A Blockchain-Based Access Control Scheme with Hidden Attributes and Policies Using Commitments and Zero-Knowledge Proofs

Yuanshao Liang, Hui Li, *Senior Member, IEEE*, Wenhui Hu, Wu Zhou, Baocheng Yan, Kedan Li, Naixing Wu, and Kaili Shao

***Abstract*—Blockchain provides public verifiability for access control in Internet of Things (IoT) data sharing, but its transparency can expose requester attributes, access policies, and their satisfaction relations. Existing privacy-preserving approaches often protect only one side of the authorization decision or rely on pairing-based ABE/HVE, interactive homomorphic computation, or circuit-specific proof parameters. This paper proposes a blockchain-based access control scheme that verifies authorization over hidden, authenticated inputs using commitments and non-interactive zero-knowledge proofs. Requester attributes and access policies are encoded as vectors. Multiple attribute authorities sign requester-bound local commitments, while the data provider independently commits to the policy vector. The requester proves that the committed vectors satisfy the required inner-product relation through a linked commit-and-prove protocol using a Bulletproof inner-product argument. A smart contract verifies the authority evidence, commitment aggregation, and access proof while learning only the authorization result. We formally establish policy hiding, attribute hiding, and authenticated attribute integrity. The recursive inner-product argument makes the dominant per-request access-verification communication grow logarithmically with the vector dimension. We implement a Rust experimental prototype and make the source code, benchmark configurations, results, and plotting scripts publicly available. Experiments over vector dimensions from 16 to 512 quantify the communication and computation costs. At the largest evaluated dimension, the modeled access-verification record remains below 1 KB, while final decryption takes 0.103 ms. These results indicate that the scheme supports bilateral authorization privacy and public verifiability without pairing-based authorization or circuit-specific trusted setup.**

***Index Terms*—Access control, blockchain, access verification, Bulletproofs, vector commitments.**

I. Introduction

IOT devices continuously generate data that are outsourced to cloud servers for storage, sharing, and analysis. These data often contain sensitive medical, behavioral, or location information and therefore require fine-grained access control. Ciphertext-policy attribute-based encryption (CP-ABE) allows a data provider to associate an access policy with encrypted data so that only users with satisfying attributes can decrypt it [2]. Blockchain further provides a tamper-resistant and publicly auditable environment for recording policy publication, access requests, and authorization results [4]. However, the same transparency exposes authorization records to ledger observers and may reveal access policies, requester attributes, and their relationships [5].

Existing studies protect authorization privacy through several representative routes. Early policy-hiding ABE schemes conceal sensitive policy values or the access structure attached to a ciphertext [10], [11]. TrustAccess extends policy and attribute hiding to blockchain-based access control [5]. EDDAC preserves requester-attribute privacy and performs blockchain-assisted access testing through an inner-product predicate [7]. Proof-based systems such as HAPPS allow an authorization statement to be verified without revealing the underlying requester attributes [6].

The remaining challenge is not only to hide policies and attributes, but also to convince a public verifier that the hidden attributes are authentic and satisfy the hidden policy. CP-ABE- and HVE-based solutions usually embed this judgment in pairing-based ciphertexts, transformation keys, or predicate materials [15]. Homomorphic approaches require ciphertext-domain computation, interaction among blockchain nodes, and verifiable partial decryption [8]. General-purpose proof systems can express authorization logic, but they require a circuit representation and may rely on circuit-specific parameters [16]. Hence, a desirable construction should bind multiauthority-certified attributes to a hidden policy and prove their satisfaction relation with concise public verification and no circuit-specific trusted setup.

Our key observation. Requester attributes and access policies can both be represented as fixed-dimensional vectors, and policy satisfaction can be reduced to a constrained inner-product relation. The key challenge is to prove this relation while ensuring that the attribute vector is authority-certified, the policy vector is the one published by the data provider, and neither vector is exposed. To achieve this binding, multiple authorities authenticate local attribute commitments, the data

Yuanshao Liang and Hui Li are with the Shenzhen Graduate School, Peking University, Shenzhen 518055, China (e-mail: lys@stu.pku.edu.cn; lih64@pkusz.edu.cn).

Wenhui Hu and Wu Zhou are with China Mobile Internet Co., Ltd., Guangzhou 510000, China (e-mail: huwenhui@cmic.chinamobile.com; zhouwu@cmic.chinamobile.com).

Baocheng Yan is with China Telecom Digital Intelligence Technology Co., Ltd., Beijing 100036, China (e-mail: yanbc@chinatelecom.cn).

Kedan Li is with the School of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL 61801, USA (e-mail: kedanli2@illinois.edu).

Naixing Wu is with China Unicom Shenzhen Branch, Shenzhen 518046, China (e-mail: wunx1@chinaunicom.cn).

Kaili Shao is with Huanghe Science and Technology University, Zhengzhou 450000, China (e-mail: sklemai@163.com).

Corresponding authors: Hui Li, Kaili Shao, and Naixing Wu.

provider publishes an independent policy commitment, and the requester proves the relation between their openings. Vector commitments provide hiding and binding [17], while a linked commit-and-prove protocol uses a recursive Bulletproof inner-product argument to compress the dominant per-request access-verification material to $O(\log m)$ for vector dimension m, without a circuit-specific trusted setup [18].

In the resulting scheme, each attribute authority signs a requester-bound local commitment, and the blockchain verifies both the authority signatures and the commitment aggregation. The data provider publishes only a commitment to the policy together with the encrypted-data index, leaving the plaintext policy off-chain. The requester submits a non-interactive zero-knowledge access proof, and the smart contract verifies only the public commitments, authority evidence, and proof before recording the authorization result. After acceptance, the data provider releases the symmetric key through an authenticated channel, and the requester checks and decrypts the ciphertext. The construction provides policy hiding, attribute hiding, and authenticated attribute integrity, avoids pairing-based authorization and circuit-specific trusted setup, and keeps the dominant access-proof material logarithmic in the vector dimension.

The main contributions of this paper are summarized as follows:

1) We propose a blockchain-based access control scheme that provides attribute hiding, policy hiding, decentralized authorization, and public verification of access eligibility.
2) We construct a linked non-interactive zero-knowledge argument for committed policy matching. Under a single witness assignment, it proves that the openings of the certified attribute commitment and the published policy commitment satisfy the required inner-product relation without revealing either committed vector.
3) We prove that the scheme satisfies policy hiding, attribute hiding, and authenticated attribute integrity, and evaluate its communication and computation costs. Theoretical analysis and prototype experiments show that the scheme requires neither pairing-based verification nor circuit-specific trusted setup. Through recursive inner-product compression, the communication cost of each access verification is $O(\log m)$, while the final data-decryption cost is independent of the attribute-vector dimension.

The remainder of this paper is organized as follows. Section II reviews the related work, and Section III presents the necessary preliminaries. Section IV describes the system and security models. Section V details the proposed construction. Section VI presents the security analysis, and Section VII gives the theoretical and experimental evaluation. Finally, Section VIII concludes this paper.

II. Related Work

Attribute-based access control and attribute-based encryption provide fine-grained protection for outsourced data, while blockchain improves decentralization, traceability, and public auditability for data sharing systems [1]–[3], [12]. The same transparency, however, also exposes authorization metadata. Access policies may reveal the data owner's sharing intention, requester attributes may disclose identity-related or context-sensitive information, and on-chain access records can make authorization behavior linkable. Therefore, the key issue in blockchain-based access control is no longer only whether the shared data are encrypted, but whether the authorization process itself leaks policies, attributes, or satisfaction relations.

Policy-hiding ABE reduces the leakage from the policy side. Nishide *et al.* [10] proposed partially hidden access structures, and Lai *et al.* [11] strengthened ciphertext-policy hiding under a fully secure construction. Blockchain-assisted schemes further combine policy privacy with auditable access control: TrustAccess [5] introduces ciphertext-policy hiding into trustworthy access control, and Zhang *et al.* [15] support credible verification for fully policy-hidden CP-ABE. Later works extend this line to dynamic IoMT sharing [13], distributed healthcare data sharing [9], searchable or proxy-assisted access [14], and multigroup cloud storage [20]. These studies protect policy contents and improve application flexibility, but they mainly organize authorization around ABE ciphertexts, user keys, search tokens, or outsourced decryption components. More importantly, hiding the policy alone does not naturally hide requester attributes, attribute-authentication sources, or the concrete relation by which a request satisfies the policy.

Several recent schemes move closer to private authorization by protecting both sides of the access decision or by making the decision verifiable under hidden inputs. TrustAccess [5] considers both ciphertext-policy and attribute hiding, but the access judgment is still embedded in a CP-ABE and ElGamal-based verification workflow, so attribute certification and authorization remain coupled with the ABE key structure. Zhang *et al.* [15] use fully policy-hidden CP-ABE and hidden-vector encryption to achieve credible access verification, but the design still depends on pairing-based CP-ABE, HVE transformation, and corresponding key materials. EDDAC [7] protects attributes through chameleon hashing and performs on-chain access testing with an inner-product predicate, whereas the authorization procedure remains bound to ABE-related key fragments, predicate materials, and decryption testing. Wu *et al.* [8] encrypt both policies and attributes with an additive homomorphic cryptosystem and derive the authorization result through multiple blockchain nodes, but this requires homomorphic ciphertext computation, node interaction, and verifiable partial decryption. HAPPS [6] and zk-Guard [16] use zero-knowledge techniques to hide authorization witnesses and enable public verification, yet such schemes typically require proof parameters or a general circuit representation of the access relation; some constructions also introduce trusted or circuit-specific setup assumptions. Taken together, existing schemes can support policy hiding, attribute hiding, or privacy-preserving authorization verification, but their authorization processes are usually attached to ABE key structures, pairing operations, homomorphic partial decryption, multi-node interaction, or general-purpose proof circuits, and they do not provide a unified binding to multiauthority-certified requester attributes.

These works form the foundation for privacy-preserving blockchain access control, but they leave a structural gap: existing schemes do not simultaneously provide policy hiding, requester-attribute hiding, binding to multiauthority-certified attributes, and publicly verifiable authorization over hidden inputs without circuit-specific trusted setup or centralized attribute-key distribution. This paper addresses this gap by authenticating local attribute commitments through multiple authorities, committing to the policy independently, and proving the satisfaction relation between the two hidden vectors with a Bulletproof inner-product argument [18]. This design separates attribute authenticity from policy satisfaction, keeps both vectors hidden from blockchain nodes, the cloud storage center, and public observers, avoids pairing-based authorization and circuit-specific setup, and enables public verification with concise proof material while revealing only the authorization result.

III. Preliminaries

In this section, we introduce the cryptographic primitives and the policy-vector representation used in our construction. Following the style of protocol-oriented constructions, each primitive is specified by its interface and only the property needed later is recalled.

A. Vector Commitment for Attributes and Policies

The proposed scheme uses a multi-generator Pedersen commitment to hide attribute and policy vectors while binding each commitment to a unique opening [21]. Let $\mathbb{G}$ be a cyclic group of prime order q in which the discrete-logarithm problem is hard. The commitment scheme COM is instantiated as follows.

1) $\mathsf{COM.Setup}(1^\lambda, m) \to pp_{\mathsf{com}}$*:* On input a security parameter 1^λ and a vector dimension m, it selects a group $\mathbb{G}$ of prime order q and derives $m+1$ generators $G_1, \ldots, G_m, B \in \mathbb{G}$ whose discrete-logarithm relations are unknown. It outputs $pp_{\mathsf{com}} = (q, \mathbb{G}, \mathbf{G}, B)$, where $\mathbf{G} = (G_1, \ldots, G_m)$.

2) $\mathsf{COM.Commit}(pp_{\mathsf{com}}, \mathbf{v}; r) \to C$*:* On input pp_{com}, a vector $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{Z}_q^m$, and randomness $r \leftarrow_R \mathbb{Z}_q$, it outputs $C = B^r \prod_{j=1}^m G_j^{v_j}$.

3) $\mathsf{COM.Open}(pp_{\mathsf{com}}, C, \mathbf{v}, r) \to b$*:* On input a commitment C and a claimed opening $(\mathbf{v}, r)$, it recomputes $C' = B^r \prod_{j=1}^m G_j^{v_j}$ and outputs $b = 1$ if $C' = C$; otherwise, it outputs $b = 0$.

For a local vector managed by AA_i, let $I_i = \{\ell_i + 1, \ldots, \ell_i + m_i\}$ be its public generator interval. We use $\mathsf{COM.Commit}_{I_i}(pp_{\mathsf{com}}, \mathbf{v}^{(i)}; r)$ and $\mathsf{COM.Open}_{I_i}$ to denote the same algorithms restricted to the generators indexed by I_i. This notation does not introduce a new commitment primitive; it only makes the local-to-global aggregation used later explicit.

The resulting commitment is perfectly hiding and computationally binding under the discrete-logarithm assumption. In the construction, the attribute and policy commitments are denoted by $C_x = \mathsf{COM.Commit}(pp_{\mathsf{com}}, \mathbf{x}; r_x)$ and $C_y = \mathsf{COM.Commit}(pp_{\mathsf{com}}, \mathbf{y}; r_y)$, respectively.

B. Policy and Attribute Vectorization

The vectorization maps policy values to polynomial roots and maps requester attributes to evaluations of the corresponding monomial basis. We first illustrate this idea with a two-attribute example and then give its general form.

Example. Suppose the system supports two attributes att_1 and att_2 with value domains $V_1 = \{v_{1,1}, v_{1,2}\}$ and $V_2 = \{v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}\}$. Consider the conjunctive policy $P = [P_1, P_2]$, where $P_1 = \{v_{1,1}\}$ and $P_2 = \{v_{2,1}, v_{2,3}\}$. To obtain fixed-degree polynomials, the data provider samples dummy roots $r_1 \leftarrow_R \mathbb{Z}_q \setminus V_1$ and $r_2, r_3 \leftarrow_R \mathbb{Z}_q \setminus V_2$, and defines

$$\begin{aligned} f_1(z) &= (z - v_{1,1})(z - r_1) = c_0 + c_1 z + c_2 z^2, \\ f_2(z) &= (z - v_{2,1})(z - v_{2,3})(z - r_2)(z - r_3) \\ &= d_0 + d_1 z + d_2 z^2 + d_3 z^3 + d_4 z^4. \end{aligned}$$

It further samples independent nonzero weights $\rho_1, \rho_2 \leftarrow_R \mathbb{Z}_q^*$ and forms the policy vector

$$\mathbf{y} = (\rho_1 c_0, \rho_1 c_1, \rho_1 c_2, \rho_2 d_0, \rho_2 d_1, \rho_2 d_2, \rho_2 d_3, \rho_2 d_4).$$

For a requester whose attribute list is $L = [L_1, L_2]$, the corresponding attribute vector is

$$\mathbf{x} = (1, L_1, L_1^2, 1, L_2, L_2^2, L_2^3, L_2^4).$$

Consequently, $\langle \mathbf{x}, \mathbf{y} \rangle = \rho_1 f_1(L_1) + \rho_2 f_2(L_2)$. For example, $L = [v_{1,1}, v_{2,3}]$ satisfies P because both polynomial evaluations are zero, and hence $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.

We now formalize the transformation. The public parameters fix the attribute order $(att_1, \ldots, att_N)$ and polynomial degrees $(n_1, \ldots, n_N)$. Hence, the ordered monomial blocks $(1, z, \ldots, z^{n_i})$ are determined implicitly and need not be transmitted with each policy. Let $P = (P_1, \ldots, P_N)$ be a conjunctive policy, where $P_i \subseteq V_i$ contains the accepted values of att_i. For each attribute, $\mathsf{Vec.Policy}$ pads P_i with distinct dummy roots sampled outside V_i until it contains n_i roots. The resulting polynomial is

$$f_i(z) = \prod_{a \in P_i} (z - a) \prod_{r \in R_i} (z - r) = \sum_{k=0}^{n_i} a_{i,k} z^k,$$

where R_i is the set of dummy roots and $|P_i| + |R_i| = n_i$.

1) $\mathsf{Vec.Policy}(P) \to \mathbf{y}$*:* It samples $\rho_i \leftarrow_R \mathbb{Z}_q^*$ and outputs

$$\mathbf{y} = (\rho_1 a_{1,0}, \ldots, \rho_1 a_{1,n_1}, \ldots, \rho_N a_{N,0}, \ldots, \rho_N a_{N,n_N}).$$

The dummy roots and weights are internal randomness used to generate $\mathbf{y}$ and therefore are omitted from the algorithm output.

2) $\mathsf{Vec.Attr}(\mathbf{u}) \to \mathbf{x}$*:* For an attribute list $\mathbf{u} = (L_1, \ldots, L_N)$, it evaluates the predetermined monomial blocks and outputs

$$\mathbf{x} = (1, L_1, \ldots, L_1^{n_1}, \ldots, 1, L_N, \ldots, L_N^{n_N}).$$

For an authority AA_i, $\mathsf{Vec.Attr}_i(L_i) \to \mathbf{x}^{(i)} = (1, L_i, \ldots, L_i^{n_i})$ denotes the corresponding local block, whose dimension is $m_i = n_i + 1$. The global vector is the ordered concatenation $\mathbf{x} = \mathbf{x}^{(1)} \| \cdots \| \mathbf{x}^{(N)}$.

3) $\mathsf{Vec.Match}(\mathbf{x}, \mathbf{y}) \to b$*:* It computes $t = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^N \rho_i f_i(L_i)$ and outputs $b = 1$ if $t = 0$; otherwise, it

outputs $b = 0$. If every $L_i \in P_i$, then every $f_i(L_i) = 0$ and the inner product is zero. If at least one policy clause is not satisfied, independently sampled weights prevent nonzero polynomial evaluations from systematically cancelling; for fixed evaluations, an erroneous zero occurs with probability at most $1/(q-1)$.

C. Zero-Knowledge Proof

Zero-knowledge proofs enable a prover to convince a verifier that a relation is satisfied without revealing the private witness. A public-coin Sigma protocol can be transformed into a non-interactive proof by the Fiat-Shamir heuristic [22]. In this paper, the access proof is finally instantiated as a non-interactive zero-knowledge proof over committed policy and attribute vectors. We recall only the proof tools that are used in the construction; the concrete relations are given in Section V-B.

1) $\mathsf{NIZK.Setup}(1^\lambda, m, pp_{\mathsf{com}}) \to pp_{\mathsf{nizk}}$*:* On input the security parameter, vector dimension, and commitment parameters, it derives the additional independent generators and Fiat–Shamir domain parameters required by the committed inner-product proof and outputs pp_{nizk}.

2) $\mathsf{NIZK.Prove}(pp_{\mathsf{nizk}}, stmt, w) \to \pi$*:* On input a public statement $stmt$ and a private witness w such that $(stmt, w) \in \mathcal{R}$, it outputs a proof π.

3) $\mathsf{NIZK.Verify}(pp_{\mathsf{nizk}}, stmt, \pi) \to b$*:* On input $stmt$ and π, it outputs $b \in \{0, 1\}$.

The proof system is required to satisfy completeness, knowledge soundness, and zero knowledge.

Representation proofs. A representation proof is a Sigma-protocol-style proof of knowledge of the exponents in a Pedersen-type commitment or representation. It allows the prover to show that a public commitment is honestly formed with respect to some hidden vector and randomness, without revealing either value. This technique follows the standard discrete-logarithm proof-of-knowledge paradigm underlying Schnorr-type protocols [23] and Pedersen commitments [21]. In our construction, these representation relations bind the attribute and policy commitments to the hidden vectors used by the committed inner-product proof.

Inner-product arguments. Inner-product arguments allow a prover to prove that two hidden vectors satisfy a claimed inner-product relation. Bootle et al. [19] gave efficient zero-knowledge arguments for arithmetic relations in the discrete-logarithm setting, and Bulletproofs [18] further provide short non-interactive arguments with logarithmic proof size for vector relations. In our construction, the access policy and the requester attributes are first vectorized, and access eligibility is reduced to a hidden inner-product matching relation. The corresponding proof is instantiated in Section V-B.

D. Signature and Symmetric Encryption Instantiations

Schnorr signature. We adopt the Schnorr signature scheme proposed by Schnorr [23]. It is instantiated as follows.

1) $\mathsf{SIG.Setup}(1^\lambda) \to pp_{\mathsf{sig}}$*:* Select a cyclic group $\mathbb{G}$ of prime order q, a generator $g \in \mathbb{G}$, and a hash function $H_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_q$. Output $pp_{\mathsf{sig}} = (q, \mathbb{G}, g, H_{\mathsf{sig}})$.

2) $\mathsf{SIG.KeyGen}(pp_{\mathsf{sig}}) \to (pk, sk)$*:* Choose $sk \leftarrow_R \mathbb{Z}_q$ and compute $pk = g^{sk}$. Output (pk, sk).

3) $\mathsf{SIG.Sign}(sk, m) \to \sigma$*:* Choose $k \leftarrow_R \mathbb{Z}_q$, compute $R = g^k$, $e = H_{\mathsf{sig}}(m \| R)$, and $s = k + e \cdot sk \bmod q$. Output $\sigma = (R, s)$.

4) $\mathsf{SIG.Verify}(pk, m, \sigma) \to b$*:* Parse $\sigma = (R, s)$, compute $e = H_{\mathsf{sig}}(m \| R)$, and output $b = 1$ if $g^s = R \cdot pk^e$; otherwise, output $b = 0$.

AES-GCM encryption. We instantiate symmetric encryption with AES-GCM, combining AES [24] with the Galois/Counter Mode specified in [25]. Let E_K denote AES under key K, GHASH denote polynomial hashing in $\mathbb{F}_{2^{128}}$, and J_0 denote the counter block derived from a nonce iv.

1) $\mathsf{SE.KeyGen}(1^\lambda) \to K$*:* Choose and output a symmetric key $K \leftarrow_R \{0,1\}^\kappa$.

2) $\mathsf{SE.Enc}(K, M) \to C_M$*:* Choose a fresh nonce iv, set $H_g = E_K(0^{128})$, encrypt M in counter mode to obtain C, and compute the authentication tag $T = E_K(J_0) \oplus \mathsf{GHASH}_{H_g}(C)$. Output $C_M = (iv, C, T)$.

3) $\mathsf{SE.Dec}(K, C_M) \to M/\perp$*:* Parse $C_M = (iv, C, T)$, recompute the tag T', and output $\perp$ if $T' \neq T$; otherwise, decrypt C in counter mode and output M.

IV. Problem Statement

In this section, we present the problem statement of the proposed scheme, including the system model, data structures, and security model. The system model describes the entities involved in the proposed blockchain-based access control scenario and their interactions. The data structures clarify the public records used by the protocol. The security model specifies the security goals considered in this paper.

A. System Model

This paper considers a blockchain-assisted data sharing scenario in which a data provider releases encrypted data only to requesters satisfying a specified access policy. Since directly exposing either user attributes or access policies may reveal sensitive information, the system represents both attributes and policies in a hidden form and verifies access eligibility through a zero-knowledge proof. The main role of the blockchain is to record commitments and execute public proof verification, while plaintext attributes, plaintext policies, and plaintext data are kept off-chain.

As shown in Fig. 1, the system model consists of five types of entities: multiple attribute authorities ($\{AA_i\}_{i=1}^N$), the data provider (DP), the data requester (DR), the cloud storage center (CSC), and the blockchain (BC).

1) Attribute Authorities $(AA_i)_{i=1}^N$*:* Attribute authorities are responsible for authenticating user attributes. Each authority manages one attribute domain and provides authentication evidence for the corresponding part of the requester's attributes. The system may contain multiple authorities so that no single authority needs to learn the requester's complete attribute profile.

2) Data Provider DP: DP owns the data to be shared and defines the corresponding access policy. Before publishing the data item, DP stores only encrypted data outside the

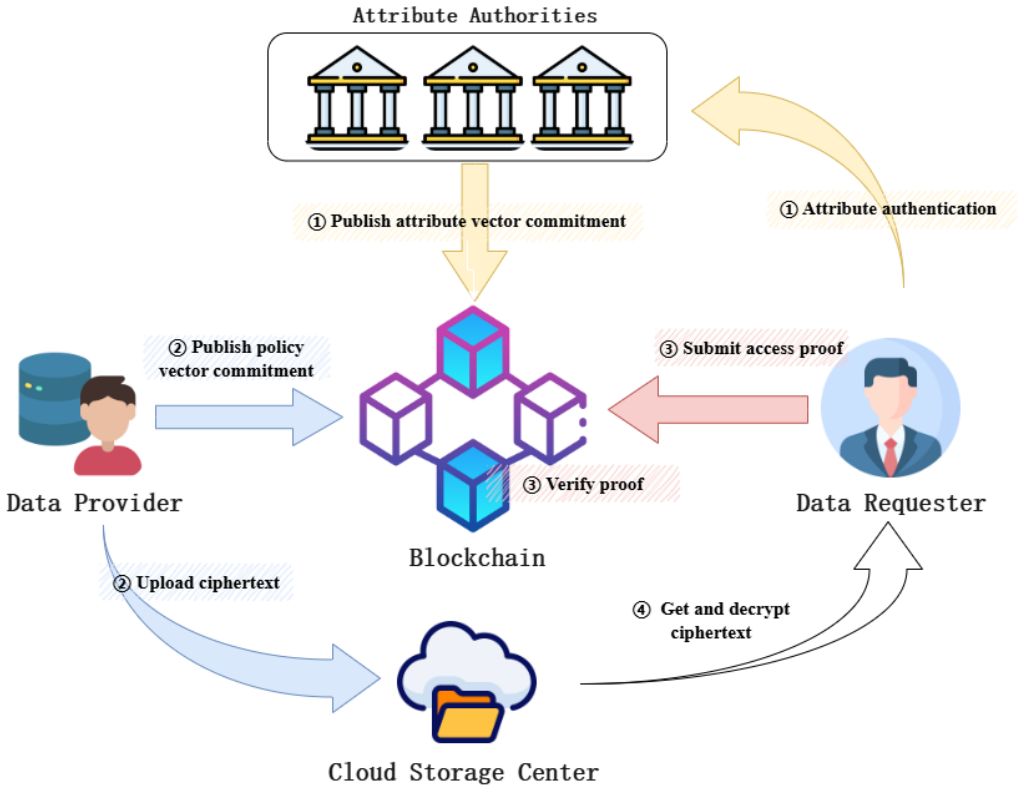

Fig. 1. System model of the proposed scheme.

blockchain and publishes a hidden policy representation to BC. Thus, the policy can be used for access verification without being directly exposed on-chain.

3) Data Requester DR: DR is the user that requests access to an encrypted data item. It first obtains attribute authentication from the relevant authorities and then submits a hidden attribute representation and an access qualification proof. The proof shows that DR satisfies the policy without revealing its plaintext attributes.

4) Cloud Storage Center CSC: CSC is only used as an off-chain ciphertext storage component. It can be any storage service trusted or selected by DP, or even storage maintained by DP itself. CSC is not a core cryptographic participant: it neither authenticates attributes nor verifies access proofs, and it cannot recover plaintext data from ciphertexts.

5) Blockchain BC: BC maintains public records required for access control, such as hidden attribute representations, hidden policy representations, proof transcripts, data indexes, and verification results. The smart contract performs the public verification of access qualification proofs and records whether an access request is accepted.

Overall, the proposed scheme consists of six main algorithms:

$$\Pi_{AC} = \{\mathsf{Setup}, \mathsf{AttrAuth}, \mathsf{PolicyUpload}, \mathsf{ProveAccess}, \mathsf{VerifyAccess}, \mathsf{DecryptData}\}.$$

1) System initialization: The system publishes public parameters and deploys the smart contract used for recording commitments and verifying proofs.

2) Attribute authentication: DR obtains authentication evidence from the relevant attribute authorities and publishes hidden commitments to its authenticated attributes. The plaintext attributes are kept locally by DR and are not revealed to BC.

3) Policy upload: DP encrypts the shared data, stores the ciphertext off-chain, and publishes a hidden policy commitment together with the data identifier and ciphertext index. The plaintext policy is not written to the blockchain.

4) Access proof generation: When DR requests a data item, it generates an access qualification proof showing that the authenticated attribute commitments satisfy the committed policy. The proof hides the plaintext attributes and the policy vector from public observers.

5) Access proof verification: The smart contract verifies the access proof and records whether the access request is accepted.

6) Data decryption: If the proof is accepted, DP releases the required symmetric key through the system's secure delivery channel. DR retrieves the ciphertext from CSC, checks the on-chain digest, and decrypts the data.

B. Data Structures

We review the main data structures used in the proposed protocol. These structures are public interfaces between the system model, the security model, and the detailed construction.

Ledger. We regard BC as a decentralized append-only ledger maintained by blockchain nodes. Every participant can read

the ledger, and the smart contract writes verification-related records to it. The ledger stores public access-control records, including attribute records, policy records, access records, data indexes, and verification results. Plaintext attributes, plaintext policies, and plaintext data are not stored on the ledger.

User key pair. Each data requester DR runs $\mathsf{SIG.KeyGen}(pp_{\mathsf{sig}})$ to obtain (pk_{DR}, sk_{DR}). The public key pk_{DR} is used as the requester's public identity in attribute certification and access verification, while sk_{DR} authenticates the requester in the off-chain key-delivery channel after an accepted request. Attribute certifications and the accepted access record are both bound to pk_{DR} so that certifications issued for different requesters cannot be freely combined.

Attribute record. For the attribute domain managed by AA_i, the corresponding public attribute record is

$$\mathsf{AttrRec}_i := (ID_{DR}, pk_{DR}, i, C_i, \sigma_i).$$

Here ID_{DR} and pk_{DR} identify the requester, i denotes the issuing attribute authority, C_i is a commitment to the local attribute vector certified by AA_i, and σ_i is the authority signature on $(ID_{DR}, pk_{DR}, i, C_i)$. This record is used to prove that the committed local attributes are certified by the corresponding authority and bound to the requester.

Policy record. For a data item identified by ID_D, the policy record published by DP is

$$\mathsf{PolicyRec} := (ID_D, C_y, idx_M, h).$$

Here C_y is a commitment to the policy vector, idx_M is the off-chain ciphertext index, and h is the digest binding the ciphertext to the ledger record. The plaintext policy and the opening of C_y are kept off-chain.

Access record. When DR requests access to a data item, the on-chain access record is

$$\mathsf{AccessRec} := (ID_D, ID_{DR}, pk_{DR}, C_x, C_y, \pi_{\mathsf{AC}}, result).$$

Here C_x is the aggregated attribute commitment, C_y is copied from the target policy record to bind the request to that committed policy, π_{AC} is the access proof, and $result \in \{\mathsf{pending}, 0, 1\}$ is the smart-contract verification state. The authority-certified local commitments are not retransmitted in the access request; the contract reads the corresponding registered attribute records from BC during verification. The requester first submits a pending record; the contract then updates its result field after verification.

Data ciphertext record. The shared data are encrypted and stored outside the blockchain. We denote the ciphertext record by

$$\mathsf{DataRec} := (ID_D, idx_M, C_M, h).$$

Here C_M is the ciphertext of the shared data, idx_M points to its off-chain storage location, and h is the digest used by DR to check that the retrieved ciphertext is the one bound to the on-chain policy record.

C. Security Model

We define three security properties for the proposed protocol: policy hiding, attribute hiding, and authenticated attribute integrity. The protocol is denoted by

$$\Pi_{\mathsf{AC}} = \{\mathsf{Setup}, \mathsf{AttrAuth}, \mathsf{PolicyUpload}, \mathsf{ProveAccess}, \mathsf{VerifyAccess}, \mathsf{DecryptData}\}.$$

Policy hiding and attribute hiding are privacy properties, while authenticated attribute integrity is an authorization-security property. Let $\mathsf{negl}(\lambda)$ denote a negligible function.

Following the ledger-query style of LedgerMaze, T denotes a protocol query type, $\mathcal{O}_0$ and $\mathcal{O}_1$ denote the system interfaces used by the adversary to prepare two candidate ledgers, and $\mathsf{Query}_{\mathsf{Ledger}_b}(T)$ denotes the public record produced by executing query T on the ledger selected by b. For each privacy game, the two candidate ledgers have the same public structure and differ only in the challenged policy or attributes. In a verification-hiding game, they also produce the same public verification result. These restrictions exclude trivial distinctions from record formats or result bits without introducing construction-level inputs into the security model.

- *Policy Hiding.* Policy hiding comprises policy-record hiding and policy-verification hiding. The former considers the public ledger record produced by policy upload, whereas the latter considers the public records produced by proof generation and verification.

  Game 1-A: Policy-record hiding. This game asks whether the policy-upload record alone reveals which candidate policy was used.

  Experiment $\mathsf{Game}^{\mathsf{PH\text{-}Rec}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$
  1: $PP \leftarrow \mathsf{Setup}(1^\lambda)$
  2: $\mathsf{Ledger}_0, \mathsf{Ledger}_1 \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(PP)$
  3: $b \xleftarrow{\$} \{0,1\}$
  4: $T \leftarrow \mathsf{PolicyUpload}$
  5: $a \leftarrow \mathsf{Query}_{\mathsf{Ledger}_b}(T)$
  6: $b' \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(\mathsf{Ledger}_0, \mathsf{Ledger}_1, a)$
  7: **return** $b = b'$

  Game 1-B: Policy-verification hiding. This game asks whether the public proof-generation or verification record reveals which candidate policy is used. The two candidate ledgers contain the same requester and authenticated attributes and produce the same verification result.

  Experiment $\mathsf{Game}^{\mathsf{PH\text{-}Ver}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$
  1: $PP \leftarrow \mathsf{Setup}(1^\lambda)$
  2: $\mathsf{Ledger}_0, \mathsf{Ledger}_1 \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(PP)$
  3: $b \xleftarrow{\$} \{0,1\}$
  4: $T \xleftarrow{\$} \{\mathsf{ProveAccess}, \mathsf{VerifyAccess}\}$
  5: $a \leftarrow \mathsf{Query}_{\mathsf{Ledger}_b}(T)$
  6: $b' \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(\mathsf{Ledger}_0, \mathsf{Ledger}_1, a)$
  7: **return** $b = b'$

  For $X \in \{\mathsf{PH\text{-}Rec}, \mathsf{PH\text{-}Ver}\}$, define $\mathsf{Adv}^X_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Game}^X_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = 1] - 1/2|$. The protocol satisfies policy hiding if both advantages are negligible for every PPT adversary.
- *Attribute Hiding.* Attribute hiding comprises attribute-record hiding and attribute-verification hiding. The former considers the public ledger record produced by

attribute authentication, whereas the latter considers the public records produced by proof generation and verification.

Game 2-A: Attribute-record hiding. This game exposes only the public records produced during attribute authentication.

Experiment $\mathsf{Game}^{\mathsf{AH\text{-}Rec}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$

$PP \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{Ledger}_0, \mathsf{Ledger}_1 \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(PP)$
$b \xleftarrow{\$} \{0,1\}$
$T \leftarrow \mathsf{AttrAuth}$
$a \leftarrow \mathsf{Query}_{\mathsf{Ledger}_b}(T)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(\mathsf{Ledger}_0, \mathsf{Ledger}_1, a)$
return $b = b'$

Game 2-B: Attribute-verification hiding. This game asks whether the public proof-generation or verification record reveals which candidate attributes are used. The two candidate ledgers contain the same policy and produce the same verification result.

Experiment $\mathsf{Game}^{\mathsf{AH\text{-}Ver}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$

$PP \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{Ledger}_0, \mathsf{Ledger}_1 \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(PP)$
$b \xleftarrow{\$} \{0,1\}$
$T \xleftarrow{\$} \{\mathsf{ProveAccess}, \mathsf{VerifyAccess}\}$
$a \leftarrow \mathsf{Query}_{\mathsf{Ledger}_b}(T)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_0,\mathcal{O}_1}(\mathsf{Ledger}_0, \mathsf{Ledger}_1, a)$
return $b = b'$

For $X \in \{\mathsf{AH\text{-}Rec}, \mathsf{AH\text{-}Ver}\}$, define $\mathsf{Adv}^X_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Game}^X_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = 1] - 1/2|$. The protocol satisfies attribute hiding if both advantages are negligible for every PPT adversary.

- *Authenticated Attribute Integrity.* This property requires that no PPT adversary can produce a new attribute authentication that is accepted as having been issued by an honest authority. The oracle $\mathcal{O}_{\mathsf{Auth}}$ executes AttrAuth, appends the resulting public attribute record to Ledger, stores the issued record and its authenticated attribute in Issued, and returns the public record. The internal fields and cryptographic checks used to determine acceptance are deferred to the detailed construction. Membership in Issued is semantic: producing different signature bytes for the same authenticated public contents and attribute is not counted as a new authentication.

Experiment $\mathsf{Game}^{\mathsf{AAI}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$

$PP \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{Ledger} \leftarrow \emptyset$; $\mathsf{Issued} \leftarrow \emptyset$
$(\mathsf{AttrRec}^*, L^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Auth}}}(PP, \mathsf{Ledger})$
If $\mathsf{AttrRec}^*$ is not accepted for L^*, **return** 0
If $(\mathsf{AttrRec}^*, L^*) \in \mathsf{Issued}$, **return** 0
return 1

The adversary's advantage is $\mathsf{Adv}^{\mathsf{AAI}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{AAI}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = 1]$. We say that Π_{AC} satisfies authenticated attribute integrity if this advantage is at most $\mathsf{negl}(\lambda)$ for every PPT adversary $\mathcal{A}$.

Data confidentiality and ciphertext integrity are provided by SE, the authenticated confidential key-delivery channel, and the on-chain digest check, and are not treated as separate access-control properties in this model.

Algorithm 1 $\mathsf{Setup}(1^\lambda, N, \{V_i, n_i\}_{i=1}^N)$

Require: Security parameter 1^λ, number of attribute authorities N, attribute domains $\{V_i\}$, and polynomial degrees $\{n_i\}$.
Ensure: Public parameters PP and authority key pairs.

Set $m_i \leftarrow n_i + 1$, compute $m \leftarrow \sum_{i=1}^N m_i$, and set $\ell_i \leftarrow \sum_{k<i} m_k$.
Generate $pp_{\mathsf{com}} \leftarrow \mathsf{COM.Setup}(1^\lambda, m)$.
Generate $pp_{\mathsf{sig}} \leftarrow \mathsf{SIG.Setup}(1^\lambda)$.
Generate $pp_{\mathsf{nizk}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, m, pp_{\mathsf{com}})$.
Select one hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ with domain-separated inputs.
for $i = 1$ to N **do**
    Generate $(pk_{AA_i}, sk_{AA_i}) \leftarrow \mathsf{SIG.KeyGen}(pp_{\mathsf{sig}})$.
end for
Set $aux = (N, \{V_i, n_i, m_i, \ell_i, pk_{AA_i}\}_{i=1}^N)$.
Set $PP = (pp_{\mathsf{com}}, pp_{\mathsf{sig}}, pp_{\mathsf{nizk}}, H, aux)$.
Publish PP on BC and keep each sk_{AA_i} locally.
return $(PP, \{(pk_{AA_i}, sk_{AA_i})\}_{i=1}^N)$.

V. Scheme Construction

Fig. 2 illustrates the workflow of the proposed scheme.

A. Detailed Construction

The proposed hidden access-control scheme is defined by six algorithms:

$$\Pi_{\mathsf{AC}} = \{\mathsf{Setup}, \mathsf{AttrAuth}, \mathsf{PolicyUpload}, \mathsf{ProveAccess}, \mathsf{VerifyAccess}, \mathsf{DecryptData}\}.$$

1) System setup. The setup algorithm initializes public parameters and authority keys, as shown in Algorithm 1. Assume that the system contains N attribute authorities $AA_1, \ldots, AA_N$. Authority AA_i manages the domain V_i and uses a degree-n_i policy polynomial, so its local vector dimension is $m_i = n_i + 1$ and the global dimension is $m = \sum_{i=1}^N m_i$. Let $\ell_i = \sum_{k<i} m_k$ be the offset of the i-th attribute domain. Thus, AA_i uses the generator subset $G_{\ell_i+1}, \ldots, G_{\ell_i+m_i}$.

The same hash function H is used with domain labels, such as $H(\text{“data-id”}\|\cdot)$ for identifiers and $H(\text{“data”}\|\cdot)$ for ciphertext digests.

2) Attribute authentication. The attribute authentication algorithm is executed between DR and AA_i, as shown in Algorithm 2. The authority authenticates the requester value $L_i \in V_i$, invokes the local vectorization interface, and signs the resulting local commitment.

The authority signs only well-formed attribute encodings. Therefore, attribute well-formedness is enforced during attribute authentication rather than by an additional zero-knowledge proof in the basic construction. After interacting with all required authorities, DR forms $\mathbf{x} = \mathbf{x}^{(1)}\| \cdots \|\mathbf{x}^{(N)}$

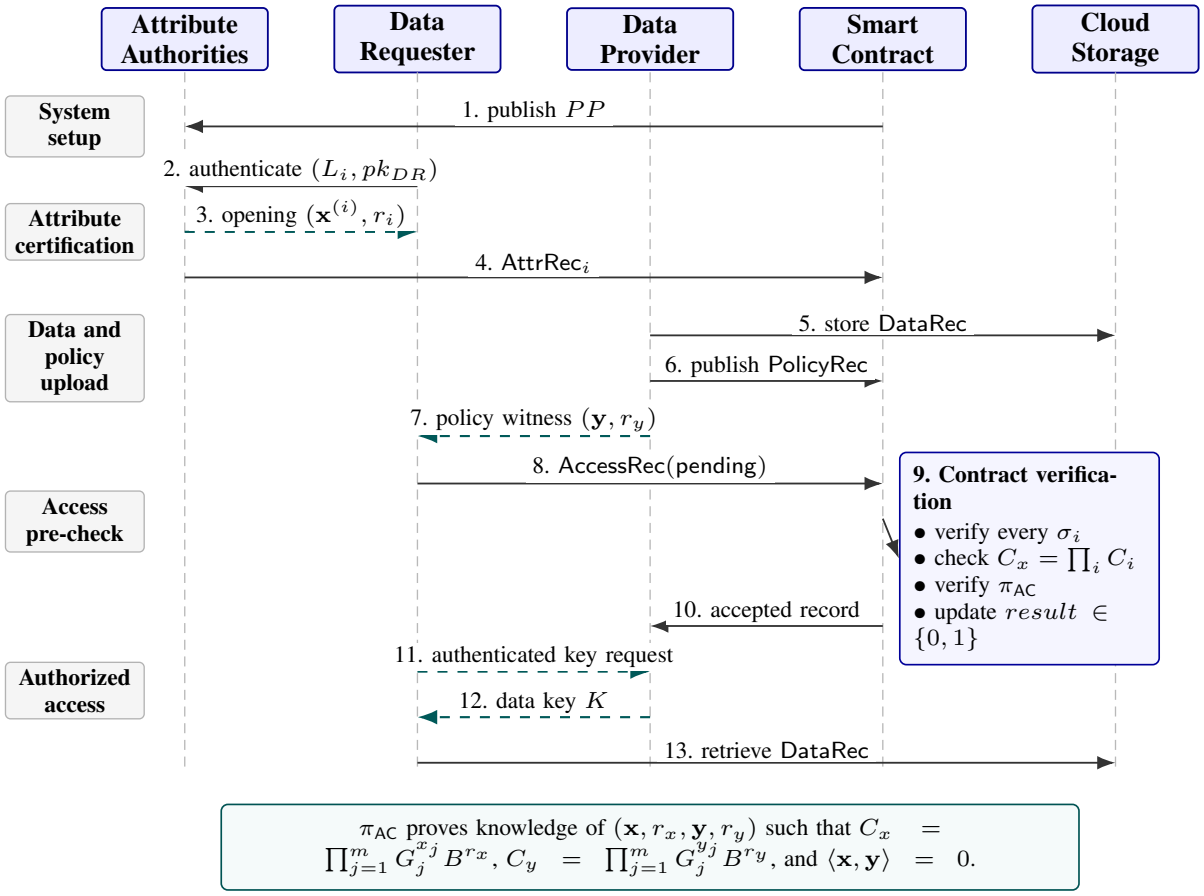

Fig. 2. Workflow of the proposed scheme.

Algorithm 2 AttrAuth$(AA_i, DR, L_i, ID_{DR}, pk_{DR}, PP)$

Require: Attribute authority AA_i, requester DR, authenticated value L_i, requester identifier ID_{DR}, requester public key pk_{DR}, and PP.

Ensure: Attribute record AttrRec$_i$ and local opening $(\mathbf{x}^{(i)}, r_i)$.

1: AA_i authenticates L_i and checks $L_i \in V_i$.
2: **if** authentication or well-formedness check fails **then**
3: **return** $\perp$.
4: **end if**
5: Compute $\mathbf{x}^{(i)} \leftarrow \mathsf{Vec.Attr}_i(L_i)$.
6: Choose $r_i \leftarrow \mathbb{Z}_q$.
7: Compute $C_i \leftarrow \mathsf{COM.Commit}_{I_i}(pp_{\mathsf{com}}, \mathbf{x}^{(i)}; r_i)$.
8: Set $msg_i \leftarrow ID_{DR} \| pk_{DR} \| i \| C_i$.
9: Compute $\sigma_i \leftarrow \mathsf{SIG.Sign}(sk_{AA_i}, msg_i)$.
10: Set AttrRec$_i \leftarrow (ID_{DR}, pk_{DR}, i, C_i, \sigma_i)$ and append it to BC.
11: Return AttrRec$_i$ and deliver $(\mathbf{x}^{(i)}, r_i)$ to DR.

and computes $r_x = \sum_{i=1}^{N} r_i$. Since each C_i uses a disjoint generator interval, the aggregated commitment satisfies

$$\begin{aligned} C_x &= \prod_{i=1}^{N} C_i \\ &= \prod_{i=1}^{N} \prod_{j=1}^{m_i} G_{\ell_i+j}^{x_j^{(i)}} B^{\sum_i r_i} \\ &= \prod_{j=1}^{m} G_j^{x_j} B^{r_x} = \mathsf{COM.Commit}(pp_{\mathsf{com}}, \mathbf{x}; r_x). \end{aligned}$$

This equality is the reason why local attribute commitments can be publicly aggregated into the global attribute commitment C_x.

3) Policy upload and data encryption. The policy upload algorithm is executed by DP, as shown in Algorithm 3. The data provider encrypts the shared data and publishes only a commitment to the policy vector.

The attribute order and polynomial degrees are fixed by PP, so no policy-specific encoding information is published. The plaintext policy and policy vector are not written to BC.

4) Access proof generation. The proof generation algorithm is executed by DR, as shown in Algorithm 4. The requester generates a zero-knowledge proof showing that the committed attributes satisfy the committed policy. The algorithm writes the proof statement explicitly; its decomposition into familiar NIZK relations is given later in Section V-B.

Algorithm 3 PolicyUpload(DP, M, P, PP)

Require: Data provider DP, plaintext data M, access policy P, and public parameters PP.
Ensure: Data identifier ID_D, ciphertext C_M, policy commitment C_y, ciphertext index idx_M, and digest h.

Choose a fresh nonce $\eta_D \leftarrow \{0,1\}^\lambda$.
Compute $ID_D \leftarrow H(\text{“data-id”} \| ID_{DP} \| \eta_D)$.
Vectorize the policy $\mathbf{y} \leftarrow \mathsf{Vec.Policy}(P)$.
Choose $r_y \leftarrow \mathbb{Z}_q$.
Compute $C_y \leftarrow \mathsf{COM.Commit}(pp_{\mathsf{com}}, \mathbf{y}; r_y)$.
Generate a data key $K \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$.
Encrypt $C_M \leftarrow \mathsf{SE.Enc}(K, M)$.
Choose an off-chain index idx_M and compute $h \leftarrow H(\text{“data”} \| ID_D \| C_M)$.
Set $\mathsf{DataRec} \leftarrow (ID_D, idx_M, C_M, h)$ and store it at CSC.
Set $\mathsf{PolicyRec} \leftarrow (ID_D, C_y, idx_M, h)$ and append it to BC.
Store $(ID_D, K, \mathbf{y}, r_y, \eta_D)$ locally.
return $(ID_D, C_M, C_y, idx_M, h)$.

Algorithm 4 ProveAccess

Require: Requester DR, provider DP, data identifier ID_D, requester identifier ID_{DR}, requester public key pk_{DR}, and PP.
Ensure: Access proof π_{AC}.

Read $\mathsf{PolicyRec} = (ID_D, C_y, idx_M, h)$ from BC.
Read $\{\mathsf{AttrRec}_i = (ID_{DR}, pk_{DR}, i, C_i, \sigma_i)\}_{i=1}^N$ from BC.
for $i = 1$ to N **do**
  Set $msg_i \leftarrow ID_{DR} \| pk_{DR} \| i \| C_i$.
  Check $\mathsf{SIG.Verify}(pk_{AA_i}, msg_i, \sigma_i) = 1$.
  if the check fails **then**
    return $\perp$.
  end if
end for
Compute $C_x \leftarrow \prod_{i=1}^N C_i$.
Set $\mathbf{x} \leftarrow \mathbf{x}^{(1)} \| \cdots \| \mathbf{x}^{(N)}$ and $r_x \leftarrow \sum_{i=1}^N r_i$.
Obtain $(\mathbf{y}, r_y)$ from DP through the off-chain proof-generation channel.
Check $\mathsf{COM.Open}(pp_{\mathsf{com}}, C_y, \mathbf{y}, r_y) = 1$; otherwise return $\perp$.
Check $\mathsf{Vec.Match}(\mathbf{x}, \mathbf{y}) = 1$; otherwise return $\perp$.
Set $stmt \leftarrow (PP, ID_D, ID_{DR}, pk_{DR}, C_x, C_y)$.
Set $w \leftarrow (\mathbf{x}, r_x, \mathbf{y}, r_y)$.
Compute $\pi_{\mathsf{AC}} \leftarrow \mathsf{NIZK.Prove}(pp_{\mathsf{nizk}}, stmt, w)$.
Set $\mathsf{AccessRec} \leftarrow (ID_D, ID_{DR}, pk_{DR}, C_x, C_y, \pi_{\mathsf{AC}}, \mathsf{pending})$ and append it to BC.
return π_{AC}.

The requester cannot replace the committed policy with another vector, because the proof must open the on-chain policy commitment C_y. Similarly, it cannot replace authenticated attributes, because C_x must be aggregated from authority-signed local commitments.

5) On-chain access verification. The verification algorithm is executed by the smart contract, as shown in Algorithm 5. Before verifying π_{AC}, the contract

Algorithm 5 VerifyAccess

Require: PP and a pending AccessRec.
Ensure: Verification result $b \in \{0,1\}$.

Parse AccessRec as $(ID_D, ID_{DR}, pk_{DR}, C_x, C_y, \pi_{\mathsf{AC}}, \mathsf{pending})$.
Read $\mathsf{PolicyRec} = (ID_D, C'_y, idx_M, h)$ from BC and check $C_y = C'_y$.
if the check fails **then**
  Update $result \leftarrow 0$ in AccessRec and **return** 0.
end if
for $i = 1$ to N **do**
  Read $\mathsf{AttrRec}_i = (ID_{DR}, pk_{DR}, i, C_i, \sigma_i)$ from BC.
  Check that $\mathsf{AttrRec}_i$ is registered and unique for the request.
  Set $msg_i \leftarrow ID_{DR} \| pk_{DR} \| i \| C_i$.
  Verify $\mathsf{SIG.Verify}(pk_{AA_i}, msg_i, \sigma_i) = 1$.
  if any check fails **then**
    Update $result \leftarrow 0$ in AccessRec and **return** 0.
  end if
end for
Check $C_x = \prod_{i=1}^N C_i$.
if the check fails **then**
  Update $result \leftarrow 0$ in AccessRec and **return** 0.
end if
Set $stmt = (PP, ID_D, ID_{DR}, pk_{DR}, C_x, C_y)$.
Compute $b \leftarrow \mathsf{NIZK.Verify}(pp_{\mathsf{nizk}}, stmt, \pi_{\mathsf{AC}})$.
Update $result \leftarrow b$ in AccessRec.
return b.

checks every registered attribute record by evaluating $\mathsf{SIG.Verify}(pk_{AA_i}, ID_{DR} \| pk_{DR} \| i \| C_i, \sigma_i) = 1$ and checks $C_x = \prod_i C_i$. These public checks ensure that the local commitments are authority-certified, requester-bound, and correctly aggregated; they are performed outside the zero-knowledge proof. The contract then verifies π_{AC} against the committed policy in PolicyRec.

The signature checks are public predicates and are not part of the zero-knowledge proof. The NIZK proof only hides the vectors and commitment openings.

6) Data decryption. The data decryption algorithm is executed after successful verification, as shown in Algorithm 6. DP releases the data key only for an accepted access record. The concrete secure delivery method is orthogonal to the access proof; it can be instantiated by an authenticated encrypted channel between DP and DR.

B. Access Proof Instantiation

Attribute-authority signatures and the aggregation equation $C_x = \prod_i C_i$ are verified publicly and therefore are not included in the zero-knowledge statement. The access proof only establishes knowledge of the openings of C_x and C_y and the hidden matching relation between their committed vectors.

Commitment-opening argument. We prove knowledge of a vector-commitment opening using the multi-base Schnorr–Okamoto representation protocol [23], [26], [27], made non-

Algorithm 6 DecryptData

Require: Requester DR, provider DP, data identifier ID_D, requester identifier ID_{DR}, requester public key pk_{DR}, and PP.

Ensure: Plaintext data M or $\perp$.

1: Read an AccessRec for (ID_D, ID_{DR}, pk_{DR}) with $result = 1$.
2: **if** no accepted record exists **then**
3: **return** $\perp$.
4: **end if**
5: Read PolicyRec $= (ID_D, C_y, idx_M, h)$ from BC.
6: DR authenticates with sk_{DR} in the secure channel bound to (ID_D, ID_{DR}, pk_{DR}).
7: DP sends K through that channel only after checking the accepted record.
8: Retrieve DataRec $= (ID_D, idx_M, C_M, h_{\mathsf{CSC}})$ from CSC.
9: Compute $h' \leftarrow H(\text{“data”} \| ID_D \| C_M)$.
10: **if** $h' \neq h$ or $h_{\mathsf{CSC}} \neq h$ **then**
11: **return** $\perp$.
12: **end if**
13: Recover $M \leftarrow \mathsf{SE.Dec}(K, C_M)$.
14: **return** M.

interactive by the Fiat–Shamir transform [22]. For a public commitment $C = \prod_{j=1}^{m} G_j^{v_j} B^r$, define

$$\mathcal{R}_{\mathsf{open}} = \left\{ (C; \mathbf{v}, r) \;\middle|\; C = \prod_{j=1}^{m} G_j^{v_j} B^r \right\}.$$

The proof $\pi_{\mathsf{open}} = \mathsf{NIZK}\{(\mathbf{v}, r) \mid (C; \mathbf{v}, r) \in \mathcal{R}_{\mathsf{open}}\}$ establishes knowledge of the committed vector and blinding factor without revealing either value. This representation relation is used for both C_x and C_y inside the unified committed inner-product proof below; separate opening transcripts are not added to the access proof. The cited representation protocol supplies completeness and knowledge soundness, while its Fiat–Shamir form is used as a zero-knowledge argument in the random-oracle model. The standard prover, verifier, and extractor equations are omitted.

Committed inner-product argument. We instantiate hidden matching with the argument of Bootle et al. for openings of two Pedersen multicommitments satisfying an inner-product relation [19], and use the Bulletproof commit-and-prove framework to obtain a short non-interactive argument [18]. For the externally committed inputs, define

$$\begin{aligned} \mathcal{R}_{\mathsf{cip}} = \{ (C_x, C_y; \mathbf{x}, \mathbf{y}, r_x, r_y) \mid \\ C_x = \textstyle\prod_{j=1}^{m} G_j^{x_j} B^{r_x}, \\ C_y = \textstyle\prod_{j=1}^{m} G_j^{y_j} B^{r_y}, \\ \langle \mathbf{x}, \mathbf{y} \rangle = 0 \}. \end{aligned}$$

The condition $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ is exactly the vectorized policy-matching predicate. The corresponding proof $\pi_{\mathsf{match}} = \mathsf{NIZK}\{(\mathbf{x}, \mathbf{y}, r_x, r_y) \mid (C_x, C_y; \mathbf{x}, \mathbf{y}, r_x, r_y) \in \mathcal{R}_{\mathsf{cip}}\}$ jointly proves the two openings and their inner-product relation. Hence, the committed wires are the openings of the public commitments rather than independently chosen IPA inputs. The recursive IPA compresses the relation to $O(\log m)$ proof size; it is only a compression argument, whereas witness hiding is supplied by the masking and commitment mechanisms of the surrounding commit-and-prove protocol.

Access-proof decomposition. The public statement and witness are fixed throughout the construction as $stmt = (PP, ID_D, ID_{DR}, pk_{DR}, C_x, C_y)$ and $w = (\mathbf{x}, \mathbf{y}, r_x, r_y)$, respectively. Because π_{match} already proves knowledge of both external commitment openings and the zero inner product under one witness assignment, the access proof is

$$\pi_{\mathsf{AC}} = \pi_{\mathsf{match}}.$$

The domain-separated Fiat–Shamir context is $\mathsf{ctx} = (ID_D, ID_{DR}, pk_{DR}, C_x, C_y)$, which binds the proof to the data item, requester, delivery key, and on-chain commitments. Thus, π_{match} proves the openings of C_x and C_y together with their zero inner product, and directly realizes the complete access relation. No separate π_{com} or opening proof is included, because either would repeat constraints already enforced by $\mathcal{R}_{\mathsf{cip}}$.

VI. Security Analysis

Theorem 1 (Policy Hiding). If the multi-generator Pedersen commitment is perfectly hiding and the instantiated access proof π_{AC} is zero knowledge, then the proposed scheme satisfies policy-record hiding and policy-verification hiding.

Proof. There are two cases in which an adversary may try to distinguish the challenge policies P_0 and P_1.

Case 1: policy-record hiding. In PolicyUpload, the policy P_b is encoded as $\mathbf{y}_b = \mathsf{Vec.Policy}(P_b)$ and committed as $C_{y,b} = B^{r_{y,b}} \prod_{j=1}^{m} G_j^{y_{b,j}}$, where $r_{y,b} \xleftarrow{\$} \mathbb{Z}_q$. Since B generates the prime-order group, $B^{r_{y,b}}$ is uniform in $\mathbb{G}$. Multiplying it by the fixed element $\prod_j G_j^{y_{b,j}}$ preserves uniformity. Hence $C_{y,0}$ and $C_{y,1}$ are identically distributed by the perfect hiding of the Pedersen commitment [21].

The remaining fields of $\mathsf{PolicyRec}_b = (ID_{D,b}, C_{y,b}, idx_{M,b}, h_b)$ do not encode P_b. The identifier is derived from ID_{DP} and a fresh nonce, while $idx_{M,b}$ and h_b locate and bind the encryption of the common message M. Their distributions are therefore independent of the challenge bit. Consequently, the two policy-record views are identical and $\mathsf{Adv}^{\mathsf{PH\text{-}Rec}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = 0$.

Case 2: policy-verification hiding. The attribute records, requester identity, requester public key, and aggregated commitment C_x are fixed in both worlds. The policy-dependent public values are $C_{y,b}$ and the unified committed inner-product proof $\pi_{\mathsf{AC},b} = \pi_{\mathsf{match},b}$. By the admissibility condition of Game 1-B, both policies satisfy the fixed attributes, and thus $result_0 = result_1 = 1$.

Consider the following hybrids. In H_0, the challenger returns the real verification view for P_0. In H_1, it replaces the real access proof with a simulated proof generated from the public statement $stmt_0 = (PP, ID_{D,0}, ID_{DR}, pk_{DR}, C_x, C_{y,0})$, without using $(\mathbf{x}, \mathbf{y}_0, r_x, r_{y,0})$. Zero knowledge makes H_0 and H_1 computationally indistinguishable. In H_2, the challenger replaces the policy record and $C_{y,0}$ in the statement with

their P_1 counterparts. Pedersen hiding makes the public statements identically distributed, and the simulator depends only on those public statements; therefore H_1 and H_2 are identically distributed. Finally, H_3 replaces the simulated proof for $stmt_1$ with the real proof for P_1, which is again indistinguishable by zero knowledge. This simulation covers the unified opening and inner-product relation proved by π_{match} [18], [19].

Thus, for an appropriate zero-knowledge distinguisher $\mathcal{B}$,

$$\mathsf{Adv}^{\mathsf{PH\text{-}Ver}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) \le 2\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda),$$

which is negligible. Both policy-hiding games are secure, so Π_{AC} satisfies policy hiding. □

Theorem 2 (Attribute Hiding). If the multi-generator Pedersen commitment is perfectly hiding and π_{AC} is zero knowledge, then the proposed scheme satisfies attribute-record hiding and attribute-verification hiding.

Proof. There are two cases in which an adversary may try to distinguish the challenge attribute lists $\mathbf{L}_0$ and $\mathbf{L}_1$.

Case 1: attribute-record hiding. For each authority AA_i, AttrAuth encodes $L_{b,i}$ as $\mathbf{x}_b^{(i)} = (1, L_{b,i}, \ldots, L_{b,i}^{n_i})$ and computes $C_{i,b} = B^{r_{i,b}} \prod_{j=1}^{m_i} G_{\ell_i+j}^{x_{b,j}^{(i)}}$ with independently uniform $r_{i,b}$. Perfect hiding implies that $C_{i,0}$ and $C_{i,1}$ are identically distributed. The public fields ID_{DR}, pk_{DR}, and i are fixed, and AA_i signs only $msg_{i,b} = ID_{DR}\|pk_{DR}\|i\|C_{i,b}$. Because the commitment distributions are identical, applying the same randomized Schnorr signing algorithm preserves their joint distribution. Thus $(C_{i,0}, \sigma_{i,0})$ and $(C_{i,1}, \sigma_{i,1})$ are identically distributed.

All authorities choose their commitment and signing randomness independently. It follows that the complete record sets $\{\mathsf{AttrRec}_{i,0}\}_{i=1}^N$ and $\{\mathsf{AttrRec}_{i,1}\}_{i=1}^N$ are identically distributed, giving $\mathsf{Adv}^{\mathsf{AH\text{-}Rec}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) = 0$.

Case 2: attribute-verification hiding. The policy record and C_y are fixed in both challenge worlds. Each local commitment $C_{i,b}$ is perfectly hiding, the accompanying signature authenticates only that hidden commitment, and the aggregate $C_{x,b} = \prod_i C_{i,b}$ is a deterministic function of the local public records. By the admissibility condition of Game 2-B, both attribute lists satisfy the fixed policy, so the verification results are equal to one.

Starting from the real view for $\mathbf{L}_0$, replace $\pi_{\mathsf{AC},0}$ with a simulated proof for $stmt_0 = (PP, ID_D, ID_{DR}, pk_{DR}, C_{x,0}, C_y)$. Next, replace the local attribute records and the derived $C_{x,0}$ with their $\mathbf{L}_1$ counterparts. The record-hiding argument above makes these public statements identically distributed; after the proof has been simulated, no hidden opening or attribute vector remains in the transcript. Finally, replace the simulated proof for $stmt_1$ with the real proof $\pi_{\mathsf{AC},1}$. The first and last transitions are computationally indistinguishable by zero knowledge, and the middle transition is exact by Pedersen hiding.

Therefore, for an appropriate distinguisher $\mathcal{B}'$,

$$\mathsf{Adv}^{\mathsf{AH\text{-}Ver}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) \le 2\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda),$$

which is negligible. Both attribute-hiding games are secure, so Π_{AC} satisfies attribute hiding. □

Theorem 3 (Authenticated Attribute Integrity). If the Schnorr signature is EUF-CMA secure and the multi-generator Pedersen commitment is computationally binding, then the proposed scheme satisfies authenticated attribute integrity.

Proof. In AttrAuth, an honest authority AA_i first computes $\mathbf{x}^{(i)} = \mathsf{Vec.Attr}_i(L_i)$ and $C_i = \mathsf{COM.Commit}_{I_i}(pp_{\mathsf{com}}, \mathbf{x}^{(i)}; r_i)$. It then signs $msg_i = ID_{DR}\|pk_{DR}\|i\|C_i$. Thus the requester identity, requester public key, authority index, and attribute commitment are jointly authenticated by σ_i.

Suppose a PPT adversary $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{AAI}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda)$ with non-negligible probability. By the detailed construction, acceptance of $\mathsf{AttrRec}^*$ for L^* means that the authority signature is valid and the committed attribute admits a valid opening for L^*. Meanwhile, the corresponding semantic authentication is absent from Issued. There are two cases.

First, if $msg^* = ID^*\|pk^*\|i^*\|C^*$ was never submitted to the signing procedure of honest AA_{i^*}, then (msg^*, σ^*) is a valid signature on a new message. An algorithm that simulates the authentication oracle using its signing oracle can therefore use $\mathcal{A}$ to produce an EUF-CMA forgery against the Schnorr signature scheme.

Second, suppose msg^* was previously signed. Because the encoding contains the complete tuple (ID^*, pk^*, i^*, C^*), the corresponding issued entry contains the same commitment C^*. Since the output is not an issued semantic tuple but $\mathsf{ValidAttr} = 1$, $\mathcal{A}$ has supplied a distinct valid opening $(\mathsf{Vec.Attr}_{i^*}(L^*), r^*)$ of that commitment. If this opening differs from the honestly issued opening, the two openings break the computational binding of the multi-generator Pedersen commitment. Exact replay is excluded by the Issued membership test.

Consequently, a successful integrity adversary yields either a Schnorr forgery or a commitment-binding violation. By a union bound, $\mathsf{Adv}^{\mathsf{AAI}}_{\Pi_{\mathsf{AC}},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{SIG},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{Bind}}_{\mathsf{COM},\mathcal{B}_2}(\lambda)$, which is negligible. Therefore, Π_{AC} satisfies authenticated attribute integrity. □

VII. Performance Analysis

This section analyzes the performance of the proposed scheme from communication and computation costs. We compare our scheme with HAPPS [6] and EDDAC [7]. HAPPS combines distributed ABE with attribute-hidden zero-knowledge proof, and EDDAC combines attribute hiding with blockchain-based decryption tests. Both are representative privacy-preserving access-control schemes for blockchain-assisted data sharing.

We implement a Rust experimental prototype following the six algorithms in Section V. The source code, configuration files, benchmark outputs, and plotting scripts are publicly available. [1] The prototype implements the commitment, signature, encryption, ledger, and end-to-end protocol workflows, while the access-proof component uses a protocol-level simulator and the logarithmic proof-size model analyzed in this paper. The prototype uses `curve25519-dalek` for Ristretto255 group operations, SHA-2 for hashing, AES-GCM

[1] https://github.com/liangyuanshao/zk_vector_commitment_access_control

for symmetric encryption, and `rand`/`rand_chacha` for randomness. The experiment environment is shown in Table I.

TABLE I
SOFTWARE AND HARDWARE ENVIRONMENT

Type	Description
Processor	12th Gen Intel® Core™ i5-12400F
Memory	32 GB
Operating system	Windows 11 with Ubuntu 22.04 through WSL
Programming language	Rust
Main libraries	curve25519-dalek, sha2, aes-gcm, rand, rand_chacha

A. Communication Overhead Analysis

First, we analyze the communication cost of the proposed scheme. Let $|\mathbb{G}|$ denote the byte length of a compressed group element, $|\mathbb{Z}_q|$ denote the byte length of a scalar, $|H|$ denote the hash length, $|ID|$ denote the identifier length, $|pk|$ denote the public-key length, and $|idx|$ denote the ciphertext-index length. In our Rust implementation, Ristretto255 is used, so $|\mathbb{G}| = |\mathbb{Z}_q| = 32$ bytes. We set $|H| = |ID| = |pk| = |idx| = 32$ bytes, $|i| = 4$ bytes, $|K| = 16$ bytes, $|result| = 1$ byte, and $|aux_{\mathsf{pub}}| = 8$ bytes. Let m be the global vector dimension used in the access proof. Each authority manages one local vector block. In the experiment, we use a fixed block size of four coordinates for each authority, namely $m = 4N$. Thus, the number of authority records is determined by m, but the dominant scaling factor remains the global vector dimension.

TABLE II
OUTPUT SIZES OF KEY CRYPTOGRAPHIC COMPONENTS

Component	Output Size	Instantiated Size
Vector commitment	$\|\mathbb{G}\|$	32 B
Authority signature	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	64 B
Hash digest	$\|H\|$	32 B
AES-GCM data key	$\|K\|$	16 B
Bulletproof IPA component	$(2\log_2 m)\|\mathbb{G}\| + 5\|\mathbb{Z}_q\|$	$(64\log_2 m + 160)$ B
Access proof π_{AC}	$(2\log_2 m + 6)\|\mathbb{G}\|$	$(64\log_2 m + 192)$ B

Table III summarizes the communication overhead of the main protocol stages. During access verification, the requester submits only the aggregated commitment and access proof; the registered attribute records are read from BC by the contract and are not retransmitted. The final decryption stage only releases the data key through an authenticated channel; the application ciphertext is stored off-chain and is not included in the blockchain communication cost.

TABLE III
COMMUNICATION OVERHEAD OF THE PROPOSED SCHEME

Stage	Output Size	Instantiated Size
System setup	$\|\mathsf{SetupRec}\|$	$(64m + 64)$ B
Attribute authentication	$N\|\mathsf{AttrRec}_i\|$	$41m$ B
Policy upload	$\|\mathsf{PolicyRec}\|$	136 B
Access verification	$\|\mathsf{AccessRec}\|$	$(64\log_2 m + 353)$ B
Final decryption	$\|\mathsf{KeyRel}\|$	16 B

Fig. 3 compares the communication overhead with EDDAC under $m = 16, 32, 64, 128, 256, 512$. EDDAC does not report byte-level transaction sizes, so its values are estimated from the message formats in its construction. The access-verification cost of our scheme grows logarithmically with m, which reflects the compression effect of the inner-product argument. Compared with EDDAC, our scheme avoids publishing an encrypted policy vector for each access policy, so the policy-upload cost remains constant. The setup and attribute-authentication costs are higher in some settings because our scheme publishes vector-commitment parameters and authority-certified attribute records. This is a one-time or reusable cost, while the access-verification transaction remains compact.

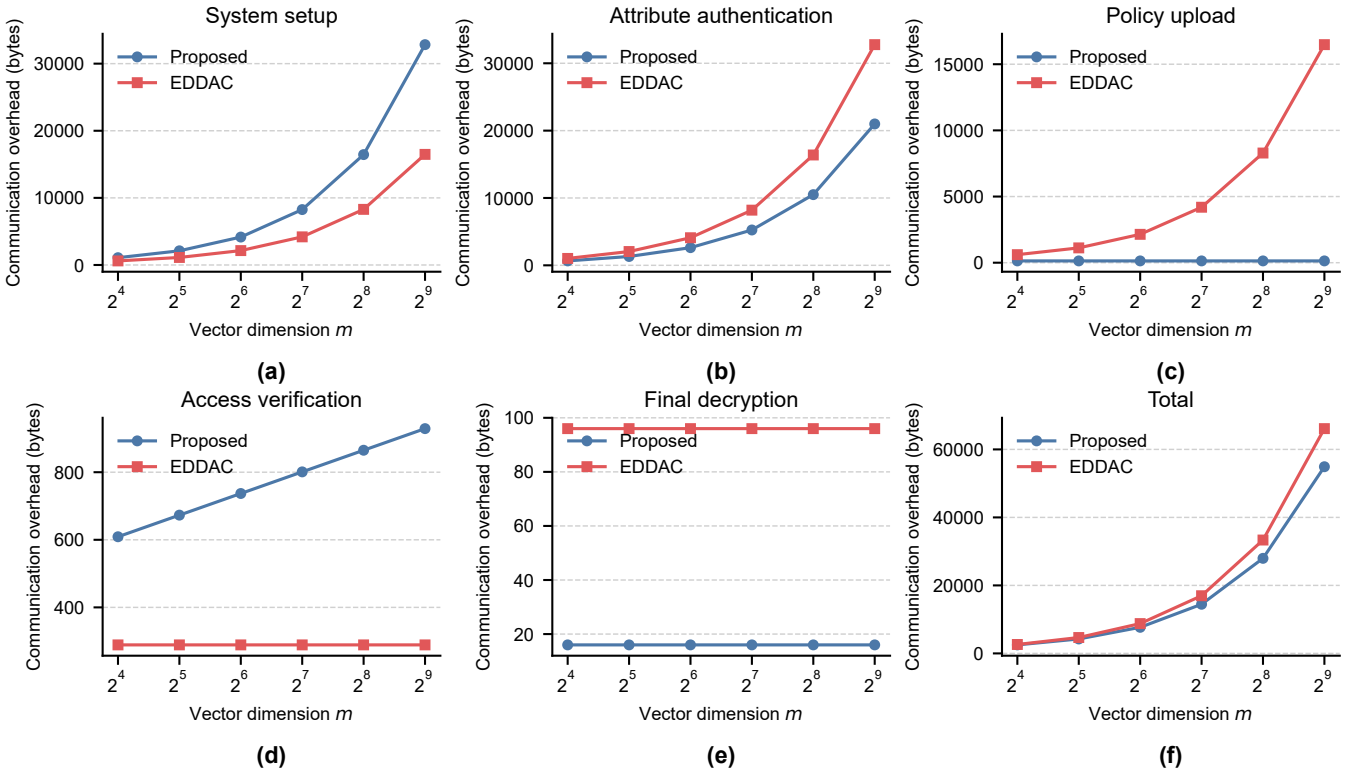

Fig. 3. Communication overhead comparison with EDDAC under different vector dimensions.

B. Computation Overhead Analysis

Next, we analyze the computation overhead. The proposed scheme is implemented in Rust with `curve25519-dalek` for Ristretto255 group operations, `ed25519-dalek` for authority signatures, SHA-2 for hashing, and AES-GCM for symmetric encryption. We measure the running time of system setup, attribute authentication, policy upload, proof generation, proof verification, and final decryption. For each dimension, the experiment is repeated five times in release mode, and Table IV reports the average time in milliseconds.

TABLE IV
COMPUTATION OVERHEAD UNDER DIFFERENT VECTOR DIMENSIONS (MS)

m	Setup	Attr. Auth.	Policy Up.	Proof Gen.	Proof Verify	Final Dec.	Total
16	0.2646	2.0428	0.6534	0.6830	1.1553	0.0937	4.8928
32	0.3636	3.6089	0.9591	1.0620	1.9505	0.0758	8.0198
64	0.6285	6.8201	1.7247	1.8279	3.6129	0.0806	14.6947
128	1.1463	13.1962	3.3056	3.5659	6.8089	0.0730	28.0958
256	2.1906	28.3197	7.1210	7.0965	14.2543	0.0857	59.0678
512	5.0257	57.6320	14.5432	15.3631	28.6063	0.1030	121.2734

For comparison, we estimate the computation trend of EDDAC according to its reported operation counts. Since EDDAC relies on CP-ABE, chameleon hashing, and blockchain-based decryption tests, its attribute processing, policy publication, and final decryption costs contain more exponentiation or pairing-related operations. Fig. 4 shows that our scheme reduces these costs, while EDDAC has a lighter matching test because it does not generate or verify a zero-knowledge access proof. In other words, EDDAC is efficient for the final inner-product matching step, but its ABE-based attribute processing and key-reconstruction workflow introduce larger costs in the surrounding stages. Our scheme shifts the main cost to the proof generation and verification stages, and in return keeps policy upload and final data access lightweight.

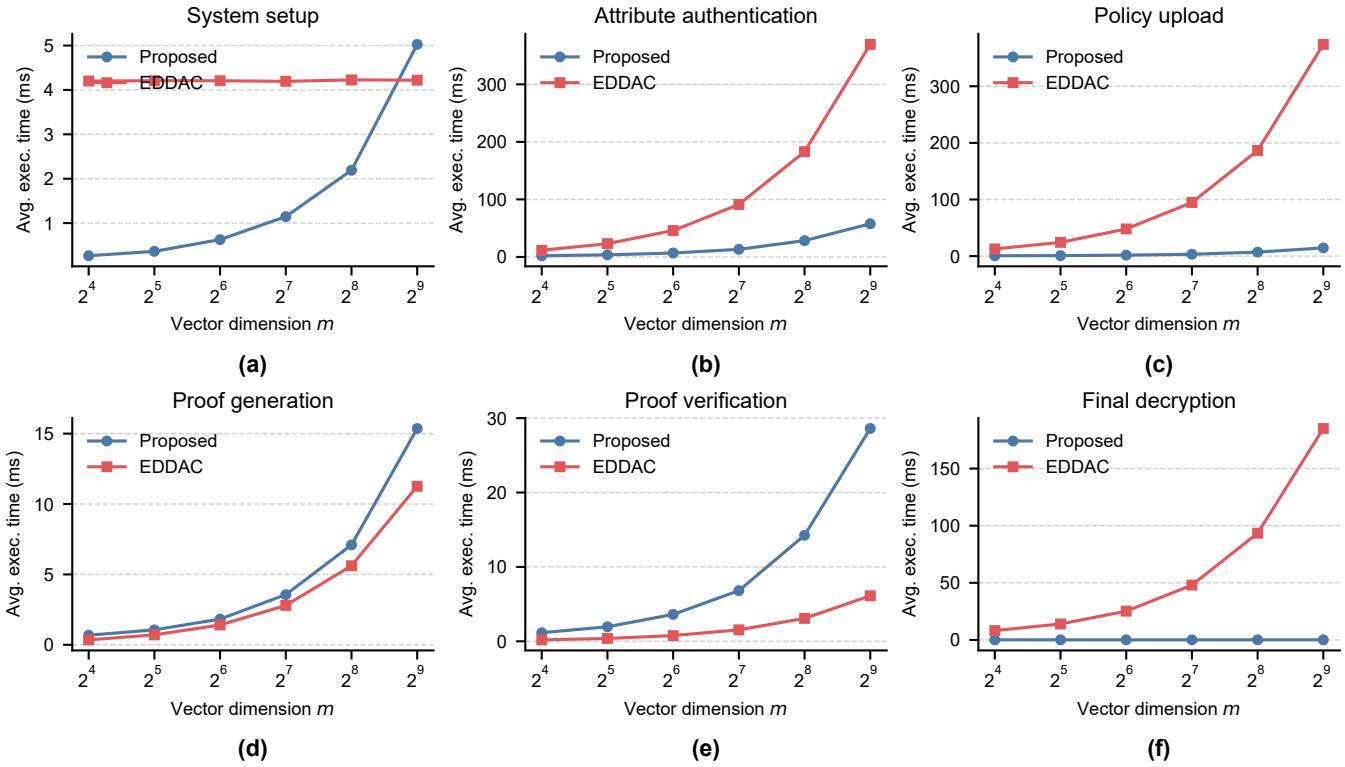

Fig. 4. Computation overhead comparison under different vector dimensions. For EDDAC, proof generation denotes access-request generation, and proof verification denotes the blockchain decryption-test phase.

VIII. Conclusion

This paper presented a blockchain-based access control scheme for publicly verifying authorization without disclosing requester attributes or access policies. The scheme encodes attributes and policies as vectors, binds requester attributes to multiauthority-certified commitments, and commits independently to each access policy. A linked non-interactive zero-knowledge argument then proves that the two committed vectors satisfy the required inner-product relation. The smart contract can therefore verify policy satisfaction from public commitments, authority signatures, and the access proof while learning only the authorization result. Under the stated assumptions, the security analysis establishes policy hiding, attribute hiding, and authenticated attribute integrity.

The theoretical and experimental results quantify the cost of obtaining this bilateral authorization privacy. The access proof has size $(64\log_2 m + 192)$ bytes, and the complete access-verification record has size $(64\log_2 m + 353)$ bytes for vector dimension m. In the Rust prototype, the access record remains below 1 KB at $m = 512$, while final data decryption takes 0.103 ms and remains independent of the attribute dimension. The construction thus keeps each access-verification transaction concise without pairing-based authorization or circuit-specific trusted setup, although setup and attribute authentication retain linear costs because commitment parameters and authority-certified records must be established. Future work will focus on integrating dynamic attribute and policy updates with efficient revocation and evaluating these mechanisms in a production smart-contract deployment.

References

[1] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology–EUROCRYPT 2005*, R. Cramer, Ed., Lecture Notes in Computer Science, vol. 3494. Berlin, Germany: Springer, 2005, pp. 457–473.

[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2007, pp. 321–334.

[3] M. Chase, "Multi-authority attribute based encryption," in *Theory of Cryptography–TCC 2007*, S. P. Vadhan, Ed., Lecture Notes in Computer Science, vol. 4392. Berlin, Germany: Springer, 2007, pp. 515–534.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] 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 Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5784–5798, Jun. 2020.

[6] W. Dai, S. Tuo, L. Yu, K.-K. R. Choo, D. Zou, and H. Jin, "HAPPS: A hidden attribute and privilege-protection data-sharing scheme with verifiability," *IEEE Internet Things J.*, vol. 9, no. 24, pp. 25538–25551, Dec. 2022.

[7] L. Wang, W. Ding, Z. Yan, S. Qiu, M. Wang, and Z. Wan, "EDDAC: An efficient and decentralized data access control scheme with attribute privacy preservation," *IEEE Internet Things J.*, vol. 11, no. 8, pp. 14579–14594, Apr. 2024.

[8] N. Wu, L. Xu, and L. Zhu, "A blockchain based access control scheme with hidden policy and attribute," *Future Gener. Comput. Syst.*, vol. 141, pp. 186–196, Apr. 2023.

[9] X. Ding, Y. Liu, J. Ning, and D. Chen, "Blockchain-enhanced anonymous data sharing scheme for 6G-enabled smart healthcare with distributed key generation and policy hiding," *IEEE J. Biomed. Health Inform.*, vol. 30, no. 1, pp. 666–675, Jan. 2026.

[10] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in *Applied Cryptography and Network Security–ACNS 2008*, S. M. Bellovin, R. Gennaro, A. Keromytis, and M. Yung, Eds., Lecture Notes in Computer Science, vol. 5037. Berlin, Germany: Springer, 2008, pp. 111–129.

[11] J. Lai, R. H. Deng, and Y. Li, "Fully secure ciphertext-policy hiding CP-ABE," in *Information Security Practice and Experience–ISPEC 2011*, F. Bao and J. Weng, Eds., Lecture Notes in Computer Science, vol. 6672. Berlin, Germany: Springer, 2011, pp. 24–39.

[12] L. Zhang, X. Li, Q. Wu, and F. Rezaeibagha, "Blockchain-aided anonymous traceable and revocable access control scheme with dynamic policy updating for the cloud IoT," IEEE Internet Things J., vol. 11, no. 1, pp. 526–542, Jan. 2024, doi: 10.1109/JIOT.2023.3287190.

[13] L. Zhang, S. Xie, Q. Wu, and F. Rezaeibagha, "Enhanced secure attribute-based dynamic data sharing scheme with efficient access policy hiding and policy updating for IoMT," IEEE Internet Things J., vol. 11, no. 16, pp. 27435–27447, Aug. 2024, doi: 10.1109/JIOT.2024.3399734.

[14] M. Li, Y. Zhang, C. Huang, H. Fang, and J. Shen, "Efficient attribute-based searchable proxy re-encryption with policy hiding over IoT data," IEEE Internet Things J., vol. 12, no. 24, pp. 52766–52780, 2025, doi: 10.1109/JIOT.2025.3613006.

[15] Z. Zhang, J. Zhang, Y. Yuan, and Z. Li, "An expressive fully policy-hidden ciphertext policy attribute-based encryption scheme with credible verification based on blockchain," *IEEE Internet Things J.*, vol. 9, no. 11, pp. 8681–8697, Jun. 2022.

[16] N. Liu, Y. Lei, W. Ren, L. Zhang, X. Zhang, T. Zhu, and G. Min, "zk-Guard: A privacy-preserving access control framework based on zk-SNARKs and blockchain for decentralized data sharing," *IEEE Trans. Dependable Secure Comput.*, early access, 2026, doi: 10.1109/TDSC.2026.3657485.

[17] D. Catalano and D. Fiore, "Vector commitments and their applications," in *Public-Key Cryptography–PKC 2013*, K. Kurosawa and G. Hanaoka, Eds., Lecture Notes in Computer Science, vol. 7778. Berlin, Germany: Springer, 2013, pp. 55–72.

[18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2018, pp. 315–334.

[19] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Advances in Cryptology–EUROCRYPT 2016*, Lecture Notes in Computer Science, vol. 9666. Berlin, Germany: Springer, 2016, pp. 327–357.

[20] J. Li, E. Zhang, J. Han, Y. Zhang, and J. Shen, "PH-MG-ABE: A flexible policy-hidden multigroup attribute-based encryption scheme for secure cloud storage," *IEEE Internet Things J.*, vol. 12, no. 2, pp. 2146–2157, 2025, doi: 10.1109/JIOT.2024.3468018.

[21] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology–CRYPTO 1991*, Lecture Notes in Computer Science, vol. 576. Berlin, Germany: Springer, 1992, pp. 129–140.

[22] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology–CRYPTO 1986*, Lecture Notes in Computer Science, vol. 263. Berlin, Germany: Springer, 1987, pp. 186–194.

[23] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[24] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS PUB 197, 2001, updated 2023.

[25] M. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, 2007.

[26] T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes," in *Advances in Cryptology–CRYPTO 1992*, Lecture Notes in Computer Science, vol. 740. Berlin, Germany: Springer, 1993, pp. 31–53.

[27] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Dept. Comput. Sci., ETH Zurich, Zurich, Switzerland, Tech. Rep. 260, 1997.