2015 IEEE 28th Computer Security Foundations Symposium

Policy Privacy in Cryptographic Access Control

Anna Lisa Ferrara
University of Surrey, UK
a.ferrara@surrey.ac.uk

Georg Fuchsbauer
IST Austria
georg.fuchsbauer@ist.ac.at

Bin Liu
University of Bristol, UK
bin.liu@bristol.ac.uk

Bogdan Warinschi
University of Bristol, UK
bogdan@cs.bris.ac.uk

***Abstract*—Cryptographic access control offers selective access to encrypted data via a combination of key management and functionality-rich cryptographic schemes, such as attribute-based encryption. Using this approach, publicly available meta-data may inadvertently leak information on the access policy that is enforced by cryptography, which renders cryptographic access control unusable in settings where this information is highly sensitive.**

We begin to address this problem by presenting rigorous definitions for policy privacy in cryptographic access control. For concreteness we set our results in the model of Role-Based Access Control (RBAC), where we identify and formalize several different flavors of privacy; however, our framework should serve as inspiration for other models of access control. Based on our insights we propose a new system which significantly improves on the privacy properties of state-of-the-art constructions. Our design is based on a novel type of privacy-preserving attribute-based encryption, which we introduce and show how to instantiate.

We present our results in the context of a cryptographic RBAC system by Ferrara et al. (CSF'13), which uses cryptography to control read access to files, while write access is still delegated to trusted monitors. We give an extension of the construction that permits cryptographic control over write access. Our construction assumes that key management uses out-of-band channels between the policy enforcer and the users but eliminates completely the need for monitoring read/write access to the data.

I. Introduction

The use of system monitors to enforce access control impacts scalability and deployability: monitors are single points of failure that need to run in protected mode, be permanently online and deal with all access requests of users. Furthermore, this model is simply not suitable for the ubiquitous cloud-based paradigm, where data resides on remote, untrusted storage unsuitable for hosting a trusted monitor.

Cryptography can alleviate these problems. The idea is to give users (quasi) unrestricted access to the data, but in encrypted form. The desired access restrictions (i.e. who can access what) are then enforced by providing the appropriate keys to the right parties. This approach has many advantages: threshold-cryptographic techniques allow for the distribution of trust, key management can be done off-line, and only minimal access restrictions (such as measures against DoS attacks) need to be implemented.

Cryptographic access control raises however some subtle issues, which are best explained starting from the simplified context of standard public-key encryption schemes. Here, classic notions of provable security, such as semantic security under chosen plaintext [22] (or ciphertext [36]) attacks, ensure that ciphertexts do not reveal any information about their underlying plaintext. These notions do not guarantee however that an observer cannot link a ciphertext with its intended recipient (i.e. the owner of the public key used to create this ciphertext). This is not surprising, as plaintext privacy is the central requirement, whereas hiding the recipient is an orthogonal (and often inconsequential) security concern. Yet, there are settings (e.g. communication via ciphertexts on a public bulletin board) where this recipient information is sensitive and needs to be hidden. The first rigorous treatment of this problem was proposed by Bellare et al. [5] in the form of *key-hiding*, a security notion that ensures that ciphertexts and public keys are not linkable. They showed that many practical schemes (e.g. ElGamal [17] and Cramer-Shoup [14]) provably satisfy this property by default (or can be easily tweaked to do so) whereas others, most notably RSA-based ones, do not.

Cryptographic access control raises similar but more general privacy concerns. Here users have access to a repository of encrypted files (much like a public bulletin board) but within a more dynamic environment (where users come and go, files are added and removed, etc.). The following example outlines a simple potential failure.

Consider a hypothetical conference reviewing system which uses a cloud provider to store the conference submissions. To protect the submissions from the cloud and yet allow reviewers to access them, the chair could encrypt them under the public keys of the assigned reviewers. If encryption is not key-hiding, a PC member (who has access to the encrypted storage) may be able to tell who the reviewers of his own submissions are. An obvious fix would be to use a secure key-hiding scheme. While intuitively the attack described above does not apply anymore, there are no guarantees that other, more subtle attacks are not possible against the overall security of the system.

Such guarantees may be critical in areas where privacy is mandated by law or regulations (e.g. application areas like healthcare, private enterprises, banking) where not only raw data but also who can access it may be highly sensitive. For instance, within a hospital it is not desirable to leak if a patient's chart can be accessed by an oncologist, AIDS specialist or psychiatrist; in an enterprise one may want to hide if and which accounting files are readable by an auditor, and which personnel files are accessible by a redundancy expert.

In these areas strong security guarantees are clearly required. Unfortunately, formal security notions that would allow to rigorously prove that no information is revealed when a security policy is enforced via cryptographic access control are currently missing. We begin to address this problem. We propose formal foundations in the form of definitions that capture the privacy of several distinct aspects of cryptographically enforced access-control policies. To the best of our knowledge, our work is the first rigorous approach to policy privacy; we expect that even though the definitions we propose are for the RBAC model, our approach will serve as inspiration in other

DOI 10.1109/CSF.2015.11

similar contexts.

Policy privacy can be achieved through designs that are costly (in terms of storage and bandwidth). For example, a simple way of hiding who has access to each encrypted file uses an implementation dual to the one outlined above for the reviewing system. Namely, associate a fresh public encryption key to each file and provide the decryption key to every user that should access it. Assuming that key management is done out-of-band, it is easy to see that the access policy is information-theoretically hidden. This implementation works well for quasi-static systems but raises scalability concerns for more dynamic ones. If users, files or access permissions are often added or deleted, the associated key-management operations may be prohibitive. The creation of each file, or revoking access to a file requires the generation of a new public key associated to the file and passing the corresponding decryption key to all of the users that should have access to it; identifying the "correct" set of recipients may also be challenging.

The appeal of schemes like the one just outlined can quickly vanish when they are modified to support dynamic changes more efficiently, as such even apparently benign changes could lead to new information leaks. For example, a simple optimization (inspired by how RBAC simplifies access-control management) is to group users into "roles", associate a public key to each role and provide the corresponding secret key to the users having that role. Each new file can now be encrypted with a symmetric key, which in turn is encrypted under the public keys of the roles with access permission to it; users with such a role can access a file by doing two decryptions: one to obtain the symmetric key and a second to obtain the file. This modification starts to leak information about the access structure: e.g., anyone with access to the encrypted storage can observe how many roles have access to a certain file. An additional level of indirection, where a public key is associated to a subset of roles would patch this leak but open a new one: a user that can create files would need to know the roles that can access that file. Other variants of the basic design that simplify and optimize key management by using attribute-based encryption (ABE) [37] may leak even more information, as we discuss below.

CRYPTOGRAPHIC RBAC. Instead of presenting our definitions and results on formally provable policy privacy in the context of ad hoc schemes, like those above, we do so in the context of RBAC systems. Before detailing our results, we recall the framework for cryptographic role-based access control (cRBAC) introduced by Ferrara et al. [19].

RBAC is a framework that simplifies access management by introducing roles: users are assigned roles, and roles are assigned permissions (such as read and write access to objects). The user-object access-control matrix is immediate: a user u has permission p if he has some role r which has had assigned permission p. RBAC commands allow adding and deleting users, roles and objects, and assigning/deassigning users to roles and roles to permissions.

Informally, a cRBAC consists of algorithms that implement the RBAC commands. A secure cRBAC ensures that parties can access (in a cryptographic sense) only the data they are entitled to according to the access-control matrix associated with the system.

In addition to syntax and security models, [19] provides a construction based on attribute-based encryption [37], [24].

In brief, ABE works as follows: messages are encrypted w.r.t. sets of attributes and an authority which set up the system parameters can derive keys corresponding to predicates (or policies) on attributes. Such a key can then decrypt any ciphertext whose attributes satisfy the predicate (policy) associated to the key. Converesely, one may associate attributes to keys and policies to ciphertexts so that only keys whose attributes satisfy the policy associated to the ciphertext can successfully decrypt it. ABE thus assumes a central authority that handles the key distribution (which e.g. in RBAC will be the manager of the file system).

Very roughly, the cRBAC implementation of [19] assigns an attribute a_r to each role r. If I is the set of roles that have read permission to some file f then f is encrypted under the attributes in I. Users are provided with decryption keys that correspond to the attributes of the roles the users are assigned.

Since the roles associated to files are public, it is immediate that the implementation sketched above reveals information about the access structure. Therefore, it is a good concrete implementation for which we will show how to achieve policy privacy and prove that it satisfies our formal provable-security definitions. One first step is to hide the mapping between roles and files from users (as the mapping is only needed to encrypt files under their appropriate attributes). However, this works only in settings where solely trusted parties (e.g. the monitor) are allowed to write to files. As soon as the system is extended to allow users to write to files (as we do in this paper), users need to know which roles have access to files in order to encrypt them under the right attributes. We show how to provide this ability in a policy-privacy-respecting manner through our new cryptographic primitive. We detail our contributions next.

A. Our Contributions

PRIVACY IN CRYPTOGRAPHIC ACCESS CONTROL. Our main contribution are rigorous security definitions for privacy of access-control policies. We view these as analogous to those by Bellare et al. [5] in the context of public-key encryption. Standard security notions for public-key encryption do not ensure that an observer cannot link a ciphertext with its intended recipient. Similarly, there are no formal security notions that allow to formally prove that no sensitive information on the access policy is revealed when the latter is enforced via cryptographic access control. Much like Bellare et al. provide a formal framework for proving that ciphertexts and public keys are not linkable, our definitions also enable mathematical proofs that an access-control system does not reveal sensitive information about the access policy.

We do not provide a single security model, but chose to allow flexibility in classifying as sensitive the various pieces of information that form the access-control structure. A benefit of this approach are pragmatic trade-offs: designers can choose to forgo the privacy of some aspects of the access-control system deemed less important in order to gain efficiency.

NEW PRIVACY-PRESERVING ENCRYPTION. Previous work eliminates the need for actively monitoring read access to files but still requires a trusted party to deal with write requests. In this paper we go one step further and propose an access

$\mathit{AddUser}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, u such that $u \notin U$ is a user.
Output: $(U \cup \{u\}, O, P, \mathit{UA}, \mathit{PA})$.

$\mathit{DelUser}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, u such that $u \in U$.
Output: $(U \setminus \{u\}, O, P, \mathit{UA} \setminus \{(u, r) \in \mathit{UA} \,|\, r \in R\}, \mathit{PA})$.

$\mathit{AddObject}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, o such that $o \notin O$ is an object.
Output: $(U, O \cup \{o\}, P \cup \{(o, \texttt{read}), (o, \texttt{write})\}, \mathit{UA}, \mathit{PA})$.

$\mathit{DelObject}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, o such that $o \in O$.
Output: $(U, O \setminus \{o\}, P \setminus \{(o, \texttt{read}), (o, \texttt{write})\}$,
$\mathit{UA}, \mathit{PA} \setminus \{(p, r) \in \mathit{PA} \,|\, p \in \{(o, \texttt{read}), (o, \texttt{write})\}, r \in R\})$.

$\mathit{AssignUser}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, (u, r) such that $u \in U$ and $r \in R$.
Output: $(U, O, P, \mathit{UA} \cup \{(u, r)\}, \mathit{PA})$.

$\mathit{DeassignUser}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, (u, r) such that $u \in U$ and $r \in R$.
Output: $(U, O, P, \mathit{UA} \setminus \{(u, r)\}, \mathit{PA})$.

$\mathit{GrantPerm}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, (p, r) such that $p \in P$ and $r \in R$.
Output: $(U, O, P, \mathit{UA}, \mathit{PA} \cup \{(p, r)\})$.

$\mathit{RevokePerm}$
Input: $(U, O, P, \mathit{UA}, \mathit{PA})$, (p, r) such that $p \in P$ and $r \in R$.
Output: $(U, O, P, \mathit{UA}, \mathit{PA} \setminus \{(p, r)\})$.

Fig. 1. RBAC symbolic implementation.

control system where access to the storage is quasi-unrestricted and the monitor is only in charge of the key-management operations. The real challenge in an access-control system that allows users to write to files is to ensure the privacy of the access policies. We have explained the problem in the context of the ad hoc constructions briefly discussed above; here we revisit the issue in the context of the construction from [19]. There, each file is encrypted using attribute-based encryption. The attributes under which a file is encrypted correspond to the roles that have read access to it. Ciphertexts may explicitly reveal their attributes (and therefore the roles), since standard security notions for ABE only guarantee secrecy of the encrypted message. Furthermore, in any system where users have the right to write to files, they would need to know the attributes under which to encrypt the data, meaning these users explicitly need access to the role-permission matrix.

Existing privacy notions for ABE address the first type of leak: predicate encryption [29] was introduced as a (key-policy) ABE whose ciphertexts hide their attributes. However, there are no ABE schemes that can deal with the second type of privacy break. To do so, parties would need a way to encrypt a message w.r.t. a set of attributes without knowing what those attributes are. Note that even functional encryption [20] does not help in this context.

We provide a solution to this problem. We propose a variant of predicate encryption, where one can generate a key pk_I for any set of attributes I, so that pk_I hides the elements of I. Ciphertexts encrypted under pk_I can be decrypted by users with a secret key for a predicate/policy that holds on I. For this new primitive we give security models and a generic construction, which we show how to instantiate concretely.

CONSTRUCTION. We employ the primitive that we design to construct a cryptographic RBAC which strengthens that from [19] in the two general directions we study in this paper. First, we show that our proposal securely enforces read and write access control. We prove that the scheme meets some of the policy-privacy properties that we put forth in this paper. Stronger properties can also be achieved via more expensive implementations but we argue that the privacy properties of our scheme suffice for practical purposes. In particular, if employed in the motivating scenario of RBAC-controlled access to hospital files, our solution would hide the specialty of a doctor that can access a patient's file.

We remark that the main contributions of our paper are rigorous foundations for policy privacy in cryptographic access control and the introduction of a new encryption primitive. The schemes that we propose are not efficient and should be regarded as a proof of concept showing that meaningful levels of policy privacy can be achieved. We leave the development of efficient schemes for future work.

B. *Related work*

The rich literature on the interplay between cryptography and access control ranges from schemes that employ encryption and/or key-distribution schemes to implement various forms of hierarchical access control [21], [3], [30], [4], [16] to the design and implementation of cryptographic file systems [27], [7], [11], [26] and on to the recent cryptographic primitives motivated by access control [37], [24], [28], [31], [12], [6]. However, while the syntax, functionality and security of the primitives (e.g. attribute-based encryption) is usually motivated by their use in access-control applications, the step that confirms that the primitive suffices to ensure the functionality and security of the system is seldom carried out.[1]

Policy privacy in the context of ABE, such as the idea that a ciphertext does not reveal its associated policy, has been studied to some extent, yielding constructions that meet stronger security notions [10], [31], [29], [32], [33]. However, none of the existent policy-privacy notions for encryption suffices to hide what access policies are in place when the schemes are used to enforce access control. No obvious fix to this problem is available, short of developing new cryptographic primitives with stronger privacy guarantees, which is one of our results.

Cryptographic enforcement of RBAC policies was previously considered in [15], [41]; Ferrara et al. [19] introduced the first formal security model for cryptographic RBAC systems and our work continues their line of research. They also give a cRBAC construction where cryptography controls read access to files and write access is delegated to trusted monitors. We extend theirs by letting cryptography also control write access.

[1]The only notable examples where this gap is bridged are [2], [25], [19].

The problem of enforcing write access to files by using cryptography has been previously considered in the design of cryptographic file systems such as Plutus [27]. Unlike our construction, Plutus requires the server to validate any write access in order to prevent malicious users from making changes to the file system. We instead propose the use of *versioning* storage, where users can only append information but not delete any. This assumption is common in distributed systems (it has been used for example to deal with poisonous write operations [23]) and is warranted by existing systems (like the Google, Microsoft, or Amazon cloud storage).

II. Cryptographic RBAC

The notion of a cryptographic RBAC (cRBAC) was formally modeled in [19], where read access to a file system is handled using cryptography, while write access is mediated by the manager. We extend their notion by allowing authorized users to execute write operations on files, thereby avoiding the use of online monitors for both read and write operations. In the following, we first recall the RBAC model and then define the notion of a cryptographic RBAC with respect to write accesses.

A. Role-Based Access Control

RBAC reduces the complexity in administration of user permissions by grouping users into roles and assigning permissions to each role [1], [18], [38]. A permission is an object-action pair that enables access of a user to an object in a particular mode. We consider RBAC policies for file systems where objects are files and the access mode belongs to the set $\{\text{read, write}\}$. A user u has permission p if there exists a role r such that u is assigned to r and r has permission p.

We follow [19] in that we consider a fixed set of roles R (reflecting the fact that the structure of any organization is usually stable). Formally, an RBAC system for the set of roles R is a tuple (U, O, P, UA, PA), where U, O and P are sets of users, objects and permissions, respectively. Permissions $p \in P$ are of the form $(o, \texttt{read})$ or $(o, \texttt{write})$ with $o \in O$. The relation $UA \subseteq U \times R$ is the *user-role assignment* relation, and $PA \subseteq P \times R$ is the *permission-role assignment* relation. A pair $(u, r) \in UA$ means that user u belongs to role r, and $(p, r) \in PA$ means that role r has permission p. A user u is authorized for permission p if there exists a role $r \in R$ such that $(u, r) \in UA$ and $(p, r) \in PA$.

Administrative RBAC. Administrative commands allow to add and delete users and objects, grant and revoke permissions and modify users' role membership. Fig. 1 shows a symbolic implementation of these commands. Each command has the form

$$(U', O', P', UA', PA') \leftarrow Cmd((U, O, P, UA, PA), arg) \enspace ,$$

where Cmd is either $AddUser$, $DelUser$, $AddObject$, $DelObject$, $AssignUser$, $DeassignUser$, $GrantPerm$, or $RevokePerm$, and arg specifies the argument of the command.

Security policies. The design of an access-control policy includes the specification of *security policies*, which express limits on the access of users to permissions (e.g. users belonging to role r cannot be granted permissions associated to role $\tilde{r}$). We consider the class of RBAC security policies identified in [19], which is defined as follows:

Definition 1 (Security Policy [19]): Let $\mathsf{HasAccess}(u, p)$ be a predicate that reflects when a user u symbolically has access to a permission p:

$$\mathsf{HasAccess}(u, p) \;\Leftrightarrow\; \exists\, r \in R : (u, r) \in UA \land (p, r) \in PA \enspace .$$

A *Security Policy* Φ is a formula of the following form:

$$\forall u \in U \;\forall p \in P : \mathsf{Cond}(u, p) \;\Rightarrow\; \neg\mathsf{HasAccess}(u, p) \enspace ,$$

where $\mathsf{Cond}(u, p)$ is a predicate over a user $u \in U$ and a permission $p \in P$.

An example of a security property captured by this definition is mutual exclusion of read/write permissions on a file. This can be formulated as: $\forall u \in U \;\forall p \in P$: $\mathsf{HasAccess}(u, \bar{p}) \Rightarrow \neg\mathsf{HasAccess}(u, p)$, where $\bar{p}$ is $(o, \texttt{write})$ ($(o, \texttt{read})$, resp.) if p is $(o, \texttt{read})$ ($(o, \texttt{write})$, resp.).

The class of security policies identified in Def. 1 captures many policies of interest, such as *separation of duties* and *privilege escalation* (see [19] for details).

B. System Model

Before we explain the computational implementation of a cRBAC system, we introduce the architecture of the underlying system, which we contrast with that of traditional access-control systems. In the latter, data is placed on a trusted server and access to it is mediated by a gateway in charge of enforcing the access-control policy. In particular, the monitor needs to be involved both in the administrative actions (adding and removing users to the system, changing permissions, etc) *and* in read and write access to data.

In a recent paper, Ferrara et al. [19] propose a system where data is stored in encrypted form on an untrusted storage server to which all of the users have unrestricted read access. Data management is still performed by an external trusted party called in that paper *a manager*, a terminology that we also adopt. The manager is in charge of implementing the administrative commands: he performs actions that have the effect of adding users to the system, granting or revoking permissions, etc. These actions are implemented as combinations of key-management operations and encrypting/re-encrypting data. The net effect is that the manager does not need to mediate read access to files anymore. However, he must ensure that the write operation to files is in accordance with the access policy, so he is still involved in these.

In this paper we show how to further reduce the dependency on policy-enforcing monitors. Concretely, we extend the cRBAC system of [19] to a setting where the users have (quasi) unrestricted read/write access to the file system and the manager is now only in charge of the administrative commands. We need to rely on minimal storage guarantees to prevent malicious users from simply overwriting files. We propose using *versioning* storage where users can only append information but not delete any. These appends are then interpreted as logical writes to files.[2] Of course, such a system requires dealing with issues specific to distributed systems e.g. atomicity of the actions that are permitted, efficiency implementations (e.g. the use of log-structured techniques [39], [40]), but an in-depth discussion of these issues is beyond the scope of this paper.

[2] As observed elsewhere [11], [26], this can be easily implemented on top of a log-structured file system; see also [13] for techniques that enforce append-only semantics in a storage system.

```
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
                $T[o] \leftarrow \mathtt{adv}$
    For all $u \in U \setminus Cr$:
        $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
    Return $(fs, \{msg_u\}_{u \in Cr})$

WRITE(u, o, m)
    If $u \in Cr$
        then Return $\perp$
    If $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$ then
        Return $\perp$
    $fs \leftarrow\!\$\ \mathsf{Write}(st[u], fs, o, m)$
    For all $u' \in Cr$:
        If $\mathsf{HasAccess}(u', (o, \mathtt{write}))$ then
        Return $fs$
    $T[o] \leftarrow m$
    Return $fs$

CORRUPTU(u)
    If $u \notin U$ then Return $\perp$
    For all $o \in O$:
        If $\mathsf{HasAccess}(u, (o, \mathtt{write}))$ then
            $T[o] \leftarrow \mathtt{adv}$
    $Cr \leftarrow Cr \cup \{u\}$; Return $st[u]$

FS(query)
    If $query =$ "STATE" then
        Return $fs$
    If $query =$ "APPEND($info$)" then
        $fs \leftarrow fs \| info$; Return $fs$
```

Fig. 2. Oracles for defining the experiment $\mathbf{Exp}^{\text{write}}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}$ that defines security of access control with respect to the write permission.

Very roughly, given such storage, "write" operations are physically realized by appending data to the file system (i.e., users post the updated versions of a file). Rather than having a reference monitor in charge of verifying each user's authorization for each write access, users only check the validity of a file version during read operations. To perform a read operation, a user fetches the latest version of the requested file and determines whether it has been uploaded by an authorized user. If not, he fetches the preceding version until a valid version is encountered. Of course, additional care needs to be taken for users not to write to files they are not entitled to, reverse the content of a file to a previous state by copying, etc; issues that will be addressed by our instantiation.

C. Cryptographic RBAC for Write Access (w-cRBAC)

A w-cRBAC scheme involves a set of users, an untrusted publicly accessible file system, and a trusted party called manager. While the manager is responsible for carrying out all administrative RBAC commands, it is not involved in read/write operations, which are executed by users only. The global state of a w-cRBAC consists of the state of the manager and that of each user u, respectively denoted by st_M and $st[u]$.

A w-cRBAC scheme $w\text{-}\mathcal{CRBAC}$ is defined by a tuple of algorithms Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, Update, Write and Read.

The initialization algorithm Init takes as input the security parameter λ and the set of roles R. It outputs the initial states of the file system fs and of the manager st_M. The other algorithms, except Update, Read and Write, implement the RBAC administrative commands defined in Fig. 1.

More precisely, we regard each of these algorithms as a non-interactive multi-party computation where the manager, after some local computations, sends messages to the other parties, who update their states using the algorithm Update. Specifically, the administrative algorithms take as input the state of the manager st_M, the file system fs and the argument for the command, and output the updated fs, messages msg_u for every user u and a possibly updated state for the manager. Like Update, the Write and Read operations are algorithms executed by the users. Update updates a user's local state according to the message received from the manager. The algorithm Read allows authorized users to retrieve the current content of a file. It takes as input the state of a user $st[u]$, the file system fs and a file name o and, provided that u has read access to o, returns the content of the latest valid version of o. If not or if the file is empty, the algorithm returns $\perp$. Finally, the algorithm Write allows a user to write content to a file. It takes as input the state of a user $st[u]$, the file system fs, a file name o and content m. If user u has write access to o, u uploads a new version of file o containing m.

W-CRBAC SECURITY. A security definition for a cryptographic RBAC with respect to read operations was introduced in [19]. A cRBAC is secure w.r.t. read accesses if whenever a user u is not authorized to read a file o then u should not be able to deduce anything about the content of o. This was formalized via an indistinguishability-based definition: an adversary, who can impersonate users, chooses two messages m_0 and m_1, one of which is randomly selected and written to a file o of the adversary's choice; the adversary must then determine which of the two messages was written to o.

In the following, we introduce a security definition for write accesses. Intuitively, a cryptographic implementation of a w-cRBAC is secure with respect to write accesses if any user who is not authorized to write to a file o is indeed not able to do so. We formalize this intuition via a game between the manager of a w-cRBAC implementation and a polynomial-time adversary $\mathcal{A}$. The adversary can ask the manager to execute any of the w-cRBAC commands, is allowed to corrupt any user u, and can request an honest user u to write some content m to a file o, provided that u has write permission for o. Also, the adversary can append arbitrary content to the file system, which models public unrestricted append-only write access.

At any time in the experiment, the adversary can output a challenge file name o^* along with a non-corrupt user identity u^*. The adversary wins the game if the content of o^*, as revealed to u^* via Read, is different from the content that was written to o^* by the last honest user. Since corrupt users that have write access to o^* should be able to modify the content, we require that no corrupt user has had permission $(o^*, \mathtt{write})$ since the last valid write to o^* by an honest user. This ensures that the experiment cannot be trivially won.

The experiment considers an adversary that can ask the manager to perform arbitrary RBAC operations. The experiment maintains the RBAC's symbolic representation (U, O, P, UA, PA) as it evolves from the initial configuration $(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$. Additionally, the experiment maintains a list $Cr \subseteq U$ of corrupt users and a table T indexed by objects. For each file o, $T[o]$ is initially set to $\perp$ when o is added via the RBAC command $AddObject$. Successively, $T[o]$ will store the last content written to o by an honest user, or a special value adv. Specifically, a valid write operation is recorded in $T[o]$ only if no corrupt user has write access to o. Moreover, the experiment maintains the following invariant: whenever some corrupt user has write access to o then $T[o] = \texttt{adv}$.

Definition 2: A w-cRBAC (Init, AddUser, DelUser, AddObject, DelObject, AssignUser, DeassignUser, GrantPerm, RevokePerm, Update, Write, Read) is *secure* with respect to write accesses if for all probabilistic polynomial-time (p.p.t) adversaries $\mathcal{A}$, it holds that

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

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

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

The oracles $\mathcal{O}$ to which the adversary has access are specified in Fig. 2 (and discussed below).

The oracle CMD, where CMD can be either ADDUSER, DELUSER, ADDOBJECT, DELOBJECT, ASSIGNUSER, DEASSIGNUSER, GRANTPERM, or REVOKEPERM, allows the adversary to execute any RBAC command: the experiment first runs the command *Cmd* symbolically and then runs the algorithm Cmd implementing the same command. Moreover, the CMD oracle maintains the invariant on T.

The oracle CORRUPTU allows the adversary to corrupt a user. It returns the user's state to the adversary, adds the user to the corrupt-user list Cr and updates T appropriately. The oracle WRITE allows the adversary to request an honest user to write some content to a file o. If such a user has write access to o, it runs the Write algorithm. The new content is stored in the table T only if no corrupt user has write access to o. In order to model adversarial read and write operations, the oracle FS allows the adversary to read the entire file system and to append file versions to it.

The adversary wins the game if he outputs a non-corrupt user u^* and a file o^* such that $T[o^*] \neq \texttt{adv}$, that is, no corrupt user has had permission $(o^*, \texttt{write})$ since the last write operation by an honest user on o^*, and $T[o^*] \neq \mathsf{Read}(st[u^*], o^*, fs)$, that is, the content written to o^* by the last honest user is different from what an honest user would read.

SECURITY POLICIES ENFORCEMENT W.R.T. WRITE ACCESSES. We say that a w-cRBAC enforces a security policy, as defined in Def. 1, with respect to write accesses if any coalition of users C such that $\mathsf{Cond}(u, (o, \texttt{write})) = 1$, for each $u \in C$, is not able to write any content to o.

We formalize this via a game between an adversary and the manager of a w-cRBAC implementation who ensures that the symbolic RBAC always satisfies the security policy ϕ. The adversary is allowed to ask the manager for the execution of any RBAC command, request an honest user to write some content to a file and corrupt any user. The experiment terminates when the adversary outputs a file name o^* and the identity u^* of an honest user who has read access to o^* and it wins the game if no corrupt user has had permission $(o^*, \texttt{write})$ at least since the last valid write operation on o^*, the content written by such an operation is different from the one stored in $T[o^*]$, and every corrupt user u satisfies $\mathsf{Cond}(u, (o^*, \texttt{write}))$.

Definition 3: For any security policy ϕ of the form

$$\forall u \in U\ \forall p \in P : \mathsf{Cond}(u, p) \ \Rightarrow\ \neg\mathsf{HasAccess}(u, p)\ ,$$

a w-cRBAC securely enforces ϕ with respect to write accesses if for all p.p.t. adversaries $\mathcal{A}$,

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

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

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

The oracles $\mathcal{O}_\phi$ are the same as those of Fig. 2 with the exception of CMD, shown in Fig. 3, which executes the corresponding RBAC command only if the resulting access-control policy does not violate the security policy ϕ.

Since the winning condition of $\mathbf{Exp}^{\text{write-}\phi}_{w\text{-}\mathcal{CRBAC}}$ implies that of the experiment in Def. 2, it is easy to see that the following theorem holds.

Theorem 1: If $w\text{-}\mathcal{CRBAC}$ is secure with respect to write accesses then for any security policy ϕ, $w\text{-}\mathcal{CRBAC}$ securely enforces ϕ with respect to write accesses.

Although the proof of Theorem 1 is immediate, we notice that the definition of a security policy being computationally satisfied is an important step to provide a rigorous approach to the analysis of cryptographic enforcement of access control. The fact that the notion of security policy of Def. 1 models requirements of users not having write permissions makes it

```
$\underline{\text{CMD}(arg)}$
  $(U', O', P', UA', PA') \leftarrow Cmd((U, O, P, UA, PA), arg)$
  If $(U', O', P', UA', PA')$ satisfies $\phi$ then
    $(U, O, P, UA, PA) \leftarrow (U', O', P', UA', PA')$
  Else Return $\perp$
  $(st_M, fs, \{msg_u\}_{u \in \mathcal{U}}) \leftarrow\!\$\ \mathsf{Cmd}(st_M, fs, arg)$
  For all $u \in Cr$:
    For all $o \in O$:
        If $\mathsf{HasAccess}(u, (o, \texttt{write}))$ then $T[o] \leftarrow \texttt{adv}$
  For all $u \in U \setminus Cr$:
    $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
  Return $(fs, \{msg_u\}_{u \in Cr})$
```

Fig. 3. RBAC oracle for $\mathbf{Exp}^{\text{write-}\phi}_{w\text{-}\mathcal{CRBAC}}$

immediate that as long as the policy is enforced symbolically, the policy is also enforced computationally. In other settings (for example, when a larger class of policies is considered) similar relations may not exist so that proving secure enforcement of policies may be more intricate.

III. Privacy Issues in Cryptographic Access Control

The examples we gave in the introduction show that cryptographic implementations (including the one from [19]) may leak sensitive information about the access policies in place. The first step towards understanding and coping with this issue is to establish models that clearly identify the abilities of an attacker and specify what are to be considered privacy breaches.

Here we are faced with a choice. One possibility is to provide an all-encompassing, general privacy definition that would ask that an adversary not be able to distinguish among *any* changes to the access structure. For example, we could require that the adversary cannot tell if at some point a user had been added to the system, or a permission had been revoked from some role. Instead, we pursue a more nuanced approach where we identify privacy requirements separately, for various aspects of the system. This approach allows for privacy/efficiency trade-offs where designers might choose to sacrifice the privacy of some components deemed less important in order to gain efficiency. One first distinction that we make is to consider privacy of the two matrices UA and PA separately, and for each of these components we identify two further refinements. For privacy notions regarding the PA matrix we define two distinct notions: the first, p2r-privacy, models the idea that a cryptographic RBAC scheme hides the assignment that maps a permission to the roles that have it. Conversely, r2p-privacy demands that an implementation hides which permissions a certain role has. At first glance, the distinction may seem strange: if an adversary can link a role to a permission, does it not also mean that it can link permissions to roles? To understand the advantage of splitting privacy of PA in this way, consider the execution of some RBAC administrative command that involves both roles and permissions, e.g. assigning a permission to a role. In an ideal situation neither the role nor the permission involved are revealed. However, it is possible that the visible effects of the action on the file system reveal to an adversary which permission had been involved in the operation but not which role (so p2r-privacy holds but r2p-privacy does not).

Similarly, we define u2r-privacy and r2u-privacy to model that a cryptographic RBAC scheme hides the assignment of a user to her roles and which users have a certain role.

We describe our formalization of these notions below. For conciseness, we present one security game for the privacy of the information in UA and another game for PA. The four different notions are obtained as instances of these two games.

The games maintain the state of a cryptographic RBAC with which the adversary can interact through several oracles, which we give in Fig. 4.

The adversary drives the execution by issuing RBAC commands to oracle $\textsc{Cmd}(arg)$ and can corrupt arbitrary users (thereby learning their local state) using oracle $\textsc{CorruptU}(u)$ and ask an arbitrary non-corrupted user to write to an arbitrary file using oracle $\textsc{Write}(u, o, m)$. Moreover, the adversary is able to query the current state of the file system and to append information to it by calling the file-system oracle FS. Privacy of the information in PA is captured via oracle $\textsc{ChllPA}$. The adversary can call this oracle once, with one of the two RBAC commands $GrantPerm$ or $RevokePerm$ and a quadruple $(p_0, p_1, r_0, r_1) \in P^2 \times R^2$. Depending on a challenge bit b, the oracle executes the command on the pair (p_b, r_b). After that, the adversary must guess the challenge bit b. The intuition behind the definition is that an adversary that observes the execution should not learn which of the two roles and which of the two permissions were involved in the command. We obtain the two notions that capture the different flavors of privacy for matrix PA by requiring that either $p_0 = p_1$ (this restriction defines p2r-privacy) or that $r_0 = r_1$ (hence obtaining r2p-privacy). For instance, the p2r-privacy notion models the fact that a cRBAC scheme hides which roles have a certain permission by requiring that, given two roles and a permission, an adversary is not able to tell to which of the two roles the permission has been granted/revoked.

We also define a weaker notion of p2r-privacy that we call p2r*-privacy. This notion is defined just like p2r-privacy except that the adversary is only allowed to query the challenge oracle with the assign permission command. Although very simple, the p2r*-privacy notion is relevant for practical purposes. Indeed, in our motivating example of RBAC controlled access to hospital files, p2r*-privacy suffices to guarantee that granting access to the clinical record of patient X to a doctor would not reveal the specialty of the doctor.

Oracle $\textsc{ChllUA}$ serves the analogous purpose for defining privacy of the information in UA; the security notions u2r-privacy and r2u-privacy are the immediate analogs.

For the notion r2p-privacy a further refinement is meaningful. Currently, the game demands that the adversary cannot tell which of two possible permissions have been granted to (revoked from) some role r. Yet, if the two permissions are of different nature (e.g. write and read), the information may be difficult to hide. A possible refinement is to demand that the adversary cannot distinguish between permissions from some class (e.g. only write permissions). Formally, this notion is obtained by demanding that the two permissions passed to the challenge oracle come from some class $\mathcal{C}$ of permissions, which is added as a parameter to the experiment.

The following definition formalizes the different notions of privacy that we consider.

Definition 4: A scheme $w\text{-}\mathcal{CRBAC}$ preserves x-privacy, where $x \in \{\mathsf{u2r}, \mathsf{r2u}, \mathsf{p2r}, \mathsf{r2p}\}$, if for all p.p.t. adversaries $\mathcal{A}$,

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

is negligible in λ, where the experiment $\mathbf{Exp}^{\text{x-privacy}}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}$ is defined as follows:

```
$\underline{\mathbf{Exp}^{x\text{-privacy}}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}(\lambda)}$
    $b \leftarrow_{\$} \{0,1\}$
    $State \leftarrow (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)$; $Cr \leftarrow \emptyset$; $\mathsf{challd} \leftarrow 0$
    $(st_M, fs, \{st[u]\}_{u\in\mathcal{U}}) \leftarrow_{\$} \mathsf{Init}(1^\lambda, R)$
    $b' \leftarrow \mathcal{A}(1^\lambda : \mathcal{O}_x)$
    Return $(b = b')$
```

Here $\mathcal{O}_{\mathsf{u2r}}$ and $\mathcal{O}_{\mathsf{r2u}}$ consist of all oracles in Fig. 4 except $\textsc{ChllPA}$. Analogously, $\mathcal{O}_{\mathsf{p2r}}$ and $\mathcal{O}_{\mathsf{r2p}}$ are the oracles defined

```
$\underline{\textsc{Cmd}(arg)}$
    If $\mathsf{challd} = 1$ then Return $\bot$
    $State \leftarrow Cmd(State, arg)$
    $(st_M, fs, \{msg_u\}_{u \in U})$
            $\leftarrow\!\!\$\ \mathsf{Cmd}(st_M, fs, arg)$
    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 $\mathsf{challd} = 1$ then Return $\bot$
    If $u \notin U$ then Return $\bot$
    $Cr \leftarrow Cr \cup \{u\}$; Return $st[u]$

$\underline{\textsc{Write}(u, o, m)}$
    If $\mathsf{challd} = 1$ then Return $\bot$
    If $u \in Cr$ then Return $\bot$
    If $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$ then Return $\bot$
    $fs \leftarrow\!\!\$\ \mathsf{Write}(st[u], fs, o, m)$; Return $fs$

$\underline{\mathrm{FS}(query)}$
    If $query =$"STATE" then Return $fs$
    If $query =$"APPEND($info$)" then $fs \leftarrow fs \| info$; Return $fs$
```

```
$\underline{\textsc{ChllUA}^{(x)}(cmd, (u_0, u_1, r_0, r_1))}$
    If $\mathsf{challd} = 1$ or $x \notin \{\mathsf{u2r}, \mathsf{r2u}\}$
            or $(x = \mathsf{u2r} \wedge u_0 \neq u_1)$
            or $(x = \mathsf{r2u} \wedge r_0 \neq r_1)$ then
        Return $\bot$
    If $cmd \notin \{AssignUser, DeassignUser\}$
        then Return $\bot$
    If $u_0 \in Cr$ or $u_1 \in Cr$ then
        Return $\bot$
    $(st_M, fs, \{msg_u\}_{u \in U})$
            $\leftarrow\!\!\$\ \mathsf{Cmd}(st_M, fs, (u_b, r_b))$
    For all $u \in U \setminus Cr$:
        $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
    $\mathsf{challd} \leftarrow 1$
    Return $(fs, \{msg_u\}_{u \in Cr})$
```

```
$\underline{\textsc{ChllPA}^{(x)}(cmd, (p_0, p_1, r_0, r_1))}$
    If $\mathsf{challd} = 1$ or $x \notin \{\mathsf{p2r}, \mathsf{r2p}\}$
            or $(x = \mathsf{p2r} \wedge p_0 \neq p_1)$
            or $(x = \mathsf{r2u} \wedge r_0 \neq r_1)$ then
        Return $\bot$
    If $cmd \notin \{GrantPerm, RevokePerm\}$
        then Return $\bot$
    For all $u \in Cr$
        If $(u, r_0) \in UA$ or $(u, r_1) \in UA$
            Return $\bot$
    $(st_M, fs, \{msg_u\}_{u \in U})$
            $\leftarrow\!\!\$\ \mathsf{Cmd}(st_M, fs, (p_b, r_b))$
    For all $u \in U \setminus Cr$:
        $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
    $\mathsf{challd} \leftarrow 1$
    Return $(fs, \{msg_u\}_{u \in Cr})$
```

Fig. 4. Oracles for $\mathbf{Exp}^{x\text{-}privacy}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}$ where $x \in \{\mathsf{u2r}, \mathsf{r2u}, \mathsf{p2r}, \mathsf{r2p}\}$ and $State = (U, O, P, UA, PA)$. Oracle $\textsc{ChllUA}^{(\mathsf{u2r})}$ defines u2r-privacy (as $u_0 = u_1$ must hold) and $\textsc{ChllUA}^{(\mathsf{r2u})}$ defines r2u-privacy ($r_0 = r_1$). Analogously, oracles $\textsc{ChllPA}^{(\mathsf{p2r})}$ and $\textsc{ChllPA}^{(\mathsf{r2p})}$ define p2r-privacy and r2p-privacy, resp.

in Fig. 4 except ChllUA. The adversary can call the challenge oracle only once, and is not allowed further queries after this.

IV. Predicate Encryption with Specific Public Keys

Ferrara et al. [19] introduce a restricted variant of predicate encryption, which is sufficient for their implementation of cryptographic RBAC and on which we base our schemes as well. A *predicate encryption for non-disjoint sets* (PENDS) is defined over an attribute universe A; messages are encrypted w.r.t. *sets of attributes* $y \subseteq A$, and keys are associated to sets $x \subseteq A$ and can decrypt any ciphertext for y with $x \cap y \neq \emptyset$.

In predicate encryption (PE), as defined in [29], ciphertexts are associated to "identities"[3] I and keys are associated to predicates f. A key for f can decrypt a ciphertext for I if $f(I) = 1$. (By letting "identities" be sets of attributes, this immediately yields a key-policy ABE.) Any PE scheme (with sufficiently expressible predicates) can be used to instantiate PENDS by letting identities be sets of attributes $y \subseteq A$ and by associating a predicate f with a subset x and defining $f_x(y) = 1 \Leftrightarrow x \cap y \neq \emptyset$.

PE was introduced as a type of ABE that not only hides the messages but also w.r.t. which identity I (set of attributes) it was encrypted. We require additional privacy in that not even the encryptor should know this identity (since in a w-cRBAC instantiation a user should be able to write to a file without knowing which roles have read access to that file). This is not achievable with classical PE, as the encryption algorithm requires the identity I as explicit input. We overcome this by introducing identity-specific public keys: we define a public-key generation algorithm that given the master public key and an identity I outputs an encryption key pk_I, which can be used to encrypt a message obliviously w.r.t. I.

[3]Katz et al. [29] call I an "attribute"; we prefer the term "identity" here in order to avoid confusion, as when instantiating ABE from PE, identities correspond to *sets* of attributes.

A. The Model

We extend PE to predicate encryption with (identity-) specific public keys (PE-SK) as a tuple of algorithms $\mathcal{PE} = (\mathsf{Setup}, \mathsf{PKGen}, \mathsf{DKGen}, \mathsf{Enc}, \mathsf{Dec})$. The setup algorithm Setup on input the security parameter λ (and optional parameters such as the attribute universe) returns a pair (mpk, mdk) of a master public and master secret (decryption) key. The public-key generation algorithm PKGen on inputs mpk and I returns a public encryption key pk_I for identity I. The decryption-key generation algorithm DKGen on inputs mdk and a predicate f returns a decryption key dk_f for f. The encryption algorithm Enc on inputs pk_I and m returns a ciphertext c. The decryption algorithm Dec on inputs sk_f and a ciphertext c returns a string m (or $\bot$).

Correctness. A PE-SK scheme $\mathcal{PE}$ is *correct* if for all λ, f, I, m, r, all (mpk, mdk) output by $\mathsf{Setup}(1^\lambda)$, all pk_I output by $\mathsf{PKGen}(mpk, I)$ and all dk_f output by $\mathsf{SKGen}(mdk, f)$ we have $\mathsf{Dec}(dk_f, \mathsf{Enc}(pk_I, m; r)) = m$ if and only if $f(I) = 1$. Since when knowing mdk one can always derive a key and then decrypt, we also directly write $\mathsf{Dec}(mdk, c)$.

Identity-hiding public keys. We first introduce a security notion that formalizes the requirement that keys do not reveal for which identity they are. An adversary must guess a random bit b after getting the master public key mpk and access to a challenge oracle LR, which on input (I_0, I_1) returns an encryption key for I_b. (Note that this also formalizes the fact that an adversary cannot tell whether two keys are for the same identity: given mpk, it can produce a key for pk_{I_0} and being given pk_{I_b} guess b by linking keys.)

The adversary is also provided a DKGen oracle, which models collusions between users. To prevent trivial attacks, we require the following restriction. When queried on f, DKGEN first checks whether $f(I_0) = f(I_1)$ for all (I_0, I_1) queried to LR (otherwise the decryption key could be used to test whether a ciphertext produced with the challenge key pk_{I_b} decrypts correctly or not). Analogously, LR only answers queries (I_0, I_1) if $f(I_0) = f(I_1)$ for all f queried to DKGEN.

Definition 5 (Identity-hiding public keys): The following game formalizes that ID-specific public keys do not reveal any non-trivial information about the identities they are for:

```
$\underline{\mathbf{Exp}^{\text{id-h-pk}}_{\mathcal{PE},\mathcal{A}}(\lambda)}$
  $b \leftarrow\!\!\$\ \{0,1\};\ F \leftarrow \emptyset;\ Ch \leftarrow \emptyset$
  $(mpk, mdk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
  $b' \leftarrow\!\!\$\ \mathcal{A}(mpk : \text{DKGEN}, \text{LR})$
  Return $(b' = b)$

$\underline{\text{DKGEN}(f)}$
  For all $(I_0, I_1) \in Ch$:
    If $f(I_0) \neq f(I_1)$ then Return $\perp$
  $F \leftarrow F \cup \{f\}$
  Return $dk_f \leftarrow\!\!\$\ \mathsf{DKGen}(mdk, f)$

$\underline{\text{LR}(I_0, I_1)}$
  For all $f \in F$:
    If $f(I_0) \neq f(I_1)$ then Return $\perp$
  $Ch \leftarrow Ch \cup \{(I_0, I_1)\}$
  Return $pk \leftarrow\!\!\$\ \mathsf{PKGen}(mpk, I_b)$
```

We say a PE-SK scheme $\mathcal{PE}$ has *identity-hiding encryption keys* if for any p.p.t. adversary $\mathcal{A}$

$$\mathbf{Adv}^{\text{id-h-pk}}_{\mathcal{PE},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\text{id-h-pk}}_{\mathcal{PE},\mathcal{A}}(\lambda) \to 1] - \tfrac{1}{2}\right|$$

is negligible in λ.

MESSAGE-HIDING. While our first security notion ensures that public keys (and ciphertexts created from them) do not reveal their associated identity, the second notion formalizes that ciphertexts of different messages should be indistinguishable. In contrast to the first notion, this also exists for standard PE, where it is sometimes called *payload-hiding* [29].

This notion is formalized via a game where the adversary must distinguish messages encrypted under a key whose corresponding secret key it must not know. We give the adversary access to an oracle that generates public keys pk_I for I of the adversary's choice. The adversary then chooses one such key and two equal-length messages (M_0, M_1) and gets an encryption of M_b under that key. More formally, the game stores queried keys pk_I and the identity I in the first empty index of two lists PK and $\mathcal{I}$, respectively. When the adversary asks for a challenge under the j-th key by querying (j, M_0, M_1), it receives an encryption of M_b under $PK[j]$. The corresponding identity $\mathcal{I}[j]$ is then added to the list of challenges Ch.

The adversary can also query decryption keys for any predicate f, which is then added to a list F. To prevent trivial attacks, the experiment maintains the invariant that for all $f \in F$ and $I \in Ch$ it should hold that $f(I) = 0$; otherwise, if for some f and I we had $f(I) = 1$, the adversary could query a challenge under the key for I and then decrypt it using dk_f.

Definition 6 (Message hiding): The following game formalizes the fact that ciphertexts hide the encrypted message:

```
$\underline{\mathbf{Exp}^{\text{msg-hide}}_{\mathcal{PE},\mathcal{A}}(\lambda)}$
  $b \leftarrow\!\!\$\ \{0,1\};\ ctr \leftarrow 1;\ PK, \mathcal{I}, F, Ch \leftarrow \emptyset$
  $(mpk, mdk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
  $b' \leftarrow\!\!\$\ \mathcal{A}(mpk : \text{PKGEN}, \text{DKGEN}, \text{LR})$
  Return $(b' = b)$

$\underline{\text{PKGEN}(I)}$
  $pk_I \leftarrow\!\!\$\ \mathsf{PKGen}(mpk, I)$
  $\mathcal{I}[ctr] \leftarrow I;\ PK[ctr] \leftarrow pk_I;\ ctr \leftarrow ctr + 1$
  Return $pk_I$

$\underline{\text{DKGEN}(f)}$
  For all $I \in Ch$:
    If $f(I) = 1$ then Return $\perp$
  $F \leftarrow F \cup \{f\}$
  Return $dk_f \leftarrow\!\!\$\ \mathsf{DKGen}(mdk, f)$

$\underline{\text{LR}(j, M_0, M_1)}$
  If $|M_0| \neq |M_1|$ then Return $\perp$
  Let $(pk_I, I) \leftarrow (PK[j], \mathcal{I}[j])$
  For all $f \in F$:
    If $f(I) = 1$ then Return $\perp$
  $Ch \leftarrow Ch \cup \{I\}$
  Return $C \leftarrow\!\!\$\ \mathsf{Enc}(pk_I, M_b)$
```

We say a PE-SK scheme $\mathcal{PE}$ has *message-hiding ciphertexts* if for any p.p.t. adversary $\mathcal{A}$

$$\mathbf{Adv}^{\text{msg-hide}}_{\mathcal{PE},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\text{msg-hide}}_{\mathcal{PE},\mathcal{A}}(\lambda) \to 1] - \tfrac{1}{2}\right|$$

is negligible in λ.

B. Instantiation

Having formally defined predicate encryption with specific public keys, we now discuss how to instantiate it. We first present a generic transformation of any predicate-encryption scheme $\mathcal{PE}$ into a PE with ID-specific public keys that hide their identities, provided that $\mathcal{PE}$ satisfies some extra properties. We then present a PE scheme with these properties.

PE satisfies two security notions: on the one hand, ciphertexts do not reveal anything about the message they encrypt; on the other hand, they do not reveal anything about their identity either. Our approach to constructing PE-SK is the following: Suppose the message space forms a group $(\mathbb{G}, \cdot)$ (which is the case for many instantiations) and let 1 denote its neutral element. Then we define a public key for an identity I as a ciphertext encrypting 1. Since keys are PE ciphertexts, which hide their identity, this immediately implies our first security notion from Def. 5.

To encrypt a message M, we need to turn a public key, which is an encryption of 1 under I, into an encryption of a specific M under I. This can be done if the PE is *malleable*, which is typically the case for existing schemes [37], [9], [29], [34], where a ciphertext contains an element $C_k = P \cdot M$, that is, M blinded by some factor $P \in \mathbb{G}$.

Being an encryption of 1, a public key would contain such P, thus it suffices to multiply it by the message that is to be encrypted. However, this would not be secure, as from a ciphertext and its key one could easily retrieve the message by dividing the ciphertext component by the key component. To amend this, we require a second property, which is *full randomizability* of encryptions of 1. This means that given an encryption of 1 under some identity I, anyone can compute a fresh encryption of 1 under the same identity—which is unknown to him (that is, an encryption that is distributed like the output of $\mathsf{Enc}_{\mathcal{PE}}(mpk, I, 1)$).

Suppose our base PE scheme has these two properties; then we can define encryption for a PE-SK as follows. Take the public key for identity I, randomize it to get a fresh encryption of 1 under I and then use malleability to transform it into an encryption of the message. Since the public key and the ciphertext are now completely unrelated (other than being for the same I), the message-hiding property (Def. 6) follows from the security of the underlying PE scheme.

We first present the definition of PE, following [29], and then formalize our new requirements in Def. 9 and 10.

Definition 7: A *predicate encryption scheme* $\mathcal{PE}$ consists of the algorithms $\mathsf{Setup}, \mathsf{DKGen}, \mathsf{Enc}$ and Dec, where Setup takes a security parameter 1^λ and outputs the public key pk and the master secret (decryption) key mdk. DKGen on input mdk and a predicate f outputs a decryption key dk_f. Enc takes the public key pk, an identity I and a message M, and outputs a ciphertext C. On input dk_f and C, Dec outputs M. We require that for all (pk, mdk) output by Setup, for all f, all sk_f output by $\mathsf{DKGen}(mdk, f)$, all I and M, and all C output by $\mathsf{Enc}(pk, I, M)$ we have $M \leftarrow \mathsf{Dec}(sk_f, C)$ iff $f(I) = 1$. Security of predicate encryption is defined as follows:

Definition 8 (Indistinguishability for PE): The following notion formalizes the fact that ciphertexts should be *ID-hiding* and *message-hiding* at the same time:

```
$\underline{\mathbf{Exp}^{\text{ind}}_{\mathcal{PE},\mathcal{A}}(\lambda)}$
  $b \leftarrow\!\!\$ \{0,1\}; F \leftarrow \emptyset; Ch \leftarrow \emptyset$
  $(pk, mdk) \leftarrow\!\!\$ \mathsf{Setup}(1^\lambda)$
  $b' \leftarrow\!\!\$ \mathcal{A}(pk : \text{DKGEN}, \text{LR})$
  Return $(b' = b)$

$\underline{\text{DKGEN}(f)}$
  For all $(I_0, I_1, M_0, M_1) \in Ch$:
    If $f(I_0) \neq f(I_1)$ then Return $\perp$
    If $f(I_0) = f(I_1) = 1$ and $M_0 \neq M_1$ then Return $\perp$
  $F \leftarrow F \cup \{f\}$
  Return $dk_f \leftarrow\!\!\$ \mathsf{DKGen}(mdk, f)$

$\underline{\text{LR}(I_0, I_1, M_0, M_1)}$
  For all $f \in F$:
    If $f(I_0) \neq f(I_1)$ then Return $\perp$
    If $f(I_0) = f(I_1) = 1$ and $M_0 \neq M_1$ then Return $\perp$
  $Ch \leftarrow Ch \cup \{(I_0, I_1, M_0, M_1)\}$
  Return $C \leftarrow\!\!\$ \mathsf{Enc}(pk, I_b, M_b)$
```

We say a predicate-encryption scheme $\mathcal{PE}$ has indistinguishability if for any p.p.t. adversary $\mathcal{A}$

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

is negligible in λ.

Definition 9 (Message malleability): Let 1 be a fixed element from the message space (typically the neutral element of the group that forms the message space). We say that a predicate-encryption scheme $\mathcal{PE}$ is *message-malleable* if there exists an algorithm Maul that takes the public key pk, a message M and an encryption of 1 w.r.t. some identity I and outputs an encryption of M under the identity I.

Definition 10 (Randomizability): We say that a (message-malleable) predicate-encryption scheme $\mathcal{PE}$ has (full) *randomizability of encryptions of 1* if there exists an algorithm Rand that takes the public key pk and an encryption of 1 w.r.t. any identity I and outputs a *fresh* encryption of 1 w.r.t. I, that is, the output of Rand is distributed as the output of $\mathsf{Enc}(pk, I, 1)$. (Note that Rand does not get I as input.)

GENERIC CONSTRUCTION. Let $\mathcal{PE} = (\mathsf{Setup}, \mathsf{DKGen}, \mathsf{Enc}, \mathsf{Dec})$ be a predicate-encryption scheme that is message-malleable and has randomizable encryptions of 1. Then we construct a predicate-encryption scheme with ID-specific public keys $\mathcal{PE}'$ as follows:

- $\mathsf{Setup}'(1^\lambda)$ returns $(mpk, mdk) \leftarrow\!\!\$ \mathsf{Setup}(1^\lambda)$.
- $\mathsf{PKGen}'(mpk, I)$ returns $pk_I \leftarrow\!\!\$ \mathsf{Enc}(mpk, I, 1)$.
- $\mathsf{DKGen}'(mdk, f)$ returns $dk_f \leftarrow\!\!\$ \mathsf{DKGen}(mdk, f)$.
- $\mathsf{Enc}'(pk_I, M)$ runs $C' \leftarrow\!\!\$ \mathsf{Rand}(mpk, pk_I)$ and returns $C \leftarrow\!\!\$ \mathsf{Maul}(mpk, M, C')$.

Correctness of this construction follows easily by inspection; we now prove that it is secure:

Theorem 2: If $\mathcal{PE}$ has indistinguishability (Def. 8), message malleability (Def. 9) and randomizability (Def. 10) then the scheme $\mathcal{PE}'$ constructed above is a PE scheme with ID-specific keys which are ID-hiding (Def. 5) and with message-hiding ciphertexts (Def. 6).

Proof: The fact that keys are ID-hiding follows immediately from the security of $\mathcal{PE}$, since keys are ciphertexts: In particular, from an adversary $\mathcal{A}$ which breaks IHPK (Def. 5) of $\mathcal{PE}'$, we construct $\mathcal{B}$ which breaks IND (Def. 8) of $\mathcal{PE}$. $\mathcal{B}$ is given pk, from its challenger and forwards it as mpk to $\mathcal{A}$. Whenever $\mathcal{A}$ queries its LR oracle on (I_0, I_1), $\mathcal{B}$ queries its own LR oracle on $(I_0, I_1, 1, 1)$ and forwards the response. Queries to DKGEN are simply relayed by $\mathcal{B}$ and $\mathcal{B}$'s final output is what $\mathcal{A}$ outputs. It is immediate that when $\mathcal{A}$ wins $\mathbf{Exp}^{\text{id-h-pk}}_{\mathcal{PE}'}$ then $\mathcal{B}$ wins $\mathbf{Exp}^{\text{ind}}_{\mathcal{PE}}$.

It remains to prove that $\mathcal{PE}'$ has message-hiding ciphertexts (Def. 6). Consider an adversary $\mathcal{A}$ winning $\mathbf{Exp}^{\text{msg-hide}}_{\mathcal{PE}'}$. Again, we construct an adversary $\mathcal{B}$ for $\mathbf{Exp}^{\text{ind}}_{\mathcal{PE}}$. $\mathcal{B}$ is given pk from its challenger and forwards it as mpk to $\mathcal{A}$. It initializes a counter $ctr \leftarrow 1$ and a list $\mathcal{I} \leftarrow \emptyset$. Whenever $\mathcal{A}$ queries PKGEN(I), $\mathcal{B}$ sets $\mathcal{I}[ctr] \leftarrow I$; $ctr \leftarrow ctr + 1$, and returns $pk \leftarrow\!\!\$ \mathsf{Enc}(mpk, I, 1)$. Queries to DKGEN are simply relayed by $\mathcal{B}$ to its own challenger. Finally, when $\mathcal{A}$ queries LR(j, M_0, M_1), $\mathcal{B}$ queries LR$(\mathcal{I}[j], M_0, M_1)$ and forwards the response to $\mathcal{A}$.

The reason why $\mathcal{B}$ perfectly simulates $\mathbf{Exp}^{\text{id-h-pk}}_{\mathcal{PE}'}$ is that encryption of $\mathcal{PE}'$ first fully randomizes the public key before mauling the message in. When querying LR(j, M_0, M_1), $\mathcal{A}$ is supposed to get $\mathsf{Enc}'(PK[j], M_b)$, which is defined as $\mathsf{Maul}(mpk, M_b, \mathsf{Rand}(mpk, PK[j]))$. By the properties of Rand and Maul, this is however distributed exactly as $C_b := \mathsf{Enc}(mpk, \mathcal{I}[j], M_b)$. Now C_b is precisely the value that $\mathcal{B}$ receives from its challenger, meaning it perfectly simulates the game for $\mathcal{A}$. Finally, $\mathcal{B}$ outputs what $\mathcal{A}$ outputs and wins exactly whenever $\mathcal{A}$ wins. ■

CANDIDATE PE SCHEME. The first predicate-encryption schemes [9], [29], [35], which were only "selectively secure", are all message-malleable, but do not seem to be fully randomizable. In 2012 Okamoto and Takashima [34] presented the first adaptively secure inner-product encryption (IPE) scheme [29], a special type of PE, which we define below. Interestingly, their scheme only relies on the decision-linear assumption [8] in prime-order pairing groups. A description of the relevant parts of the scheme, following the notation from [34], can be found in the full version, where we show that the scheme satisfies both message malleability (Def. 9) and randomizability (Def. 10).

C. Predicate Encryption for Non-Disjoint Sets from IPE

In Section IV-B we showed the existence of a PE-SK scheme, whose security is based on a standard hardness assumption, for inner-product predicates, which were shown to encompass many classes of predicates [29]. Here we show that *predicate encryption for non-disjoint sets*, as defined in [19] and used for RBAC, can also be expressed via inner products.

INNER-PRODUCT ENCRYPTION. Katz et al. [29] define PE for inner-product predicates, and call it *inner-product encryption* (IPE). In IPE, identities I (with respect to which messages are encrypted) correspond to vectors $\vec{z}$ of some length n, whose elements are typically from $\mathbb{Z}_q$ (where q is the order of the group over which the IPE is defined). Predicates f, for which decryption keys can be issued, are also associated to vectors $\vec{v} \in \mathbb{Z}_q^n$ and an identity $\vec{z}$ satisfies a predicate $f_{\vec{v}}$ if and only if the inner product of $\vec{v}$ and $\vec{z}$ is 0, i.e. $f_{\vec{v}}(I_{\vec{z}}) = 1 \Leftrightarrow \langle \vec{v}, \vec{z} \rangle := \sum_{i=0}^{n-1} v_i \cdot z_i = 0$ over $\mathbb{Z}_q$. (For consistency we index all vectors starting from 0.)

PENDS FROM IPE. As shown in [29], IPE can be used to implement predicates corresponding to polynomial evaluation: keys are associated to degree-$(n-1)$ polynomials p, and messages are encrypted w.r.t. values y. A key for p should be able to decrypt a ciphertext for y if and only if $p(y) = 0$.

To implement this with IPE, a key for a polynomial p is issued for $\vec{v}$ where v_i is set to the i-th coefficient of p. When encrypting w.r.t. to a value y, one constructs an IPE encryption w.r.t. $\vec{z}$ where we set $z_i \leftarrow y^i$. Since $p(y) = \sum_{i=0}^{n-1} v_i z_i$, we have

$$p(y) = 0 \Leftrightarrow \langle \vec{v}, \vec{z} \rangle = 0 \ .$$

Our goal is to implement *predicate encryption for non-disjoint sets* (PENDS), for subsets of $A = \{0, \ldots, \ell-1\}$, where ciphertexts and keys are associated to sets of attributes x and y, respectively, and decryption should succeed iff

$$\bigvee_{i=0}^{\ell-1} (i \in y \wedge i \in x) \ . \tag{1}$$

We define the characteristic vector $\vec{c}$ for y (that is, $c_i = 1$ if $i \in y$ and $c_i = 0$ otherwise); we define a slightly different vector $\vec{d}$ for x with $d_i = 1$ if $i \in x$ and $d_i = 2$ otherwise. Now (1) can be rewritten as $\prod_{i=0}^{\ell-1}(c_i - d_i) = 0$, whose left-hand side we will express as a sum via a multivariate polynomial in order to apply IPE.

In detail, in order to issue a key for a set of attributes $x \subseteq A$, define $\vec{d} \in \{1,2\}^\ell$ with $d_i \leftarrow 1$ if attribute i is contained in x, and $d_i \leftarrow 2$ if it is not. Then consider the polynomial $p(X_0, \ldots, X_{\ell-1}) := \prod_{i=0}^{\ell-1}(X_i - d_i)$, which (letting $j[i]$, for $i = 0, \ldots, \ell-1$, denote the i-th bit of $j \leq 2^\ell - 1$ in binary) can be expanded to $\sum_{j=0}^{2^\ell-1} v_j \big(\prod_{i=0}^{\ell-1} X_i^{j[i]} \big)$ for some coefficients v_j, $0 \leq j \leq 2^\ell - 1$. A key for $x \subseteq A$ is then an IPE key for the vector $\vec{v} \in \mathbb{Z}_q^{2^\ell}$.

When encrypting w.r.t. to a set of attributes $y \subseteq A$, we represent y by its characteristic string $\vec{c} \in \{0,1\}^\ell$ and compute an IPE ciphertext w.r.t. $\vec{z}$ with $z_j = \prod_{i=0}^{\ell-1} c_i^{j[i]}$ for $j = 0, \ldots, 2^\ell - 1$. Now decryption succeeds if

$$0 = \langle \vec{v}, \vec{z} \rangle = \sum_{j=0}^{2^\ell-1} v_j z_j = \sum_{j=0}^{2^\ell-1} v_j \big(\prod_{i=0}^{\ell-1} c_i^{j[i]}\big) = \\ = p(c_0, \ldots, c_{\ell-1}) = \prod_{i=0}^{\ell-1}(c_i - d_i) \ ,$$

which holds if and only if for some i we have $c_i = 1 = d_i$, which is the case iff the set of attributes y corresponding to the ciphertext and the set of attributes x for the key share a common attribute.

The drawback of this construction is of course that for ℓ attributes we need an inner-product encryption for vectors of length 2^ℓ (which linearly determines the size of ciphertexts and keys). We however chose to construct a PE with ID-specific keys for predicates as general as possible. We leave it as an open problem to construct more efficient PE-SK schemes for the specific functionality of non-disjoint sets.

V. AN IMPLEMENTATION OF CRYPTOGRAPHIC RBAC WITH POLICY PRIVACY

In this section we provide a high-level description of an implementation for w-cRBAC. Here we use notation that differs slightly from that used in the full version, but which has the advantage of being more compact and readable. We first describe how files are stored and then briefly sketch how RBAC administrative actions are implemented via a combination of key-management and resigning/re-encrypting operations.

The main ingredient of our scheme is a PE-SK scheme which is used as follows. To each role r in the system we associate two attributes: attribute r_r to which we refer as the read attribute of role r and which we use to control the reading rights associated to the role, and attribute r_w to which we refer as the write attribute of role r. We use the latter to control writing rights, as described below. To all users that have role r we provide the decryption keys associated to r_r and r_w. More precisely, each user will actually only have two keys: one corresponding to the write attributes for all of the roles to which he belongs, and another corresponding to the read attributes of the same roles.

To control read access to file o, we simply encrypt the file under a public key that corresponds to *all* of the read attributes of roles that have read right to o (recall that computing such keys is one of the functionalities provided by PE-SK schemes). A user with role r that has read access to o can simply use r_r to access it. To control write access, we use a standard digital signature scheme. Since all users can append to the storage, the challenge is to ensure that only users that can write to a file can actually append a *valid* update. We proceed as follows. To each file o we associate a signing/verification key pair sk_o, vk_o. Users can update o by adding the modified variant to the storage, but only the updates that are signed with sk_o are valid updates, indeed, only such updates could be verified by using vk_o. To ensure that only the right users can obtain sk_o, we encrypt it under the write attributes that correspond to the roles that have the right to write to o.

In more detail, we assume an append-only file system fs (logically) organized as a matrix. Each row corresponds to a file and each column to a version of the file. The structure of a row in the file system fs is thus a vector of the form

$$(pk; vk; sk), (ctx_1; sig_1), (ctx_2; sig_2), \ldots, (ctx_i; sig_i)$$

whose components we describe next.

In position $i = 0$ is the header of the file o to which the row corresponds. The information there is publicly readable but can be written only by the manager. It consists of three fields which, for a file o, we identify by $fs[o][0].pk$, $fs[o][0].vk$ and $fs[o][0].sk$. These are an encryption key $fs[o][0].pk$ that corresponds to the read attributes of the roles that can read o, a verification key $fs[o][0].vk$ for the signature scheme, and an encryption $fs[o][0].sk$ of the signing key associated to vk. This last encryption is under the write attributes of all roles that have write access to o. We control write access through an appropriate distribution of keys which allow to recover encrypted signing keys.

Users can add new versions of the file o by appending them to the row that corresponds to o. Thus, any position i such that $i > 0$ contains the i-th update of the file which we identify by ($fs[o][i].ctx$, $fs[o][i].sig$). However, a valid entry on the row of file o is of the form (ctx, sig) where sig is a valid signature on ctx with respect to $fs[o][0].vk$; in a normal execution ctx is an encryption of the file under $fs[o][0].pk$. The last valid row entry corresponds to the latest version of the file. To add a new version to o, an authorized user first encrypts the new content under $fs[o][0].pk$, obtains sk_o by decrypting $fs[o][0].sk$ and uses it to sign the ciphertext. To prevent roll-back attacks where a malicious user simply copies some old entry, the signature is on the ciphertext together with the index i that corresponds to the next empty position of the row corresponding to file o. The user then posts the ciphertext-signature pair to position i, and this becomes the most recent version of the file.

Whenever an authorized user wishes to read a file o, she fetches the latest version of the file ($fs[o][i].ctx$, $fs[o][i].sig$), for some $i > 0$, and determines whether $fs[o][i].sig$ is a valid signature and whether the signed index is equal to i. If not, she fetches a previous version until a valid entry is encountered.

Whenever a role loses writing privileges to o, a new signature key pair for o is freshly generated. The new verification key is made public and the signing key is encrypted under the roles that still have the right to write. The latest valid version of the file is signed by the manager (who for space efficiency, can also erase all the previous versions), so that the signature is valid under the new verification key associated to o.

Since multiple (encrypted) versions of the file are present in the system, the management of keys needs to be carefully crafted to avoid pitfalls where newly assigned rights permit access to old content. For example, whenever a read access is revoked from role r, the manager (1) assigns a fresh read attribute to role r, (2) recomputes all the public keys for files to which r has still read access (to account for the changed attribute for r), (3) re-encrypts all latest (valid) ciphertexts under these public keys, and (4) sends the decryption keys associated to r_r to all users assigned to r.

Whenever a user is deassigned from a role r, the attribute for r is also changed and all the steps (1)–(4) are executed as above. In addition, all signature key pairs for files to which r has write permission are also renewed; in particular, the new signing key is encrypted, and the concerned files will be re-signed to maintain validity under the new verification key.

A detailed description of the algorithms of our w-cRBAC implementation $\mathcal{CRBAC}[\mathcal{PE}, \Sigma]$ based on a PE-SK scheme $\mathcal{PE}$ and a signature scheme Σ can be found in the full version of the paper.

There we also provide the following theorem which establishes that our construction is secure with respect to read accesses, provided that the encryption scheme used in the implementation is message-hiding. Our proof is for the scenario where users that have the right to write to some file also have read access to it.

Recall that in the model proposed by Ferrara et al. a cRBAC is said to be secure with respect to read access if for any user who does not have the read permission of a file should not be able to deduce anything about the content of that file. Since w-cRBAC extends the notion of cRBAC, we need to reintroduce the security model regarding read accesses to make it work for w-cRBAC.

Definition 11: A w-cRBAC is *secure* with respect to read accesses if for all p.p.t. adversaries $\mathcal{A}$, we have

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

is negligible in λ, where $\mathbf{Exp}^{\mathrm{ind}}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}$ is defined as follows:

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

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

Theorem 3: If PE-SK scheme $\mathcal{PE}$ is message-hiding, then $\mathcal{CRBAC}[\mathcal{PE}, \Sigma]$ is secure with respect to read accesses.

This theorem extends the earlier result of [19]. Some care is needed to deal with situations specific to our new scheme (e.g., in [19] all write operations were performed by the manager, meaning the simulator always knew the content of the file system). Although we require a more involved security reduction, the proof relies on a similar intuition.

The next theorem (which we prove in the full version) shows the security of our scheme with respect to write access. Here, security relies both on the privacy ensured by the encryption scheme (used to encrypt the signing keys associated to files) and on the security of the signature scheme itself.

Theorem 4: If the signature scheme Σ is existentially unforgeable under adaptive chosen-message attacks and PE-SK scheme $\mathcal{PE}$ is message-hiding, then $\mathcal{CRBAC}[\mathcal{PE}, \Sigma]$ is secure with respect to write accesses.

PROOF IDEA. Intuitively, the only way an adversary could break the security of the scheme with respect to the write-access is to somehow modify the content of a file, that is, appending a new version of the file with a valid signature attached to it. More formally, we prove the theorem by a

```
$\underline{\text{CMD}(arg)}$
    $(U', O', P', UA', PA')$
            $\leftarrow Cmd((U, O, P, UA, PA), arg)$
    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 $\perp$
    $(U, O, P, UA, PA) \leftarrow (U', O', P', UA', PA')$
    If $Cmd = $ DELUSER then
        Parse $arg$ as $u$; $Cr \leftarrow Cr \setminus \{u\}$
    If $Cmd = $ DELOBJECT then
        Parse $arg$ as $o$; $Ch \leftarrow Ch \setminus \{o\}$
    $(st_M, fs, \{msg_u\}_{u \in U}) \leftarrow\!\!\$\ \mathsf{Cmd}(st_M, fs, arg)$
    For all $u \in U \setminus Cr$:
        $st[u] \leftarrow \mathsf{Update}(st[u], msg_u)$
    Return $(fs, \{msg_u\}_{u \in Cr})$
```

```
$\underline{\text{WRITE}(u, o, m)}$
    If $u \notin U$ then Return $\perp$
    If $\neg\mathsf{HasAccess}(u, (o, \mathtt{write}))$
        then Return $\perp$
    $fs \leftarrow\!\!\$\ \mathsf{Write}(st_M, fs, o, m)$
    Return $fs$

$\underline{\text{CORRUPTU}(u)}$
    If $u \notin U$ then Return $\perp$
    For all $o \in Ch$:
        If $\mathsf{HasAccess}(u, (o, \mathtt{write}))$
            then Return $\perp$
    $Cr \leftarrow Cr \cup \{u\}$; Return $st[u]$
```

```
$\underline{\text{CHALLENGE}(o, m_0, m_1)}$
    If $o \notin O$ then Return $\perp$
    For all $u \in Cr$:
        If $\mathsf{HasAccess}(u, (o, \mathtt{write}))$
            then Return $\perp$
    $Ch \leftarrow Ch \cup \{o\}$
    $fs \leftarrow\!\!\$\ \mathsf{Write}(st_M, fs, o, m_b)$
    Return $fs$

$\underline{\text{FS}(query)}$
    If $query = $"STATE" then
        Return $fs$
    If $query = $"APPEND($info$)" then
        $fs \leftarrow fs \| info$; Return $fs$
```

Fig. 5. Oracles for defining the experiment $\mathbf{Exp}^{\text{ind}}_{w\text{-}\mathcal{CRBAC},\mathcal{A}}$ that defines security of access control with respect to the read permission.

sequence of games. The first game is the original attack game for the scheme regarding write access. The second game proceeds as the first game, except that it does not encrypt the signing keys but random strings of the same length instead, and stores these in the headers of files. In order to distinguish these two games, one must distinguish the encrypted signing keys from the encrypted random strings of the same length and therefore break the message-hiding notion of the underlying PE-SK scheme. In this modified game an adversary that succeeds in breaking security regarding write access must be able to produce a forgery corresponding to one of the signing keys, meaning it breaks EUF-CMA security of the signature scheme. We thus conclude that the advantage that adversary can gain in winning the original attack game is negligible, which proves the theorem.

POLICY PRIVACY. As explained in the introduction, the cryptographic RBAC scheme of [19] leaks the entire PA matrix that records the association between roles and permissions: since read access is controlled by requiring users to encrypt files under all of the (attributes associated to) roles that should have access to these files, this information is in the clear. The implementation that we presented in the previous section bypasses this problem by associating to a set of attributes I an encryption key pk_I, which users can use to encrypt for the roles associated to the attributes in I. Intuitively, if the encryption key hides the associated set of attributes then one should not be able to determine which roles have access to a certain file, in other words, our scheme should satisfy p2r-privacy. This is indeed true, in a static sense: an adversary that can access the file system cannot observe the link between the ciphertexts in the system and the roles which have access to them.

As discussed in Section III, dynamic changes to the files may reveal some of this information, especially if the adversary has some partial knowledge on the access structure. For example, deassigning a read permission from a role requires changing the attribute that corresponds to the role in question, *and* re-encrypting the files to which the role still has read access. If, say, there is only one particular role that has access to particular set of re-encrypted files then information about the role is leaked by what the adversary can observe. We discuss later in the paper on ways that prevent this leak. Our scheme satisfies however p2r*-privacy, that is, assigning permissions to roles preserves policy privacy.

Theorem 5: The scheme $\mathcal{CRBAC}[\mathcal{PE}, \Sigma]$ is p2r*-private if $\mathcal{PE}$ has attribute-hiding keys.

PROOF IDEA. Our scheme hides the mapping between write permissions and the roles to which these are assigned. This is because the ciphertexts that encrypt the signing keys do not reveal to which attributes they correspond. In particular, we show how to construct from an adversary $\mathcal{A}$ that breaks p2r*-privacy an adversary $\mathcal{B}$ that breaks the privacy notion (Def. 5) of the underlying encryption scheme PE-SK. At a high level, the signature scheme involved in the implementation is run entirely by $\mathcal{B}$, whereas the identity-specific keys are used from the experiment that $\mathcal{B}$ is involved in. The crux of the simulation is that whenever adversary $\mathcal{A}$ issues a grant-permission challenge request, which demands that one of two roles r_0 and r_1 be granted read permission p, adversary $\mathcal{B}$ uses its challenge oracle to obtain an encryption key that corresponds to roles $S \cup \{r_b\}$ (where S is the set of roles that already have permission p). If $\mathcal{A}$ guesses correctly which of the two roles was involved, adversary $\mathcal{B}$ also guesses correctly his challenge bit. The proof can be found in the full version.

VI. CONCLUSION AND FUTURE WORK

In this paper we started to address the issue of policy privacy in cryptographic access control, which we study in the context of cryptographically enforced Role Based Access Control. We show through realistic examples that access policies may be rather sensitive information. We take advantage of the fact that access-control policies are not specified through some monolithic quantity but have clearly identifiable parts, and we design models for each of these parts separately. We also propose a construction that securely enforces read and write access to a file system, while offering a certain degree of privacy for the underlying policy. Our work opens many interesting avenues for future research, some of which we discuss below.

ALTERNATIVE SECURITY MODELS. Our privacy definitions are based on games that measure the ability of an adversary to distinguish between two possible executions. This type of definition come with compelling intuition and are standard in cryptographic literature. An alternative definitional framework is based on simulation: a system is secure if whatever information it leaks to an adversary is necessarily leaked even by an idealized functionality for the same task. These definitions are also intuitive but often cumbersome to use. It would be interesting however to design such definitions for policy-hiding access control, investigate the links with our approach, and assess the feasibility of implementations that meet this (typically stronger) notions.

EFFICIENT IMPLEMENTATIONS. Finding more efficient implementations for schemes that meet game-based security notions is also worth pursuing. Our scheme leaves room for both several obvious refinements (e.g. use hybrid encryption to encrypt files, allow the manager to periodically clean the file system only keeping the last valid version of each file) but also for more conceptual changes. For example, an intriguing question is whether we can employ policy-based signatures [6] in a way similar to our use of encryption. The use would, in some sense, be dual to our current approach. Specifically, we could assign to each file a verification key that encodes the roles that have the right to sign that file, and provide users with signing keys that correspond to their roles. This approach however seems to require new privacy definitions for policy-based signatures, as a straightforward implementation would link the roles to the files to which they have write permission.

STRONGER FORMS OF PRIVACY. Our scheme is trivially r2u private. The reason is that assigning users to roles does not require any change to the file system while the changes due to deassignment do not leak the identity of the user. Unfortunately, as we have discussed previously, our scheme does not meet p2r-privacy. Indeed, revoking read permission p from some role r may reveal information about the role, as the files pertaining to r need to be re-encrypted. The resulting change may be sufficient to identify r. For similar reasons, our scheme does not achieve r2p-privacy and u2r-privacy. One inefficient way to achieve p2r-privacy is to re-encrypt not only the files pertaining to r, but also all files pertaining to some role r' which has permission p. Another, more efficient, possibility but one that requires more intricate key management is not to re-encrypt all files, but have users that belong to that role keep both the old and the new key. The latter can be used to decrypt newly encrypted files for role r whereas the old keys can be used to still access the old files. Finding a way to combine these keys in order to simplify key management and reduce storage space is a research direction that we plan to pursue. Similar expedients can be employed to achieve r2p-privacy and u2r-privacy at the expense of efficiency.

ACKNOWLEDGEMENTS

We thank Ulfar Erlingsson and the anonymous reviewers for sugestions that have significantly improved the presentation of the paper. The second author is supported by ERC starting grant (259668-PSPC). Part of this work was carried out while Bogdan Warinschi was a visiting Microsoft Research, Cambridge, UK. He has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant EP/H043454/1, and has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement 609611 (PRACTICE).

REFERENCES

[1] American National Standards Institute: ANSI INCITS 359-2004 for Role Based Access Control, 2004.

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

[3] 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.

[4] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, January 2009.

[5] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, December 2001.

[6] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer, March 2014.

[7] Matt Blaze. A cryptographic file system for UNIX. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 9–16, New York, NY, USA, 1993. ACM.

[8] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, August 2004.

[9] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, February 2007.

[10] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 290–307. Springer, August 2006.

[11] Giuseppe Cattaneo, Luigi Catuogno, Aniello Del Sorbo, and Pino Persiano. The design and implementation of a transparent cryptographic file system for unix. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 199–212, Berkeley, CA, USA, 2001. USENIX Association.

[12] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 121–130. ACM, 2009.

[13] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 189–204, New York, NY, USA, 2007. ACM.

[14] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[15] Jason Crampton. Cryptographic enforcement of role-based access control. In *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers*, pages 191–205, 2010.

[16] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. *Theor. Comput. Sci.*, 412(41):5684–5699, 2011.

[17] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[18] David Ferraiolo and Richard Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[19] Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced RBAC. In *CSF*, pages 115–129. IEEE, 2013.

[20] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[21] David K. Gifford. Cryptographic sealing for information secrecy and authentication. *Commununications of the ACM*, 25(4):274–286, 1982.

[22] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[23] Garth R. Goodson, Jay J. Wylie, Gregory R. Ganger, and Michael K. Reiter. Efficient byzantine-tolerant erasure-coded storage. In *2004 International Conference on Dependable Systems and Networks (DSN 2004)*, pages 135–144. IEEE Computer Society, 2004.

[24] 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 CCS 06*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[25] 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.

[26] Anthony Harrington and Christian Jensen. Cryptographic access control in a distributed file system. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, pages 158–165, New York, NY, USA, 2003. ACM.

[27] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In Jeff Chase, editor, *Proceedings of the FAST'03 Conference on File and Storage Technologies*. USENIX, 2003.

[28] Apu Kapadia, Patrick P. Tsang, and Sean W. Smith. Attribute-based publishing with hidden credentials and hidden policies. In *In The 14th Annual Network and Distributed System Security Symposium (NDSS07)*, pages 179–192, 2007.

[29] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, April 2008.

[30] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB 2003: 29th International Conference on Very Large Data Bases*, pages 898–909, 2003.

[31] Takashi Nishide, Kazuki Yoneyama, and Kazuo Ohta. Attribute-based encryption with partially hidden encryptor-specified access structures. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08*, volume 5037 of *LNCS*, pages 111–129. Springer, June 2008.

[32] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 214–231. Springer, December 2009.

[33] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, August 2010.

[34] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 591–608. Springer, April 2012.

[35] Jong Hwan Park. Inner-product encryption under standard assumptions. *Des. Codes Cryptography*, 58(3):235–257, 2011.

[36] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, August 1992.

[37] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, May 2005.

[38] 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.

[39] Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Gregory R. Ganger. Metadata efficiency in versioning file systems. In Jeff Chase, editor, *Proceedings of the FAST '03 Conference on File and Storage Technologies*. USENIX, 2003.

[40] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised systems. In Michael B. Jones and M. Frans Kaashoek, editors, *4th Symposium on Operating System Design and Implementation (OSDI 2000)*, pages 165–180. USENIX Association, 2000.

[41] Yan Zhu, Gail-Joon Ahn, Hongxin Hu, and Huaixi Wang. Cryptographic role-based security mechanisms based on role-key hierarchy. In *ASIACCS 2010*, pages 314–319, 2010.