.

[4] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” in Proc. EUROCRYPT, 2005, pp. 457–473.

[5] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in Proc. ACM CCS, 2006, pp. 89–98.

[6] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in Proc. IEEE Symposium on Security and Privacy, 2007, pp. 321–334.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

[7] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” in Proc. Public Key Cryptography, Springer, 2011, pp. 53–70.

[8] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[9] J. P. Cruz, Y. Kaji, and N. Yanai, “RBAC-SC: Role-based access control using smart contract,” IEEE Access, vol. 6, pp. 12240–12251, 2018.

[10] J. Lai, R. H. Deng, and Y. Li, “Expressive CP-ABE with partially hidden access structures,” in *Proc. 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012, pp. 18–19.

[11] J. Lai, R. H. Deng, and Y. Li, “Fully secure ciphertext-policy hiding CP-ABE,” in *Information Security Practice and Experience*, LNCS, vol. 6672, Springer, 2011, pp. 24–39.

[12] S. Gao, G. Piao, J. Zhu, X. Ma, and J. Ma, “TrustAccess: A trustworthy secure ciphertext-policy and attribute hiding access control scheme based on blockchain,” IEEE Transactions on Vehicular Technology, vol. 69, no. 6, pp. 5784–5798, 2020.