Safety and Consistency of Mutable Attributes Using Quotas: A Formal Analysis

Mehrnoosh Shakarami and Ravi Sandhu
Institute for Cyber Security (ICS), Center for Security and Privacy Enhanced Cloud Computing (C-SPECC)
Department of Computer Science, University of Texas at San Antonio
San Antonio, Texas, USA
mehrnoosh.shakarami@my.utsa.edu, ravi.sandhu@utsa.edu

***Abstract*—Attribute-based Access Control (ABAC) systems make access decisions utilizing attributes of subjects, objects and environment with respect to a policy. Acquiring real-time values of these attributes is not practical in distributed multi-authority environments due to cost and performance considerations as well as intrinsic delays of distributed systems. So it is possible to make decisions based on outdated policy and attribute values resulting in access violations. This is known as the safety and consistency problem. This problem has been previously studied in trust negotiation and ABAC context. Previous works have assumed attributes to be immutable, to wit their values could be changed only via administrative actions. However, so far there is no research carried out in the context of mutable attributes, values of which could be changed as a result of users access.**

In this paper we investigate safety and consistency in the context of mutable subject attributes which introduces additional complexity to the problem. In particular, there might be multiple concurrent sessions manipulating the same mutable attribute. Therefore, in addition to exposure of the decision point to stale attribute values, safety and consistency can be compromised due to concurrent utilization of the same attribute. While the general consistency problem has vast literature in distributed systems arena, practical solutions are typically dependent on the specific application domain. We identify two categories of use cases of practical benefit in context of ABAC, which turn out to be amenable to quota-based solutions. We provide a formal analysis of the resulting solutions.

***Index Terms*—ABAC, safety, consistency, mutable attributes**

I. Introduction

Attribute Based Access Control (ABAC) regulates subjects' access to protected objects in the system using policies relying on subject, object and environment attributes. These attributes are typically supplied by multiple Attribute Authorities (AA) in a distributed provider network. ABAC is rapidly emerging as the preferred access control model since it subsumes traditional methods of access control (i.e., discretionary, mandatory and role-based) while providing additional capabilities [1–3].

Usually more than one subject attribute is required by policy to govern requested access. Attribute values are communicated in attribute credentials, which could be cryptographically signed (if obtained over an untrusted path) or unsigned (if obtained over a trusted path). Incrementally assembling attribute credentials with the inherent delays and network failures of distributed environments, exposes the risk of stale information to be provided to the decision point leading to access control violations, known as safety and consistency problem.

There are previous studies to limit the exposure of the decision point to outdated attribute values in ABAC environments through frequent attribute revocation checks [4] or attribute freshness checks [5], with the frequency to be determined by the system administrator based on tradeoff between tolerable staleness and the burden of frequent updates of attribute values.

Common in previous studies is consideration of attributes to be immutable in sense of UCON_{ABC} model [6], wherein attribute values can be changed only via administrative actions. A seminal contribution in UCON_{ABC} was the notion of mutable attributes, which change automatically as a consequence of utilizing the granted access.

Our central goal in this paper is to investigate the safety and consistency problem in the context of mutable attributes. Mutability adds further complication to establish consistency requirement and specifications, as it requires synchronization mechanisms in place to update attribute values. There is a rich body of literature in distributed environments dealing with concurrency. However, practical solutions are typically dependent on the specific application domain. We identify two categories of use cases of practical benefit in context of ABAC, which turn out to be amenable to quota-based solutions. More general treatment of consistency beyond quota-based solutions is beyond the scope of this paper. We develop a formal characterization of required consistency using refresh [5] in this context. We also observe that revocation is inappropriate to be used in the context of mutable attributes (which has been the traditional approach to attribute freshness for immutable attributes).

The paper is organized as follows. A review of related works and a review of UCON_{ABC} model is given in Section II. In Section III, we explain the problem and our system assumptions and definitions. Also, we give a classification of quota-based approach and its variations studied in this paper. Practical use-case scenarios based on this classification along with formal specification of two of them are presented in Section IV. In Section V, the contrast of refresh and revocation in our problem context as well as some consistency considerations are presented, followed by two proposed levels of consistency. Section VI concludes the paper.

978-1-7281-6741-1/19/$31.00 ©2019 IEEE
DOI 10.1109/TPS-ISA48467.2019.00010

II. Background and Related Work

A. Related Work

We define consistency problem informally as updating decision point with most recent values of required attributes in the soonest possible time. Lee and Winslett pioneered investigation of safety and consistency in trust negotiation systems [7, 8]. They propose four levels of consistency to avoid insecure decision making in policy-based authorization systems. Lee and Winslett extended their work in [9] and proposed three consistency levels for distributed proof construction in context-sensitive environments in which parts of the decision tree need to remain hidden for the sake of privacy.

Another related research is reported in [10, 11] in which authors determine the concept of stale-safety which concentrates on safe use of stale attributes by refreshing subject and object attributes in a way which limits the exposure of access decisions to outdated information. Authors try to limit the risk of utilizing the outdated information by relying on the latest time the attributes have known to be valid (refresh time). Their model only applies to a single attribute authority environment. Closest to our study is [4, 5], where [4] uses revocation check to possibly invalidate attribute values that change, while refresh is proposed in [5] to acquire the freshest attribute values.

Common to all prior works, attributes have been considered to be immutable in sense of UCON_{ABC} [6, 12]. Immutable attribute values can be updated only by administrative actions. Mutable attribute values, on the other hand, change as the consequence of access utilization by subjects. Usage of target object may result in updates of object or subject attributes before, during or after usage. Mutability is a crucial requirement of history-based systems or those with consumable attributes. One of the challenging issues regarding mutable attributes in distributed environments is using one attribute in concurrent accesses to the system, each of which might result in attribute value change. Concurrency control in distributed systems has been well studied in the literature [13–15], however there are very limited research works specifically regarding consistency problem of access control in distributed systems.

B. Background: $UCON_{ABC}$ Model

UCON_{ABC}[1] [6, 12] has been proposed as a family of reference models for usage control in which authorizations are based on attributes of involved parties. UCON extends traditional access control models by introducing three decision factors, namely Authorization, Obligation and Conditions. Authorizations regulate access to objects in the system based on attributes of involved parties and predefined access control policies. Obligations are requirements to be fulfilled by the subject in order to be conferred with the requested access. Conditions are environmental and system-wide prerequisites which need to be satisfied for the access to be granted.

UCON further extends access control with decision continuity and attribute mutability. Continuity provides ongoing authorization which re-evaluates granted access while being utilized by the subject. So, in addition to pre-authorization models which make access decisions once before granting access, there is ongoing control during access utilization. Mutability deals with the attribute changes as the side effect of subject's access. So instead of being admin-controlled, attributes are system-controlled which means updates are done by the system without involving any administrative actions. For instance, account balance decreases after each time it is used to buy an item. In this paper we focus on the pre-authorization models of UCON and do not consider continuity of enforcement during access.

[1]We call it UCON hereafter

III. Problem Statement and Assumptions

Assuming an ABAC model is in place, we discuss the safety and consistency problem for mutable attributes in this research. We assume attribute credentials are provided through different attribute authorities and there is a single decision point in the system. The main goal is to propose a practical approach to limit the exposure of the decision point to outdated attribute values. We concentrate on subject's attribute consistency supposing that policy as well as attributes of objects and environment are known with high assurance to the decision point.

Administrative changes of attribute values are always done at the AA for both mutable and immutable attributes. It is worth reminding that we consider attributes mutable if their changes are the consequence of access utilization by subjects. So, if any administrative change happens that should be managed administratively. As an example if the user's credit line changes, that change has to be managed and take effect via AA whereas using the credit line to purchase services is done automatically by the system. It is also possible to check the most recent values of mutable attributes with AA for each utilization, for example after each credit card utilization. However, it makes AA as a single point of failure and is as inefficient as any centralized approach.

There is a big space of analysis to deal with consistency problem in distributed environments with mutable attributes. Our analysis focuses on a quota-based approach in which every mutable attribute value would be treated as a quota and AA would delegate the quota to some predetermined distributed servers and those servers take care of utilization of delegated quotas and update them locally.

A. Quota-Based Approach

Quota for reusable resources could be considered as reimbursable deposit which is refunded after usage has finished [16]. There is another type of resources known as consumable which would decrease in amount at each access without being refunded. A system that allows a user up to five concurrent sessions for content streaming is an example of the former, since termination of a session enables another one to be initiated. A system that allows a total of five uses is an example of the latter; once consumed, the quota would not be refunded.

Access quota has been previously used in risk-based access control. Authors in [17] assigned quota to users and obligations in order to regulate access risks. Access quota has been used in [18] to specify a threshold on tolerable risk by the system through assigning quotas based on estimation of access needs during a specific period of time.

We assume each mutable attribute has a global limit known to the corresponding AA. This global limit can be managed in a centralized way by AA to be distributed and managed among users whose access requires utilizing that attribute. In another approach, the global limit could be delegated to local servers which would be responsible to distribute the global limit as local quotas to be manipulated through different access utilizations. The totality of local quotas distributed in this manner cannot exceed the global quota.

In any case, we recognize two approaches to apportion the global limit as follows.

1) Service-Based: In this approach AA assigns portions of the global limit to each service point, regardless of the users who are utilizing access to the service. So, a global limit would be set on concurrent number of service usage by all users. It is notable that provided global limit would be set for each service instance of a service. As an example, an Internet Service Provider supplies internet connection to hotel rooms. Each room's internet connection would be a service instance. Regardless of which users/devices are connected to the service, an overall internet usage limit would be set for all provided service instances.
2) User-Based: In this approach a subscriber (user) to a service would be assigned limited number of concurrent sessions to use the service. As an example, assign a specific amount of storage to every user on the cloud or put an upper limit on the number of concurrent sessions that each subscriber to a TV channel can have. Note that attribute-based access control could be done either idless (anonymous) as proposed in Idemix [19] or non-anonymous as we address it in user-based global limit assignment.

In both approaches a global count limit would determine the upper bound of usage which restrains the number of concurrent usage number of the service objects. Upper bound could be set either as a countdown which is consumable and non-refundable after each usage or it can be restored after access utilization has been completed.

B. Definitions and Assumptions

We examine the safety and consistency problem from the perspective of a single access decision point within a larger distributed ABAC authorization system (which would include multiple such decision points amongst other distributed components). Attributes of objects and environment and the policy are presumed to be known with high assurance to the decision point. Our focus is on subject attributes in this paper.

As previously discussed subject attributes could be either mutable or immutable. Regardless decision point is the entity which checks the policy and determines the set of attributes whose values should satisfy policy requirements. We call this set *relevant attributes set* or *relevant attributes* for short, following previous studies [4, 5, 7], which could be distributed over multiple attribute providers.

The set of relevant attributes might change over time as the decision point examines the policy expression which we assume is expressed in Disjunctive Normal Form. If an attribute's value does not fulfil policy requirements, it would be replaced with next conjunct in the same clause, so the set of relevant attributes changes.

Definition 1. We call the set of relevant attributes to the policy P at time t determined by decision point DP, the *view* of the decision point at time t and denote it as $V_{DP}^{P,t}$.

IV. Use Case Scenarios

In this section we discuss two sets of practical use case scenarios to illustrate how utilizing the conferred access may require updating mutable subject attributes. These use cases are provided in the context of ABAC and are amenable to quota-based approach. Throughout use case explanations we use quota-based solution to manage concurrent accesses of users to service instances which is a well-established approach to handle concurrency in distributed environments.

We consider pre-authorization and pre-obligation models with respect to UCON. In order to explain each use case we follow UCON notation of [6]. Moreover, we define following symbols to describe our use cases. U and S represent the set of all users and services in the system respectively. Each concurrent session of the user is shown as a User Instance (UI) (a.k.a subject). Considering the service as the object in our system, we represent each service instance as a Service Instance Object (SIO). The number of existing sessions is treated as a user's attribute which ought to appropriately change as new sessions are created or terminated while using the service.

$ATT(.)$ indicates the set of attributes for the entity enclosed in parenthesis. $X.Y$ denotes attribute Y of the entity X. $preUpdate(.)$ and $postUpdate(.)$ indicate functions which have to be done to update attribute values before usage is started and after usage is terminated respectively. The notation $allowed(s, o, a) \Rightarrow$ indicates necessary requirements for subject s to be allowed to do action a on object o.

A. Centralized Approach

AA distributes attribute values among entities to be used during access evaluation and utilization, so dissemination is done in a centralized way under a single authority. This strategy makes the AA as the single point of failure which monitors and tracks assigned limits.

Attribution of assigned portion could be done for all users of a specific service, a.k.a service-based, or it could be user-centric, a.k.a user-based, as follows. Either of following two approaches could also be changed to be processed as a global countdown limit, which means the global limit would be consumable and would not be refunded after each usage.

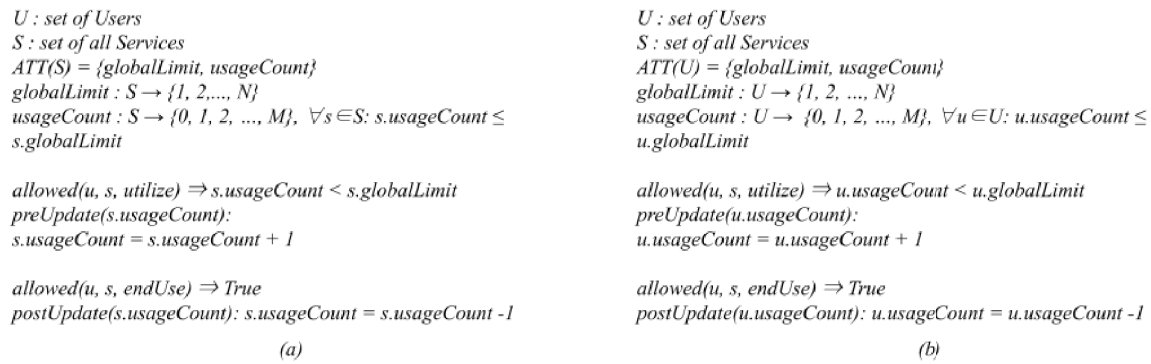

Fig. 1. Centralized Approach to Manage Global Limit: a) service-based b) user-based

1) Service-Based Distribution: In this type of distribution a global limit is assigned for all users of a service. The AA would manage sharing the service among different users scattered throughout local or distributed locations. Formal specification of this method is shown in Figure 1a. As an example, an Internet Service Provider supplies internet connection to hotel rooms. Regardless of which users/devices are connected to the internet, the central server would set a usage global limit on the service it provides to the hotel.

2) User-Based Distribution: This strategy disseminates the global limit between different users of a service. The global limit can determine the maximum number of service usage per user. As an example a subscriber to a service could have up to M concurrent sessions for using that service. Each time a user wants to start/end utilization of the service, AA would check the prerequisites (if any) and update mutable attributes accordingly. Formal specification is given in Figure 1b.

B. Distributed Quota-Based Approach

Although centralized approach provides the benefit of avoiding inconsistency because of its central management, it presents scalability and fault tolerance difficulties. To provide a system with better fault tolerance and to avoid AA from becoming the single point of failure, a distributed approach could be exploited. In this approach, a global limit which has been assigned to every mutable attribute would be distributed among some local servers which delegate their assigned share as *quotas* to each service/user throughout access assessment and practice. Furthermore, this limit could be allocated using a service-centric or user-centric approach as follows.

1) Service-Based Quota Distribution: A global limit would be set per service to be used by different users. This limit would be delegated to different service providers in the distributed environment which then could be allocated to different users of the system per request. As an example, consider a Software as a Service (SaaS) [20] which provides all users from a university's IP address up to a total of M concurrent sessions to use the software. Then the global limit could be further distributed among Service Instance Objects (SIO) as quotas to be assigned to different colleges and departments which could be managed locally. Formal specification is provided in Figure 2a.

2) User-Based Quota Distribution: In this approach a global limit would be allocated to local servers for each user. The global limit determines the maximum number of simultaneous service usage a user can have. A user (U) can have multiple concurrent user instances (UI) which is created with a predetermined share ($ui.Quota$) of its parents (the user who created that UI) global limit. The sum of all assigned quotas to different user instances cannot exceed their parent global limit. As an instance, a subscriber to a TV channel can have up to 5 concurrent live sessions and then this global limit can be used on different devices to watch that channel. Formal specification is provided in Figure 2b.

To delete one of the user's user instances, we enforce one of the two following approaches to ensure assigned quota to deleted user instance would be set free for further utilization.

- user instance which the user wants to be deleted, should have no service utilization ($ui.usageCount = 0$).
- terminate all services which are being practiced by to-be-deleted user instance. To satisfy this requirement, we enforce an obligation to be fulfilled by the user to first discontinue all user instance services and then proceed to delete the user instance. This approach has been used in Figure 2b.

The quota upper limit could also be set as a countdown limit which conveys that the quota is non-refundable after usage termination.

C. Distributed vs. Centralized Quota Management

Each of centralized and distributed quota management approaches discussed in Section IV has its own advantages and drawbacks. In the rest of this section, we provide some properties of each method of quota management to compare their pros and cons.

Property 1. Centralized quota management provides correct access control decision.

Proof. In the centralized approach, all required attribute credentials are kept in one place which is a highly assured AA.

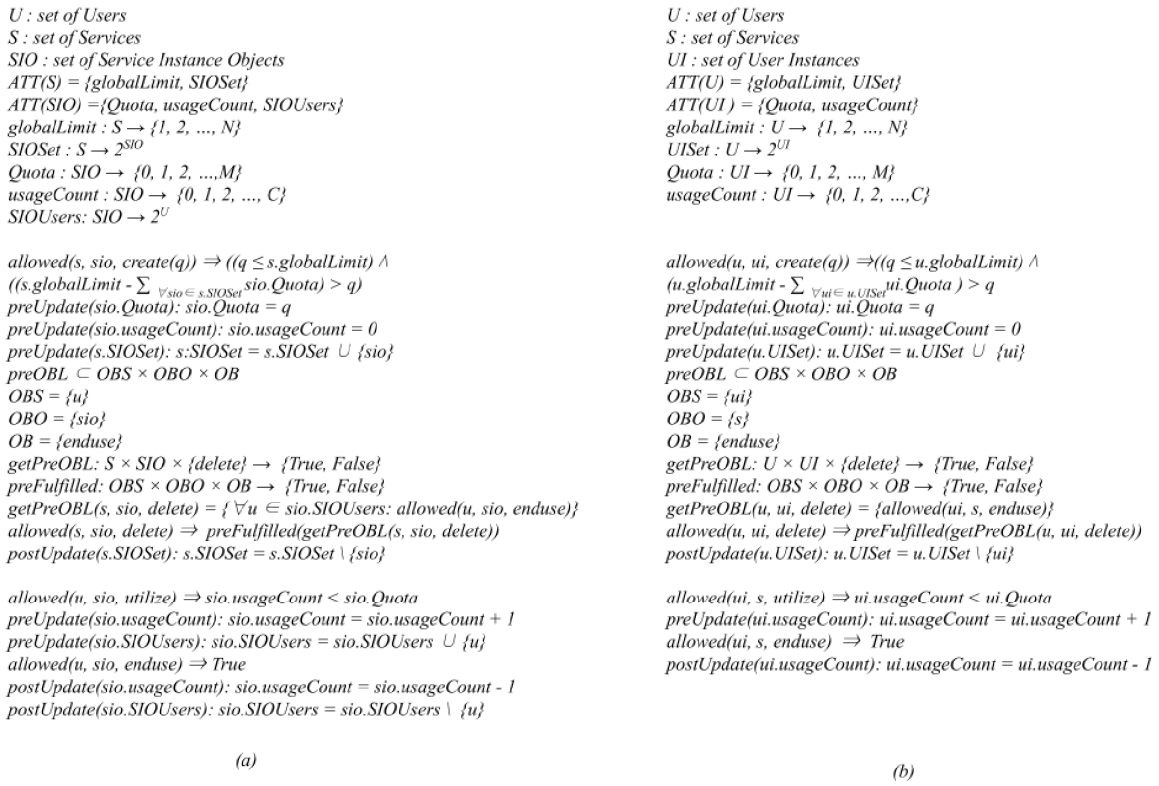

Fig. 2. Distributed Approach to Manage Global Limit: a) service-based b) user-based

As long as there is no network failure and post updates could succeed, this would result in granting access only when it is correct based upon the policy.

Property 2. Distributed quota management approach provides less availability and less utilization, comparing to the centralized approach.

Proof. It is possible to block an access in the distributed approach due to lack of available quota, while it would be conferred in the centralized approach. In distributed approach, the global limit of every attribute is dispersed between local distributors (servers) as quotas. If an access permission requires assessing a specific attribute, access would be granted provided a spare quota is available. If not, access would be denied. This could happen even if there are some spare quotas for the same attribute sitting unused on other servers. If this request was delivered to a system with a centralized quota management, access would not be denied as long as there is any available spare quota and it would be allocated as the global limit is managed centrally by the AA. So, even while post updates succeed, distributed approach could be deficient in availability and resource utilization comparing to centralized approach.

Property 3. Distributed quota management access provision is correct.

Proof. Based on Property 2, distributed approach would provide less availability compared with centralized approach. This conveys less access would be granted while applying distributed approach. In other words, granted accesses in distributed method is a subset of accesses granted in centralized one, which are correct based on Property 1.

V. Consistency Levels For Distributed Quota-Based Distribution Methods

In this section we only look at refresh-based solution as revocation is not applicable that which we discuss in Subsection V-A. Subsection V-B accentuates the fact that consistency problem would arise only when multiple (more than one) attributes are included in the view of decision point. Two consistency levels in Subsection V-C are provided to reduce decision point exposure to outdated values while its view contains both mutable and immutable attributes, recognizing that mutable attributes would increase the risk of exposure to stale values, a.k.a safety and consistency problem.

A. Revocation vs. Refresh

Authors in [5] compared the two possible ways to obtain latest attribute values as shown in Figure 3. While in revocation it is only possible to check if attribute credential is either valid or revoked, in refresh scenario new values of attributes could be returned in case of any changes other than revocation.

Fig. 3. Revocation vs. Refresh [5]

So instead of only invalidating the old value, the new value would be communicated to the requesting party.

Taking mutability into account, decision point needs to be updated by recent values of relevant attributes. Since revocation check only evaluates the validation of previous attribute values, it is appropriate only for immutable attributes which solely could be updated by administrative actions. If used for mutable attributes, revocation check is useful only if the attribute value has been revoked, but any changes in the value would not be reflected in revocation check response.

Both revocation and refresh scenarios are considered as pulling approaches, in which the recent attribute value information will be recovered via querying AA. We consider refresh scenario to be appropriate to obtain the freshest values of attributes in this work. However, if the global limit changes, there should be a pushing mechanism from AA to distributing servers to make them aware of the change. Consideration of this latter mechanism is out of scope for this paper.

B. Consistency Considerations For Mutable Attributes

Consistency problem arises when the decision point needs more than one attribute value to make an access decision. We call the set of required attributes to make an access decision *relevant attributes* as speccified in Definition 1. Practical use cases compliant to quota-based approach have been discussed in Section IV, all of which consider only one mutable attribute.

When relevant attributes include more than one attribute, there is always the risk of some attribute values to be outdated while the decision point trying to acquire other attributes' values from distributed attribute authorities. Previous research has been done toward definition of different consistency levels by imposing restrictions on timeliness of attribute checks [4, 5], but all attributes presumed to be immutable.

The following example demonstrates that the consistency problem when the set of relevant attributes include more than one attribute including mutable attributes as well.

Example 1. A user tries to create a backup of his phone contents on Apple iCloud. To grant the access to iCloud storage, decision point needs to confirm the validity of Apple ID which is considered as an immutable attribute as well as remaining storage assigned to that ID as a mutable attribute. Suppose a scenario in which the Apple ID has been validated previously and the decision point tries to check the remaining iCloud storage. It is possible that Apple ID has been invalidated while trying to acquire remained storage, which indicates an inconsistency situation. The reverse order of checks is also possible to cause consistency problem where iCloud storage has been consumed up with content from other devices connected to the same Apple ID, while checking the authenticity of the ID.

TABLE I
SUMMARY TABLE OF SYMBOLS

Symbol	Meaning
t_{req}	request time
t_d	decision time
c_i^m	i^{th} credential which is mutable
c_i^{im}	i^{th} credential which is immutable
c_i	i^{th} credential, regardless of being mutable/immutable
$t^i_{ref,k}$	time of k-th refresh of c_i^{im}
$t^i_{update,k}$	time of k-th refresh of c_i^m
$t^i_{start,k}$	attribute start time of c_i after k-th refresh
$t^i_{end,k}$	attribute expiration time of c_i after k-th refresh
$kmax(t)$	latest refresh of c_i before time t (c_i is determined by context)
$val^i_{kmax(t)}$	the value of c_i after $kmax(t)$-th refresh
$t^i_{ref,kmax(t)}$	time of $kmax(t)$-th refresh of c_i^{im}
$t^i_{update,kmax(t)}$	time of $kmax(t)$-th refresh of c_i^m
$t^i_{start,kmax(t)}$	attribute start time of c_i after $kmax(t)$-th refresh
$t^i_{end,kmax(t)}$	attribute expiration time of c_i after $kmax(t)$-th refresh

C. Formal Specification Of Consistency Levels

Before defining formal consistency levels, we emphasize that mutable attributes would be updated more frequently than immutable ones. As a justification, we remind the reader of the definition of each category of attributes. Immutable attributes are assigned and only could be changed by administrative actions, however mutable attribute values could change as a side effect of each access utilization. Since any access could change the relevant mutable attributes values, it is reasonable to assume mutable attribute changes more frequent.

Based on previous statement, unlike what has been defined in previous works to define consistency levels based on different recommended freshness/validity overlaps, we would not assume the same degree of freedom for system administrators to decide lower levels of freshness overlap to be provided to the decision point. So, at least all of mutable attributes have to be refreshed after the request is submitted to the decision point. It is notable that we consider the request time as the anchor point in that it is the closest recognizable point in time to the decision time and we want to check the value of mutable attributes at the closest possible point to the decision time.

Nonetheless for immutable attribute we could rely on refresh results which have been done before the request time. In contrast to mutable attribute values which are available locally at local distributors in distributed approach and so could be updated at any arbitrary time, immutable attributes updates require the decision point to consult the AA which might not be possible at any desired time in distributed environments. That said we can give more freedom about timeliness of immutable attributes. However, simultaneous freshness of mutable and immutable attributes might not necessarily be provided.

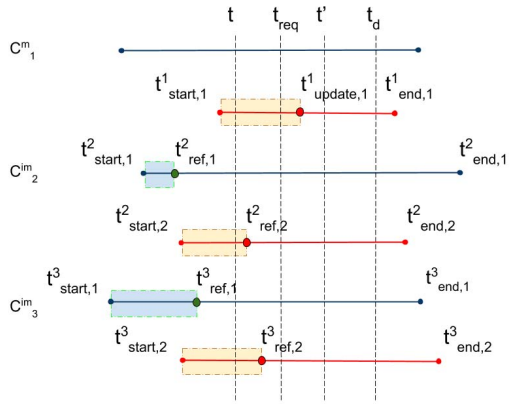

Fig. 4. Lifetime Overlap Consistency Level

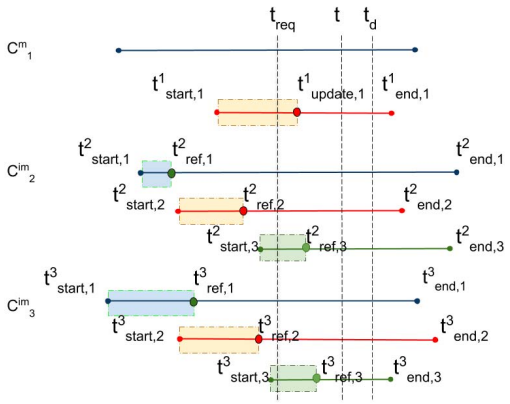

Fig. 5. Freshness Overlap Consistency Level

We propose two levels of consistency assuming the set of relevant attributes includes both mutable and immutable attributes as the most general case. Following [5], Table I shows required symbols with a brief explanation of each.

1) Lifetime Overlap Level: This level of consistency guarantees that all relevant credentials would be overlapping in their lifetimes. It also provides the decision point with the freshest value of relevant mutable attributes at the decision time. However immutable attributes freshness cannot be assured as it may not be possible to refresh them after the request time. So, immutable attribute values might be outdated but correct in the past.

Decision point would rely on values of latest available refresh results for immutable attribute credentials whenever refresh after request is unfeasible. Yet, refreshing mutable attribute credentials after request time is indispensable, in that their values could have been altered since last refresh as the usage side effect.

The decision point in Figure 4 relies on three attributes, the first of which is mutable and two others are immutable. As depicted the mutable attribute has been refreshed after the access request, however for mutable attributes the latest refresh results, which have been acquired before the request time, have been used. Formal specification of this level is as follows.

Specification. Every immutable attribute has to be refreshed at least once before the decision time and found to be fresh. Every mutable attribute has to be refreshed at least once after the request time and before the decision time and found fresh based on the latest refresh results.

$$
\begin{aligned}
&LifetimeOverlap(V_{DP}^{P,t_d}) \iff \\
&(\forall c_i^{im} \in V_{DP}^{P,t_d})(\exists t \;\; t \leq t_d) \\
&[(\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t)} \leq t^i_{ref,kmax(t)} < \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t)}) \\
&\wedge Fresh(c_i, t^i_{ref,kmax(t)})] \\
&\wedge (\forall c_i^{m} \in V_{DP}^{P,t_d})(\exists t' \;\; t_{req} \leq t' \leq t_d) \\
&[(\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t')} \leq t_{req} \leq t^i_{update,kmax(t')} \\
&< \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t')}) \wedge Fresh(c_i^m, t^i_{update,kmax(t_d)})] \\
&\wedge \max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t_d)} < t_d < \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t_d)}
\end{aligned}
\tag{1}
$$

Property 1. All relevant attributes lifetime intervals would overlap.

Proof. Based on Equation 1, all credentials would be refreshed when other credentials are in their lifetimes. Moreover, decision time lies in the last known lifetime interval for all relevant credentials at the decision time, as stated in the last part of Equation 1. This conveys at least one point (t_d) lies in intersection of all credentials lifetimes. So, all lifetimes would overlap in $[\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t_d)}, \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t_d)}]$. Although there is no guarantee for simultaneous freshness of all relevant credentials, Figure 4 depicts a lucky situation in which all credentials are fresh during $[t^1_{start,1}, t^2_{ref,2}]$. But even in this case there is no simultaneous freshness of all attributes after the request time.

2) Freshness Overlap: This level of consistency guarantees that all relevant credentials would be fresh simultaneously after the access request turned into the system. It requires all relevant credentials, including both mutable and immutable, to be refreshed after request time.

As depicted in Figure 5, three attributes have been considered to be relevant, first of which is mutable and two others are immutable. All three relevant attributes have been refreshed after the request time which supplies decision point

with the freshest values of each relevant credential while all freshness intervals are guaranteed to overlap after the request time. Following is the formal specification of this level.

Specification. Every credential has to be refreshed after request time and before the decision time. It is required for all relevant credentials to be started at or before the request time. So, simultaneous freshness of all relevant attributes is guaranteed. If some credentials start time fall after the request time, both mutable and immutable attributes would be assured to be fresh at some time interval after the request time but simultaneousness of freshness intervals is not assured. So, we restrict the start times to fall at/before request time.

$$
\begin{aligned}
&FreshnessOverlap(V_{DP}^{P,t_d}) \iff (\exists t \;\; t_{req} < t \leq t_d) \\
&[(\forall c_i^{im} \in V_{DP}^{P,t_d})(t^i_{start,kmax(t)} \leq t_{req} < t^i_{ref,kmax(t)}) \\
&\wedge (\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t)} \leq t^i_{ref,kmax(t)} \\
&< \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t)}) \wedge Fresh(c_i^{im}, t^i_{ref,kmax(t)})] \\
&[(\forall c_i^{m} \in V_{DP}^{P,t_d})(t^i_{start,kmax(t)} \leq t_{req} < t^i_{update,kmax(t)}) \\
&\wedge (\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t)} \leq t^i_{update,kmax(t)} \\
&< \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t)}) \wedge Fresh(c_i^{m}, t^i_{update,kmax(t_d)})] \\
&\wedge \max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t_d)} < t_d < \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{end,kmax(t_d)}
\end{aligned}
\tag{2}
$$

Property 1. All relevant credentials would be simultaneously fresh during a time interval before the decision time which includes the request time.

Proof. All credentials have to be refreshed after request time and before decision time. Also latest start time of all credentials should happen before/at the request time. So all credentials would be simultaneously fresh during $[\max_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{start,kmax(t)}, \min_{\forall c_i \in V_{DP}^{P,t_d}} t^i_{ref,kmax(t)}]$ time interval which includes the request time.

Property 2. Every view at Freshness Overlap level would be at Lifetime Overlap level as well.

Proof. This property is obvious, since freshness interval for every credential is a subinterval of its lifetime interval. Therefore, when there is freshness overlap, lifetime overlap is self-evident.

Property 3. Not every view at Lifetime Overlap level is at Freshness Overlap level as well.

Proof. As presented before, at the Lifetime Overlap level it is possible to have some immutable attributes with refresh time even before the request time. On the contrary, to be at Freshness Overlap level, every immutable credential requires to be updated at least once after the request time. Thus although the lifetimes would overlap based on the last condition stated in Eq. 1, freshness overlap could not be guaranteed.

Tip. *It is significant to note that enforcing the start time of refreshed attributes to lie before the request time ($t^i_{start,kmax(t)} \leq t_{req} < t^i_{ref,kmax(t)}$) is the key requirement in this level. Otherwise it is possible to have non-overlapping freshness intervals although all credentials have been found fresh after the request time. For example, as seen in Figure 6, latest start time of c_2^{im} has fallen after request time ($t^2_{start,3} > t_{req}$) and its freshness does not overlap with freshness interval of other two credentials.*

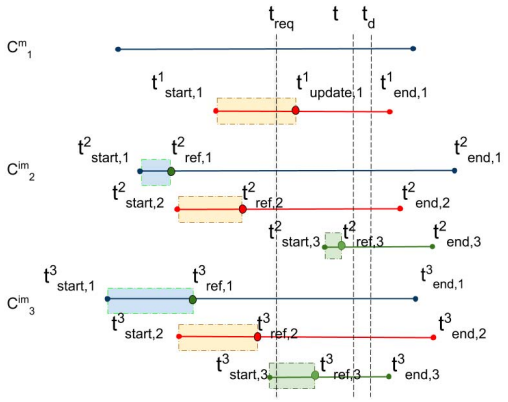

Fig. 6. Start Time Has Fallen After Request Time

VI. Conclusion

Safety and consistency problem in the context of mutable attributes in distributed ABAC environments has been studied in this paper for the first time to the best of our knowledge. To tackle concurrent usage of mutable attributes in a distributed environment we propose two categories of practical scenarios, from which we justified the distributed approach to be a better match with modern system environments and requirements. We further discuss two types of service-based and user-based subcategories. Both subcategories of distributed approach are amenable to quota-based approach. Formal specification of use cases have been proposed relying on nomenclature of UCON paper.

We assert that revocation check is inappropriate given mutable attributes values would be changing frequently and new values of them have to be sought and utilized in decision making process. Therefore, refresh should be used to pull recent values of attributes from attribute authorities. We proposed two levels of consistency well adapted to specifications in distributed ABAC environments.

Acknowledgement

This work is partially supported by NSF CREST Grant HRD-1736209 and DoD ARL Grant W911NF-15-1-0518.

References

[1] S. Das, B. Mitra, V. Atluri, J. Vaidya, and S. Sural, "Policy engineering in RBAC and ABAC," in *Database to Cyber Security*, 2018.

[2] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *DBSec XXVI*, 2012.

[3] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *IEEE Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[4] M. Shakarami and R. Sandhu, "Safety and consistency of subject attributes for attribute-based pre-authorization systems," in *NCS*. Springer, 2019.
[5] ——, "Refresh instead of revoke enhances safety and availability: A formal analysis," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2019, pp. 301–313.
[6] J. Park and R. Sandhu, "The UCON$_{ABC}$ usage control model," in *ACM TISSEC*, 2004.
[7] A. J. Lee and M. Winslett, "Safety and consistency in policy-based authorization systems," in *CCS*. ACM, 2006.
[8] ——, "Enforcing safety and consistency constraints in policy-based authorization systems," in *TISSEC*. ACM, 2008.
[9] A. J. Lee, K. Minami, and M. Winslett, "Lightweight consistency enforcement schemes for distributed proofs with hidden subtrees," in *ACM SACMAT*, 2007.
[10] R. Krishnan, J. Niu, R. Sandhu, and W. H. Winsborough, "Stale-safe security properties for group-based secure information sharing," in *ACM FMSE*, 2008.
[11] R. Krishnan and R. Sandhu, "Authorization policy specification and enforcement for group-centric secure information sharing," in *ICISS*. Springer, 2011.
[12] J. Park, X. Zhang, and R. Sandhu, "Attribute mutability in usage control," in *Research Directions in Data and Applications Security XVIII*. Springer, 2004, pp. 15–29.
[13] P. A. Bernstein, P. A. Bernstein, and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys (CSUR)*, vol. 13, no. 2, pp. 185–221, 1981.
[14] X. Yu, Y. Xia, A. Pavlo, D. Sanchez, L. Rudolph, and S. Devadas, "Sundial: harmonizing concurrency control and caching in a distributed OLTP database management system," *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1289–1302, 2018.
[15] R. Harding, D. Van Aken, A. Pavlo, and M. Stonebraker, "An evaluation of distributed concurrency control," *Proceedings of the VLDB Endowment*, vol. 10, no. 5, pp. 553–564, 2017.
[16] B. C. ord Neuman, "Scale in distributed systems," *ISI/USC*, 1994.
[17] Q. Ni, E. Bertino, and J. Lobo, "Risk-based access control systems built on fuzzy inferences," in *ASIACCS*. ACM, 2010, pp. 250–260.
[18] Q. Wang and H. Jin, "Quantified risk-adaptive access control for patient privacy protection in health information systems," in *ASIACCS*. ACM, 2011, pp. 406–410.
[19] J. Camenisch and E. Van Herreweghen, "Design and implementation of the Idemix anonymous credential system," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 21–30.
[20] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service," *Computer*, vol. 36, no. 10, pp. 38–44, 2003.