# An Authorization Model for a Public Key Management Service

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

Public key management has received considerable attention by 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 paper we propose such a framework for a key management service supporting 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 different 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: Public key infrastructure, Authorizations specification and enforcement, Access control

---

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

## 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 extensively explored (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 would seem to be quite simple: as their names indicate, a public key should be accessible to anyone, and an escrowed private key should be accessible only to its owner. This simple model is far too simplistic, however, for many realistic settings.

—One purpose of 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 rather available to only a selected few for the purpose of verifying signatures on private messages. In this way, if a message is accidentally leaked from an organization, then 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 value 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 or not be considered still valid.

In this paper we present an authorization model for controlling 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 by allowing specification of, and reasoning about, joint authority, delegation, and roles. It also addresses how to deal with 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 for establishing keys and managing them. And, while previous proposals have focused mostly on certification authority and certificate management, our proposal specifically focuses on keys and related services.

The contributions of this paper 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 for enforcing the proposed access control policies.

The generic key management service that forms the framework for this paper was inspired by the $\Omega$ service [Reiter et al. 1996], and our work complements $\Omega$ nicely in that authorization and access control in $\Omega$ was left largely undeveloped [Reiter et al. 1996, Section 6]. We note that, while for concreteness this paper 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, consistently with $\Omega$, in this paper 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 paper 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 discusses 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 paper we consider a KMS that closely mimics the $\Omega$ system [Reiter et al. 1996] in functionality. In particular, $\Omega$ 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,

—*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

| 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. |

Table 1. Actions on keys and their semantics

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

To all these actions, which are summarized in Table 1, 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 when 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 rather 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 will be discussed in Section 4.2, revoked keys are available for restricted use, that is, only a subset of the actions in Table 1 can be exercised on old keys (for instance, a revoked key cannot be used to sign messages).

## 3. BASIC CONCEPTS OF THE AUTHORIZATION MODELS

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 another type of principals, namely those submitting requests on the stored keys and for which therefore 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** These are entities that can directly make requests 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** Roles are *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** Groups are *named* collections of users.

Note the distinctions we are making 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 grouping of people while roles define grouping 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 of 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 of specifying a different authorization for each employee. Also, the specified authorization will be applicable only to users for which there is a proof of the membership of the user in group `acme-employees`. Hence, if the membership of a user in the group ceases, the authorization granted to the group will automatically become not applicable to the user anymore.

The consideration of roles allows the specification of authorizations that apply to users when performing some 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 activities concerning the project management 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 is authorized to do so) and may be subject to some restrictions. For instance, some roles may be considered incompatible with each other 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, while privileges granted to groups add to those of the users (i.e., by being member of a group a user can execute more actions that she could as an individual)[1] when playing a role a user may not be able to execute the actions for which she 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 this role is active for the user has a double advantage. First, it allows the use of the privileges needed to perform a task only within the task execution (not granting them indiscriminately as it would instead be if authorizations were specified for users or groups). Second, it allows the enforcement of the *least privilege principle* according to which each role (user assuming it) is confined to the execution of only those actions needed to perform the task.

Note that for the different semantics groups and roles carry, group and role identifiers are required to be disjoint. However, the same concept can be interpreted under the two different views, 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 job. This situation is completely proper and must be considered neither ambiguous nor 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 $\mathsf{U}$, $\mathsf{R}$, and $\mathsf{G}$ 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 comprise of any of the following:

—a user $u \in \mathsf{U}$;

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

—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 introduced in Section 2 as entities associated with keys stored at the service, 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 will be discussed in Section 4. For the time being, for the sake of 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 to represent cases in which 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* $\mathsf{U}$ *and* $\mathsf{R}$ *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* $\epsilon$ *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. Non-conjuncted 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,$\epsilon$]}
$p_2$: {[alice,acme-manager]}
$p_3$: {[bob,acme-administrator]}
$p_4$: {[alice,$\epsilon$],[bob,$\epsilon$],[chris,$\epsilon$]}
$p_5$: {[alice,acme-manager],[bob,acme-administrator]}
$p_6$: {[bob,$\epsilon$],[alice,$\epsilon$],[chris,$\epsilon$]}
$p_7$: {[alice,acme-manager],[alice,acme-administrator]}
$p_8$: {[alice,acme-manager],[bob,acme-manager]}

The order of the 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 the set notation, subject equality therefore reduces to set equality.

For simplicity, Definition 3.1 restricts subjects to be of the form "single user in single role" or a conjunction of them. This restriction is purely syntactical and therefore our definition encompasses subjects where a given user activates more roles or different users activate a 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 the sake of simplicity, in the following we will omit the set notation in the case of singleton sets (i.e., non-conjuncted subjects), and the square brackets in case of users (i.e., pairs with role element null), when such a simplification does not bring any confusion. For instance, subjects $p_1$ and $p_2$ may be denoted as `bob` and `[alice,acme-manager]`, respectively.

Note that groups, which have been discussed in the previous subsection as one of the basic elements of the authorization model, do not appear in the definition of subject. The reason for this 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 distinction that we made about the fact that group membership (the privileges that 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 paper, we characterize access requests as triples of the form ⟨`subject`, `action`, `key`⟩ stating that `subject` requests to execute `action` on `key`. Here, `subject` is any subject allowed by Definition 3.1; `action` is any action listed in Table 1; 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 since 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 will be addressed in Section 7.2. Also, for the time being we ignore the problem of authenticating subjects presenting requests. This issue will be 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 grant other subjects the privilege of executing actions on the key. Section 4 discusses the basic privileges that the KMS must warrant to principals storing their keys at the service. Section 5 discusses the 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 actions on them, we must first clarify the relationship between principals (cf. Section 2) and subjects (cf. Section 3.2).

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

Beside principals, our model supports two additional concepts, namely *ownership* and *authority*, that tie subjects to keys. These concepts are discussed in the next subsection.

### 4.1 Key's principal, ownership and authority

The following three concepts describe the relationships that can relate subjects with 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 be either a user or a role. As we will 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 one or more owners. The owner of a key can exercise any actions on the key as well as grant this ability to other 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 to be 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

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

key. However, we can expect that usually 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 conjunct) subject, however 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 conjunct) subject. Like with ownership, more than one subject may have authority over a key. A subject having the authority over a key can always execute `lookup`, `revoke`, and `decrypt` actions on the key. Intuitively, a subject who has authority over a key can control how the key is used and can revoke the key. The authority cannot however "impersonate" the principal (it 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 for which the authority and the ownership of a key can rest 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. For the purpose the keys are to be used, the organization may wish to maintain the 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 her key and as such granted all the accesses that this privilege implies. However, at any time `acme` will be able to `lookup` the key, to `decrypt` messages encrypted with the key, and to `revoke` the key. The organization can use this last privilege, for instance, if an employee leaves the job. Authority can also be different 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 having 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 conjuncted subject {`jim,tim`} can exercise authority, while neither `jim` nor `tim` individually can.

The specification of the subject/s having ownership and authority over a key is made at the time of the registration of 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 having ownership (authority resp.) on a key being revoked acquires ownership (authority resp.) on the new key. This aspect is particularly important, for instance, in the case of organization's controlled keys, like the ones discussed above: the organization should maintain authority over an employee's key despite modifications to the key by the employee. To this purpose, at the KMS, ownership and authority information are specified with reference to principals (users and roles), rather than with reference to their specific keys. We will 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 on the key all the actions that this privilege implies. Besides these basic accesses, other accesses may need to be provided on the keys stored at the service. As a matter of fact, one of the purposes of having a key management service is to allow owners to make their keys available to others. The owner of a key must therefore be given the ability of granting privileges on her key to others. Consequently, the KMS must grant, beside the accesses implied by the ownership and authority privileges, also all the accesses that owners wish to be granted. At the same time, to correctly respond to the trust of the owners that store their keys at the service, the KMS must ensure that no other accesses will be permitted. Also, it must guarantee that only operations that can be executed will be granted (for instance, no `sign` action can be executed on revoked keys).

We therefore 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 permit, however, only a subset of such actions on revoked keys. Revoked keys can still be available for lookup, decrypt, and recovery services. However, not being valid anymore, the system will not permit their use for `sign`ing messages. Also, they cannot be `revoke`d, re-`register`ed, or `escrow`ed.

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 to exercise any actions on its keys.*
—*revoke the privilege to exercise 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 the following conditions hold:*

*—the action is allowed (Definition 4.1);*

*—the subject can exercise the action because of ownership, authority, or in virtue of an authorization granted by a key's owner (Definition 4.2).*

*All other accesses which do not satisfy any of 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 to (and revoke from) others authorizations to exercise actions on the key. We note that delegation policies could be considered by which the owner can grant, together with the authorization to execute an action, the ability to grant such privilege to others. The reason for not considering delegation of administration in our model is twofold. First, for the specific objects (keys) and actions considered, delegation of administration does not seem to be needed: we can imagine that the owner of the key wish to retain control of whomever can access and use the key. Second, as we will see shortly, our language allows the specification of authorizations supporting variables and conditions, resulting in expressiveness and flexibility in terms of specifications that can be stated. However, the coupling of delegation management with such expressiveness would introduce complications. In particular, the revoke operations, by itself not trivial in the case of chains of delegation [Samarati and De Capitani di Vimercati 2001], would be further complicated as the revocation of the authorization to administer an access may correspond to the partial invalidation of an authorization granted through it instead of a complete revocation. This would require 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 to their keys to others. In Section 6 we will illustrate how the authorizations so specified, together with the ownership and authority information, are used in the access control enforcement.

## 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`⟩ (Section 3.3) should be allowed. This framework, although used effectively in systems where subjects are not conjuncted and objects are information containers, is limiting in our context 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 be considered still valid when the key is modified? Second, the consideration of composite and conjuncted subjects introduces 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 [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's 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 with reference to 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 too 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 re-register operations.

Regardless of whether the authorizations on a key propagate or not to the new key registered when the key is revoked, 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, for instance, to verify a principal's signature on an old message signed with a key that has then been revoked). Again, no answer is good for all scenarios, and while in some cases the 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's subjects

Authorization's 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's subjects with their keys, thus associating permissions with keys rather than with user identifiers [Blaze et al. 1997; Ellison 2000]. This solution has two main drawbacks. First, granting of privileges to keys does not allow the enforcement of expressive and flexible access control policies taking into consideration group membership and roles. Second, unlike user identifiers, keys are subject to modifications, which would result in the automatic invalidation of all the key's privileges and in the corresponding principal not able to exercise them anymore.

As it should be clear from Section 3, our model assumes the granting of authorizations to subjects with reference to the subjects' identifiers. This way a subject's authorizations will 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 able also to support authorizations to exercise actions which are granted to "a specific key", or better to the principal corresponding to that key.

The consideration of roles and conjuncted subjects introduces several questions regarding the interpretation of authorizations. Questions to be addressed 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 considered valid also for `[alice,acme-manager]`?
(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 also for `alice`?
(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 different ways of answering the questions above should 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 $\mathsf{U}$ is the set of users, $\mathsf{R}$ is the set of roles, $\mathsf{G}$ is the set of groups, $\mathsf{K}$ is the set of key identifiers (digests), $\mathsf{A}$ is the set of actions, and $\mathsf{S}$ is the set of all possible subjects that can be obtained from $\mathsf{U}$ and $\mathsf{R}$.
(2) **Variable symbols:** There are five sets of variable symbols $V_u, V_r, V_g, V_k, V_s$ ranging over the sets $\mathsf{U}, \mathsf{R}, \mathsf{G}, \mathsf{K}$, and $\mathsf{S}$ respectively.
In future, the following terminology will be used. Variables in $V_u$ and members of $\mathsf{U}$ are user terms. Variables in $V_r$, and members of $\mathsf{R}$ 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:** The following predicate symbols are considered.

(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 it 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 non-ground 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`$, \epsilon], [s,$ `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 $\cup$ R *and* $k \in$ K *iff key* $k$ *was registered for* $p$*;*

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

—$\mathsf{subsequent}(k, k')$ *is satisfied for* $k, k' \in \mathsf{K}$ *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 non-ground 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 specify authorizations referred 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* $\mathsf{principal}(p, k)$, $\mathsf{current}(k)$, $\mathsf{subsequent}(k, k')$, $\mathsf{subsequent}(k', k)$, *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* $\mathsf{memberof}, \mathsf{principal}, \mathsf{current}, \mathsf{subsequent}, =, \in, \subseteq$.

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* $\Theta$ *of values to variables in* $E$ *such that the resulting expression* $E\Theta$ *is satisfied.*

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

Example 5.2. *Some examples of key/subject expressions are as follows.*

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

—**Key term:** `0xA40B96C7`. **Expression:** $\mathsf{current}(\texttt{0xA40B96C7})$.
*Satisfied by key* `0xA40B96C7` *when not revoked.*

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

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

—**Subject term:** $\{[u, \epsilon]\}$. **Expression:** $\mathsf{principal}(u, \texttt{0xA40B96C7})\ \&\ \mathsf{current}(\texttt{0xA40B96C7})$.
*Satisfied by the user who is the principal of valid (i.e., unrevoked) key* `0xA40B96C7`.

—**Subject term:** $s$. **Expression:** $\{[u, \epsilon]\} \subseteq s\ \&\ \mathsf{memberof}(u, \texttt{acme-employee})\ \&\ \mathsf{memberof}(u, \texttt{US-citizens})$.
*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.*

| | | |
|---|---|---|
| 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

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 key's owners to specify authorizations. Authorizations, which can be specified and revoked through a simple language like that illustrated 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.*

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 already 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

[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.

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

Fig. 2. Some examples of authorizations

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, time but must be enforced at access control time. Intuitively, from the point of view of the semantics of authorizations, this constraint can be easily enforced 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 will discuss this 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* $\Theta_k$ *of ground terms for variables such that* `key`$\Theta_k$=`key′` *and* `k-cond`$\Theta_k$ *is satisfied.*

—*There exists a substitution* $\Theta_s$ *for variables in* $s$ *such that* `subject`$\Theta_s$=`subject′` *and* `s-cond`$\Theta_s$ *is satisfied.*

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 not apply anymore to the key that has been revoked.

**A4)** Users who are member of group `acme-people` can lookup the key registered for principal `bob`. The authorizations applies also 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 the past keys (so that if the key is revoked subjects can still exercise the privilege 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 retro-active effect of the authorization: the authorization does not hold on keys which 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 re-registered). 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 illustrated 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 the 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 already discussed (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.

—**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):
$\mathsf{owner}(s_i, k) \leftarrow \mathsf{principal}(\mathtt{princ}, k) \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$.
$\mathsf{authority}(s_j, k) \leftarrow \mathsf{principal}(\mathtt{princ}, k) \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$.

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

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

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

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

(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 as established by Definition 4.1, that is, $\mathsf{allowed}(a, k)$ 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 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:*

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

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

DEFINITION 6.2 (AUTHORITY RULE). *An* authority rule *is a rule of the form:*

$$\mathsf{authority}(s,k) \leftarrow \mathsf{principal}(p,k)\,\&\; L_1\,\&\; \ldots\,\&\; L_n$$

*where s is a subject term, p is a* ground *principal term, k is a variable term, and* $L_1,\ldots,L_n$ *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 simply the principal-literal on the right hand side, giving, for example, each user the ownership/authority on the key registered for her. 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)** $\mathsf{owner}(\{[\texttt{bob},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{bob},k)$.
**R2)** $\mathsf{authority}(\{[\texttt{bob},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{bob},k)$.
**R3)** $\mathsf{owner}(\{[\texttt{bob-at-work},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{bob-at-work},k)$.
**R4)** $\mathsf{authority}(\{[\texttt{acme},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{bob-at-work},k)$.
**R5)** $\mathsf{owner}(\{[u,\texttt{acme-manager}],[u',\texttt{acme-manager}]\},k) \leftarrow \mathsf{principal}(\texttt{acme-manager},k)$, $u \neq u'$.
**R6)** $\mathsf{authority}(\{[\texttt{acme-headoffice},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{acme-manager},k)$.
**R7)** $\mathsf{authority}(\{[\texttt{acme-admin},\epsilon]\},k) \leftarrow \mathsf{principal}(\texttt{acme-manager},k)$.

*According to these rules:*

—`bob` *has ownership and authority on the key for which he is principal (rules* **R1** *and* **R2***).*
—`bob-at-work` *is the owner of the key of which it is principal, but* `acme` *has authority over such a 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 they can be specified, modified, deleted, or not obeyed 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 (authority resp.) on a given key correspond to shared ownership (authority resp.).

DEFINITION 6.3 (AUTHORIZATION RULE). *An* authorization rule *is a rule of the form:*

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

*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

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

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

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

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

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

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

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

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

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

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* $\langle\mathtt{subject},\mathtt{action},\mathtt{key}\rangle$ *be a request by* `subject` *to execute* `action` *on* `key`*. The following algorithm determines whether the request must be granted or denied.*
**If** $P_{AS} \models \mathsf{authorize}(\mathtt{subject},\mathtt{action},\mathtt{key})$ **then** *grant request*
**else** *deny request.*

Intuitively, the access control algorithm allows all and only those requests $r = \langle\mathtt{subject},\mathtt{action},\mathtt{key}\rangle$ for which $\mathsf{authorize}(r)$ can be derived from the logic program expressing the authorization state. For 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 possible ownership and authority privileges of the subject and on the authorizations applicable to it. 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 the access control therefore rests on the correctness of the requesting subject's identity [Samarati and De Capitani di Vimercati 2001]. Authentication must then be performed prior to authorization checking.

### 7.1 Subject's authentication

We can assume that the KMS has the ability to perform authentication of 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 conjunct subjects can be authenticated on the basis of the keys of the users and roles appearing in them [Abadi et al. 1993]. For the escrow of the private key, the client's ability to successfully complete the escrow 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 (like, for instance, in [Reiter et al. 1996]).

Note also that requiring that the KMS be able to 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 allows in fact to specify authorizations that are generally applicable. For instance, all subjects not registered at the service can be considered as a special subject (corresponding to user identifier `anonymous`) and authorizations applicable to them determined accordingly. In particular, authorizations specified with a variable as subject and with subject condition equal to `true` will be applicable.

By exploiting the keys associated with users and roles, the KMS is therefore able to perform authentication independently. It is, however, worth noting that our access control can be complemented nicely with approaches where subject's identities 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 need of any external certification authority requires special consideration of two actions on keys: `register` and `revoke`.

The register action is the action through 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 requires essentially to re-register, that is, modify its registered key, can instead be performed by authenticating the principal (more precisely the owner of the principal's key) on the basis of the key being modified. First time registration can be performed for instance through the use of 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 (similarly to what done for handling login assignment in a computer system). An alternative approach consists in performing some out-of-band negotiation between the principal and the administrator/s of the service that results 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 in [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 that 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 action that allows one 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 such a delegation will be rare (as any subject allowed to recover $K_p^{-1}$ will be then 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, it may be completely legitimate, for any of them 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 key the possibility of retrieving it. In this case, since the private key is unavailable to the principal, the principal (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 therefore be 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 to recover 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 have assumed that the sets of user, role, and group names are known at the key management service. We have also considered that principals can be composed by users in some roles 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 consider 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 for this is that activation and deactivation of roles, like any action, must be regulated by policies and authorizations, whose control should rest completely with whom defined the role [Sandhu 1996; Winslett et al. 1997]. It would be improper and impractical to require activation of roles to 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 key of the users 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, and 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` sent to the KMS will be digitally signed with the key of `acme-manager` before being sent to `acme`.

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 key 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 for this is that requiring groups to be maintained at the key management service would be 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 approach consists in requiring principals presenting requests to provide membership certificates. The other approach consists in having the KMS itself explicitly request from some external authorities the group membership certificates for users. The model proposed in this paper is completely independent from the approach used to provide membership information and any of these two alternatives can be applied. We note that the drawback of assuming group management

[4]In alternative users authorized to operate in some roles can be given the key of the role (knowledge of the key would in this case be equivalent to the authorization to activate the role). This approach has however the drawback of requiring to change the key of the role every time the authorization for a user has to be revoked [Lampson et al. 1992].

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. The alternative adopted therefore affects the enforcement of predicate memberof. As noticed in [Abadi et al. 1993], remote group management implies that only group membership can be certainly proved; in contrast it is not possible to prove nonmembership. In terms of the model, if memberof(u,g) evaluates true then it means that user u is a member of group g. The vice versa 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 in fact simply be not provided or not be available. Note that this has 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 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 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. A main drawback of this approach is that removal of a user from a group would require a modification of the key associated with the group. Moreover in this approach users should explicitly state, when presenting a request, the groups that must be considered in the evaluation of 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 todate for regulating access to keys at a Key Management System and this paper is the first paper addressing the problem. Related work has therefore 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 the key management field has focused on the problems of key storage and escrow [ACM 1996], on the problems of binding identities to keys and on trust management possibly in the absence of certification authorities (e.g., [Maurer 1996; Ellison 2000]). Also, the attention has mostly been focused on certificate management rather that on key management. All these approaches are complementary to our proposal.

Within the different context of trust management, Blaze et al. [Blaze et al. 1997] present an approach to formulate security policies. With respect to the aspect of policy (authorization) specification, the proposal in [Blaze et al. 1997] takes the approach of specifying authorizations with respect to keys (as subjects), rather than identities. The motivation behind such a decision is that this way the application does not need to manage the mapping between personal identities and authorities, thus allowing also 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 taking into consideration group membership and roles. Second, unlike identities, keys are subject to modifications, which would result in the automatic invalidation of all the key's privileges and in the corresponding principal not able to exercise them anymore. Also, the motivations of anonymity and of avoiding maintenance of identity mappings do not seem to be relevant or applicable in the context considered by us, where the KMS system stores the keys and is assumed to enforce authentication.

Other related work is in the area of authorization-based access control models, were recent proposals have been devoted to enhancing expressiveness and flexibility of the 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 peculiarities of such a context. Also, 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 different logic-based languages are investigated which, in some cases provide almost the expressive power of first order logic [Woo and Lam 1993; Li et al. 1999], thus bearing a computational complexity and consequent loss of efficiency often unbalanced with respect to the gain in expressiveness. The expressiveness of such languages allow in some cases 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 appear, however, of little applicability in our context, where objects are keys and group and role management is carried out remotely. Therefore, the expressiveness of such models would be of little applicability. On the other hand such models do not provide instead, for instance, support of conditional predicates based on key-subject relationships which are needed in our context. They also provide no, or very limited, support for roles and conjunct subjects.

Explicit support for users in roles and conjunct subjects is provided by Abadi et al. in [Abadi et al. 1993] in their proposal of 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 on the 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 some roles. Moreover, in [Abadi et al. 1993], all authorizations are ground and it is not possible to specify authorizations applicable to different objects (keys) in our cases, or subjects depending on some conditions.

## 10. CONCLUSIONS

We have investigated the problem of regulating access to keys stored at a key management service. We have 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 of specifying, while granting an authorization, whether the 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 some 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 have also discussed the problems of first time registration and of group management.

Future issues that can be investigated include: the extension of the model to the consideration of keys other than the 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; the extension to the consideration of different parties collaborating in the PKI with consequent support for expressing and reasoning on trust and recommendations.

## REFERENCES

Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. 1993. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems 15*, 4 (September), 706–734.

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

Blaze, M., Feigenbaum, J., and Lacy, J. 1997. Decentralized trust management. In *Proc. of the 1996 IEEE Symposium on Security and Privacy* (Oakland, CA, May 1997), pp. 164–173.

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

Jajodia, S. Samarati, P., Sapino, M.L., and Subrahmanian, V.S. 1992. Flexible Support for Multiple Access Control Policies. *ACM Transactions on Database Systems 26*, 2 (June), 214–260.

Lampson, B., Abadi, M., Burrows, M., and Wobber, E. 1992. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems 10*, 4 (November), 265–310.

Li, N., Feigenbaum, J., and Grosof, B. 1999. A logic-based knowledge representation for authorization with delegation. In *Proc. of the 12th Computer Security Foundations Workshop* (Mordano, Italy, June 1999), pp. 162–174.

Maurer, U. 1996. Modeling a public key infrastructure. In *Proc. of the Fourth European Symposium on Research in Security and Privacy*, LNCS 1146 (Rome, Italy, September 1996), pp. 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* (February 1995), pp. 114–131.

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

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

PARK, J.S., SANDHU, R., AND AHN, G-J. 2001. Role-based access control on the Web. *ACM Transactions on Information and System Security 4*, 1 (February).

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, Louisiana, December 2000), pp. 120–127.

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

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

SAMARATI, P. AND DE CAPITANI DI VIMERCATI, S. 2001. Access Control: Policies, Models, and Mechanisms. In R. FOCARDI AND R. GORRIERI Eds., *Foundations of Security Analysis and Design*. LNCS 2172. Springer-Verlag.

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

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

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

FERRAIOLO, D., SANDHU, R., GAVRILA, S., KUHN, R. AND CHANDRAMOULI, R. 2001. Proposed NIST Standard for Role-Based Access Control *ACM Transactions on Information and Systems Security 4*, 3 (August).

WINSLETT, M., CHING, N., JONES, V., AND SLEPCHIN, I. 1997. Using digital credentials on the World Wide Web. *Journal of Computer Security 5*, 3, 255–267.

WOO, T. Y. C. AND LAM, S. S. 1993. Authorizations in distributed systems: A new approach. *Journal of Computer Security 2*, (2,3), 107–136.