© ACM, 2001. This is the authors' version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version is available at http://doi.acm.org/10.1145/503339.503343.

An Authorization Model for a Public Key Management Service

PIERANGELA SAMARATI
Università di Milano
MICHAEL K. REITER
Carnegie Mellon University
and
SUSHIL JAJODIA
George Mason University

Public key management has received considerable attention from both the research and commercial communities as a useful primitive for secure electronic commerce and secure communication. While the mechanics of certifying and revoking public keys and escrowing and recovering private keys have been widely explored, less attention has been paid to access control frameworks for regulating access to stored keys by different parties. In this article we propose such a framework for a key management service that supports public key registration, lookup, and revocation, and private key escrow, protected use (e.g., to decrypt selected messages), and recovery. We propose an access control model using a policy based on principal, ownership, and authority relationships on keys. The model allows owners to grant to others (and revoke) privileges to execute various actions on their keys. The simple authorization language is very expressive, enabling the specification of authorizations for composite subjects that can be fully specified (ground) or partially specified, thus making the authorizations applicable to all subjects satisfying some conditions. We illustrate how the access control policy and the authorizations can easily be expressed through a simple and restricted, hence efficiently computable, form of logic language.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: Access control, authorizations specification and enforcement, public key infrastructure

The work of Pierangela Samarati was partially supported by the European Community within the Fifth (EC) Framework Programme under contract IST-1999-11791–FASTER project.

Authors' addresses: Pierangela Samarati, Dipartimento di Tecnologie dell'Informazione, Università di Milano, Via Bramante, 65, 26013 Crema (CR), Italy; email: samarati@dsi.unimi.it; Michael K. Reiter, Carnegie Mellon University, Pittsburgh, PA 15213; email: reiter@cmu.edu; Sushil Jajodia, Center for Secure Information Systems, George Mason University, Fairfax, VA 22030-4444; email: jajodia@gmu.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in others works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept, ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2001 ACM 1094-9224/01/1100–0453 $5.00

1. INTRODUCTION

The mechanics of public key management—i.e., the creation and distribution of public key certificates and revocation lists, and the secure backup (escrow) of private keys—has been explored extensively (e.g., see ACM [1996]). However, much less attention has been paid to the authorization model that must surround these mechanisms. At first glance, the authorization model seems quite simple: a public key should be accessible to anyone, and an escrowed private key should be accessible only to its owners, as their names indicates. This simple model is, however, far too underdeveloped for many realistic settings.

—One purpose in escrowing a private key, relevant mainly in business settings, is to delegate the authority to use that key to the escrowing service. For example, upon leaving for vacation, an executive might escrow her private key at the service and authorize that it be used to decrypt messages upon request by at least two of her direct reports. When she returns from her vacation, she can check the logs at the service to see what her key was used to decrypt.

—It may be desirable that some "public" keys not be public at all, but available to only a selected few for verifying signatures on private messages. In this way, if a message is accidentally leaked from an organization, outsiders must still obtain a certified verifying key to prove that the message actually originated from within the organization. Hiding "public" keys can also be useful for preventing outsiders from sending encrypted email to targets within the organization. Encrypted email could, for instance, be used to slip a virus-laden document by virus-detection software at the organization's firewall.

—The owner of a public key may confer revocation authority to others, so that one of them can revoke the public key if the private key is stolen and the owner's copy is destroyed.

Access control for public key management services must also address the peculiarities of dealing with keys as objects of authorizations. In particular, keys enjoy a more dynamic behavior than traditional objects/resources, which raises the issue of how keys should be referred to (e.g., via their values or their principals). Also, the fact that keys can expire or be modified introduces the problem of dealing with the authorizations specified on them and whether they should still be considered valid.

In this article we present an authorization model for controling access to the public and private keys held by a key management service. The model is both powerful and flexible, as it addresses the issues above by allowing principals to specify different protection requirements on their keys. In addition, it is simple enough to be realized in practice. It differs from authorization models proposed for general database systems because it allows specification of, and reasoning about, joint authority, delegation, and roles. It also addresses the issue of authorizations on a key once the key is revoked or replaced. Our model departs from previous proposals in the context of key management systems by considering identities, groups, and roles in the specification of authorizations. It allows users to interact with the service to establish keys and manage them.

And, while previous proposals have focused mostly on certification authority and certificate management, our approach specifically focuses on keys and related services.

The contributions of this article can be summarized as follows. First, it investigates the nuances of key management systems with respect to access control and authorization specifications. Second, it proposes access and authorization policies supporting ownership and authority. Third, it develops a simple, yet expressive language for the specification of authorizations and enforcing the proposed access control policies.

The generic key management service that forms the framework for this article was inspired by the Ω service [Reiter et al. 1996], and our work complements Ω nicely in that authorization and access control in Ω was left largely undeveloped [Reiter et al. 1996, Section 6]. We note that, while for concreteness this article refers to the Omega key management service, our access control model can be applied to any public key management system, including Yaksha [Ganesan 1995], Sesame [McMahon 1995], and Micali's proposed key management service [Micali 1992]. Also, consistent with Ω, in this article we focus on user keys. The model proposed can however be extended to regulate management of different types of keys, such as session or encryption keys.

The remainder of this article is organized as follows. Section 2 describes the basic services of the key management system, illustrating the different actions that can be exercised on the stored keys. Section 3 introduces the basic elements of the authorization model. Section 4 defines the relationships between subjects and principals and keys stored at the service. It also introduces the administrative and access control policies applied by the KMS. Section 5 discusses the nuances of a key management service with respect to access control, and defines the basic elements of the access control language and the specification of authorizations. Section 6 describes how the proposed policy and authorizations can be represented and enforced through a generic, simple, and efficiently computable logic language. Section 7 discusses authentication of principals submitting requests to the key management service, and the particular execution of first time registration and recovery. Section 8 discusses support of user groups and roles by the key management service. Section 9 covers related work. Finally, Section 10 presents our conclusions.

2. THE KEY MANAGEMENT SERVICE

The key management service stores the public keys of principals and escrowed private keys to allow the use of these keys by the principals who are authorized to do so. In this article we consider a KMS that closely mimics the Ω system [Reiter et al. 1996] in functionality. In particular, Ω supports interfaces by which a client can, if access control policy allows,

–*register a public key* at the service;

–*retrieve a public key* that was registered at the service (or retrieve the principal corresponding to a public key);

–*revoke a registered public key* from the service;

Table I. Actions on Keys and Their Semantics

Action	Semantics
Register(p, K)	Registers public key K for principal p. Any key K' previously registered for p is revoked, i.e., *Revoke*(K') is automatically performed (see Section 7.2). *Register*(p, K) fails if K was ever given as an argument to any *Register* action before.
Lookup(p)	Returns public key K last registered for p, provided that *Revoke*(K) has not been performed. *Lookup*(p) fails if no key has been registered for p or if the last key registered for p has been revoked.
Lookup(K)	Returns the name of the (unique) principal p for which K was previously registered, and a flag indicating whether K has been revoked. This operation fails if K has never been used in a register action. Note that this operation will succeed even after *Revoke*(K) has been performed, but will indicate that the key is revoked in its return value.
Revoke(K)	Revokes public key K. This operation fails if K has never been registered at the service.
Escrow(K^{-1})	Escrows the private key K^{-1} at the service. This operation fails if K, the corresponding public key, was not previously registered.
Decrypt(m, K)	Decrypts message m with K^{-1} and returns the result. This operation fails if K^{-1} has not been escrowed at the service.
Sign(m, K)	Returns the signature of m using K^{-1}. This operation fails if K^{-1} was not previously escrowed or if K has been revoked.
Recover(K)	Returns K^{-1}. This operation fails if K^{-1} was not previously escrowed.

–*escrow a private key* at the service;

–*decrypt a message using a private key* escrowed at the service;

–*sign a message using a private key* escrowed at the service; and

–fully *recover a private key* that was registered at the service.

To all these actions, which are summarized in Table I, other services could be added. We also note that all of these actions need not be supported for every registered key in a real KMS. For example, additional arguments to the *Register* action might limit the use of the registered key K to signature verification only. In this case, *Decrypt*(m,K) can be made to fail (even for those subjects that would otherwise be authorized to perform this action). And, presumably, other input arguments would be added to *Lookup*(p) to specify the type of key requested, and another output from *Lookup*(K) would indicate the intended use of the key. To minimize the complexity of these interfaces, however, here we omit further treatment of such functional restrictions on keys, though we realize that such restrictions are sometimes important in practice.

In the following, we use the term *key* to denote either the public or the private key—whether we refer to public key K or private key K^{-1} is clear from the context. A key can be registered for at most one principal. A principal has a single valid key registered for it at any given time (the principal's "current key"), and registration of a new key automatically revokes the one registered previously. However, different keys may be registered for a given principal at different times. We assume the KMS does not discard old keys, but maintains a history of all keys registered at the service (and to whom they were registered). If keys were simply discarded at their revocation, it would not be possible for subjects to decrypt messages encrypted with revoked keys or to verify a signature of a principal on a message signed before the key was revoked. As discussed in Section 4.2, revoked keys are available for restricted use, that is, only a subset of the actions in Table I can be exercised on old keys (for instance, a revoked key cannot be used to sign messages).

3. AUTHORIZATION MODEL: BASIC CONCEPTS

In the description of the key management service, we generally talk about *principals* as entities with whom keys stored at the service are associated. Besides those principals, the key management service has to recognize and reason about other types of principals, namely those submitting requests on the stored keys, and hence for which authorizations should be specified. In this section we discuss the basic entities distinguished by the authorization model. In the next section we discuss how they are considered in the characterization of entities presenting requests and how they relate to the keys stored at the service.

3.1 Users, Groups, and Roles

The access control model distinguishes the following concepts:

Users: Entities that can make requests directly at the service and correspond to user identifiers. Note that the term *user* is intended in the sense of user identifier and not "human user." A single human user may be associated with more than one identifier.

Roles: *Named* collections of privileges [Sandhu 1996]. Intuitively, a role identifies a *privileged hat* that users need to activate to perform specific organizational activities. Examples of roles can be `secretary`, `manager`, `programmer`, `project-head`, and so on. By assuming a given role, a user is able to exercise the privileges associated with the role. Multiple roles can be simultaneously active for a user.

Groups: *Named* collections of users.

Note the distinctions we make among these three concepts. Users correspond to identifiers associated with individuals connecting to the service; groups are sets of users; and roles are sets of privileges needed to perform specific activities. The basic difference between groups and roles is therefore that groups define groupings of people, while roles define groupings of privileges [Samarati and De Capitani di Vimercati 2001; Sandhu 1996].

The consideration of groups allows the specification of authorizations which hold for all the members of a group. This avoids the inconvenience of specifying authorizations for each individual user and keeping track of her membership in the group—upon termination of which, her authorizations should be revoked. To illustrate, consider the case where an access should be allowed to all users who are employed at `acme`. By grouping them in a group, `acme-employees`, and specifying the access authorization for the group, we avoid the need for specifying a different authorization for each employee. Also, the specified authorization will be applicable only to users for whom there is proof of membership in group `acme-employees`. Hence, if the membership of a user in the group ceases, the authorization granted to the group will automatically become nonapplicable to the user.

The consideration of roles allows the specification of authorizations that apply to users when performing certain activities. Intuitively, a role identifies and isolates specific privileges related to the execution of a particular job, granting them to users only when they need to execute the job. Consider, for example, a user `alice` who is `project-manager`. As `project-manager`, `alice` has special privileges, such as looking up budget information and signing contracts. However, these privileges should be available to `alice` only when executing project management activities and not, for instance, when accessing personal files.

A major difference between groups and roles is that roles can be "activated" and "deactivated" by users at their discretion, while group membership always applies [Samarati and De Capitani di Vimercati 2001]. Also, activation of roles must be controlled (a user should be allowed to activate a role only if she or he is authorized to do so) and may be subject to some restrictions. For instance, some roles may be incompatible with one another, and their simultaneous activation be forbidden, even if the user requesting it is allowed to activate each of the roles singularly [Ferraiolo et al. 2001]. Moreover, privileges granted to groups add to those of the users (i.e., by being member of a group a user can execute more actions than the user could as an individual).[1] By contrast, when playing a role, a user may not be able to execute the actions for which the user is authorized either as an individual or as a member of a group. The fact that the authorizations given to a role are applicable only when the role is active for the user has a double advantage. First, it allows the use of privileges needed to perform a task only within task execution (not granting them indiscriminately as would happen if authorizations were specified for users or groups). Second, it allows the enforcement of the *least privilege principle*, according to which each role (the user assuming it) is confined to the execution of only those actions needed to perform the task.

Note that given the different semantics between groups and roles, their identifiers are required to be disjoint. However, under the two different views, the same concept can be interpreted as a group or as a role. For instance, consider the concept `acme-manager`. A group `G_acme-manager` can be defined that collects all the users who are managers at `acme`. A role `R_acme-manager` can be defined to which specific privileges related to the managerial activity at `acme` are associated, and which `acme` managers can assume in order to carry out their jobs. This situation is completely proper and must not be considered either ambiguous or redundant. `G_acme-manager` groups together a set of people by collectively referring to them with this name. Members of this group will be allowed all the accesses that such membership implies. `R_acme-manager` will have associated authorizations for privileges necessary to perform the managerial activity. These privileges will be available to acme managers only when they have this role active.

In the following, we indicate with U, R, and G the set of identifiers of users, roles, and groups respectively, known at the key management service.

[1]Assuming that only positive authorizations are considered.

3.2 Subjects

In our model, entities that can present requests at the key management service can be any of the following:

—a user $u \in \mathsf{U}$;

—a user $u \in \mathsf{U}$ in a particular role $r \in \mathsf{R}$; and

—a conjunction of elements of any of the above type (conjuncted subjects).

Entities recognized at a system as sources of requests have often been referred to as principals [Saltzer and Schroeder 1975]. To avoid confusion between such entities and the principals associated with keys stored at the service (cf., Section 2), we use the term *subject* to refer to entities that issue requests to the service. The relationship between subjects (and elements composing them) and principals whose keys are stored at the service are discussed in Section 4. For the time being, for clarity, we keep the two concepts separated.

Intuitively, a conjuncted subject represents the situation where multiple subjects jointly present a request to the service. Conjuncted subjects are useful in cases where some subjects can execute an action when operating jointly but cannot when operating individually. This may happen when allowing an action to a single user (possibly in some roles) would give him too much privilege. As a simple example of conjunction of subjects, consider the case when two users in the role `secretary` are needed to sign a message with a manager's key.

By representing conjuncted subjects as sets and individual users as users associated with a *null* role, we can represent subjects in a uniform form, as captured by the following definition.

Definition 3.1 (*Subject*). Let U and R be the set of users and roles. A subject is a set, possibly singleton, of pairs of the form $[u, r]$, with $u \in \mathsf{U}$ and $r \in \mathsf{R} \cup \{\epsilon\}$, and where ϵ denotes the "null" role.

Hence, in the following, a subject is always a set of pairs where the first element is a user and the second element is either null or it is a role. A user (with no role) corresponds to a pair with second element null. Nonconjuncted subjects correspond to singleton sets.

Example 3.1. Let $U = \{$`alice,bob,chris`$\}$ and $\mathsf{R} = \{$`acme-manager,acme-administrator`$\}$ be the set of users and roles respectively. Some examples of subjects are as follows:

p_1: $\{$`[bob,`ϵ`]`$\}$
p_2: $\{$`[alice,acme-manager]`$\}$
p_3: $\{$`[bob,acme-administrator]`$\}$
p_4: $\{$`[alice,`ϵ`],[bob,`ϵ`],[chris,`ϵ`]`$\}$
p_5: $\{$`[alice,acme-manager],[bob,acme-administrator]`$\}$
p_6: $\{$`[bob,`ϵ`],[alice,`ϵ`],[chris,`ϵ`]`$\}$
p_7: $\{$`[alice,acme-manager],[alice,acme-administrator]`$\}$
p_8: $\{$`[alice,acme-manager],[bob,acme-manager]`$\}$

The order of component subjects in a conjuncted subject does not matter. Two conjuncted subjects are said to be *equal* if they contain the same elements. For instance, with reference to the subjects in Example 3.1, p_4 and p_6 denote exactly the same subject. By exploiting set notation, subject equality reduces to set equality.

For simplicity, Definition 3.1 restricts subjects to the form "single user in single role" or their conjunction. This restriction is purely syntactical, and therefore our definition encompasses subjects where a given user activates more roles or different users activate the same role. In particular, a given user u activating a set of roles $\{r_1, \ldots, r_n\}$ can be represented as conjuncted subject $\{[u, r_1], \ldots, [u, r_n]\}$, as in the case of principal p_7 in Example 3.1. Analogously, a set of users $\{u_1, \ldots, u_n\}$ activating a given role r can be represented as conjuncted subject $\{[u_1, r], \ldots, [u_n, r]\}$, as in the case of principal p_8 in Example 3.1.

For simplicity, in the following we omit set notation in the case of singleton sets (i.e., nonconjuncted subjects), and the square brackets in case of users (i.e., pairs with role element null), when such a simplification does not cause any confusion. For instance, subjects p_1 and p_2 may be denoted as `bob` and `[alice,acme-manager]`, respectively.

Note that groups, discussed in the previous section as one of the basic elements of the authorization model, do not appear in the definition of subject. The reason is that users, even if members of groups, always operate *individually*. This aspect is very important to guarantee accountability [Sandhu and Samarati 1997]. Also, the fact that roles appear in subjects, and groups do not, is consistent with the point we made that group membership (and the privileges it bears) always applies, while role activation is selectively chosen.

3.3 Access Requests

Access requests are requests by subjects to execute actions on keys stored at the service. In the remainder of this article, we characterize access requests as triples of the form ⟨`subject`, `action`, `key`⟩, stating that `subject` requests permission to execute `action` on `key`. Here, `subject` is any subject allowed by Definition 3.1; `action` is any action listed in Table I; and `key` is a public key identifier (e.g., a digest of the public key computed using a collision-resistant hash function). Note that requests can always be translated in this form, even if the key is not explicitly indicated by the requesting subject. This is possible because the server can always establish a correspondence between a principal and its private or public key. For instance, the operation *Lookup*(p), where p is a principal, can be translated into a triple where `key` is the key currently registered for p at the KMS.

For the sake of uniformity with other actions, we see a request for the `register` operation as a request on a key stored at the service. In particular, since the register operation modifies the key currently associated with a principal, we see the register request as a request on the key being modified. Clearly, we are dealing here with keys stored at the service, and whose principal is therefore already known at the service. Registration of *new* keys, or more precisely *new* principals, requires separate treatment, and is addressed in Section 7.2. Also, for the time being, we ignore the problem of authenticating subjects presenting requests; this issue is addressed in Section 7.1.

Each request is evaluated by the KMS and allowed or denied according to the KMS access control policy. The access control policy determines the actions to be allowed or denied on keys according to basic privileges that must be warranted (e.g., a key's owner[2] should always be able to revoke the key), as well as to authorizations established by the owner of a key that grants other subjects the privilege of executing actions on the key. Section 4 discusses the basic privileges that the KMS must warrant to principals that store their keys at the service. Section 5 discusses authorizations.

4. ACCESS AND ADMINISTRATIVE POLICIES

To describe the basic privileges that the KMS must allow on keys stored at the service to subjects that can request action from them, we must first clarify the relationship between principals (cf., Section 2) and subjects (cf., Section 3.2).

Consistent with other proposals [Lampson et al. 1992], we assume that only users and roles have an associated key registered at the service. In other words, a principal is either a user or a role. Each user/role p can have an associated pair (K_p, K_p^{-1}) indicating its current public and private keys. The pair of keys associated with a principal can change over time due to a new `register` operation, but only one key at most is valid at any given time (a new register operation has the automatic effect of revoking the previous key). Note that the assumption that each principal has at most one associated key at a time restricts each user identifier to at most one key, but it does not necessarily imply that each human user will have only one associated key. As a matter of fact, each given human user may correspond to more user identifiers. For instance, user `alice` can register at the service as two different principals `alice_at_work` and `alice_at_home`, each with its own key.

In addition to principals, our model supports two additional concepts, namely *ownership* and *authority*, which tie subjects to keys. These concepts are discussed in the next section.

4.1 The Key's Principal, Ownership, and Authority

The following three concepts describe the relationships that can relate subjects to keys stored at the service.

Principal. As discussed above, each key pair (K, K^{-1}) corresponds to one principal. The principal associated with a key can either be a user or a role. As we discuss in Section 7.1, the KMS will use the keys stored at the service to authenticate the identity of subjects presenting requests to the service when such a key is available.

Ownership. Each key has associated with it one or more owners. The owner of a key can exercise any action on the key as well as grant this ability to other

[2]The precise meaning of the term "owner" will be clear in the next subsection.

subjects. For keys corresponding to users, the owner is usually unique and coincides with the principal of the key. The owner may be different from the principal in the case of a key whose principal is a role. If the role is considered as owner of the key and allowed to manage it, the role identifier should also be interpreted as a user identifier (in which case $\mathsf{U} \cap \mathsf{R} \neq \emptyset$). This would allow the role as such to make statements, and therefore interact with the service to manage its key. However, we can usually expect that a different subject will be considered owner of keys associated with roles. In such a case, the owner of a key can be any subject. Also, we expect the owner of the key to be a single (possibly conjuncted) subject; but a set of subjects (*shared ownership*) is also possible.

Authority. Our model also supports the concept of *authority* over a key. The authority of a key can be any (simple or conjuncted) subject. Like ownership, more than one subject may have authority over a key. A subject with authority over a key can always execute `lookup`, `revoke`, and `decrypt` actions on the key. Intuitively, a subject with authority over a key can control how the key is used and can revoke the key. The authority cannot however "impersonate" the principal (cannot execute `recover` or `sign` actions) or interfere with its activity. For instance, it cannot re-`register` the key or grant/revoke privileges on the key to/from others.

There are different reasons why authority and ownership of a key can remain with different subjects. To illustrate, consider an organization `acme` whose employees are registered at the service. Suppose that each employee has (is the principal of) a key. Depending on why the keys are to be used, the organization may wish to maintain authority over the key to be able to control how the keys are used and, possibly, revoke them. Each employee will be considered the owner of his or her key and, as such, granted all the accesses that this privilege implies. However, at any time, `acme` will be able `lookup` the key, `decrypt` messages encrypted with the key, and `revoke` the key. The organization can use this last privilege when, for instance, an employee leaves the job. Authority can also vary in the case of keys whose principals correspond to the same user. For instance, consider user `alice` who is registered at the service as different principals: `alice`, `alice_at_work`, `alice_at_home`. Although the principals (and the owners) of the keys are different, the subject(s) with authority over the keys can be the same. For instance, `alice` may have the authority on all the three keys.

When more than one subject has ownership/authority on a given key, we say that the ownership/authority is *shared*. Shared ownership/authority represents the fact that multiple subjects can exercise ownership/authority privileges. Each of the subjects can exercise this privilege independently of the others. For instance, there can be a key `bob_and_alice` on which both users `alice` and `bob` have authority. The ownership/authority over a key can also be a conjuncted subject. For instance, subject {`jim,tim`} can have authority on key `jim_and_tim`. In the case of shared authority, any of the principals (either `bob` or `alice` in the example above) can exercise authority. In the case of authority by a conjuncted subject, authority can be exercised by the subjects in the conjunct only when operating *jointly*. With reference to our example, only a conjuncted subject {`jim`,`tim`} can exercise authority, while neither `jim` nor `tim` individually can.

The specification of the subject(s) with ownership and authority over a key is made at the time the first registered key for the principal, which involves human control and interaction (Section 7.2). Modifications of a key due to new register operations do not affect ownership/authority. In other words, a subject with ownership (resp. authority) on a key being revoked acquires ownership (resp. authority) on the new key. This aspect is particularly important, for instance, in the case of an organization's controlled keys, like those discussed above: the organization should maintain authority over an employee's key despite modifications to the key by the employee. To this purpose, ownership and authority information at the KMS is specified with respect to principals (users and roles), rather than with respect to their specific keys. We discuss the rules establishing ownership and authority in Section 6.

4.2 Administration and Access Control Policies

The support of ownership and authority implicitly dictates some basic services that the KMS must provide. In particular: the owner of a key must be able to exercise all the actions on the key; the authority for a key must be able to exercise all the actions on the key that this privilege implies. Besides basic accesses, other accesses may need to be provided on the keys stored at the service. As a matter of fact, one of the reasons for a key management service is to allow owners to make their keys available to others. The owner of a key must therefore be able to grant privileges on her key to others. Consequently, the KMS must grant, beside the accesses implied by ownership and authority privileges, all the accesses that owners wish to be granted. At the same time, to keep the trust of the owners that store their keys at the service, the KMS must ensure that no other accesses will be permitted. It must also guarantee that only operations that can be executed will be granted (for instance, no `sign` action can be executed on revoked keys).

We start by defining the actions that can be allowed on keys and the access privileges warranted by ownership and authority.

Definition 4.1 (*Allowed Action*). The following actions are allowed on keys:

—Current keys: `register`, `lookup`, `escrow`, `decrypt`, `revoke`, `recover` and `sign`.
—Revoked keys: `lookup`, `decrypt`, `recover`.

In other words, the KMS access control will permit on current keys all the actions supported by the service, provided that the subject is authorized for it. It will, however, permit only a subset of such actions on revoked keys. Revoked keys can still be available for lookup, decrypt, and recovery services. However, since they are no longer valid, the system will not permit their use for `signing` messages. They also cannot be `revoked`, re-`registered`, or `escrowed`.

Definition 4.2 (*Ownership/Authority Privileges*). The following privileges are warranted by ownership and authority.

Ownership: The owner of a key can
execute all the actions allowed on its key(s);
grant the privilege of exercising any actions on its keys;
revoke the privilege of exercising any actions on its keys.

Authority: The authority of a key can
execute the `lookup`, `decrypt`, and `revoke` actions when the key is current;
execute the `lookup` and `decrypt` actions when the key is revoked.

Definition 4.2 simply makes explicit the concepts of ownership and authority discussed in Section 4.1.

The access control policy can then be stated as follows:

POLICY 1 (*Access Control Policy*). A subject can execute an action on a key only if both of the following conditions hold:

(1) the action is allowed (Definition 4.1);
(2) the subject can exercise the action due to ownership, authority, or in virtue of an authorization granted by a key's owner (Definition 4.2).

All other accesses that do not satisfy the conditions above must be denied (closed-world assumption).

We have adopted the ownership policy for administration: only the owner(s) of a key can grant (and revoke) authorization to others to exercise actions on the key. We note that delegation policies could be considered by which the owner can grant, together with authorizations to execute an action, the ability to grant such privileges to others. The reason for not delegating administration in our model is twofold. First, for the specific objects (keys) and actions we consider, delegating administration does not seem to be needed: we imagine that the owner of the key wishes to retain control of those who can access and use the key. Second, as we will see shortly, our language allows specification of authorizations supporting variables and conditions, resulting in expressiveness and flexibility in specifications. However, coupling delegation management with expressiveness introduces complications. In particular, the revoke operation (not trivial in the case of chains of delegation [Samarati and De Capitani di Vimercati 2001]) would be complicated further, as revoking authorization to administer an access may correspond to the partial invalidation of an authorization granted through it, instead of a complete revocation. This requires the system to execute, at access control time, the cumbersome task of recursively determining if a chain of delegations for an authorization still holds.

In the next section we illustrate the authorizations through which owners can grant access on their keys to others. In Section 6 we illustrate how the authorizations so specified, together with ownership and authority data, are used in enforcing access control.

5. THE AUTHORIZATION SPECIFICATION LANGUAGE

Any access control system requires the support of rules (authorizations) that specify the actions that subjects can exercise on objects (keys in our case). In our framework, traditional authorizations would then correspond to triples of the form ⟨`subject`,`action`,`key`⟩, whose semantics is that `subject` can execute `action` on object `key`, i.e., request ⟨`subject`,`action`,`key`⟩ should be granted (Section 3.3). This framework, although used effectively in systems where subjects are not conjuncted and objects are information containers, in our context is limiting for two reasons. First, unlike object names, keys are subject to relatively frequent modifications. This introduces some complications. For instance, how should keys be referenced? Should the authorizations specified on a key still be valid when the key is modified? Second, composite and conjuncted subjects introduce several interesting issues about how authorizations should be interpreted. For instance, should the authorizations specified for a user be considered valid when the user is in some roles? Should the authorization specified for a subject be considered valid when the subject is conjuncted with other subjects?

Models that consider authorizations in the form of simple triples as above give a specific interpretation to the authorizations, which may limit expressiveness in some cases. This is true even for models semantically very rich, such as that of Abadi et al. [1993]. Our model provides a simple, yet flexible, language capable of representing the different answers that can be given to the questions above. In particular, the language allows the specification of authorizations referring to specific keys or to all keys satisfying some conditions, and/or to specific subjects or to all subjects satisfying some conditions. In the remainder of this section we discuss these issues in more detail and illustrate the authorizations supported by our model.

5.1 Authorization Objects

Traditional access control systems are based on authorizations to exercise actions on specific objects. This paradigm, which works well in environments where objects are relatively static, may be limiting when objects are keys, which are, by contrast, subject to change. A natural way to overcome this limitation is to specify authorizations on a key with reference to the key's principal, rather than the key identifier. This way, changes to the key do not affect authorizations, which will be "propagated" to the new key. However, this approach may also be limiting in certain situations (where authorizations are intended to apply to a specific key, and should not be applied to new keys registered in its place). Our model supports both ways of referring to keys, thus allowing the specification of authorizations on keys with reference to either the key identifier (digest) or the principal of the key. Authorizations referred to a key digest hold only for that specific key and do not "propagate" to the new key when the key is modified. Authorizations specified with respect to the principal propagate instead to the new key upon reregister operations.

Regardless of whether authorizations on a key propagate or not to the new key, another issue to be addressed is whether the authorizations specified for a key should still hold when the key is revoked. As discussed in Section 2, revoked keys can remain available at the service for restricted use. Beside being available to their owners and authorities, revoked keys could also remain available to other subjects (who can then still be able to verify a principal's signature on an old message signed with a key that has been revoked). Again, no answer is good for all scenarios, and while in some cases authorizations on a key should continue to hold even after the key is revoked, in other cases they should be invalidated upon key revocation. Hence, our model gives the owner specifying an authorization on its key the ability to state whether the authorization should continue to hold once the key is revoked.

5.2 Authorization Subjects

Authorization subjects are the entities for which authorizations to exercise actions on keys stored at the service are specified. Approaches proposed in the context of public key systems and certification authority management tend to identify authorization subjects with their keys, thus associating permissions with keys rather than user identifiers [Blaze et al. 1997; Ellison 2000]. This solution has two main drawbacks. First, granting privileges to keys does not allow the enforcement of expressive and flexible access control policies that take group membership and roles into consideration. Second, unlike user identifiers, keys are subject to modification, which results in the automatic invalidation of all the key's privileges and in the corresponding principal's inability to exercise them.

It should be clear from Section 3 that our model assumes the granting of authorizations to subjects with reference to the subjects' identifiers. This way, a subject's authorizations continue to hold despite changes to the keys with which the subject is authenticated (see Section 7.1). We note, however, that our language, allowing the specification of variables and conditions, is also able to support authorizations to exercise actions which are granted to "a specific key," or better, to the principal corresponding to that key.

Roles and conjuncted subjects introduce several questions regarding the interpretation of authorizations. Questions include the following:

(1) Can authorizations explicitly refer to users in roles, or should they instead refer to either users or roles, where authorizations of a role are applicable to every user in that role? For instance, should authorizations be specified for `acme-manager` regardless of the user who assumes the role, or can they refer to a specific user in that role, like `[alice,acme-manager]`?
(2) Should the authorizations specified for a user be considered valid when the user is in some roles? For instance, should an authorization specified for `alice` be valid for `[alice,acme-manager]` also?
(3) Should the authorizations specified for a user in a role be considered valid for the user itself? For instance, should an authorization specified for `[alice,acme-manager]` be considered valid for `alice` also?
(4) Should an authorization specified for a subject s be considered valid for conjuncted subjects containing s as a conjunct? For instance, should an authorization specified for `bob` be considered valid for `{bob,alice}`?

Different scenarios can be imagined where various ways of answering the questions above could be supported. Therefore, instead of including specific policy decisions in the model itself, we allow subjects to express the approach they want to be applied through the authorization language.

5.3 Basic Elements of the Language

The authorization language is based on the following alphabet:

(1) **Constant symbols:** Every member of $\mathsf{U} \cup \mathsf{R} \cup \mathsf{G} \cup \mathsf{S} \cup \mathsf{K} \cup \mathsf{A}$. Where U is the set of users, R is the set of roles, G is the set of groups, K is the set of key identifiers (digests), A is the set of actions, and S is the set of all possible subjects that can be obtained from U and R.

(2) **Variable symbols:** There are five sets of variable symbols V_u, V_r, V_g, V_k, V_s ranging over the sets U, R, G, K, and S, respectively.
In future, the following terminology will be used. Variables in V_u and members of U are user terms. Variables in V_r and members of R are role terms. Variables in V_g and members of G are group terms. Variables in V_k and values in K are key terms.

(3) **Predicate symbols:** We assume the following predicate symbols:
   (a) A binary predicate symbol memberof, whose first argument is a user term and whose second argument is a group term. It expresses the membership relationships between users and groups. Given a user u and a group G, if memberof(u, G) is `true` then it means that u is a member of group *G according to the membership information known at the KMS* (cf., Section 7).
   (b) A binary predicate symbol principal, whose first argument is a key term and whose second argument is a principal term. It captures the relationship between principals and their keys (cf., Section 4.1).
   (c) A unary predicate symbol current, whose argument is a key term. It tests whether a key is currently valid or has been revoked.
   (d) A binary predicate subsequent, whose arguments are key terms (one of which must necessarily be ground). It captures the time relationship between keys registered for a principal.
   (e) Three basic symbols $=$, $\in$, $\subseteq$, expressing equality, inclusion, and set containment. By exploiting the representation of subjects as sets, we use these operations to test equality, inclusion, and containment between subjects.

Borrowing from the logic programming terminology, if p is one of the above predicate symbols with arity n, and $t_1, \ldots, t_n$ are terms appropriate for p (as defined above), we refer to $p(t_1, \ldots, t_n)$ as an *atom*. We use the expression *literal* to denote an atom or its negation. For instance, if PT is a principal term and KT is a key term, then principal(KT, PT) and $\neg$principal(KT, PT) are *literals*. We call a literal containing predicate *pred* a *pred*-literal, e.g., principal-literal. The literal is said to be negative or positive according to whether or not the predicate appears negated. We refer to constants as *ground* terms. A subject/literal is said to be *ground* if it contains only ground terms.

Note that since subjects can be composed of different elements, the case can be where both constants and variables are used to denote a subject. Therefore, subject terms can include, at the same time, both values and variables.

Example 5.1. Let U = {`alice`,`bob`} be the set of users and R = {`acme-manager`, `acme-administrator`} be the set of roles. Some examples of nonground subject terms are as follows.

s	(any subject);
{[`alice`, r]}	(user `alice` in any role);
{[s, `acme-administrator`]}	(any user in role `acme-administrator`);
{[s, r]}	(any user in any role);
{[`alice`, ϵ], [s, `acme-administrator`]}	(a conjunct composed of user `alice` and any user in role `acme-administrator`).

Definition 5.1 (*Satisfaction of predicates*). Satisfaction of ground predicates is as follows:

—memberof(u, g) is satisfied for $u \in$ U and $g \in$ G iff user u is a member of group g (w.r.t. the membership information known at the KMS);

—principal(p, k) is satisfied for $p \in$ U ∪ R and $k \in$ K iff key k was registered for p;

—current(k) is satisfied for $k \in$ K if k has been registered and is not revoked.

—subsequent(k, k') is satisfied for $k, k' \in$ K iff key k and k' were registered for the same principal and either $k = k'$ or k was registered after k'.

—A literal $\neg L$ is satisfied iff L is not satisfied.

A nonground literal L is satisfied iff there exists a ground instance of L, i.e., an instance obtained by substituting values in place of variables, which is satisfied.

The predicates above can be used to grant authorizations to all keys and/or all subjects that satisfy given conditions. Conditions on keys or subjects are specified through conditional expressions, defined as follows.

Definition 5.2 (*Key conditional expression*). A key conditional expression over a key term k is a (possibly negated) literal of the form principal(p, k), current(k), subsequent(k, k'), subsequent(k', k), or a conjunction of such (possibly negated) literals.

Definition 5.3 (*Subject conditional expression*). A subject conditional expression is a conjunction of (possibly negated) literals of the form memberof, principal, current, subsequent, $=$, $\in$, $\subseteq$.

Definition 5.4 (*Satisfaction of conditional expressions*). A key/subject conditional expression E that contains only ground literals is satisfied iff all the literals in E are satisfied. A key/subject conditional expression E whose literals contain some variables is satisfied iff there exists a possible instantiation of the variables, that is, a substitution Θ of values to variables in E such that the resulting expression $E\Theta$ is satisfied.

We refer to the pair composed by a key (resp. subject) term and a key (resp. subject) conditional expression as a key (resp. subject) expression.

Example 5.2. Some examples of key/subject expressions follow:

—**Key term:** k. **Expression:** principal(`bob`, k).
Satisfied by all keys k that have `bob` as principal (of which one will be the current valid key, while the others will be past revoked keys).

—**Key term :** k. **Expression:** principal(`bob`, k) & subsequent(k, `0xA40B96C7`).
Satisfied by all keys k that have `bob` as principal and have been registered for `bob` after key `0xA40B96C7` (key `0xA40B96C7` included).

—**Key term:** `0xA40B96C7`. **Expression:** current(`0xA40B96C7`).
Satisfied by key `0xA40B96C7` when not revoked.

—**Subject term:** {[`alice`, r]}. **Espression:** `true`.
Satisfied by user `alice` in any role.

—**Subject term:** {[u, `manager`], [u', `manager`]}. **Expression:** $u \neq u'$.
Satisfied by any subject that is a conjunct of two distinct users in role manager.

—**Subject term:** {[u, ϵ]}. **Expression:** principal(u, `0xA40B96C7`) & current(`0xA40-B96C7`).
Satisfied by the user who is the principal of valid (i.e., unrevoked) key `0xA40B96C7`.

—**Subject term:** s. **Expression:** $\{[u, \epsilon]\} \subseteq s$ & memberof(u, `acme-employee`) & memberof(u, `US-citizens`).
Satisfied by subjects that are either (1) a user member of groups `acme-employee` and `US-citizens` or (2) a conjunct containing such a user.

Having given the basic elements and constructs of the language, we are now ready to introduce authorizations.

5.4 Authorizations

The basic elements introduced in the previous section allow a key's owners to specify authorizations. Authorizations, which can be specified and revoked through a simple language like the one in Figure 1[3] are defined as follows:

Definition 5.5 (*Authorization*). An authorization is a 6-tuple composed of the following elements:

subject: is a subject term;

action: is an action in A;

key: is a key term;

k-cond: is either the value `true` or a key conditional expression over the key term;

s-cond: is either the value `true` or a subject conditional expression;

grantor: is the *ground* subject that specified the authorization where `k-cond` and `s-cond` have no variables in common.

[3]In the syntax, `key-term`, `user-term`, and `role-term` are key, user, and role terms, as discussed above; `subj-entity` is either a user, a role, or a group term; and `variable` is any variable.

```
command          ::= ⟨grant⟩ | ⟨revoke⟩
⟨grant⟩          ::= GRANT ⟨action⟩ ON ⟨key-term⟩ TO ⟨subj-term⟩
                     [WHERE [KEY IS ⟨k-cond⟩] [SUBJECT IS ⟨s-cond⟩]]
⟨revoke⟩         ::= REVOKE ⟨authorization-id⟩
⟨action⟩         ::= register | lookup | revoke | escrow | decrypt | recover | sign
⟨subj-term⟩      ::= ⟨variable⟩ | '{''['⟨user-term⟩, ⟨role-term⟩']' [,'['⟨user-term⟩,⟨role-term⟩']']* '}'
⟨k-cond⟩         ::= [not]⟨key-pred⟩ ['&'[not]⟨key-pred⟩]*
⟨s-cond⟩         ::= [not]⟨subj-pred⟩ ['&'[not]⟨subj-pred⟩]*
⟨key-pred⟩       ::= current(⟨key-term⟩) | subsequent(⟨key-term⟩,⟨key-term⟩) |
                     principal(⟨principal-term⟩,⟨key-term⟩) | ⟨key-term⟩ '=' ⟨key-term⟩
⟨subj-pred⟩      ::= ⟨key-pred⟩ | memberof(⟨user-term⟩,⟨group-term⟩) | ⟨subj-term⟩ '='
                     ⟨subj-term⟩ | ⟨subj-term⟩ '⊆' ⟨subj-term⟩ | ⟨subj-entity⟩ '='
                     ⟨subj-entity⟩ | ⟨subj-entity⟩ '∈' ⟨{subj-entity,[subj-entity]+}⟩
⟨subj-entity⟩    ::= ⟨user-term⟩ | ⟨role-term⟩ | ⟨group-term⟩
```

Fig. 1. The authorization specification language.

The semantics of an authorization of the form above is that subject can exercise action on key, provided that key satisfies conditional expression k-cond and subject satisfies conditional expression s-cond. Note that, as discussed in Section 3.3, the semantics of register authorizations referred to a key is that the subject can register a new key in place of key (i.e., the subject can modify the key). Therefore, register authorizations will generally have a variable for the key, that is, they will refer to the key of a principal instead of a specific key.

The grantor field reports the subject who specified the authorization and is always *ground*, that is, no variables appear in it. This aspect is important for accountability, as well as to restrict the applicability of authorizations to keys on which the grantor has ownership privileges. Given the possibility for subjects to specify authorizations with variables in place of keys, this constraint cannot be enforced when authorizations are specified, but must be enforced at access control time. Intuitively, from the point of view of the semantics of authorizations, this constraint can be enforced easily by adding predicate owner(grantor, key) to the key condition k-cond, restricting the keys to which the authorization applies. For the time being, we focus on the authorizations specified and ignore the issue of restricting their applicability according to the subjects who specified them. We discuss the latter issue in the next section.

The semantics of authorizations is better clarified with reference to the requests they authorize, as stated by the following definition.

Definition 5.6 (*Authorized Request*). A request ⟨subject′,action′,key′⟩ is authorized by an authorization ⟨subject, action, key, k-cond, s-cond, grantor⟩ iff action′ and action are the same action, and

—there exists a substitution Θ_k of ground terms for variables such that keyΘ_k=key′ and k-condΘ_k is satisfied; and

—there exists a substitution Θ_s for variables in s such that subjectΘ_s=subject′ and s-condΘ_s is satisfied.

	Subject	Action	Key	K-cond	S-cond	Grantor
A1	$\{[\texttt{tim}, \epsilon]\}$	decrypt	0xA40B96C7	true	true	$\{[\texttt{bob}, \epsilon]\}$
A2	$\{[\texttt{tim}, \epsilon]\}$	decrypt	0xA40B96C7	current(0xA40B96C7)	true	$\{[\texttt{bob}, \epsilon]\}$
A3	s	lookup	k	principal(k,bob) & current(k)	true	$\{[\texttt{bob}, \epsilon]\}$
A4	$\{[u, \epsilon]\}$	lookup	k	principal(k,bob)	memberof(u,acme-people)	$\{[\texttt{bob}, \epsilon]\}$
A5	$\{[u, \epsilon]\}$	decrypt	k	principal(k,bob) & subsequent(k, 0xA40B96C7)	memberof(u,acme-people)	$\{[\texttt{bob}, \epsilon]\}$
A6	$\{[u, \texttt{manager}], [u', \texttt{manager}]\}$	sign	0xA40B96D9	true	$u \neq u'$	$\{[\texttt{bob\_as\_director}, \epsilon]\}$
A7	$\{[u, \texttt{manager}], [u', \texttt{manager}]\}$	sign	k	principal(k,bob_as_director)	$u \neq u'$ & memberof(u,bob-friends)	$\{[\texttt{bob\_as\_director}, \epsilon]\}$
A8	$\{[u, \epsilon]\}$	sign	k	principal(k,bob)	principal(0xA40B96D9,u) & current(0xA40B96D9)	$\{[\texttt{bob}, \epsilon]\}$

Fig. 2. Some examples of authorizations.

Figure 2 illustrates some examples of authorizations that can be expressed in our model. The semantics of the authorizations is as follows:

(**A1**) User `tim` can exercise action `decrypt` using key `0xA40B96C7`. If a new key is registered for the principal associated with `0xA40B96C7`, `tim` will not be able to exercise the action on the new key; `tim` will however be able to decrypt with key `0xA40B96C7` even after the key has been revoked.

(**A2**) Same as **A1**, but `tim` will not be able to decrypt with key `0xA40B96C7` after the key has been revoked.

(**A3**) Any subject can lookup the *current* key registered for principal `bob`, whatever this key is. When the key changes, the authorization will be applicable to the new key just registered. It will no longer apply to the key that has been revoked.

(**A4**) Users who are members of group `acme-people` can lookup the key registered for principal `bob`. The authorizations also apply to expired keys (allowing subjects to lookup past keys).

(**A5**) Members of group `acme-people` can `decrypt` with the key whose principal is `bob`. It applies to the current as well as past keys (so that if the key is revoked, subjects can still exercise their privileges on it). The condition requiring the key to be subsequent to `0xA40B96C7` (which we assume to be the key of `bob` at the time the authorization is specified) avoids a retroactive effect of the authorization: the authorization does not hold on keys that have been registered for the principal prior to the registration of `0xA40B96C7`, and therefore the granting of the authorization.

(**A6**) Any two distinct users operating in the `manager` role can sign with key `0xA40B96C7`. Note that even if not specified in the authorizations, the authorized operation will be allowed only when the key is current, since action `sign` cannot be executed on revoked keys.

(**A7**) Any two distinct users operating in the `manager` role, at least one of which is a member of group `bob-friends`, can `sign` with the key whose principal is `bob_as_director`.

(**A8**) The user principal of key `0xA40B96C7`, when this key is valid, can sign messages with the key of `bob`.

Note the difference between authorizations with key element ground (e.g., authorization A1 above) and authorizations with key element variable (e.g., authorization A3 above). Authorizations with key element ground refer to a *specific* key, which is registered for a principal at a specific time. Authorizations with key element variable refer to the key of a principal (specified in the conditional expression), regardless of the specific value of the key. As already discussed, the use of variables allows the expression of authorizations on a key, or more precisely on the key of a principal that will continue to hold even if the key changes (because the principal has reregistered). For both cases, whether the authorizations should continue to hold on the revoked key depends on whether the key is required by the key expression to be current (e.g., authorization A1 vs authorization A2 above).

Note also how the inclusion of predicates over keys in the subject conditional expression allows the granting of authorizations to keys, in contrast to principals (e.g., authorization A8 above), thus subsuming approaches applying such an assumption (e.g., Blaze et al. [1997]; Ellison [2000]).

6. REPRESENTATION OF AUTHORIZATIONS AND ACCESS CONTROL

In this section we illustrate how the access control policies discussed in Section 4 and the authorizations in Section 5 can be expressed and enforced at the KMS through a simple, yet expressive, logic language. The logic language captures ownership and authority definitions (stated directly at the service), authorizations stated by the keys' owners, which must be obeyed at the service, and consequent enforcement of the access control and administration policies.

The logic language is based on the variables and predicates of the authorization language, whose satisfactions have been discussed already (Definition 5.1), plus the following predicates.

(1) A binary predicate allowed, whose first argument is an action term and whose second argument is a key term. It states the actions that can be executed on the key.
(2) A binary predicate owner, whose first argument is a subject term and whose second argument is a key term. It states the ownership of a key.
(3) A binary predicate authority, whose first argument is a subject term and whose second argument is a key term. It states the authority for a key.
(4) A ternary predicate authorize, whose first argument is a subject term, whose second argument is an action, and whose third argument is a key identifier. Intuitively, authorize predicates represent accesses that must be granted due to authority/ownership privileges or to specified authorizations.

Satisfaction of the allowed predicate is established by Definition 4.1; that is, allowed(a, k) evaluates true if a is an action allowed on *k at the time the predicate is evaluated*. According to Definition 4.1, the predicate evaluates true on every action when key k is current; it evaluates true on the subset of actions allowed on expired keys when key k is not current.

Satisfaction of the other three predicates is established through three different kinds of rules: *ownership* and *authority* rules, establishing ownership and authority relationships between subjects and keys, and *authorization* rules, representing authorizations and establishing accesses to be granted as a consequence of the authorizations, as well as of the ownership/authority

—**Ownership and authority establishment**
For each principal `princ`, registering for the first time at the service. For each subject s_i and for each subject s_j to have ownership and authority over the key registered for `princ` (see Definitions 6.1 and 6.2):
$\text{owner}(s_i, k) \leftarrow \text{principal}(\texttt{princ}, k) \,\&\, L_1 \,\&\, \dots \,\&\, L_n.$
$\text{authority}(s_j, k) \leftarrow \text{principal}(\texttt{princ}, k) \,\&\, L_1 \,\&\, \dots \,\&\, L_n.$

—**Ownership enforcement**
$\text{authorize}(s, a, k) \leftarrow \text{owner}(s, k) \,\&\, \text{allowed}(a, k).$

—**Authority enforcement**
$\text{authorize}(s, a, k) \leftarrow \text{authority}(k, s), a \in \{\texttt{lookup}, \texttt{decrypt}, \texttt{revoked}\} \,\&\text{allowed}(a, k).$

—**Authorization enforcement**
For each authorization ⟨`subject`,`action`,`key`,`k-cond`,`s-cond`,`grantor`⟩,
the following authorization rule inserted in P_{AS} :
$\text{authorize}(\texttt{subject},\texttt{action},\texttt{key}) \leftarrow \texttt{k-cond} \,\&\, \texttt{s-cond} \,\&\, \text{allowed}(\texttt{action},\texttt{key}) \,\&\, \text{owner}\,(\texttt{grantor},\texttt{key}).$

Fig. 3. Rules establishing ownership/authority and enforcing access control at the service.

relationships defined. The definition of these rules at the service is as summarized in Figure 3 and discussed in the remainder of this section.

Definition 6.1 (*Ownership Rule*). An *ownership rule* is a rule of the form:

$$\text{owner}(s, k) \leftarrow \text{principal}(p, k) \,\&\, L_1 \,\&\cdots\&\, L_n$$

where s is a subject term, p is a *ground* principal term, k is a variable term, and $L_1, \dots, L_n$ are literals of the language.

Definition 6.2 (*Authority Rule*). An authority rule is a rule of the form:

$$\text{authority}(s, k) \leftarrow \text{principal}(p, k) \,\&\, L_1 \,\&\cdots\&\, L_n$$

where s is a subject term, p is a *ground* principal term, k is a variable term, and $L_1, \dots, L_n$ are literals of the language.

In general, we can expect ownership and authority rules to have a ground subject on the left-hand side and only the principal-literal on the right-hand side, giving, for example, each user the ownership/authority on the key registered for her or him. However, richer forms of specification can be possible, as for instance, when establishing authority for keys associated with roles.

Example 6.1. The following are examples of ownership and authority rules.

(R1) $\text{owner}(\{[\texttt{bob}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{bob}, k).$
(R2) $\text{authority}(\{[\texttt{bob}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{bob}, k).$
(R3) $\text{owner}(\{[\texttt{bob-at-work}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{bob-at-work}, k).$
(R4) $\text{authority}(\{[\texttt{acme}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{bob-at-work}, k).$
(R5) $\text{owner}(\{[u, \texttt{acme-manager}], [u', \texttt{acme-manager}]\}, k) \leftarrow \text{principal}(\texttt{acme-manager}, k), u \neq u'.$
(R6) $\text{authority}(\{[\texttt{acme-headoffice}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{acme-manager}, k).$
(R7) $\text{authority}(\{[\texttt{acme-admin}, \epsilon]\}, k) \leftarrow \text{principal}(\texttt{acme-manager}, k).$

According to these rules:

—`bob` has ownership and authority on the key where he is principal (rules **R1** and **R2**).

—`bob-at-work` is the owner of the key of where he is principal, but `acme` has authority over this key (rules **R3** and **R4**).

—Any two distinct users in role `acme-manager` can exercise ownership privileges on keys register for `acme-manager` (rule **R5**). Both `acme-headoffice` and `acme-admin` have authority (shared authority) over such keys (rules **R6** and **R7**).

Ownership and authority rules for a principal (keys associated with it) are defined at the service at the time of the principal first-time registration (Section 7.2). In no way can they be specified, modified, deleted, or disobeyed by subjects using the service. For each principal registered at the service, at least one ownership and one authority rule must be present. Multiple rules establishing ownership (resp. authority) on a given key correspond to shared ownership (resp. authority).

Definition 6.3 (*Authorization Rule*). An authorization rule is a rule of the form:

$$\mathsf{authorize}(s, a, k) \leftarrow L_1 \,\&\, \cdots \,\&\, L_n.$$

where s, a, k are a subject term, an action term, and a key term, respectively, and $L_1, \ldots, L_n$ are literals.

Authorization rules determine the accesses that the system must grant or deny according to the Access Control Policy (Policy 1). Two different kinds of rules can be distinguished: those enforcing authority/ownership privileges and those enforcing authorization rules. Rules of the first kind simply derive the accesses to be warranted in virtue of the ownership/authority privileges. One rule for ownership and one rule for authority are specified at the KMS; they apply to every subject and key stored at the system (Figure 3). Rules of the second kind are derived from authorizations stated by the subjects. One rule is inserted for each authorization specified. The rule is simply a translation of the authorization in logic form with an allowed and an owner predicate added to the body, which enforce the restrictions demanded by the access control policy. The owner predicate restricts the applicability of the authorizations to keys that the grantor owns. The allowed predicate ensures that only actions allowed by the service will be granted. As an example, Figure 4 illustrates the translation into authorize rules of the authorizations specified in Figure 2.

Given the ownership, authority, and authorization rules specified, the access control policy can be easily expressed in terms of evaluation of the corresponding logic program (Figure 3), as follows:

ALGORITHM 6.1 (*Access Control Algorithm*). Let P_{AS} be the logic program defined at the service according to the ownership/authority statements established at registration and the authorizations granted by subjects. Let

R1) $\text{authorize}(\{[\mathtt{tim},\epsilon]\},\mathtt{decrypt},\mathtt{0xA40B96C7}) \leftarrow \text{allowed}(\mathtt{decrypt},\mathtt{0xA40B96C7})$
$\quad\& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},\mathtt{0xA40B96C7}).$

R2) $\text{authorize}(\{[\mathtt{tim},\epsilon]\},\mathtt{decrypt},\mathtt{0xA40B96C7}) \leftarrow \text{current}(\mathtt{0xA40B96C7})$
$\quad\& \ \text{allowed}(\mathtt{decrypt},\mathtt{0xA40B96C7}) \ \& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},\mathtt{0xA40B96C7}).$

R3) $\text{authorize}(s,\mathtt{lookup},k) \leftarrow \text{current}(\mathtt{0xA40B96C7})\text{principal}(k,\mathtt{bob}) \ \& \ \text{allowed}(\mathtt{lookup},k)$
$\quad\& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},k).$

R4) $\text{authorize}(\{[u,\epsilon]\},\mathtt{lookup},k) \leftarrow \text{principal}(k,\mathtt{bob}) \ \& \ \text{memberof}(u,\mathtt{acme\text{-}people})$
$\quad\& \ \text{allowed}(\mathtt{lookup},k) \ \& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},k).$

R5) $\text{authorize}(\{[u,\epsilon]\},\mathtt{decrypt},k) \leftarrow,\& \ \text{principal}(k,\mathtt{bob}) \ \& \ \text{memberof}(u,\mathtt{acme\text{-}people})$
$\quad\& \ \text{subsequent}(k,\mathtt{0xA40B96C7}) \ \& \ \text{allowed}(\mathtt{decrypt},k) \ \& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},k).$

R6) $\text{authorize}(\{[u,\mathtt{manager}],[u',\mathtt{manager}]\},\mathtt{sign},\mathtt{0xA40B96D9}) \leftarrow u \neq u' \ \&$
$\quad\text{allowed}(\mathtt{sign},\mathtt{0xA40B96C7}) \ \& \ \text{owner}(\{[\mathtt{bob\_as\_director},\epsilon]\},\mathtt{0xA40B96C7}).$

R7) $\text{authorize}(\{[u,\mathtt{manager}],[u',\mathtt{manager}]\},\mathtt{sign},k) \leftarrow u \neq u'$
$\quad\& \ \text{memberof}(u,\mathtt{bob\text{-}friends}) \ \& \ \text{allowed}(\mathtt{sign},k)$
$\quad\& \ \text{owner}(\{[\mathtt{bob\_as\_director},\epsilon]\},k).$

R8) $\text{authorize}(\{[u,\epsilon]\},\mathtt{sign},k) \leftarrow \text{principal}(k,\mathtt{bob}) \ \& \ \text{principal}(\mathtt{0xA40B96D9},u)$
$\quad\& \ \text{current}(\mathtt{0xA40B96D9}) \ \& \ \text{allowed}(\mathtt{sign},k) \ \& \ \text{owner}(\{[\mathtt{bob},\epsilon]\},k).$

Fig. 4. Representation of the authorizations in Figure 2 at the service.

⟨subject,action,key⟩ be a request by subject to execute action on key. The following algorithm determines whether the request must be granted or denied.

```
If $P_{AS} \models$ authorize(subject, action, key) then grant request
else deny request.
```

Intuitively, the access control algorithm allows only those requests $r =$ ⟨subject,action,key⟩ for which authorize(r) can be derived from the logic program expressing the authorization state. As to how the logic program has been defined, the access control algorithm trivially enforces the access control policy stated in Section 4.2.

7. AUTHENTICATION OF SUBJECTS REQUESTING ACCESS

Actions requested by subjects on keys are granted or denied on the basis of the subject's possible ownership and authority privileges and on the authorizations applicable to the request. The identity of a subject determines the authorizations applicable to the subject and possible ownership or authority privileges the subject may have. The correctness of access control therefore rests on the correctness of the requesting subject's identity [Samarati and De Capitani di Vimercati 2001]. Hence, authentication must be performed prior to checking authorization.

7.1 Subject Authentication

We assume that the KMS can authenticate subjects without using any external certification authority. Typically, subjects can be authenticated on the basis of the keys registered at the service. In particular, a user can be authenticated by requiring that the digital signature on the request be verifiable by the public key registered for the user. Users in roles and conjuncted subjects can be authenticated on the basis of the keys of the users and of the roles in the conjunct

[Abadi et al. 1993]. For the escrow of the private key, the client's ability to complete the escrow successfully may be sufficient proof that the client knows the private key, and thus a digital signature for the escrow request may be unnecessary. The signature might still be required, however, if the validity checks on the escrowed key allow greater probability of a false positive than a digital signature algorithm (Reiter et al. [1996]).

Note also that requiring that the KMS independently perform authentication does not necessarily imply that subjects not registered at the service will not be allowed to execute actions. The use of variables and conditions in fact allows specifying generally applicable authorizations. For instance, all subjects not registered at the service can be considered special subjects (corresponding to user identifier `anonymous`) and authorizations applicable to them are determined accordingly. In particular, authorizations specified with a variable as subject and with subject condition equal to `true` are applicable.

By exploiting the keys associated with users and roles, the KMS is able to perform authentication independently. It is, however, worth noting that our access control can be nicely complemented with approaches where a subject's identity or active roles are established via certificates signed by external authorities (e.g., Park et al. [2001; Park and Sandhu [2000].)

7.2 First-Time Registration and Recovery

The assumption that the KMS authenticates subjects without the need for any external certification authority requires special consideration on the `register` and `revoke` action on keys.

Register is the action by which a principal (user/role) becomes known to the service by introducing its public key. Prior to this action, the service cannot authenticate the principal. First-time registration therefore requires special treatment, such as out-of-band communication between the principal to be registered and the KMS. Subsequent registrations, in which a principal essentially requires reregistration, that is, modification of its registered key, can instead be done by authenticating the principal (more precisely, the owner of the principal's key), depending on the key being modified. First-time registration can be done, for instance, with a manual procedure in which the identity of the user being registered is controlled on the basis of some proofs checked by the administrators of the service (similar to handling login assignments in a computer system). An alternative approach consists in some out-of-band negotiation between the principal and the administrator(s) of the service, resulting in a message digest of the principal's public key, which is then stored as the key of the principal. The principal, which can now be authenticated on the basis of the computed digest, can then register its real public key. This solution, adopted by Reiter et al. [1996], supports a registration scenario in which a principal generates its (potentially large) private key and public key in isolation. It then computes a short message digest of the public key and communicates this digest to the service's administrator(s) who authenticate the principal (through some external proofs of identity) and record the digest as the key of the principal.

A similar problem arises for recovery. The recover action is the one that allows subjects to retrieve the private key of a principal. Although the owner of a key can grant others the authorization to recover its key, we can imagine that such delegation will be rare (as any subject allowed to recover K_p^{-1} will then be able to impersonate p). For instance, it may be used in cases where different principals correspond to the same human user. To illustrate, suppose that Alice is registered at the service as two principals (corresponding to two different user identifiers for Alice): `alice_at_work` and `alice_at_home`. Since the person behind them is actually the same one, it may be completely legitimate for either to specify that the other is allowed to recover its key. Apart from this, however, recovery is primarily offered in order to give principals who lost their keys the possibility of retrieving them. In this case, since the private key is unavailable to the principal, the principal (or the subject corresponding to it) will not be able to digitally sign the request, and therefore the KMS will not be able to authenticate it. To solve this problem, we require a principal that has lost its key to undergo another registration. Given that the principal cannot be authenticated, this registration must be considered as a first-time registration and hence treated with a special procedure, as illustrated above. Since a principal can have at most one valid key, the registration will automatically revoke the old key. At this point the principal, who can be authenticated with the new key, can request recovery of the old key at the service. The ability of the principal to do so is guaranteed by the access control policy (Section 4).

8. GROUP MANAGEMENT AND ROLE ACTIVATION

In presenting our model we assumed that the sets of user, role, and group names are known at the key management service. We also considered that principals can be composed by users with some roles active, and we have introduced a predicate memberof to evaluate membership of users in groups. However, management of roles and groups is not required to be, and generally will not be, performed at the KMS. Although the KMS must know if a user has activated certain roles or s/he belongs to a group, it is not task of the KMS to control role activation or maintain group membership.

As for roles, we believe role activation and deactivation to be completely outside the key management service and carried out at the "domain" where roles are defined. The basic reason is that activation and deactivation of roles, like any action, must be regulated by policies and authorizations whose control should rest completely with those who defined the roles [Sandhu 1996; Winslett et al. 1997]. It would be improper and impractical to require that activation of roles be controlled at the KMS itself. Activation of roles by users is therefore carried out outside the KMS, and subjects can present themselves to the KMS as users in some roles. As already discussed, the KMS will authenticate these composite principals on the basis of the users keys and roles in them. Note that the fact that role activation is outside the key management service, together with the fact that roles are identified though their keys and that these keys should not be disclosed to users, requires the existence of a trusted process at the domain where the role is managed.[4] The task of this trusted process is to digitally sign requests by users in given role(s) with the key of the role(s) before submitting it to the KMS. For instance, consider organization `acme` where a role `acme-manager` has been defined which users can assume to perform some tasks. A principal acting for `acme` will register the key of the role at the KMS. Moreover, it will specify authorizations and policies regulating activation and deactivation of `acme-manager` at `acme`. (Note that this request will be signed with the key of `acme-manager`.) Every request by users in `acme-manager` will be digitally signed with the key of `acme-manager` before being sent to the KMS.

In a similar way, we assume group membership information will generally be managed locally at the domain where groups have been defined. This does not mean that no group information will be maintained at the KMS. If needed, some groups can be maintained at the KMS. As an example, a group `new-users` could be maintained at the KMS, grouping all users that have registered their keys for the first time in the current year. In general, however, groups will be defined and maintained outside the control of the KMS. The reason is that requiring groups to be maintained at the key management service is too impractical and cumbersome for both the entity managing the group (which would have to propagate group membership updates to the KMS) and the key management service (which would have to store and maintain group membership information). This assumption has been made in previous models dealing with groups in a decentralized distributed systems [Abadi et al. 1993; Lampson et al. 1992]. There are two different approaches that can be taken to collect membership information at the KMS. The first consists in requiring principals presenting requests to provide membership certificates. The second approach consists in having the KMS itself explicitly request group membership certificates for users from some external authorities. The model proposed in this article is completely independent of the approach used to provide membership information, and either of these two alternatives can be applied. We note that the drawback of assuming group management outside the KMS is that the KMS can evaluate membership or nonmembership of a user in a group only with respect to the information it has available at a specific time. Hence, the alternative adopted affects the enforcement of predicate memberof. As noted by Abadi et al. [1993], remote group management implies that only group membership can be proved with certainty; in contrast, it is not possible to prove nonmembership. In terms of the model, if memberof(u,g) evaluates true, it means that user u is a member of group g. The contrary is not necessarily true, and the fact that memberof(u,g) evaluates false does not necessarily imply that user u is not a member of group g. Membership information can simply not be provided or be unavailable. Note that this has an effect on the evaluation of conditional expressions in authorizations. A user will be allowed to use authorizations granted to members of some groups only if the

[4]An alternative approach consists in disclosing the key of a role to all users allowed to activate the role (knowledge of the key would in this case be equivalent to authorization to activate the role). This approach however, has the drawback of requiring changing the key of the role every time the authorization for a user has to be revoked [Lampson et al. 1992].

necessary membership certificates are presented to the KMS. Also, a user will be allowed to use authorizations granted to *nonmembers* (negated memberof literals) of some groups if there are no membership certificates proving his or her membership in these groups. Hence, negative memberof literals, although useful at times, must be used with care. Having noticed that evaluation of negative memberof literal suffers from this limitation, we do not forbid their use in the model. Note also that nonmembership can be proved in case of groups managed at the KMS or for which the KMS can explicitly require membership certificates to a trusted third party [Winslett et al. 1997].

A possible alternative approach to manage groups is to associate keys with groups [Lampson et al. 1992]. However, all members of a group will in this case be able to obtain the private key of the group. An important drawback of this approach is that removing a user from a group requires modification of the key associated with the group. Moreover in this approach, when presenting a request users should explicitly state the groups that must be considered in evaluating a given request. This last aspect, while proper for roles, may be limiting for groups.

9. RELATED WORK

To our knowledge, no authorization model has been proposed to date for regulating access to keys at a key management system, and this is the first article to address the problem. Hence, related work has to be found either in the context of PKI management or in the context of general authorizations and access control models. Most of the work in key management has focused on the problems of key storage and escrow [ACM 1996], on binding identities to keys, and on trust management, possibly in the absence of certification authorities (e.g., Maurer [1996]; Ellison [200]). Also, attention has mostly been focused on certificate management rather that key management. All these approaches are complementary to our proposal.

Within the trust management context, Blaze et al. [1997] present an approach to formulate security policies. With respect to policy (authorization) specification, the proposal by Blaze et al. [1997] takes the approach of specifying authorizations with respect to keys (as subjects), rather than identities. The motivation behind this decision is that with this approach the application does not then need to manage the mapping between personal identities and authorities, thus also allowing support for anonymity. As we have already discussed in Section 5, the approach may be limiting, in that it does not allow the enforcement of expressive and flexible access control policies that take group membership and roles into account. Second, unlike identities, keys are subject to modifications that would result in the automatic invalidation of all the key's privileges and in the corresponding principal being unable to exercise them any longer. Also, the motivations of anonymity and of avoiding maintenance of identity mappings do not seem to be relevant or applicable in our context, where the KMS system stores the keys and is assumed to enforce authentication.

Other related work is in authorization-based access control models where recent proposals have been devoted to enhancing the expressiveness and flexibility of authorizations [Jajodia et al. 2001; Li et al. 1999; Woo and Lam 1993]. However, these approaches have not been examined for key management services, and they do not address the peculiarities of such a context. The focus of these proposals is mainly the combined support of positive and negative authorizations together with authorization derivation/implication to provide multiple policies and support for exceptions. To this purpose, various logic-based languages are investigated which, in some cases, almost provide the expressive power of first-order logic [Woo and Lam 1993; Li et al. 1999]. But these languages bear a computational complexity, and consequent loss of efficiency, which is often not balanced by a gain in expressiveness. The expressiveness of such languages in some cases allows the specification of models which may not even be decidable [Woo and Lam 1993], and therefore implementable. Even in the case of more restricted and therefore efficiently computable models, the focus remains on the combination of positive and negative authorizations coupled with the derivation of authorizations along different subject or object hierarchies [Jajodia et al. 2001]. These aspects however, appear, to be of little applicability in our context, where objects are keys and group and role management is carried out remotely. So the expressiveness of such models would be of little applicability. And such models do not provide support for conditional predicates based on key-subject relationships, which are needed in our context. They also provide very limited or no support for roles and conjuncted subjects.

Explicit support for users in roles and conjuncted subjects is provided by Abadi et al. [1993] in their proposal for a calculus for distributed access control. However, the concept of roles in Abadi et al. [1993] and in our model is different, and consequently requires different, reasoning about authorizations. In particular, in Abadi et al. [1993], roles are means for users to restrict their set of privileges for particular executions. Consequently, the authorizations specified for a user in a role are applicable to the user when operating as an individual; by contrast, authorizations specified for a user are not applicable when the user operates in certain roles. Moreover, in Abadi et al. [1993], all authorizations are ground, and it is not possible to specify authorizations applicable to different objects (keys) or subjects that depend on certain conditions.

10. CONCLUSIONS

We investigated the problem of regulating access to keys stored at a key management service. We proposed a basic policy based on principal, ownership, and authority relationships on keys, together with an authorization language by which owners can grant and revoke privileges to execute different actions on their keys. The language is flexible, and gives owners the ability to specify, while granting an authorization, whether an authorization applies to a specific key or to the key registered for a principal (regardless of its digest), and whether the privileges should still hold upon expiration/revocation of the key. The authorization language, although very simple, is sufficiently expressive to enable the specification of authorizations for composite and conjuncted subjects that can be fully specified (grounded) or partially specified, thus making the authorizations applicable to all subjects satisfying certain conditions. We have also illustrated how the access control policy and authorizations can easily be expressed through a simple restricted form of logic language. Finally, we discussed the problems of first-time registration and of group management.

Future issues that can be investigated include the extension of the model to keys other than user keys; the possible inclusion in the authorization of time intervals or expiration dates, upon which authorizations are automatically revoked; the interaction with other certification authorities and support of credential-based authorizations; and extending the idea to enable various parties to collaborate in the PKI, with consequent support for expressing and reasoning about trust and recommendations.

REFERENCES

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst. 15*, 4 (Sept.), 706–734.

ACM. 1996. Special section: How to use key escrow. *Commun. ACM 39*, 3, 32–60.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1997. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (Oakland, CA, May), 164–173.

ELLISON, C. 2000. SPKI certificate documentation. http://www.pobox.com/cme/html/spki.html.

GANESAN, R. 1995. Yaksha: Augmenting Kerberos with public key cryptography. In *Proceedings of the 1995 Internet Society Symposium on Network and Distributed System Security* (Feb.), 132–143.

JAJODIA, S., SAMARATI, P., SAPINO, M. L., AND SUBRAHMANIAN, V. S. 1992. Flexible support for multiple access control policies. *ACM Trans. Database Syst. 26*, 2 (June), 214–260.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst. 10*, 4 (Nov.), 265–310.

LI, N., FEIGENBAUM, J., AND GROSOF, B. 1999. A logic-based knowledge representation for authorization with delegation. In *Proceedings of the 12th Computer Security Foundations Workshop* (Mordano, Italy, June), 162–174.

MAURER, U. 1996. Modeling a public key infrastructure. In *Proceedings of the Fourth European Symposium on Research in Security and Privacy*, LNCS 1146 (Rome, Sept.), 325–350.

MCMAHON, P. 1995. SESAME V2 public key and authorisation extensions to Kerberos. In *Proceedings of the 1995 Internet Society Symposium on Network and Distributed System Security* (Feb.), 114–131.

MICALI, S. 1992. Fair public-key cryptosystems. In *Advances in Cryptology—Proceedings of CRYPTO '92*, E. Brickell Ed., LNCS 740, 113–138. Springer-Verlag.

PARK, J. S., SANDHU, R., AND AHN, G.-J. 2001. Role-based access control on the Web. *ACM Trans. Inf. Syst. Secur. 4*, 1 (Feb.).

PARK, J. AND SANDHU, R. 2000. Binding identities and attributes using digitally signed certificates. In *Proceedings of the 16th Annual Computer Security Applications Conference* (New Orleans, LA, Dec.), 120–127.

REITER, M. K., FRANKLIN, M. K., LACY, J. B., AND WRIGHT, R. N. 1996. The Ω key management service. *J. Comput. Secur. 4*, 4, 267–287.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proc. IEEE 63*, 9 (Sept.), 1278–1308.

SAMARATI, P. AND DE CAPITANI DI VIMERCATI, S. 2001. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design*. R. Focardi and R. Gorrieri Eds., LNCS 2171. Springer-Verlag.

SAMARATI, P. AND JAJODIA, S. 1999. Data security. In *Wiley Encyclopedia of Electrical and Electronics Engineering*. J. Webster Ed., John Wiley.

SANDHU, R., COYNE, E., FEINSTEIN, H., YOUMAN, C. 1996. Role-based access control models. *IEEE Computer 29*, 2 (Feb.), 38–47.

SANDHU, R. AND SAMARATI, P. 1997. Authentication, access control, and intrusion detection. In *The Computer Science and Engineering Handbook*. A. Tucker Ed., CRC Press.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, R., and Chandramouli, R. 2001. Proposed NIST standard for role-based access control *ACM Trans. Inf. Syst. Secur. 4*, 3 (Aug.).

Winslett, M., Ching, N., Jones, V., and Slepchin, I. 1997. Using digital credentials on the World Wide Web. *J. Comput. Secur. 5*, 3, 255–267.

Woo, T. Y. C. and Lam, S. S. 1993. Authorizations in distributed systems: A new approach. *J. Comput. Secur. 2* (2,3), 107–136.

Received February 2000; revised October 2001; accepted October 2001