Universally Composable Cryptographic Role-Based Access Control

Bin Liu and Bogdan Warinschi

University of Bristol, UK
`bin.liu@bristol.ac.uk` `bogdan@cs.bris.ac.uk`

Abstract. In cryptographic access control sensitive data is protected by cryptographic primitives and the desired access structure is enforced through appropriate management of the secret keys. In this paper we study rigorous security definitions for the cryptographic enforcement of Role Based Access Control (RBAC). We propose the first *simulation-based* security definition within the framework of Universal Composability (UC). Our definition is natural and intuitively appealing, so we expect that our approach would carry over to other access models.
Next, we establish two results that clarify the strength of our definition when compared with existing ones that use the game-based definitional approach. On the positive side, we demonstrate that both read and write-access guarantees in the sense of game-based security are implied by UC security of an access control system. Perhaps expected, this result serves as confirmation that the definition we propose is sound.
Our main technical result is a proof that simulation-based security requires impractical assumptions on the encryption scheme that is employed. As in other simulation-based settings, the source of inefficiency is the well known "commitment problem" which naturally occurs in the context of cryptographic access control to file systems.

Keywords: Universal Composability, cRBAC, game-based security

1 Introduction

Access control is one of the cornerstones of computer security. It comprises mechanisms and techniques that ensure that subjects (users, processes, etc) get access only to the objects (files, memory locations, etc) in a way that preserves the privacy/integrity of the objects per some access policy that is in place. Traditional access control mechanisms rely on reference monitors. Since monitors need to be permanently on-line and have to be executed in trust domains outside the control of the data owner(s), this has limitations that directly affect scalability and deployability of applications.

A solution to this problem employs cryptography and is based on a simple and elegant idea: protect the objects using cryptographic primitives (i.e. encryption to guarantee privacy and signatures for integrity) and then enforce the desired security policies by providing the right secret keys to the right parties. This type of implementation eliminates the need for an on-line monitor: the objects being protected can be made publicly available in encrypted form and access is only provided to the users that have the right secret keys.

Security models for cryptographic access control. Much of the prior work in this area was concerned with designing access control systems from basic cryptographic primitives [18, 15, 2, 11, 10, 9, 8] and/or designing new primitives tailored for the problem of access control [17, 23, 24, 14]. For the most part, the security of cryptographic access control systems was only heuristically studied. Yet, precise definitions are particularly important in this area: recent constructions employ complex cryptographic primitives for which the level of security is not always easy to ascertain and for which it is important to understand how they fit within the higher level systems that employ them. For instance, in attribute based encryption there is a sizable gap between security against adversaries that decide statically the keys they will attack and those that take this decision adaptively. Which of these types of schemes should be used in access control systems to ensure security requires rigorous definitions of what such systems aim to achieve and proofs that a given security level suffices.

Throughout the literature, rigorous models that look at the security of systems for access control have only sporadically been developed and were usually concerned with particular schemes or applications [19, 1, 7, 6]. Security models for broader frameworks have only recently been developed [13, 3]. One line of work in this direction is due to Ferrara et al. who consider cryptographic enforcement of Role-Based Access Control models [13, 12]. Specifically, they define a general syntax for cryptographic RBAC schemes (cRBAC in short), propose a security model that captures privacy guarantees for objects protected with such a system, and suggest an implementation based on predicate-encryption (PE).

The models proposed by Ferrara et al. use the so-called *game-based* approach. Here, the model formalizes the interaction between an adversary and the system and rigorously clarifies what is a security breach, e.g. as an event that occurs during the execution. The appeal of this approach is its relative simplicity: executions consider stand-alone scenarios where the system is in complete isolation from other systems and the different security goals (e.g. privacy and integrity of sensitive data) are treated independently from one another. At the same time, simplicity is also cause of some concern. Since the security games must specify precisely the information that an adversary can obtain when attacking the scheme, threats from arbitrary environments may not always be appropriately captured. Similarly, individual treatment of security properties may overlook unwanted interaction since oftentimes security properties are contradictory. Furthermore, it may not always be possible to exhaustively enumerate the different properties that one may desire from a system.

In this paper we consider a definitional alternative that does not suffer from the above shortcomings. Under this paradigm, called *simulation-based* approach, security is defined by comparing a system with an idealized version and demands that the real execution of a system reveals at most as much information is revealed by an ideal version of the system. As a consequence of this definition, the real system inherits *all* of the security properties of the ideal one, so there is no need to enumerate security properties separately. One important class of simulation-based security considers executions determined by an arbitrary environment (tasked, e.g. to provide inputs to the system), so security in this sense is *composable* in the sense that it is preserved in any environment in which the system is employed [4, 20, 22]. Unfortunately, simulation security is often difficult to establish and impose stringent restrictions on the implementations which rule out construction with no obvious weakness or, at the very least, require inefficient realizations [5, 25]. In particular, the only attempt at a simulation-based definition for access control is the work of Halevi, Karger, and Naor [19] who provide such a security notion for access control in distributed file storage. Their definition is for a specific system rather than for a general model as the one developed in this paper.

Our contribution. The observation that motivates this paper is that simulation-based security with composability properties is an excellent fit for the context of cryptographic access control. Such systems involve multiple parties, are quite complex and need to satisfy several security requirements (e.g. both individual and joint privacy for the protected objects). Moreover, by their raison d'être, access control systems need to maintain their security properties when employed within higher level protocols. Below, we overview our results.

Security definition. We start with the formalization of an ideal functionality that captures the security guarantees expected from a cryptographic RBAC system. Our functionality reflects directly the semantics associated to RBAC systems [26]. Roughly speaking, the functionality keeps track of all of the operations performed on the system and maintains the induced access control matrix; user requests to access files are then granted/refused based on this matrix. Security in the sense that we define requires that an adversary cannot do more against the concrete implementation than it can do against the functionality. This requirement essentially says that the implementation enforces the expected semantics of the RBAC system. Technically, to show security one needs to construct a *simulator* which can simulate the complete view that an adversary has against the real system but only from access to the information that the ideal functionality provides.

We note that our approach should work for any other model that benefits from a precise semantics with an induced access control matrix.

Relation with game-based definitions. Next, we study the relation between the existent game-based security notions and the level of security that our definition entails. It is generally believed that, for the same task, simulation-based security is stronger than game-based security, even if only because the former is supposed to capture *all* of the security properties expected of a system. Indeed, we show that our notion of security entails both security with respect to read access (the game-based variant introduced in [13]) and write access (the game-based variant introduced in [12]). While expected these types of results help build confidence in the definitions.

Lower-bounds for UC-secure cRBAC. Our main result is a gap between simulation-security and game-based security. More precisely, we show that it is impossible for a cryptographic RBAC system to be UC-secure. In technical terms, we show that the so-called commitment problem [5] occurs in the context of access control. Roughly, the problem is that the simulator required by the security definition needs to produce valid looking encryptions of the objects that are protected without actually knowing the actual content of these objects (e.g. files). The problem is that when the adversary gains access to such a file (e.g. by corrupting a user who has access to this file), the simulator needs to produce a decryption key that explains the ciphertext as an encryption of some particular content which the simulator did not know when the ciphertext was created.

In a bit more detail, our proof proceeds in two steps. First, we provide a generic construction of a universally composable non-interactive communication protocol (NICP) out of any universally composable cRBAC system. A classical result by Nielsen shows that such schemes do not exist if no setup assumptions are in place [25]. Nielsen's result does not apply directly in our setting since our construction of a NICP inherits several assumptions that are in place for the cR-BAC system; in particular, it requires a publicly available file-system and secure channels between some of the parties. We bypass this difficulty by extending Nielsen's impossibility result to settings that involve these setup assumptions whenever their use are restricted in certain ways, and argue that these restrictions are natural in access control.

2 Preliminaries

In this section, we give a brief overview of Canetti's UC framework [4], provide some background of the Role-Based Access Control and recall Ferrara et at.'s notion of cryptographic Role-Based Access Control.

The Universal Composability (UC) framework. The UC framework is based on the "*real-world/ideal-world*" paradigm, which originates in Goldreich, Micali and Wigderson's paper [16]. The basic idea of this paradigm is to show that the execution of a real-world protocol emulates a process which carries out the given task in an idealized way: all the participants securely provide their individual inputs to a trusted party, who then locally computes the outputs and provide them to the participants according to the specification of the task. The emulation property essentially requires that every possible damage caused by an adversary against the real system can also be simulated by an adversary (the simulator) in the ideal world. Since an adversary cannot really break the idealized protocol, the real-world protocol should also be secure.

This paradigm has been further developed by the UC framework. In the UC framework, the trusted party of the ideal process is modelled as an entity called *ideal functionality* and denoted by $\mathcal{F}$. In addition to handling the inputs obtained repeatedly from the parties and generating the prescribed outputs, $\mathcal{F}$ is allowed to interact with the adversary, in a way that captures the allowed leakage of the protocol. To provide security guarantee under composition, the

UC framework introduces an adversarial entity called the *environment* $\mathcal{Z}$, which represents all possible settings in which the protocol can be executed. $\mathcal{Z}$ acts as an interactive distinguisher which aims to tell if it is interacting with the real protocol or with the ideal one. In the process, the environment is allowed to exchange information with the adversary, to provide inputs to the participants of it choice and to obtain outputs from them. A protocol Π is said to securely realize the functionality $\mathcal{F}$, if for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that no environment can distinguish between its interactions with parties running Π and $\mathcal{A}$ and the interactions with the ideal process for $\mathcal{F}$ and $\mathcal{S}$.

An special type of adversary is the so-called *dummy adversary* $\mathcal{D}$. This adversary simply delivers the messages from the environment to the parties and forwards the messages from the parties to the environment: this adversary essentially allows the environment to fully control the input/output and the communication between the parties. A simulator that works for the dummy adversary essentially gives rise to a simulator for any other adversary.

An important concept in the UC framework is the *hybrid model*, an execution setting which is a mix between a real protocol and an idealized setting. Specifically, in an $\mathcal{F}$-hybrid the parties running the protocol can use multiple copies of an ideal functionality $\mathcal{F}$. The extension of the notion of realizing of an ideal functionality in the hybrid model is immediate. In fact, it captures the essence of the general composition theorem specific to UC. If a protocol ρ securely realizes an ideal functionality $\mathcal{G}$ in $\mathcal{F}$-hybrid model and there is a protocol π securely realizes $\mathcal{F}$, then the composed protocol $\rho^{\pi/\mathcal{F}}$ where all the calls to $\mathcal{F}$ are replaced by calls to π securely realizes $\mathcal{G}$. Hence π provides the same security guarantee as the ideal functionality $\mathcal{F}$ even if used within an arbitrary protocol ρ; furthermore the composed protocol $\rho^{\pi/\mathcal{F}}$ still provides the same security guarantee as the ideal functionality $\mathcal{G}$.

One particular application of hybrid models is to capture various communication models. This is achieved by formulating an appropriate ideal functionality $\mathcal{F}$ that represents the abstraction from the communication, then real-world protocols in the communication model can be presented in the $\mathcal{F}$-hybrid model. To exemplify this approach, we present $\mathcal{F}_{\text{SMT}}$, the ideal functionality for secure message transmission (aka secure communication) in Fig. 1. In $\mathcal{F}_{\text{SMT}}$, a sender P_S with input m sends its input to a receiver P_R, while the adversary only learns $|m|$, the length of the message m, and can delay the message delivery. Notice that $\mathcal{F}_{\text{SMT}}$ can only transmit a single message, to transmit multiple messages we need to use multiple instances of $\mathcal{F}_{\text{SMT}}$. We refer to [4] for more details and formal descriptions about the UC framework.

Cryptographic Role-Based Access Control. Role-based access control (RBAC) is a general access control model that offers many benefits including allowing for fine-grained access controls and simplifying the management of user permissions. Instead of assigning users with the permissions directly, it introduces an indirection namely the roles such that the access control policies are decomposed into two associations: the user-role assignment relation and permission-role assignment relation. More formally, at any point a (core) RBAC

Functionality $\mathcal{F}_{\text{SMT}}$

$\mathcal{F}_{\text{SMT}}$ proceeds as follows, with a sender P_S, a receiver P_R and an adversary $\mathcal{S}$.

1. Upon receiving an input $(\texttt{Send}, sid, P_R, m)$ from P_S, send $(\texttt{Sent}, sid, P_S, P_R, |m|)$ to the adversary and generate a delayed output $(\texttt{Sent}, P_S, sid, m)$ to P_R then halt.
2. Upon receiving $(\texttt{Corrupt}, sid, P)$ from $\mathcal{S}$, where $P \in \{P_S, P_R\}$, reveal m to the adversary. If $P = P_S$ and the message has not yet been sent to P_R, then ask $\mathcal{S}$ for a value m' and output $(\texttt{Sent}, P_S, sid, m')$ to P_R then halt.

Fig. 1. Ideal functionality for the secure message transmission, $\mathcal{F}_{\text{SMT}}$.

system is in a state which consists of a set of users U, a set of roles R, a set of permissions P, a relation $UA \subseteq U \times R$ which records the assignment of users to roles and a relation $PA \subseteq P \times R$ which maintains the assignment of permissions to roles. Intuitively, a user u has the permission p if there exists a role r such that $(u, r) \in UA$ and $(p, r) \in PA$.

The state of an RBAC system changes dynamically. Throughout the paper we make the simplifying assumption that the set of roles R is fixed (the assumption reflects the reality that in many organizations the role structure is usually stable). The remaining components of the state change following *administrative commands* of the form $(U', O', P', UA', PA') \leftarrow Cmd((U, O, P, UA, PA), arg)$. We summarize the typical commands and their intended semantics in Fig. 2.

Command	**Semantics**	**Description**
$\mathsf{AddUser}(u)$	$U \cup \{u\}$	Add a new user u to the system
$\mathsf{DelUser}(u)$	$U \setminus \{u\}$, $UA \setminus \{(u, r) \in UA \mid r \in R\}$	Remove an existing user u
$\mathsf{AddObject}(o)$	$O \cup \{o\}$	Add a new object o to the system
$\mathsf{DelObject}(o)$	$O \setminus \{o\}, P \setminus \{(o, \cdot)\}$, $PA \setminus \{((o, \cdot), r) \in PA \mid r \in R\}$	Remove an existing object o
$\mathsf{AssignUser}(u, r)$	$UA \cup \{(u, r)\}$	Assign the user u to the role r
$\mathsf{DeassignUser}(u, r)$	$UA \setminus \{(u, r)\}$	Deassign the user u from the role r
$\mathsf{GrantPerm}(p, r)$	$PA \cup \{(p, r)\}$	Grant the permission p to the role r
$\mathsf{RevokePerm}(p, r)$	$PA \setminus \{(p, r)\}$	Revoke the permission p from the role r

Fig. 2. Administrative RBAC commands.

At a high-level, a cryptographic implementation of RBAC (cRBAC) consists of algorithms that implement the administrative commands outlined above, in a way that enforces the desired access matrix of the system. Formal syntax and security for such systems have been introduced by Ferrara et al. for the case where access is only concerned with reading sensitive data [13] and was later extended to also enforcing writing to sensitive files [12]. As in these works, we assume a setting that involves three main entities: a manager, a file system and a set of users. The manager is assumed to be a trusted party and is tasked with carrying out the administrative commands. The file system is publicly-accessible and is assumed to support versioning – users are only allowed to append content to the file system but not to delete any data.

More precisely, a cRBAC scheme $\mathcal{CRBAC}$ consists of the following algorithms: Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, Update, Write and Read. As suggested above (and by their names) most of the algorithms correspond to the different administrative commands of RBAC. There are three additional algorithms which we describe below. We assume that they define non-interactive multi-party computations which proceeds as follows: first, the manager carries out some local computations according to the RBAC command (including updates to the file system) and produces a set of update messages, one for each of the users. The messages are sent over private channels to users who use it to update their local state. Therefore, these algorithms take as input the locate state of the manager st_M, the current state of the file system fs and the argument for the command arg as input and output an updated state fs' for file system accordingly, a new state for the manager and a set of update messages $\{msg_u\}_{u \in U}$ for the users. When a user u receives an update message msg_u from the manager, it then executes the Update algorithm with its local state $st[u]$ and the message to update its local state accordingly. The Read and Write are the algorithms that allow users to read/write from the files. Both algorithms take as input the local state of the user $st[u]$, a file name o (potentially some content to be written to the file m) and the state of the file system fs. The Read algorithm should return the content of the file; the Write algorithm should return the content that should be appended to the file system.

3 A UC security definition for cRBAC

This section presents a universally composable security definition for cRBAC systems. We formalize the security requirements by designing an ideal functionality $\mathcal{F}_{\text{CRBAC}}$.

Functionality $\mathcal{F}_{\text{CRBAC}}$. The ideal functionality we present in Fig. 3 captures the intuitive security properties of cRBAC systems in the way of simply behaving as a server-mediated access control on the files being protected. Very roughly, $\mathcal{F}_{\text{CRBAC}}$ keeps track of every operation performed on the system and maintain the induced access control matrix within, while it preserves that only the authorized access requests will be granted. This is achieved by having $\mathcal{F}_{\text{CRBAC}}$ maintain a built-in database to store the content of every file, along with a symbolic RBAC state of the system. Then it handles every access request according to the RBAC state.

More specifically, $\mathcal{F}_{\text{CRBAC}}$ embodies the essential interfaces of a cRBAC system, including system initialization, RBAC administration and read/write access to the file system. It proceeds as follows. Having received an initialization request

Functionality $\mathcal{F}_{\text{CRBAC}}$

$\mathcal{F}_{\text{CRBAC}}$ proceeds as follow, with a manager M, users u_1, ..., u_n and an adversary $\mathcal{S}$.

Initialization: Upon receiving an input ($\texttt{Initialization}, sid, R$) from M where R is a set of roles, send ($\texttt{Initialization}, sid, R$) to $\mathcal{S}$, initialize an object-indexed list $FS \leftarrow \emptyset$ and the symbolic RBAC state $(U, O, P, R, PA, UA) \leftarrow (\emptyset, \emptyset, \emptyset, R, \emptyset, \emptyset)$. After that, mark the system as initialized and ignore all the inputs of the form ($\texttt{Initialization}, sid, R'$) for some R' from now on.

RBAC administration: Upon receiving an input ($\texttt{RBAC}, sid, cmd, arg$) from M where cmd is one of the administrative RBAC commands specified in Fig. 2 and arg is the command-specific arguments, proceed as follows: If the system has not yet been initialized, or the execution of cmd with arg is invalid (e.g. it is considered as invalid if $cmd =$"GrantPerm" and $arg = (p, r)$, where $r \notin R$), return an error. Otherwise, update the current RBAC state symbolically $(U', O', P', UA', PA') \leftarrow cmd((U, O, P, UA, PA), arg)$ and then send ($\texttt{RBAC}, sid, cmd, arg$) to $\mathcal{S}$. If $cmd =$"DelObject" and $arg = o$, also delete the content of $FS[o]$.

Write: Upon receiving an input ($\texttt{Write}, sid, o, m$) from some user u where o is an object and m is some content, if the system has not been initialized or there exists no role r such that both $(u, r) \in UA$ and $((o, \texttt{write}), r) \in PA$ are satisfied, return an error. Otherwise, set $FS[o] \leftarrow m$ and send ($\texttt{Wrote}, sid, o, |m|$) to $\mathcal{S}$, where $|m|$ is the length of m.

Read: Upon receiving an input ($\texttt{Read}, sid, o$) from some user u where o is an object, if the system has not been initialized or there exists no role r such that both $(u, r) \in UA$ and $((o, \texttt{read}), r) \in PA$ are satisfied, return an error. Otherwise, set $m \leftarrow FS[o]$ (if $FS[o]$ stores no content then set m as an empty value), and return m to u.

Corruption: $\mathcal{F}_{\text{CRBAC}}$ is a standard corruption ideal functionality, with the exception that any request for corrupting M will be ignored.

Fig. 3. The cryptographic Role-Based Access control functionality, $\mathcal{F}_{\text{CRBAC}}$.

with a set of roles R from the manager M, $\mathcal{F}_{\text{CRBAC}}$ initializes an object-indexed list fs and the symbolic system RBAC state. Then it notices the adversary that the access control system is initialized with a set of roles R. Once $\mathcal{F}_{\text{CRBAC}}$ is initialized, it ignores the other initialization request afterwards. Having received a request of executing an administrative RBAC command from M, $\mathcal{F}_{\text{CRBAC}}$ checks if the command and its arguments specified in the request are valid. If so, it executes the command symbolically and updates the system RBAC state. The administrative RBAC command can be either of the commands presented in Fig. 2. Having received a request to write some content m on a file o from some arbitrary user u, $\mathcal{F}_{\text{CRBAC}}$ first checks if u has the write permission. If so, it stores m in $FS[o]$ and leaks o and the length of m to the adversary. Having received a request to read the content of a file o from some user u, $\mathcal{F}_{\text{CRBAC}}$ also checks if u has the read permission. If so, $\mathcal{F}_{\text{CRBAC}}$ returns the content stored in $FS[o]$ to u. If $FS[o]$ stores no content, it returns an empty value. With the use of the built-in

Functionality $\mathcal{F}_{\text{VFS}}$

$\mathcal{F}_{\text{VFS}}$ proceeds as follows, running with users $u_1, ..., u_n$, a file system manager M and an adversary $\mathcal{S}$. At the first activation $\mathcal{F}_{\text{VFS}}$ initializes a list L to be empty.

Status: Upon receiving an input $(\texttt{Status}, \mathit{sid})$ from a party P, output $(\texttt{Content}, \mathit{sid}, L)$ to P.

Write (user): Upon receiving an input $(\texttt{Write}, \mathit{sid}, o, c)$ from some user u, if no record $r \in L$ of the form $(\mathit{sid}, o, \cdot, \cdot)$ exists, set $L \leftarrow L \cup \{(\mathit{sid}, o, 1, c)\}$; otherwise, set $L \leftarrow L \cup \{(\mathit{sid}, o, \mathit{ver}_m + 1, c)\}$, where $\mathit{ver}_m = \mathit{max}(\{\mathit{ver} | (\mathit{sid}, o, \mathit{ver}, \cdot) \in L\})$. Then send $(\texttt{Wrote}, \mathit{sid}, o, \mathit{ver}, c)$ to $\mathcal{S}$ and send $(\texttt{Updated}, \mathit{sid})$ to every user.

Write (manager): Upon receiving an input $(\texttt{Write}, \mathit{sid}, o, \mathit{ver}, c)$ from M, if a record $r \in L$ of the form $(\mathit{sid}, o, \mathit{ver}, \cdot)$ exists, modify r as $(\mathit{sid}, o, \mathit{ver}, c)$; otherwise, set $L \leftarrow L \cup \{(\mathit{sid}, o, \mathit{ver}, c)\}$. Then send $(\texttt{Wrote}, \mathit{sid}, o, \mathit{ver}, c)$ to $\mathcal{S}$ and send $(\texttt{Updated}, \mathit{sid})$ to every user.

Remove: Upon receiving an input $(\texttt{Remove}, \mathit{sid}, o, \mathit{ver})$ from M, set $L \leftarrow L \setminus \{(\mathit{sid}, o, \mathit{ver}, c)\}$. Then send $(\texttt{Removed}, \mathit{sid}, o, \mathit{ver}, c)$ to $\mathcal{S}$ and send $(\texttt{Updated}, \mathit{sid})$ to every user.

Fig. 4. Ideal functionality for the versioning file storage, $\mathcal{F}_{\text{VFS}}$.

database, $\mathcal{F}_{\text{CRBAC}}$ guarantees correctness: the content that has been written to a file by an authorized user will be read by a user who is entitled to read that file. $\mathcal{F}_{\text{CRBAC}}$ is a standard corruption ideal functionality, with an exception that the manager M cannot be corrupted. It captures the reasonable trust on the manager to administrate the access control system.

Several remarks on $\mathcal{F}_{\text{CRBAC}}$ are in order. First, $\mathcal{F}_{\text{CRBAC}}$ is an ideal functionality for the general cryptographic role-based access controls. Due to the purpose of studying the relation between the previous game-based security notions, $\mathcal{F}_{\text{CRBAC}}$ does not handle any administrative request of adding a new role or removing an existing role. Second, $\mathcal{F}_{\text{CRBAC}}$ only guarantees secure access to the file system and preserves no policy privacy (when handling an administrative request, it simply reveals the command and the arguments to the adversary). There still exists some design choices on policy privacy preserving (e.g. only leaks the executed command but not its arguments), which is left as further study. Third, $\mathcal{F}_{\text{CRBAC}}$ makes no explicit restriction on the form of the file system and the file system is not designed as an individual party of the system. Thus in a real-world cRBAC system, the file system should be implemented by the protocol itself. It also captures that the file system does not implement any access control mechanism. Fourth, $\mathcal{F}_{\text{CRBAC}}$ does not have any authentication mechanism on the parties' identities. The authentication is left to the protocols that make calls to $\mathcal{F}_{\text{CRBAC}}$.

Before presenting our definition of universally composable cRBAC system, we first need to transform a cRBAC scheme $\mathcal{CRBAC} = (\mathsf{Init}, \mathsf{AddUser}, \mathsf{DelUser}, \mathsf{AddObject}, \mathsf{DelObject}, \mathsf{AssignUser}, \mathsf{DeassignUser}, \mathsf{GrantPerm}, \mathsf{RevokePerm})$ into an associated protocol $\Pi_{\mathcal{CRBAC}}$ in the UC setting accordingly. Recall that $\mathcal{CRBAC}$ assumes private channels between the manager and the users. To model this, we let the parties have access to $\mathcal{F}_{\text{SMT}}$, the ideal functionality of secure message transmission which is presented in Fig. 1. Also, $\mathcal{CRBAC}$ makes use of a public-accessible versioning file system. This is modelled by an appropriate functionality $\mathcal{F}_{\text{VFS}}$ which is presented in Fig. 4. $\mathcal{F}_{\text{VFS}}$ proceeds with a set of users and a data manager. Essentially, it serves as an ideal versioning file system which guarantees the correct ordering of the file versions. The users can "write" to the file system by appending new versions to the files instead of overwriting existing contents. The data manager is provided with richer interfaces: it can remove and even rewrite some existing version of a file. All the users in the system can check the current state of the file system by providing a status request to $\mathcal{F}_{\text{VFS}}$. In implementation, the state of the file system would be a bitstring which consists of an array of (possibly encrypted) files; while in $\mathcal{F}_{\text{VFS}}$, it is presented as a list of entries with no loss of generality. When any change happens to the file system, the ideal functionality reveals the change to the adversary and also notices the users about the change. These reflect the public-accessible feature of the file system. In addition, any write operation to the file system is done in an anonymous manner, $\mathcal{F}_{\text{VFS}}$ will not reveal information about the identity of the party who carries out the write operation.

To simplify the protocol presentation, we will also define some shorthand notations. When a party runs some algorithm, it may generate a set of order-preserving instructions to be carried out on the file system. We use $\{info_i\}_{i\in\mathbb{N}}$ to denote this set of instructions. If the party is the manager, each instruction $info_i \in \{info_i\}_{i\in\mathbb{N}}$ can be either $(\texttt{Write}, sid, o, ver, c)$ or $(\texttt{Remove}, sid, o, ver)$, where sid is the session id of $\mathcal{F}_{\text{VFS}}$. If the party is a user, it can only be the form $(\texttt{Write}, sid, o, c)$. A party may also need to come up with a set of order-preserving instructions $\{info_i^{fs\to fs'}\}_{i\in\mathbb{N}}$ such that after carrying out the instructions on the file system in order, the current state of the file system fs would become fs'. We say a party sends $\{info_i\}_{i\in\mathbb{N}}$ (or $\{info_i^{fs\to fs'}\}_{i\in\mathbb{N}}$) to $\mathcal{F}_{\text{VFS}}$, it means the party provides every instruction $info_i$ of the set as the input to $\mathcal{F}_{\text{VFS}}$ in order.

We now present the associated protocol $\Pi_{\mathcal{CRBAC}}$ (in Fig. 5) and define universally composable cRBAC system.

Definition 1. *Let $\mathcal{CRBAC}$ = (Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm) be a cRBAC scheme, we say $\mathcal{CRBAC}$ is UC-secure if the associated protocol $\Pi_{\mathcal{CRBAC}}$ securely realizes $\mathcal{F}_{\text{CRBAC}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model and in a setting that the manager never gets corrupted.*

4 UC security is stronger than game-based security of cRBAC

Based on the transformation above, we now study the relation between UC security and game-based security. We treat separately security of read access from that of write access. The game-based security notions are shown in the appendix A.

The Protocol $\Pi_{\mathcal{CRBAC}}$

The participants: a manger M and a set of users $u_1, ..., u_n$.

Initialization: Upon receiving an input $(\texttt{Initialization}, sid, R)$ where R is a set of roles, M computes $(st_M, fs, \{msg_u\}_{u\in U}) \leftarrow\$ \mathsf{Init}(1^\lambda, R)$. It then invokes an instance of $\mathcal{F}_{\text{VFS}}$ as the data manager with session id (M, sid), parses fs as $\{info_i\}_{i\in\mathbb{N}}$ and sends $\{info_i\}_{i\in\mathbb{N}}$ to $\mathcal{F}_{\text{VFS}}$. If $\{msg_u\}_{u\in U}$ is non-empty, M sends msg_u to every user u using $\mathcal{F}_{\text{SMT}}$.

Administration: Upon receiving an input $(\texttt{RBAC}, sid, cmd, arg)$ where cmd can be either of the administrative commands specified in Fig. 2 and arg is the arguments of the command, M sends $(\texttt{Status}, (M, sid))$ to $\mathcal{F}_{\text{VFS}}$ to obtain $(\texttt{Content}, sid, fs)$ and then computes $(st'_M, fs', \{msg_u\}_{u\in U}) \leftarrow\$ Cmd(st_M, fs, arg)$, where Cmd is the algorithm that implements the administrative command cmd. M sets $st_M \leftarrow st'_M$, and then comes up with $\{info_i^{fs\rightarrow fs'}\}_{i\in\mathbb{N}}$. If $\{info_i^{fs\rightarrow fs'}\}_{i\in\mathbb{N}}$ is non-empty, M sends $\{info_i^{fs\rightarrow fs'}\}_{i\in\mathbb{N}}$ to $\mathcal{F}_{\text{VFS}}$. If $\{msg_u\}_{u\in U}$ is non-empty, M sends msg_u to every user u using $\mathcal{F}_{\text{SMT}}$.

Update: Upon receiving a message $(\texttt{Update}, sid, msg_u)$ from M, a user u computes $st'_u \leftarrow\$ \mathsf{Update}(st_u, msg_u)$, where st_u is u's local state (st_u is an empty value when u receives the first update message from M). Then it sets $st_u \leftarrow st'_u$.

Write: Upon receiving an input $(\texttt{Write}, sid, o, m)$, a user u sends $(\texttt{Status}, (M, sid))$ to $\mathcal{F}_{\text{VFS}}$ to get $(\texttt{Content}, sid, fs)$ and computes $fs' \leftarrow\$ \mathsf{Write}(st_u, fs, o, m)$. Then u comes up with $\{info_i^{fs\rightarrow fs'}\}_{i\in\mathbb{N}}$ and sends it to $\mathcal{F}_{\text{VFS}}$.

Read: Upon receiving an input $(\texttt{Read}, sid, o)$, a user u sends $(\texttt{Status}, (M, sid))$ to $\mathcal{F}_{\text{VFS}}$ to get $(\texttt{Content}, sid, fs)$ and then outputs $(\texttt{Read}, sid, \mathsf{Read}(st_u, fs, o))$.

Fig. 5. The Protocol $\Pi_{\mathcal{CRBAC}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model.

Theorem 1. *Any cRBAC scheme $\mathcal{CRBAC}$ which is UC-secure (in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model) is secure with respect to write accesses.*

Proof sketch. We show that if $\mathcal{CRBAC}$ is not secure with respect to write accesses, then it cannot be UC-secure. Given an adversary $\mathcal{A}_W$ that breaks write security of $\mathcal{CRBAC}$, an environment $\mathcal{Z}$ can distinguish its interactions with parties running $\Pi_{\mathcal{CRBAC}}$ and a dummy adversary, and the interactions with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and a simulator. The idea is, $\mathcal{Z}$ runs a local copy of $\mathcal{A}_W$ and simulates to it the experiment that defines write security of cRBAC schemes. Then $\mathcal{Z}$ proceeds according $\mathcal{A}_W$'s queries such that the protocol execution is consistent to $\mathcal{A}_W$'s view. Since $\mathcal{A}_W$ is a successful adversary by assumption, $\mathcal{Z}$ should be able to write some valid content without having the permission in the real-world execution with non-negligible probability. But in the ideal world, from the specification of $\mathcal{F}_{\text{CRBAC}}$ we can infer that $\mathcal{Z}$ will not be able to write any content to the file system in this case. The full proof can be found in Appendix B.

Theorem 2. *Any cRBAC scheme $\mathcal{CRBAC}$ which is UC-secure (in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model) is secure with respect to read accesses.*

Proof sketch. The proof idea of this theorem is analogous to Theorem 1's. Given an adversary $\mathcal{A}_R$ that breaks read security of $\mathcal{CRBAC}$, an environment $\mathcal{Z}$ can tell its interactions with the execution of $\Pi_{\mathcal{CRBAC}}$ and a dummy adversary from the interactions with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and a simulator. Similarly, $\mathcal{Z}$ runs a local copy of $\mathcal{A}_R$ and simulates to it the experiment that defines read security. Then $\mathcal{Z}$ transforms every query from $\mathcal{A}_R$, which will not lead to a trivial win, into appropriate inputs being provided to the parties and the adversary. By the assumption on $\mathcal{A}_R$, $\mathcal{Z}$ would be able to distinguish its interactions in the two worlds with the help of $\mathcal{A}_R$. The full proof of this theorem can be found in Appendix C.

5 Impossibility of UC-secure cRBAC scheme

In this section we establish our main result. We show that the level of security demanded by a universally composable cryptographic RBAC system cannot be achieved, even in a setting where a protocol has access to an idealized file system and secure channels between all parties. Our impossibility result is in a setting where the adversary can adaptively corrupt honest protocol participants.

Functionality $\mathcal{F}_{\text{NCE}}$

$\mathcal{F}_{\text{NCE}}$ works as follows. It interacts with a message sender P_S, a receiver P_R and an adversary $\mathcal{S}$.

Pre-processing phase: Upon receiving an input $(\texttt{Init}, sid, P_R)$ from P_S, send $(\texttt{Init}, sid, P_S)$ to P_R and send $(\texttt{Init}, sid, P_S, P_R)$ to $\mathcal{S}$. In addition, mark the channel as established.

Communication phase: Upon receiving an input $(\texttt{Send}, sid, P_R, m)$ from P_S, if the channel has not been established, ignore this input. Otherwise, deliver the message $(\texttt{Send}, sid, P_S, m)$ to P_R and reveal $(\texttt{Sent}, sid, P_S, P_R, |m|)$ to $\mathcal{S}$, where $|m|$ is the length of the message.

Corruption: Upon receiving $(\texttt{Corrupt}, sid, P)$ from $\mathcal{S}$ where $P \in \{P_S, P_R\}$, reveal m to $\mathcal{S}$. If $P = P_S$ and the message has not yet been delivered to P_R, ask $\mathcal{S}$ for a value m' then output $(\texttt{Sent}, sid, P_S, m')$ to P_R.

Fig. 6. Ideal functionality for the non-committing encryption, $\mathcal{F}_{\text{NCE}}$.

Theorem 3. *There exists no UC-secure cRBAC scheme (in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model) with adaptive corruptions.*

Proof: The proof of this theorem proceeds in two steps. First, we show that the existence of any UC-secure cRBAC scheme implies the existence of a universally composable NICP. Specifically, we provide a generic construction of a NICP that securely realizes the functionality $\mathcal{F}_{\text{NCE}}$ of non-committing encryption (which is presented in Fig. 6), from any UC-secure cRBAC scheme. Next, we argue that the resulting communication protocol in fact cannot securely realize $\mathcal{F}_{\text{NCE}}$ – this step is an extension of a well known result by Nielsen to a setting where parties have access to a secure file system and secure channels.

The Protocol $\mathit{\Pi}_{\text{NICP}}$

The participants: a message sender P_S, a receiver P_R and a trusted party M namely the manager.

Pre-processing phase. M establishes the communication channel for P_S and P_R. In this stage, some content might be written to $\mathcal{F}_{\text{VFS}}$ for the channel set-up.

1. Upon receiving an input $(\texttt{Init}, \mathit{sid}, P_R)$, P_S sends $(\texttt{Init}, \mathit{sid}, P_R)$ to M using $\mathcal{F}_{\text{SMT}}$.
2. Upon receiving a message $(\texttt{Init}, \mathit{sid}, P_R)$ from P_S, M selects a random role r and computes $(\mathit{st}_M, \mathit{fs}, \{\mathit{st}[u_S], \mathit{st}[u_R]\}) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda, \{r\})$, where u_S and u_R are two users to be added to the system. It initializes two lists $\mathit{msg}_S \leftarrow \mathit{st}[u_S]$ and $\mathit{msg}_R \leftarrow \mathit{st}[u_R]$. M then invokes an instance of $\mathcal{F}_{\text{VFS}}$ with session id (M, sid) as the data manager and parses fs as $\{\mathit{info}_i\}_{i\in\mathbb{N}}$. If $\{\mathit{info}_i\}_{i\in\mathbb{N}}$ is non-empty, M sends $\{\mathit{info}_i\}_{i\in\mathbb{N}}$ to $\mathcal{F}_{\text{VFS}}$. After that, M runs a sequence of algorithms which implement the related administrative RBAC commands to add two users u_S, u_R and an object o to the system, to grant the write permission of o to u_S via the role r and to grant the read permission of o to u_R via r. The run of any of the algorithms might lead to the file system's current state fs gets updated to fs'. If so, M comes up with $\{\mathit{info}_i^{\mathit{fs}\to\mathit{fs}'}\}_{i\in\mathbb{N}}$ and sends it to $\mathcal{F}_{\text{VFS}}$. Whenever an update message msg for u_S (u_R resp.) is generated, M appends it to the list msg_S (msg_S resp.). Finally, after the run of the algorithms M sends $(\texttt{Update}, \mathit{sid}, \mathit{msg}_S)$ to P_S and sends $(\texttt{Update}, \mathit{sid}, \mathit{msg}_R)$ to P_R using $\mathcal{F}_{\text{SMT}}$.
3. Upon receiving a message $(\texttt{Update}, \mathit{sid}, \mathit{msg}_X)$ from M where $X \in \{S, R\}$, the party P_X updates its local state by running the update algorithm $\mathit{st}_X \leftarrow\!\!\$\ \mathsf{Update}(\mathit{st}_X, \mathit{msg})$ on each update message $\mathit{msg} \in \mathit{msg}_X$ in order.

Communication Phase. Once the channel has been established, P_S can send arbitrarily many messages to P_R via $\mathcal{F}_{\text{VFS}}$.

1. Upon receiving an input $(\texttt{Send}, \mathit{sid}, P_R, m)$, P_S sends $(\texttt{Status}, (M, \mathit{sid}))$ to $\mathcal{F}_{\text{VFS}}$ to get $(\texttt{Content}, (M, \mathit{sid}), \mathit{fs})$, and then computes $\mathit{fs}' \leftarrow\!\!\$\ \mathsf{Write}(\mathit{st}_S, \mathit{fs}, o, m)$. Next, P_S comes up with $\{\mathit{info}_i^{\mathit{fs}\to\mathit{fs}'}\}_{i\in\mathbb{N}}$ and sends it to $\mathcal{F}_{\text{VFS}}$.
2. Upon receiving an subroutine output $(\texttt{Updated}, (M, \mathit{sid}))$ from $\mathcal{F}_{\text{VFS}}$, P_R sends $(\texttt{Status}, (M, \mathit{sid}))$ to $\mathcal{F}_{\text{VFS}}$ to get $(\texttt{Content}, (M, \mathit{sid}), \mathit{fs})$, and then outputs $m' \leftarrow \mathsf{Read}(\mathit{st}_R, \mathit{fs}, o)$.

Fig. 7. The Protocol $\mathit{\Pi}_{\text{NICP}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model.

We start by describing the generic construction for the universally composable NICP. Recall that based on our transformation, the associated protocol of a cRBAC scheme works in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model and in a setting that the manager never gets corrupted, the resulting communication protocol therefore works in the same hybrid model and makes use of such a trusted party in a restricted way.

The communication protocol involves a message sender, a receiver and a trusted party namely the manager. We restrict that there exists no direct communication channel between the sender and the receiver. They have to communicate with each other in an indirect way: after a pre-processing phase in which the manager interacts with the other two parties over secure channels to establish the communication, the sender can send messages to the receiver by writing to the file system and then the receiver performs read operations to get the messages. Notice that the read operation will not bring any change to the file system and the manager only works in the pre-processing phase and does not involve in the communication phase. The communication protocol in fact requires no interaction between the sender and the receiver. Hence it can be considered as non-interactive.

More specifically, let $\mathcal{CRBAC} = \{\mathsf{Init}, \mathsf{AddUser}, \mathsf{DelUser}, \mathsf{AddObject}, \mathsf{DelObject}, \mathsf{AssignUser}, \mathsf{DeassignUser}, \mathsf{GrantPerm}, \mathsf{RevokePerm}, \mathsf{Update}, \mathsf{Write}, \mathsf{Read}\}$ be a UC-secure cRBAC scheme. We denote the NICP by Π_{NICP} and present it Fig. 7.

Then we show that Π_{NICP} securely realizes $\mathcal{F}_{\text{NCE}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model. By assumption, the scheme $\mathcal{CRBAC}$ is UC-secure implies that there exists a simulator $\mathcal{S}$ such that no environment can tell with non-negligible probability whether it interacts with the parties running $\Pi_{\mathcal{CRBAC}}$ in $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{VFS}})$-hybrid model and a dummy adversary $\mathcal{D}$, or it interacts with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ with $\mathcal{S}$. Then we give the construction of the simulator $\mathcal{S}_{\text{NCE}}$ for Π_{NICP} as follows. $\mathcal{S}_{\text{NCE}}$ internally runs an instance of $\mathcal{S}$. Then it interacts with $\mathcal{S}$ as the environment and simulates to $\mathcal{S}$ the ideal process for $\mathcal{F}_{\text{CRBAC}}$. It proceeds as follow.

1. **Simulating the pre-processing phase.** Upon receiving from $\mathcal{F}_{\text{NCE}}$ a message $(\texttt{Init}, sid, P_S, P_R)$, $\mathcal{S}_{\text{NCE}}$ selects a random role r. It then simulates the pre-processing phase by sending messages to $\mathcal{S}$ sequentially in the name of $\mathcal{F}_{\text{CRBAC}}$ indicating that the cRBAC system is initialized with a role r, two users u_S and u_R, an object o are added to the system, u_S is granted the write permission of o via the role r and u_R is granted the read permission of o via r. When the environment requests $\mathcal{S}_{\text{NCE}}$ to provide any information that it can obtain during this phase including the length of any final update message sent by the manager in Π_{NICP} and any content written to $\mathcal{F}_{\text{VFS}}$, $\mathcal{S}_{\text{NCE}}$ instructs $\mathcal{S}$ to provide the related information and hands it to the environment appropriately.
2. **Simulating the communication phase.** Upon receiving from $\mathcal{F}_{\text{NCE}}$ a message $(\texttt{Sent}, sid, P_S, P_R, |m|)$, $\mathcal{S}_{\text{NCE}}$ sends $(\texttt{Wrote}, sid', o, |m|)$ in the name of $\mathcal{F}_{\text{CRBAC}}$ to $\mathcal{S}$, where $sid' = (M, sid)$. When the environment requests $\mathcal{S}_{\text{NCE}}$ to report the content written to $\mathcal{F}_{\text{VFS}}$, $\mathcal{S}_{\text{NCE}}$ instructs $\mathcal{S}$ to report such content and forwards it as its output appropriately.
3. **Party corruption.** When the environment instructs $\mathcal{S}_{\text{NCE}}$ to corrupt P_S (P_R resp.), $\mathcal{S}_{\text{NCE}}$ delivers the corruption message to $\mathcal{F}_{\text{NCE}}$ and also requests $\mathcal{S}$ to corrupt u_S (u_R resp.). If the corruption happens after P_S has ever sent some message to P_R, $\mathcal{S}_{\text{NCE}}$ will also obtain the messages sent so far from $\mathcal{F}_{\text{NCE}}$.

Then it provides the obtained information to $\mathcal{S}$ in the name of $\mathcal{F}_{\text{CRBAC}}$. Once $\mathcal{S}$ outputs the internal state of the corrupt party, $\mathcal{S}_{\text{NCE}}$ forwards it to the environment. After that, any message provided by the environment to the corrupt party would be modified as the message for u_S (u_R resp.) accordingly and forwarded to $\mathcal{S}$ (e.g. if the environment instructs the corrupt sender to send some message c, $\mathcal{S}_{\text{NCE}}$ then instructs $\mathcal{S}$ to write the message c to the file o on behave of u_S). Any request from the environment to corrupt the manager will be ignored.

We briefly analyse the validity of $\mathcal{S}_{\text{NCE}}$. Suppose there exists an environment $\mathcal{Z}$ which can tell its interactions with parties running Π_{NICP} in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model and a dummy adversary from the interactions with the ideal process for $\mathcal{F}_{\text{NCE}}$ and $\mathcal{S}_{\text{NCE}}$ with non-negligible probability. We show that an environment $\mathcal{Z}'$ can be constructed to tell whether it is interacting with parties running $\Pi_{\mathcal{CRBAC}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model and a dummy adversary or the interactions with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and the simulator $\mathcal{S}$ with non-negligible probability. The main idea is that $\mathcal{Z}'$ runs an internal copy of $\mathcal{Z}$ towards which it simulates the view of the ideal process for $\mathcal{F}_{\text{NCE}}$ and the simulator $\mathcal{S}_{\text{NCE}}$. The simulation depends the information that $\mathcal{Z}'$ can obtain during the protocol execution. From the construction of $\mathcal{S}_{\text{NCE}}$ above, it can be inferred that every instruction for $\mathcal{S}_{\text{NCE}}$ can be broken down to corresponding instructions to $\mathcal{S}$. Also, for the inputs that $\mathcal{Z}$ provides to the dummy parties in the ideal process for $\mathcal{F}_{\text{NCE}}$, $\mathcal{Z}'$ can modify them appropriately and provide to the parties it interacts with. Hence we havethe simulation $\mathcal{Z}'$ provides to $\mathcal{Z}$ is perfectly identical to the view which $\mathcal{Z}$ expects to see. Then by assumption, $\mathcal{Z}$ can tell its interactions in the two worlds with non-negligible probability, and so can $\mathcal{Z}'$ in this case. Thus, $\mathcal{S}$ cannot be a valid simulator for $\Pi_{\mathcal{CRBAC}}$ which reaches a contradiction. So if $\mathcal{S}$ is a valid simulator for $\Pi_{\mathcal{CRBAC}}$, $\mathcal{S}_{\text{NCE}}$ is also a valid simulator for Π_{NICP} and therefore Π_{NICP} securely realizes $\mathcal{F}_{\text{NCE}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model.

Now we argue that in fact such a simulator $\mathcal{S}$ does not exist. In [25], it has been shown that no non-interactive communication protocol that securely realizes $\mathcal{F}_{\text{NCE}}$ exists in the plain model. However, we cannot apply directly that result to complete our proof, since Π_{NICP} makes use of $\mathcal{F}_{\text{VFS}}$, $\mathcal{F}_{\text{SMT}}$, albeit in a restricted way. Nonetheless, we show that under these usage restrictions, we can extend Nielsen's result to our setting.

Since Π_{NICP} securely realizes $\mathcal{F}_{\text{NCE}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model, it allows the sender to send arbitrarily many messages to the receiver non-interactively (e.g. by performing write operations to the file system). Any real-world adversary that attacks the protocol cannot obtain more than the length of the transmitted message. Consider the following environment $\mathcal{Z}$. After the communication is established between the message sender P_S and the receiver P_R, $\mathcal{Z}$ activates P_S with an input $(\texttt{Send}, sid, m)$ and requests the adversary to report the content c that has been written to some file o of $\mathcal{F}_{\text{VFS}}$. Once $\mathcal{Z}$ obtains c, it instructs the adversary to corrupt P_R to obtain its internal state st_R. Then $\mathcal{Z}$ produces the current state of the file system from the update information provided by the adversary as fs and computes $m' \leftarrow \mathsf{Read}(st_R, fs, o)$. By assumption $\mathcal{Z}$ should have $m' = m$ except for negligible probability. Then we consider the ideal-world case, the simulator $\mathcal{S}_{\text{NCE}}$ should be able to come up with c given the length of m by $\mathcal{F}_{\text{NCE}}$, and later it should be able to provide the internal state st_R which is consistent to the transmitted message c when m is available by the time P_R is corrupt. Notice that the ideal functionality $\mathcal{F}_{\text{NCE}}$ guarantees correctness on the transmitted message, which means for every message sent by the sender, the receiver should be able to recover the original message except for negligible probability. Hence for Π_{NICP}, there should not exist any local state of the receiver that allows it to decrypt any written content to the file system into two different messages with non-negligible probability each. Otherwise an environment can distinguish its interactions in the two worlds with non-negligible probability. Thus if we fix a file version c, there exists an injective mapping from the underlying messages to the local state of the receiver, which implies that the number of possible internal states st_R of P_R should be at least the same as the number of the possible messages. Notice that the only way P_R can receive the message from P_S is to execute the Read algorithm to retrieve the current content of o from the file system. The injective mapping will not be affected by executing read operations since (by assumption) Read updates neither the file system nor the local state of P_S. Therefore it is impossible for P_R to use the unchanged local state to receiver arbitrary many messages from P_S. Thus we can conclude that Π_{NICP} does not securely realize $\mathcal{F}_{\text{NCE}}$ in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model, which contradicts the existence of the simulator $\mathcal{S}$. Hence there exists no UC-secure $\mathcal{CRBAC}$ (in $(\mathcal{F}_{\text{VFS}}, \mathcal{F}_{\text{SMT}})$-hybrid model) with adaptive corruptions.

6 Conclusion

We present a security definition for cryptographic role-based access control in the UC framework. We study its relation with existent game-based notions and show that simulation-based security is strictly stronger. In essence, our results imply that composable simulation-based security for access control may be difficult to achieve to the point that it is impractical[1] Interestingly, similar results were derived empirically in a recent study of the efficiency of cryptographic RBAC based on both standard asymmetric encryption and identity-based encryption schemes [21].

In future work, we plan to study the efficiency implications of UC security for cryptographic access control schemes that hide the access policy that is in place at any given time during the execution of the system.

References

1. Martín Abadi and Bogdan Warinschi. Security analysis of cryptographically controlled access to XML documents. *J. ACM*, 55(2), 2008.

[1] One possibility which we did not explore in this paper is to rely on additional setup assumptions, e.g. a common reference string, and employ a non-committing encryption scheme.

2. Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, August 1983.
3. James Alderman, Carlos Cid, Jason Crampton, and Christian Janson. Access control in publicly verifiable outsourced computation. *IACR Cryptology ePrint Archive*, 2014:762, 2014.
4. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.
5. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.
6. Arcangelo Castiglione, Alfredo De Santis, Barbara Masucci, Francesco Palmieri, Aniello Castiglione, and Xinyi Huang. Cryptographic hierarchical access control for dynamic structures. *IEEE Trans. Information Forensics and Security*, 11(10):2349–2364, 2016.
7. Arcangelo Castiglione, Alfredo De Santis, Barbara Masucci, Francesco Palmieri, Aniello Castiglione, Jin Li, and Xinyi Huang. Hierarchical and shared access control. *IEEE Trans. Information Forensics and Security*, 11(4):850–865, 2016.
8. Ya-Fen Chang. A flexible hierarchical access control mechanism enforcing extension policies. *Security and Communication Networks*, 8(2):189–201, 2015.
9. Jason Crampton. Practical constructions for the efficient cryptographic enforcement of interval-based access control policies. *CoRR*, abs/1005.4993, 2010.
10. Jason Crampton. *FAST 2010. Revised selected papers.*, chapter Cryptographic Enforcement of Role-Based Access Control, pages 191–205. 2011.
11. Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management of access control evolution on outsourced data. In *VLDB*, pages 123–134. ACM, 2007.
12. 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.
13. 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.
14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. *TCC 2016-A, Proceedings, Part II*, chapter Functional Encryption Without Obfuscation, pages 480–511. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
15. David K. Gifford. Cryptographic sealing for information secrecy and authentication. *Commununications of the ACM*, 25(4):274–286, 1982.
16. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.
17. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.
18. E. Gudes. The Design of a Cryptography Based Secure File System. *IEEE Transactions on Software Engineering*, 6(5):411–420, 1980.
19. 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.

20. Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *IACR Cryptology ePrint Archive*, 2011:303, 2011.
21. 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.
22. Ralf Küsters and Max Tuengerthal. The IITM model: a simple and expressive model for universal composability. *IACR Cryptology ePrint Archive*, 2013:25, 2013.
23. Benoît Libert and Damien Vergnaud. *Topics in Cryptology – CT-RSA 2009*, chapter Adaptive-ID Secure Revocable Identity-Based Encryption, pages 1–15. 2009.
24. Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. *Topics in Cryptology – CT-RSA 2011*, chapter Attribute-Based Signatures, pages 376–392. 2011.
25. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in CryptologyCrypto 2002*, pages 111–126. Springer Berlin Heidelberg, 2002.
26. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

A The security notions of cRBAC schemes in [12]

Secure write access. A cRBAC scheme $\mathcal{CRBAC} = (\mathsf{Init}, \mathsf{AddUser}, \mathsf{DelUser}, \mathsf{AddUser}, \mathsf{AddObject}, \mathsf{GrantPerm}, \mathsf{RevokePerm}, \mathsf{AssignUser}, \mathsf{DeassignUser}, \mathsf{Update}, \mathsf{Read}, \mathsf{Write})$ is said to be secure with respect to write accesses if no user can write some content to a file without having the permission. Particularly, in the case of open-accessible file system, the content wrote by an unauthorized user should not be considered as valid. It is formalized by the experiment $\mathbf{Exp}^{\mathrm{write}}_{\mathcal{CRBAC},\mathcal{A}}$. The cRBAC system is initialized with a set of role R. The adversary $\mathcal{A}$ is allowed to request for executing any of the administrative RBAC commands, to corrupt a user, to request an honest user to write some content to a file and to get access to the file system. At some point, $\mathcal{A}$ must output a target file with an honest user's id. It wins if it can write any valid content without the permission(read by the honest user). To prevent trivial wins, from the point when the last write operation to the target file is carried out by an honest user who has the permission till $\mathcal{A}$ generates its output, no corrupt user can get write access to the target file. A $\mathcal{CRBAC}$ is said to be secure with respect to write accesses if no adversary can win in the above experiment with non-negligible probability. The adversarial abilities are captured by the oracles described in Fig. 8.

A predicate $\mathsf{HasAccess}(u, p)$ is used to reflect that symbolically a user u has access to a permission p. It is defined by: $\mathsf{HasAccess}(u, p) \leftrightarrow \exists r \in R : (u, r) \in UA \wedge (p, r) \in PA$.

Definition 2. *A cRBAC scheme $\mathcal{CRBAC}$ is secure with respect to write accesses if for any probabilistic polynomial-time adversaries $\mathcal{A}$, we have*

$$\mathbf{Adv}^{write}_{\mathcal{CRBAC},\mathcal{A}}(\lambda) := \Pr\left[\mathbf{Exp}^{write}_{\mathcal{CRBAC},\mathcal{A}}(\lambda) \to 1\right]$$

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

```
$\underline{\textsc{Cmd}(arg)}$
  $(U, O, P, UA, PA)$
      $\leftarrow Cmd((U, O, P, UA, PA), arg)$
  $(st_M, fs, \{msg_u\}_{u \in U})$
      $\leftarrow\$\ \mathsf{Cmd}(st_M, fs, arg)$
  For all $u' \in Cr$:
    For all $o \in O$:
      If $\mathsf{HasAccess}(u', (o, \mathtt{write}))$ then
        $T[o] \leftarrow \mathsf{adv}$
  For all $u \in U \setminus Cr$:
    $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
  Return $(fs, \{msg_u\}_{u \in Cr})$
$\underline{\textsc{CorruptU}(u)}$
  If $u \notin U$ then Return $\bot$
  For all $o \in O$:
    If $\mathsf{HasAccess}(u, (o, \mathtt{write}))$ then
      $T[o] \leftarrow \mathsf{adv}$
  $Cr \leftarrow Cr \cup \{u\}$; Return $st[u]$

$\underline{\textsc{Write}(u, o, m)}$
  If $u \in Cr$ then Return $\bot$
  If $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$
      then Return $\bot$
  $fs \leftarrow\$\ \mathsf{Write}(st[u], fs, o, m)$
  For all $u' \in Cr$:
      If $\mathsf{HasAccess}(u', (o, \mathtt{write}))$
        then Return $fs$
  $T[o] \leftarrow m$; Return $fs$
$\underline{\mathrm{FS}(query)}$
  If $query =$ "STATE" then
      Return $fs$
  If $query =$ "APPEND$(info)$"
      and $Cr \neq \emptyset$ then
      $fs \leftarrow fs \| info$; Return $fs$
```

Fig. 8. Oracles for defining the experiment $\mathbf{Exp}^{\mathrm{write}}_{\mathcal{CRBAC},\mathcal{A}}$.

```
$\underline{\mathbf{Exp}^{write}_{\mathcal{CRBAC},\mathcal{A}}(\lambda)}$
$(U, O, P, UA, PA) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$; $Cr \leftarrow \emptyset$
$(st_M, fs, \{st[u]\}_{u \in U}) \leftarrow\$\ \mathsf{Init}(1^\lambda, R)$
$(u^*, o^*) \leftarrow\$\ \mathcal{A}(1^\lambda : \mathcal{O}_w)$
If all of the following are satisfied then return 1:
  – $u^* \in U \setminus Cr \wedge \mathsf{HasAccess}(u^*, (o^*, \mathtt{read}))$
  – $T[o^*] \neq \mathsf{adv} \wedge T[o^*] \neq \mathsf{Read}(st[u^*], o^*, fs))$
Else Return 0
```

Secure read access. A cRBAC scheme $\mathcal{CRBAC}$ = (Init, AddUser, DelUser, AddUser, AddObject, GrantPerm, RevokePerm, AssignUser, DeassignUser, Update, Read, Write) is said to be secure with respect to read accesses if no user can deduce any content of a file without having the read permission. It is formalized by the experiment $\mathbf{Exp}^{\mathrm{read}}_{\mathcal{CRBAC},\mathcal{A}}$. In the experiment, a random bit is selected at the beginning and the cRBAC system is initialized with a set of roles R. The adversary $\mathcal{A}$ is allowed to request for executing any administrative RBAC command, to take over users, to request an honest user to write some content to a file and to get access to the file system. $\mathcal{A}$ can also specify a file as his challenge and provides two messages, of which one will be written to the file according to the random bit. It can specify multiple challenges and finally output his guess of the bit. To prevent trivial wins, no corrupt user can get read access to any of the challenge files. We say the adversary wins if its guess is correct. A $\mathcal{CRBAC}$ is said to be secure with respect to read accesses if no adversary can win the above experiment with probability significantly better than a half. The oracles to which the adversary has access are specified in Fig. 9.

Definition 3. *A cRBAC scheme $\mathcal{CRBAC}$ is secure with respect to read accesses if for any probabilistic polynomial-time adversary $\mathcal{A}$, we have*

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

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

```
$\underline{\mathbf{Exp}^{read}_{\mathcal{CRBAC},\mathcal{A}}(\lambda)}$
  $b \leftarrow\$ \{0,1\}$; $Cr, Ch \leftarrow \emptyset$
  $(st_M, fs, \{st[u]\}_{u\in U}) \leftarrow\$ \mathsf{Init}(1^\lambda, R)$
  $b' \leftarrow\$ \mathcal{A}(1^\lambda : \mathcal{O}_r)$
  Return $(b' = b)$
```

```
$\underline{\textsc{Cmd}(arg)}$
  $(U', O', P', UA', PA')$
      $\leftarrow Cmd((U, O, P, UA, PA), arg)$
  For all $u \in Cr$ and all $o \in Ch$:
    If $\exists r \in R$:
      $(u, r) \in UA' \wedge ((o, read), r) \in PA'$
      then Return $\perp$
  $(U, O, P, UA, PA) \leftarrow (U', O', P', UA', PA')$
  $(st_M, fs, \{msg_u\}_{u\in U}) \leftarrow\$ \mathsf{Cmd}(st_M, fs, arg)$
  For all $u \in U \setminus Cr$:
    $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
  Return $(fs, \{msg_u\}_{u\in Cr})$

$\underline{\textsc{CorruptU}(u)}$
  If $u \notin U$ then Return $\perp$
  For all $o \in Ch$:
    If $\mathsf{HasAccess}(u, (o, \mathtt{read}))$ 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 $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$
    then Return $\perp$
  $fs \leftarrow\$ \mathsf{Write}(st_M, fs, o, m)$
  Return $fs$

$\underline{\textsc{Challenge}(u, o, m_0, m_1)}$
  If $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$
    then Return $\perp$
  For all $u' \in Cr$:
    If $\mathsf{HasAccess}(u', (o, \mathtt{read}))$
      then Return $\perp$
  $Ch \leftarrow Ch \cup \{o\}$
  $fs \leftarrow\$ \mathsf{Write}(st_M, fs, o, m_b)$
  Return $fs$

$\underline{\textsc{FS}(query)}$
  If $query =$ "STATE" then
    Return $fs$
  If $query =$ "APPEND$(info)$"
    and $Cr \neq \emptyset$ then
    $fs \leftarrow fs \| info$; Return $fs$
```

Fig. 9. Oracles for defining the experiment $\mathbf{Exp}^{\mathrm{read}}_{\mathcal{CRBAC},\mathcal{A}}$.

B Proof of Theorem 1

We prove this theorem by showing that if any cRBAC scheme is not secure with respect to write accesses, it cannot be UC-secure. Assumes that a cRBAC scheme

$\mathcal{CRBAC}$ is not secure with respect to write accesses, then there exists an adversary $\mathcal{A}_W$ can win in $\mathbf{Exp}^{\text{write}}_{\mathcal{CRBAC},\mathcal{A}_W}$ with non-negligible probability. We show that given $\mathcal{A}_W$, an environment $\mathcal{Z}$ can be constructed such that it can distinguish its interactions with the associated protocol $\Pi_{\mathcal{CRBAC}}$ and a dummy adversary $\mathcal{D}$, and the interactions with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and a simulator $\mathcal{S}$ with non-negligible probability, which means $\mathcal{CRBAC}$ is not UC-secure.

We now describe how $\mathcal{Z}$ works. During its execution, $\mathcal{Z}$ maintains three lists: an object-indexed list T for recording the last valid file contents written by users, *fs* for recording the current state of the file system and Cr for recording the corrupt users. $\mathcal{Z}$ first activates the manager M with an input $(\texttt{Initialization}, sid, R)$, where *sid* is an arbitrary string and R is a random set of roles. It then obtains a sequence of messages regarding the changes on $\mathcal{F}_{\text{VFS}}$ (via the dummy adversary $\mathcal{D}$ who just simply delivers the messages) and updates *fs* accordingly such that *fs* is identical to the list L maintained by $\mathcal{F}_{\text{VFS}}$. Then $\mathcal{Z}$ runs a local copy of $\mathcal{A}_W$ with *fs* as input and starts to simulate $\mathbf{Exp}^{\text{write}}_{\mathcal{CRBAC},\mathcal{A}_W}$ as follows.

1. When $\mathcal{A}_W$ asks for executing any RBAC command *cmd* with arguments *arg*, $\mathcal{Z}$ activates M with an input $(\texttt{RBAC}, sid, cmd, arg)$ and updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$. If the execution of the command will lead to any user in the list Cr has the write permission of any file o, $\mathcal{Z}$ sets $T[o]$ as a special value `adv`. If M sends update messages to the corrupt users when executing the command, $\mathcal{Z}$ receives the update messages (via $\mathcal{D}$) and then sends them to $\mathcal{A}_W$. In addition, if $\mathcal{A}_W$ requests to delete some user which is in the list Cr, then $\mathcal{Z}$ removes this user from Cr after executing the command. If $\mathcal{A}_W$ requests to delete some file o, the content in $T[o]$ will be also deleted. Finally, $\mathcal{Z}$ hands *fs* to $\mathcal{A}_W$.
2. When $\mathcal{A}_W$ requests an honest user u to write some content m to a file o, if u does not have the write permission of o, $\mathcal{Z}$ returns an error; otherwise $\mathcal{Z}$ activates u with an input $(\texttt{Write}, sid, o, m)$ and updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$. Then $\mathcal{Z}$ hands *fs* to $\mathcal{A}_W$. If there exists no user in the list Cr that has write access to o, $\mathcal{Z}$ sets $T[o]$ as m.
3. When $\mathcal{A}_W$ requests for corrupting a user u, $\mathcal{Z}$ corrupts u (via $\mathcal{D}$) and returns its local state to $\mathcal{A}_W$. For every file o to which u has write access, $\mathcal{Z}$ sets $T[o]$ as the special value `adv`. Then it adds u to Cr.
4. When $\mathcal{A}_W$ queries the current state of the file system, $\mathcal{Z}$ hands *fs* to $\mathcal{A}_W$.
5. When $\mathcal{A}_W$ requests to update the file system with some information *info*, if Cr is empty, $\mathcal{Z}$ then ignores the request; otherwise $\mathcal{Z}$ phases *info* as (o, c), where o is a file and c is the content to be appended to. $\mathcal{Z}$ chooses a user u from Cr and sends u a message (via $\mathcal{D}$) to let it provide an input $(\texttt{Write}, sid', o, c)$ to $\mathcal{F}_{\text{VFS}}$, where sid' is the session id of $\mathcal{F}_{\text{VFS}}$. Then Z updates the *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$ and hands it to $\mathcal{A}_W$.
6. When $\mathcal{A}_W$ outputs a target file o and a user u, where $u \notin Cr$ and u has the read permission of o, $\mathcal{Z}$ activates u with $(\texttt{Read}, sid, o)$ to obtain an output m. If $T[o] \neq \texttt{adv}$ and $T[o] \neq m$, $\mathcal{Z}$ outputs 1; otherwise it outputs 0.

We now discuss $\mathcal{Z}$'s outputs in the two worlds separately. In the case that $\mathcal{Z}$ interacts with real-world execution of $\Pi_{\mathcal{CRBAC}}$ and $\mathcal{D}$, from $\mathcal{A}_W$'s perspective, $\mathcal{Z}$'s simulation is indistinguishable from the real experiment. Therefore, by assumption $\mathcal{A}_W$ should have written some valid content to the file system without having the permission with non-negligible probability. Hence $\mathcal{Z}$ will output 1 with the same probability.

If $\mathcal{Z}$ interacts with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and $\mathcal{S}$, we show that $\mathcal{Z}$ will always output 0 since $\mathcal{A}_W$ can never win in this case. First recall that, to win the write security experiment, the following two conditions must hold when $\mathcal{A}_W$ terminates with an output (o, u): (1) $T[o]$ must not equal to the special value `adv` (the experiment maintains an invariant that if there exists any corrupt user has the write permission of some file, the related record in T must be set as `adv`) and (2) the current content of o (read by u) must be different from the record in $T[o]$. Next we discuss that the above two winning conditions cannot be both satisfied when $\mathcal{A}_W$ generates its output.

Suppose that condition (1) holds when $\mathcal{A}_W$ outputs (o, u). Since $T[o] \neq \texttt{adv}$, $T[o]$ can only be one of the two possible values, either an empty value or the content written by the last operation to o by some honest user who has the write permission (otherwise $\mathcal{Z}$ will not record that in $T[o]$). In the former case, $T[o]$ is an empty value implies that $\mathcal{Z}$ has not yet handled any write request to o since the recent initialization of o (o might have been deleted before but it is added back to the system later). Therefore, the value of $FS[o]$ in $\mathcal{F}_{\text{CRBAC}}$ would be also an empty value. From the specification of $\mathcal{F}_{\text{CRBAC}}$, it is clear that when $\mathcal{Z}$ activates u with the input $(\texttt{Read}, sid, o)$, it will obtain the content stored in $FS[o]$ which is the empty value here. Thus we have, the content read by u must equal to the record in $T[o]$ in this case. For the other possibility, if $T[o]$ equals to the content m which is written by the last write operation to o by some honest user, $\mathcal{Z}$ should have activated that user with an input $(\texttt{Write}, sid, o, m)$ when $\mathcal{A}_W$ requests for this write operation. Once $\mathcal{F}_{\text{CRBAC}}$ receives such an input, it stored m in $FS[o]$. Then when $\mathcal{Z}$ activates the specified user u with an input $(\texttt{Read}, sid, o)$, $\mathcal{F}_{\text{CRBAC}}$ will always return m in this case, Thus the content read by u also equals to the record in $T[o]$.

So, if $T[o] \neq \texttt{adv}$, $T[o]$ must equal to the content read by the user u, which means the two winning conditions can never be both satisfied. Therefore $\mathcal{A}_W$ can never win in the experiment and $\mathcal{Z}$ outputs 1 with probability 0.

Finally, it can be concluded that $\mathcal{Z}$'s outputs in the two worlds differ by a non-negligible amount, which means $\Pi_{\mathcal{CRBAC}}$ does not securely realize $\mathcal{F}_{\text{CRBAC}}$ and the theorem is proved.

C Proof of Theorem 2

The proof idea of this theorem is analogous to Theorem 1's. We show that if $\mathcal{CRBAC}$ is not secure with respect to read accesses, then the associated protocol $\Pi_{\mathcal{CRBAC}}$ does not securely realize $\mathcal{F}_{\text{CRBAC}}$. Let $\mathcal{A}_R$ be an adversary that breaks read security of $\mathcal{CRBAC}$. By definition, $\mathcal{A}_R$ is supposed to be able to predict the random bit chosen in $\mathbf{Exp}^{\text{read}}_{\mathcal{CRBAC},\mathcal{A}_R}$ correctly with probability significantly better than 1/2. Then an environment $\mathcal{Z}$ can be constructed from it to tell its interactions with parties running $\Pi_{\mathcal{CRBAC}}$ and a dummy adversary $\mathcal{D}$, from the interactions with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ with a simulator $\mathcal{S}$ with non-negligible probability.

$\mathcal{Z}$ works as follows. It maintains three lists: *fs*, *Cr* and *Ch* to record the current state of the file system, corrupt users and the challenge files respectively. Initially, $\mathcal{Z}$ selects a random bit $b \leftarrow_\$ \{0,1\}$ and activates the manager with an input $(\texttt{Initialization}, sid, R)$, where *sid* is an arbitrary string and R is a random set of roles. $\mathcal{Z}$ then obtains a sequence of messages regarding the changes to the file system from $\mathcal{F}_{\text{VFS}}$ (via $\mathcal{D}$), and then updates *fs* accordingly to make it identical to the list L maintained by $\mathcal{F}_{\text{VFS}}$. After that, $\mathcal{Z}$ runs a local copy of $\mathcal{A}_R$ and hands it *fs*. Then $\mathcal{Z}$ simulates $\mathbf{Exp}^{\text{read}}_{\mathcal{CRBAC},\mathcal{A}_R}$ by answering $\mathcal{A}_R$'s queries as follows.

1. When $\mathcal{A}_R$ asks for executing any RBAC command *cmd* with arguments *arg*, $\mathcal{Z}$ first checks if the execution of the RBAC command will lead to any user in the list *Cr* can get read access to any file in the list *Ch*. If this is the case, $\mathcal{Z}$ returns an error; otherwise, $\mathcal{Z}$ activates M with an input $(\texttt{RBAC}, sid, cmd, arg)$ and updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$. Then $\mathcal{Z}$ hands *fs* to $\mathcal{A}_R$. If M sends update messages to the corrupt users when executing the command, $\mathcal{Z}$ receives the messages (via $\mathcal{D}$) and hands them to $\mathcal{A}_R$. If $\mathcal{A}_R$ requests to delete some user which is in the list *Cr*, then $\mathcal{Z}$ removes this user from *Cr* after executing the command. Similarly, if $\mathcal{A}_R$ requests to delete any object which is in the list *Ch*, then $\mathcal{Z}$ also removes the object from *Ch*.
2. When $\mathcal{A}_R$ requests an honest user u to write some content m to a file o, $\mathcal{Z}$ returns an error if u does not have the write permission of o; otherwise, $\mathcal{Z}$ activates u with an input $(\texttt{Write}, sid, o, m)$, and then updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$. Then $\mathcal{Z}$ hands *fs* to $\mathcal{A}_R$.
3. When $\mathcal{A}_R$ asks for corrupting a user u, if u has the read permission of any file in the list *Ch*, $\mathcal{Z}$ then returns an error; otherwise $\mathcal{Z}$ corrupts u (via $\mathcal{D}$) and returns the local state to $\mathcal{A}_R$. Then it adds u to *Cr*.
4. When $\mathcal{A}_R$ specifies a challenge file o with two messages (m_0, m_1) and a user u, if u does not have the write permission of o or there exists some corrupt user that has the read permission of o, $\mathcal{Z}$ returns an error; otherwise, $\mathcal{Z}$ activates u with an input $(\texttt{Write}, sid, o, m_b)$ and adds o to *Ch*. Then $\mathcal{Z}$ updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$ and hands *fs* to $\mathcal{A}_R$.
5. When $\mathcal{A}_R$ queries the current state of the file system, $\mathcal{Z}$ hands *fs* to $\mathcal{A}_R$.
6. When $\mathcal{A}_R$ requests to update the file system with some information *info*, if the list *Cr* is empty, $\mathcal{Z}$ then ignores the request; otherwise $\mathcal{Z}$ chooses a user u from *Cr* and sends u a message (via $\mathcal{D}$) to let it provide an input $(\texttt{Write}, sid', o, c)$ to $\mathcal{F}_{\text{VFS}}$, where sid' is the session id of $\mathcal{F}_{\text{VFS}}$. Then $\mathcal{Z}$ updates *fs* according to the messages received from $\mathcal{F}_{\text{VFS}}$ and hands it to $\mathcal{A}_R$.
7. Finally, when $\mathcal{A}_R$ outputs a guess of the random bit b', $\mathcal{Z}$ outputs 1 if $b = b'$ or 0 otherwise, and then halts.

From the description above, we can infer that if $\mathcal{Z}$ is interacting with the real-world execution of $\Pi_{\mathcal{CRBAC}}$ and $\mathcal{D}$, from the perspective of $\mathcal{A}_R$, the simulation is identical to the real experiment. Therefore, $\mathcal{Z}$ will output 1 with the probability non-negligibly better than 1/2, since $\mathcal{A}_R$ is assumed to be able to predict the random bit b correctly with probability significantly better than 1/2.

It remains to discuss the case that $\mathcal{Z}$ is interacting with the ideal process for $\mathcal{F}_{\text{CRBAC}}$ and $\mathcal{S}$. Since $\mathcal{F}_{\text{CRBAC}}$ guarantees that only the users who have the read permission of a file can read the content of that file, the only way that $\mathcal{A}_R$ can learn some partial information about the contents written to the challenge files is to via the users who have read access to those files. Notice that $\mathcal{Z}$ proceeds the simulation strictly according to $\mathbf{Exp}^{\text{read}}_{\mathcal{CRBAC},\mathcal{A}_R}$ such that no corrupt user can be granted the read permission of any challenge file. On the other hand, $\mathcal{A}_R$ does not have the ability to let any honest user to retrieve the file contents for him. Hence it is clear that $\mathcal{A}_R$ would not be able to deduce which of the specified contents have been written to the challenge files and therefore the best it can do is to output a random guess. Thus, $\mathcal{Z}$ outputs 1 with probability exactly 1/2 in this case.

Finally, we can conclude that $\mathcal{Z}$'s outputs in the two worlds differ by a non-negligible amount, which means $\Pi_{\mathcal{CRBAC}}$ does not securely realize $\mathcal{F}_{\text{CRBAC}}$ and the theorem is proved.