Hindawi
Wireless Communications and Mobile Computing
Volume 2023, Article ID 4644778, 20 pages
https://doi.org/10.1155/2023/4644778

Research Article

Authorization Recycling in Attribute-Based Access Control

Yan An and Nurmamat Helil

College of Mathematics and System Science, Xinjiang University, Urumqi, China

Correspondence should be addressed to Nurmamat Helil; nur924@sina.com

Received 6 May 2022; Revised 22 March 2023; Accepted 17 April 2023; Published 31 May 2023

Academic Editor: Hamza Mohammed Ridha Al-Khafaji

In most access control scenarios, the communication between the PDP (policy decision point) and the PEP (policy enforcement point) can cause high authorization overhead. Authorization recycling enables PEP to use the previous access control decisions fetched from the PDP to handle some upcoming access control requests, reduce authorization costs, and increase the efficiency of access control decision-making. Inspired by the RBAC (role-based access control) authorization recycling mechanism, this article first presents an ABAC (attribute-based access control) model based on Boolean expressions of subject and object attributes. It then proposes an authorization recycling approach for this model. In this approach, we provide construction and update methods for authorization data caches and access control decision-making rules for SDP (secondary decision point) by using the caches. The proposed approach can deduce precise and approximate access control decisions from the cache of authorization data, reducing communication between the PEP and the PDP. Finally, the feasibility of the proposed method is verified by conducting a small-scale test. ABAC, SDP, authorization recycling, and authorization caching.

1. Introduction

Access control [1] is to restrict users of an application system from performing illegal operations or prevent their unauthorized access that can lead to security problems. The main access control types are DAC (discretionary access control) [2], MAC (mandatory access control) [3], and RBAC (role-based access control) [4]. However, compared to the DAC, MAC, and RBAC, the ABAC has the advantages of high flexibility, strong scalability, and fine granularity, making it widely used in distributed environments [5–9] because it overcomes the limitations of the traditional access control schemes.

As the scale of distributed applications has been expanding, the authorization mechanism based on the traditional single PDP (policy decision point) has become more fragile and challenging to extend to large-scale systems. The architecture of traditional access control solutions has certain shortcomings. Namely, the PDP can be a single point of failure, thus becoming a latent performance bottleneck, which affects the system's reliability and availability. Moreover, the communication delay between the PDP and the PEP (Policy Enforcement Point) can cause high authorization overhead. To improve the above shortcomings, researchers adopted authorization recycling [10]. The authorization recycling mechanism makes full use of historical access control decision data to improve the efficiency of future access control decisions. It has long been used to improve the availability and performance of access control systems. Some simple authorization recycling methods have been proposed [11, 12], which are based on the request-response model, in which the PEP intercepts application requests and sends the request to the PDP. The PDP returns the decision results to the PEP according to the access control policy, and the PEP enforces these decisions.

In order to save storage space and enhance the usability and performance of a system, it is necessary to propose an approximate authorization recycling mechanism. Beznosov [13] extends the precise authorization recycling mechanism and introduces approximate authorization recycling. Crampton et al. [14] propose the SAAM (secondary approximate authorization model) for the BLP (Bell-LaPadula) model. The SDP (secondary decision point) is added to the authorization architecture [15], as depicted in Figure 1.

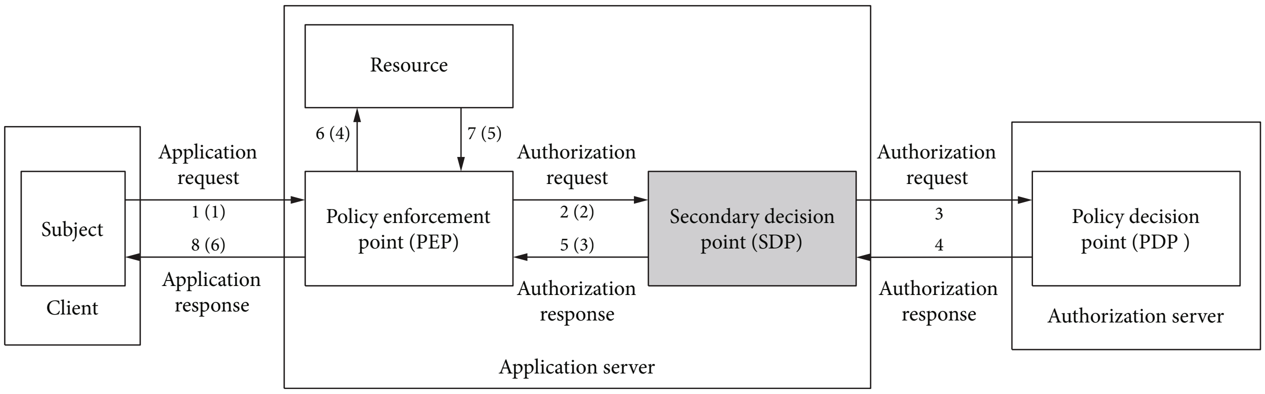


Figure 1: General request-response mode access control architecture with authorization recycling.

The SDP is embedded with the PEP, which cannot only handle repeated authorization requests but also can solve approximate authorization requests through the calculation in the authorization cache. The SAAM infers precise and approximate responses using the relationship between subject and object inferred from the previous responses. In Figure 1, paths 1–8 illustrate the authorization workflow when the SDP cannot make a decision on the request and the request needs to be sent to the PDP; paths (1)–(6) illustrate the case where the SDP can make a decision on the request.

Wei et al. [15] introduce the authorization recycling method in a hierarchical RBAC system, which can not only compress authorization data in the cache but also effectively infer approximate decisions from the cached data. Reeja [16] proposes an authorization recycling method using the CSAR (cooperative secondary authorization recycling) in the cloud computing system, which utilizes the cooperative cache of access control decisions to improve the hit rate. Wei et al. [17] propose a publish-subscribe system in which authorization requests and responses are passed between applications and authorization servers, which can enhance the availability of the authorization system.

Inspired by the literature [15], we present an authorization recycling approach in the ABAC system. However, the two models are quite different. In RBAC, roles are associated with permissions, which can be regarded as ABAC single attribute; RBAC only considers positive authorization; in ABAC, we consider the combination of multiple attributes associated with permissions, so the processing of cache construction and update is also different, which makes authorization recycling more complicated but worthwhile. Furthermore, the ABAC [18, 19] neither follows the closed world policy nor the open world policy. Therefore, this article mainly introduces the ABAC authorization recycling approach under the closed-world, open-world, and hybrid policy [20]. Firstly, we build a general-purpose ABAC model which supports positive and negative but also hybrid authorizations. CP-ABE [21, 22] (ciphertext-policy attribute-based encryption) is one of the prominent attribute-based cryptographic access control approaches for "*read*" permissions on data objects in a cloud environment. Its access policy is a Boolean expression of attributes and has the essential characteristics of the ABAC model. Therefore, we first construct the ABAC model based on Boolean expressions of subject and object attributes. Secondly, provide methods for building and updating the authorization data caches and specify SDP's access control decision rules. Finally, the feasibility of the authorization recycling approach is verified by conducting a small-scale experiment.

The rest of the paper is organized as follows. Section 2 introduces the ABAC model and assumptions of authorization recycling. Section 3 describes the ABAC authorization recycling approach in detail. Section 4 presents the test results of the proposed recycling approach. A discussion of related works and comparisons is provided in Section 5. Finally, the main conclusions and directions for future work are given in Section 6.

2. ABAC Model

2.1. ABAC Model. The ABAC uses attributes to describe the subject, object, and environment and uses these attributes to formulate access control policies [23–25] to evaluate whether the subject is allowed to perform specific actions on the object.

The ABAC model considered in this paper basically follows the specifications in [26], and its essential components are as follows:

(1) Attribute: attributes represent characteristics of the subject and object; attributes have names and values

(2) Subject: a user or entity, such as a device, that makes an access request and performs operations on an object; the subject can have one or more attributes

(3) Object: system resources accessed by the subject; the object can have one or more attributes

(4) Operation: operation represents the subject's actions on the object, including *read*, *write*, *edit*, *delete*, and other actions

(5) Permission: permission is composed of an object and an operation (action) on the object

(6) Policy: a policy consists of access control rules about a subject and an object that determine whether to allow the subject's access request to an object based on the attributes of the subject and the object

The rule is the basic unit of an ABAC policy [27]. We first discuss the rule. R is used to represent a rule, and SR and OR are used to describe rules related to subject and object, respectively. The precise recursive definitions of rules related to subject and object are given in the following.

Definition 1. The subject-related rule (subject-rule) can be recursively defined as follows:

(1) Any subject attribute is a subject rule, and it is called an atomic subject rule

(2) If $\mathrm{SR}_1, \mathrm{SR}_2$ are subject rules, then their conjunction $\mathrm{SR}_1 \wedge \mathrm{SR}_2$ is also a subject rule

(3) If $\mathrm{SR}_1, \mathrm{SR}_2$ are subject rules, then their disjunction $\mathrm{SR}_1 \vee \mathrm{SR}_2$ is also a subject rule

(4) An expression obtained by (1)–(3) a finite number of times is a subject rule

Definition 2. The object-related rule (object-rule) can be recursively defined as follows:

(1) Any object attribute is an object rule, and it is called an atomic object rule

(2) If $\mathrm{OR}_1, \mathrm{OR}_2$ are object rules, then their conjunction $\mathrm{OR}_1 \wedge \mathrm{OR}_2$ is also an object rule

(3) If $\mathrm{OR}_1, \mathrm{OR}_2$ are object rules, then their disjunction $\mathrm{OR}_1 \vee \mathrm{OR}_2$ is also an object rule

(4) An expression obtained by (1)–(3) a finite number of times is an object rule

If an attribute of a subject (object) in an access request matches the atomic subject (object) rule, then we say the subject (object) attribute satisfies the atomic subject (object) rule.

In this work, the subject and object in the ABAC are modeled as a set of attributes, respectively. Environmental attributes have the same status as subject and object attributes. For simplicity, environmental attributes are not considered in this article.

As we know, an attribute usually refers to a tuple $<attribute-name : attribute-value>$. In most CP-ABE schemes, an attribute name and a value are combined and represented as an attribute. We follow the characteristics of the CP-ABE access policy to define our ABAC policy. We also try to improve the expressivity of access control policy in our ABAC model. Here, we enumerate attributes that can be supported in our ABAC and discuss how to define rules about these attributes.

(1) Nominal attribute: it appears in the form of an attribute name and an attribute value in the rule. For example, $SR = <name : Alice>$. If a subject attribute $satt = <name : Alice>$ is included in a request, then we say $satt = <name : Alice>$ satisfies SR since both the attribute name and attribute value match the attribute name and attribute values in SR

(2) Binary attributes: they appear in the form of an attribute name and an attribute value in the rule. For example, $SR = <gender : 1>$ (1 denotes male, 0 denotes female). If a subject attribute $satt = <gender : 1>$ is included in a request, then we say $satt = <gender : 1>$ satisfies SR since both the attribute name and attribute value matches the attribute name and attribute values in SR

(3) Orderial attributes: there is a full order relation among all values of an orderial attribute. For example, an attribute *level* has five different values: *excellent*, *good*, *average*, *fair*, *poor*, and $excellent \geq good \geq average \geq fair \geq poor$. We can express a raw subject-related rule "$level \geq average$" as $\mathrm{SR}_1 \vee \mathrm{SR}_2 \vee \mathrm{SR}_3$ in our ABAC rule, where $SR_1 = <level : excellent>$, $SR_2 = <level : good>$,$SR_3 = <level : average>$. If a subject attribute $satt = <level : good>$ is included in a request, then we say $satt = <level : good>$ satisfies $\mathrm{SR}_1 \vee \mathrm{SR}_2 \vee \mathrm{SR}_3$ since both the attribute name and attribute value match the attribute name and attribute values in SR_2

(4) Numeric attributes: it only supports discrete, finite attribute values. For example, an attribute *score* has five different values: 1, 2, 3, 4, and 5. We can express a raw object-related rule $3 \leq score \leq 5$ as $OR_1 \vee OR_2 \vee OR_3$ in our ABAC rule, where $\mathrm{OR}_1 = <score : 3>$, $\mathrm{OR}_2 = <score : 4>$, $\mathrm{OR}_3 = <score : 5>$. If an object attribute $\mathrm{oatt} = <score : 3>$ is included in a request, then we say object attribute $\mathrm{oatt} = <score : 3>$ satisfies $\mathrm{OR}_1 \vee \mathrm{OR}_2 \vee \mathrm{OR}_3$ since both the attribute name and attribute value match the attribute name and attribute values in OR_1

According to the explanation above, we will directly discuss attributes. We will not discuss attribute name and value separately and also will not discuss other kinds of mathematical operations such as ≤, ≥, <, and >.

The formal specification of the ABAC model is as follows:

(1) $\mathrm{S(USERS), OB, OP, SATTR, OATRR, PERMS}$ represent the subjects (the subject is usually assumed to be a user), objects, operations, subject attributes, object attributes, and permissions, respectively

(2) $\mathrm{PERMS} \subseteq \mathrm{OP} \times \mathrm{OB}$, where $\mathrm{p} = <\mathrm{op}, \mathrm{ob}>$, $\mathrm{op} \in \mathrm{OP}$, $\mathrm{ob} \in \mathrm{OB}$. Permissions represent the authorized behavior of a subject; the object is expressed by a

boolean combination of object attributes in object rules

Closed-world policy, open-world policy, and a combination of both, hybrid policy, are meta-policies used in most access control scenarios. Unfortunately, the hybrid policy can lead to the absence of a satisfactory policy or the occurrence of policy conflicts against an access request [28]. Satisfiable policy absence indicates that the user access request meets no authorization policy. Policy conflict means the user access request meets both positive and negative authorization policies. To enhance system security, if there is a policy conflict, the denial-take-precedence [29] is adopted. If there occurs satisfiable policy absence, the denial-for-absence is used. In practice, different systems can define their own meta-policies for satisfiable policy absence and policy conflicts.

Based on the definitions mentioned above, we define three variants of the ABAC model, ABAC (P), ABAC (N), and ABAC (H), that follow the closed-world policy, the open-world policy, and the combination of these two policies, respectively.

In this article, it is assumed that there exists at least one policy for each permission. An access control policy for permission p is defined as a triple policy $= <\mathrm{SR} \wedge \mathrm{OR}, p, \text{effect}>$, where *SR* denotes the subject rule and OR denotes the object rule, $p = <op, ob>$, $effect \in \{permit,\ deny\}$. We use $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ to represent the request, where S_Attr, O_Attr, and p represent the subject attribute set, object attribute set, and permission to access, respectively. The supplementary definitions for the three variants are given in the following.

Definition 3 (ABAC(P)) model. For a positive authorization policy for permission p is defined as a triple policy$^+ = <\mathrm{SR}^+ \wedge \mathrm{OR}^+, \mathrm{p}, \mathrm{permit}>$ and an access request$(S_\mathrm{Attr}, O_\mathrm{Attr}, p)$, only if the subject and object attribute sets in the access request satisfy $\mathrm{SR}^+ \wedge \mathrm{OR}^+$, the final access control decision for permission p will be permit; otherwise, it will be *deny*.

Definition 4 (ABAC(N)) model. For a negative authorization policy for permission p is defined as a triple policy$^- = <\mathrm{SR}^- \wedge \mathrm{OR}^-, \mathrm{p}, \mathrm{deny}>$ and an access request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$, only if the subject and object attribute sets in the access request satisfy $\mathrm{SR}^- \wedge \mathrm{OR}^-$,the final access control decision for permission p will be *deny*; otherwise, it will be *permit*.

Definition 5 (ABAC(H)) model. For an authorization policy for permission p is defined as a triple $p$olice $= <\mathrm{SR} \wedge \mathrm{OR}, p, \mathrm{effect}>$, where $\mathrm{effect} \in \{\mathrm{permit}, \mathrm{deny}\}$ and a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$, only if both the subject and object attribute sets in the access request satisfy SR∧OR, the request meets the access control policy. The final access control decision for permission p is specified as follows:

(1) If there are only one positive authorization policy about permission p, then this case is regarded as ABAC (P); if the request meets the policy, then the decision result will be *permit*; otherwise, it will be *deny*

(2) If there exists only one negative authorization policy about permission p, then this case is regarded as ABAC (N); if the request meets the policy, then the decision result will be *deny*; otherwise, it will be *permit*

(3) If there exist both positive and negative authorization policies about permission p, then

  (a) If the request meets only the positive authorization policy, then the decision result will be *permit*

  (b) If the request meets only the negative authorization policy, then the decision result will be *deny*

  (c) If the request meets both polices, then the denial-take-precedence will be adopted, and the decision result will be *deny*

  (d) If the request meets none of the policies, then the denial-for-absence will be adopted, and the decision result will be *deny*

2.2. Authorization Recycling Assumptions. Before the authorization recycling approach for the ABAC model is presented, the assumptions of the ABAC system should be given, and they are as follows:

(1) Only the PDP can reach the whole access control policy, while the SDP cannot

(2) The SDP generates precise, approximate responses based on the cached decision data and request

Definition 6 (Precise and approximate decision). For an authorization request, the authorization decision is precise if PDP makes the decision or the SDP makes it, and there exists the exact same previous request with a decision by PDP. Otherwise, we say the decision is approximate.

The SDP is considered safe if it permits any request that the PDP permits. Furthermore, the SDP is considered consistent if it denies any request that the PDP denies [14]. Typically, a safe and consistent SDP that will return the same response as the PDP for any request is expected. This article aims to construct a safe and consistent SDP.

Next, the definition of minimal subject and object attribute sets is given as follows to help us construct the cache.

Definition 7 (Minimal subject attribute set). Attribute set SATT is a minimal subject attribute set of a subject-related rule SR, if it satisfies the following requirements [30]:

(1) Attribute set SATT satisfies SR

(2) For all $\mathrm{SATT}' \subset \mathrm{SATT}$, attribute set SATT' does not satisfy SR

Example 1. Suppose $\mathrm{SR} = ((\mathrm{satt}_1 \wedge \mathrm{satt}_2) \vee \mathrm{satt}_3) \wedge (\mathrm{satt}_4 \wedge \mathrm{satt}_5)$. Minimal subject attribute sets of SR are $\mathrm{SATT}_1 = \{\mathrm{satt}_1, \mathrm{satt}_2, \mathrm{satt}_4, \mathrm{satt}_5\}$ and $\mathrm{SATT}_2 = \{\mathrm{satt}_3, \mathrm{satt}_4, \mathrm{satt}_5\}$. Attribute sets SATT_1 and SATT_2 satisfy subject-related rule SR; for all $\mathrm{SATT}_1' \subset \mathrm{SATT}_1$, and $\mathrm{SATT}_2' \subset \mathrm{SATT}_2$, attribute sets SATT_1' and SATT_2' do not satisfy the subject-related rule SR.

Definition 8 (Minimal object attribute set). Attribute set OATT is a minimal object attribute set of an object-related rule OR, if it satisfies the following requirements:

(1) Attribute set OATT satisfies OR

(2) For all $\mathrm{OATT}' \subset \mathrm{OATT}$, attribute set OATT' does not satisfy OR

3. Authorization Recycling for ABAC Model

3.1. Preliminaries. It is reasonable to assume that there is at least one policy for each permission, either a positive authorization policy, a negative authorization policy, or both. If there are multiple positive (or negative) authorization policies, then they can be merged into one positive (negative) authorization policy using a Boolean expression, while different types of authorization policies, i.e., positive and negative, cannot be merged.

Suppose a subject has an access request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$, where S_Attr, O_Attr, and p represent the subject attribute set, object attribute set, and permission to access, respectively. We use $\mathrm{policy}^+ = <\mathrm{SR}^+ \wedge \mathrm{OR}^+, p, \mathrm{permit}>$ to represent the positive authorization policy, $S^+\_\mathrm{ATTR}(p) = \{\mathrm{SATT}_1^+, \mathrm{SATT}_2^+, \cdots, \mathrm{SATT}_s^+\}$ to represent a set of minimal subject attribute sets of SR^+; $O^+\_\mathrm{ATTR}(p) = \{\mathrm{OATT}_1^+, \mathrm{OATT}_2^+, \cdots, \mathrm{OATT}_t^+\}$ represents a set of minimal object attribute sets of OR^+. Similarly, $\mathrm{policy}^- = <\mathrm{SR}^- \wedge \mathrm{OR}^-, p, \mathrm{deny}>$ represents the negative authorization policy, $S^-\_\mathrm{ATTR}(p) = \{\mathrm{SATT}_1^-, \mathrm{SATT}_2^-, \cdots, \mathrm{SATT}_s^-\}$ represents a set of minimal subject attribute sets of SR^-; $O^-\_\mathrm{ATTR}(p) = \{\mathrm{OATT}_1^-, \mathrm{OATT}_2^-, \cdots, \mathrm{OATT}_t^-\}$ represents a set of minimal object attribute sets of OR^-

This section considers different types of caches relying on the responses fetched from the PDP and the existing policies.

(1) There exists only one positive authorization policy for permission $\boldsymbol{p}$:

(a) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ satisfies the positive authorization policy for p at the PDP side, then the PDP permits the request and returns some attribute sets $\{\mathrm{SATT}_{i_k}^+\}_{k \le s}$ and $\{\mathrm{OATT}_{j_l}^+\}_{l \le t}$ together with the *permit* response to the SDP, where

$$\begin{aligned} &\mathrm{SATT}_{i_k}^+ \in S^+\_\mathrm{ATTR}(p) \text{ and } \mathrm{SATT}_{i_k}^+ \subseteq S\_\mathrm{Attr}, k \le s, \\ &\mathrm{OATT}_{j_l}^+ \in O^+\_\mathrm{ATTR}(p) \text{ and } \mathrm{OATT}_{j_l}^+ \subseteq O\_\mathrm{Attr}, l \le t. \end{aligned} \tag{1}$$

The corresponding *permit* response is denoted as $++(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

(b) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ does not satisfy the positive authorization policy for p at the PDP side, then the PDP denies the request, and return attribute sets S_Attr and O_Attr together with the *deny* response to the SDP, the corresponding *deny* response is denoted as $*-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

(2) There exists only one negative authorization policy for permission $\boldsymbol{p}$:

(a) If a request (S_Attr, O_Attr, p) satisfies the negative authorization policy for p at the PDP side, then then the PDP denies the request and returns some attribute sets $\{SATT_{i_k}^-\}_{k \le s}$ and $\{OATT_{j_l}^-\}_{l \le t}$ together with the *deny* response to the SDP, where

$$\begin{aligned} &\mathrm{SATT}_{i_k}^- \in S^-\_\mathrm{ATTR}(p) \text{ and } \mathrm{SATT}_{i_k}^- \subseteq S\_\mathrm{Attr}, k \le s, \\ &\mathrm{OATT}_{j_l}^- \in O^-\_\mathrm{ATTR}(p) \text{ and } \mathrm{OATT}_{j_l}^- \subseteq O\_\mathrm{Attr}, l \le t. \end{aligned} \tag{2}$$

The corresponding *deny* response is denoted as $--(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

(b) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ does not satisfy the negative authorization policy for p at the PDP side, then the PDP permits the request, and return attribute sets S_Attr and O_Attr together with the *permit* response to the SDP, the corresponding *permit* response is denoted as $*+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

(3) There exist both positive and negative authorization policies for permission $\boldsymbol{p}$:

(a) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ satisfies both the positive and negative authorization policies, then the PDP denies the request because the denial-take-precedence is adopted, or if the request satsfies only the negative authorization policy, then the PDP denies the request, and the PDP returns all attribute sets SATT_i^- and OATT_j^- together with the deny response to the SDP, where

$$\begin{aligned} &\mathrm{SATT}_i^- \in S^-\_\mathrm{ATTR}(p), i = 1, 2, \cdots, s, \\ &OATT_j^- \in O^-\_ATTR(p), j = 1, 2, \cdots, t. \end{aligned} \tag{3}$$

The corresponding *deny* response is denoted as $-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

This case is slightly different from the other cases above. The PDP returns all the minimal sets that meet the negative authorization policy to the SDP while returning the *deny* response.

(b) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ satisfies only the positive authorization policy but not the negative one, then the PDP permits the request and returns some attribute sets $\{\mathrm{SATT}^{+}_{i_k}\}_{k\le s}$ and $\{\mathrm{OATT}^{+}_{j_l}\}_{l\le t}$ together with the *permit* response to the SDP, where

$$\begin{aligned}&\mathrm{SATT}^{+}_{i_k}\in S^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{SATT}^{+}_{i_k}\subseteq S\_\mathrm{Attr},\ k\le s,\\&\mathrm{OATT}^{+}_{j_l}\in O^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{OATT}^{+}_{j_l}\subseteq O\_\mathrm{Attr},\ l\le t.\end{aligned}\tag{4}$$

The corresponding *permit* response is denoted as $+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

(c) If a request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ does not satisfy both the positive and negative authorization policies, then the PDP denies the request because the denial-for-absence is adopted, and returns attribute sets S_Attr and O_Attr together with the *deny* response to the SDP, the corresponding *deny* response is denoted as $\sim(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$.

The response content is specified as

$$\begin{aligned}&++(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{permit}, \left(\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{+}_{j_l}\right\}_{l\le t}, p\right)>,\text{where}\\&\mathrm{SATT}^{+}_{i_k}\in S^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{SATT}^{+}_{i_k}\subseteq S\_\mathrm{Attr},\ k\le s,\\&\mathrm{OATT}^{+}_{j_l}\in O^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{OATT}^{+}_{j_l}\subseteq O\_\mathrm{Attr},\ l\le t,\\&*-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{deny}, \left(\left\{\tilde{S}\_\mathrm{Attr}\right\}, \left\{\tilde{O}\_\mathrm{Attr}\right\}, p\right)>,\end{aligned}\tag{5}$$

where

$$\begin{aligned}&\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}\ \text{and}\ \tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr},\\&--(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{deny}, \left(\left\{\mathrm{SATT}^{-}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{-}_{j_l}\right\}_{l\le t}, p\right)>,\end{aligned}\tag{6}$$

where

$$\begin{aligned}&\mathrm{SATT}^{-}_{i_k}\in S^{-}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{SATT}^{-}_{i_k}\subseteq S\_\mathrm{Attr},\ k\le s,\\&\mathrm{OATT}^{-}_{j_l}\in O^{-}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{OATT}^{-}_{j_l}\subseteq O\_\mathrm{Attr},\ l\le t,\\&*+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{permit}, \left(\left\{\tilde{S}\_\mathrm{Attr}\right\}, \left\{\tilde{O}\_\mathrm{Attr}\right\}, p\right)>,\end{aligned}\tag{7}$$

where

$$\begin{aligned}&\tilde{S}\_Attr = S\_Attr\ \text{and}\ \tilde{O}\_Attr = O\_Attr\\&-(S\_Attr, O\_Attr, p):\ \triangleq <\mathrm{deny}, \left(\left\{\mathrm{SATT}^{-}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{-}_{j_l}\right\}_{l\le t}, p\right)>,\end{aligned}\tag{8}$$

where

$$\begin{aligned}&\mathrm{SATT}^{-}_{i_k}\in S^{-}\_\mathrm{ATTR}(p),\ i=1,2,\cdots,s,\\&\mathrm{OATT}^{-}_{j_l}\in O^{-}\_\mathrm{ATTR}(p),\ j=1,2,\cdots,t,\\&+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{permit}, \left(\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{+}_{j_l}\right\}_{l\le t}, p\right)>,\end{aligned}\tag{9}$$

where

$$\begin{aligned}&\mathrm{SATT}^{+}_{i_k}\in S^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{SATT}^{+}_{i_k}\subseteq S\_\mathrm{Attr},\ k\le s,\\&\mathrm{OATT}^{+}_{j_l}\in O^{+}\_\mathrm{ATTR}(p)\ \text{and}\ \mathrm{OATT}^{+}_{j_l}\subseteq O\_\mathrm{Attr},\ l\le t,\\&\sim(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p):\ \triangleq <\mathrm{deny}, \left(\left\{\tilde{S}\_\mathrm{Attr}\right\}, \left\{\tilde{O}\_\mathrm{Attr}\right\}, p\right)>,\end{aligned}\tag{10}$$

where

$$\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}\ \text{and}\ \tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}.\tag{11}$$

It is assumed that the PDP sends attribute sets included in response to the SDP, which may threaten the privacy of the policy. Therefore, these contents can be encrypted, and the PDP sends the ciphertext to the SDP. However, since our emphasis is on authorization recycling, we did not consider policy privacy in our work.

3.2. Cache Building. Seven relations below are used for cache construction:

$$\begin{aligned}&\mathrm{Cache}^{++} = \left\{\left(\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{+}_{j_l}\right\}_{l\le t}, p\right)\middle| p\in\mathrm{PERMS}, \mathrm{SATT}^{+}_{i_k}\in S^{+}\_\mathrm{ATTR}(p), \mathrm{OATT}^{+}_{j_l}\in O^{+}\_\mathrm{ATTR}(p)\right\},\\&\mathrm{Cache}^{*-} = \left\{\left(\{S_u\_\mathrm{Attr}\}_{u\ge1}, \{O_v\_\mathrm{Attr}\}_{v\ge1}, p\right)\middle| p\in\mathrm{PERMS}, S_u\_\mathrm{Attr}\subseteq\mathrm{SATTR}, O_v\_\mathrm{Attr}\subseteq\mathrm{OATTR}\right\},\\&\mathrm{Cache}^{--} = \left\{\left(\left\{\mathrm{SATT}^{-}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{-}_{j_l}\right\}_{l\le t}, p\right)\middle| p\in\mathrm{PERMS}, \mathrm{SATT}^{-}_{i_k}\in S^{-}\_\mathrm{ATTR}(p), \mathrm{OATT}^{-}_{j_l}\in O^{-}\_\mathrm{ATTR}(p)\right\},\\&\mathrm{Cache}^{*+} = \left\{\left(\{S_u\_\mathrm{Attr}\}_{u\ge1}, \{O_v\_\mathrm{Attr}\}_{v\ge1}, p\right)\middle| p\in\mathrm{PERMS}, S_u\_\mathrm{Attr}\subseteq\mathrm{SATTR}, O_v\_\mathrm{Attr}\subseteq\mathrm{OATTR}\right\},\\&\mathrm{Cache}^{-} = \left\{\left(\{\mathrm{SATT}^{-}_{i}\}_{i=1,2,\cdots,s}, \left\{\mathrm{OATT}^{-}_{j}\right\}_{j=1,2,\cdots,t}, p\right)\middle| p\in\mathrm{PERMS}, \mathrm{SATT}^{-}_{i}\in S^{-}\_\mathrm{ATTR}(p), \mathrm{OATT}^{-}_{j}\in O^{-}\_\mathrm{ATTR}(p)\right\},\\&\mathrm{Cache}^{+} = \left\{\left(\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}, \left\{OATT^{+}_{j_l}\right\}_{l\le t}, p\right)\middle| p\in\mathrm{PERMS}, \mathrm{SATT}^{+}_{i_k}\in S^{+}\_\mathrm{ATTR}(p), \mathrm{OATT}^{+}_{j_l}\in O^{+}\_\mathrm{ATTR}(p)\right\},\\&\mathrm{Cache}^{*} = \left\{\left(\{S_u\_\mathrm{Attr}\}_{u\ge1}, \{O_v\_\mathrm{Attr}\}_{v\ge1}, p\right)\middle| p\in\mathrm{PERMS}, S_u\_\mathrm{Attr}\subseteq\mathrm{SATTR}, O_v\_\mathrm{Attr}\subseteq\mathrm{OATTR}\right\}.\end{aligned}\tag{12}$$

The basic idea is that under the condition that there is only a positive authorization policy for permission p, the *permit* response $++(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cach e^{++}, and the *deny* response $*-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cache^{*-}

Similarly, under the condition that there is only a negative authorization policy for permission p, the *deny* response $--(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cache^{--}, and the *permit* response $*+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cache^{*+}

Lastly, under the condition that there exist both positive and negative authorization policies for permission p, the *deny* response $-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cache^{-}, the *permit* response $+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ is used to build Cache^{+}, and the *deny* response $\sim(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ caused by satisfiable policy absence is used to build Cache*

Caches Cache^{++}, Cache^{*-}, Cache^{--}, Cache^{*+}, Cache^{-}, Cache^{+}, and Cache* are all initialized as empty. When a subject sends access request $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ to the PDP, the PDP returns the access control decision to the SDP, relying on the access control policies $\{\mathrm{policy}_i = <R_j, p, \mathrm{effect}_j>\}$ that are associated with permission p

Next, we mainly discuss the construction process of the cache when the PDP traverses the policy repository. For a certain permission p in the policy repository, there is either only one positive authorization policy, or only one negative authorization policy, or there are two policies, one positive authorization policy and one negative authorization policy. Here are different cases:

(1) There exists only one positive authorization policy for permission p. According to Algorithm 1 in Appendix, if the PDP permits the request, then it adds $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p)$ to *Cache*$^{++}$; otherwise, the PDP denies the request, and the construction of *Cache*$^{*-}$ is as follows

   (a) If attributes of both the subject and object in the request do not meet rules SR^{+}, OR^{+}, then $(\{\tilde{S}\_\mathrm{Attr}\}, \{\tilde{O}\_\mathrm{Attr}\}, p)$ is directly added to Cach e^{*-}, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$, and $\tilde{O}\_Attr = O\_Attr$

   (b) If, in the request, the subject's attributes do not meet SR^{+}, but the object's attributes meet OR^{+}, then $(\{\tilde{S}\_\mathrm{Attr}\}, \varnothing, p)$ is added to Cache^{*-}, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$

   (c) If, in the request, the subject's attributes meet S R^{+}, but the object's attributes do not meet OR^{+}, then $(\varnothing, \{\tilde{O}\_\mathrm{Attr}\}, p)$ is added to Cache^{*-}, where $\tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}$

(2) There exists only one negative authorization policy for permission p. Reference Algorithm 1 , if the PDP denies the request, then it adds $(\{\mathrm{SATT}^{-}_{i_k}\}_{k\le s}, \{\mathrm{OATT}^{-}_{j_l}\}_{l\le t}, p)$ to Cache^{--}; otherwise, the PDP permits the request, and the construction of Cache^{*+} is as follows:

   (a) If attributes of both the subject and object in the request do not meet rules SR^{-}, and OR^{-}, then $(\{\tilde{S}_\mathrm{Attr}\}, \{\tilde{O}_\mathrm{Attr}\}, p)$is directly added to Cach e^{*+}, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$, and $\tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}$

   (b) If, in the request, the subject's attributes do not meet SR^{-} but the object's attributes meet OR^{-}, then $(\{\tilde{S}\_\mathrm{Attr}\}, \varnothing, p)$ is added to Cache^{*+}, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$

   (c) If, in the request, the subject's attributes meet S R^{-} but the object's attributes do not meet OR^{-}, then $(\varnothing, \{\tilde{O}\_\mathrm{Attr}\}, p)$ is added to Cache^{*+}, where $\tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}$

(3) Both positive and negative authorization policies coexist. Denote two policies as $\mathrm{policy}^{+} = <R^{+}, p, \mathrm{permit}>$ and $\mathrm{policy}^{-} = <R^{-}, p, \mathrm{deny}>$, where $R^{+} = \mathrm{SR}^{+} \wedge \mathrm{OR}^{+}$, $R^{-} = \mathrm{SR}^{-} \wedge \mathrm{OR}^{-}$. According to Algorithm 3 in Appendix, the cache is built as follows:

   (a) If the PDP denies the request and the SDP receives the *deny* response $-(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$, then it adds $(\{\mathrm{SATT}^{-}_{i}\}_{i=1,2,\cdots,s}, \{\mathrm{OATT}^{-}_{j}\}_{j=1,2,\cdots,t}, p)$ to Cache^{-}. If only the minimal sets included by the current request are returned, the SDP may produce a *permit* response based on the content of the cache for subsequent requests. However, the request may consist of minimal sets of the negative authorization policy, which do not appear in the cache. The request should be denied according to the requirement for the SDP to be consistent with PDP. The PDP returns all minimal attribute sets that meet the negative authorization policy to avoid producing such an incorrect response

   (b) If the PDP permits the request and the SDP receives the *permit* response $+(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$, then it adds $(\{\mathrm{SATT}^{+}_{i_k}\}_{k\le s}, \{\mathrm{OATT}^{+}_{j_l}\}_{l\le t}, p)$ to Cache^{+}

   (c) If the PDP denies the request and the SDP receives a *deny* response $\sim(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ together with the information on the satisfiable policy absence. Then the building process of Cache* can be divided into three following cases:

      (i) If attributes of both the subject and object in the request do not meet rules SR^{+}, SR^{-}, OR^{+}, and OR^{-}, then $(\{\tilde{S}\_\mathrm{Attr}\}, \{\tilde{O}\_\mathrm{Attr}\}, p)$ is directly added to Cache*, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$, and $\tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}$

      (ii) If, in the request, the subject's attributes do not meet SR^{+} and SR^{-}, but the object's attributes meet OR^{+} and/or OR^{-}, then $(\{\tilde{S}\_Attr\}, \varnothing, p)$ is added to Cache*, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$

```
Input: $Cache = Cache^{++} \bigcup Cache^{*-}$; response $q$
Output: $Cache$
1: AddResponse(q)
2: if $q == ++(S\_Attr, O\_Attr, p)$ then
3:   if the record of $Cache^{+}$ w.r.t. permission $p$ is empty then
4:     add $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p)$ to $Cache^{++}$;
5:   else  // $(\{SATT^{+}_{i_{k'}}\}_{k'\le s}, \{OATT^{+}_{j_{l'}}\}_{l'\le t}, p) \in Cache^{++}$
6:     $Cache^{++} = \{((\{SATT^{+}_{i_{k'}}\}_{k'\le s} \bigcup \{SATT^{+}_{i_k}\}_{k\le s}\}, \{\{OATT^{+}_{j_{l'}}\}_{l'\le t} \bigcup \{OATT^{+}_{j_l}\}_{l\le t}\}, p)\}$;
7:   end if
8: else  // $q == *-(S\_Attr, O\_Attr, p)$
9:   if $q == q_1$ then
10:      if the record of $Cache^{*-}$ w.r.t. permission $p$ is empty then
11:        add $(\{\tilde{S}\_Attr\}, \{\tilde{O}\_Attr\}, p)$ to $Cache^{*-}$;
12:      else  // $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*-}$
13:        if $\exists\tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\exists\tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
14:          $Cache^{*-} = \{((\{(\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)$;
15:        else if $\exists\tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\forall\tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, O\_Attr U \tilde{O}'\_Attr$ then
16:          $Cache^{*-} = \{((\{(\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
17:        else if $\forall\tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, S\_Attr U \tilde{S}'\_Attr$ and $\exists\tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
18:          $Cache^{*-} = \{((\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
19:        else $Cache^{*-} = \{((\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
20:        end if
21:      end if
22:   else if $q == q_2$ then
23:      Step1:    // update $Cache^{*-}$
24:      if the record of $Cache^{*-}$ w.r.t. permission $p$ is empty then
25:        add $(\{\tilde{S}\_Attr\}, \varnothing, p)$ to $Cache^{*-}$;
26:      else //   $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*-}$
27:        if $\exists\tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ then
28:          $Cache^{*-} = \{((\{(\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\ge1}\}, p)\}$;
29:        else $Cache^{*-} = \{((\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p)\}$;
30:        end if
31:      end if
32:      Step2:    // update $Cache^{++}$
33:      if the record of $Cache^{++}$ w.r.t. permission $p$ is empty then
34:        add $(\varnothing, \{OATT^{+}_{j_l}\}_{l\le t}, p)$ to $Cache^{++}$;
35:      else $Cache^{++} = \{((\{SATT^{+}_{i_{k'}}\}_{k'\le s}, \{\{OATT^{+}_{j_{l'}}\}_{l'\le t} \bigcup \{OATT^{+}_{j_l}\}_{l\le t}\}, p)\}$;
36:      end if
37:   else if $q == q_3$ then
38:      Step1:    // update $Cache^{*-}$
39:      if the record of $Cache^{*-}$ w.r.t. permission $p$ is empty then
40:        add $(\varnothing, \{\tilde{O}\_Attr\}, p)$ to $Cache^{*-}$;
41:      else   // $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*-}$
42:        if $\exists\tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
43:          $Cache^{*-} = \{((\{\tilde{S}_u\_Attr\}_{u\ge1}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
44:        else $Cache^{-} = \{((\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
45:        end if
46:      end if
47:      Step2:   //update $Cache^{++}$
48:      if the record of $Cache^{++}$ w.r.t. permission $p$ is empty then
49:        add $(\{SATT^{+}_{i_k}\}_{k\le s}, \varnothing, p)$ to $Cache^{++}$;
50:      else $Cache^{++} = \{((\{\{SATT^{+}_{i_{k'}}\}_{k'\le s} \bigcup \{SATT^{+}_{i_k}\}_{k\le s}\}, \{OATT^{+}_{j_{l'}}\}_{l'\le t}, p)\}$;
51:      end if
52:   end if
53: end if
```

Algorithm 1: Cache construction algorithm for ABAC(P) Model.

```
Input: request $(S\_Attr, O\_Attr, p)$, $Cache^{++}$, $Cache^{*-}$
Output: permit; deny
1: $EvaluateRequest\ (S\_Attr, O\_Attr, p)$;
2: for all $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p) \in Cache^{++}$ do
3:    compare $S\_Attr$ with all $SATT^{+}_{i_k}$ and $O\_Attr$ with all $OATT^{+}_{j_l}$;
4:    if $(S\_Attr \supseteq SATT^{+}_{i_k}) \wedge (O\_Attr \supseteq OATT^{+}_{j_l})$ then
5:        return permit;
6:    end if
7: end for
8: for all $(\{\tilde{S}_u\_Attr\}_{u\ge 1}, \{\tilde{O}_v\_Attr\}_{v\ge 1}, p) \in Cache^{*-}$ do
9:    compare $S\_Attr$ with all $\tilde{S}_u\_Attr$ and $O\_Attr$ with all $\tilde{O}_v\_Attr$;
10:   if $(S\_Attr \subseteq \tilde{S}_u\_Attr) \vee (O\_Attr \subseteq \tilde{O}_v\_Attr)$ then
11:       return deny;
12:   else send the request to PDP
13:   end if
14: end for
```

ALGORITHM 2: Access control decision algorithm by the SDP in ABAC(P) model.

(iii) If, in the request, the subject's attributes meet SR^{+} and/or SR^{-}, but the object's attributes do not meet OR^{+} and OR^{-}, then $(\varnothing, \{\tilde{O}\_Attr\}, p)$ is added to $Cache^{*}$, where $\tilde{O}\_Attr = O\_Attr$

Note: if, in the request, the subject's attributes meet SR^{+} and the object's attributes meet OR^{-} or if the subject's attributes meet SR^{-} and the object's attributes meet OR^{+}, then nothing needs to be added to $Cache^{*}$

3.3. SDP Decision Rules. This subsection gives the SDP decision rules, which are used to infer accurate or approximate access control decisions from a cache.

Case 1. If there is only one positive authorization policy for permission p, the $Cache^{++}$ and $Cache^{*-}$ can be built. According to Algorithm 2 in Appendix, the following two rules are used to infer accurate or approximate access control decisions.

Rule^{++}. Assume $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p) \in Cache^{++}$, then request $(S^{*}\_Attr, O^{*}\_Attr, p)$ will be permitted by the SDP, if $\exists SATT^{+} \in \{SATT^{+}_{i_k}\}_{k\le s}, SATT^{+} \subseteq S^{*}\_Attr$ and $\exists OATT^{+} \in \{OATT^{+}_{j_l}\}_{l\le t}, OATT^{+} \subseteq O^{*}\_Attr$.

Theorem 9. *When a user sends a request, the SDP's decision to permit the request according to Rule*$^{++}$ *is safe.*

Proof. Assuming that SDP makes a permit decision for the request $(S^{*}\_Attr, O^{*}\_Attr, p)$ by $Rule^{++}$. According to $Rule^{++}$, $\exists i, j$ such that $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p) \in Cache^{++}$, where $SATT^{+}_{i} \in \{SATT^{+}_{i_k}\}_{k\le s}$, $OATT^{+}_{j} \in \{OATT^{+}_{j_l}\}_{l\le t}$ and $SATT^{+}_{i} \subseteq S^{*}\_Attr$, $OATT^{+}_{j} \subseteq O^{*}\_Attr$.

Based on cache construction, $SATT^{+}_{i}$ and $OATT^{+}_{j}$ satisfy rules SR^{+} and OR^{+} of $policy^{+} = <SR^{+} \wedge OR^{+}, p, permit>$, respectively; additionally, we also have $SATT^{+}_{i} \subseteq S^{*}\_Attr$, $OATT^{+}_{j} \subseteq O^{*}\_Attr$. So, the PDP should permit the request $(S^{*}\_Attr, O^{*}\_Attr, p)$. That is, the request permitted by SDP according to the $Rule^{++}$ would also be permitted by the PDP.

Rule^{*-}. Assume $(\{\tilde{S}_u\_Attr\}_{u\ge 1}, \{\tilde{O}_v\_Attr\}_{v\ge 1}, p) \in Cache^{*+}$, then the request $(S^{*}\_Attr, O^{*}\_Attr, p)$ will be denied by the SDP, if $\exists \tilde{S}\_Attr \in \{S_u\_Attr\}_{u\ge 1}, S^{*}\_Attr \subseteq \tilde{S}\_Attr$ or $\exists \tilde{O}\_Attr \in \{O_v\_Attr\}_{v\ge 1}, O^{*}\_Attr \subseteq \tilde{O}\_Attr$. □

Theorem 10. *When a user sends a request, the SDP's decision to deny the request according to Rule*$^{*-}$ *is consistent.*

Proof. Assuming that SDP makes a deny decision for request $(S^{*}\_Attr, O^{*}\_Attr, p)$ by $Rule^{*-}$. According to $Rule^{*-}$, $\exists(\{\tilde{S}_u\_Attr\}_{u\ge 1}, \{\tilde{O}_v\_Attr\}_{v\ge 1}, p) \in Cache^{*-}$, where $\exists \tilde{S}\_Attr \in \{S_u\_Attr\}_{u\ge 1}$, $S^{*}\_Attr \subseteq \tilde{S}\_Attr$ or $\exists \tilde{O}\_Attr \in \{O_v\_Attr\}_{v\ge 1}$, $O^{*}\_Attr \subseteq \tilde{O}\_Attr$.

Based on cache construction, $\tilde{S}\_Attr$ and $\tilde{O}\_Attr$ do not satisfy rules SR^{+} and OR^{+} of $policy^{+} = <SR^{+} \wedge OR^{+}, p, permit>$, respectively; additionally, we also have $S^{*}\_Attr \subseteq \tilde{S}\_Attr$, $O^{*}\_Attr \subseteq \tilde{O}\_Attr$. So, the PDP should deny the request $(S^{*}\_Attr, O^{*}\_Attr, p)$. That is, the request denied by SDP according to the $Rule^{*-}$ the PDP would also deny. □

Case 2. If there is only one negative authorization policy for permission p, the $Cache^{--}$ and $Cache^{*+}$ can be built. Reference Algorithm 2, the following two rules are used to infer accurate or approximate access control decisions.

Rule^{--}. Assume $(\{SATT^{-}_{i_k}\}_{k\le s}, \{OATT^{-}_{j_l}\}_{l\le t}, p) \in Cache^{--}$, then request $(S^{*}\_Attr, O^{*}\_Attr, p)$ will be denied by the SDP, if $\exists SATT^{-} \in \{SATT^{-}_{i_k}\}_{k\le s}, SATT^{-} \subseteq S^{*}\_Attr$ and $\exists OATT^{-} \in \{OATT^{-}_{j_l}\}_{l\le t}, OATT^{-} \subseteq O^{*}\_Attr$.

```
**Input:** $Cache = Cache^{-} \bigcup Cache^{+} \bigcup Cache^{*}$; response $q'$
**Output:** $Cache$
1: $AddResponse(\ q')$
2: if $q' == -(S\_Attr, O\_Attr, p)$ then
3:   add $(\{SATT^{-}_{i_k}\}_{i=1,2,\cdots,s}, \{OATT^{-}_{j_l}\}_{j=1,2,\cdots,t}, p)$ to $Cache^{-}$;
4: else if $q' == +(S\_Attr, O\_Attr, p)$ then
5:   if the record of $Cache^{+}$ w.r.t. permission $p$ is empty then
6:     add $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p)$ to $Cache^{+}$;
7:   else   // $(\{SATT^{+}_{i_{k'}}\}_{k'\le s}, \{OATT^{+}_{j_{l'}}\}_{l'\le t}, p) \in Cache^{+}$
8:     $Cache^{+} = \{(\{\{SATT^{+}_{i_{k'}}\}_{k'\le s} \bigcup \{SATT^{+}_{i_k}\}_{k\le s}\}, \{\{OATT^{+}_{j_{l'}}\}_{l'\le t} \bigcup \{OATT^{+}_{j_l}\}_{l\le t}\}, p)\}$;
9:   end if
10: else   // $q' == \sim (S\_Attr, O\_Attr, p)$
11:   if $q' == q'_1$ then
12:     if the record of $Cache^{*}$ w.r.t. permission $p$ is empty then
13:       add $(\{\tilde{S}\_Attr\}, \{\tilde{O}\_Attr\}, p)$ to $Cache^{*}$;
14:     else   // $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*}$
15:       if $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
16:         $Cache^{*} = \{((\{\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)$;
17:       else if $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\forall \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, O\_Attr \mathrm{U} \tilde{O}'\_Attr$ then
18:         $Cache^{*} = \{((\{(\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
19:       else if $\forall \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, S\_Attr \mathrm{U} \tilde{S}'\_Attr$ and $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
20:         $Cache^{*} = \{(\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
21:       else $Cache^{*} = \{(\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
22:       end if
23:     end if
24:   else if $q' == q'_2$ then
25:     Step1:     // update $Cache^{*}$
26:     if the record of $Cache^{*}$ w.r.t. permission $p$ is empty then
27:       add $(\{\tilde{S}\_Attr\}, \varnothing, p)$ to $Cache^{*}$;
28:     else   // $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*}$
29:       if $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\ge1}, \tilde{S}'\_Attr \subset S\_Attr$ then
30:         $Cache^{*} = \{((\{(\{\tilde{S}_u\_Attr\}_{u\ge1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\ge1}\}, p)\}$;
31:       else $Cache^{*} = \{(\{\{\tilde{S}_u\_Attr\}_{u\ge1} \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p)\}$;
32:       end if
33:     end if
34:     Step2:   // update $Cache^{+}$
35:     if the record of $Cache^{+}$ w.r.t. permission $p$ is empty then
36:       add $(\varnothing, \{OATT^{+}_{j_l}\}_{l\le t}, p)$ to $Cache^{+}$;
37:     else $Cache^{+} = \{(\{SATT^{+}_{i_{k'}}\}_{k'\le s}, \{\{OATT^{+}_{j_{l'}}\}_{l'\le t} \bigcup \{OATT^{+}_{j_l}\}_{l\le t}\}, p)\}$;
38:     end if
39:   else if $q' == q'_3$ then
40:     Step1':     // update $Cache^{*}$
41:     if the record of $Cache^{*}$ w.r.t. permission $p$ is empty then
42:       add $(\varnothing, \{\tilde{O}\_Attr\}, p)$ to $Cache^{*}$;
43:     else   // $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*}$
44:       if $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\ge1}, \tilde{O}'\_Attr \subset O\_Attr$ then
45:         $Cache^{*} = \{(\{\tilde{S}_u\_Attr\}_{u\ge1}\}, \{(\{\tilde{O}_v\_Attr\}_{v\ge1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
46:       else $Cache^{*} = \{(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\{\tilde{O}_v\_Attr\}_{v\ge1} \bigcup \{\tilde{O}\_Attr\}\}, p)\}$;
47:       end if
48:     end if
49:     Step2':   // update $Cache^{+}$
50:     if the record of $Cache^{+}$ w.r.t. permission $p$ is empty then
51:       add $(\{SATT^{+}_{i_k}\}_{k\le s}, \varnothing, p)$ to $Cache^{+}$;
```

Algorithm 3: Continued.

```
52:     else $Cache^+ = \{(\{\{SATT^+_{i_{k'}}\}_{k'\le s} \bigcup \{SATT^+_{i_k}\}_{k\le s}\}, \{OATT^+_{j_{l'}}\}_{l'\le t}, p)\}$;
53:     end if
54:   else if the subject's attributes meet $SR^+$ and the object's attributes meet $OR^-$ then
55:     goto Step2$'$;
56:   else the subject's attributes meet $SR^-$ and the object's attributes meet $OR^+$
57:     goto Step2;
58:   end if
59: end if
```

Algorithm 3: Cache construction algorithm for ABAC(H) model.

Theorem 11. *When a user sends a request, the SDP's decision to deny the request according to Rule^{--} is consistent.*

Proof. Assuming that SDP makes a deny decision for the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ by Rule^{--}. According to Rul e^{--}, $\exists i, j$such that $(\{\mathrm{SATT}^-_{i_k}\}_{k\le s}, \{\mathrm{OATT}^-_{j_l}\}_{l\le t}, p) \in \mathrm{Cache}^{--}$, where $\mathrm{SATT}^-_i \in \{\mathrm{SATT}^-_{i_k}\}_{k\le s}$, $\mathrm{OATT}^-_j \in \{\mathrm{OATT}^-_{j_l}\}_{l\le t}$ and $\mathrm{SATT}^-_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^-_j \subseteq O^*\_\mathrm{Attr}$.

Based on cache construction, SATT^-_i and OATT^-_j satisfy rules SR^- and OR^- of $\mathrm{policy}^- = <\mathrm{SR}^-\wedge\mathrm{OR}^-, p, \mathrm{deny}>$, respectively; additionally, we also have $\mathrm{SATT}^-_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^-_j \subseteq O^*\_\mathrm{Attr}$. So, the PDP should deny the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$. That is, the request denied by SDP according to the Rule^{--} the PDP would also deny.

Rule^{*+}. Assume $(\{\tilde{S}_u\_\mathrm{Attr}\}_{u\ge1}, \{\tilde{O}_v\_\mathrm{Attr}\}_{v\ge1}, p) \in$ Cach e^{*+}, then request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ will be permitted by the SDP, if $\exists\tilde{S}\_\mathrm{Attr} \in \{\tilde{S}_u\_\mathrm{Attr}\}_{u\ge1}$, $S^*\_\mathrm{Attr} \subseteq \tilde{S}\_\mathrm{Attr}$ or $\exists\tilde{O}\_\mathrm{Attr} \in \{\tilde{O}_v\_\mathrm{Attr}\}_{v\ge1}$, $O^*\_\mathrm{Attr} \subseteq \tilde{O}\_\mathrm{Attr}$. □

Theorem 12. *When a user sends a request, the SDP's decision to permit the request according to Rule^{*+} is safe.*

Proof. Assuming that SDP makes a permit decision for request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ by Rule^{*+}. According to Rul e^{*+}, $\exists(\{\tilde{S}_u\_\mathrm{Attr}\}_{u\ge1}, \{\tilde{O}_v\_\mathrm{Attr}\}_{v\ge1}, p) \in \mathrm{Cache}^{*+}$, where $\exists\tilde{S}\_\mathrm{Attr} \in \{S_u\_\mathrm{Attr}\}_{u\ge1}$, $S^*\_\mathrm{Attr} \subseteq \tilde{S}\_\mathrm{Attr}$ or $\exists\tilde{O}\_\mathrm{Attr} \in \{O_v\_\mathrm{Attr}\}_{v\ge1}$, $O^*\_\mathrm{Attr} \subseteq \tilde{O}\_\mathrm{Attr}$.

Based on cache construction, $\tilde{S}\_\mathrm{Attr}$ and $\tilde{O}\_\mathrm{Attr}$ do not satisfy rules SR^- and OR^- of $\mathrm{policy}^- = <\mathrm{SR}^-\wedge\mathrm{OR}^-, p, \mathrm{deny}>$, respectively; additionally, we also have $S^*\_\mathrm{Attr} \subseteq \tilde{S}\_\mathrm{Attr}$, $O^*\_\mathrm{Attr} \subseteq \tilde{O}\_\mathrm{Attr}$. So, the PDP should permit the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$. That is, the request permitted by SDP according to the Rule^{*+} would also be permitted by the PDP. □

Case 3. If both positive and negative authorization policies coexist for permission p, the Cache$^-$, Cache$^+$, and Cache* can be built. According to Algorithm 4 in Appendix, the following three rules are used to infer accurate or approximate access control decisions.

Rule$^-$. Assume $(\{\mathrm{SATT}^-_i\}_{i=1,2,\cdots,s}, \{\mathrm{OATT}^-_j\}_{j=1,2,\cdots,t}, p) \in \mathrm{Cache}^-$, then request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ will be denied by the SDP, if $\exists\mathrm{SATT}^- \in \{\mathrm{SATT}^-_i\}_{i=1,2,\cdots,s}$, $\mathrm{SATT}^- \subseteq S^*\_\mathrm{Attr}$ and $\exists\mathrm{OATT}^- \in \{\mathrm{OATT}^-_j\}_{j=1,2,\cdots,t}$, $\mathrm{OATT}^- \subseteq O^*\_\mathrm{Attr}$.

Theorem 13. *When a user sends a request, the SDP's decision to deny the request according to Rule$^-$ is consistent.*

Proof. Assuming that SDP makes a deny decision for the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ by Rule$^-$. According to Rule$^-$, $\exists i, j$ such that $(\{\mathrm{SATT}^-_u\}_{u=1,2,\cdots,s}, \{\mathrm{OATT}^-_v\}_{v=1,2,\cdots,t}, p) \in \mathrm{Cache}^-$, where $\mathrm{SATT}^-_i \in \{\mathrm{SATT}^-_u\}_{u=1,2,\cdots,s}$, $\mathrm{OATT}^-_j \in \{\mathrm{OATT}^-_v\}_{v=1,2,\cdots,t}$ and $\mathrm{SATT}^-_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^-_j \subseteq O^*\_\mathrm{Attr}$.

Based on cache construction, SATT^-_i and OATT^-_j satisfy rules SR^- and OR^- of $\mathrm{policy}^- = <\mathrm{SR}^-\wedge\mathrm{OR}^-, p, \mathrm{deny}>$, respectively; additionally, we also have $\mathrm{SATT}^-_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^-_j \subseteq O^*\_\mathrm{Attr}$. So the PDP should deny the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$. That is, the request denied by SDP according to the Rule$^-$ the PDP would also deny.

Rule$^+$. Assume $(\{\mathrm{SATT}^+_{i_k}\}_{k\le s}, \{\mathrm{OATT}^+_{j_l}\}_{l\le t}, p) \in \mathrm{Cache}^+$, then request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ will be permitted by the SDP, if $\exists\mathrm{SATT}^+ \in \{\mathrm{SATT}^+_{i_k}\}_{k\le s}$, $\mathrm{SATT}^+ \subseteq S^*\_\mathrm{Attr}$ and $\exists$O ATT$^+ \in \{\mathrm{OATT}^+_{j_l}\}_{l\le t}$, $\mathrm{OATT}^+ \subseteq O^*\_\mathrm{Attr}$. □

Theorem 14. *When a user sends a request, the SDP's decision to permit the request according to Rule$^+$ is safe.*

Proof. Assuming that SDP makes a permit decision for the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ by Rule$^+$. According to Rule$^+$, $\exists i, j$ such that $(\{\mathrm{SATT}^+_{i_k}\}_{k\le s}, \{\mathrm{OATT}^+_{j_l}\}_{l\le t}, p) \in \mathrm{Cache}^+$, where $\mathrm{SATT}^+_i \in \{\mathrm{SATT}^+_{i_k}\}_{k\le s}$, $\mathrm{OATT}^+_j \in \{\mathrm{OATT}^+_{j_l}\}_{l\le t}$ and $\mathrm{SATT}^+_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^+_j \subseteq O^*\_\mathrm{Attr}$.

Based on cache construction, SATT^+_i and OATT^+_j satisfy rules SR^+ and OR^+ of $\mathrm{policy}^+ = <\mathrm{SR}^+\wedge\mathrm{OR}^+, p, \mathrm{permit}>$, respectively; Additionally, we also have $\mathrm{SATT}^+_i \subseteq S^*\_\mathrm{Attr}$, $\mathrm{OATT}^+_j \subseteq O^*\_\mathrm{Attr}$. So the PDP should permit the request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$. That is, the request permitted by SDP according to the Rule$^+$ would also be permitted by the PDP.

Rule*. Assume $(\{\tilde{S}_u\_\mathrm{Attr}\}_{u\ge1}, \{\tilde{O}_v\_\mathrm{Attr}\}_{v\ge1}, p) \in \mathrm{Cache}^*$, then request $(S^*\_\mathrm{Attr}, O^*\_\mathrm{Attr}, p)$ will be denied by the SDP, if $\exists\tilde{S}\_\mathrm{Attr} \in \{\tilde{S}_u\_\mathrm{Attr}\}_{u\ge1}$, $S^*\_\mathrm{Attr} \subseteq \tilde{S}\_\mathrm{Attr}$ or $\exists\tilde{O}\_\mathrm{Attr} \in \{\tilde{O}_v\_\mathrm{Attr}\}_{v\ge1}$, $O^*\_\mathrm{Attr} \subseteq \tilde{O}\_\mathrm{Attr}$. □

```
Input: request $(S\_Attr, O\_Attr, p)$, $Cache^{-}$, $Cache^{+}$, $Cache^{*}$
Output: permit; deny
1: EvaluateRequest $(S\_Attr, O\_Attr, p)$;
2: for all $(\{SATT^{-}_{i_k}\}_{i=1,2,\cdots,s}, \{OATT^{+}_{j_l}\}_{j=1,2,\cdots,t}, p) \in Cache^{-}$ do
3:   compare $S\_Attr$ with all $SATT^{-}_{i_k}$ and $O\_Attr$ with all $OATT^{-}_{j_l}$;
4:   if $(S\_Attr \supseteq SATT^{-}_{i_k}) \wedge (O\_Attr \supseteq OATT^{-}_{j_l})$ then
5:     return deny;
6:   end if
7: end for
8: for all $(\{SATT^{+}_{i_k}\}_{k\le s}, \{OATT^{+}_{j_l}\}_{l\le t}, p) \in Cache^{+}$ do
9:   compare $S\_Attr$ with all $SATT^{+}_{i_k}$ and $O\_Attr$ with all $OATT^{+}_{j_l}$;
10:  if $(S\_Attr \supseteq SATT^{+}_{i_k}) \wedge (O\_Attr \supseteq OATT^{+}_{j_l})$ then
11:    return permit;
12:  end if
13: end for
14: for all $(\{\tilde{S}_u\_Attr\}_{u\ge1}, \{\tilde{O}_v\_Attr\}_{v\ge1}, p) \in Cache^{*}$ do
15:  compare $S\_Attr$ with all $\tilde{S}_u\_Attr$ and $O\_Attr$ with all $\tilde{O}_v\_Attr$;
16:  if $(S\_Attr \subseteq \tilde{S}_u\_Attr) \vee (O\_Attr \subseteq \tilde{O}_v\_Attr)$ then
17:    return deny;
18:  else send the request to PDP
19:  end if
20: end for
```

Algorithm 4: Access control decision algorithm by the SDP in ABAC(H) model.

Theorem 15. *When a user sends a request, the SDP's decision to deny the request according to Rule* is consistent.*

Proof. Assuming that SDP makes a deny decision for request $(S^*\_\text{Attr}, O^*\_\text{Attr}, p)$ by Rule*. According to Rule*, $\exists(\{\tilde{S}_u\_\text{Attr}\}_{u\ge1}, \{\tilde{O}_v\_\text{Attr}\}_{v\ge1}, p) \in \text{Cache}^*$, where $\exists\tilde{S}\_\text{Attr} \in \{S_u\_\text{Attr}\}_{u\ge1}$, $S^*\_\text{Attr} \subseteq \tilde{S}\_\text{Attr}$ or $\exists\tilde{O}\_\text{Attr} \in \{O_v\_\text{Attr}\}_{v\ge1}$, and $O^*\_\text{Attr} \subseteq \tilde{O}\_\text{Attr}$.

Based on cache construction, $\tilde{S}\_\text{Attr}$ and $\tilde{O}\_\text{Attr}$ do not satisfy rules SR$^+$ and OR$^+$ of policy$^+$ = <SR$^+$∧OR$^+$, p, permit >, respectively; or $\tilde{S}\_\text{Attr}$ and $\tilde{O}\_\text{Attr}$ do not satisfy rules SR$^-$ and OR$^-$ of policy$^-$ = <SR$^-$∧OR$^-$, p, deny >, respectively; additionally, we also have $S^*\_\text{Attr} \subseteq \tilde{S}\_\text{Attr}$, $O^*\_\text{Attr} \subseteq \tilde{O}\_\text{Attr}$. So, the PDP should deny the request $(S^*\_\text{Attr}, O^*\_\text{Attr}, p)$. That is, the request denied by SDP according to the Rule^{*-} the PDP would also deny.

The symbol $'' \succ ''$ is used to indicate the inference rule reference priority, if there is only one positive authorization policy for permission p, and the priority of the following two rules is Rule^{++} ≻ Rule^{*-}. If there is only one negative authorization policy for permission p, the priority of the following two rules is Rule^{--} ≻ Rule^{*+}. If both positive and negative authorization policies coexist for permission p, the priority of the following three rules is Rule$^-$ ≻ Rule$^+$ ≻ Rule*. Once a rule is applied to a request, the SDP stops evaluating the request against lower priority rules. □

3.4. Cache Update. When the SDP cannot make a decision on the request, the request needs to be sent to the PDP, and then the cache may be updated.

Under the condition, both positive and negative authorization policies coexist, once the SDP receives the deny response $-(S\_\text{Attr}, O\_\text{Attr}, p)$, and then, all minimal sets are added to Cache$^-$, so Cache$^-$ does not need to be updated. The update of Cache$^+$ and Cache* are as follows:

(1) If a request $(S\_\text{Attr}, O\_\text{Attr}, p)$ satisfies only the positive authorization policy but not the negative one, the PDP permits the request, and the SDP receives the permit response $+(S\_\text{Attr}, O\_\text{Attr}, \text{p})$. The update of Cache$^+$ is divided into the following cases:

Case 1. If the record of Cache$^+$ w.r.t. permission p is empty, then the corresponding record is added to Cache$^+$ according to the building process of Cache$^+$ that is given in Section 3.2.

Case 2. If $(\{\text{SATT}^+_{i_{k'}}\}_{k'\le s}, \{\text{OATT}^+_{j_{l'}}\}_{l'\le t}, \text{p}) \in \text{Cache}^+$, then Cache$^+$is updated by replacing $(\{\text{SATT}^+_{i_{k'}}\}_{k'\le s}, \{\text{OATT}^+_{j_{l'}}\}_{l'\le t}, \text{p})$ with $(\{\{\text{SATT}^+_{i_{k'}}\}_{k'\le s} \bigcup \{\text{SATT}^+_{i_k}\}_{k\le s}\}, \{\{\text{OATT}^+_{j_{l'}}\}_{l'\le t} \bigcup \{\text{OATT}^+_{j_l}\}_{l\le t}\}, \text{p})$, where $\text{SATT}^+_{i_k} \in \text{S}^+\_\text{ATTR(p)}$, $\text{SATT}^+_{i_k} \subseteq \text{S\_Attr}, \text{k} \le \text{s}$, and $\text{OATT}^+_{j_l} \in \text{O}^+\_\text{ATTR(p)}, \text{OATT}^+_{j_l} \subseteq \text{O\_Attr}, \ 1 \le \text{t}$

(2) If both positive and negative authorization policies are not satisfied, then the PDP denies the request, and the SDP receives a *deny* response $\sim(\text{S\_Attr}, \text{O\_Attr}, p)$ with the information on the satisfiable policy absence. Then, the updating process of Cache* is divided into the following cases:

Case 1. If attributes of both subject and object do not meet rules SR^+, SR^-, OR^+, OR^-, the $Cache^*$ is updated as follows:

(1) If the record of $Cache^*$ w.r.t. permission p is empty, then the corresponding record is added to $Cache^*$ according to the building process of $Cache^*$ that is given in Section 3.2

(2) If $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p) \in Cache^*$, then $Cache^*$ is updated as follows:

   (a) If $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, \tilde{O}'\_Attr \subset O\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{(\{\tilde{S}_u\_Attr\}_{u\geq1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\geq1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{S}\_Attr = S\_Attr$, and $\tilde{O}\_Attr = O\_Attr$

   (b) If $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, \tilde{S}'\_Attr \subset S\_Attr$ and $\forall \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, O\_Attr \not\subseteq \tilde{O}'\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{(\{\tilde{S}_u\_Attr\}_{u\geq1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\geq1} \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{S}\_Attr = S\_Attr$, and $\tilde{O}\_Attr = O\_Attr$

   (c) If $\forall \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, S\_Attr \not\subseteq \tilde{S}'\_Attr$ and $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, \tilde{O}'\_Attr \subset O\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{\{\tilde{S}_u\_Attr\}_{u\geq1} \bigcup \{\tilde{S}\_Attr\}\}, \{(\{\tilde{O}_v\_Attr\}_{v\geq1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{S}\_Attr = S\_Attr$, and $\tilde{O}\_Attr = O\_Attr$

   (d) If $\forall \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, S\_Attr \not\subseteq \tilde{S}'\_Attr$ and $\forall \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, O\_Attr \not\subseteq \tilde{O}'\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{\{\tilde{S}_u\_Attr\}_{u\geq1} \bigcup \{\tilde{S}\_Attr\}\}, \{\{\tilde{O}_v\_Attr\}_{v\geq1} \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{S}\_Attr = S\_Attr$, and $\tilde{O}\_Attr = O\_Attr$

Case 2. If the subject's attributes do not meet SR^+ and SR^-, but the object's attributes meet OR^+ and/or OR^-, then $Cache^*$ and $Cache^+$ are updated as follows:

(1) Update $Cache^*$:

   (a) If the record of $Cache^*$ w.r.t. permission p is empty, then the corresponding record is added to $Cache^*$ according to the building process of $Cache^*$ that is given in Section 3.2

   (b) If $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p) \in Cache^*$, then $Cache^*$ is updated as follows:

      (i) If $\exists \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, \tilde{S}'\_Attr \subset S\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u_Attr\}_{u\geq1}, \{\tilde{O}_v_Attr\}_{v\geq1}, p)$with $(\{(\{\tilde{S}_u\_Attr\}_{u\geq1} \setminus \tilde{S}'\_Attr) \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$, where $\tilde{S}\_Attr = S\_Attr$

      (ii) If $\forall \tilde{S}'\_Attr \in \{\tilde{S}_u\_Attr\}_{u\geq1}, S\_Attr \not\subseteq \tilde{S}'\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u_Attr\}_{u\geq1}, \{\tilde{O}_v_Attr\}_{v\geq1}, p)$with $(\{\{\tilde{S}_u\_Attr\}_{u\geq1} \bigcup \{\tilde{S}\_Attr\}\}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$, where $\tilde{S}\_Attr = S\_Attr$

(2) Update $Cache^+$:

   (a) If the record of $Cache^+$ w.r.t. permission p is empty, then $(\varnothing,\{OATT^+_{j_l}\}_{l\leq t}, p)$ is added to $Cache^+$, where $OATT^+_{j_l} \in O^+\_ATTR(p)$, and $OATT^+_{j_l} \subseteq O\_Attr, l \leq t$

   (b) If$(\{SATT^+_{i_{k'}}\}_{k'\leq s}, \{OATT^+_{j_{l'}}\}_{l'\leq t}, p) \in Cache^+$,then $Cache^+$ is updated by replacing $(\{SATT^+_{i_{k'}}\}_{k'\leq s}, \{OATT^+_{j_{l'}}\}_{l'\leq t}, p)$ with $(\{SATT^+_{i_{k'}}\}_{k'\leq s}, \{\{OATT^+_{j_{l'}}\}_{l'\leq t} \bigcup \{OATT^+_{j_l}\}_{l\leq t}\}, p)$, where $OATT^+_{j_l} \in O^+\_ATTR(p), OATT^+_{j_l} \subseteq O\_Attr, l \leq t$

Case 3. If the subject's attributes meet SR^+ and/or SR^-, but the object's attributes do not meet OR^+ and OR^-, then $Cache^*$ and $Cache^+$ are updated as follows:

(1) Update $Cache^*$:

   (a) If the record of $Cache^*$ w.r.t. permission p is empty, then the corresponding record is added to $Cache^*$ according to the building process of $Cache^*$ that is given in Section 3.2

   (b) If $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p) \in Cache^*$, then $Cache^*$ is updated as follows:

      (i) If $\exists \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, \tilde{O}'\_Attr \subset O\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{(\{\tilde{O}_v\_Attr\}_{v\geq1} \setminus \tilde{O}'\_Attr) \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{O}\_Attr = O\_Attr$

      (ii) If $\forall \tilde{O}'\_Attr \in \{\tilde{O}_v\_Attr\}_{v\geq1}, O\_Attr \not\subseteq \tilde{O}'\_Attr$, then $Cache^*$ is updated by replacing $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\tilde{O}_v\_Attr\}_{v\geq1}, p)$ with $(\{\tilde{S}_u\_Attr\}_{u\geq1}, \{\{\tilde{O}_v\_Attr\}_{v\geq1} \bigcup \{\tilde{O}\_Attr\}\}, p)$, where $\tilde{O}\_Attr = O\_Attr$

(2) Update $Cache^+$:

   (a) If the record of $Cache^+$ w.r.t. permission p is empty, then $(\{SATT^+_{i_k}\}_{k\leq s},\varnothing,p)$ is added to $Cache^+$, where $SATT^+_{i_k} \in S^+\_ATTR(p)$, and $SATT^+_{i_k} \subseteq S\_Attr, k \leq s$

Table 1: Comparison of different approaches.

Scheme	Policy expression	Granularity	Authorization recycling	Application scenario
Reference [5]	Flexibility	Fine grained	No	Specific application
Reference [9]	Flexibility	Fine grained	No	General purpose
Reference [15]	—	—	Yes	General purpose
Ours	Flexibility	Fine grained	Yes	General purpose

(b) If $(\{\mathrm{SATT}^{+}_{i_{k'}}\}_{k'\leq s}, \{\mathrm{OATT}^{+}_{j_{l'}}\}_{l'\leq t}, p) \in \mathrm{Cache}^{+}$, then Cache^{+} is updated by replacing $(\{\mathrm{SAT\ T}^{+}_{i_{k'}}\}_{k'\leq s}, \{\mathrm{OATT}^{+}_{j_{l'}}\}_{l'\leq t}, p)$ with $(\{\{\mathrm{SATT}^{+}_{i_{k'}}\}_{k'\leq s} \bigcup \{\mathrm{SATT}^{+}_{i_k}\}_{k\leq s}\}, \{\mathrm{OATT}^{+}_{j_{l'}}\}_{l'\leq t}, p)$, where $\mathrm{SATT}^{+}_{i_k} \in \mathrm{S}^{+}\_\mathrm{ATTR}(p)$, and $\mathrm{SATT}^{+}_{i_k} \subseteq \mathrm{S\_Attr}, k \leq s$

Case 4. If the subject's attributes meet SR^{+} and the object's attributes meet OR^{-}; or if the subject's attributes meet SR^{-} and the object's attributes meet OR^{+}, then Cache^{-} does not need to be updated. The update of Cache^{+} is as follows:

(1) If the subject's attributes meet SR^{+}, then the update of Cache^{+} refer to (2) in Case 3

(2) If the object's attributes meet OR^{+}, then the update of Cache^{+} refer to (2) in Case 2

If there exists only one positive authorization policy for permission p, the update of Cache^{++} refer to Cache^{+}; the update of Cache^{*+} refer to Cache^{*}

If there exists only one negative authorization policy for permission p, the update of Cache^{--} refer to Cache^{+}; the update of Cache^{*+} refer to Cache^{*}

Above, we provide the authorization recycling method under the hybrid policy, including how to construct the cache, define reasonable decision rules, and update the cache.

The environment attributes E_Attr can be added to the presented ABAC model. Consequently, the access request will be changed to (S_Attr, O_Attr, E_Attr, p), and the access control policy will be $\mathrm{policy} = <\mathrm{SR}\wedge\mathrm{OR}\wedge\mathrm{ER}, p, \mathrm{effect}>$, where ER represents the environment rule. Only if the subject, object, and environment attribute sets in the access request satisfy the rule SR∧OR∧ER, the request satisfies the access control policy. We can similarly provide an authorization recycling method. This article does not put the environment attribute into the model for brevity.

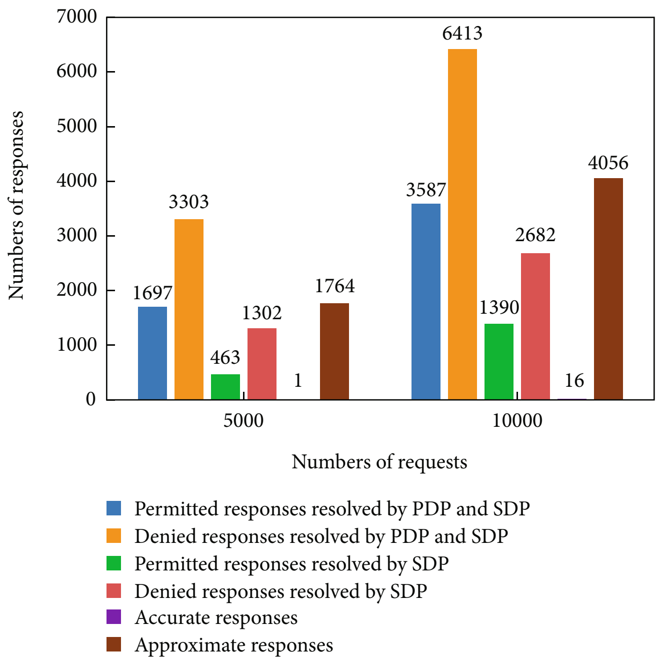
4. Experimental Results and Evaluation

4.1. Comparison with Different Approaches. Table 1 compares the proposed work with some existing work in terms of flexibility of policy expression, access granularity, authorization recycling, and application scenarios. Compared to [15], our approach is for ABAC and supports negative authorization; ours is more flexible, has higher policy expression ability, and supports fine-grained access control. Compared to [5, 9], our emphasis is on providing authorization recycling method, including building and updating the cache and making precise or approximate decisions. Our solution is based on the general-purpose ABAC model and is not directed to specific applications.

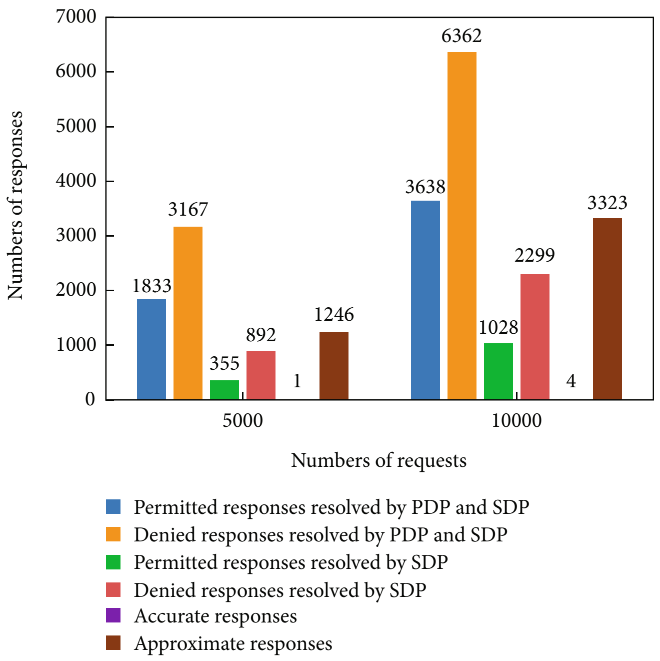
4.2. Test Results. A small-scale experiment was conducted to evaluate the hit rate of the authorization recycling approach for the ABAC model under the hybrid policy. This approach was carried out on a PC with an Intel (R) Core (TM) i7-8750h CPU at 2.20 GHz-2.21 GHz, 8GB memory, and Windows 10 x64. The application software was JetBrains PyCharm Community Edition 2019.3, and the interpreter was Python 3.7.

The experimental procedure was as follows. First, 50 subject attributes, 50 object attributes, and 10000 different permissions were randomly generated. For one-third of permissions, there was only one positive authorization policy for each permission; for another one-third of permissions, there was only one negative authorization policy for each permission; and lastly, for the remaining permissions, there were both positive and negative authorization policies for each permission. Each policy had a rule $R = \mathrm{SR}\wedge\mathrm{OR}$, where SR represented a Boolean expression of subject attributes. Attributes in the expression were randomly selected from 50 subject attributes in a random amount. Further, OR was a Boolean expression of object attributes, and the attribute in the expression was randomly selected from 50 object attributes in a random amount. Next, we randomly generated a request set containing 10000 access requests, and $(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$ represented a request, where S_Attr denoted the subject attribute set, whose attributes were randomly selected from 50 subject attributes in a random amount, and O_Attr represented the object attribute set, whose attributes were randomly selected from 50 object attributes in a random amount, and lastly, p represented the permission to access. The average number of both subject attributes and object attributes in each policy is about 13, and the average number of both subject attributes and object attributes in each access request is about 25.

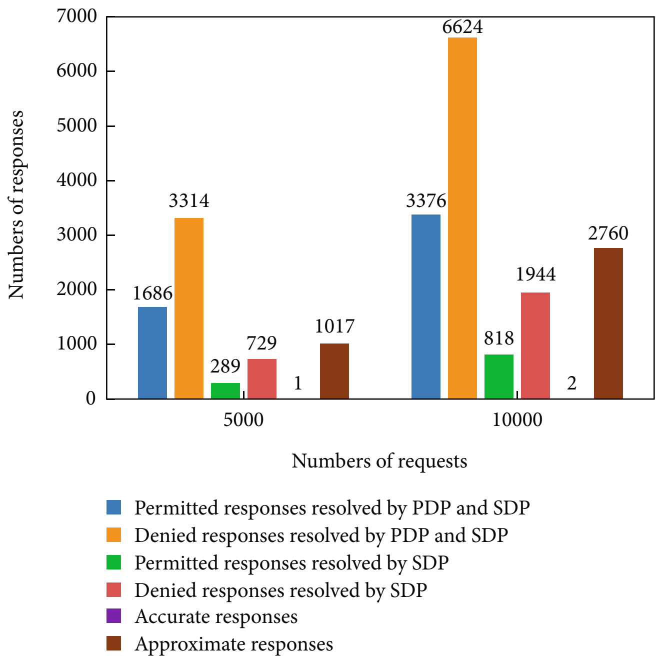
The method proposed in this article was more suitable for those situations where the same user or similar users in the same organization have more concentrated permission access over a certain period. So we randomly accessed 200, 500, 800, 1000, 2000, and 3000 out of 10000 permissions. When a subject sent an access request for permission, the PDP searched for the policy associated with that permission. Then, it calculated the minimal subject and object attribute sets according to Definitions 7 and 8, respectively. The obtained test results are shown in Figure 2.



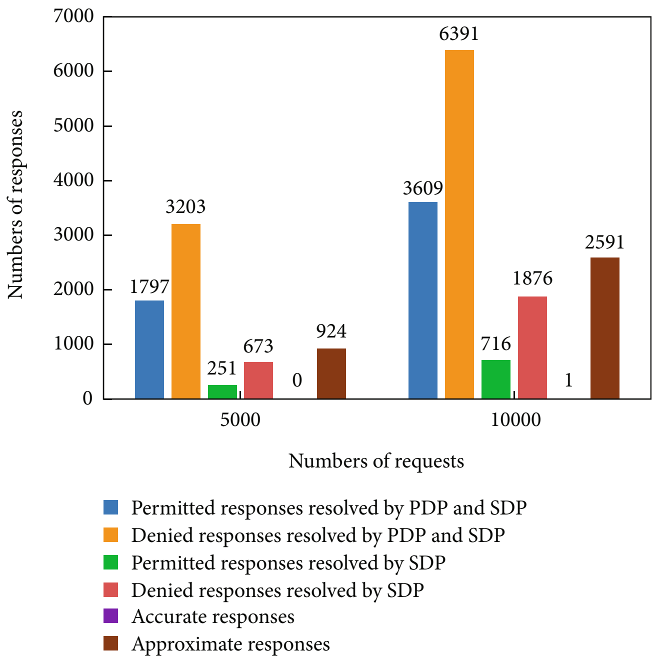
(a) Random access to 200 permissions, the number of various responses



(b) Random access to 500 permissions, the number of various responses

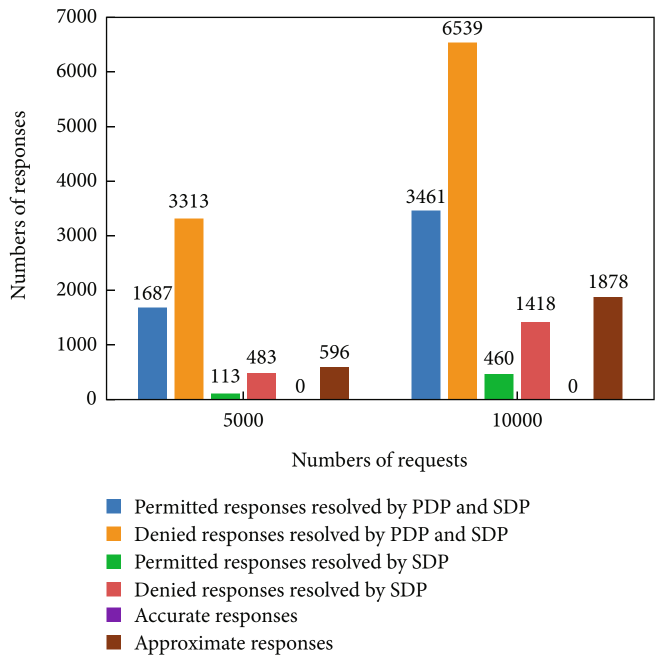


(c) Random access to 800 permissions, the number of various responses

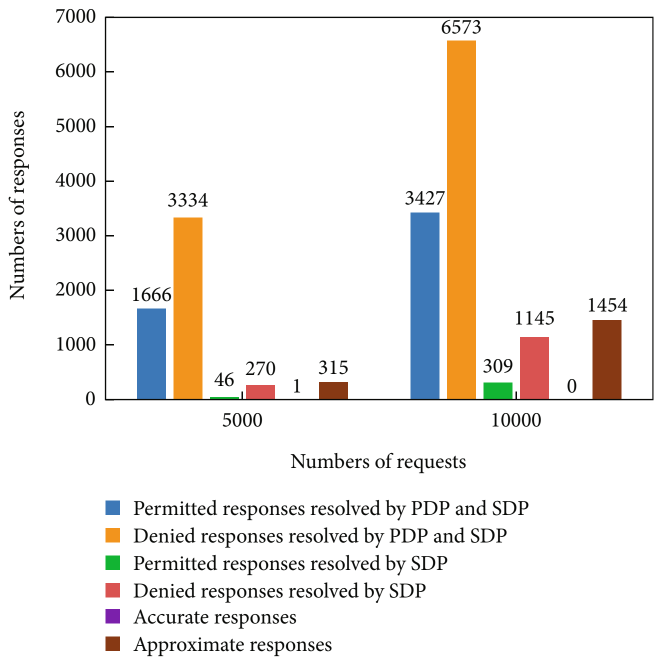


(d) Random access to 1000 permissions, the number of various responses

Figure 2: Continued.



(e) Random access to 2000 permissions, the number of various responses



(f) Random access to 3000 permissions, the number of various responses

Figure 2: Test results of authorization recycling for the ABAC model.

The cache was initially empty, and the SDP used the information fetched from the PDP to gradually build and update the cache. As the number of requests increased, the SDP achieved more information from the PDP, and the cache records continued to increase. From Figure 2, it can be seen that the number of approximate responses increased with the number of requests and was much larger than the number of precise responses. In the case of the same number of requests sent, the more permission for random access, the fewer the requests resolved locally by the SDP. In the case of certain access permissions, the more the number of requests, the more the requests are resolved locally by the SDP, and the hit rate gradually increases, as shown in Figure 3.

In addition, we also tested the decision time in two conditions: on-cache and off-cache. Under the off-cache condition, users' requests are directly decided by the PDP and sent to the PEP for enforcement. In the on-cache condition, the PDP decides on the users' requests first, the matching policy contents or requests are cached in the corresponding caches by the SDP, and then later requests are solved by the SDP according to SDP decision rules. In the on-cache test, we randomly accessed 100 permissions from 10,000 permissions, sent 1,000 requests for these permissions, and recorded the decision time. Then, we used the exact requests and their related policies and did the same test with off-cache. We conducted the same test for five rounds, and the decision time was recorded and averaged. The average response time for each round is listed in Table 2.

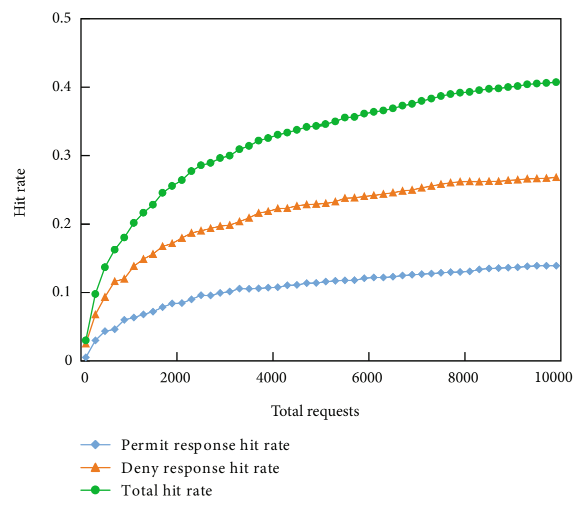
The average decision time in the on-cache condition is 119.743 ms less than that in the off-cache condition. Moreover, if the network communication time is considered, the average decision time with on-cache will be shorter than that with off-cache.

Making access control decisions with the cache is more advantageous than without the cache. In addition, if the PDP cannot be reached, the SDP can handle some requests using the cache to cover up the failure of the PDP. The proposed system can overcome the shortcomings of the PDP to some extent and reduce the workload of the PDP.

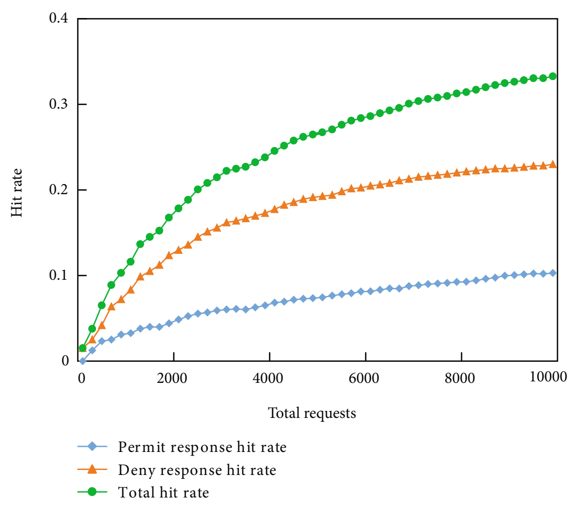
5. Related Work

The ABAC has been widely used in distributed environments in recent years due to its flexibility, scalability, and fine-granularity access control. Li et al. [31] described the overall framework of the ABAC. Hu et al. [26] provided a comprehensive definition and guidance for implementing the ABAC, becoming the NIST ABAC standard. In this work, an ABAC model, which supports hybrid authorizations, is proposed. The proposed model relies on Boolean expressions of subject and object attributes, capturing common characteristics of the ABAC practices.

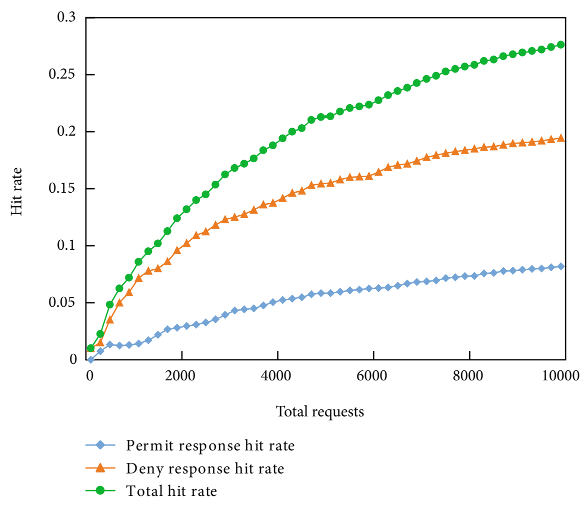
In access control, caching authorization responses [11, 12] and reuse of them can increase the usability and performance of the access control system, so authorization recycling has attracted much attention recently. Beznosov [13] presented approximate authorization recycling and proposed to use the publish-subscribe architecture to share and actively recycle authorization. However, this approach uses many precomputed authorizations in the delivery channel and actively recycles authorizations on an immediate basis; it also extends the precise caching mechanism. In [17], a publish-subscribe model was proposed to transfer access requests and responses between the application and authorization server. This work presents an accurate and approximate authorization recycling method for the ABAC.



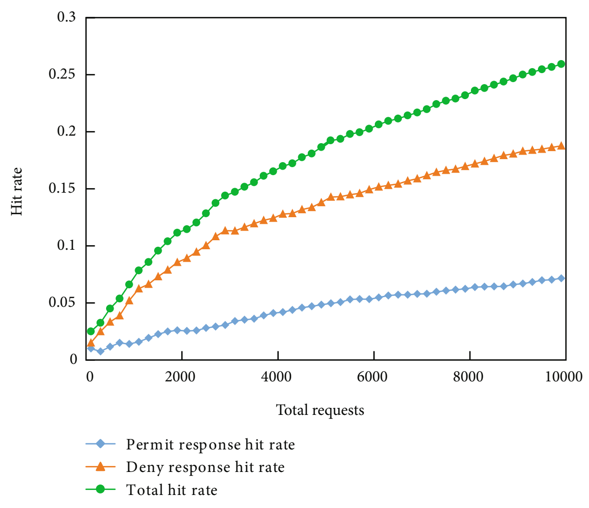
(a) Random access to 200 permissions, the hit rate variation with the number of requests



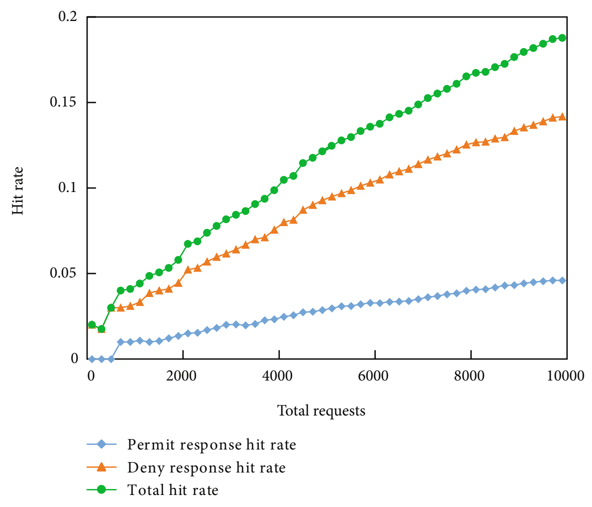
(b) Random access to 500 permissions, the hit rate variation with the number of requests



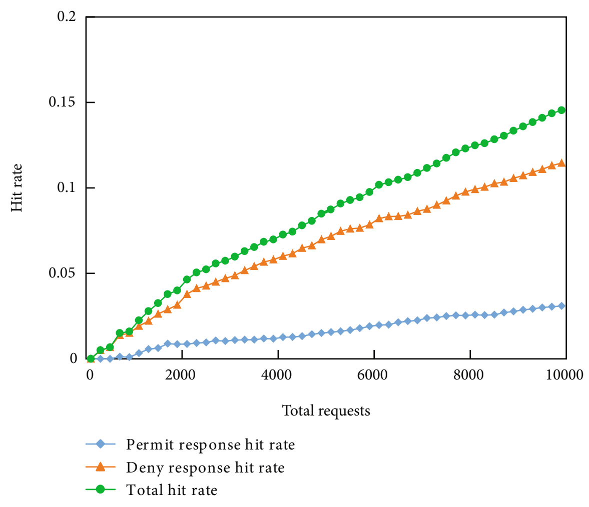
(c) Random access to 800 permissions, the hit rate variation with the number of requests



(d) Random access to 1000 permissions, the hit rate variation with the number of requests



(e) Random access to 2000 permissions, the hit rate variation with the number of requests



(f) Random access to 3000 permissions, the hit rate variation with the number of requests

Figure 3: The trend of the hit rate when accessing different numbers of permissions.

Table 2: Average decision time in different states.

Rounds	Round 1	Round 2	Round 3	Round 4	Round 5	Average
Average decision time per request with cache (ms)	312.677	316.53	374.909	334.898	307.645	329.332
Average decision time per request without cache (ms)	395.275	457.177	476.651	420.729	495.544	449.075

Table 3: Possible response contents of ABAC(P) model.

Response q	Corresponding content
$q = ++(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$	$++(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)\colon \triangleq <\mathrm{permit}, \left(\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}, \left\{\mathrm{OATT}^{+}_{j_l}\right\}_{l\le t}, p\right) >$, where $\mathrm{SATT}^{+}_{i_k} \in S^{+}\_\mathrm{ATTR}(p)$ and $\mathrm{SATT}^{+}_{i_k} \subseteq S\_\mathrm{Attr}, k \le s$ $\mathrm{OATT}^{+}_{j_l} \in O^{+}\_\mathrm{ATTR}(p)$ and $\mathrm{OATT}^{+}_{j_l} \subseteq O\_\mathrm{Attr}, l \le t$
$q = * -(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)$	$S\_\mathrm{Attr}, O\_\mathrm{Attr}$ do not meet the rule: $q = q_1 == * -(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)\colon \triangleq <\mathrm{deny}, \left(\left\{\tilde{S}\_\mathrm{Attr}\right\}, \left\{\tilde{O}\_\mathrm{Attr}\right\}, p\right) >$, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$ and $\tilde{O}\_\mathrm{Attr} = O\_\mathrm{Attr}$ S_Attr does not meet the rule, but O_Attr meets the rule: $q = q_2 = * -(S\_Attr, O\_Attr, p)\colon \triangleq <\mathrm{deny}, \left(\left\{\tilde{S}\_\mathrm{Attr}\right\}, \varnothing, p\right) >$, where $\tilde{S}\_\mathrm{Attr} = S\_\mathrm{Attr}$; $\left\{\mathrm{OATT}^{+}_{j_l}\right\}_{l\le t}$, where $\mathrm{OATT}^{+}_{j_l} \in O^{+}\_\mathrm{ATTR}(p), \mathrm{OATT}^{+}_{j_l} \subseteq O\_\mathrm{Attr}, l \le t$ S_Attr meets the rule, but O_Attr does not meet the rule: $q = q_3 = * -(S\_\mathrm{Attr}, O\_\mathrm{Attr}, p)\colon \triangleq <\mathrm{deny}, \left(\varnothing, \left\{\tilde{O}\_\mathrm{Attr}\right\}, p\right) >$, where $\tilde{O}\_\mathrm{Attr} = O\_Attr$; $\left\{\mathrm{SATT}^{+}_{i_k}\right\}_{k\le s}$, where $\mathrm{SATT}^{+}_{i_k} \in S^{+}\_\mathrm{ATTR}(p), \mathrm{SATT}^{+}_{i_k} \subseteq S\_\mathrm{Attr}, k \le s$

Crampton et al. [14] proposed an authorization recycling model-SAAM based on the BLP model, which uses the relationship between the subject and object of the previous responses to deduce approximate responses. This model formally defines SAAM and introduces the SDP. Namely, SAAM provides an alternative source for access control decisions. It builds a general-purpose, application-independent authorization recycling framework to achieve a new authorization response from the SDP. It reuses the previous response to infer information about the underlying access control policy and makes an approximate response consistent with the policy. Compared to the approach presented in [14], our authorization recycling approach is for the ABAC model, in which permission access is based on rules about subject and object attributes. It should be noted that handling access requests in the ABAC model is much more complicated than in the BLP model. Consequently, authorization recycling in the ABAC is more complex than that of the BLP model. Additionally, our authorization recycling supports the hybrid policy, while the BLP model supports only the closed-world policy.

In [16, 32], collaborative secondary authorization recycling was studied. In [16], an authorization recycling method using CSAR in the cloud computing system, which uses cooperative caching of access control decisions to improve the hit rate, was developed. Wei et al. [32] discussed the collaboration between multiple SDPs and proposed a collaborative secondary authorization recovery method. This method takes advantage of a large-scale, distributed cooperative system to improve the hit rate of access control decisions. Application servers share their recycled authorization results with each other. Compared to analyses in [16, 32], in this work, the cooperation between SDPs is not considered. However, this work focuses on making full use of all information fetched from the PDP when making secondary authorization decisions to increase the hit rate. Additionally, authorization recycling for a relatively complex ABAC model is presented.

The RABC authorization recycling was studied in [15]. Crampton et al. [15] proposed the caching strategy SAAMRBAC. They provided an authorization recycling algorithm, which includes the compression of an authorization cache. The SDP can effectively infer precise and approximate decisions from the cached data to overcome the shortcomings of the PDP. In contrast to the RBAC, the ABAC has the advantages of high flexibility, scalability, and fine granularity. Inspired by the results in [15], a general-purpose ABAC model, which supports hybrid authorizations, is developed in this work. However, in the RBAC, roles associated with permissions and only positive authorization is considered. In the ABAC, a combination of several attributes associated with a permission is considered, which makes authorization recycling more complex but worthy.

6. Conclusion

This paper presents a general-purpose ABAC model that supports the hybrid policy. It also introduces the authorization recycling approach for this model, specifies reasonable rules to construct the cache, and effectively renders precise and approximate decisions from the cache. In the presented model, the Boolean expressions of subject and object

Table 4: Possible response contents of ABAC(H) model.

Response q'	Corresponding content
$q' = -(S\_\text{Attr}, O\_\text{Attr}, p)$	$-(S\_\text{Attr}, O\_\text{Attr}, p)\text{:} \triangleq \text{<deny}, \left(\left\{\text{SATT}^-_{i_k}\right\}_{k\le s}, \left\{\text{OATT}^-_{j_l}\right\}_{l\le t}, p\right) >$, where $\text{SATT}^-_{i_k} \in S^-\_\text{ATTR}(p), i = 1, 2, \cdots, s,$ $OATT^-_{j_l} \in O^-\_\text{ATTR}(p), j = 1, 2, \cdots, t\,;$
$q' = +(S\_\text{Attr}, O\_\text{Attr}, p)$	$+(S\_\text{Attr}, O\_\text{Attr}, p)\text{:} \triangleq \text{<permit}, \left(\left\{\text{SATT}^+_{i_k}\right\}_{k\le s}, \left\{\text{OATT}^+_{j_l}\right\}_{l\le t}, p\right) >$, where $\text{SATT}^+_{i_k} \in S^+\_\text{ATTR}(p)$ and $\text{SATT}^+_{i_k} \subseteq S\_\text{Attr}, k \le s$ $\text{OATT}^+_{j_l} \in O^+\_\text{ATTR}(p)$ and $\text{OATT}^+_{j_l} \subseteq O\_\text{Attr}, l \le t$
$q' = \sim(S\_\text{Attr}, O\_\text{Attr}, p)$	Attributes of both the subject and object in the request do not meet rules SR^+, SR^-, OR^+ and OR^- $q' = q'_1 = \sim(S\_\text{Attr}, O\_\text{Attr}, p)\text{:} \triangleq \text{<deny}, \left(\left\{\tilde{S}\_\text{Attr}\right\}, \left\{\tilde{O}\_\text{Att}r\right\}, p\right) >$, where $\tilde{S}\_\text{Attr} = S\_\text{Attr}$ and $\tilde{O}\_\text{Attr} = O\_\text{Att}r$ The subjects attributes do not meet SR^+ and SR^-, but the objects attributes meet OR^+ and/or OR^- $q' = q'_2 = \sim(S\_\text{Attr}, O\_\text{Attr}, p)\text{:} \triangleq \text{<deny}, \left(\left\{\tilde{S}\_\text{Attr}\right\}, \varnothing, p\right) >$, where $\tilde{S}\_\text{Attr} = S\_\text{Attr};$ $\left\{\text{OATT}^+_{j_l}\right\}_{l\le t}$, where $\text{OATT}^+_{j_l} \in O^+\_\text{ATT}R(p), \text{OAT}T^+_{j_l} \subseteq O\_\text{Attr}, l \le t$ The subjects attributes meet SR^+ and/or SR^-, but the objects attributes do not meet OR^+ and OR^- $q' = q'_3 = \sim(S\_\text{Attr}, O\_\text{Attr}, p)\text{:} \triangleq \text{<deny}, \left(\varnothing, \left\{\tilde{O}\_\text{Attr}\right\}, p\right) >$, where $\tilde{O}\_\text{Attr} = O\_\text{Attr};$ $\left\{\text{SATT}^+_{i_k}\right\}_{k\le s}$, where $\text{SATT}^+_{i_k} \in S^+\_\text{ATTR}(p), \text{SATT}^+_{i_k} \subseteq S\_\text{Attr}, k \le s$

attributes are used, so an application system can extend this model, adding other attributes, such as environmental attributes. In the cache construction process, two principles are adopted: (1) full use of information fetched from the PDP in making authorization decisions at the SDP; and (2) do not use the PDP if there is any chance to make a safe and consistent authorization decision at the SDP, since fetching a new response increases communication cost and searching overhead. Finally, a small-scale experiment is conducted to evaluate the effectiveness of the proposed approach.

It should be noted that the ABAC models and their workflows can be much more complicated in practice, and authorization policies may change all the time in some ABAC systems. This work does not consider the impact of policy changes on the cache. The ABAC cache supporting dynamic policy change will be studied in our future work. The applicability of the proposed approach in practical environments will also be considered in the future.

Appendix

Cache Construction and Access Control Decision Algorithm

If there exists only one positive authorization policy for permission p. The possible contents of the response are as listed in Table 3.

The corresponding cache construction (build and update) algorithm is as follows.

The corresponding access control decision algorithm by the SDP is as follows.

In the case of only one negative authorization policy for permission p, the model becomes a dual model of that there is only one positive authorization policy for permission p, cache construction, and access control decision algorithm are similar and will not be repeated here.

If both positive and negative authorization policies coexist for permission p, the possible contents of the response are listed in Table 4.

The corresponding cache construction (build and update) algorithm is as follows.

The corresponding access control decision algorithm by the SDP is as follows.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research work was supported by the National Natural Science Foundation of China (grant numbers: 61862059, 61562085).

References

[1] R. Sandhu and P. Samarati, “Access control: principle and practice,” *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[2] B. W. Lampson, "Protection," *ACM Sigops Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

[3] D. Bell and J. Lapadula, "Secure computer systems: a mathematical model," *The MITRE Corporation*, vol. 4, pp. 229–263, 1973.

[4] D. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramoull, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 224–274, 2001.

[5] Z. H. Xin, L. Liu, and G. Hancke, "AACS: attribute-based access control mechanism for smart locks," *Symmetry*, vol. 12, no. 6, 2020.

[6] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Dynamic groups and attribute-based access control for next-generation smart cars," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pp. 61–72, 2019.

[7] M. Gupta, F. Awaysheh, J. Benson, M. Alazab, F. Patwa, and R. Sandhu, "An attribute-based access control for cloud enabled industrial smart vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4288–4297, 2021.

[8] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Secure V2V and V2I communication in intelligent transportation using cloudlets," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 1912–1925, 2022.

[9] L. Karijmi, M. Abdelhakim, and J. Joshi, "Adaptive ABAC policy learning: a reinforcement learning approach," 2021, https://arxiv.org/abs/2105.08587.

[10] K. Beznosov, *Recycling Authorizations: Toward Secondary and Approximate Authorizations Model (SAAM).*, University of British Columbia, 2005.

[11] G. Karjoth, "Access control with IBM Tivoli access manager," *ACM Transactions on Information and System Security*, vol. 6, no. 2, pp. 232–257, 2003.

[12] K. Borders, X. Zhao, and A. Prakash, "CPOL: high-performance policy evaluation," in *Proceedings of the 12th ACM conference on Computer and Communications Security*, pp. 147–157, 2005.

[13] K. Beznosov, "Flooding and recycling authorizations," in *Proceedings of the 2005 workshop on New security paradigms - NSPW '05*, pp. 67–72, 2005.

[14] J. Crampton, W. Leung, and K. Beznosov, "The secondary and approximate authorization model and its application to Bell-LaPadula policies," in *Proceedings of the eleventh ACM symposium on Access control models and technologies - SACMAT '06*, pp. 111–120, 2006.

[15] Q. Wel, J. Crampton, K. Beznosov, and M. Ripeanu, "Authorization recycling in hierarchical RBAC systems," *ACM Transactions on Information and System Security*, vol. 14, no. 1, pp. 1–29, 2011.

[16] S. Reeja, "Role based access control mechanism in cloud computing using cooperative secondary authorization recycling method," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 10, pp. 444–450, 2012.

[17] Q. Wei, M. Ripeanu, and K. Beznosov, "Authorization using the publish-subscribe model," in *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 53–62, Sydney, NSW, Australia, 2013.

[18] M. U. Aftab, Z. Qin, K. Hussain et al., "Negative authorization by implementing negative attributes in attribute-based access control model for internet of medical things," in *2019 15th International Conference on Semantics, Knowledge and Grids (SKG)*, pp. 167–174, Guangzhou, China, 2019.

[19] P. Iyer and A. Masoumzadeh, "Mining positive and negative attribute-based access control policy rules," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pp. 161–172, Indianapolis, IN, USA, 2018.

[20] D. Brossard, G. Gebel, and M. Berg, "A systematic approach to implementing ABAC," in *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pp. 53–59, 2017.

[21] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334, Berkeley, CA, USA, 2007.

[22] H. Cui, R. H. Deng, J. Lai, X. Yi, and S. Nepal, "An efficient and expressive ciphertext-policy attribute-based encryption scheme with partially hidden access structures, revisited," *Computer Networks*, vol. 133, pp. 157–165, 2018.

[23] L. Lin, J. P. Huai, and X. X. Li, "Attribute-based access control policies composition algebra," *Journal of Software*, vol. 20, no. 2, pp. 403–414, 2009.

[24] K. Ge and B. Lang, "Research on the policy definition in the attribute based access control," *Microcomputer Information*, vol. 24, no. 33, pp. 7–9, 2008.

[25] Z. C. Zhong, *Research on Security Policy Optimization in Attribute-Based Access Control*, XIDIAN University, 2020.

[26] V. Hu, D. Ferraiolo, R. Kuhn et al., "Guide to attribute based access control (ABAC) definition and considerations," *National Institute of Standards and Technology Special Publication*, vol. 162-800, p. 47, 2014.

[27] OASIS, "The eXtensible access control markup language (XACML), version 3.0 plus errata 01, OASIS standard incorporating approved errata," 2017, http://docs.oasis-open.org/xacml/3.0/errata01/os/xacml-3.0-core-spec-errata01-os-complete.html.

[28] C. Liu, *Research on Conflict Detection and Resolution Method of ABAC Security Policy*, XIDIAN University, 2020.

[29] E. Bertino, P. Samarati, and S. Jajodia, "Authorizations in relational database management systems," in *Proceedings of the 1st ACM conference on Computer and communications security - CCS '93*, pp. 130–139, New York, NY, USA, 1993.

[30] J. J. Du and N. Helil, "Fine-grained and traceable key delegation for ciphertext-policy attribute-based encryption," *KSII Transactions on Information Systems*, vol. 15, no. 9, pp. 3274–3297, 2021.

[31] X. F. Li, D. G. Feng, Z. W. Chen, and Z. H. Fang, "Model for attribute based access control," *Journal of Communications*, vol. 29, no. 4, pp. 90–98, 2008.

[32] Q. Wei, M. Ripeanu, and K. Beznosov, "Cooperative secondary authorization recycling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 2, pp. 275–288, 2009.