Cryptographic Role-Based Access Control, Reconsidered *

Bin Liu[1], Antonis Michalas[1,2], and Bogdan Warinschi[3,4]

[1] Tampere University
[2] University of Westminster
[3] DFINITY
[4] University of Bristol
{bin.liu,antonios.michalas}@tuni.fi
csxbw@bristol.ac.uk

Abstract. A significant shortcoming of traditional access control mechanisms is their heavy reliance on reference monitors. Being single points of failure, monitors need to run in protected mode and have permanent online presence in order to handle all access requests. Cryptographic access control offers an alternative solution that provides better scalability and deployability. It relies on security guarantees of the underlying cryptographic primitives and the appropriate key distribution/management in the system. In order to rigorously study security guarantees that a cryptographic access control system can achieve, providing formal security definitions for the system is of great importance, since the security guarantee of the underlying cryptographic primitives cannot be directly translated into those of the system.

In this paper, we follow the line of the existing studies on the cryptographic enforcement of Role-Based Access Control (RBAC). Inspired by the study focusing on the relation between the existing security definitions for such systems, we identify two types of attacks not described in the existing works. Therefore, we propose two new security definitions with the goal of appropriately modeling cryptographic enforcement of Role-Based Access Control policies and studying the relation between our new definitions and the existing ones. In addition, we show that the cost of supporting dynamic policy updates is inherently expensive by presenting two lower bounds for such systems that guarantee correctness and secure access.

1 Introduction

Traditional access control mechanisms heavily rely on reference monitors, meaning single points of failure operating under protected mode. Though online access control mechanisms with a permanent online presence are necessary in order to serve all access requests by users, this approach must be critically re-examined. The mechanism's

* This work was partially funded by the HARPOCRATES project, Horizon Europe and the Technology Innovation Institute (TII), Abu Dhabi, United Arab Emirates, for the project ARROWSMITH: Living (Securely) on the edge.

inherent limitations greatly affect its scalability and deployability. This issue can be addressed with the help of cryptographic techniques that implement access control policies with the help of cryptographic primitives. This alternative approach is referred to as cryptographic access control. It aims at reducing or even eliminating monitor reliance, as the cryptographic access control policy is achieved in an indirect way: the data is protected by cryptographic primitives and the policies are enforced by distributing appropriate keys to appropriate users.

A main concern in the existing cryptographic access control studies is the gap between the specifications of the enforced access control policies and the actual implementation of the access control systems. In traditional monitor-based access control mechanisms, the correct enforcement of access control policies holds by design. However, in cryptographic access control, the issue becomes more complex. The enforcement relies on security guarantees by underlying cryptographic primitives as well as on the appropriate key distribution/management. Even though some advanced cryptographic primitives are seemingly well-suited for cryptographic access control, their security guarantees cannot be directly translated to security guarantees for the entire system. It is common understanding that there is often a gap between primitives and the applications motivating them. The gap is obscured by uses of similar terms and jargon at both the application and the primitive level. It is unfortunate that in the case of primitives, the security of the motivating application is often omitted.

Formal security definitions for cryptographic access control systems are of crucial importance in order to bridge this gap. However, they are often neglected in the existing research on cryptographic access control. There have been some initial works in this area that focus on new primitives motivated by access control systems [7, 2, 13] and on access control systems based on said primitives [9, 12, 14, 6].

Throughout the literature, rigorous definitions that examine the security of systems for access control have only been heuristically studied. In the aim of reasoning about the confinement problem, Halevi et al. proposed a simulation-based security definition for access control on a distributed file storage system [5]. Their result concerns a particular system rather than a general one. Ferrara et al. defined a precise syntax for cryptographic role-based access control (cRBAC) systems and proposed a formal security definition with respect to secure read access in [4]. Following, they extend their results in a setting that supports write access [3], their goal being to reduce the need for trusted monitors mediation for every write access request. Liu et. al. studied security of cRBAC systems in the UC framework [10]. They proposed a UC security definition for such systems and showed that this type of security presents an impossibility result due to the commitment problem.

Garrison III et al. studied the practical implications of cryptographic access control systems implementing RBAC policies [8]. They analysed the computational costs of two different constructions of cryptographic role-based access control systems via simulations with the use of real-world datasets. Their results indicate that supporting a dynamic access control policy enforcement may prove prohibitively expensive, even under the assumption that write access is enforced with the minimum use of reference monitors.

NEW SECURITY DEFINITIONS. The results presented in [10] show a gap between game-based and simulation-based security definitions for cRBAC systems, which raises the following question:

Do the existing security definitions appropriately capture the secure enforcement of access control policies?

The first security definition is called *past confidentiality* and refers to security concerns by users, who acquire unauthorised read access to the previous file versions. It serves as a refinement to the existing definition of read security for cRBAC systems. In traditional monitor-based access control, when a user acquires access to the authorised file, it only acquires access to the current content. By "previous contents" we refer to previous file versions written in the past and not considered a part of the file's current content. In cryptographic access control, due to the file system's public accessibility, users can easily obtain previous file versions (even in an encrypted form) by simply monitoring the state of the file system. Therefore, a user recently granted the read permission of a file might retrieve previous contents written at a time, when it did not have access permission - this can be considered as a violation of the implemented access control policy.

Game-based security definitions in the existing work do not appropriately capture the security concern mentioned above [4, 3]. One should keep in mind that in games, where security is defined with respect to read access, the adversary is not allowed read access to challenge files at any point during the game. This restriction on the adversary cannot rule out the attack mentioned above. In fact, the attack can easily be carried out in the constructions proposed in [4, 3]. Interestingly, some recently proposed constructions of cryptographic access control systems pose a similar security concern [1, 8, 11], though they have been proven to securely enforce the corresponding access control policies within their individual frameworks.

LOWER BOUNDS FOR SECURE CRBAC SYSTEMS. Garrison III et al. studied the practical implications of using cryptography to enforce RBAC policies in their recent work [8]. They considered a system model making the necessary use of reference monitors to enforce access control on write access and to maintain the metadata of each file in the file system. For this purpose, they developed two different constructions of cryptographic RBAC systems: one is developed with identity-based encryption (IBE) and identity-based signature (IBS) schemes, while the other one is based on traditional public key cryptography and makes use of public key infrastructure (PKI). In order to analyse construction costs they carried out the simulation over real-world RBAC datasets to generate traces. Their experimental results show that even with the minimum use of reference monitors, the computational costs of the cryptographic RBAC systems supporting the idea of a dynamic policy update are still prohibitively expensive.

Motivated by Garrison III et al.'s work, we study lower bounds for secure cRBAC systems to locate the source of the inefficiency. We show that the costs are inherent in secure cRBAC systems and also in those cryptographic access control systems that greatly or solely rely on cryptographic techniques to enforce access control on both read and write access. The main idea is, since the manager is not involved in any read and

write operation in the file system, both the users' local states and the file system should reflect the enforcement of the access control policy. Whenever a policy update occurs, the system may inevitably require re-keying and re-encryption to preserve secure access and system correctness. We present two lower bounds for secure cRBAC systems. Our results can be valuable in the design of such systems for practical purposes.

2 Preliminaries

2.1 Notations

For assignment, we write $x \leftarrow y$ to denote the assignment of the value y to the variable x. If S is a set, $x \leftarrow\!\!\$\ S$ denotes that x is being assigned with a value selected uniformly at random from S. Let $\mathcal{A}$ be an algorithm, $x \leftarrow \mathcal{A}(y)$ denotes the assignment of x with the output of running it on the input y, if $\mathcal{A}$ is deterministic, while we write $x \leftarrow\!\!\$\ \mathcal{A}(y)$ for the assignment, if $\mathcal{A}$ is probabilistic.

For any integer $n \geq 0$, we write 1^n to denote the string of n $1s$. If S is a set, $|S|$ denotes its size. If s is a string, $|s|$ denotes its length. Given two strings s_0 and s_1, $s_0 \| s_1$ denotes their concatenation. ϵ denotes the empty string. $\perp$ denotes an error, though its meaning depends on the context: it could indicate a decryption error or an error returned by an oracle due to an invalid oracle query.

We say f is a *negligible function*, if for every positive polynomial p, there exists an integer N such that for all integers $n > N$, it holds that $f(n) < \frac{1}{p(n)}$.

2.2 Role-Based Access Control

Role-Based Access Control (RBAC) is one of the most popular access control models adopted in large-scale systems. RBAC introduces the significant concept of *roles*, which are typically associated to a collection of job functions. Roles allow for specifying access control policies that naturally map the organisation structures and therefore reduce complexity when administering permissions. RBAC policies are decomposed into two assignments: the user-role assignment and the permission-role assignment. A user is authorised to a permission, if a user role has been assigned with the permission. In this paper, we will only focus on core RBAC, as it is the most common between the standard models.

The state of a (core) RBAC system consists of:

- U: a finite set of users,
- R: a finite set of roles,
- O: a finite set of objects,
- P: a finite set of permissions, where each permission is an *object-operation* pair,
- $UA \subseteq U \times R$: a relation modelling the user-role assignment,
- $PA \subseteq P \times R$: a relation modelling the permission-role assignment,

For simplicity we assume that the set of roles R is fixed, since role structures in organisations are less frequent to change. Therefore, the state of an RBAC system over a fixed role set R is a tuple (U, O, P, UA, PA). We describe an RBAC system in terms

of a state-transition system. We define the set of state-transition rules RULES as the RBAC administrative commands. Given two states $S = (U, O, P, UA, PA)$ and $S' = (U', O', P', PA', UA')$, there is a *transition* from S to S' with $q \in$ RULES denotes $S \xrightarrow{q}_{\mathcal{S}} S'$, if one of the following conditions holds:

- **[AddUser]** $q = (\mathsf{AddUser}, u)$, $u \notin U$, $U' = U \cup \{u\}$, $O' = O$, $P' = P$, $PA' = PA$ and $UA' = UA$;
- **[DelUser]** $q = (\mathsf{DelUser}, u)$, $u \in U$, $U' = U \setminus \{u\}$, $O' = O$, $P' = P$, $PA' = PA$ and $UA' = UA \setminus \{(u, r) \in UA \parallel r \in R)\}$;
- **[AddObject]** $q = (\mathsf{AddObject}, o)$, $o \notin O$, $O' = O \cup \{o\}$, $U' = U$, $P' = P \cup \{(o, \mathtt{read}), (o, \mathtt{write})\}$, $PA' = PA$ and $UA' = UA$;
- **[DelOject]** $q = (\mathsf{DelObject}, o)$, $o \in O$, $O' = O \setminus \{o\}$, $U' = U$, $P' = P \setminus \{(o, \cdot)\}$, $PA' = PA \setminus \{((o, \cdot), r) \in PA \parallel r \in R)\}$ and $UA' = UA$;
- **[AssignUser]** $q = (\mathsf{AssignUser}, (u, r))$, $u \in U$, $r \in R$, $U' = U$, $O' = O$, $P' = P$, $PA' = PA$ and $UA' = UA \cup \{(u, r)\}$;
- **[DeassignUser]** $q = (\mathsf{DeassignUser}, (u, r))$, $u \in U$, $r \in R$, $U' = U$, $O' = O$, $P' = P$, $PA' = PA$ and $UA' = UA \setminus \{(u, r)\}$;
- **[GrantPerm]** $q = (\mathsf{GrantPerm}, (p, r))$, $p \in P$, $r \in R$, $U' = U$, $O' = O$, $P' = P$, $PA' = PA \cup \{(p, r)\}$ and $UA' = UA$;
- **[RevokePerm]** $q = (\mathsf{RevokePerm}, (p, r))$, $p \in P$, $r \in R$, $U' = U$, $O' = O$, $P' = P$, $PA' = PA \setminus \{(p, r)\}$ and $UA' = UA$.

An execution of an RBAC system is a finite sequence of transitions $S_0 \xrightarrow{q_0}_{\mathcal{S}} S_1 \xrightarrow{q_1}_{\mathcal{S}} \ldots \xrightarrow{q_n}_{\mathcal{S}} S_{n+1}$, where S_0 is called the *initial* state of the RBAC system.

We denote the *read* permission and the *write* permission of a file $o \in O$ by $(o, \mathtt{read})$ and $(o, \mathtt{write})$ respectively. A predicate $\mathsf{HasAccess}(u, p)$ reflects that a user u has symbolical access to a permission p. It is defined as follows:

$$\mathsf{HasAccess}(u, p) \Leftrightarrow \exists r \in R : (u, r) \in UA \wedge (p, r) \in PA$$

2.3 System Model and Syntax

The system model we consider in this paper is the one proposed by Ferrara et al. in [3]. In their system model, a versioning append-only file system is employed for enforcing access control on (quasi-) unrestricted read and write access to the files. Hence, the need for online monitors is eliminated.

It should be noted that using a file system of this kind, does not consist a limitation. In contrast, it allows for modelling a *general* class of access control systems. Since the file system itself does not implement any access control mechanisms and the enforcement of access control policies on files is handled using cryptography solely, it can be used to apprehend data outsourcing scenarios, where hosting trusted reference monitors is impossible. Users may even be able to keep track of files (e.g. storing data on public clouds, repositories, decentralised distributed storage networks, etc.).

We consider a cRBAC system as consisting of three main entities: a *manager*, a *file system* and a set of *users*.

The manager is responsible for the administration of access control policies. More specifically, it is in charge of executing RBAC administrative commands that could involve key management/distribution and data encryption/re-encryption. In contrast to the traditional access control policy enforcer (i.e. the reference monitor) that has to be placed in the critical path to check whether an access request is considered compliant to the policy, the manager is not related to any read and write access to the file system. In addition, the manager is assumed to be a trusted party.

The *file system* is tasked with storing the files subjected to access control and it is publicly accessible to users. In the implementation phase, the system could contain arrays of encrypted files and the related metadata. The file system is assumed to be untrusted in terms of data privacy, but it guarantees the availability of the data it stores. Consider that if users are provided unrestricted write access to the file system, no amount of cryptographic techniques can prevent a malicious user from overwriting the existing contents. Therefore, the file system is further assumed to be append-only and supporting versioning. In this case, users can only append contents, but cannot delete any. The append operations can be interpreted as logical writes to the files. When reading a file, a user first needs to fetch the file versions and then identifies the most recently "validated" one to retrieve the content. As the data owner, the manager may have richer interfaces to the file system than users. Therefore, it can overwrite the file contents and add/delete files.

Users may have (quasi-) direct and unrestricted read and write access to the file system without involving the manager. Since the enforcement of access control policies solely relies on cryptographic primitives and the file system does not implement any access control functionality, only the users with appropriate keys may get authorised access to the files.

Secure channels are assumed between each of any two entities. For simplicity, the execution of any RBAC administrative command is assumed to be implemented by non-interactive multi-party computations. That is, when executing any RBAC command, the manager carries out a local computation in accordance to the 'update messages' command for users and potentially updates the file system. Following, update messages are sent to the users via secure channels. Once a user receives said message from the manager, it updates its local state accordingly. The file system proceeds with the update in a similar manner.

The global state of a cRBAC system at any point during its execution is a tuple $(st_M, fs, \{st_u\}_{u \in U})$, where st_M is the local state of the manager, fs is the state of the file system and st_u is the local state for each user $u \in U$ in the system. Since the manager is tasked with the access control policy administration, the symbolic RBAC state $S = (U, O, P, UA, PA)$ is considered to be a part of st_M. Let $\phi(st_G)$ denote the RBAC state of the global state st_G.

A cRBAC system is defined by a cRBAC scheme consisting in the following algorithms:

– Init, the **initialisation** algorithm: A probabilistic algorithm that takes the security parameter 1^λ and a set of roles R as input and outputs the initialised global state of the cRBAC system.

– AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, the **RBAC administrative** algorithms: Probabilistic algorithms that implement the corresponding RBAC administrative commands. As mentioned, they are non-interactive multi-party computations. Each of the algorithms takes the state of the manager st_M, the state of the file system fs and the additional argument for the command arg as input and outputs the updated state for the manager and the file system as well as a set of update messages $\{msg_u\}_{u \in U}$ for each user $u \in U$.
– Read, the **read** algorithm: A deterministic algorithm that allows a user to retrieve the current content of a file. It takes the local state of a user st_u, the current state of the file system fs and a file name o as input and outputs the current content of the file o, if the user has the read permission. If not or if the file is empty, the algorithm returns an error $\perp$.
– Write, the **write** algorithm: A probabilistic algorithm that allows a user to write content to a file. It takes the local state of a user st_u, the current state of the file system fs, a file name o and the content m as input and outputs the updated file system.
– Update, the **update** algorithm: A deterministic algorithm that takes the local state of a user st_u and an update message msg_u received from the manager and outputs the updated local state.

A comment should be made about the updated file system, forming the output part of some of the algorithms outlined above. More specifically, the algorithms produce update instructions to be carried out on the file system. For example, after running the Write algorithm, a user will get the update instruction $info$ that includes the information of the file name and also the content to be appended to the file system. Then the user uploads $info$ to the file system and the latter gets updated accordingly. The manager proceeds similarly, but the update instructions might be different from that of the users due to the data owner privilege. For simplicity, we will let those algorithms output the updated file system. In terms of effect, all the above algorithms except Read can potentially update the global state of the cRBAC system. Therefore, we may write the execution of a cRBAC algorithm in the following form:

$$st_G \xrightarrow{Q} st'_G \Leftrightarrow st'_G \leftarrow\!\!\$\ A(st_G, arg),$$

where A is one of the algorithms defined above (except for Read), arg is its arguments, Q is an implementation of the algorithm, st_G and st'_G are the global states of the cRBAC system.

Let $\vec{Q} = (Q_0, ..., Q_n)$. We write the execution trace of the cRBAC system as:

$$st_{G_0} \xrightarrow{\vec{Q}} st_{G_{n+1}} \Leftrightarrow st_{G_0} \xrightarrow{Q_0} st_{G_1} \xrightarrow{Q_1}, ..., \xrightarrow{Q_{n-1}} st_{G_n} \xrightarrow{Q_n} st_{G_{n+1}},$$

where $\{st_{G_i}\}_{i \in \{0,...,n+1\}}$ are global states of the cRBAC system.

We say a sequence of operations is *efficient*, if the length of its execution trace is polynomially bounded.

We also introduce the following two notations $\mathbb{P}_r$ and $\mathbb{O}_r$. $\mathbb{P}_r$ is the set of objects the read permissions of which a user has been *"symbolically"* assigned with at a certain

point. Let $\mathcal{S} = (U, O, P, UA, PA)$ be the RBAC state of a system:

$$\mathbb{P}_r(\mathcal{S}, u) \Leftrightarrow \{o | \mathsf{HasAccess}(u, (o, \mathtt{read}))\}.$$

$\mathbb{Q}_r$ is the set of objects to which a user has *"computational"* read access, i.e. the objects whose contents can be retrieved by performing the read operations with the user's local state. Consider that some file may be empty (i.e. it contains no content) after the initialisation, while a user may be granted the read permission of that file. Therefore, we define $\mathbb{Q}_r$ as the set of objects. If any user u' with write permissions writes content to them at this point, u will be able to read it. Let st_G be the global state of a cRBAC system. $\mathbb{Q}_r$ is defined as:

$$\begin{aligned}\mathbb{Q}_r(st_G, u) \Leftrightarrow \{o |\ \forall u' \in U, m \in \{0,1\}^\lambda : \mathsf{HasAccess}(u', (o, \mathtt{write})) \wedge \\ (fs' \leftarrow\!\!\$\ \mathsf{Write}(st_{u'}, fs, o, m), m' \leftarrow \mathsf{Read}(st_u, fs', o) : m' = m)\}\end{aligned}$$

3 Security Definitions

In this section, we present our formal security definitions of correctness, past confidentiality and local correctness for cRBAC systems.

3.1 Correctness

Correctness was first proposed by Ferrara et al. in [4], but it was omitted in their later work [3], where a new system model was introduced to support write access. Therefore, we will need to reintroduce the definition of correctness.

Intuitively, a cryptographic access control system is said to be *correct*, if every user in the system can acquire access to the resources to which it is authorised according to the symbolic state of the system. In a cRBAC system enforcing access control on both read and write access to a publicly accessible file system, the correctness requirements are specialised as follows:

1. any user that has the read permission for a file should be able to retrieve the current content of the file by reading it, and
2. the current content of a file written by a user with a write permission will be correctly read by any user with a read permission for the file.

We formalise the two requirements above via a game between a challenger, who acts as a manager of a cRBAC system defined by cRBAC scheme Π and a polynomial-time adversary $\mathcal{A}$ that attacks the system. The adversary is allowed to request the manager to execute any RBAC administrative command, such that the symbolic state of the system evolves according to its queries. The adversary can also request a user to write to the file system and to query the current state of the file system. At some point of the game, $\mathcal{A}$ needs to show that there exists some user, who cannot correctly retrieve the current content of a file to which it has read access.

More specifically, we define the following experiment $\mathbf{Exp}^{\text{corr}}_{\Pi,\mathcal{A}}$. The experiment maintains the symbolic RBAC state of the system $State$, which is set to be $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

initially, and an object-indexed list T to record the contents written to the files by authorised users.

After the initialisation of the cRBAC system with a fixed set of roles R, the adversary can ask for the execution of any RBAC administrative command by calling the oracle CMD. Upon receiving a query that consists of an RBAC command Cmd and its arguments arg, the oracle will execute the command symbolically and run the algorithm Cmd, that implements the command. Then the adversary will be provided the current state of the file system fs as a response. The oracle WRITE allows the adversary to request a user u to write some content m to a specified file o. If u has the write permission of o , the oracle runs the write algorithm Write to carry out the write operation and then sets $T[o] \leftarrow m$. In addition, the adversary can check the current state of the file system by calling the oracle FS with the query "STATE".

In this experiment, the adversary is not allowed to take over any user in the system nor update the file system on its own. At some point, $\mathcal{A}$ outputs a user-object pair (u^*, o^*) and the experiment terminates here. In the case that u^* has the read permission of o^* , but the content that it can retrieve from o^* by running the read algorithm Read does not match the record in $T[o^*]$, the adversary wins the game. Correctness is defined by requiring that any probabilistic polynomial-time adversary does not win the above game with a probability greater than 0.

Definition 1 (Correctness). *A cRBAC system Π defined by a cRBAC scheme for a fixed set of roles. R is **correct** if for any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{corr}_{\Pi,\mathcal{A}}(\lambda) := \Pr\left[\mathbf{Exp}^{corr}_{\Pi,\mathcal{A}}(\lambda) \rightarrow \texttt{true}\right]$$

is 0, where the experiment $\mathbf{Exp}^{corr}_{\Pi,\mathcal{A}}$ is defined as follows:

$\underline{\mathbf{Exp}^{corr}_{\Pi,\mathcal{A}}(\lambda)}$

$T \leftarrow \emptyset$; $State \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
$(st_M, fs, \{st_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R)$
$(u^*, o^*) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda : \mathcal{O}_{corr})$
if $\mathsf{HasAccess}(u^*, (o^*, \texttt{read})) \wedge T[o^*] \neq \mathsf{Read}(st_{u^*}, o^*, fs)$ **then**
  return `true`
else return `false`

The oracles $\mathcal{O}_{corr}$ that the adversary has access to are specified in Figure 1.

3.2 Past confidentiality

In the extended cRBAC system model, the enforcement of access control on write access is supported by employing a versioning file storage, where users can append contents only. The versioning file storage allows users to have (quasi-) unrestricted read and write access to the file system, but it is also accompanied by subtle security issues, though the file system itself does not implement any access control mechanism. One of the security issues relates to unauthorised access to the previous contents. This is a severe security concern in cryptographic access control, but not when it comes to

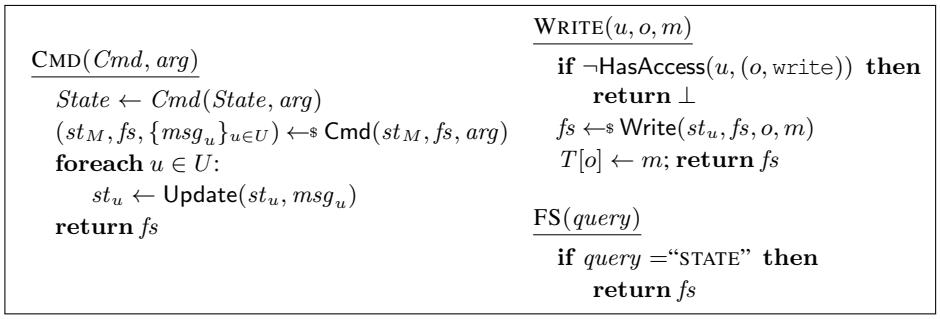

Fig. 1. $\mathcal{O}_{corr}$: Oracles for defining the experiment $\mathbf{Exp}^{\text{corr}}_{\Pi,\mathcal{A}}$.

traditional mechanisms. Unfortunately, the existing game-based security definitions for secure read access do not suffice to capture this security concern. We propose a security definition called *past confidentiality*, which is an improved version of the one presented in [3].

The security property is formalised via the following experiment $\mathbf{Exp}^{\text{pc}}_{\Pi,\mathcal{A}}$, which proceeds similarly to the experiment that defines read security in [3]. However, the adversary here is allowed to corrupt the users, who have the read permission for the challenge files under certain conditions. More specifically, the adversary is not allowed to corrupt any user in a position to read the challenge contents (rather than the challenge files in the read security game), until said challenge contents no longer exist in the file.The adversary's goal is to still determine a random bit $b \leftarrow\$ \{0, 1\}$ selected at the beginning of the game.

The experiment maintains the symbolic RBAC state of the system $State$ initialised as $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ and updated according to $\mathcal{A}$'s request for the execution of RBAC commands. The experiment maintains the following lists during the execution: Cr for the corrupt users, Ch for the files with contents specified as challenges, L for the users who have read access to challenge contents and Ud for the files whose current contents are specified as challenges.

In the experiment, the adversary can request any RBAC administrative command for execution, taking over users and requesting an honest user to write to a file with the content it specifies. The adversary can query the current state of the file system and also write (append) new content to it. $\mathcal{A}$ can ask for a challenge by specifying a tuple (u, o, m_0, m_1), where u is a user that has the write permission of the file o, and m_0 and m_1 are two messages of the same length. Then the challenger will carry out $\mathsf{Write}(st_u, o, m_b)$ and provide the current state of the file system to the adversary as the response. $\mathcal{A}$ can ask for multiple challenges. When $\mathcal{A}$ terminates with an output b', it wins the game if $b' = b$.

To prevent the adversary from winning the game trivially by corrupting a user, with-read access to the challenge contents, the experiment maintains the following invariants:

1. No user in Cr can have read access to any file o in Ud: the adversary is not allowed to request read permission of any file from a corrupt user, if the file's current content is specified as a challenge.

2. No user in the list L can be corrupted: any user with direct access to the challenge contents cannot be taken over by the adversary.

Definition 2 (Past Confidentiality). *A cRBAC system Π defined by a cRBAC scheme for a fixed set of roles R is said to preserve **past confidentiality**, when for any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{pc}_{\Pi,\mathcal{A}}(\lambda) := \left| \Pr[\mathbf{Exp}^{pc}_{\Pi,\mathcal{A}}(\lambda) \to \mathtt{true}] - \tfrac{1}{2} \right|$$

is negligible in λ, where the experiment $\mathbf{Exp}^{pc}_{\Pi,\mathcal{A}}$ is defined as follows:

$\underline{\mathbf{Exp}^{pc}_{\Pi,\mathcal{A}}(\lambda)}$
$b \leftarrow\!\!\$\ \{0,1\}$; $Cr, Ch, L, Ud \leftarrow \emptyset$
$State \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
$(st_M, fs, \{st_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R)$
$b' \leftarrow\!\!\$\ \mathcal{A}(1^\lambda : \mathcal{O}_{pc})$
$\mathbf{return}\ (b' = b)$

The oracles $\mathcal{O}_{pc}$ that the adversary has access to are specified in Figure 2 and discussed below.

The oracle CMD allows the adversary to request for the execution of any valid RBAC command. When $\mathcal{A}$'s query leads to an update to Cr, Ch, L or Ud, the lists get updated accordingly. When a user in L loses the read permission of any file in Ch, the file will be removed from the list L. When $\mathcal{A}$ requests to grant an honest user with a read permission for the files in Ud, the user will be added to L.

When the adversary requests from an honest user to write content to a file whose current content is specified as a challenge, the file will be removed from the list Ud. From then on, the file's read permission can be granted to a corrupt user. When $\mathcal{A}$ requests to place a challenge by calling the oracle CHALLENGE, the oracle returns an error, if there exists some corrupt user that has read access to the specified file. Otherwise, it carries out the write operation and adds the file to lists Ch and Ud.

Compared to the adversary in the game that defines the read security of a cRBAC system, the adversary in the past confidentiality game is obviously more powerful, as it has the ability to take over users, who can acquire read access to the challenged files under certain restrictions. The following theorem confirms the implication between the two security definitions.

Theorem 1. *Past confidentiality is strictly stronger than secure read access.*

Proof sketch. First, we briefly show that any cRBAC system preserving past confidentiality is secure in terms of read access. Given any adversary $\mathcal{A}$ attacking a cRBAC system with respect to read security, an adversary $\mathcal{B}$ for past confidentiality can be easily constructed. After the initialisation of the cRBAC system in its own game, $\mathcal{B}$ runs a local copy of $\mathcal{A}$ with the input of the file system's initial state received from the challenger. Then $\mathcal{B}$ starts to simulate the read security game for $\mathcal{A}$ with the use of the oracles it has access to. During the simulation, $\mathcal{B}$ does not maintain the global state of the cRBAC

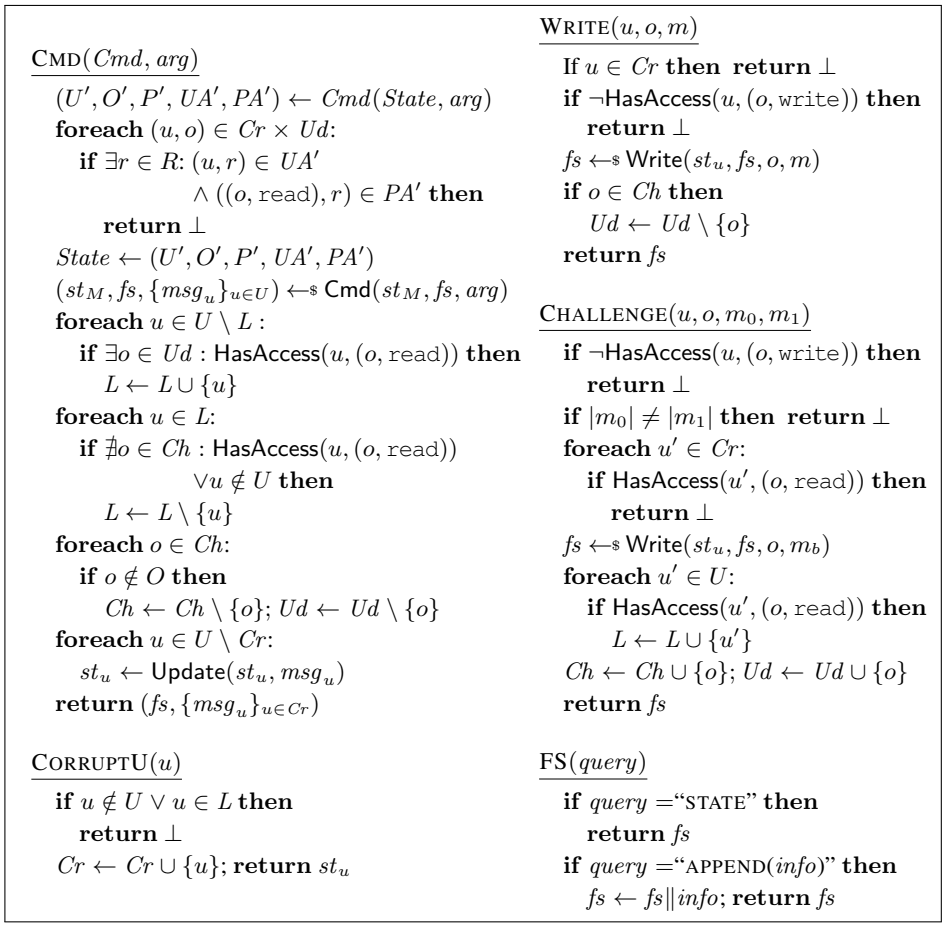

Fig. 2. $\mathcal{O}_{pc}$: Oracles for defining the experiment $\mathbf{Exp}^{\text{pc}}_{\mathcal{CRBAC},\mathcal{A}}$.

system. It only keeps the two lists: Cr for corrupt users and Ch for the challenge files, as defined in the read security game. Following, $\mathcal{B}$ simply forwards any query received from $\mathcal{A}$, to the same oracles in its game and replies $\mathcal{A}$ with the response it received. If $\mathcal{A}$'s query violates the restrictions of the read security game (i.e. by granting any user in Cr the read permission for the files in Ch), $\mathcal{B}$ responds with an error and ignores the query. Whenever $\mathcal{A}$ outputs a guess as to the random bit, $\mathcal{B}$ outputs the same bit in its game.

We will now argue that the simulation $\mathcal{B}$ provides is perfect. First, the global states in both $\mathcal{B}$'s game and the simulated game are identical. Secondly, not all of $\mathcal{A}$'s oracle queries will violate the restrictions of the game that defines past confidentiality, since queries from $\mathcal{A}$ not violating the invariant in the read security game will not violate the invariants in the past confidentiality game. Moreover, the simulated game fully depends on the random bit chosen in $\mathcal{B}$'s game, thus $\mathcal{B}$ wins its game with the same probability as

$\mathcal{A}$ wins the simulated game. Therefore, any cRBAC system not secure with read access does not preserve past confidentiality.

In addition, the construction of cRBAC scheme presented in [3] has been proven to be secure with respect to read accessThough it clearly does not preserve past confidentiality, because granting the read permission of any file to a user will allow it to access the previous contents encrypted under the same public key. Therefore, one may conclude that past confidentiality is strictly stronger than secure read access. □

3.3 Local Correctness

The *local correctness* of a cRBAC system can be considered as a sort of correctness, though it is not implied by correctness. It captures the threat from "insiders" with respect to data availability. The append-only versioning file system allows users to get (quasi-) unrestricted write access to the files, but it also poses a new security concern: a user who has the write permission of a file might be able to invalidate the file's future versions written by authorised users. Local correctness guarantees that these systems thwart.

This security requirement is formalised via the following experiment $\mathbf{Exp}^{\text{l-corr}}_{\Pi,\mathcal{A}}$ that involves an adversary $\mathcal{A}$. The experiment maintains a list Cr to record the corrupt users and another object-indexed list T to record the contents written to files by the honest users. After the initialisation of the cRBAC system, the adversary can request the execution of any RBAC administrative command, taking over any user and writing content to a file on behalf of any honest user. $\mathcal{A}$ can also query for the current state of the file system and request to append arbitrary content to it.

Here, the list T serves to record whether the files have been unauthorised touched (rather than authorised write access) or not. When an honest user writes content to a file o, this content is recorded in $T[o]$. If the adversary requests to update the file by appending any entry to it, $T[o]$ will store a special value `adv`. This means the file has been touched after the previously authorised write access.

The experiment terminates when the adversary outputs auser-object pair (u^*, o^*), where u^* has the read permission of o^*. $\mathcal{A}$ wins the game, if the content of o^* read by u^* is different from the record in $T[o^*]$, while $T[o^*]$ cannot be the special value `adv`.

Definition 3 (Local Correctness). *A cRBAC system Π defined by a cRBAC scheme for a fixed set of roles R is said to preserve **local correctness**, if for any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\textit{l-corr}}_{\Pi,\mathcal{A}}(\lambda) := \Pr\left[\mathbf{Exp}^{\textit{l-corr}}_{\Pi,\mathcal{A}}(\lambda) \to \texttt{true}\right]$$

is negligible in λ, where $\mathbf{Exp}^{\textit{l-corr}}_{\Pi,\mathcal{A}}$ is defined as follows:

$\underline{\mathbf{Exp}^{\textit{l-corr}}_{\Pi,\mathcal{A}}(\lambda)}$

$T, Cr \leftarrow \emptyset$; $State \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
$(st_M, fs, \{st_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R)$
$(u^*, o^*) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda : \mathcal{O}_{\textit{l-corr}})$
if $T[o^*] \neq \texttt{adv} \wedge T[o^*] \neq \mathsf{Read}(st_{u^*}, o^*, fs)$ **then**
  return `true`
else return `false`

The oracles $\mathcal{O}_{\text{l-corr}}$ that the adversary has access to are specified in Figure 3.

$\underline{\textsc{Cmd}(Cmd, arg)}$

$State \leftarrow Cmd(State, arg)$
$(st_M, fs, \{msg_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Cmd}(st_M, fs, arg)$
foreach $u \in Cr$:
  if $u \notin U$ **then**
    $Cr \leftarrow Cr \setminus \{u\}$
if $Cmd =$ "DELOBJECT" **then**
  Parse arg as o; $T[o] \leftarrow \emptyset$
if $Cmd =$ "DELUSER" **then**
  Parse arg as u; $Cr \leftarrow Cr \setminus \{u\}$
foreach $u \in U \setminus Cr$:
  $st_u \leftarrow \mathsf{Update}(st_u, msg_u)$
return $(fs, \{st_u\}_{u \in Cr})$

$\underline{\textsc{CorruptU}(u)}$

if $u \notin U$ **then return** $\perp$
$Cr \leftarrow Cr \cup \{u\}$; **return** st_u

$\underline{\textsc{Write}(u, o, m)}$

if $u \in Cr$ **then return** $\perp$
if $\neg\mathsf{HasAccess}(u, (o, \texttt{write}))$ **then**
  return $\perp$
$fs \leftarrow\!\!\$\ \mathsf{Write}(st_u, fs, o, m)$
$T[o] \leftarrow m$; **return** fs

$\underline{\textsc{FS}(query)}$

if $query =$"STATE" **then**
  return fs
if $query =$"APPEND($info$)" **then**
  Parse $info$ as (o, c)
  $T[o] \leftarrow \texttt{adv}$; $fs \leftarrow fs \| info$
  return fs

Fig. 3. $\mathcal{O}_{\textit{l-corr}}$: Oracles for defining the experiment $\mathbf{Exp}^{\textit{l-corr}}_{\Pi,\mathcal{A}}$.

We further show that the cRBAC construction proposed by Ferrara et al. in [3] preserves this security property.

Theorem 2. *If both the predicate encryption scheme and the digital signature scheme are correct, the construction in [3] preserves local correctness.*

Proof sketch. We will show that in this cRBAC construction, no matter how the adversary touches a file, any content written by an authorised user will be correctly retrieved by another user, with the read permission of the file.

The specification of their write algorithm Write, shows us that the algorithm will come up with a new entry to be appended to the file. The content of the new entry is completely independent from any of the previous entries and it only depends on the next available index of the file versions and also the metadata stored in the header of

said file. Since the file system is assumed to preserve the file's correct ordering and the metadata can only be updated by the manager, these two parameters will not be affected by potential corrupt user behaviours to the file system. In case, a different option would be that the predicate encryption scheme or the signature scheme are not correct, therefore the authorised user cannot correctly retrieve the content written to the target file. Thus, we can conclude that the construction preserves local correctness under the assumption that both the predicate encryption scheme and the digital signature are correct. □

4 Lower Bounds for secure cRBAC systems

In this section, we present two lower bounds for secure cRBAC systems. By lower bounds, we refer to the efficiency implications of secure cRBAC systems. To some extent, our results explain why cRBAC systems supporting dynamic policy updates may be prohibitively expensive: permission revocation can be costly. Prior to presenting our results, we will introduce a technical term called *Permission Adjustment* for an RBAC system. Informally, permission adjustment is a sequence of administrative commands changing the access rights of some users with respect to a set of permissions. In comparison with any sequence of typical RBAC administrative commands that might not bring any change to the access matrix of the system, permission adjustment emphasises the *change* it will bring to the access matrix. The term can be better understood using the following example. Consider that a user that has been deassigned from a reviewer role still has access to the conference papers, as the user may also serve as a programme committee member. This allows the user to get authorised access. However, for the permission adjustment of cancelling its access to the papers, the user will no longer be able to do so due to the access control policy change.

Definition 4. *(Permission Adjustment) Let $S_0 = (U, O, P, UA, PA)$ be the state of an RBAC system over a set of roles R. Given a set of users $\tilde{U} \subseteq U$ and a set of permissions $\tilde{P} \subseteq P$, where both $\tilde{U}$ and $\tilde{P}$ are non-empty, a sequence of RBAC administrative commands $\vec{q} = (q_0, ..., q_n)$ is called a **permission adjustment** for S_0 with respect to $\tilde{U}$ and $\tilde{P}$:*

(1) *if $\forall u \in \tilde{U}, p \in \tilde{P} : \neg\mathsf{HasAccess}(u, p)$ holds for S_0 and after a sequence of transitions $S_0 \xrightarrow{q_0}_{\mathcal{S}} S_1 \xrightarrow{q_1}_{\mathcal{S}}, \ldots, \xrightarrow{q_{n-1}}_{\mathcal{S}} S_n \xrightarrow{q_n}_{\mathcal{S}} S_{n+1}$, $\forall u \in \tilde{U}, p \in \tilde{P} :$ $\mathsf{HasAccess}(u, p)$ holds for S_{n+1} or*

(2) *if $\forall u \in \tilde{U}, p \in \tilde{P} : \mathsf{HasAccess}(u, p)$ holds for S_0 and after a sequence of transitions $S_0 \xrightarrow{q_0}_{\mathcal{S}} S_1 \xrightarrow{q_1}_{\mathcal{S}}, \ldots, \xrightarrow{q_{n-1}}_{\mathcal{S}} S_n \xrightarrow{q_n}_{\mathcal{S}} S_{n+1}$, $\forall u \in \tilde{U}, p \in \tilde{P} : \neg\mathsf{HasAccess}(u, p)$ holds for S_{n+1}.*

We denote the set of all possible $\vec{q}$ in case (1) by $\tilde{U}\uparrow\tilde{P}(S_0)$ and the set of all possible $\vec{q}$ in case (2) by $\tilde{U}\downarrow\tilde{P}(S_0)$.

In addition, we introduce two key properties with respect to efficiency.

Definition 5. *Let $st_G = (st_M, fs, \{st_u\}_{u \in U})$ be the global state of a cRBAC system over a set of roles R at some point during its execution. Given a sequence of RBAC*

administrative commands $\vec{q} = (q_0, ..., q_n)$ and a sequence of efficient operations $\vec{Q} = (Q_0, ..., Q_n)$ such that for each $i \in \{0, ..., n\}$: Q_i implements the command q_i. After carrying out $\vec{Q}$:

(1) *if the state of the file system remains unchanged, we say that $\vec{q}$ is **file system preserving** for st_G. It is reflected by the following predicate:*

$$\mathsf{FSP}(\vec{q}, st_G) \Leftrightarrow \Pr[\forall \vec{Q} : st_G \xrightarrow{\vec{Q}} st'_G; fs = fs'] = 1,$$

where $st'_G = (st'_M, fs', \{st'_u\}_{u \in U'})$ and $\phi(st'_G) = (U', O', P', UA', PA')$;

(2) *if the local states of a set of users $\mathbb{U}$ remain unchanged, we say that $\vec{q}$ is $\mathbb{U}$**-user local state preserving** for st_G. It is reflected by the following predicate:*

$$\mathsf{LSP}(\vec{q}, st_G, \mathbb{U}) \Leftrightarrow \Pr[\forall \vec{Q} : st_G \xrightarrow{\vec{Q}} st'_G; \forall u \in \mathbb{U} : st_u = st'_u] = 1,$$

where $st'_G = (st'_M, fs', \{st'_u\}_{u \in U'})$, $\phi(st'_G) = (U', O', P', UA', PA')$ and $\mathbb{U} \subseteq U'$.

Finally, we introduce the concept of *non-trivial execution* for a cRBAC system. A non-trivial execution consists of a sequence of operations such that after executing every operation in order for each file in the system, there should exist at least one user with the read permission for it and at least a user with the write permission for it. The non-trivial execution serves as a mild assumption on the execution of a cRBAC system, for the purpose of studying the lower bound of cRBAC systems commonly used in practice. Also, non-trivial execution can prevent trivial implementations of a cRBAC system. For example, in a cRBAC system with users authorised to read but not write to the file system, there is no need to worry about unauthorised read access, while no content will be written to the file system. The same holds in the case of write security.

Before introducing the lower bounds, we present the following auxiliary results.

Lemma 1. *For any correct cRBAC system, it holds that:*

$$\Pr[st_G \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R); st_G \xrightarrow{\vec{Q}} st'_G; \forall u \in U : \mathbb{P}_r(\phi(st'_G), u) \subseteq \mathbb{Q}_r(st'_G, u)] = 1,$$

where $\vec{Q}$ an efficient non-trivial execution and $\phi(st'_G) = (U, O, P, UA, PA)$.

Proof. Assume that for a cRBAC system Π, the probability that after carrying out the non-trivial execution $\vec{Q}$, the system will reach some global state st'_G such that $\mathbb{P}_r(\phi(st'_G), u) \subseteq \mathbb{Q}_r(st'_G, u)$ holds for every user $u \in U$ is $\epsilon < 1$. We show that Π cannot be correct in this case.

Consider the following adversary $\mathcal{A}$ for $\mathbf{Exp}^{\text{corr}}_{\Pi,\mathcal{A}}(\lambda)$. After the system gets initialised, $\mathcal{A}$ is provided λ and then calls the corresponding oracles to carry out $\vec{Q}$ in order. Let st'_G be the current global state of the system. Now $\mathcal{A}$ makes a random guess of a tuple $(u, u', o) \in U \times U \times O$ and also comes up with a message $m \in \{0,1\}^\lambda$. It

then calls the write oracle WRITE with (u', o, m). If WRITE returns an error, $\mathcal{A}$ only terminates here; otherwise $\mathcal{A}$ terminates with a (u, o) output. Since $\vec{Q}$ is a finite sequence of opertions, $\mathcal{A}$ is an polynomial-time adversary.

By assumption, the probability of a user $\bar{u} \in U$ and a file $\bar{o} \in O$ satisfying both $o \in \mathbb{P}_r(\phi(st'_G), \bar{u})$ and $o \notin \mathbb{Q}_r(st'_G, \bar{u})$ is $1 - \epsilon$. Since $\vec{Q}$ is a non-trivial execution, the condition $o \notin \mathbb{Q}_r(st'_G, \bar{u})$ further implies that there exists a user $\bar{u}' \in U$ has the write permission of $\bar{o}$ such that, if, by this point, the message m is written to $\bar{o}$ by $\bar{u}'$ it will not be retrieved correctly by running Read with $\bar{u}$'s local state. Therefore, if $\mathcal{A}$ made a good guess regarding $\bar{u}, \bar{o}, \bar{u}'$, the challenger will not be able to retrieve m. The advantage that $\mathcal{A}$ can gain in the experiment is:

$$\mathbf{Adv}^{\text{corr}}_{\Pi,\mathcal{A}}(\lambda) \geq \frac{1}{|U| \cdot |U| \cdot |O|},$$

which is obviously non-zero. Therefore, Π cannot be correct. □

Lemma 2. *If a cRBAC system is secure with respect to read access, it holds that:*

$$\Pr[st_G \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R); st_G \xrightarrow{\vec{Q}} st'_G; \forall u \in U : \mathbb{P}_r(\phi(st'_G), u) \supseteq \mathbb{Q}_r(st'_G, u)] \geq 1 - \epsilon,$$

where $\vec{Q}$ is an efficient non-trivial execution, $\phi(st'_G) = (U, O, P, UA, PA)$ and ϵ is a negligible function in λ.

Proof. Assume that for a cRBAC system Π, the probability that after carrying out the non-trivial execution $\vec{Q}$, it will reach some global state st'_G such that for all ϵ, $\mathbb{P}_r(\phi(st'_G), u) \supseteq \mathbb{Q}_r(st'_G, u)$ holds for all $u \in U$ with probability $\epsilon_0 < 1 - \epsilon$. We show that Π cannot be secure with respect to read access.

Consider the following adversary $\mathcal{A}$ for $\mathbf{Exp}^{\text{read}}_{\Pi,\mathcal{A}}(\lambda)$. After being provided the security parameter λ, $\mathcal{A}$ makes queries to the corresponding oracles to carry out $\vec{Q}$ in order. Let st'_G be the current global state of Π. Now $\mathcal{A}$ randomly chooses a user $u \in U$, a file $o \in O$ and another user $u' \in \{u | \mathsf{HasAccess}(u, (o, \mathtt{write}))\}$. It then requests to corrupt u to obtain the local state st_u. Next, $\mathcal{A}$ calls the challenge oracle CHALLENGE with (u', o, m_0, m_1), where $m_0, m_1 \in \{0,1\}^\lambda$ are two random messages of the same length. If CHALLENGE returns an error, $\mathcal{A}$ terminates here and outputs a random bit. If CHALLENGE returns the updated state of the file system *fs*, $\mathcal{A}$ then computes $m^* \leftarrow \mathsf{Read}(st_u, fs, o)$. Finally, it outputs 0 if $m^* = m_0$ and 1 if $m' = m_1$; otherwise, it only outputs a random bit. Since $\vec{Q}$ is efficient, $\mathcal{A}$ is an polynomial-time adversary.

It is possible that u has the read permission of o, because $\mathcal{A}$ chose u and o randomly from all existing users and files in the system. Therefore, the request of taking over u might lead to an error returned by CHALLENGE. Notice that $\mathcal{A}$ can read the content written to the file specified as its challenge only when there exists a user $\bar{u} \in U$ and $\bar{o} \in O$ satisfying both $\bar{o} \notin \mathbb{P}_r(\phi(st'_G), \bar{u})$ and $\bar{o} \in \mathbb{Q}_r(st'_G, \bar{u})$, and $\mathcal{A}$ has made a good guess of them. In all the other cases, $\mathcal{A}$ has to output a random bit. By assumption, such $\bar{u}$ and $\bar{o}$ exist with probability $1 - \epsilon_0$. Then, the advantage that $\mathcal{A}$ can gain in the

experiment is:

$$\begin{aligned}\mathbf{Adv}^{\text{read}}_{\Pi,\mathcal{A}}(\lambda) \geq & \big|(1-\epsilon_0)\cdot\frac{1}{|U|\cdot|O|}+(1-\epsilon_0)\cdot(1-\frac{1}{|U|\cdot|O|})\cdot\frac{1}{2}+\epsilon_0\cdot\frac{1}{2}-\frac{1}{2}\big| \\ = & \big|\frac{1}{2}\cdot\frac{1}{|U|\cdot|O|}-\epsilon_0\cdot\frac{1}{2}\cdot\frac{1}{|U|\cdot|O|}\big| \\ = & \frac{1}{2}\cdot\frac{1}{|U|\cdot|O|}\cdot(1-\epsilon_0) > \frac{1}{2}\cdot\frac{1}{|U|\cdot|O|}\cdot\epsilon\end{aligned}$$

which is non-negligible. Hence Π cannot be secure with respect to read access. □

Following, we present our first lower bound for cRBAC systems that are both correct and secure with respect to read access.

Theorem 3. *For any cRBAC system which is **correct** and **secure with respect to read access**, it holds that:*

$$\Pr\left[st_G \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R); st_G \xrightarrow{\vec{Q}} st'_G; \forall \vec{q} \in U_r{\downarrow}P_r(\phi(st'_G)) : \mathsf{FSP}(\vec{q}, st'_G) \wedge \mathsf{LSP}(\vec{q}, st'_G, U_w)\right] \leq \epsilon,$$

where $\vec{Q}$ is any non-trivial execution for the system, $st'_G = (st'_M, fs', \{st'_u\}_{u\in U'})$, $\phi(st'_G) = (U', O', P', UA', PA')$, $U_r \subseteq U'$, $P_r \subseteq \{(o, \texttt{read})|o \in O'\}$, $U_w = \{u|\forall(o, \texttt{read}) \in P_r : \mathsf{HasAccess}(u, (o, \texttt{write}))\}$ and ϵ is a negligible function in λ.

Proof. We prove the theorem by showing that, if the above condition is not satisfied, the cRBAC system cannot be both correct and secure with respect to read access. Assume by contradiction that, if there exists a cRBAC system Π, which is correct and read secure, the above condition holds with probability ϵ_0, which is greater than any ϵ.

Consider the following attack against the system. After Π is initialised with the role set R, the non-trivial execution $\vec{Q}$ is carried out. Let st'_G be the current global state. Since the cRBAC system is correct, for all users $u \in U_r$, $\mathbb{P}_r(\phi(st'_G), u) \subseteq \mathbb{Q}_r(st'_G, u)$ should hold with probability 1 according to Lemma 1. We let a user $u_0 \in U_r$ keep its current local state st_{u_0} and refuse update in future, which means u_0 will ignore all the update messages sent from the manager as of this point. After the sequence of RBAC administrative commands in $\vec{q}$ is carried out, the global state becomes st''_G. Now, by assumption, $\mathsf{FSP}(\vec{q}, st'_G)$ and $\mathsf{LSP}(\vec{q}, st'_G, U_w)$ holds with non-negligible probability ϵ_0. In the case that both properties are satisfied simultaneously, the state of the file system and the local states of the users in U_w remain the same. This implies that $\mathbb{Q}_r(st'_G, u) = \mathbb{Q}_r(st''_G, u)$, but $\mathbb{P}_r(\phi(st''_G), u) \leftarrow \mathbb{P}_r(\phi(st'_G), u) \setminus P_r$ with overviewming probability, leading to a violation of the Lemma 2. Therefore, we can conclude that the cRBAC system cannot be both correct and secure with respect to read access.

We have the following lower bound for cRBAC systems preserving both correctness and write security. The proof of it can be carried out in a similar manner. But it is obtained via a weaken security definition of secure write access, where the manager carries out the read operation to the specified file instead of the user chosen by the adversary.

Theorem 4. *For any cRBAC system that is* ***correct*** *and* ***secure with respect to write access****, it holds that:*

$$\Pr\left[st_G \leftarrow_{\$} \mathsf{Init}(1^\lambda, R); st_G \xrightarrow{\vec{Q}} st'_G; \forall \vec{q} \in U_w \downarrow P_w(\phi(st'_G)) : \mathsf{FSP}(\vec{q}, st'_G) \wedge \mathsf{LSP}(\vec{q}, st'_G, U_r)\right] \leq \epsilon,$$

where $\vec{Q}$ *is any non-trivial execution for the system,* $st'_G = (st'_M, fs', \{st'_u\}_{u \in U'})$, $\phi(st'_G) = (U', O', P', UA', PA')$, $U_w \subseteq U'$, $P_w \subseteq \{(o, \texttt{write}) | o \in O'\}$, $U_r = \{u | \forall (o, \texttt{write}) \in P_r : \mathsf{HasAccess}(u, (o, \texttt{read}))\}$ *and* ϵ *is a negligible function in* λ*.*

5 Conclusion

We proposed two new formal security definitions for cRBAC systems in a game-based setting. The first one is called past confidentiality and it captures the security concern of unauthorised access to the previous versions of the files. We show that this security definition is strictly stronger than the existing one regarding secure read access. The other security definition we proposed is called local correctness and captures the security concern from the insider of the system, which might undermine data availability. We presented two lower bounds for secure cRBAC systems that explain the inefficient aspects ofsystems that support permission revocation.

References

1. James Alderman, Jason Crampton, and Naomi Farley. A framework for the cryptographic enforcement of information flow policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2017, Indianapolis, IN, USA, June 21-23, 2017*, pages 143–154, 2017.
2. Michael Clear, Arthur Hughes, and Hitesh Tewari. Homomorphic encryption with access policies: Characterization and new constructions. In *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, pages 61–87, 2013.
3. Anna Lisa Ferrara, Georg Fuchsbauer, Bin Liu, and Bogdan Warinschi. Policy privacy in cryptographic access control. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 46–60, 2015.
4. Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced RBAC. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 115–129, 2013.
5. Shai Halevi, Paul A. Karger, and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. *IACR Cryptology ePrint Archive*, 2005:169, 2005.
6. Jie Huang, Mohamed A. Sharaf, and Chin-Tser Huang. A hierarchical framework for secure and scalable EHR sharing and access control in multi-cloud. In *41st International Conference on Parallel Processing Workshops, ICPPW 2012, Pittsburgh, PA, USA, September 10-13, 2012*, pages 279–287, 2012.
7. Luan Ibraimi. Cryptographically enforced distributed data access control. *University of Twente*, 2011.

8. William C. Garrison III, Adam Shull, Adam J. Lee, and Steven Myers. Dynamic and private cryptographic access control for untrusted clouds: Costs and constructions (extended version). *CoRR*, abs/1602.09069, 2016.
9. Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, Hong Kong, China, March 22-24, 2011*, pages 411–415, 2011.
10. Bin Liu and Bogdan Warinschi. Universally composable cryptographic role-based access control. In Liqun Chen and Jinguang Han, editors, *Provable Security - 10th International Conference, ProvSec 2016, Nanjing, China, November 10-11, 2016, Proceedings*, volume 10005 of *Lecture Notes in Computer Science*, pages 61–80, 2016.
11. Saiyu Qi and Yuanqing Zheng. Crypt-dac: Cryptographically enforced dynamic access control in the cloud. *IEEE Trans. Dependable Secur. Comput.*, 18(2):765–779, 2021.
12. Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 735–737, 2010.
13. Stefan G Weber. Designing a hybrid attribute-based encryption scheme supporting dynamic attributes. *IACR Cryptology ePrint Archive*, 2013:219, 2013.
14. Yan Zhu, Gail-Joon Ahn, Hongxin Hu, and Huaixi Wang. Cryptographic role-based security mechanisms based on role-key hierarchy. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13-16, 2010*, pages 314–319, 2010.

A Security definitions of cRBAC schemes in [3]

A.1 Secure Read Access

A cRBAC system is said to be secure with respect to read accesses, if no user can deduce any partial content of a file without a read permission. It is formalised via the following experiment $\mathbf{Exp}^{\text{read}}_{\Pi,\mathcal{A}}$, which involves a challenger who plays the role of the manager of a cRBAC system and an adversary $\mathcal{A}$. During the game, an adversary with no read permission to a file, can choose to write two messages. Then one of the two messages will be written to that file and $\mathcal{A}$'s goal is to determine which message that is.

Definition 6 (Secure Read Access). *A cRBAC system Π defined by the cRBAC scheme* (Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, Read, Write, Update) *is **secure with respect to read access**, if for any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{read}_{\Pi,\mathcal{A}}(\lambda) := \left| \Pr[\mathbf{Exp}^{read}_{\Pi,\mathcal{A}}(\lambda) \rightarrow \mathtt{true}] - \tfrac{1}{2} \right|$$

is negligible in λ, where $\mathbf{Exp}^{read}_{\Pi,\mathcal{A}}$ is defined as follows:

$\underline{\mathbf{Exp}^{read}_{\Pi,\mathcal{A}}(\lambda)}$
$b \leftarrow\!\!\$\ \{0,1\}$; $Cr, Ch \leftarrow \emptyset$
$State \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
$(st_M, fs, \{st_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R)$
$b' \leftarrow\!\!\$\ \mathcal{A}(1^\lambda : \mathcal{O}_{read})$
return $(b' = b)$

The oracles $\mathcal{O}_{read}$ that the adversary has access to are specified in Figure 4.

A.2 Secure Write Access

A cRBAC system is said to be secure with respect to write access, if no user can write content to a file without permission. Particularly, in the case of an open-access file system, the content wrote by an unauthorised user should not be considered as valid. It is formalised by the following experiment $\mathbf{Exp}^{\text{write}}_{\Pi,\mathcal{A}}$. The adversary needs to specify a target file with an honest user. It wins the game, if it can manage to write any valid content (read by the honest user), without the help of any authorised user. To prevent trivial wins, no corrupt user can acquire write access to the target file, from the point when the last write operation to the target file is carried out by an honest user with a permission until $\mathcal{A}$ generates its output.

Definition 7. *A cRBAC system Π defined by the cRBAC scheme* (Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, Read, Write, Update) *is **secure with respect to write access** if for any probabilistic polynomial-time adversaries $\mathcal{A}$, we have*

$$\mathbf{Adv}^{write}_{\Pi,\mathcal{A}}(\lambda) := \Pr\left[\mathbf{Exp}^{write}_{\Pi,\mathcal{A}}(\lambda) \rightarrow \mathtt{true}\right]$$

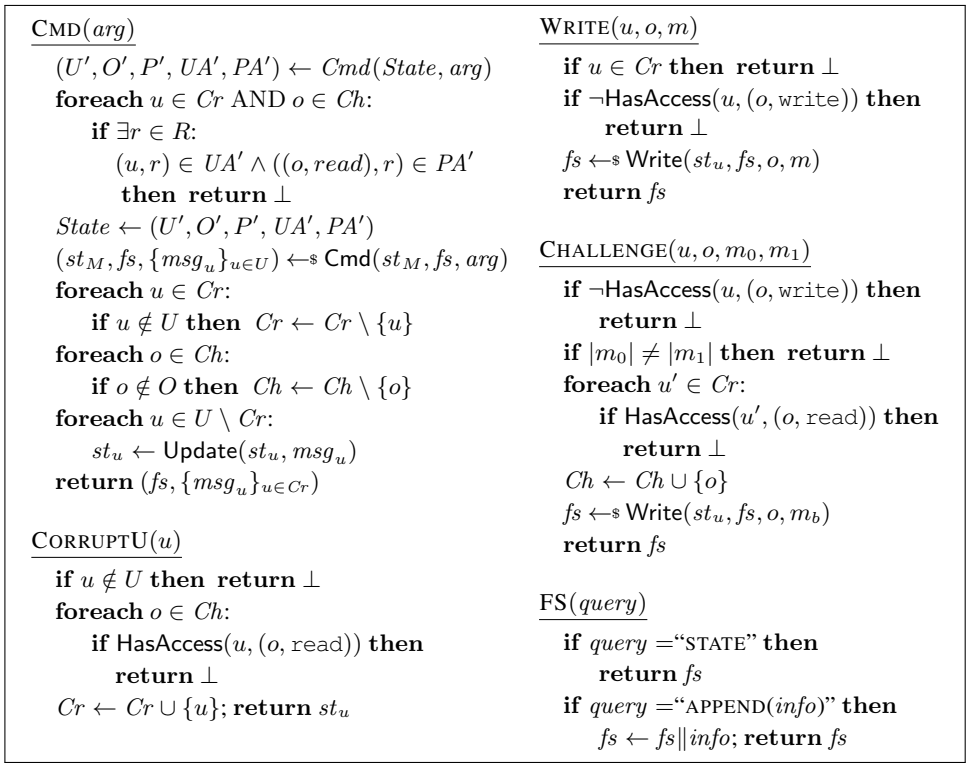

Fig. 4. $\mathcal{O}_{read}$: Oracles for defining the experiment $\mathbf{Exp}^{\text{read}}_{\Pi,\mathcal{A}}$.

is negligible in λ, where $\mathbf{Exp}^{write}_{\Pi,\mathcal{A}}$ is defined as follows:

$\underline{\mathbf{Exp}^{write}_{\Pi,\mathcal{A}}(\lambda)}$
$Cr, T \leftarrow \emptyset$
$State \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
$(st_M, fs, \{st_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, R)$
$o^* \leftarrow\!\!\$\ \mathcal{A}(1^\lambda : \mathcal{O}_{write})$
if $T[o^*] \neq \texttt{adv} \wedge T[o^*] \neq \mathsf{Read}(st_M, fs, o^*)$ **then**
return `true`
else return `false`

The oracles $\mathcal{O}_{write}$ that the adversary has access to are specified in Figure 5.

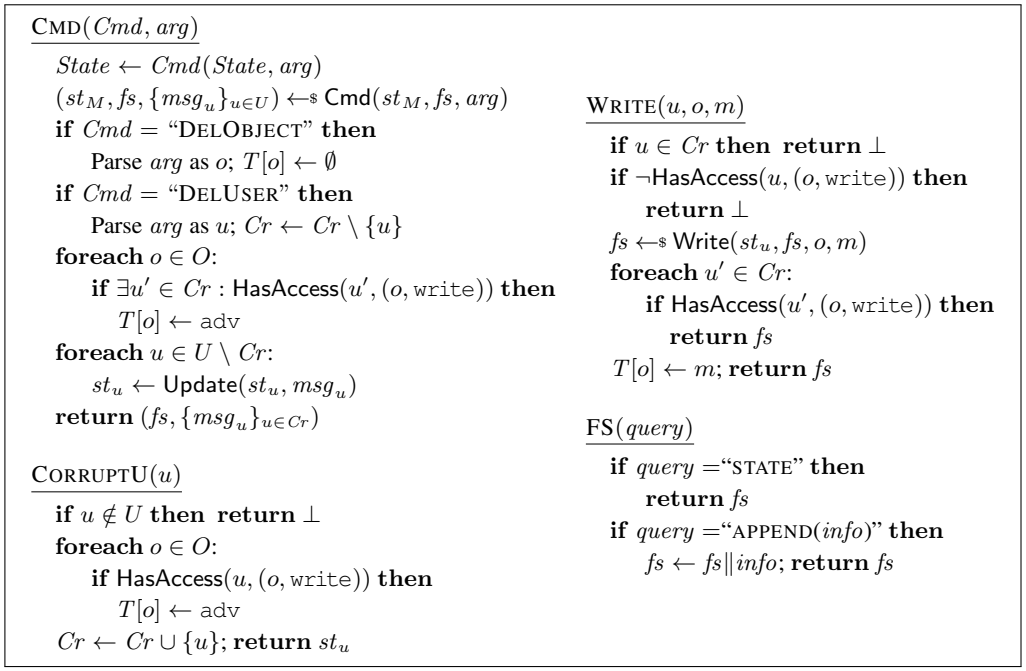

Fig. 5. $\mathcal{O}_{write}$: Oracles for defining the experiment $\mathbf{Exp}^{\text{write}}_{\Pi,\mathcal{A}}$.