Relying on Trust to Balance Protection and Performance in Cryptographic Access Control

Simone Brunello
Stefano Berlato
Roberto Carbone
{sbrunello,sberlato,carbone}@fbk.eu
Fondazione Bruno Kessler
Trento, Italy

Adam J. Lee
adamlee@pitt.edu
University of Pittsburgh
Pittsburgh, United States

Silvio Ranise
ranise@fbk.eu
University of Trento
Trento, Italy
Fondazione Bruno Kessler
Trento, Italy

Abstract

Cryptographic Access Control (CAC) allows organizations to control cloud-hosted data sharing among users while preventing external attackers, malicious insiders, and honest-but-curious cloud providers from accessing the data. However, CAC entails an overhead often impractical for real-world scenarios due to the many cryptographic computations involved. Hence, we put forth a hybrid Access Control (AC) scheme — combining CAC and (traditional) centralized AC — that considers trust assumptions (e.g., on users) and data protection requirements of the underlying scenario on a case-by-case basis to reduce the number of cryptographic computations to execute in CAC. Besides, we design a consistency check to ensure the correctness and safety properties of the enforcement of the hybrid AC scheme, provide a proof-of-concept implementation in Prolog, and conduct a preliminary experimental evaluation.

CCS Concepts

• **Security and privacy → Access control**; *Cryptography*.

Keywords

Cryptographic Access Control; Performance; Optimization

ACM Reference Format:
Simone Brunello, Stefano Berlato, Roberto Carbone, Adam J. Lee, and Silvio Ranise. 2025. Relying on Trust to Balance Protection and Performance in Cryptographic Access Control. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies (SACMAT '25), July 8–10, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3734436.3734452

1 Introduction

Organizations need to protect and prevent unauthorized access to their cloud-hosted data from many threats (e.g., external attackers, malicious insiders); even cloud providers are generally deemed honest-but-curious [16], i.e., they perform operations assigned by their paying customers (e.g., store and ensure data integrity — loss of revenue is an incentive to perform to contract) while accessing customers' data for purposes such as advertising and AI training. The literature offers a wide range of proposals to provide confidentiality and integrity of cloud-hosted data, including approaches based on cryptography, steganography, and confidential computing. One such proposal is CAC, that is, enforcing AC policies by encrypting data client-side and distributing decrypting keys to authorized users only — key distribution (which amounts at distributing permissions) is typically carried out by trusted administrators.

Problem. Although cryptographically guaranteeing data confidentiality and integrity while enforcing AC policies, CAC requires many cryptographic computations, entailing a significant overhead which limits its applicability to real-world scenarios [16]. For instance, when a user loses access to a piece of data in CAC (e.g., due to a policy change), all corresponding decrypting keys need to be rotated (and re-distributed) and the data re-encrypted — lest the user (if untrusted) may have cached such decrypting keys and still access the data without authorization (possibly even colluding with cloud providers). Unfortunately, key rotation, re-distribution, and data re-encryption are expensive cryptographic computations and, even worse, are always executed [5]. In fact, CAC generally considers a fixed security model in which all data is deemed sensitive and all users are untrusted. However, fixed security models hardly suit multifaceted real-world scenarios where some data may not be sensitive or may be already protected with means other than CAC. Moreover, data re-encryption may need to be carried out as soon as possible ("eager re-encryption") or deferred to periods of low system activity, e.g., at night ("lazy re-encryption") [16]. Finally, only some users may be deemed untrusted or only under certain circumstances (e.g., after having been fired, not after a promotion).

Solution and Contributions. As a solution to the overhead and fixed security model usually assumed in CAC, we propose a hybrid AC scheme combining CAC and (traditional) centralized AC: our hybrid AC scheme is capable of considering tunable security models and accordingly reduce the number of cryptographic computations in CAC when enforcing AC policies — specifically, Role-Based Access Control (RBAC) — while preserving maximum data protection (Section 5). Although the concept underlying our solution is ideally applicable to several AC models, such as Attribute-Based Access Control (ABAC), we choose to focus on RBAC preliminarily due to its simplicity. Besides specifying the AC policy in our hybrid AC scheme (Section 5.1), administrators can tune the security model for their scenario by specifying predicates — i.e., facts about users, roles, and data — representing trust assumptions or data protection

SACMAT '25, Stony Brook, NY, USA

ACM ISBN 979-8-4007-1503-7/2025/07
https://doi.org/10.1145/3734436.3734452

requirements (Section 5.2). Our hybrid AC scheme then reasons on the given security model (i.e., the predicates) relying on common concepts in AC; in particular, the concept of queries allows administrators to define *how* predicates are used and combined (Section 5.3). Consequently, our hybrid AC scheme identifies the optimal balance between security and performance by automatically dividing the enforcement of AC policies into CAC and centralized AC (Section 5.4); even when CAC is required, our hybrid AC scheme can identify superfluous cryptographic computations (Section 5.5). For a concrete application, we consider a specific CAC scheme [16] (Section 4), enucleating its most expensive cryptographic computations (Section 4.1) and adapting it to our hybrid AC model (Section 4.2) — we remark that the CAC scheme is not a novel contribution of ours; the hybrid AC scheme is. To ensure that our hybrid AC scheme enforces AC policies as expected, we design a consistency check to verify the correctness and safety of the enforcement (Section 6). Finally, we provide a proof-of-concept implementation in Prolog and conduct a preliminary experimental evaluation (Section 7).

Structure. We provide background information and discuss related work in Sections 2 and 3, respectively. We analyze the specific CAC scheme [16] in Section 4 and describe our hybrid AC scheme in Section 5. Finally, we present the consistency check in Section 6, the proof-of-concept implementation and evaluation in Section 7, and conclude the paper with final remarks and future work in Section 8.

2 Related Work

We aim at *enhancing the performance* of CAC by integrating *trust* to reduce the number of cryptographic computations. Below, we investigate related work from these two perspectives, respectively.

Improving Performance of CAC. Crampton [12] represents AC policies as graphs and characterizes the performance of a CAC scheme as the distance — that is, minimum number of arcs — between nodes. Consequently, performance can be enhanced by modifying the graph to reduce such distance, hence acting on the structure of the AC policy. Similarly, Alderman et al. [1] represent AC policies as binary trees by using a Hierarchical Key Assignment Scheme (HKAS) for CAC; the derivation of keys then occurs in logarithmic time with respect to the number of nodes. In [14], Ferrara et al. improve on existing HKASs by proposing a Verifiable Hierarchical Key Assignment Scheme (VHKAS), enabling users to verify the trustworthiness of public information (e.g., preventing users from accepting incorrect keys). Finally, in [10] the authors put forth a comprehensive generic Key Assignment Scheme (KAS) that places an emphasis on performance as well as scalability and flexibility. Summarizing, the aforementioned works aim to optimize key distribution within a CAC scheme through KASs to improve performance, whereas we provide a (higher level) hybrid AC scheme aim at avoiding key distribution entirely — when made possible by the specified security model. Noticeably, our approach is not mutually exclusive with the aforementioned works, and in fact they can ideally be combined to further enhance performance (i.e., use a KAS-based CAC scheme in our hybrid AC scheme). The same is true for other works that focus on designing performant CAC schemes by, e.g., delegating data re-encryption to the cloud provider [22].

Including Trust in RBAC. Several researchers explored the intersection between trust and AC, especially in partially trusted (e.g., the cloud), decentralized, or dynamic environments in which users are not known in advance. For instance, in [13] the authors propose TrustBAC, an extended RBAC model in which users are not assigned to roles but to trust levels instead — determined through an evaluation of past behavior and other users' recommendation. A similar work but in the context of the Internet of Things (IoT) is that of Mahalle et al. [20], who propose an AC model named FTBAC including, within identity management, the notion of trust levels — determined from linguistic information of IoT devices. Analogously, Gwak et al. [17] propose TARAS, a trust-aware RBAC model allowing for establishing trust levels — determined by adopting the concept of I-sharing from psychology — between users and IoT devices without prior interaction. Finally, in [21] the authors propose a blockchain-based trust and reputation system for AC in IoT, in which trust levels are determined based on past behavior and recommendations (similarly to [13]). A diametrically opposite approach is to evaluate the *lack of* trustworthiness of users to identify malicious behavior and internal threats. For instance, Babu et al. [3] propose a support vector machine (SVM)-based approach to predict whether users' requests are legitimate or not, while Baracaldo et al. [4] extend RBAC with a risk assessment to evaluate (the risk level inherent to) users' behaviors. Summarizing, the aforementioned works provide different methods for evaluating trust (of, e.g., users and users' requests), whereas we use trust to divide the enforcement of AC policies into CAC and centralized AC. Again, our approach is not mutually exclusive with the aforementioned works, and in fact they can ideally be combined (e.g., instead of assuming that trust is simply determined by the administrator as we currently do, we could incorporate one of such methods in our hybrid AC scheme).

3 Background

Below, we present background information on AC and RBAC (Section 3.1) and provide an overview of CAC (Section 3.2). We collect all symbols used here and in the rest of the paper in Table 1.

3.1 Access Control

AC is fundamental in any cyber-physical application [2] and is typically divided into policy, model, and enforcement mechanism [23]. As described in [19], a model can be seen as a state transition system $S = \langle \Gamma, Q, \vdash, \Psi \rangle$ where Γ is a set of states, Q is a set of queries, Ψ is a set of state-change rules, and $\vdash : \Gamma \times Q \longrightarrow \{true, false\}$ is the entailment relation evaluating whether a query $q \in Q$ is true in a given state $\gamma \in \Gamma$. A state-change rule $\psi \in \Psi$ usually modifies the current state γ, i.e., $\gamma \mapsto_{\psi} \gamma'$ — that is, ψ transforms γ into γ' where γ' is usually (but not necessarily) different than γ. A model is also called AC scheme [19] — hence, the two terms are equivalent.

Role-Based Access Control. RBAC is a popular model in organizations due to its simple implementation and intuitive role assignment process [9]. A policy in RBAC (i.e., an RBAC policy) comprises three kinds of elements: users, roles, and resources (i.e., containers for data). Formally, in the core RBAC model considered by the National Institute of Standards and Technology (NIST) [15], a state $\gamma \in \Gamma$ is described as a tuple $\langle U, R, F, UR, PA \rangle$, where U is the set of users, R is the set of roles, F is the set of resources, $UR \subseteq U \times R$ is the set of user-role assignments and $PA \subseteq R \times PR$ is the set of role-permission assignments, being $PR \subseteq F \times OP$ a derivative set of F combined with a fixed set of operations OP (e.g., read, write). Both OP and PR are not part of the state, as OP remains constant over time and PR is derivative of F and OP. We note that role hierarchies can always be compiled away by adding suitable pairs to UR. The set of queries in the core RBAC model contains one query only, i.e., $Q = \{\boldsymbol{canDo}(u, r, op)\}$; intuitively, the query $q = \boldsymbol{canDo}(u, r, op)$ denotes whether the user $u \in U$ can use the permission $\langle f, op \rangle \in PR$, where $f \in F$ and $op \in OP$. We report the set of state-change rules Ψ for the core RBAC model and the resulting states in Table 2. For the sake of simplicity, we consider the entailment relation $\gamma \vdash (q)$ (or simply "entailment") as a special state-change rule that does not necessarily transform the state $\gamma \in \Gamma$ to which it is applied — i.e., $\gamma \mapsto_{\gamma \vdash (q)} \gamma$ — but has the side effect of evaluating the query $q \in Q$. In other words, the entailment corresponds to users' requests, while the other state-change rules correspond to administrative requests.

Table 1: Symbols

	Symbol	Description	Symbol	Description
Misc	$*$	a generic wildcard value	$-$	an empty or unused value
	$\perp$	the error symbol	$\top$	the success symbol
AC	S	a state transition system	Γ	the set of states
	γ	a state $\gamma \in \Gamma$	Q	the set of queries
	q	a query $q \in Q$	$\vdash$	the entailment function
	Ψ	the set of state-change rules	ψ	a state-change rule $\psi \in \Psi$
Core RBAC and Hybrid AC Scheme	U	the set of users	u	a user $u \in U$
	R	the set of roles	r	a role $r \in R$
	F	the set of resources	f	a resource $f \in F$
	P	the set of predicates	p	a predicate $p \in P$
	fc	the content of f (when encrypted, we use fc^{enc})		
	UR	the set of user-role assignments $UR \subseteq U \times R$		
	OP	the set of all possible operations ($OP = \{read, write\}$)		
	op	an operation $op \in OP$		
	ops	a subset of the set of operations $ops \subseteq OP$		
	PR	the set of all possible permissions $PR = OP \times F$		
	PA	the set of permission-role assignments $PA \subseteq R \times PR$		
	e	an element (i.e., either a user, role, or resource)		
	EP	the set of predicate-element assignments $EP \subseteq P \times (U \cup R \cup F)$		
	adm	the (username of the) administrator		
CAC	$\mathsf{k}^{\mathsf{enc}}$	a public encryption key	$\mathsf{k}^{\mathsf{dec}}$	a private decryption key
	$\mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}}(\cdot)$	encryption of $\cdot$ with $\mathsf{k}^{\mathsf{enc}}$	$\mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}}(\cdot)$	decryption of $\cdot$ with $\mathsf{k}^{\mathsf{dec}}$
	$\mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}}(\cdot)$	encryption of $\cdot$ with $\mathsf{k}^{\mathsf{sym}}$	$\mathsf{Dec}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}}(\cdot)$	decryption of $\cdot$ with $\mathsf{k}^{\mathsf{sym}}$
	$\mathsf{Gen}^{\mathsf{Pub}}$	generation of a pair of asymmetric public-private keys		
	$\mathsf{Gen}^{\mathsf{Sig}}$	generation of a symmetric key		
	$\mathsf{Gen}^{\mathsf{Pse}}$	generation of a pseudonym		
	p_e	the pseudonym of the element e		
	$\mathsf{k}^{\mathsf{sig}}$	a private signature creation key		
	$\mathsf{k}^{\mathsf{ver}}$	a public signature verification key		
	$\mathsf{k}^{\mathsf{sym}}$	a secret symmetric key		
	$\mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}}$	a digital signature created with $\mathsf{k}^{\mathsf{sig}}$		
	$\widehat{v_e}$	a version number relative to the element e		
	v_e	the latest version number relative to the element e		
	fc^{enc}	the encrypted content of f (when unencrypted, we use fc)		

Table 2: The set Ψ for the core RBAC model [15]*

State-Change Rule ψ_i	Resulting State γ^{i+1}
addUser(u)	$\langle U \cup \{u\}, R, F, UR, PA \rangle$
deleteUser(u)	$\langle U \setminus \{u\}, R, F, UR, PA \rangle$
addRole(r)	$\langle U, R \cup \{r\}, F, UR, PA \rangle$
deleteRole(r)	$\langle U, R \setminus \{r\}, F, UR, PA \rangle$
addResource(f)	$\langle U, R, F \cup \{f\}, UR, PA \rangle$
deleteResource(f)	$\langle U, R, F \setminus \{f\}, UR, PA \rangle$
assignUserToRole(u, r)	$\langle U, R, F, UR \cup \{(u, r)\}, PA \rangle$
revokeUserFromRole(u, r)	$\langle U, R, F, UR \setminus \{(u, r)\}, PA \rangle$
assignPermissionToRole(r, $\langle f, op \rangle$)	$\langle U, R, F, UR, PA \cup \{(r, \langle f, op \rangle)\} \rangle$
revokePermissionFromRole(r, $\langle f, op \rangle$)	$\langle U, R, F, UR, PA \setminus \{(r, \langle f, op \rangle)\} \rangle$
$\gamma \vdash (\boldsymbol{canDo}(u, f, op)) : true \Leftrightarrow \exists r \in R \mid (u, r) \in UR \wedge (r, \langle f, op \rangle) \in PA$	

*Before deleting a user, role, or resource, an administrator should first delete all relative assignments

3.2 Cryptographic Access Control

An enforcement mechanism may either rely on a trusted central agent to enforce AC policies by mediating requests to resources — hence, centralized AC — or employ cryptography for distributing the enforcement — hence, CAC [6]. In other words, CAC allows for enforcing policies in a distributed fashion while ensuring confidentiality and integrity of resources even if in presence of honest-but-curious agents (e.g., cloud providers). Typically, CAC employs both symmetric and asymmetric cryptography to encrypt data contained in resources and distribute the corresponding decrypting keys to authorized users only, respectively. The use of both symmetric and asymmetric cryptography is called *hybrid cryptography* [16].

Role-based Cryptographic Access Control. A CAC scheme compatible with the core RBAC model — like the CAC scheme proposed in [16] — works as follows: each user u and role r is equipped with a pair of public-private keys $(\mathsf{k}^{\mathsf{enc}}_u, \mathsf{k}^{\mathsf{dec}}_u)$ and $(\mathsf{k}^{\mathsf{enc}}_r, \mathsf{k}^{\mathsf{dec}}_r)$, respectively. To assign a user u to a role r, r's decryption key $\mathsf{k}^{\mathsf{dec}}_r$ is encrypted with u's encryption key $\mathsf{k}^{\mathsf{enc}}_u$, resulting in $\{\mathsf{k}^{\mathsf{dec}}_r\}_{\mathsf{k}^{\mathsf{enc}}_u}$. To give read permission to a role r over a resource f, f's symmetric key $\mathsf{k}^{\mathsf{sym}}_f$ is encrypted with r's encryption key $\mathsf{k}^{\mathsf{enc}}_r$, resulting in $\{\mathsf{k}^{\mathsf{sym}}_f\}_{\mathsf{k}^{\mathsf{enc}}_r}$. Hence, read and write operations over encrypted data require users to first obtain $\mathsf{k}^{\mathsf{dec}}_r$ — by decrypting $\{\mathsf{k}^{\mathsf{dec}}_r\}_{\mathsf{k}^{\mathsf{enc}}_u}$ with $\mathsf{k}^{\mathsf{dec}}_u$ — and then $\mathsf{k}^{\mathsf{sym}}_f$ — by decrypting $\{\mathsf{k}^{\mathsf{sym}}_f\}_{\mathsf{k}^{\mathsf{enc}}_r}$ with $\mathsf{k}^{\mathsf{dec}}_r$. As a result, an RBAC policy in such a CAC scheme is encoded by the encrypted keys (e.g., $\{\mathsf{k}^{\mathsf{dec}}_r\}_{\mathsf{k}^{\mathsf{enc}}_u}$ and $\{\mathsf{k}^{\mathsf{sym}}_f\}_{\mathsf{k}^{\mathsf{enc}}_r}$, which would be added to decorated versions of UR and PA, respectively). Encrypted keys in CAC are often enriched with auxiliary information called metadata. For instance, roles' and resources' keys are sometimes associated with incremental version numbers to handle revocations — the topic of revocation is discussed more in detail in Section 4.1. Typically, the encrypted keys and the metadata are digitally signed by the administrator to guarantee their integrity.

4 Reviewing the Cryptographic Access Control Scheme

In order to mitigate the overhead of CAC, we first need to clearly identify where and under what conditions such an overhead occurs. To this end, below we consider a specific CAC scheme [16], identifying the most computationally expensive procedures (Section 4.1) and proposing adjustments required to then being able to mitigate its overhead (Section 4.2). Using the notation in Section 3.1, we see the CAC scheme as a state transition system $S^C = \langle \Gamma^C, Q^C, \vdash^C, \Psi^C \rangle$ — hereafter, we use the superscript C (short for "Cryptographic") when symbols (see Table 1) refer to the CAC scheme.

4.1 Computationally Expensive Procedures

As a specific instance of CAC scheme for enforcing RBAC policies, we consider the work initially proposed in [16] — and further developed in [5, 7, 8] — whose basic functioning was explained in Section 3.2. In particular, 5 of the state-change rules of the last version of the CAC scheme [5] — which are $\texttt{deleteUser}^{\mathsf{C}}(u)$, $\texttt{deleteRole}^{\mathsf{C}}(r)$, $\texttt{deleteResource}^{\mathsf{C}}(f)$, $\texttt{revokeUserFromRole}^{\mathsf{C}}(u, r)$, and $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle op, f\rangle)$ — entails the most overhead. Indeed, revoking privileges from users and roles often requires executing one or more of the following 3 procedures:

- *role key rotation*: if a user u is revoked from a role r, the administrator has to rotate r's keys $(\mathsf{k}_r^{\mathsf{enc}}, \mathsf{k}_r^{\mathsf{dec}})$; in fact, u could have previously cached $(\mathsf{k}_r^{\mathsf{enc}}, \mathsf{k}_r^{\mathsf{dec}})$. Hence, to prevent u from still accessing resources through the cached keys $(\mathsf{k}_r^{\mathsf{enc}}, \mathsf{k}_r^{\mathsf{dec}})$, the administrator has to create new keys for r and distribute them to all users assigned to r — except u;
- *resource key rotation*: similarly, if a permission $\langle f, op\rangle$ is revoked from a role r — and $\langle f, op\rangle$ was the only permission of r over f — the administrator has to rotate f's key $\mathsf{k}_f^{\mathsf{sym}}$ and distribute it to all roles with a permission over f — except r;
- *resource re-encryption*: whenever the rotation of a resource f's key $\mathsf{k}_f^{\mathsf{sym}}$ occurs, fc may need to be (decrypted with $\mathsf{k}_f^{\mathsf{sym}}$ and then re-)encrypted with the new key of f. The practice of immediately re-encrypting fc after the rotation of $\mathsf{k}_f^{\mathsf{sym}}$ is called *eager re-encryption*. The alternative, *lazy re-encryption* [16], expects the next user that writes new data to the resource to re-encrypt also fc with the new key.

The CAC scheme in [5] assumes a fixed security model: the cloud provider is honest-but-curious, all users are untrusted, and all resources need encryption. Unfortunately, this assumption requires the administrator to always perform role key rotation, resource key rotation, and resource (either eager or lazy) re-encryption.

4.2 Adjustments

The 3 procedures described in Section 4.1 (i.e., role key rotation, resource key rotation, and resource re-encryption) ensure that the CAC scheme in [5] enforces RBAC policies as expected — that is, providing the correctness and safety properties even when users cache roles' and resources' keys. As defined in [18], correctness means that the CAC scheme evaluates queries (e.g., whether a user can use a permission) exactly as the core RBAC model would, and safety means that the CAC scheme does not grant or revoke unnecessary permissions during state transitions caused by the execution of a single state-change rule. Although ensuring correctness and safety, these procedures may also make the CAC scheme unusable due to their overhead. However, as we hint in Section 1, role key rotation, resource key rotation, and resource re-encryption may at times be superfluous. In particular, the execution of such procedures could sometimes be avoided by allowing the administrator to consider tunable security models while still providing correctness and safety. Therefore, we make some adjustments — which we discuss below — to the CAC scheme in [5] to allow for avoiding the execution of these procedures. We report the resulting pseudocode describing (the state-change rules of) the adjusted CAC scheme in Table 9. Unfortunately, space limitations prevent a comprehensive explanation of each aspect of the CAC scheme. However, the CAC scheme itself is not a novel contribution or focus of our work (the adjustments are), hence we refer the reader to [5] for more details.

Partitioning by Status. As introduced in Section 3.2, a state $\gamma^C \in \Gamma^C$ of the CAC scheme is a tuple of sets $\gamma^C = \langle U^C, R^C, F^C, UR^C, PA^C\rangle$, where each set is decorated with (encrypted) keys and further metadata (e.g., version numbers, digital signatures). Below, we report a simplified version of these decorated sets:[1]

$$\langle u, \mathsf{k}_u^{\mathsf{enc}}\rangle \in U^C; \qquad \langle r, \mathsf{k}_r^{\mathsf{enc}}\rangle \in R^C; \qquad \langle f, \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}_f^{\mathsf{sym}}}(fc)\rangle \in F^C;$$

$$\langle u, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}_u^{\mathsf{enc}}}(\mathsf{k}_r^{\mathsf{dec}})\rangle \in UR^C; \qquad \langle r, f, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}_r^{\mathsf{enc}}}(\mathsf{k}_f^{\mathsf{sym}})\rangle \in PA^C;$$

We adjust the aforementioned sets by defining the concept of "status" to partition them. This partitioning is necessary to track information such as revoked privileges, hence possibly cached roles' and resources' keys. In detail, we define 4 statuses: incomplete `inc`, operational `ope`, revoked `hide` (for tracking keys that may have been cached), and deleted `del` (which can be considered deleted effectively). For instance, when an existing user-role assignment $\langle u, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}_u^{\mathsf{enc}}}(\mathsf{k}_r^{\mathsf{dec}})\rangle \in R^C_{\mathtt{ope}}$ is revoked, we move it from $R^C_{\mathtt{ope}}$ to $R^C_{\mathtt{hide}}$ — as to keep track that the assignment was revoked but u may have cached $\mathsf{k}_r^{\mathsf{dec}}$. Then, when $\mathsf{k}_r^{\mathsf{dec}}$ is also rotated, we move the assignment from $R^C_{\mathtt{hide}}$ to $U^C_{\mathtt{del}}$ — as to keep track that $\mathsf{k}_r^{\mathsf{dec}}$, even if it was cached, is now useless to u. Consequently, each set in the state of the CAC policy is partitioned into 4 distinct subsets; for instance, U^C is partitioned into $U^C_{\mathtt{inc}}$, $U^C_{\mathtt{ope}}$, $U^C_{\mathtt{hide}}$, and $U^C_{\mathtt{del}}$.

Modifying State-Change Rules. We modify the state-change rules of the CAC scheme to make the aforementioned procedures self-contained. In other words, the 5 state-change rules $\texttt{deleteUser}^{\mathsf{C}}(u)$, $\texttt{deleteRole}^{\mathsf{C}}(r)$, $\texttt{deleteResource}^{\mathsf{C}}(f)$, $\texttt{revokeUserFromRole}^{\mathsf{C}}(u, r)$, and $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle op, f\rangle)$ do not longer automatically execute role key rotation, resource key rotation, and resource re-encryption. Instead, these procedures are executed only when required by the underlying security model (see Section 5).

Adding State-Change Rules. We add 4 state-change rules corresponding to the (now self-contained) aforementioned procedures: $\texttt{rotateRoleKeyUserRole}^{\mathsf{C}}(r)$, $\texttt{rotateRoleKeyPermissions}^{\mathsf{C}}(r)$ — which update UR^C and PA^C, respectively — implement role key rotation, $\texttt{rotateResourceKey}^{\mathsf{C}}(f, fc^{enc})$ implements resource key rotation, while $\texttt{eagerReEncryption}^{\mathsf{C}}(f, fc^{enc})$ implements (eager) resource re-encryption. We also add $\texttt{cleanup}^{\mathsf{C}}()$ for removing unnecessary information (e.g., resources' old keys not used anymore).

Adding a Set of Queries Q^C. We add a set of 10 queries Q^C (se Table 8) to take into account the actions that a user could take, possibly behaving maliciously; for instance, an untrusted user may use cached keys to access resources without authorization; Q^C is fundamental for the consistency check discussed in Section 6.

First, we add 7 queries akin to the ***canDo***(u, r, op) query of core RBAC (see Section 3.1). In particular, these queries allow for understanding what permissions a user can access, possibly using cached keys. The query ***canDo***$^C(u, op, f)$ determines whether the user u can access the permission $\langle f, op\rangle$ legitimately, the query ***canUserDoViaRole***$^C(u, r, op, f)$ determines whether the user u can access the permission $\langle f, op\rangle$ through the role r legitimately,

[1]"Simplified" as we omit 3 components secondary in our discussion: *pseudonyms* (strings used to hide the actual identifiers of users, roles, and resources), *version numbers* (integers used for differentiating between old and new information), and *digital signatures* (against tampering or accidental modifications); please refer to [5].

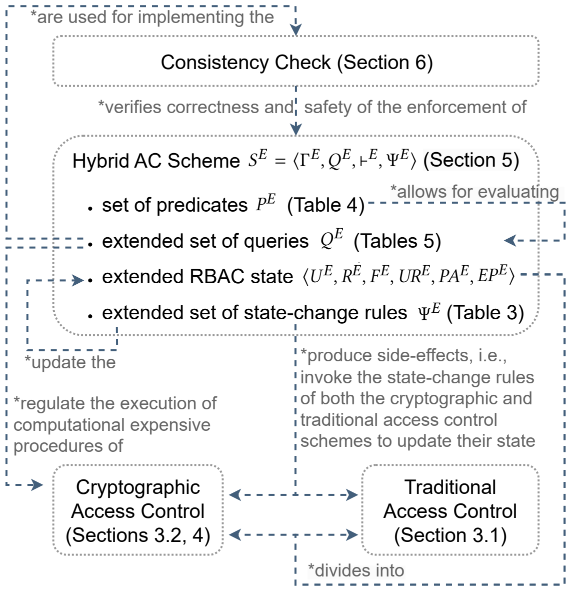

Figure 1: Conceptual map

and the query $\boldsymbol{canRoleDo}^C(r, op, f)$ determines whether the role r can access the permission $\langle f, op \rangle$ legitimately. Conversely, the query $\boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f)$ determines whether the user u can access the permission $\langle f, op \rangle$ on at least one version of f through the role r possibly using cached keys, the query $\boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$ determines whether the user u can access the permission $\langle f, op \rangle$ on the latest version of f through the role r possibly using cached keys, the query $\boldsymbol{canRoleDoCache}^C(r, op, f)$ determines whether the role r can access the permission $\langle f, op \rangle$ on at least one version of f possibly using cached keys, and the query $\boldsymbol{canRoleDoCacheLast}^C(r, op, f)$ determines whether the role r can access the permission $\langle f, op \rangle$ on the latest version of f possibly using cached keys.

Then, the add 2 queries to understand what roles a user can assume, possibly using cached keys: the query $\boldsymbol{canUserBe}^C(u, r)$ determines whether the user u can assume the role r, while the query $\boldsymbol{canUserBeCache}^C(u, r)$ determines whether the user u can assume the role r possibly using cached keys.

Finally, we add the query $\boldsymbol{isProtectedWithCAC}^C(f)$ which determines whether the resource f is protected with CAC.

Table 3: The Proposed Predicates P^E

Predicate	Description
`cloudNoEnforce`(f)	The administrator does not trust the cloud provider to protect the resource f with centralized AC
`untrusted`(u)	u may collude with the cloud provider to gain unauthorized access to resources when lazy re-encryption is used
`cac`(f)	f must be protected with CAC, e.g., as it contains sensitive data whose confidentiality is worth the overhead of CAC
`eager`(f)	The re-encryption of f should happen with eager re-encryption rather than with lazy re-encryption

5 The Hybrid Access Control Scheme

We now describe our hybrid AC scheme for enforcing RBAC policies combining CAC and centralized AC seamlessly. As done for the CAC scheme (S^C), we use the superscript E (short for "Extended", as our hybrid AC scheme extends core RBAC) when symbols refer to the hybrid AC scheme (hence, S^E), and the superscript T (short for "Traditional") when symbols refer to core RBAC (hence, S^T).

We report an overview of the hybrid AC scheme in the conceptual map in Figure 1. The hybrid AC scheme comprises a set of predicates P^E — used by the administrator to specify trust assumptions and data protection requirements — for expressing tunable security models. Predicates are then used as the basis for evaluating an extended set of queries Q^E (recall the definition of query in Section 3.1), allowing for regulating (i.e., reducing) the execution of computationally expensive procedures and reduce the overhead of CAC (recall the discussion in Section 4.1). Accordingly, the extended RBAC state automatically divides into two (possibly nonidentical and smaller) sub-states for the core RBAC model S^T and the CAC scheme S^C. This division already limits the use of CAC by relying solely on centralized AC for protecting that subset of the resources which — according to the security model (that is, the predicates) specified by the administrator — does not require CAC. The administrator can update the extended RBAC state by invoking the extended set of state-change rules Ψ^E, which also produce what we call "side effects": simply, side effects consist in invoking (some of) the state-change rules in Ψ^T and in Ψ^C of the core RBAC model S^T and the CAC scheme S^C. In other words, updates to the extended RBAC state are automatically reported into the two sub-states. Finally, queries are also used to implement a consistency check that verifies the correctness and safety of the enforcement.

5.1 Extended Role-Based Access Control State Γ^E

A state $\gamma^E \in \Gamma^E$ in the hybrid AC scheme is described as a tuple $\langle U^E, R^E, F^E, UR^E, PA^E, EP^E \rangle$, where $EP^E \subseteq P^E \times (U^E \cup R^E \cup F^E)$ is the set of predicate-element assignments. Similarly to OP, P^E is not part of the state as it remains constant over time.

5.2 Set of Predicates P^E

Predicates are used to specify trust assumptions and data protection requirements, but in general they may express any characteristic or fact regarding an element which is relevant to AC purposes. As an example, consider an organization where the administrator issues smart cards containing cryptographic material to employees; smart cards allow for regulating access to offices and laboratories containing sensitive resources and equipped with smart locks embedding the AC policy. In this context, an employee u whose smart card was stolen could be assigned a predicate `smartCardStolen`(u) — for readability, `predicate`$(\cdot)$ is equivalent to $($`predicate`$, \cdot) \in EP^E$. Intuitively, such a predicate may influence the process through which the administrator revokes access privileges. For instance, revoking privileges from an employee $u \in U^E$ s.t. `smartCardStolen`$(u) \in EP^E$ may require the administrator to update the policy of all smart locks — to prevent unauthorized accesses from the attacker who stole the smart card. Conversely, revoking privileges from an employee $u' \in U^E$ s.t. `smartCardStolen`$(u') \notin EP^E$ may require the administrator to simply get the issued smart card back from u' —

as there is no need to update the policy of smart locks. The administrator can define the predicates in P^E on a case-by-case basis according to the underlying scenario — for concreteness, we define some simple predicates in Table 3; to maintain a straightforward notation, we consider only unary predicates.

5.3 Extended Set of Queries Q^E

The extended set of queries Q^E in the hybrid AC scheme comprises the ***canDo***(u, r, op) query derived from core RBAC as well as 6 new queries for regulating the execution of computationally expensive procedures in CAC. We report each query in Q^E in Table 4. The evaluation of a query — that is, whether the query is *true* or *false* — depends on three aspects: (*i*) the predicates defined by the administrator (see Section 5.2), (*ii*) the predicate-element assignments EP^E (see Section 5.1), and (*iii*) how the administrator defines the entailment for each query. For instance, consider again the example of the organization issuing smart cards to employees. For simplicity, assume that a smart card contains the private keys of all roles to which the corresponding employee is assigned. Consequently, the query ***isRoleKeyRotationNeeded***$^E(u, r)$ may be evaluated at *true* when u's smart card is stolen — formally, ***isRoleKeyRotationNeeded***$^E(u, r) \Leftrightarrow$ `smartCardStolen`(u). The administrator can define the entailment for the queries according to the underlying scenario, deciding on a case-by-case basis whether to prioritize security or efficiency. For concreteness, we define in Table 4 the entailment for the new queries using the predicates listed in Table 3. Finally, when a query is evaluated at *true* it may trigger the execution of a computationally expensive procedure of CAC.

5.4 Extended Set of State-Change Rules Ψ^E

As said in Section 3.1, a state-change rule is defined as $\psi^E : \Gamma^E \to \Gamma^E$, being $\gamma^{E^i} \in \Gamma^E$ the initial state and $\gamma^{E^{i+1}} \in \Gamma^E$ the final state. The extended set of state-change rules Ψ^E in the hybrid AC scheme is defined starting from the set of state-change rules Ψ^T of core RBAC presented in Table 2 on top of which we propose the changes listed below; we report Ψ^E in Table 5 (we remark that the entailment $\gamma \vdash (q)$ is a state-change rule). First, we add 8 state-change rules:

- `init`$^{\mathsf{E}}()$: set $\gamma^E = \langle\{adm\}, \{adm\}, \{\}, \{(adm, adm)\}, \{\}, \{\}\rangle$;
- `assignPredicate`$^{\mathsf{E}}(p, e)$: add (p, e) to EP^E;
- `revokePredicate`$^{\mathsf{E}}(p, e)$: remove (p, e) from EP^E;
- `readResource`$^{\mathsf{E}}_u(f)$: read f, possibly decrypting fc^{enc};
- `writeResource`$^{\mathsf{E}}_u(f, fc)$: write f, possibly encrypting fc;
- `rotateResourceKey`$^{\mathsf{E}}(f)$ and `eagerReEncryption`$^{\mathsf{E}}(f)$: invoke resource key rotation and resource re-encryption;
- `consistencyCheck`$^{\mathsf{E}}()$: the consistency check (see Section 6).

Then, we also modify `deleteUser`$^{\mathsf{E}}(\cdot)$, `deleteRole`$^{\mathsf{E}}(\cdot)$, and `deleteResource`$^{\mathsf{E}}(\cdot)$ to remove all predicate-element assignments related to the deleted element. Also, we add the *preds* parameter to `addUser`$^{\mathsf{E}}(\cdot, preds)$, `addRole`$^{\mathsf{E}}(\cdot, preds)$, and `addResource`$^{\mathsf{E}}_u(f, fc, preds)$ — and the fc parameter to `addResource`$^{\mathsf{E}}_u(f, fc, preds)$ — to enable the assignments of predicates when creating new elements.

Side Effects. As mentioned in Section 5, the state-change rules in Ψ^E — besides updating the state γ^E — may have side effects, i.e., they may invoke some of the state-change rules in Ψ^T (see Table 2) and in Ψ^C (see Table 9). Side effects allow for automatically dividing γ^E into the two sub-states γ^T and γ^C. In other words, any update to the extended RBAC state is automatically reported into the two sub-states. For instance, adding a user to γ^E results in adding a user to both γ^T (see Section 3.1) and γ^C (see Section 4). To enhance clarity, in Table 5 we mark with "`T`" side effects related to the core RBAC model S^T and with "`C`" side effects related to the CAC scheme S^C — we mark with "`E`" those instructions related to the hybrid AC scheme S^E. In some cases, whether to invoke the state-change rules of the CAC scheme S^C ultimately depends on the new queries shown in Table 4. For instance, consider the `deleteRole`$^{\mathsf{E}}(r)$ state-change rule in Table 5 and assume that $(r, \langle op, f\rangle) \in PA$: the state-change rule `revokePermissionFromRole`$^{\mathsf{C}}(r, \langle\{op\}, f\rangle)$ is invoked if and only if ***isCacNeeded***$^E(f)$. Similarly, `rotateResourceKey`$^{\mathsf{C}}(f, fc^{enc})$ is invoked if and only if ***isResourceKeyRotationNeededOnRevP***$^E(r, op, f)$, and `eagerReEncryption`$^{\mathsf{C}}(f, fc^{enc})$ is invoked if and only if ***isEagerNeededOnRevP***$^E(r, op, f)$.

5.5 Concrete Example

We now provide a concrete example to show how the new queries (see Table 4) in S^E regulate the execution of computationally expensive procedures in S^C according to the security model (that is,

Table 4: The extended set of queries Q^E of the hybrid AC scheme

Query $q \in Q^E$	Description	Entailment	CAC Procedures Triggered
canDo$^E(u, op, f)$	Querying whether u can use the permission $\langle f, op\rangle$	***canDo***$^E(u, op, f) \Leftrightarrow (\exists r \in R^E \mid (u, r) \in UR^E \wedge (r, \langle f, op\rangle) \in PA^E)$	–
isCacNeeded$^E(f)$	Querying whether CAC must protect f	***isCacNeeded***$^E(f) \Leftrightarrow$ `cac`(f)	–
isRoleKeyRotationNeeded$^E(u, r)$	Querying whether r's keys must be rotated when u is revoked from r lest u may have cached r's keys and access resources without authorization	***isRoleKeyRotationNeeded***$^E(u, r) \Leftrightarrow$ `untrusted`(u)	If *true*, execute the role key rotation procedure for r when u is revoked r
isResourceKeyRotationNeeded-OnRevUR$^E(u, r, \langle op, f\rangle)$	Querying whether f's key must be rotated when u is revoked from r — and, as a consequence, u loses the permission $\langle f, op\rangle$ — lest u may have cached f's keys and access f without authorization	***isResourceKeyRotationNeededOnRevUR***$^E(u, r, \langle op, f\rangle) \Leftrightarrow$ `cac`$(f) \wedge$ `cloudNoEnforce`$(f) \wedge$ `untrusted`(u)	If *true*, execute the resource key rotation procedure for f when u is revoked from r
isResourceKeyRotationNeeded-OnRevP$^E(r, op, f)$	Querying whether f's key must be rotated when the permission $\langle f, op\rangle$ is revoked from r lest a user $u \in U^E$ s.t. $(u, r) \in UR^E$ may have cached r's keys and access f through r without authorization	***isResourceKeyRotationNeededOnRevP***$^E(r, op, f) \Leftrightarrow$ `cac`$(f) \wedge$ `cloudNoEnforce`$(f) \wedge \exists u \in U^E$ s.t. ***canDo***$^E(u, *, f) \wedge$ `untrusted`(u)	If *true*, execute the resource key rotation procedure for f when $\langle op, f\rangle$ is revoked from r
isEagerNeededOnRevUR$^E(u, r, \langle op, f\rangle)$	Querying whether f must be immediately re-encrypted when u is revoked from r — and, as a consequence, u loses the permission $\langle f, op\rangle$ — lest u may have cached f's key and access f without authorization	***isEagerNeededOnRevUR***$^E(u, r, \langle op, f\rangle) \Leftrightarrow$ `cac`$(f) \wedge$ `cloudNoEnforce`$(f) \wedge$ `eager`$(f) \wedge$ `untrusted`(u)	If *true*, execute the eager resource re-encryption procedure for f when u is revoked from r
isEagerNeededOnRevP$^E(r, op, f)$	Querying whether f must be immediately re-encrypted the permission $\langle f, op\rangle$ is revoked from r lest a user $u \in U^E$ s.t. $(u, r) \in UR^E$ may have cached r's keys and access f through r without authorization	***isEagerNeededOnRevP***$^E(r, op, f) \Leftrightarrow$ `cac`$(f) \wedge$ `cloudNoEnforce`$(f) \wedge$ `eager`$(f) \wedge \exists u \in U^E$ s.t. ***canDo***$^E(u, *, f) \wedge$ `untrusted`(u)	If *true*, execute the eager resource re-encryption procedure for f when $\langle op, f\rangle$ is revoked from r

Table 5: The pseudocode of the state-change rules Ψ^E of the hybrid AC scheme S^E

- $\texttt{init}^{\mathsf{E}}()$
  - E Set state to $\langle \{adm\}, \{adm\}, \{\}, \{(adm, adm)\}, \{\}, \{\}\rangle$
  - T Invoke $\texttt{addUser}^{\mathsf{T}}(adm)$ and $\texttt{addRole}^{\mathsf{T}}(adm)$
  - T Invoke $\texttt{assignUserToRole}^{\mathsf{T}}(adm, adm)$
  - C Invoke $\texttt{init}^{\mathsf{C}}()$
- $\texttt{addUser}^{\mathsf{E}}(u, preds = \{\})$
  - E Update state to $\langle U^E \cup \{u\}, R^E, F^E, UR^E, PA^E, EP^E \cup \{(p, u) \mid p \in preds\}\rangle$
  - T Invoke $\texttt{addUser}^{\mathsf{T}}(u)$ and $\texttt{addUser}^{\mathsf{C}}(u)$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{deleteUser}^{\mathsf{E}}(u)$
  - C For each $r \in R^E$ s.t. $(u, r) \in UR^E$:
    - ∗ Invoke $\texttt{revokeUserFromRole}^{\mathsf{C}}(u, r)$
    - ∗ If $\boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$:
      - · Invoke $\texttt{rotateRoleKeyUserRole}^{\mathsf{C}}(r)$
  - T Invoke $\texttt{deleteUserT}(u)$ and $\texttt{deleteUser}^{\mathsf{C}}(u)$
  - C For each $f \in F^E$ s.t. $\exists op \in OP \wedge \boldsymbol{canDo}^E(u, op, f) \wedge \boldsymbol{isCacNeeded}^E(f)$:
    - ∗ If $\exists r \in R^E, op \in OP$ s.t. $\boldsymbol{canUserDoViaRole}^C(u, r, op, f) \wedge \boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f\rangle)$:
      - · Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
    - ∗ If $\exists r \in R^E, op \in OP$ s.t. $\boldsymbol{canUserDoViaRole}^C(u, r, op, f) \wedge \boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f\rangle)$:
      - · Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - C For each $r \in R^E$ s.t. $(u, r) \in UR^E \wedge \boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$:
    - ∗ Invoke $\texttt{rotateRoleKeyPermissions}^{\mathsf{C}}(r)$
  - E Update state to $\langle U^E \setminus \{u\}, R^E, F^E, UR^E \setminus \{(u, r) \mid r \in R^E\}, PA^E, EP^E \setminus \{(p, u) \mid p \in P^E\}\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{addRole}^{\mathsf{E}}(r, preds = \{\})$
  - E Update state to $\langle U^E, R^E \cup \{r\}, F^E, UR^E \cup \{(adm, r)\}, PA^E, EP^E \cup \{(p, r) \mid p \in preds\}\rangle$
  - T Invoke $\texttt{addRole}^{\mathsf{T}}(r)$ and $\texttt{addRole}^{\mathsf{C}}(r)$
  - T Invoke $\texttt{assignUserToRole}^{\mathsf{T}}(adm, r)$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{deleteRole}^{\mathsf{E}}(r)$
  - C For each $f \in F^E$ s.t. $\exists ops \subseteq OP \wedge (r, \langle ops, f\rangle) \in PA^E \wedge \boldsymbol{isCacNeeded}^E(f)$:
    - ∗ Invoke $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f)$:
      - · Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isEagerNeededOnRevP}^E(r, op, f)$:
      - · Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - C For each $u \in U^E$ s.t. $(u, r) \in UR^E$:
    - ∗ Invoke $\texttt{revokeUserFromRole}^{\mathsf{C}}(u, r)$
  - T Invoke $\texttt{deleteRole}^{\mathsf{T}}(r)$
  - C Invoke $\texttt{deleteRole}^{\mathsf{C}}(r)$
  - E Update state to $\langle U^E, R^E \setminus \{r\}, F^E, UR^E \setminus \{(u, r) \mid u \in U^E\}, PA^E \setminus \{(r, \langle ops, f\rangle) \mid ops \subseteq OP, (f, *) \in F^E\}, EP^E \setminus \{(p, r) \mid p \in P^E\}\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{addResource}^{\mathsf{E}}_u(f, fc, preds = \{\})$
  - E Update state to $\langle U^E, R^E, F^E \cup \{f\}, UR^E, PA^E \cup \{(adm, \langle OP, f\rangle)\}, EP^E \cup \{(p, f) \mid p \in preds\}\rangle$
  - C If $\boldsymbol{isCacNeeded}^E(f)$:
    - ∗ Invoke $\texttt{addResource}^{\mathsf{C}}_u(f)$
    - ∗ Invoke $fc^{enc} \leftarrow \texttt{writeResource}^{\mathsf{C}}_u(f, fc)$
  - C Else:
    - ∗ Set $fc^{enc} \leftarrow fc$
  - T Invoke $\texttt{addResource}^{\mathsf{T}}(f)$
  - T invoke $\texttt{assignPermissionToRole}^{\mathsf{T}}(adm, \langle OP, f\rangle)$
  - E Store fc^{enc} in the cloud
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{deleteResource}^{\mathsf{E}}(f)$
  - T Invoke $\texttt{deleteResource}^{\mathsf{T}}(f)$
  - C If $\boldsymbol{isCacNeeded}^E(f)$:
    - ∗ For each $r \in R^E$ s.t. $\exists ops \subseteq OP \wedge (r, \langle ops, f\rangle) \in PA^E$:
      - · Invoke $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
    - ∗ Invoke $\texttt{deleteResource}^{\mathsf{C}}(f)$
  - E Update state to $\langle U^E, R^E, F^E \setminus \{f\}, UR^E, PA^E \setminus \{(r, \langle ops, f\rangle) \mid r \in R^E, ops \subseteq OP\}, EP^E \setminus \{(p, f) \mid p \in P^E\}\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
  - E Delete the content associated with f from the cloud
- $\texttt{assignUserToRole}^{\mathsf{E}}(u, r)$
  - E Update state to $\langle U^E, R^E, F^E, UR^E \cup \{(u, r)\}, PA^E, EP^E\rangle$
  - T Invoke $\texttt{assignUserToRole}^{\mathsf{T}}(u, r)$
  - C Invoke $\texttt{assignUserToRole}^{\mathsf{C}}(u, r)$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{assignPermissionToRole}^{\mathsf{E}}(r, \langle ops, f\rangle)$:
  - E Find $(r, \langle ops', f\rangle) \in PA^E$, otherwise set $ops' \leftarrow \{\}$
  - E Update state to $\langle U^E, R^E, F^E, UR^E, PA^E \setminus \{(u, \langle ops', f\rangle)\} \cup \{r, \langle ops' \cup ops, f\rangle\}, EP^E\rangle$
  - T Invoke $\texttt{assignPermissionToRole}^{\mathsf{T}}(r, \langle ops, f\rangle)$
  - C Invoke $\texttt{assignPermissionToRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{revokeUserFromRole}^{\mathsf{E}}(u, r)$
  - T Invoke $\texttt{revokeUserFromRole}^{\mathsf{T}}(u, r)$
  - C Invoke $\texttt{revokeUserFromRole}^{\mathsf{C}}(u, r)$
  - C If $\boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$:
    - ∗ Invoke $\texttt{rotateRoleKeyUserRole}^{\mathsf{C}}(r)$
  - C For each $f \in F^E$ s.t. $\exists ops \subseteq OP \wedge (r, \langle ops, f\rangle) \in PA^E \wedge \boldsymbol{isCacNeeded}^E(f)$:
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f\rangle)$:
      - · Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isEagerNeededOnRevP}^E(u, r, \langle op, f\rangle)$:
      - · Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - C If $\boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$:
    - ∗ Invoke $\texttt{rotateRoleKeyPermissions}^{\mathsf{C}}(r)$
  - E Update state to $\langle U^E, R^E, F^E, UR^E \setminus \{(u, r)\}, PA^E, EP^E\rangle$
- $\texttt{revokePermissionFromRole}^{\mathsf{E}}(r, \langle ops, f\rangle)$
  - T Invoke $\texttt{revokePermissionFromRole}^{\mathsf{T}}(r, \langle ops, f\rangle)$
  - C If $\boldsymbol{isCacNeeded}^E(f)$:
    - ∗ Invoke $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
  - C Set $\underline{encrypted} \leftarrow \boldsymbol{isCacNeeded}^E(f)$
  - E Find $(r, \langle ops', f\rangle) \in PA^E$
  - E If $ops' \setminus ops = \emptyset$:
    - ∗ Update state to $\langle U^E, R^E, F^E, UR^E, PA^E \setminus \{(r, \langle ops', f\rangle)\}, EP^E\rangle$
  - E Else:
    - ∗ Update state to $\langle U^E, R^E, F^E, UR^E, PA^E \setminus \{(r, \langle ops', f\rangle)\} \cup \{(r, ops' \setminus ops, f)\}, EP^E\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
  - C If $\boldsymbol{isCacNeeded}^E(f) = \underline{encrypted} = true$:
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f)$:
      - · Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
    - ∗ If $\exists op \in ops$ s.t. $\boldsymbol{isEagerNeededOnRevP}^E(r, op, f)$:
      - · Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
- $\texttt{assignPredicate}^{\mathsf{E}}(p, e)$
  - E Update state to $\langle U^E, R^E, F^E, UR^E, PA^E, EP^E \cup \{(p, e)\}\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{revokePredicate}^{\mathsf{E}}(p, e)$
  - E Update state to $\langle U^E, R^E, F^E, UR^E, PA^E, EP^E \setminus \{(p, e)\}\rangle$
  - E Invoke $\texttt{consistencyCheck}^{\mathsf{E}}()$
- $\texttt{readResource}^{\mathsf{E}}_u(f)$
  - E Retrieve fc^{enc} from the cloud
  - T If $\neg\boldsymbol{canDo}^T(u, read, f)$:
    - ∗ Return $\perp$
  - C If $\boldsymbol{isCacNeeded}^E(f)$:
    - ∗ Invoke $fc \leftarrow \texttt{readResource}^{\mathsf{C}}_u(f, fc^{enc})$
  - C Else:
    - ∗ Set $fc \leftarrow fc^{enc}$
  - E Return fc
- $\texttt{writeResource}^{\mathsf{E}}_u(f, fc)$
  - C If $\boldsymbol{isCacNeeded}^E(f)$
    - ∗ Invoke $fc^{enc} \leftarrow \texttt{writeResource}^{\mathsf{C}}_u(f, fc)$
  - C Else:
    - ∗ Set $fc^{enc} \leftarrow fc$
  - E Send fc^{enc} to the administrator
  - E The administrator checks whether $\boldsymbol{canDo}^T(u, write, f)$:
    - ∗ Send fc^{enc} to the cloud
    - ∗ Return $\top$
  - E Else:
    - ∗ Return $\perp$
- $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
  - E Retrieve fc^{enc} from the cloud
  - C Invoke $\texttt{rotateResourceKey}^{\mathsf{C}}(f, fc^{enc})$
- $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - E Retrieve fc^{enc} from the cloud
  - C Invoke $fc^{enc}_{new} \leftarrow \texttt{eagerReEncryption}^{\mathsf{C}}(f, fc^{enc})$
  - E Send fc^{enc}_{new} to the cloud
- $\texttt{consistencyCheck}^{\mathsf{E}}()$
  - C (*check 1.a*) For each $f \in F^E$ s.t. $\boldsymbol{isCacNeeded}^E(f) \wedge \neg\boldsymbol{isProtectedWithCAC}^C(f)$:
    - ∗ Retrieve and delete fc from the cloud
    - ∗ Invoke $\texttt{addResource}^{\mathsf{C}}_{adm}(f)$
    - ∗ For each $r \in R^E$ s.t. $\exists ops \subseteq OP \wedge (r, \langle ops, f\rangle) \in PA^E$:
      - · Invoke $\texttt{assignPermissionToRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
    - ∗ Invoke $fc^{enc} \leftarrow \texttt{writeResource}^{\mathsf{C}}_{adm}(f, fc)$
    - ∗ Send fc^{enc} to the cloud
  - C (*check 1.b*) For each $f \in F^E$ s.t. $\neg\boldsymbol{isCacNeeded}^E(f) \wedge \boldsymbol{isProtectedWithCAC}^C(f)$:
    - ∗ Retrieve and delete fc^{enc} from the cloud
    - ∗ Invoke $fc \leftarrow \texttt{readResource}^{\mathsf{C}}_{adm}(f, fc^{enc})$
    - ∗ For each $r \in R^E$ s.t. $\exists ops \subseteq OP \wedge (r, \langle ops, f\rangle) \in PA^E$:
      - · Invoke $\texttt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle ops, f\rangle)$
    - ∗ Invoke $\texttt{deleteResource}^{\mathsf{C}}(f)$
    - ∗ Send fc to the cloud
  - – Create set $\underline{rotating\_roles}$
  - C (*check 2*) For each $u \in U^E, r \in R^E$ s.t. $\neg\boldsymbol{canUserBe}^C(u, r) \wedge \boldsymbol{isRoleKeyRotationNeeded}^E(u, r) \wedge \boldsymbol{canUserBeCache}^C(u, r)$:
    - ∗ Invoke $\texttt{rotateRoleKeyUserRole}^{\mathsf{C}}(r)$
    - ∗ Add r to $\underline{rotating\_roles}$
  - C (*check 3*) For each $u \in U^E, f \in F^E$ s.t. $\exists r \in R^E \wedge \boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f\rangle) \wedge \neg\boldsymbol{canDo}^C(u, op, f) \wedge \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f) \wedge \exists op \in OP$:
    - ∗ Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
  - C (*check 4*) For each $u \in U^E, f \in F^E$ s.t. $\exists r \in R^E \wedge \boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f\rangle) \wedge \neg\boldsymbol{canDo}^C(u, op, f) \wedge \boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f) \wedge \exists op \in OP$:
    - ∗ Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - C (*check 5*) For each $r \in R^E, f \in F^E$ s.t. $\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f) \wedge \neg\boldsymbol{canRoleDo}^C(r, op, f) \wedge \boldsymbol{canRoleDoCacheLast}^C(r, op, f) \wedge \exists op \in OP$:
    - ∗ Invoke $\texttt{rotateResourceKey}^{\mathsf{E}}(f)$
  - C (*check 6*) For each $r \in R^E, f \in F^E$ s.t. $\boldsymbol{isEagerNeededOnRevP}^E(r, op, f) \wedge \neg\boldsymbol{canRoleDoCache}^C(r, op, f) \wedge \boldsymbol{canRoleDoCache}^C(r, op, f) \wedge \exists op \in OP$:
    - ∗ Invoke $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$
  - C (*part of check 2*) For each $r \in \underline{rotating\_roles}$:
    - ∗ Invoke $\texttt{rotateRoleKeyPermissions}^{\mathsf{C}}(r)$

the predicates) of the administrator; consider the initial state γ^{E^i}:

$$U^{E^i}=\{alice,bob\}; \quad R^{E^i}=\{staff,accounting\}; \quad F^{E^i}=\{budget\};$$
$$UR^{E^i}=\{(alice,staff),(bob,accounting)\};$$
$$PA^{E^i}=\{(staff,\langle\{read\},budget\rangle),(accounting,\langle\{read,write\},budget\rangle)\};$$
$$EP^{E^i}=\{(\texttt{cloudNoEnforce},budget),(\texttt{enc},budget),(\texttt{untrusted},alice)\};$$

Assume now that the administrator wants to remove *alice* from γ^{E^i} (because, e.g., *alice* has left the organization). Hence, the administrator invokes $\texttt{deleteUser}^{\mathsf{E}}(alice)$ (see Table 5). Since *alice* is assigned to *staff*, $\texttt{revokeUserFromRole}^{\mathsf{C}}(alice, staff)$ is invoked and the query $\boldsymbol{isRoleKeyRotationNeeded}^E(alice, staff)$ is evaluated as specified by the entailment: given that $\texttt{untrusted}(alice) \in EP^{E^i}$, the query evaluates at *true* and $\texttt{rotateRoleKeyUserRole}^{\mathsf{C}}(staff)$ — which represents the role key rotation procedure for UR^C — is invoked. Then, *alice* is removed from γ^{T^i} and γ^{C^i} by invoking $\texttt{deleteUserT}(alice)$ and $\texttt{deleteUser}^{\mathsf{C}}(alice)$, respectively. Afterwards, for each resource to which *alice* had access that is protected with CAC — i.e., *budget* — the query $\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(alice, staff, \langle\cdot, budget\rangle)$ is evaluated: given that $\texttt{cac}(budget), \texttt{cloudNoEnforce}(budget), \texttt{untrusted}(alice) \in EP^{E^i}$, the query evaluates at *true* and $\texttt{rotateResourceKey}^{\mathsf{E}}(budget)$ — which represents the resource key rotation procedure — is invoked. Instead, the query $\boldsymbol{isEagerNeededOnRevUR}^E(alice, staff, \langle\cdot, budget\rangle)$ evaluates at *false* since $\texttt{eager}(budget) \notin EP^{E^i}$; hence, $\texttt{eagerReEncryption}^{\mathsf{E}}(f)$ is not invoked. Finally, since $\boldsymbol{isRoleKeyRotationNeeded}^E(alice, staff)$ was already evaluated at *true* before, $\texttt{rotateRoleKeyPermissions}^{\mathsf{C}}(staff)$ — which represents the role key rotation procedure for PA^C — is invoked. At the end, γ^{E^i} is modified to $\gamma^{E^{i+1}}$ and the consistency check is executed.

Conversely, if the administrator wanted to remove *bob* from γ^{E^i}, no state-change rule of the CAC scheme would have been invoked (given that $\texttt{untrusted}(bob) \notin EP^{E^i}$). By defining predicates and the entailment for each query, administrators can fine-tune the deployment of our hybrid AC scheme to suit their scenarios.

6 Consistency Check

Of course, the hybrid AC scheme is unusable without guarantees on both the correctness and safety properties of the enforcement.

Safety. The safety of the core RBAC model and the CAC scheme were already demonstrated in [15, 16], respectively. In our hybrid AC scheme, to guarantee safety we need to ensure that no unnecessary permissions are granted or revoked during state transitions caused by the execution of a single state-change rule. We argue that safety is easily inherited from the core RBAC model by considering that in the hybrid AC scheme (*i*) the entailment for the $\boldsymbol{canDo}^E(u, \langle f, op\rangle)$ query is equal to that for the $\boldsymbol{canDo}^T(u, \langle f, op\rangle)$ query, and (*ii*) the new state-change rules in Ψ^E (see Section 5.4) modify neither UR^E nor PA^E — in simpler words, permissions are neither granted nor revoked during state transitions triggered by the new state-change rules in Ψ^E, hence safety is preserved.

Correctness. To guarantee correctness, we need to ensure that, for any given state, the $\boldsymbol{canDo}^E(u, f, op)$ query always evaluates at the same value (be it *true* or *false*) as the $\boldsymbol{canDo}^T(u, f, op)$ and the $\boldsymbol{canDo}^C(u, op, f)$ queries — assuming that CAC is applied over f. However, understanding what resources a user can access in the CAC scheme is tricky. In fact, besides considering the current state of the CAC policy γ^{C^i}, we must also consider all those roles' and resources' keys that were not rotated and therefore users may have cached. Note that this was not a problem in the original version of the CAC scheme [16] (nor in further refinements, e.g., [5]), since the role key rotation, resource key rotation, and resource re-encryption procedures — which make cached keys useless — were always executed. However, this is not true in the hybrid AC scheme, given that the execution of such procedures may be avoided depending on the security model specified by the administrator. In other words, the correctness must not consider the $\boldsymbol{canDo}^C(u, op, f)$ query only, but instead all 7 queries in Table 4. Below, we discuss the correctness for each of the 7 queries (see the summary in Table 7) and consequently derive as many invariants — we define an invariant as a boolean condition which should always be true regardless of state transitions. When all invariants hold, correctness is guaranteed. Verifying these invariants constitutes the consistency check that we report as the state-change rule $\texttt{consistencyCheck}^{\mathsf{E}}()$ (see Table 5).

$\boldsymbol{canDo}^E(u, op, f)$*:* determines whether u can access the permission $\langle f, op\rangle$ legitimately. The correctness is guaranteed by the invariant $\boldsymbol{canDo}^E(u, op, f) \Leftrightarrow (\boldsymbol{canDo}^T(u, op, f) \wedge \boldsymbol{canDo}^C(u, op, f))$; that is, if u can access $\langle f, op\rangle$ in S^E, then u must also access $\langle f, op\rangle$ in both S^T and S^C — assuming that $\boldsymbol{isProtectedWithCAC}^C(f)$.

$\boldsymbol{isCacNeeded}^E(f)$*:* determines whether CAC must be used to protect f. The correctness is guaranteed by the invariant $\boldsymbol{isCacNeeded}^E(f) \Leftrightarrow \boldsymbol{isProtectedWithCAC}^C(f)$. However, some state transitions may invalidate this invariant. For instance, consider a resource $f \in F^E$ s.t. $(\texttt{cac}, f) \notin EP$, therefore $\neg\boldsymbol{isCacNeeded}^E(f)$ (i.e., f is not currently protected with CAC). Now, assume the administrator invokes $\texttt{assignPredicate}^{\mathsf{E}}(\texttt{cac}, f)$. As a consequence, $\boldsymbol{isCacNeeded}^E(f)$ evaluates to *true* as shown in Table 4, while $\boldsymbol{isProtectedWithCAC}^C(f)$ still evaluates to *false*, invalidating the invariant since f is not protected with CAC (while now it should be). In cases like this, we need to check and update the state of the CAC policy to respect the invariant: for each $f \in F^E$ s.t. $\boldsymbol{isCacNeeded}^E(f) \wedge \neg\boldsymbol{isProtectedWithCAC}^C(f)$, we add the resource to F^C, encrypt fc, and carry all permissions $(*, \langle *, f\rangle) \in PA^E$ to PA^C. Similarly, for each $f \in F^E$ s.t. $\neg\boldsymbol{isCacNeeded}^E(f) \wedge \boldsymbol{isProtectedWithCAC}^C(f)$, we remove the resource from F^C, decrypt fc^{enc}, and delete all permissions $\langle *, f, *\rangle \in PA^C$. We report these checks as *check 1.a* and *check 1.b* in $\texttt{consistencyCheck}^{\mathsf{E}}()$.

$\boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$*:* determines whether r's keys must be rotated in the event u is revoked from r. The correctness is guaranteed by the invariant $\boldsymbol{isRoleKeyRotationNeeded}^E(u, r) \Rightarrow \boldsymbol{canUserBe}^C(u, r) \vee \neg\boldsymbol{canUserBeCache}^C(u, r)$; that is, either u was not revoked from r ($\boldsymbol{canUserBe}^C(u, r)$) — hence, rotating r's keys is not necessary — or u cannot assume r even with cached keys ($\neg\boldsymbol{canUserBeCache}^C(u, r)$). Similarly to $\boldsymbol{isCacNeeded}^E(f)$, some state transitions may invalidate this invariant. For instance, consider $u \in U^E, r \in R^E$ s.t. $(u, r) \in UR^E \wedge (\texttt{untrusted}, u) \in EP$, therefore $\boldsymbol{isRoleKeyRotationNeeded}^E(u, r)$. Now, assume the administrator invokes $\texttt{revokeUserFromRole}^{\mathsf{E}}(u, r)$. Being untrusted, u may have cached r's keys. As a consequence, $\boldsymbol{canUserBeCache}^C(u,r)$ evaluates to *true* as shown in Table 4, invalidating the invariant. In cases like this, we need to check and update the state of the CAC policy to respect the invariant: for each $u \in U^E, r \in R^E$ s.t. $\neg \boldsymbol{canUserBe}^C(u, r) \wedge \boldsymbol{isRoleKeyRotationNeeded}^E(u,r) \wedge \boldsymbol{canUserBeCache}^C(u,r)$, we rotate r's keys by invoking `rotateRoleKeyUserRole`$^{\mathsf{C}}(r)$ and `rotateRoleKeyPermissions`$^{\mathsf{C}}(r)$. We report this check as *check 2* in `consistencyCheck`$^{\mathsf{E}}()$.

$\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f \rangle)$*:* determines whether f's key must be rotated in the event u is revoked from r. The correctness is guaranteed by the invariant $\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f \rangle) \implies \boldsymbol{canDo}^C(u, op, f) \vee \neg \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$; that is, either u can access $\langle f, op \rangle$ via r (or even via another role $r' \neq r$) — hence, rotating f's key is not necessary — or u cannot access $\langle f, op \rangle$ even with r's cached keys ($\neg \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$ — "last" because $\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f \rangle)$ only provides key rotation (protecting the last version of f) and not re-encryption (see the last two queries below for re-encryption), hence u may still access older versions of f. Whenever a state transition invalidates this invariant, we check and update the state of the CAC policy: for each $u \in U^E, f \in F^E$ s.t. $\exists r \in R^E \wedge \boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f \rangle) \wedge \neg \boldsymbol{canDo}^C(u, op, f) \wedge \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f) \wedge \exists op \in OP$, we rotate f's keys by invoking `rotateResourceKey`$^{\mathsf{C}}(f, fc^{enc})$. We report this check as *check 3* in `consistencyCheck`$^{\mathsf{E}}()$.

$\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f)$*:* determines whether f's key must be rotated when $\langle f, op \rangle$ is revoked from r. The correctness is guaranteed by the invariant $\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f) \implies \boldsymbol{canRoleDo}^C(r, op, f) \vee \neg \boldsymbol{canRoleDoCacheLast}^C(r, op, f)$; that is, either $\langle f, op \rangle$ was not revoked from r ($\boldsymbol{canRoleDo}^C(r, op, f)$) — hence, rotating f's keys is not necessary — or no user can access $\langle f, op \rangle$ via r's cached keys ($\neg \boldsymbol{canRoleDoCacheLast}^C(r, op, f)$). Whenever a state transition invalidates this invariant, we check and update the state of the CAC policy: for each $r \in R^E, f \in F^E$ s.t. $\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f) \wedge \neg \boldsymbol{canRoleDo}^C(r, op, f) \wedge \boldsymbol{canRoleDoCacheLast}^C(r, op, f) \wedge \exists op \in OP$, we rotate f's keys by invoking `rotateResourceKey`$^{\mathsf{E}}(f)$. We report this check as *check 5* in `consistencyCheck`$^{\mathsf{E}}()$.

$\boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f \rangle)$*:* determines whether to apply eager re-encryption over fc when u is revoked from r. The correctness is guaranteed by the invariant $\boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f \rangle) \implies \boldsymbol{canDo}^C(u, op, f) \vee \neg \boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f)$; that is, either u can access $\langle f, op \rangle$ via r (or even via another role $r' \neq r$) — hence, apply eager re-encryption over fc is not necessary — or u cannot access $\langle f, op \rangle$ even with r's cached keys ($\neg \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$). Whenever a state transition invalidates this invariant, we check and update the state of the CAC policy: for each $u \in U^E, f \in F^E$ s.t. $\exists r \in R^E \wedge \boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f \rangle) \wedge \neg \boldsymbol{canDo}^C(u, op, f) \wedge \boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f) \wedge \exists op \in OP$, we immediately re-encrypt fc by invoking `eagerReEncryption`$^{\mathsf{C}}(f, fc^{enc})$. We report this check as *check 4* in `consistencyCheck`$^{\mathsf{E}}()$.

$\boldsymbol{isEagerNeededOnRevP}^E(r, op, f)$*:* determines whether to apply eager re-encryption over fc when $\langle f, op \rangle$ is revoked from r. The correctness is guaranteed by the invariant $\boldsymbol{isEagerNeededOnRevP}^E(r, op, f) \implies \boldsymbol{canRoleDo}^C(r, op, f) \vee \neg \boldsymbol{canRoleDoCache}^C(r, op, f)$; that is, either $\langle f, op \rangle$ was not revoked from r ($\boldsymbol{canRoleDo}^C(r, op, f)$) — hence, apply eager re-encryption over fc is not necessary — or no user can access $\langle f, op \rangle$ via r's cached keys ($\neg \boldsymbol{canRoleDoCache}^C(r, op, f)$). Whenever a state transition invalidates this invariant, we check and update the state of the CAC policy: for each $r \in R^E, f \in F^E$ s.t. $\boldsymbol{isEagerNeededOnRevP}^E(r, op, f) \wedge \neg \boldsymbol{canRoleDoCache}^C(r, op, f) \wedge \boldsymbol{canRoleDoCache}^C(r, op, f) \wedge \exists op \in OP$, we immediately re-encrypt fc by invoking `eagerReEncryption`$^{\mathsf{C}}(f, fc^{enc})$. We report this check as *check 6* in `consistencyCheck`$^{\mathsf{E}}()$.

7 Preliminary Experimentation

We implement the pseudocode of S^E (Table 5), S^T (Table 2), and S^C (Table 9) in Prolog; we also implement a script invoking the APIs of `CryptoAC`, a tool implementing the CAC scheme in [5] to actually execute (and measure the execution time of) cryptographic computations. We deploy all software on a laptop running Ubuntu 24.04 on an Intel(R) Core(TM) i7-1355U and 16GB of RAM. As a starting point, we consider a RBAC state mined from real-world datasets (`domino` — see [16]) to then generate a random sequence of 100 state-change rules equally distributed among those in Table 2. Then, we execute the sequence on 6 different configurations (C): $C0$ corresponds to S^T, $C100$ corresponds to S^C, and $C20 \rightarrow C80$ correspond to S^E in which each predicate proposed in Table 3 is randomly assigned to 20%→80% of the elements in the RBAC state (the more predicates assigned, the more CAC is employed); that is, $EP_{C0} \subset EP_{C20} \subset EP_{C40} \subset EP_{C60} \subset EP_{C80} \subset EP_{C100}$. As metrics, we collect the number of times a state-change rule is executed in S^T and S^C and the total execution time of S^E, S^T, S^C, and `CryptoAC`. We report the results of the experimental evaluation in Table 6.[2]

[2]The Prolog implementation, raw data results, the sequence of 100 state-change rules, and a replication package are available at https://github.com/stfbk/SACMAT2025-results/. We will publish the data and results of further experiments and future developments in https://aleph.fbk.eu/tools/CryptoAC.

Table 6: Results of the preliminary experimental evaluation

state-change rule		$C0$	$C20$		$C40$		$C60$		$C80$		$C100$
		S^T	S^T	S^C	S^T	S^C	S^T	S^C	S^T	S^C	S^C
`addUser()`		7	7	7	7	7	7	7	7	7	7
`addRole()`		10	10	10	10	10	10	10	10	10	10
`addResource()`		10	10	2	10	5	10	7	10	10	10
`assignUserToRole()`		6	6	6	6	6	6	6	6	6	6
`assignPermissionToRole()`		12	12	8	12	16	12	22	12	29	32
`deleteUser()`		6	6	6	6	6	6	6	6	6	6
`deleteRole()`		7	7	7	7	7	7	7	7	7	7
`deleteResource()`		10	10	3	10	5	10	6	10	8	10
`revokeUserFromRole()`		10	10	10	10	10	10	10	10	10	10
`revokePermissionFromRole()`		13	13	21	13	45	13	66	13	90	112
`readResource()`		3	3	7	3	22	3	38	3	66	78
`writeResource()`		6	6	8	6	24	6	45	6	77	91
total execution time (in ms)	S^E only	—	83		271		451		731		—
	S^T only	<1	<1		<1		<1		<1		—
	S^C only	—	<1		2		2		3		4
	`CryptoAC` only	—	83		432		370		490		742
	sum of the 4 above	<1	166		705		823		1,224		746

The results show that our hybrid AC scheme does reduce the number of invoked state-change rules of S^C, and the execution time of CryptoAC decreases accordingly. However, the reasoning effort in S^E — mainly due to the consistency check — may even outweigh the performance gain (e.g., $C60$ and $C80$ take longer than $C100$). Nonetheless, our experimental evaluation is still preliminary and further analysis should include (*i*) non-random workloads, (*ii*) key distribution and network latency, and (*iii*) study of the interplay among predicates. In summary, the Prolog implementation is a proof-of-concept showing efficiency improvements for CAC in specific configurations. For the other configurations, we can further enhance performances by optimizing the Prolog rule structure. Notably, our hybrid schema's design allows implementations across different technologies, such as Datalog [11], offering the advantage of integrating with powerful database engines.

8 Conclusion

In this paper, we proposed a hybrid AC scheme to reduce the computational overhead of CAC by allowing administrators to express (via predicates) trust assumptions and data protection requirements on users, roles, and resources — that is, tunable security models. The final goal is to make CAC a viable solution for enforcing RBAC policies in real-world scenarios by automatically dividing enforcement between CAC and centralized AC while preserving maximum data protection. Furthermore, administrators can easily define new predicates and adapt queries' entailment to their scenario. We also designed a consistency check to ensure the correctness and safety of the enforcement of the hybrid AC scheme and conducted a preliminary experimental evaluation on a proof-of-concept implementation. The results show that our hybrid AC scheme often performs better than CAC by a degree that depends on the security model.

Future work. We can identify a number of compelling research directions stemming from our work. First, we plan to conduct a more thorough performance evaluation, assessing the performance of our hybrid AC scheme under real-world scenarios in which to apply our hybrid AC scheme and conduct surveys to assess its usability. Then, we intend to apply our solution to ABAC relying on an Attribute-Based Encryption (ABE) scheme for CAC [24]. In addition, we plan to enhance the expressiveness of the hybrid AC model by supporting negative permissions, that is, allowing for permitting or prohibiting specific permissions from specific users, overriding the normal behavior of RBAC. Finally, we plan to integrate other strategies (e.g., those in Sections 2 and 7) to further improve the efficiency of CAC and reasoning.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU and by Ministero delle Imprese e del Made in Italy (IPCEI Cloud DM 27 giugno 2022 – IPCEI-CL-0000007) and European Union (Next Generation EU).

References

[1] J. Alderman, N. Farley, and J. Crampton. 2017. Tree-Based Cryptographic Access Control. In Computer Security – ESORICS 2017, S. N. Foley, D. Gollmann, and E. Snekkenes (Eds.). Springer International Publishing, Cham, 47–64.

[2] Y. Ashibani and Q. H. Mahmoud. 2017. Cyber physical systems security: Analysis, challenges and solutions. Computers & Security 68 (2017), 81–97. https://doi.org/10.1016/j.cose.2017.04.005

[3] B. M. Babu and M. S. Bhanu. 2015. Prevention of Insider Attacks by Integrating Behavior Analysis with Risk based Access Control Model to Protect Cloud. Procedia Computer Science 54 (2015), 157–166. https://doi.org/10.1016/j.procs.2015.06.018

[4] N. Baracaldo and J. Joshi. 2013. An adaptive risk management and access control framework to mitigate insider threats. Computers & Security 39 (2013), 237–254. https://doi.org/10.1016/j.cose.2013.08.001

[5] S. Berlato. 2024. A Security Service for Performance-Aware End-to-End Protection of Sensitive Data in Cloud Native Applications. PhD thesis. University of Genoa. Available at https://hdl.handle.net/11567/1174596.

[6] S. Berlato, R. Carbone, A. J. Lee, and S. Ranise. 2021. Formal Modelling and Automated Trade-off Analysis of Enforcement Architectures for Cryptographic Access Control in the Cloud. ACM Trans. Priv. Secur. 25, 1, Article 2 (nov 2021), 37 pages. https://doi.org/10.1145/3474056

[7] S. Berlato, R. Carbone, and S. Ranise. 2021. Cryptographic Enforcement of Access Control Policies in the Cloud: Implementation and Experimental Assessment. In Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021, Sabrina De Capitani di Vimercati and Pierangela Samarati (Eds.). SCITEPRESS, 370–381. https://doi.org/10.5220/0010608003700381

[8] S. Berlato, U. Morelli, R. Carbone, and S. Ranise. 2022. End-to-End Protection of IoT Communications Through Cryptographic Enforcement of Access Control Policies. In Data and Applications Security and Privacy XXXVI, Shamik Sural and Haibing Lu (Eds.). Vol. 13383. Springer International Publishing, 236–255. https://doi.org/10.1007/978-3-031-10684-2_14 Series Title: Lecture Notes in Computer Science.

[9] F. Cai, N. Zhu, J. He, P. Mu, W. Li, and Y. Yu. 2019. Survey of access control models and technologies for cloud computing. Cluster Computing 22 (2019), 6111–6122. Issue S3. https://doi.org/10.1007/s10586-018-1850-7

[10] I. Celikbilek, B. Celiktas, and E. Ozdemir. 2024. A Hierarchical Key Assignment Scheme: A Unified Approach for Scalability and Efficiency. IEEE Access 12 (2024), 70568–70580. https://doi.org/10.1109/ACCESS.2024.3401844

[11] S. Ceri, G. Gottlob, and L. Tanca. 1989. What you always wanted to know about Datalog (and never dared to ask). IEEE Transactions on Knowledge and Data Engineering 1, 1 (1989), 146–166. https://doi.org/10.1109/69.43410

[12] J. Crampton. 2011. Time-Storage Trade-Offs for Cryptographically-Enforced Access Control. In Computer Security – ESORICS 2011, Vijay Atluri and Claudia Diaz (Eds.). Vol. 6879. Springer Berlin Heidelberg, 245–261. https://doi.org/10.1007/978-3-642-23822-2_14 Series Title: Lecture Notes in Computer Science.

[13] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, and K. Moody. 2004. Using trust and risk in role-based access control policies. In Proceedings of the ninth ACM symposium on Access control models and technologies (Yorktown Heights New York USA, 2004-06-02). ACM, 156–162. https://doi.org/10.1145/990036.990062

[14] A. L. Ferrara, F. Paci, and C. Ricciardi. 2021. Verifiable Hierarchical Key Assignment Schemes. In Data and Applications Security and Privacy XXXV, Ken Barker and Kambiz Ghazinour (Eds.). Vol. 12840. Springer International Publishing, 357–376. https://doi.org/10.1007/978-3-030-81242-3_21 Series Title: Lecture Notes in Computer Science.

[15] American National Standard for Information Technology. 2004. Role Based Access Control. Standard. American National Standards Institute, Inc.

[16] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee. 2016. On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud. In 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 819–838. https://doi.org/10.1109/SP.2016.54 event-place: San Jose, CA.

[17] B. Gwak, J.-H. Cho, D. Lee, and H. Son. 2018. TARAS: Trust-Aware Role-Based Access Control System in Public Internet-of-Things. In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (New York, NY, USA, 2018-08). IEEE, 74–85. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00022

[18] T. L. Hinrichs, D. Martinoia, W. C. Garrison, A. J. Lee, A. Panebianco, and L. Zuck. 2013. Application-Sensitive Access Control Evaluation Using Parameterized Expressiveness. In 2013 IEEE 26th Computer Security Foundations Symposium (New Orleans, LA, USA, 2013-06). IEEE, 145–160. https://doi.org/10.1109/CSF.2013.17

[19] N. Li and M. V. Tripunitara. 2006. Security analysis in role-based access control. ACM Trans. Inf. Syst. Secur. 9, 4 (nov 2006), 391–420. https://doi.org/10.1145/1187441.1187442

[20] P. N. Mahalle, P. A. Thakre, N. R. Prasad, and R. Prasad. 2013. A fuzzy approach to trust based access control in internet of things. In Wireless VITAE 2013 (Atlantic City, NJ, USA, 2013-06). IEEE, 1–5. https://doi.org/10.1109/VITAE.2013.6617083

[21] G. D. Putra, V. Dedeoglu, S. S. Kanhere, and R. Jurdak. 2020. Trust Management in Decentralized IoT Access Control System. In 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) (Toronto, ON, Canada, 2020-05). IEEE, 1–9. https://doi.org/10.1109/ICBC48266.2020.9169481

[22] S. Qi and Y. Zheng. 2021. Crypt-DAC: Cryptographically Enforced Dynamic Access Control in the Cloud. IEEE Transactions on Dependable and Secure Computing 18, 2 (2021), 765–779. https://doi.org/10.1109/TDSC.2019.2908164

[23] P. Samarati and S. de Capitani di Vimercati. 2000. Access Control: Policies, Models, and Mechanisms. In Foundations of Security Analysis and Design, Riccardo Focardi and Roberto Gorrieri (Eds.). Vol. 2171. Springer Berlin Heidelberg, 137–196. https://doi.org/10.1007/3-540-45608-2_3

[24] Yinghui Zhang, Robert H. Deng, Shengmin Xu, Jianfei Sun, Qi Li, and Dong Zheng. 2020. Attribute-based Encryption for Cloud Computing Access Control: A Survey. ACM Comput. Surv. 53, 4, Article 83 (Aug. 2020), 41 pages. https://doi.org/10.1145/3398036

A Additional Content

Below, in Table 7 we provide the invariant and the description of each of the queries in the set Q^E of the hybrid AC scheme S^E — note that the information in Table 7 is just a summary of what has already been explained in Section 5 and shown in Table 5 (see the state-change rule $\mathtt{consistencyCheck}^{\mathsf{E}}()$). Also, we provide additional information on the CAC scheme [5] and the adjustments described in Section 4. In detail, in Table 8 we provide the extended set of queries Q^C of the CAC scheme S^C — along with a textual description and the entailment — adjusted from [5]. Then, in Table 9 we provide the pseudocode of the state-change rules in Ψ^C of the CAC scheme S^C adjusted from [5]. Finally, for the sake of completeness, we provide the complete version of the decorated sets $\gamma^C = \langle U^C, R^C, F^C, UR^C, PA^C \rangle$ that compose a state $\gamma^C \in \Gamma^C$ of the CAC scheme S^C in [5]:

$$\langle \mathsf{U}, u, \mathsf{p}_u, \mathsf{k}_u^{\mathsf{enc}}, \mathsf{k}_u^{\mathsf{ver}}, \mathtt{Sign}^{\mathtt{Sig}} \rangle \in U^C_{\mathsf{ope}};$$
$$\langle \mathsf{R}, r, \mathsf{p}_{(r,v_r)}, \mathsf{k}_{(r,v_r)}^{\mathsf{enc}}, \mathsf{k}_{(r,v_r)}^{\mathsf{ver}}, v_r, \mathtt{Sign}^{\mathtt{Sig}} \rangle \in R^C_{\mathsf{ope}};$$
$$\langle \mathsf{F}, f, \mathsf{p}_{(f,\widehat{v_f})}, \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}}(\mathsf{k}^{\mathsf{sym}}_{(f,\widehat{v_f})}), \widehat{v_f}, \mathtt{Sign}^{\mathtt{Sig}} \rangle;$$
$$\langle \mathsf{UR}, u, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}_u^{\mathsf{enc}}}((\mathsf{k}^{\mathsf{enc}}_{(r,v_r)}, \mathsf{k}^{\mathsf{dec}}_{(r,v_r)}, \mathsf{k}^{\mathsf{ver}}_{(r,v_r)}, \mathsf{k}^{\mathsf{sig}}_{(r,v_r)})), v_r, \mathtt{Sign}^{\mathtt{Sig}} \rangle \in UR^C_{\mathsf{ope}};$$
$$\langle \mathsf{PA}, r, f, \mathsf{p}_{(r,v_r)}, \mathsf{p}_{(f,v_f)}, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{(r,v_r)}}(\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}), v_r, v_f, ops, \mathtt{Sign}^{\mathtt{Sig}} \rangle.$$

Received 31 March 2025; revised ?? ????? ????; accepted ?? ????? ????

Table 7: Invariants for queries in Q^E

Invariant	Description
$\boldsymbol{canDo}^E(u, op, f) \Leftrightarrow (\boldsymbol{canDo}^T(u, op, f) \wedge \boldsymbol{canDo}^C(u, op, f))$	u can access $\langle f, op \rangle$ in S^E if and only if u can access $\langle f, op \rangle$ also in S^T and S^C
$\boldsymbol{isCacNeeded}^E(f) \Leftrightarrow \boldsymbol{isProtectedWithCAC}^C(f)$	If the security model states that CAC is needed for f, then f is protected with CAC (and vice versa)
$\boldsymbol{isRoleKeyRotationNeeded}^E(u, r) \Rightarrow \boldsymbol{canUserBe}^C(u, r) \vee \neg \boldsymbol{canUserBeCache}^C(u, r)$	If r's keys must be rotated in the event u is revoked from r, then either u was not revoked from r yet or u cannot assume r even with cached keys
$\boldsymbol{isResourceKeyRotationNeededOnRevUR}^E(u, r, \langle op, f \rangle) \Rightarrow \boldsymbol{canDo}^C(u, op, f) \vee \neg \boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$	If f's key must be rotated in the event u is revoked from r, then either u can access $\langle f, op \rangle$ via r (or even via another role $r' \neq r$) or u cannot access $\langle f, op \rangle$ even with r's cached keys
$\boldsymbol{isResourceKeyRotationNeededOnRevP}^E(r, op, f) \Rightarrow \boldsymbol{canRoleDo}^C(r, op, f) \vee \neg \boldsymbol{canRoleDoCacheLast}^C(r, op, f)$	If f's key must be rotated in the event $\langle f, op \rangle$ is revoked from r, then either $\langle f, op \rangle$ was not revoked from r or no user can access $\langle f, op \rangle$ via r's cached keys
$\boldsymbol{isEagerNeededOnRevUR}^E(u, r, \langle op, f \rangle) \Rightarrow \boldsymbol{canDo}^C(u, op, f) \vee \neg \boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f)$	If eager re-encryption must be applied over fc in the event u is revoked from r, then either u can access $\langle f, op \rangle$ via r (or even via another role $r' \neq r$) or u cannot access $\langle f, op \rangle$ even with r's cached keys
$\boldsymbol{isEagerNeededOnRevP}^E(r, op, f) \Rightarrow \boldsymbol{canRoleDo}^C(r, op, f) \vee \neg \boldsymbol{canRoleDoCache}^C(r, op, f)$	If eager re-encryption must be applied over fc in the event $\langle f, op \rangle$ is revoked from r, then either $\langle f, op \rangle$ was not revoked from r or no user can access $\langle f, op \rangle$ via r's cached keys

Table 8: The extended set of queries Q^C of the CAC scheme S^C adjusted from [5] (see Section 4)

Query $q \in Q^E$	Description	Entailment — that is, how the query is evaluated
$\boldsymbol{canDo}^C(u, op, f)$	Querying whether u can access the permission $\langle f, op \rangle$ legitimately	$\boldsymbol{canDo}^C(u, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{UR}, u, r, *, v_r, * \rangle \in UR^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, v_r, v_f, ops, * \rangle \in PA^C_{\mathsf{ope}} \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, v_f, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canUserDoViaRole}^C(u, r, op, f)$	Querying whether u can access the permission $\langle f, op \rangle$ specifically through the role r legitimately	$\boldsymbol{canUserDoViaRole}^C(u, r, op, f) \Leftrightarrow \exists \langle \mathsf{UR}, u, r, *, v_r, * \rangle \in UR^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, v_r, v_f, ops, * \rangle \in PA^C_{\mathsf{ope}} \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, v_f, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canRoleDo}^C(r, op, f)$	Querying whether r can access the permission $\langle f, op \rangle$ legitimately	$\boldsymbol{canRoleDo}^C(r, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, v_r, v_f, op, * \rangle \in PA^C_{\mathsf{ope}} \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, v_f, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f)$	Querying whether u can access the permission $\langle f, op \rangle$ on at least one version of f through the role r possibly using cached keys	$\boldsymbol{canUserDoViaRoleCache}^C(u, r, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle \in (R^C_{\mathsf{ope}} \cup R^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{UR}, u, r, *, \widehat{v_r}, * \rangle \in (UR^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, \widehat{v_r}, \widehat{v_f}, op, * \rangle \in (PA^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f)$	Querying whether u can access the permission $\langle f, op \rangle$ on the latest version of f through the role r possibly using cached keys	$\boldsymbol{canUserDoViaRoleCacheLast}^C(u, r, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle \in (R^C_{\mathsf{ope}} \cup R^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{UR}, u, r, *, \widehat{v_r}, * \rangle \in (UR^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, \widehat{v_r}, v_f, op, * \rangle \in (PA^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, v_f, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canRoleDoCache}^C(r, op, f)$	Querying whether r can access the permission $\langle f, op \rangle$ on at least one version of f possibly using cached keys	$\boldsymbol{canRoleDoCache}^C(r, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle \in (R^C_{\mathsf{ope}} \cup R^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, \widehat{v_r}, \widehat{v_f}, op, * \rangle \in (PA^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canRoleDoCacheLast}^C(r, op, f)$	Querying whether r can access the permission $\langle f, op \rangle$ on the latest version of f possibly using cached keys	$\boldsymbol{canRoleDoCacheLast}^C(r, op, f) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle \in (R^C_{\mathsf{ope}} \cup R^C_{\mathsf{hide}}) \wedge \exists \langle \mathsf{PA}, r, f, *, *, *, \widehat{v_r}, v_f, op, * \rangle \in (PA^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}}) \wedge op \in ops \wedge \exists \langle \mathsf{F}, f, *, *, v_f, * \rangle \in F^C_{\mathsf{ope}}$
$\boldsymbol{canUserBe}^C(u, r)$	Querying whether u can assume the role r	$\boldsymbol{canUserBe}^C(u, r) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{UR}, u, r, *, v_r, * \rangle \in UR^C_{\mathsf{ope}}$
$\boldsymbol{canUserBeCache}^C(u, r)$	Querying whether u can assume the role r possibly using cached keys	$\boldsymbol{canUserBeCache}^C(u, r) \Leftrightarrow \exists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{UR}, u, r, *, v_r, * \rangle \in (UR^C_{\mathsf{ope}} \cup UR^C_{\mathsf{hide}})$
$\boldsymbol{isProtectedWithCAC}^C(f)$	Querying whether f is protected with CAC	$\boldsymbol{isProtectedWithCAC}^C(f) \Leftrightarrow \exists \langle \mathsf{F}, f, *, *, *, * \rangle \in F^C_{\mathsf{ope}}$

Table 9: The state-change rules in Ψ^C of the CAC scheme S^C adjusted from [5] (see Section 4)

- $\mathtt{init}^{\mathsf{C}}()$
  - Generate $(\mathsf{k}^{\mathsf{enc}}_{adm}, \mathsf{k}^{\mathsf{dec}}_{adm}) \leftarrow \mathsf{Gen}^{\mathsf{Pub}}$ and $(\mathsf{k}^{\mathsf{ver}}_{adm}, \mathsf{k}^{\mathsf{sig}}_{adm}) \leftarrow \mathsf{Gen}^{\mathsf{Sig}}$
  - Add $\langle \mathsf{U}, adm, adm, \mathsf{k}^{\mathsf{enc}}_{adm}, \mathsf{k}^{\mathsf{ver}}_{adm}, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to U^C_{ope}
  - Add $\langle \mathsf{R}, adm, adm, \mathsf{k}^{\mathsf{enc}}_{adm}, \mathsf{k}^{\mathsf{ver}}_{adm}, 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to R^C_{ope}
  - Add $\langle \mathsf{UR}, adm, adm, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{adm}}(\mathsf{k}^{\mathsf{enc}}_{adm}, \mathsf{k}^{\mathsf{dec}}_{adm}, \mathsf{k}^{\mathsf{ver}}_{adm}, \mathsf{k}^{\mathsf{sig}}_{adm}), 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to UR^C_{ope}
- $\mathtt{addUser}^{\mathsf{C}}(u)$
  - Add $\langle \mathsf{U}, u, -, -, -, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to U^C_{inc}
- $\mathtt{initUser}^{\mathsf{C}}(u)$
  - Generate $(\mathsf{k}^{\mathsf{enc}}_{u}, \mathsf{k}^{\mathsf{dec}}_{u}) \leftarrow \mathsf{Gen}^{\mathsf{Pub}}$, $(\mathsf{k}^{\mathsf{ver}}_{u}, \mathsf{k}^{\mathsf{sig}}_{u}) \leftarrow \mathsf{Gen}^{\mathsf{Sig}}$ and $\mathsf{p}_u \leftarrow \mathsf{Gen}^{\mathsf{Pse}}$
  - Delete $\langle \mathsf{U}, u, -, -, -, * \rangle$ from U^C_{inc}
  - Add $\langle \mathsf{U}, u, \mathsf{p}_u, \mathsf{k}^{\mathsf{enc}}_{u}, \mathsf{k}^{\mathsf{ver}}_{u}, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{u}} \rangle$ to U^C_{ope}
- $\mathtt{deleteUser}^{\mathsf{C}}(u)$
  - Verify that $\nexists \langle \mathsf{UR}, u, *, *, *, * \rangle \in UR^C_{\mathsf{ope}}$
  - Move $\langle \mathsf{U}, u, *, *, *, * \rangle$ from $U^C_{\mathsf{inc}} \cup U^C_{\mathsf{ope}}$ to U^C_{hide}
- $\mathtt{addRole}^{\mathsf{C}}(r)$
  - Generate $(\mathsf{k}^{\mathsf{enc}}_{(r,1)}, \mathsf{k}^{\mathsf{dec}}_{(r,1)}) \leftarrow \mathsf{Gen}^{\mathsf{Pub}}$, $(\mathsf{k}^{\mathsf{ver}}_{(r,1)}, \mathsf{k}^{\mathsf{sig}}_{(r,1)}) \leftarrow \mathsf{Gen}^{\mathsf{Sig}}$ and $\mathsf{p}_{(r,1)} \leftarrow \mathsf{Gen}^{\mathsf{Pse}}$
  - Add $\langle \mathsf{R}, r, \mathsf{p}_{(r,1)}, \mathsf{k}^{\mathsf{enc}}_{(r,1)}, \mathsf{k}^{\mathsf{ver}}_{(r,1)}, 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to R^C_{ope}
  - Add $\langle \mathsf{UR}, adm, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{adm}}(\mathsf{k}^{\mathsf{enc}}_{(r,1)}, \mathsf{k}^{\mathsf{dec}}_{(r,1)}, \mathsf{k}^{\mathsf{ver}}_{(r,1)}, \mathsf{k}^{\mathsf{sig}}_{(r,1)}), 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to UR^C_{ope}
- $\mathtt{deleteRole}^{\mathsf{C}}(r)$
  - Verify that $\nexists \langle \mathsf{UR}, *, r, *, *, * \rangle \in UR^C_{\mathsf{ope}}$
  - Verify that $\langle \mathsf{PA}, r, *, *, *, *, *, *, *, * \rangle \notin PA^C_{\mathsf{ope}}$
  - Move $\langle \mathsf{R}, r, *, *, *, *, * \rangle$ from R^C_{ope} to R^C_{hide}
- $\mathtt{addResource}^{\mathsf{C}}_u(f)$
  - Generate $\mathsf{k}^{\mathsf{sym}}_{(f,1)} \leftarrow \mathsf{Gen}^{\mathsf{Sym}}$ and $\mathsf{p}_{(f,1)} \leftarrow \mathsf{Gen}^{\mathsf{Pse}}$
  - Send $\langle \mathsf{F}, f, \mathsf{p}_{(f,1)}, \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,1)}}(\mathsf{k}^{\mathsf{sym}}_{(f,1)}), 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{u}} \rangle$ and $\langle \mathsf{PA}, adm, f, adm, \mathsf{p}_{(f,1)}, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{adm}}(\mathsf{k}^{\mathsf{sym}}_{(f,1)}), 1, 1, OP, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{u}} \rangle$ to the administrator
  - The administrator adds received information to F^C_{ope} and PA^C_{ope}, respectively.
- $\mathtt{assignPermissionToRole}^{\mathsf{C}}(r, \langle ops, f \rangle)$
  - If $\exists \langle \mathsf{PA}, r, f, *, *, *, *, ops', *, * \rangle \in PA^C_{\mathsf{ope}}$:
    - Replace $\langle \mathsf{PA}, r, f, *, *, *, *, *, ops', * \rangle$ with $\langle \mathsf{PA}, r, f, *, *, *, *, *, ops \cup ops', \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ in PA^C_{ope}
  - Else:
    - Find $\langle \mathsf{R}, r, \mathsf{p}_{(r,v_r)}, \mathsf{k}^{\mathsf{enc}}_{(r,v_r)}, *, v_r, * \rangle \in R^C_{\mathsf{ope}}$
    - Find $\langle \mathsf{PA}, adm, f, *, \mathsf{p}_{(f,v_f)}, c, 1, v_f, *, * \rangle \in PA^C_{\mathsf{ope}}$
    - Decrypt $\mathsf{k}^{\mathsf{enc}}_{(r,v_r)} \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{adm}}(c)$
    - Add $\langle \mathsf{PA}, r, f, \mathsf{p}_{(r,v_r)}, \mathsf{p}_{(f,v_f)}, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{(r,v_r)}}(\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}), v_r, v_f, ops, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to PA^C_{ope}
- $\mathtt{revokePermissionFromRole}^{\mathsf{C}}(r, \langle ops, f \rangle)$
  - Find $\langle \mathsf{PA}, r, f, \mathsf{p}_{(r,v_r)}, \mathsf{p}_{(f,v_f)}, c, v_r, v_f, ops', * \rangle \in PA^C_{\mathsf{ope}}$
  - If $ops' \subseteq ops$:
    - Move $\langle \mathsf{PA}, r, f, *, *, *, *, *, *, * \rangle$ from PA^C_{ope} to UR^C_{hide}
  - Else:
    - Add $\langle \mathsf{PA}, r, f, \mathsf{p}_{(r,v_r)}, \mathsf{p}_{(f,v_f)}, c, v_r, v_f, ops \cap ops', \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to UR^C_{hide}
    - Replace $\langle \mathsf{PA}, r, f, *, *, *, *, *, *, * \rangle$ with $\langle \mathsf{PA}, r, f, *, *, *, *, *, ops' \setminus ops, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ in PA^C_{ope}
- $\mathtt{deleteResource}^{\mathsf{C}}(f)$
  - Verify that $\nexists \langle \mathsf{PA}, *, f, *, *, *, *, *, *, * \rangle \in PA^C_{\mathsf{ope}}$
  - Move all $\langle \mathsf{F}, f, *, *, *, * \rangle$ from F^C_{ope} to F^C_{del}
  - Invoke $\mathtt{cleanup}^{\mathsf{C}}()$
- $\mathtt{assignUserToRole}^{\mathsf{C}}(u, r)$
  - Find $\langle \mathsf{U}, u, *, \mathsf{k}^{\mathsf{enc}}_{u}, *, * \rangle \in U^C_{\mathsf{ope}}$
  - Find $\langle \mathsf{UR}, adm, r, c, v_r, * \rangle \in UR^C_{\mathsf{ope}}$
  - Decrypt $m \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{adm}}(c)$
  - Add $\langle \mathsf{UR}, u, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{u}}(m), v_r, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to UR^C_{ope}
- $\mathtt{rotateResourceKey}^{\mathsf{C}}(f, fc^{enc})$
  - Find $\langle \mathsf{PA}, adm, f, *, *, c_{v_f}, 1, v_f, *, * \rangle \in PA^C_{\mathsf{ope}}$
  - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,v_f)} \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{adm}}(c_{v_f})$
  - Generate $\mathsf{k}^{\mathsf{sym}}_{(f,v_f+1)} \leftarrow \mathsf{Gen}^{\mathsf{Sym}}$ and $\mathsf{p}_{(f,v_f+1)} \leftarrow \mathsf{Gen}^{\mathsf{Pse}}$
  - For each $\langle \mathsf{F}, f, *, c, \widehat{v_f}, * \rangle \in F^C_{\mathsf{ope}}$
    - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,\widehat{v_f})} \leftarrow \mathsf{Dec}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}}(c)$
    - Replace $\langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle$ with $\langle \mathsf{F}, f, *, \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f+1)}}(\mathsf{k}^{\mathsf{sym}}_{(f,\widehat{v_f})}), \widehat{v_f}, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ in F^C_{ope}
  - Add $\langle \mathsf{F}, f, \mathsf{p}_{(f,v_f+1)}, \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f+1)}}(\mathsf{k}^{\mathsf{sym}}_{(f,v_f+1)}), v_f + 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to F^C_{ope}
  - For each $\langle \mathsf{R}, r, *, \mathsf{k}^{\mathsf{enc}}_{(r,v_r)}, *, v_r, * \rangle \in R^C_{\mathsf{ope}}$ s.t. $\langle \mathsf{PA}, r, f, *, *, *, v_r, v_f, *, * \rangle \in PA^C_{\mathsf{ope}}$:
    - Move $\langle \mathsf{PA}, r, f, *, *, *, *, *, *, * \rangle$ from PA^C_{ope} to UR^C_{hide} and replace it with $\langle \mathsf{PA}, r, f, *, \mathsf{p}_{(f,v_f+1)}, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{(r,v_r)}}(\mathsf{k}^{\mathsf{sym}}_{(f,v_f+1)}), *, v_f + 1, *, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$
  - Move all $\langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle$ s.t. $\widehat{v_f} \leq v_f$ from F^C_{ope} to F^C_{del}
- $\mathtt{eagerReEncryption}^{\mathsf{C}}(f, fc^{enc})$
  - Set $fc \leftarrow \mathtt{readResource}^{\mathsf{C}}_{adm}(f, fc^{enc})$
  - Set $fc^{enc}_{new} \leftarrow \mathtt{writeResource}^{\mathsf{C}}_{adm}(f, fc)$
  - Return fc^{enc}_{new}
- $\mathtt{rotateRoleKeyPermissions}^{\mathsf{C}}(r)$
  - Find $\langle \mathsf{R}, r, \mathsf{p}_{(r,v_r)}, \mathsf{k}^{\mathsf{enc}}_{(r,v_r)}, *, v_f, * \rangle \in R^C_{\mathsf{ope}}$
  - For each $\langle \mathsf{PA}, r, f, *, \mathsf{p}_{(f,v_f)}, *, *, v_f, ops, * \rangle \in PA^C_{\mathsf{ope}}$:
    - Find $\langle \mathsf{PA}, adm, f, *, *, c, 1, v_f, *, * \rangle \in PA^C_{\mathsf{ope}}$
    - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,v_f)} \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{adm}}(c)$
    - Add $\langle \mathsf{PA}, r, f, \mathsf{p}_{(r,v_r)}, \mathsf{p}_{(f,v_f)}, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{(r,v_r)}}(\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}), v_r, v_f, ops, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to PA^C_{ope}
  - Move all $\langle \mathsf{PA}, r, f, *, *, *, \widehat{v_r}, *, *, * \rangle$ s.t. $\widehat{v_r} < v_r$ from PA^C_{ope} to UR^C_{hide}
- $\mathtt{revokeUserFromRole}^{\mathsf{C}}(u, r)$
  - Move $\langle \mathsf{UR}, u, r, *, *, * \rangle$ form UR^C_{ope} to UR^C_{hide}
- $\mathtt{rotateRoleKeyUserRole}^{\mathsf{C}}(r)$
  - Find $\langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}}$
  - Generate $(\mathsf{k}^{\mathsf{enc}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{dec}}_{(r,v_r+1)}) \leftarrow \mathsf{Gen}^{\mathsf{Pub}}$, $(\mathsf{k}^{\mathsf{ver}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{sig}}_{(r,v_r+1)}) \leftarrow \mathsf{Gen}^{\mathsf{Sig}}$ and $\mathsf{p}_{(r,v_r+1)} \leftarrow \mathsf{Gen}^{\mathsf{Pse}}$
  - Move $\langle \mathsf{R}, r, *, *, *, *, * \rangle$ from R^C_{ope} to R^C_{hide} and replace it with $\langle \mathsf{R}, r, \mathsf{p}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{enc}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{ver}}_{(r,v_r+1)}, v_r + 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$
  - For each $\langle \mathsf{UR}, u, r, *, v_r, * \rangle \in UR^C_{\mathsf{ope}}$:
    - Find $\langle \mathsf{U}, u, *, \mathsf{k}^{\mathsf{enc}}_{u}, *, * \rangle \in U^C_{\mathsf{ope}}$
    - Add $\langle \mathsf{UR}, u, r, \mathsf{Enc}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{enc}}_{u}}(\mathsf{k}^{\mathsf{enc}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{dec}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{ver}}_{(r,v_r+1)}, \mathsf{k}^{\mathsf{sig}}_{(r,v_r+1)}), v_r + 1, \mathsf{Sign}^{\mathsf{Sig}}_{\mathsf{k}^{\mathsf{sig}}_{adm}} \rangle$ to UR^C_{ope}
  - Move all $\langle \mathsf{UR}, *, r, *, v_r, * \rangle$ from UR^C_{ope} to UR^C_{hide}
- $\mathtt{readResource}^{\mathsf{C}}_u(f, fc^{enc})$
  - If $\nexists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}}$ s.t. $\exists \langle \mathsf{UR}, u, r, c_r, v_r, * \rangle \in UR^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{PA}, r, f, *, *, c_f, v_r, v_f, ops, * \rangle \in PA^C_{\mathsf{ope}} \wedge read \in ops$:
    - Return $\perp$
  - Decrypt $(-, \mathsf{k}^{\mathsf{dec}}_{(r,v_r)}, -, -) \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{u}}(c_r)$
  - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,v_f)} \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{(r,v_r)}}(c_f)$
  - If $\exists \langle \mathsf{F}, f, *, \widehat{c_f}, \widehat{v_f}, * \rangle \in F^C_{\mathsf{ope}}$ s.t. $\widehat{v_r} \neq v_r$:
    - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,\widehat{v_f})} \leftarrow \mathsf{Dec}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}}(\widehat{c_f})$
    - Set $fc \leftarrow \mathsf{Dec}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,\widehat{v_f})}}(fc^{enc})$
  - Else:
    - Set $fc \leftarrow \mathsf{Dec}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}}(fc^{enc})$
  - Return fc
- $\mathtt{writeResource}^{\mathsf{C}}_u(f, fc)$
  - If $\nexists \langle \mathsf{R}, r, *, *, *, v_r, * \rangle \in R^C_{\mathsf{ope}}$ s.t. $\exists \langle \mathsf{UR}, u, r, c_r, v_r, * \rangle \in UR^C_{\mathsf{ope}} \wedge \exists \langle \mathsf{PA}, r, f, *, *, c_f, v_r, v_f, ops, * \rangle \in PA^C_{\mathsf{ope}} \wedge write \in ops$:
    - Return $\perp$
  - Move $\langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle$ s.t. $\widehat{v_f} < v_f$ from F^C_{ope} to F^C_{del}
  - Invoke $\mathtt{cleanup}^{\mathsf{C}}()$
  - Decrypt $(-, \mathsf{k}^{\mathsf{dec}}_{(r,v_r)}, -, -) \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{u}}(c_r)$
  - Decrypt $\mathsf{k}^{\mathsf{sym}}_{(f,v_f)} \leftarrow \mathsf{Dec}^{\mathsf{Pub}}_{\mathsf{k}^{\mathsf{dec}}_{(r,v_r)}}(c_f)$
  - Encrypt $fc^{enc} \leftarrow \mathsf{Enc}^{\mathsf{Sym}}_{\mathsf{k}^{\mathsf{sym}}_{(f,v_f)}}(fc)$
  - Return fc^{enc}
- $\mathtt{cleanup}^{\mathsf{C}}()$
  - Move all $\langle \mathsf{PA}, *, f, *, *, *, *, \widehat{v_f}, *, * \rangle$ s.t. $\nexists \langle \mathsf{F}, f, *, *, \widehat{v_f}, * \rangle \in PA^C_{\mathsf{ope}}$ from F^C_{hide} to F^C_{del}
  - Move all $\langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle$ s.t. $\nexists \langle \mathsf{PA}, r, *, *, *, *, \widehat{v_r}, *, *, * \rangle \in UR^C_{\mathsf{hide}}$ from R^C_{hide} to R^C_{del}
  - Move all $\langle \mathsf{UR}, *, r, *, \widehat{v_r}, * \rangle$ s.t. $\nexists \langle \mathsf{R}, r, *, *, *, \widehat{v_r}, * \rangle \in (R^C_{\mathsf{ope}} \cup R^C_{\mathsf{hide}})$ from UR^C_{hide} to UR^C_{del}
  - Move all $\langle \mathsf{U}, u, *, *, *, * \rangle$ s.t. $\nexists \langle \mathsf{UR}, u, *, *, *, * \rangle \in UR^C_{\mathsf{hide}}$ from R^C_{hide} to UR^C_{del}