Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.061046

ARTICLE

Smart Contract-Aided Attribute-Based Signature Algorithm with Non-Monotonic Access Structures

Xin Xu[1,*], **Zhen Yang**[2] **and Yongfeng Huang**[1]

[1]Department of Electronic Engineering, Tsinghua University, Beijing, 100084, China
[2]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, 100876, China
*Corresponding Author: Xin Xu. Email: xuxin527@tsinghua.edu.cn

Received: 15 November 2024; Accepted: 06 March 2025; Published: 19 May 2025

ABSTRACT: Attribute-Based Signature (ABS) is a powerful cryptographic primitive that enables fine-grained access control in distributed systems. However, its high computational cost makes it unsuitable for resource-constrained environments, and traditional monotonic access structures are inadequate for handling increasingly complex access policies. In this paper, we propose a novel smart contract-assisted ABS (SC-ABS) algorithm that supports non-monotonic access structures, aiming to reduce client computing overhead while providing more expressive and flexible access control. The SC-ABS scheme extends the monotonic access structure by introducing the concept of negative attributes, allowing for more complex and dynamic access policies. By utilizing smart contracts, the algorithm supports distributed trusted assisted computation, and the computation code is transparent and auditable. Importantly, this design allows information about user attributes to be deployed on smart contracts for computation, both reducing the risk of privacy abuse by semi-honest servers and preventing malicious users from attribute concealment to forge signatures. We prove that SC-ABS satisfies unforgeability and anonymity under a random oracle model, and test the scheme's cost. Compared with existing schemes, this scheme has higher efficiency in client signature and authentication. This scheme reduces the computing burden of users, and the design of smart contracts improves the security of aided computing further, solves the problem of attribute concealment, and expresses a more flexible access structure. The solution enables permission control applications in resource-constrained distributed scenarios, such as the Internet of Things (IoT) and distributed version control systems, where data security and flexible access control are critical.

KEYWORDS: Attribute-based signature; non-monotone; smart contract

1 Introduction

With the growth of information technology, distributed file systems and information management have become essential components of modern data management. Systems like Git, vehicle networking, and the Internet of Things (IoT) require efficient data sharing and collaboration while ensuring information security and privacy. These systems face complex access control challenges. Traditional public key encryption, relying on certificates and key maintenance, is insufficient for addressing access control needs in distributed environments.

Attribute-based cryptography (ABC), including Attribute-Based Encryption (ABE) and Attribute-Based Signatures (ABS) [1], offers a novel solution for access control in distributed file systems by using attributes to define access policies. Xu et al. [2] proposed an access control scheme using ABE and ABS for read-write permissions on distributed version control systems. Hong et al. [3] proposed an attribute-based

online/offline signature scheme for mobile crowdsensing scenarios. Li et al. [4] and Patil et al. [5] proposed an access control scheme based on ABC for electronic health records. In the above application scenarios, ABS is often used to verify or manage permissions, especially for writing. In this framework, users' signing keys are linked to their attributes, and signatures are associated with specific access policies. Only entities whose attributes meet predefined conditions can generate valid signatures. This simplifies key distribution, reduces maintenance costs, and enhances flexibility, making it ideal for distributed environments.

In practical applications, several challenges hinder the effectiveness of existing solutions. Most ABS implementations rely on monotonic access structures, limiting their ability to express complex access control logic. Compared to traditional access control lists (ACL), ABS struggles to offer fine-grained revocation or permission adjustments at the individual attribute or user level. To better align with real-world needs, ABS must support non-monotonic access policies, such as AND, OR, NOT, and Threshold gates. These enhancements would improve policy expressiveness and enable more precise permission management, such as "Faculty AND Accounting AND (non-math department)". However, extending monotonic ABS to non-monotonic schemes is challenging. In non-monotonic designs, user key attributes are often independent, allowing malicious users to forge signatures by concealing certain attributes(such as "math department"), which poses a security risk. We refer to this type of forgery as "attribute concealment." Okamoto et al. propose an ABS supporting non-monotonic structures [6]. To avoid attribute concealment, the scheme requires that the key must contain information about all attributes in the attribute space, but its low verification efficiency makes it impractical.

This also highlights another critical challenge faced by ABS: ABS algorithms typically involve complex bilinear pairing operations, which pose significant challenges for resource-constrained devices. The introduction of non-monotonic access structures further increases this computational burden, conflicting with the lightweight computing demands of IoT devices. To address this, researchers have explored server-aided mechanisms [7–11] where remote servers take on most of the computational tasks. However, this approach raises new concerns about server trustworthiness. Although server assistance can reduce client-side computational load, it also introduces security and trust issues, as users must trust that servers will not leak sensitive information or tamper with computation results. Additionally, the trustworthiness of servers hinders the delegation of attribute-related computations in ABS. If the server is responsible for verifying the correctness of user attributes, it introduces the risk of compromising user attribute privacy, making challenges like attribute concealment difficult to address. Thus, the trust issue with external servers and the forgery risk of non-monotonic access structures present conflicting problems.

Therefore, developing ABS algorithms that support non-monotonic access policies while maintaining efficiency is an urgent research topic. This paper proposes a smart contract-aided ABS scheme (SC-ABS) that supports non-monotonic access policies, effectively addressing the conflict between server trust issues and attribute concealment. Our contributions are outlined below:

1. We present a new attribute-based signature scheme that supports non-monotonic access structures on attributes. Our scheme can handle any access structure that can be represented by a boolean formula involving AND, OR, NOT, and threshold operations.
2. Our scheme introduces a smart contract-aided computation mechanism in which the computation program is transparently recorded on the blockchain and automatically executed upon meeting predefined conditions. This ensures the security of intermediate results. Compared to traditional server-aided schemes, this approach is secure against collusion attacks and simultaneously reduces computational overhead for users.
3. We implemented SC-ABS, then tested its computational overhead and compared it with other server-assisted ABS algorithms. Additionally, we conducted a comprehensive security analysis. SC-ABS is

more suitable for data security protection and permission control in the distributed scenario of limited resources.

2 Related Works

2.1 Attribute-Based Signature

Attribute-based signatures (ABS), first proposed in [1], ensure that only entities whose attributes satisfy the access policy can generate valid signatures. The most widely used existing ABS schemes are monotonic access structures such as the schemes of Maji et al. [1,12]. Okamoto et al. [6] present a fully secure ABS scheme in the standard model that supports general non-monotonic predicates. However, this scheme requires a fixed classification for attributes, and users must possess all attribute types, resulting in low efficiency due to the need for multiple bilinear pairings, making it impractical.

The computational overhead of ABS is a significant barrier to its widespread use. To address this, several server-aided ABS schemes have been proposed. Cui et al. [7] introduced server-aided ABS, transferring the signature and verification work to the server to reduce overhead. However, Xiong et al. [13] demonstrated that Cui et al.'s scheme lacks collusion security, and their scheme had correctness issues. Huang et al. [8] proposed a secure server-aided ABS that meets stronger security definitions. Chen et al. [9] used an access tree structure and delegated verification to the server, while Li et al.'s scheme [10] only supports threshold-based access strategies. These server-aided ABS schemes typically face server trust issues and do not support non-monotonic access policies.

ABS supporting non-monotonic access structures offers more expressive access control strategies, but it also incurs high computational overhead and poses a risk of attribute concealment. Current server-aided ABS are limited by server trust and struggle to address attribute concealment. To address these challenges, this paper proposes a smart contract-based trusted aided ABS algorithm that incorporates a non-monotonic access structure.

2.2 Smart Contract

Smart contracts, introduced by Nick Szabo in 1994, are "computerized transaction protocols that execute the terms of a contract" [14]. They aim to fulfill common contractual conditions without the need for intermediaries [14,15]. Smart contracts are facilitated by blockchain platforms like Ethereum. The scheme presented in this paper leverages Ethereum [16] for the design and deployment of smart contracts aimed at assisted computing. As a widely deployed blockchain, Ethereum enables developers to create smart contracts using languages like Solidity and Vyper, compile them, and deploy them. To deploy a smart contract, the owner initiates a transaction, creating a contract account. Any Ethereum account interacts with the smart contract through the contract account. When an Ethereum account invokes a function to update data stored on the Ethereum network, it submits a corresponding transaction, which is recorded on the Ethereum network.

Advantages of Smart Contract Assistance. Smart contracts aided ABS is different from server-aided ABS, and blockchain-based smart contracts have more advantages than traditional third-party servers.

- Distributed services: Smart contracts, deployed on the blockchain, provide distributed and decentralized services, mitigating the single point of failure risk faced by traditional third-party servers.
- Reliability: Once issued, a smart contract cannot be tampered with. It is automatically executed when conditions are met, and no party can alter the result. Multiple independent nodes process and verify the contract logic, effectively preventing manipulation and ensuring timely execution according to specified terms. This makes smart contract results more reliable than those from traditional servers.

- Security: Smart contracts function as digital protocols, with open and transparent logic that can be reviewed by all users. As long as the code logic is correct, the stored information cannot be maliciously exploited, and the contract cannot perform actions outside the agreement, such as colluding with malicious users. In comparison to traditional third-party servers, smart contracts offer greater security for aided services.

3 Preliminaries

First, we provide the definition of an access structure. Then we give background information on bilinear maps and our cryptographic assumption. Finally, we give some background on linear secret-sharing schemes.

Definition 1: Access Structure [17]: Let $\mathscr{P} = \{P_1, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathscr{P}}$ is monotonic. For $\forall B$ and C, if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. A monotonic access structure is a monotonic collection $\mathbb{A}$ for non-empty subsets of $\{P_1, \ldots, P_n\}$, namely $\mathbb{A} \subseteq 2^{\mathscr{P}} \backslash \emptyset$. The sets in $\mathbb{A}$ are considered authorized sets, while those not in $\mathbb{A}$ are unauthorized sets.

3.1 Bilinear Maps

The ABS algorithms use the idea of bilinear pairs based on elliptic curves. Let $\mathbb{G}_0$ be a cyclic group of prime order p generated by g. Let $\mathbb{G}_1$ be a group of order p. We can define e as a bilinear map, $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. The bilinear map e has the following properties:

1. Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$.
2. Non-degeneracy: $e(g, g) \neq 1$.

3.2 Security Model

The security model is defined as a game between a polynomial-time adversary A and a challenger C, where the challenger C interacts with a smart contract to perform its operations.

Setup: Adversary A allocates the target attribute set S^* and forwards it to C. Challenger C generates public parameters PK and master key MSK, and then returns PK to A.

Query: The adversary A adaptively issues the following query to C.

- Key extraction query: The adversary A selects the access policy Γ and additional attributes ρ. The challenger C then executes the key generation algorithm to produce the corresponding partial signature key psk and the user signing key usk for A.
- Signature query: The adversary A selects a message m and sends a request to the challenger C. The smart contract then generates a partial signature based on the attribute set S associated with A's account. If the attribute set S satisfies the access policy, the smart contract completes the signature generation process, producing the final signature σ, which is returned to A.
- User signature verification query: The adversary A has the signature σ of the message m under the attribute set S. After receiving the signature, C applies the signature extraction algorithm to produce a converted signature $\hat{\sigma}$, which is sent back to A. A computes an intermediate signature $\widetilde{\sigma_1}$ and returns it to C. Finally, C performs the user signature verification and returns the verification result.

Signature forgery: A generates a forged signature σ^* for a message m^* under an attribute set S^*. A is considered successful if the following conditions are met:

- A sends σ^* to C. C generates a converted signature $\hat{\sigma^*}$ and returns it to A. A executes the aided signature verification algorithm to compute an intermediate signature $\widetilde{\sigma^*}$, which is sent to C, and C returns the verification correct.

- Adversary A has never issued a signature query for message m^* and attribute set S^*.

3.3 Linear Secret-Sharing Schemes

First, we introduce the design of a non-monotonic access policy based on the Linear Secret-Sharing Scheme (LSSS).

Definition 2: Linear Secret-Sharing Schemes [18]: Let $\mathscr{P}$ be the set of all attributes, L be a matrix of $l \times m$, and let $\pi : \{1, \ldots, l\} \to \mathscr{P}$ be a function that maps each row in the matrix to an attribute. A linear secret-sharing scheme (LSSS) on $\mathbb{Z}_p$ is a secret-sharing scheme π on an access policy $\mathbb{A}$ based on the attribute set $\mathscr{P}$. The linear secret-sharing scheme consists of two efficient algorithms:

$\mathsf{Share}_{L,\pi}$: Input $s \in \mathbb{Z}_p$ as the shared secret, choose $s_2, \ldots, s_m \in \mathbb{Z}_p$, let $\mathbf{s} = (s, s_2, \ldots, s_m)$. For each attribute $\pi(i)$, compute its share $\lambda_i = \langle \mathbf{L}_i \cdot \mathbf{s} \rangle$, where $\mathbf{L}_i$ is the ith row of L. Finally, output l shares $L \cdot s$.

$\mathsf{Recon}_{L,\pi}$: Input an authorization set $S \in \mathbb{A}$, let $I = \{i \mid \pi(i) \in S\}$. The output is a set of constants $(i, \mu_i)_{i \in I}$, which satisfies the linear reconstruction property $\sum_{i \in I} \mu_i \lambda_i = s$.

Non-Monotonic Access Structures. We refer to the method proposed by [18] to transform the monotonic access policy into a non-monotonic access policy. Assume a linear secret-sharing scheme family $\{\prod_{\mathbb{A}}\}_{\mathbb{A} \in \mathscr{AS}}$. For each access policy $\mathbb{A} \in \mathscr{AS}$, the attribute set $\mathscr{P}$ covered by it has the following properties: There are two types of attribute names in $\mathscr{P}$, positive attributes (such as "English department", recorded as x) and negative attributes (such as "non-English department", recorded as x'). If $x \in \mathscr{P}$, then $x' \in \mathscr{P}$, and vice versa. Positive attributes and negative attributes are conceptually related.

Definition 3: non-monotonic access policy family $\mathscr{AS}$: For each access policy $\mathbb{A} \in \mathscr{AS}$, the attribute set it covers is $\mathscr{P}$. Define a possible non-monotonic access policy $NM(\mathbb{A})$ where the attribute set it covers is $\widetilde{\mathscr{P}}$, the set of all positive attributes in $\mathscr{P}$. For each set $\widetilde{S} \subset \widetilde{\mathscr{P}}$, define $N(\widetilde{S}) = \widetilde{S} \cup \{x' \mid x \in \widetilde{\mathscr{P}} \backslash \widetilde{S}\}$. If and only if $N(\widetilde{S})$ is an authorized set in $\mathbb{A}$, $\widetilde{S}$ is an authorized set in $NM(\mathbb{A})$. For each authorization set $X \in NM(\mathbb{A})$, there is a set in $\mathbb{A}$ that contains all attributes in X and all negative attributes not in X.

4 Scheme Construction

We designed a smart contract-aided attribute-based signature algorithm with non-monotonic access structures (SC-ABS). The algorithm involves three entities: attribute authorities, smart contracts, and users. As shown in Fig. 1, the attribute authority is responsible for key initialization and issuance, while the smart contract assists with calculations that users can sign and verify as needed. Compared to traditional ABS, our algorithm supports non-monotonic access strategies. Additionally, it introduces smart contracts as trusted, autonomous computing agents that execute programs automatically upon invocation, thereby eliminating the need for third-party involvement.

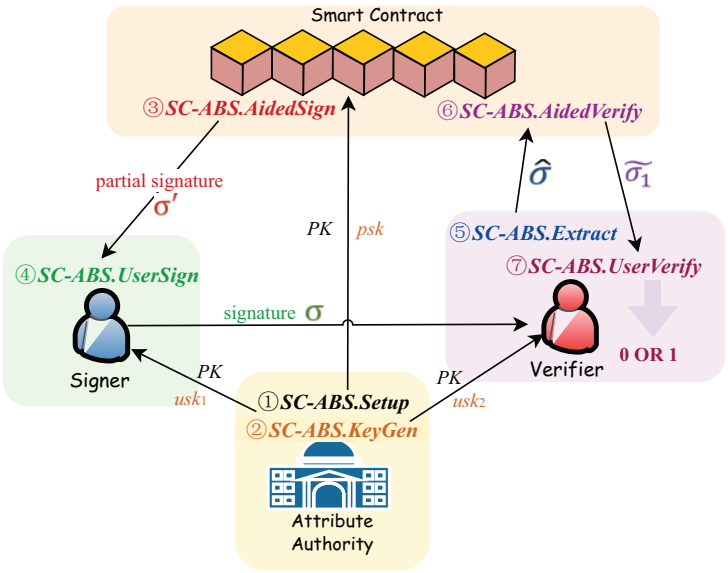

Figure 1: Algorithm framework of SC-ABS

4.1 Smart Contract Aided ABS

As shown in the Fig. 1, referring to Xiong et al., we designed the SC-ABS algorithm as follows, of which the meaning of symbols is shown in Table 1:

Table 1: Mathematical symbols table

Symbol	Description
$\mathbb{G}, \mathbb{G}_T$	A cyclic group.
PK	Public parameters.
SK	The master key.
$\widetilde{\mathbb{A}}, \mathbb{A}$	The access policies.
a_u	The ID of user u.
S_u	The attribute set of user u.
psk	The key for the smart contract.
usk	The key for the user.
σ	The signature.

SC-ABS.Setup(λ): The algorithm, executed by the attribute authority, takes the security parameter λ as input and then generates the system parameters and the master key. Divided into four steps:

1. Select the multiplicative cyclic group $\mathbb{G}, \mathbb{G}_T$ of prime order $p > 2^{\lambda}$, and define the bilinear pair $e: \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$.
2. Randomly select $r_0, u_0, u_1, \ldots, u_n \in \mathbb{G}$. Define the collision-resistant hash function $H(m) = u_0 \prod_{j=1}^{n} u_j^{m[j]}$, where $m[j]$ represents the message string The jth position of m.
3. Set the attribute universe $U = 1, 2, \ldots, l$, and for each attribute $i \in U$, randomly select $r_i \in \mathbb{G}$. Define an additional attribute ρ and randomly select $r_\rho \in \mathbb{G}$.

4. Randomly select $\alpha_1, \alpha_2 \in \mathbb{Z}_p$ and generator $g \in \mathbb{G}$, and calculate $Z = e(g,g)^{\alpha_1+\alpha_2}$.

Then, the system's public parameters are denoted as $PK = \{g, e, \mathbb{G}, \mathbb{G}_T, H, Z, r_0, \{r_i\}_{i\in U\cup\rho}, u_0, \ldots, u_n\}$, the master key is $SK = \{\alpha_1, \alpha_2\}$. PK is published to both smart contracts and all users, and SK is protected by the attribute authority.

SC-ABS.KeyGen($PK, SK, \widetilde{\mathbb{A}}, a_u, S_u$): The algorithm is executed by the attribute authority. It uses the public parameters PK, master key SK, access policy $\widetilde{\mathbb{A}}$, user ID a_u, and its attribute set S_u as inpu. Based on a linear secret-sharing scheme that supports non-monotonic access policies, the algorithm generates partial signature keys and user signature keys. The algorithm is divided into the following steps:

1. The non-monotonic access policy $\widetilde{\mathbb{A}}$ and a certain monotonic access policy $\mathbb{A}$ satisfy $\widetilde{\mathbb{A}} = NM(\mathbb{A})$. The non-monotonic access policy $\widetilde{\mathbb{A}}$ covers the attribute set $P \subseteq U$, then the monotonic access policy $\mathbb{A}$ covers the attribute set $\mathscr{P}$, $\mathscr{P}$ contains the attributes in P and the corresponding negative attributes. $\mathbb{A}$ can be related to the linear secret-sharing scheme (L, π), where L is a matrix of $l \times m$. For each row $k \in [1, l]$ in L, $\pi(k)$ maps to an attribute in $\mathscr{P}$.
2. Randomly select $v_2, v_3, \ldots, v_m \in \mathbb{Z}_p$, and define vector $\vec{v} = \{\alpha_1, v_2, \ldots, v_m\}$, calculate $\varphi_{\pi(k)} = \vec{L_k} \cdot \vec{v}$, where $\vec{L_k}$ represents the kth row of matrix L.
3. For each row $k \in [1, l]$ in L, randomly select $x_k \in \mathbb{Z}_p$. The user has a unique ID value a_u. For the additional attribute ρ, the user also selects a random value $x_{\rho,a_u} \in \mathbb{Z}_p$. This random value x_{ρ,a_u} is associated with the user ID and the attribute ρ.
4. Compute $d_k = g^{\varphi_{\pi(k)}}(r_0 \cdot r_{\pi(k)})^{x_k}, d'_k = g^{x_k}, d''_k = \left\{r_i^{x_k}\right\}_{i\in U\cup\rho, i\neq\pi(k)}$.
5. Similarly, calculate $d_{a_u} = g^{\alpha_2}(r_0 \prod_{i\in S_u\cup\rho} r_i)^{x_{\rho,a_u}}, d'_{a_u} = g^{x_{\rho,a_u}}$.

Then, based on the linear secret-sharing scheme (L, π), the partial signature key $psk = \left\{d_k, d'_k, d''_k\right\}_{k\in[1,l]}$ and S_u is assigned to the smart contract, and the user signature key is $usk = \left\{d_{a_u}, d'_{a_u}\right\}$. To protect the security of psk and S_u, we use the eWEB scheme [19], which is a blockchain encryption scheme using dynamic active secret sharing. eWEB securely stores encoded secrets, relying on an honest majority of dynamic point sets and granting access only to authorized users meeting the creator's criteria. This ensures secure storage of private information and prevents unauthorized access to smart contract data.

SC-ABS.AidedSign($H(m), psk, \widetilde{\mathbb{A}}, a_u, S_u$): This algorithm will be executed by the smart contract when the user initiates an aided signature request. With message digest $H(m)$, public parameters PK, access policy $\widetilde{\mathbb{A}}$, signer ID a_u and its attribute set S_u as input, the smart contract calculates the output partial signature. The algorithm is divided into the following steps:

1. Confirm whether the attribute set S_u of the signer a_u satisfies the access policy. If so, define $K = \{k\}_{\pi(k)\in S_u}$ and optionally a set of constants $\{\omega_k\} \in \mathbb{Z}_p$, satisfying $\sum_{k\in K}\omega_k\varphi_{\pi(k)} = \alpha_1$.
2. Randomly select $\eta, \zeta, s \in \mathbb{Z}_p$, based on the current time t, select $\beta \in \mathbb{Z}_p$ for the time point in the same period, and calculate:

$$\begin{cases} \sigma'_0 &= \prod_{k\in K}(d_k \cdot \prod_{i\in S_u\cup\rho, i\neq\pi(k)} d''^{\omega_k}_{k,i}) \cdot (r_0 \prod_{i\in S_u\cup\rho} r_i)^{\eta} \cdot H(m)^{\zeta} \cdot H(t)^{a_u+s}, \\ \sigma'_1 &= g^{\eta}\prod_{k\in K}(d'_k)^{\omega_k}, \\ \sigma'_2 &= g^{\zeta}, \sigma'_3 = g^{s}, \sigma'_4 = (r_0 \prod_{i\in S_u\cup\rho} r_i \cdot H(t))^{\frac{1}{\beta}}. \end{cases} \tag{1}$$

Then the partial signature generated by the smart contract for the signer a_u based on the message m is $\sigma' = \{\sigma'_0, \sigma'_1, \sigma'_2, \sigma'_3, \sigma'_4, H(m)\}$.

SC-ABS.UserSign(usk, σ', m, t): The algorithm will be executed by the user after he receives a partial signature returned by the smart contract. With the user signature key usk, partial signature σ', message m, and time t as input, the user calculates the final signature. Firstly, the user randomly selects $x'_{\rho,a_u}, \zeta', s' \in \mathbb{Z}_p$, and computes:

$$\begin{cases} \sigma_0 = \sigma'_0 \cdot d_{a_u} \cdot (r_0 \prod_{i \in S_u \cup \rho} r_i)^{x'_{\rho,a_u}} \cdot H(m)^{\zeta'} \cdot H(t)^{s'}, \\ \sigma_1 = \sigma'_1 \cdot g^{x'_{\rho,a_u}} \cdot d'_{a_u}, \\ \sigma_2 = \sigma'_2 \cdot g^{\zeta'}, \sigma_3 = \sigma'_3 \cdot g^{a_u + s'}, \sigma_4 = \sigma'_4. \end{cases} \tag{2}$$

Finally, the user gets the signature of message m based on his own attribute set S_u: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, m, t\}$.

SC-ABS.Extract(σ): The algorithm is executed by the user verifying the signature. After extracting the signature, the result is sent to the smart contract for assisted verification. The user selects $\{\sigma_0, \sigma_1, \sigma_3, \sigma_4, t\}$ in signature σ, randomly selects $\delta \in \mathbb{Z}_p$, and calculates $\hat{\sigma}_0 = \sigma_0^\delta, \hat{\sigma}_1 = \sigma_1^\delta, \hat{\sigma}_3 = \sigma_3^\delta, \hat{\sigma}_4 = \sigma_4^\delta$ as a result of extraction $\hat{\sigma} = \{\hat{\sigma}_0, \hat{\sigma}_1, \hat{\sigma}_3, \hat{\sigma}_4, t\}$.

SC-ABS.AidedVerify($\hat{\sigma}$): The smart contract accepts the extraction result $\hat{\sigma}$ and calls the algorithm to perform aided signature verification. The specific calculation is as follows:

$$\frac{e(\hat{\sigma}_0, g)}{e(\hat{\sigma}_1, \frac{(\hat{\sigma}_4)^\beta}{H(t)}) e(\hat{\sigma}_3, H(t))} = \widetilde{\sigma_1}. \tag{3}$$

The smart contract returns $\widetilde{\sigma_1}$ to the user who needs to verify the signature.

SC-ABS.UserVerify($\sigma, \widetilde{\sigma_1}$): The user receives the verification result $\widetilde{\sigma_1}$ provided by the smart contract and will perform the final calculation:

$$\widetilde{\sigma_2} = e(\sigma_2^\delta, H(m)) \cdot Z^\delta. \tag{4}$$

Determine whether $\widetilde{\sigma_1}$ is equal to $\widetilde{\sigma_2}$. If they are equal, the signature verification is successful, and the system outputs 1. Otherwise, signature verification fails, and the system outputs 0.

In SC-ABS, the smart contract is responsible for the auxiliary calculation of signature and verification. After completing the local calculation, the user initiates a calculation request to the smart contract in the form of publishing a transaction, and the smart contract verifies the user identity, performs the calculation, and returns the calculation result.

4.2 Proof of Correctness

Theorem 1: SC-ABS is correct if for all $\lambda \in N$:

$$\Pr\left[\begin{array}{ll} PK, SK & \leftarrow Setup(\lambda); \\ psk, usk & \leftarrow KeyGen(PK, SK, \widetilde{\mathbb{A}}, a_u, S_u); \\ \sigma' & \leftarrow AidedSign(H(m), psk, \widetilde{\mathbb{A}}, a_u, S_u) \\ \sigma & \leftarrow UserSign(usk, \sigma', m, t) \end{array} : UserVerify(\sigma, AidedVerify(Extract(\sigma))) = 1\right] = 1. \tag{5}$$

Proof Proving the correctness of SC-ABS requires showing the equivalence between $\widetilde{\sigma_1}$ and $\widetilde{\sigma_2}$. In the signature extraction conversion process, the δ parameters can be effectively offset against each other. As a

result, there is no need to include the δ exponent in subsequent derivations. We first derive the result of $\widetilde{\sigma_1}$: Let $\sum_{k\in K}\omega_k\varphi_{\pi(k)} = p_1, \sum_{k\in K}\omega_k x_k = p_2, r_0\prod_{i\in S_u\cup\rho} r_i = R, \eta + x_{\rho,a_u} + x'_{\rho,a_u} = X, a_u + s + s' = S$, obviously $p_1 = \alpha_1$.

$$\begin{cases} e(\sigma_0, g) = e(g^{p_1}\cdot(R)^{p_2+X}\cdot H(m)^{\zeta+\zeta'}\cdot H(t)^S\cdot g^{\alpha_2}, g) \\ \qquad\quad = e(g^{\alpha_1+\alpha_2}, g)\cdot e((R)^{p_2+X}, g)\cdot e(H(m)^{\zeta+\zeta'}, g)\cdot e(H(t)^S, g); \\ e\left(\sigma_1, \dfrac{(\sigma_4)^\beta}{H(t)}\right) = e\left(g^{p_2+X}, \dfrac{R\cdot H(t)}{H(t)}\right) = e(g, (R)^{p_2+X}); \\ e(\sigma_3, H(t)) = e(g^S, H(t)). \end{cases} \tag{6}$$

Therefore, we can deduce:

$$\begin{aligned} \widetilde{\sigma_1} &= \frac{e(\sigma_0, g)}{e(\sigma_1, \frac{(\sigma_4)^\beta}{H(t)})e(\sigma_3, H(t))} = \frac{e(g^{\alpha_1+\alpha_2}, g)\cdot e((R)^{p_2+X}, g)\cdot e(H(m)^{\zeta+\zeta'}, g)\cdot e(H(t)^S, g)}{e(g, (R)^{p_2+X})e(g^S, H(t))} \\ &= e(g^{\alpha_1+\alpha_2}, g)e(H(m)^{\zeta+\zeta'}, g). \end{aligned} \tag{7}$$

For $\widetilde{\sigma_2}$, we can derive:

$$\begin{aligned} \widetilde{\sigma_2} &= e(\sigma_2, H(m))\cdot Z \\ &= e(g^{\zeta+\zeta'}, H(m))e(g, g)^{\alpha_1+\alpha_2}. \end{aligned} \tag{8}$$

Therefore, $\widetilde{\sigma_1}$ and $\widetilde{\sigma_2}$ are equivalent. □

5 Security Analysis

This section is the analysis of SC-ABS security. We will analyze the security of the algorithm and smart contracts.

5.1 Unforgeability and Anonymity

We refer to Waters et al.'s [20] definitions and demonstrate in the appendix that SC-ABS satisfies unforgeability and anonymity under the random oracle model.

In SC-ABS, the smart contract manages aided signing and verification. During signing, the signer initiates a transaction, and the smart contract determines their attribute information based on the account, enhancing security. SC-ABS relies not only on the difficulty of bilinear pairing problems but also on the trustworthiness of the smart contract. During aided signing, when an adversary requests a signature, the challenger provides the attribute set and signature for the message. However, users cannot specify their attributes in the smart contract signing phase; it strictly adheres to the rules and uses the set linked to the user's account. Therefore, as long as the user's account is secure, attackers cannot query a signature for any attribute set.

5.2 Smart Contract Security

By introducing the smart contract, the computational process can be transparently displayed on the blockchain, and all user transactions are immutable. This ensures that the entire SC-ABS system is auditable and traceable. Additionally, implementing an audit penalty mechanism in practice can effectively raise the cost of malicious attacks.

This section will focus on analyzing the security of the data stored in the smart contract on the blockchain, specifically addressing the variables security, the intermediate results security, and collusion security.

Data Stored by Smart Contracts. The data stored by the smart contract on the blockchain can be classified into two categories:

1. Variables in the Ethereum Virtual Machine (EVM) environment: public parameters PK, the partial signature key psk, and the user attribute set S_u.
2. Transactions on the blockchain: intermediate results generated by the smart contract, namely partial signature σ' and aided verification result $\widetilde{\sigma_1}$.

5.2.1 Variables Security

For the stored variables, PK is the public parameter, accessible to all participants as part of the system. Since PK does not introduce any privacy risks, it can be stored in plaintext without concerns for data security. In contrast, psk and S_u involve key and user attribute information, which must be securely stored. By utilizing the eWEB scheme, we can securely encrypt and store psk and S_u on the blockchain. Only the smart contract, when executing according to predefined logic, can access the relevant parameters. Even if a malicious user attempts to monitor the execution of the smart contract by tracking its memory, this memory data is temporary and not synced to the blockchain. Therefore, malicious users cannot monitor memory data from other nodes running the smart contract.

5.2.2 Intermediate Result Security

The intermediate result generated by the smart contract contains a partial signature σ' and an aided verification result $\widetilde{\sigma_1}$, which is stored in plaintext on the blockchain as part of the transaction. Concerns may arise about whether these intermediate results could lead to information leakage or facilitate collusion. Therefore, it is essential to analyze the security of these intermediate results, focusing on protection against man-in-the-middle attacks and replay attacks.

1. **Man-in-the-middle attack:** Suppose there is a middleman a'_u who obtains the intermediate result of the signature applied by other users a_u, but because the signature σ'_0 contains the user's unique ID information a_u, the middleman cannot generate σ_3 that also contains a_u, so the intermediate result cannot be used.
2. **Replay attack:** Assume that user r_u once had written permission to the file, but after the permission was revoked later, it no longer satisfies the access policy. The access policy of the file has not changed, and then user r_u attempts to use the signature intermediate results obtained in the past to generate a signature that satisfies the access policy. σ' contains the time information $H(t)$ of the aided signing and the corresponding value β of the time period. Suppose the user uses the intermediate result at time t to forge a signature at time t', then when the smart contract verifies the signature, there is:

$$\frac{e(\hat{\sigma_0}, g)}{e\left(\hat{\sigma_1}, \frac{(\hat{\sigma_4})^{\beta'}}{H(t')}\right) e(\hat{\sigma_3}, H(t'))}$$

Since $H(t)$ and β are used in $\hat{\sigma_0}$ and $\hat{\sigma_4}$, the aided signature verification result cannot pass SC-ABS. UserVerify. Therefore, signature forging cannot succeed.

5.2.3 Collusion Security Analysis

In SC-ABS, collusion attacks among multiple users occur when users with attribute sets that don't satisfy the access policy try to generate valid signatures by combining their signature keys. Since the full signature is generated from partial signatures that contain attributes of the user, conspirators must merge their partial signatures. The σ_0' is tied not only to the user's attribute set but also to their ID information a_u. Additionally, each user's attribute set is randomized with different random numbers, and parameters related to the additional attribute ρ vary across users. Therefore, users whose attribute sets do not meet the access policy cannot combine their partial signatures to generate valid ones, and α_1 cannot be recovered.

In collusion attacks between signers and smart contracts, all transactions are stored on the blockchain, ensuring that intermediate signatures and aided verification results are auditable and non-repudiable. Smart contracts issued by trusted attribute authorities are deployed as open, transparent programs on the blockchain, executing automatically when specific conditions are met. Unlike third-party servers with opaque logic, participants can review the code's logic and security, significantly reducing the risk of malicious behavior. A correct smart contract generates intermediate signatures only based on the attribute set associated with the user account that initiated the request. In the signature verification algorithm, the smart contract cannot access the message m, and any malicious attempt to upload a message will be rejected due to format non-compliance. As transparent, self-executing programs issued by trusted institutions, rather than third-party institutions or individuals, the smart contract is secure against collusion attacks.

In summary, the use of smart contracts for aided computation is secure against collusion attacks. Additionally, the information stored on the blockchain is secure under the design of this solution, effectively protecting against privacy threats.

6 Performance

In this section, we conduct a comprehensive evaluation of the SC-ABS scheme. This includes analyzing its storage, computing, and communication overhead; comparing its computing overhead with that of other server-assisted ABS algorithms; and performing a functional comparison with related algorithms.

We implemented the SC-ABS algorithm based on the PBC (Pairing-Based Cryptography) library, and our experiments were conducted using C on a system with an Intel Core CPU (1.60 GHz, 2 GB RAM). We tested the computing time overhead of the smart contract and the client respectively. We set the size of the attribute space to 10 and the number of attributes involved in the access policy to 5, for a total of 100 times. The average time of smart contract-aided signature is 0.0082 s, the average time of user signature is 0.0044 s, the average time of smart contract-aided verification is 0.0034 s, and the average time of user signature extraction and verification is 0.006 s, which the verification time is 0.0012 s.

Let d be the size of the default attribute set, a the user's attribute set size, a' the minimum attributes required for access, and l the number of attributes in the policy. In the SC-ABS scheme, we minimize client-side computational overhead. Specifically, the smart contract signature involves $2a' + 7$ exponentiations ($E_{\mathbb{G}}$) over the group $\mathbb{G}$, while user signing requires only 6 exponentiations over $\mathbb{G}$. The aided verification process entails merely 3 pairing operations (P) performed by the smart contract over $\mathbb{G}$. Additionally, the user-end verification cost is relatively low, consisting of one $E_{\mathbb{G}}$, one exponentiation ($E_{\mathbb{G}_T}$) over the group $\mathbb{G}_T$, and one P.

As shown in Table 2, we compare our SC-ABS scheme with other 5 server-aided attribute-based signature (SAABS) schemes [7–11]. Compared to other SAABS, SC-ABS features the shortest user secret key length and incurs the least user-end signing cost. The aided verification cost is also minimal, involving just one additional pairing operation compared to the two pairings of SA-ABSR [7]. Moreover, the user-end

verification cost is comparable to that of most SAABS schemes. For aided signing costs, when $l \geq a' + 4$, SC-ABS exhibits the lowest computational cost, which is advantageous as access policies become increasingly complex. Moreover, the signature length of SC-ABS is shorter than that of ABSAVS [9], DABSAS [10], and SAABS-CR [11] but slightly longer than SA-ABSR and InSAABS. This is due to the introduction of time-dependent information to implement replay protection in distributed file systems. On the other hand, we also compare SC-ABS with another non-aided non-monotonic ABS [6]: while SC-ABS costs 4 pairing operations during verification, the latter algorithm costs $l + 3$ pairing operations during verification. Thus SC-ABS proves to be efficient on algorithm and further more efficient with smart contract aiding.

Table 2: Performance comparison of SC-ABS and other server-aided ABS

Schemes	SA-ABSR [7]	InSAABS [8]	ABSAVS [9]	DABSAS [10]	SAABS-CR [11]	SC-ABS
usk.Size	$(2d + 2)(a + d)\|\mathbb{G}\|$	$2\|\mathbb{G}\|$	$(1 + 2n)\|\mathbb{G}\|$	$n(n-1)\|\mathbb{G}\|$	$(a + 2)\|\mathbb{G}\|$	$2\|\mathbb{G}\|$
Sig.Size	$3\|\mathbb{G}\|$	$4\|\mathbb{G}\|$	$(2l + 2)\|\mathbb{G}\|$	$(l + 2)\|\mathbb{G}\|$	$(a' + 3)\|\mathbb{G}\|$	$5\|\mathbb{G}\|$
SSig.Cost	$(6d + 5)E_{\mathbb{G}}$	$2lE_{\mathbb{G}}$	–	$(2l + 1)E_{\mathbb{G}}$	–	$(2a' + 7)E_{\mathbb{G}}$
USig.Cost	$(2d + 2)E_{\mathbb{G}}$	$8E_{\mathbb{G}}$	$(2 + 2l)E_{\mathbb{G}}$	$(2d + 1)E_{\mathbb{G}}$	$(a' + 3)E_{\mathbb{G}}$	$6E_{\mathbb{G}}$
SVer.Cost	$(2d + 1)E_{\mathbb{G}} + 3P$	$2P$	$(l-1)E_{\mathbb{G}_T} + 2\|K\|P$	$(l + 1)P$	$a'E_{\mathbb{G}} + (a' + 1)P$	$3P$
UVer.Cost	P	$E_{\mathbb{G}_T} + 2P$	$nE_{\mathbb{G}} + E_{\mathbb{G}_T} + P$	$E_{\mathbb{G}} + E_{\mathbb{G}_T} + P$	$E_{\mathbb{G}} + P$	$E_{\mathbb{G}} + E_{\mathbb{G}_T} + P$

We also compared the security features of the above six schemes, as shown in Table 3. Only SC-ABS employs the most expressive non-monotonic LSSS access structure, which supports both fine-grained attribute revocation and replay resistance. While SA-ABSR supports coarse-grained attribute revocation, it is vulnerable to collusion attacks; InSAABS and SAABS-CR utilize more expressive LSSS access structures but do not support revocation functionality. The other five SAABS fail to protect against malicious user replay attacks on signatures in distributed systems (such as Git). Therefore, compared with other SAABS, SC-ABS supports a richer set of security features without increasing computational overhead, making it better suited for the requirements of distributed systems.

Table 3: Function comparison of SC-ABS and other server-aided ABS

Schemes	SA-ABSR [7]	InSAABS [8]	ABSAVS [9]	DABSAS [10]	SAABS-CR [11]	SC-ABS
Predicate	Threshold	LSSS	Tree	Threshold	LSSS	Non-monotonic LSSS
Revocation	Coarse	×	×	×	×	Fine
Anti-replay	×	×	×	×	×	✓

7 Conclusion

In this paper, we propose a smart contract-aided ABS scheme that supports a non-monotonic access structure. We use negative attributes to extend the monotonic access structure to the non-monotonic access structure. Based on the distributed and reliable characteristics of smart contracts, we perform auxiliary computations to reduce the computing cost of users and prevent users from forging signatures by attribute concealment. We prove that SC-ABS satisfies unforgeability and anonymity under a random oracle model, and evaluate its computational performance. Experimental results show that compared with other server-assisted schemes, our scheme can improve security and reduce the computing cost of the client.

In the future, we plan to apply this solution to popular distributed file systems such as Git and design fine-grained user permission granting and revocation schemes based on this approach. The proposed scheme will require users to have closer interactions with blockchain technology. However, it is also essential to consider whether the introduction of smart contracts might introduce additional security risks and overheads. Evaluating and mitigating these potential issues will be a key focus of our further research.

Acknowledgement: This work was supported by National Natural Science Foundation of China. The authors wish to thank anonymous reviewers for their valuable comments and suggestions that improved this paper.

Funding Statement: The specific funding Grant Number is No. 82090053.

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Xin Xu, Zhen Yang; analysis and interpretation of results: Xin Xu; draft manuscript preparation: Xin Xu, Yongfeng Huang. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, Xin Xu, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

Appendix A SC-ABS Security Proofs

We analyze the security of the SC-ABS algorithm. The security analysis of the ABS algorithm usually includes two points: unforgeability and anonymity.

Appendix A.1 Unforgeability

Unforgeability means that a valid signature can only be generated using a partial signing key and the corresponding user signing key.

Theorem A1: If there exists a polynomial-time adversary A that wins the above game with non-negligible probability ε under a chosen message attack, it means that a polynomial-time challenger C can be built, with non-negligible probability $\varepsilon' \geq \frac{\varepsilon}{q_s(n+1))}$ solves the q-DHE problem, where q_s represents the number of allowed signature queries, and n represents the bit length of message m.

Proof: Assuming that under the polynomial time algorithm A can destroy the proposed solution with a non-negligible probability ε, then there is a polynomial time challenger C: given $g, g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q}$, the challenger can calculate $g_{q+1} = g^{a^{q+1}}$ through the following steps.

Initialization: Challenger C manages an attribute domain $U = 1, 2, \ldots, q$, and the adversary specifies a target attribute set S^* and forwards it to C. C randomly selects $\alpha_1', \alpha_2', a \in \mathbb{Z}_p$, and defines $Z = e(g_1, g_q) \cdot e(g, g)^{\alpha_1' + \alpha_2'}$, implicitly defined $\alpha_1 = \alpha_1' + a^{q+1}, \alpha_2 = \alpha_2'$, for simplicity, we set $g_i = g^{a^i}$ here. C

selects $y_0, \{y_i\}_{i\in U} \in \mathbb{Z}_p$, and calculates $r_0 = g^{y_0}\prod_{i\in S^*} r_i^{-1}$ and $r_i = g^{y_i} g_{q+1-i}$. Randomly select constants $\theta \in [1,n]$, $x_0, y_0, \{x_j\}, \{y_j\} \in \mathbb{Z}_p$, $j \in [1,n]$. For each bit $j \in [1,n]$ of message m, define $u_0 = g^{a^q(x_0-2\theta q_s)} g^{y_0}$, $u_j = g^{a^q x_j} g^{y_j}$. Finally, C selects the hash function $H(\cdot)$, and the system public parameters are $PK = \{g, e, \mathbb{G}, \mathbb{G}_T, H, Z, r_0, \{r_i\}_{i\in U}, u_0, \ldots, u_n\}$.

According to the security proof of [20], we define the following function for each message:

$$\begin{aligned} &J(m) = x_0 + \textstyle\sum_{j=1}^{n} x_j m[j] - 2\theta q_s; \\ &K(m) = y_0 + \textstyle\sum_{j=1}^{n} y_j m[j]. \end{aligned} \tag{A1}$$

According to the above function, for a message m, there is:

$$H(m) = u_0 \textstyle\prod_{j=1}^{n} u_j^{m[j]} = g_q^{J(m)} g^{K(m)}. \tag{A2}$$

where $m[j]$ represents the jth position of the message string m.

Query: Adversary A issues the following query to challenger C:

- Key extraction query: Suppose there is an access policy (L,π), where L is an LSSS matrix of $l \times m$. Since the attribute set S^* specified by adversary A does not satisfy the access policy (L,π), we define a set K, and the elements in K are matrix L that satisfy $pi(k) \in Linenumberkof S^*$. Challenger C defines a vector $\vec{\omega} = (-1, \omega_2, \ldots, \omega_m) \in \mathbb{Z}_p^{m-1}$, satisfying $\vec{L_k}\cdot\vec{\omega} = 0$, $k \in K$ and $\vec{\omega}\cdot(1,0,\ldots,0) = -1$. The challenger defines a vector $\vec{v'} = (0, v_2', \ldots, v_m') \in \mathbb{Z}_p^{m-1}$ and calculate $\vec{v} = -(\alpha_1' + a^{q+1})\vec{\omega} + \vec{v'}$.
  - If $\pi(k) \in S^*$, then $\vec{L_k}\cdot\vec{\omega} = 0$, $\varphi_{\pi(k)} = \vec{L_k}\cdot\vec{v} = -(\alpha_1' + a^{q+1})\vec{L_k}\cdot\vec{\omega} + \vec{L_k}\cdot\vec{v'} = 0 + \vec{L_k}\cdot\vec{v'} = \vec{L_k}\cdot\vec{v'}$. C chooses $\{x_k\} \in \mathbb{Z}_p$ and compute:

$$\begin{cases} d_k = g^{\varphi_{\pi(k)}}\left(r_0 \cdot r_{\pi(k)}\right)^{x_k} = g^{\vec{L}_k\vec{v'}}\left(r_0 \cdot r_{\pi(k)}\right)^{x_k}, \\ d_k' = g^{x_k}, \left\{d_k'' = r_i^{x_k}\right\}_{i\in U, i\neq\pi(k)}. \end{cases} \tag{A3}$$

  - If $\pi(k) \notin S^*$, then $\vec{L_k}\cdot\vec{\omega} \neq 0$, $\varphi_{\pi(k)} = \vec{L_k}\cdot\vec{v} = \vec{L_k}\cdot(\vec{v'} - \alpha_1'\vec{\omega}) - \vec{L_k}\cdot\vec{\omega}\cdot a^{q+1} = \vec{L_k}\cdot(\vec{v'} - \vec{\omega}(\alpha_1' + a^{q+1})$. C chooses $\{x_k'\} \in \mathbb{Z}_p$ and compute:

$$D_k = \begin{cases} d_k = g^{\vec{L}_k\cdot(\vec{v'}-\alpha_1'\vec{\omega})}\left(r_0 r_{\pi(k)}\right)^{x_k'} \cdot \left(g_{\pi(k)}^{-(y_0+y_{\pi(k)})} \prod_{i\in S^*} g_{\pi(k)}^{y_i} g_{q+1-i+\pi(k)}\right)^{-\vec{L}_k\vec{\omega}}; \\ d_k' = g^{x_k'} g_{\pi(k)}^{\vec{L}_k\cdot\vec{\omega}}; \\ \left\{d_k'' = r_i^{x_k'} \cdot \left(g^{y_i} g_{q-i+1}\right)^{\vec{L}_k\vec{\omega}a^{\pi(k)}}\right\}_{i\in U, i\neq\pi(k)}. \end{cases} \tag{A4}$$

Set $x_k = x_k' + a^{\pi(k)}\cdot\vec{L_k}\cdot\vec{\omega}$, exists:

$$D_k = \begin{cases} d_k = g^{\varphi_{\pi(k)}} \cdot \left(r_0 r_{\pi(k)}\right)^{x_k}; \\ d_k' = g^{x_k'} g_{\pi(k)}^{L_k\vec{\omega}} = g^{x_k}; \\ \left\{d_k'' = r_i^{x_k'} \cdot \left(g^{y_i} g_{q-i+1}\right)^{\vec{L}_k\vec{\omega}a^{\pi(k)}} = r_i^{x_k}\right\}_{i\in U, i\neq\pi(k)}. \end{cases} \tag{A5}$$

For the additional attribute ρ, C randomly selects $x_\rho \in \mathbb{Z}_p$ and calculates $d_\rho = g^{\alpha_2'}\left(r_0 \prod_{i\in S^*\cup\rho} r_i\right)^{x_\rho}$, $d_\rho' = g^{x_\rho}$. Finally, challenger C returns the following key: $psk = \{D_k | k \in [1,l]\}$, $usk = \left(d_\rho, d_\rho'\right)$.

- Signature query Adversary A selects message m and initiates a request to challenger C. The signature includes time t and the signer's identity attribute to prevent replay attacks. Here we hide the calculation related to t and prove that without t, forging signatures is still impossible to achieve in polynomial time. Challengers follow the steps below to sign.
  - If $J(m) = 0$, the challenger aborts the calculation.
  - If $J(m) \neq 0$, the challenger randomly selects $x'_\rho, \eta, \zeta, \zeta', \ddot{\zeta} \in \mathbb{Z}_p$, and let $\zeta + \zeta' = \ddot{\zeta} - \frac{a}{J(m)}$. The signature is generated as follows:

$$\begin{cases} \sigma_0 = g^{\alpha'_2} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{x_\rho} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{x'_\rho} \cdot g^{\alpha'_1} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{\eta} \cdot \left(g_q^{J(m)} g^{K(m)}\right)^{\ddot{\zeta}} \cdot g_1^{-\frac{K(m)}{J(m)}} \\ \quad = g^{\alpha_1 + \alpha_2} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{x_\rho + x'_\rho + \eta} \cdot \left(g_q^{J(m)} g^{K(m)}\right)^{\zeta + \zeta'}; \\ \sigma_1 = g^{\eta} \cdot g^{x_\rho + x'_\rho} = g^{\eta + x_\rho + x'_\rho}; \\ \sigma_2 = g^{\ddot{\zeta}} g_1^{-\frac{1}{J(m)}} = g^{\ddot{\zeta} - \frac{a}{J(m)}} = g^{\zeta + \zeta'}. \end{cases} \tag{A6}$$

Finally, for the signature query of message m, challenger C will return$\sigma = \{\sigma_0, \sigma_1, \sigma_2, m\}$.

- User signature verification query: The adversary A has the signature σ of the message m under the attribute set S. After receiving the signature, C executes the signature extraction algorithm and randomly selects $\delta \in \mathbb{Z}_p$, calculate $\hat{\sigma_0} = \sigma_0^\delta$, $\hat{\sigma_1} = \sigma_1^\delta$, and the converted signature $\hat{\sigma} = \{\hat{\sigma_0}, \hat{\sigma_1}\}$ is returned to A. A calculates the intermediate signature $\widetilde{\sigma_1}$ and returns it to C. A executes the aided verification algorithm and calculates the following intermediate results:

$$\frac{e(\hat{\sigma_0}, g)}{e(\hat{\sigma_1}, r_0 \prod_{i \in S^* \cup \rho} r_i)} = \widetilde{\sigma_1}. \tag{A7}$$

The challenger computes $\widetilde{\sigma_2} = e(\sigma_2^\delta, H(m)) \cdot Z^\delta$ and Determine whether $\widetilde{\sigma_1}$ is equal to $\widetilde{\sigma_2}$. If they are equal, the signature verification is successful, otherwise the signature verification fails.

Signature forgery: The adversary A forges the signature of the message m^* under the attribute set $S^* \sigma^* = \{\sigma_0^*, \sigma_1^*, sigma_2^*, m^*\}$. A has never initiated a signature query for message m^* under attribute set S^*. If $J(m) \neq 0$, challenger C interrupts the calculation. If $J(m) = 0$, then there is:

$$\begin{aligned} \sigma_0^* &= g^{\alpha'_2} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{x_\rho} \cdot \left(r_0 \prod_{i \in S^* \cup \rho} r_i\right)^{x'_\rho} \cdot g^{\alpha'_1} \cdot \left(r_0 \prod_{i \in S^*} r_i\right)^{\eta} \cdot \left(g_q^{J(m^*)} g^{K(m^*)}\right)^{\zeta} \cdot g_1^{-\frac{K(m^*)}{J(m^*)}} \\ &= g^{\alpha'_1} \cdot g^{a^{q+1}} \cdot g^{\alpha'_2} \cdot (\sigma_1^*)^{\gamma_0} \cdot (\sigma_2^*)^{K(m^*)}. \end{aligned} \tag{A8}$$

The challenger can then calculate:

$$g^{a^{q+1}} = \frac{\sigma_0^*}{g^{\alpha'_1} \cdot g^{\alpha'_2} \cdot (\sigma_1^*)^{\gamma_0} \cdot (\sigma_2^*)^{K(m^*)}}. \tag{A9}$$

If the challenger can calculate $g^{a^{q+1}}$, the following event is defined:

1. E_μ: Random μth signature query for message m, $J(m) \neq 0$;

2. E': For the forged signature σ^* of message m^*, there exists $J(m^*) = 0$.

In addition, define an event E, indicating that C did not terminate the calculation during the above query process. Using the method in [20], the probability of event E is calculated as follows:

$$\Pr[E] = \Pr\left[\bigcap_{\mu=1}^{q_s} E_\mu \bigcap E'\right] \geq \frac{1}{4q_s(n+1)}. \tag{A10}$$

□

Appendix A.2 Anonymity

Anonymity refers to: assuming there are two signatures σ^A and σ^B that comply with the unified access policy, the user cannot distinguish between the two signatures. Let $a_u + s + s' = S$. We can analyze the structure of the signature:

$$\begin{cases} \sigma_0 = g^{\sum_{k\in K}\rho_{\pi(k)}\omega_k+\alpha_2} \cdot (r_0 \prod_{i\in S_u\cup\rho} r_i)^{\sum_{k\in K} x_k\omega_k+\eta+x_{\rho,a_u}+x'_{\rho,a_u}} \cdot H(m)^{\zeta+\zeta'} \cdot H(t)^S, \\ \sigma_1 = g^{\sum_{k\in K} x_k\omega_k+\eta+x_{\rho,a_u}+x'_{\rho,a_u}}, \\ \sigma_2 = g^{\zeta+\zeta'}, \sigma_3 = g^S, \sigma_4 = (r_0 \prod_{i\in S_u\cup\rho} r_i H(t))^{1/\beta}. \end{cases} \tag{A11}$$

Among them, there are x_{ρ,a_u} and a_u parts of user-specific information, which only exist at the index position, and the random numbers $\eta, x'_{\rho,a_u}, s, s' \in \mathbb{Z}_p$, which are also exponential, such that $\sum_{k\in K} x_k\omega_k + \eta + x_{\rho,a_u} + x'_{\rho,a_u}$ and $a_u + s + s'$ are random. The signature does not contain attribute information, so the user cannot confirm the identity of the signer from the signature, or distinguish between two signatures.

References

1. Maji H, Prabhakaran M, Rosulek M. Attribute-based signatures: achieving attribute-privacy and collusion-resistance. Cryptology ePrint Archive; 2008 [cited 2025 Feb 28]. Available from: http://eprint.iacr.org/2008/328.
2. Xu X, Cai Q, Lin J, Pan S, Ren L. Enforcing access control in distributed version control systems. In: 2019 IEEE International Conference on Multimedia and Expo (ICME) 2019; 2019 Jul 8–12; Shanghai, China. New York, NY, USA: IEEE. 2019. p. 772–7. doi:10.1109/ICME.2019.00138.
3. Hong H, Hu B, Sun Z. An efficient and secure attribute-based online/offline signature scheme for mobile crowdsensing. Hum-Cent Comput Inf Sci. 2021;11:26. doi:10.22967/HCIS.2021.11.026.
4. Li X, Wang H, Ma S, Xiao M, Huang Q. Revocable and verifiable weighted attribute-based encryption with collaborative access for electronic health record in cloud. Cybersecurity. 2024;7(1):1–19. doi:10.1186/s42400-024-00211-1.
5. Patil RY. A secure privacy preserving and access control scheme for medical internet of things (MIoT) using attribute-based signcryption. Int J Inf Technol. 2024;16(1):181–91. doi:10.1007/s41870-023-01569-0.
6. Okamoto T, Takashima K. Efficient attribute-based signatures for non-monotone predicates in the standard model. IEEE Trans Cloud Comput. 2014;2(4):409–21. doi:10.1109/TCC.2014.2353053.
7. Cui H, Deng RH, Liu JK, Yi X, Li Y. Server-aided attribute-based signature with revocation for resource-constrained industrial-internet-of-things devices. IEEE Trans Industr Inform. 2018;14(8):3724–32. doi:10.1109/TII.2018.2813304.
8. Huang Z, Lin Z. Secure server-aided attribute-based signature with perfect anonymity for cloud-assisted systems. J Inf Secur Appl. 2022;65(1):103066. doi:10.1016/j.jisa.2021.103066.
9. Chen Y, Li J, Liu C, Han J, Zhang Y, Yi P. Efficient attribute based server-aided verification signature. IEEE Trans Serv Comput. 2022;15(6):3224–32. doi:10.1109/TSC.2021.3096420.

10. Li J, Chen Y, Han J, Liu C, Zhang Y, Wang H. Decentralized attribute-based server-aid signature in the internet of things. IEEE Internet Things J. 2022;9(6):4573–83. doi:10.1109/JIOT.2021.3104585.
11. Chen B, Xiang T, Li X, Zhang M, He D. Efficient attribute-based signature with collusion resistance for internet of vehicles. IEEE Trans Veh Technol. 2023;72(6):7844–56. doi:10.1109/TVT.2023.3240824.
12. Maji HK, Prabhakaran M, Rosulek M. Attribute-based signatures. In: Cryptographers' Track at the RSA Conference; 2011; Berlin, Germany: Springer. p. 376–92.
13. Xiong H, Bao Y, Nie X, Asoor YI. Server-aided attribute-based signature supporting expressive access structures for industrial internet of things. IEEE Trans Indust Inform. 2019;16(2):1013–23. doi:10.1109/TII.2019.2921516.
14. SZABO N. Smart contracts; 1994 [cited 2024 Feb 29]. Available from: https://nakamotoinstitute.org/library/smart-contracts/.
15. Christidis K, Devetsikiotis M. Blockchains and smart contracts for the internet of things. IEEE Access. 2016;4:2292–303. doi:10.1109/ACCESS.2016.2566339.
16. Wood G et al. Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper. 2014;151:1–32.
17. Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy (SP'07); 2007; Oakland, CA, USA. Washington, DC, USA: IEEE Computer Society. p. 321–34. doi:10.1109/SP.2007.11.
18. Ostrovsky R, Sahai A, Waters B. Attribute-based encryption with non-monotonic access structures. In: Proceeding of the 2007 ACM Conference on Computer and Communications Security, CCS 2007; 2007 Oct 28–31; Alexandria, VA, USA. New York, NY, USA: ACM; 2007. p. 195–203. doi:10.1145/1315245.1315270.
19. Goyal V, Kothapalli A, Masserova E, Parno B, Song Y. Storing and retrieving secrets on a blockchain. In: IACR International Conference on Public-Key Cryptography; 2022 Mar 8–11; Virtual Event. Cham, Switzerland: Springer; 2022. Vol. 13177, p. 252–82. doi:10.1007/978-3-030-97121-2_10.
20. Waters B. Efficient identity-based encryption without random oracles. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques; 2005 May 22–26; Aarhus, Denmark. Cham, Switzerland: Springer; 2005. Vol. 3494, p. 114–27. doi:10.1007/11426639_7.