HIERARCHY-BASED ACCESS CONTROL IN DISTRIBUTED ENVIRONMENTS

Jean-Camille Birget[△], *Xukai Zou*[†], *Guevara Noubir*[⊕] *and Byrav Ramamurthy*[†]

[△]Dept. of Computer Science, Rutgers University, Camden, USA
[†]Dept.of Computer Science and Engineering, University of Nebraska-Lincoln, USA
[⊕]College of Computer Science, Northeastern University, USA
E-mail: birget@camden.rutgers.edu, {xkzou, byrav}@cse.unl.edu, noubir@ccs.neu.edu

Abstract. Access control is a fundamental concern in any system that manages resources, e.g., operating systems, file systems, databases and communications systems. The problem we address is how to specify, enforce, and implement access control in distributed environments. This problem occurs in many applications such as management of distributed project resources, e-newspaper and payTV subscription services.

Starting from an access relation between users and resources, we derive a user hierarchy, a resource hierarchy, and a unified hierarchy. The unified hierarchy is then used to specify the access relation in a way that is compact and that allows efficient queries. It is also used in cryptographic schemes that enforce the access relation. We introduce three specific cryptography based hierarchical schemes, which can effectively enforce and implement access control and are designed for distributed environments because they do not need the presence of a central authority (except perhaps for set-up).

Keywords: Distributed access control, access hierarchies, information and communication security.

Note: This work was conducted when G. Noubir and J.-C. Birget were visiting the University of Nebraska-Lincoln. J.-C. Birget and B. Ramamurthy were supported in part by NSF grants.

1. INTRODUCTION

The domain we consider in this paper is that of distributed applications in environments such as distributed operating systems, distributed database systems, and communication networks where different users access different resources with different access rights. This problem is called distributed access control. Typical examples include access to rooms (e.g., class and lab) in a building, management of project resources, e-newspaper and payTV subscription services.

For example, in the context of management of project resource, users include directors, group leaders, project managers, technical managers, engineers, consultants, administrative staff, customers and accounting staff. Resources include financial data, internal technical documents, public project documents, laboratories, etc. Different users have different access rights to different resources, which need to be concisely specified and correctly enforced.

Access control deals with the specification and enforcement of users' access permissions (and access restrictions) relative to the resources of a system. This is a fundamental concern in any system that manages resources, e.g., operating systems, file systems, databases and communications systems. Traditionally, access control is specified by an *access relation* (or "access matrix") that lists explicitly which users can access which resources.

In this paper we uncover a user hierarchy and a resource hierarchy, that are implicit in any access relation. Intuitively the hierarchies arise from the fact that some users have more access rights than others, and some resources carry more access constraints than others (a formal definition will be given later). We show that these hierarchies can give useful information.

Another contribution of this paper is an algorithm that merges these implicit user and resource hierarchies into a single hierarchy. This unified hierarchy contains the user and resource hierarchy as sub-hierarchies; moreover, a user is above a resource in the unified hierarchy if and only if this user has access to this resource. Thus the unified hierarchy contains all the information of the access relation, while also displaying the useful hierarchy information. In addition, the unified hierarchy merges 'equivalent' users, and merges 'equivalent' resources (rigorous definitions will be given); thus the unified hierarchy will usually be a compact description of the access rights.

Having a unified hierarchy can simplify access control. The literature contains secure access control protocols [1, 3, 6, 8, 10] that assume (without justification) that we have such a pre-existing unified hierarchy (see Subsection 3.4). We show how various secure access control schemes make use of hierarchy information in order to enforce access permissions and restrictions.

For a centralized system, access control is usually implemented by a centrally stored access table [2, 5, 7, 11, 12]. However, applications in distributed environments call for distributed access control (e.g., networks, Internet, distributed databases, web services, distributed operating systems, satellite-TV, etc.). In Section 3 we give access control schemes that are specifically designed for distributed applications.

In this paper we do not consider the dynamics of access control (when users and resources are added and removed and when access rights change). Our results are applicable when systems change only slowly. Dynamical distributed access control is a very difficult problem that does not have easy solutions. We have been studying the problem by beginning with restricted domains and have proposed some dynamic hierarchical access control schemes for specific applications fields such as tree-hierarchies and secure group communications [13].

To illustrate the hierarchies we will use the following simple example, inspired from a college environment. The users, the resources, and the access relation (user-dominant adjacency lists) are given as follows (see box). We will derive a unified hierarchy for these relations in Section 2.

In the next section we define the user and resource hierarchies as well as the unified hierarchy and prove the existence and uniqueness of the unified hierarchy. In Section 3, we discuss the speci-

fication of an access relation (in particular, using the unified hierarchy) and introduce three cryptography based schemes which enforce the access relation; these schemes use the unified hierarchy and are specifically designed for distributed applications.

prof1 → c1, c1A, c3, lab1, lab2, pr1, pr2
prof2 → c2, c3, lab1, lab2, pr1, pr2
grStu1 → c1A, c3, lab1, lab2, pr2
grStu2 → c1A, c3, lab1, lab2, pr2
ugrStu1 → c3, lab1, lab2, pr2
ugrStu2 → c3, lab1, lab2, pr2
. . .
ugrStu100 → c3, lab1, lab2, pr2
secr → c3, pr1, pr2
sysMgr → all resources
sysHelp → all resources except c1 and c2

2. IMPLICIT HIERARCHIES IN AN ACCESS RELATION

A *hierarchy* is formalized by a directed acyclic graph (DAG), which defines a partial order ("hierarchical order") among vertices. A vertex v_i is below a vertex v_j in the hierarchical order ($v_i \leq v_j$) if and only if there exists a directed path in the graph from v_j to v_i.

Let $U = \{u_1, u_2, \dots\}$ be the set of **users** in the system, and let $R = \{r_1, r_2, \dots\}$ be the set of **resources** in the system. The **access relation A** of the system determines which resources each user can legally access and use:

A = $\{(u, r) \in U \times R$: the user u can access the resource $r\}$.

For a user $u \in U$, let $R(u) \subseteq R$ denote the set of resources that u can access; for a resource $r \in R$, let $U(r) \subseteq U$ denote the set of users that can access r. So, $(u, r) \in$ **A** is equivalent to $r \in R(u)$, and also equivalent to $u \in U(r)$. In the following, we assume that the complete access relation is known, and hence all the sets $R(u)$ and $U(r)$ are known.

The user and resource hierarchies are defined as follows:

Definition 1. *Let $u_i, u_j \in U,\ r_i, r_j \in R$.*

$u_i \leq_U u_j$ if and only if $R(u_i) \subseteq R(u_j)$ (i.e., u_i is below u_j in the user hierarchy if and only if the resources that u_i can access form a subset of the resources that u_j can access).

$r_i \leq_R r_j$ if and only if $U(r_j) \subseteq U(r_i)$ (i.e., r_i is below r_j in the resource hierarchy if and only if every user that can access r_j can also access r_i).

$u_i \equiv_U u_j$ if and only if $R(u_i) = R(u_j)$; so, two users are equivalent (regarding access control) if and only if they have exactly the same access rights.

$r_i \equiv_R r_j$ if and only if $U(r_i) = U(r_j)$; so two resources are equivalent (regarding access control) if and only if they are accessible by exactly the same users.

Note that the subset order is reversed for resources, compared to users.

The 'order' relations defined so far are in general not antisymmetric (i.e., $x \leq y$ and $x \geq y$ does not imply $x = y$; see e.g., [4]). To obtain partial orders we **merge equivalent ($\equiv_U$) users** into single groups, and we **merge equivalent resources** into single groups. Note that the result is the same, whether we first merge equivalent users, and then equivalent resources, or vice-versa. From now on, when we say "user" (or "resource"), we will mean a group of equivalent users (respectively, resources). The set of users U, the set of resources R, and the access relation **A** will refer to groups from now on.

Example: For our example from the Introduction, the Figures 1 and 2 represent the user and the resource hierarchies with several mergers between users and resources, obtained from the access relation.

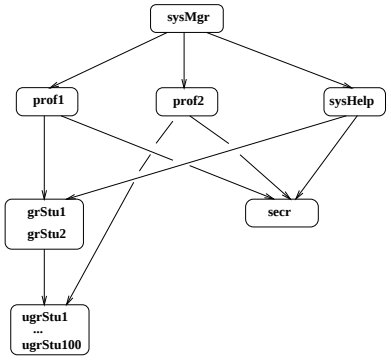

Figure 1: User hierarchy

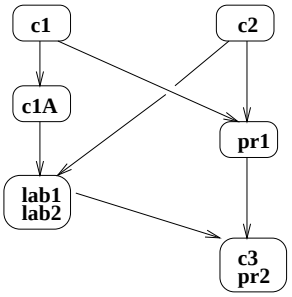

Figure 2: Resource hierarchy

Now (after merging equivalent users, and merging equivalent resources), the users form a partial order (p.o.), called the **user hierarchy**, and denoted by $(U, \leq_U)$; similarly, the resources form a p.o., called the **resource hierarchy**, and denoted by $(R, \leq_R)$.

We will now combine the user hierarchy and the resource hierarchy into a unified hierarchy, defined as follows:

Definition 2. *Let $(U, \leq_U)$ and $(R, \leq_R)$ be p.o.'s (user hierarchy, resource hierarchy respectively), obtained from an access relation* **A**. *The* **unified hierarchy** *is a p.o. $(V, \preceq)$ satisfying the following conditions:*

(1) The user hierarchy is a sub-p.o. of the unified hierarchy; this means: $(U, \leq_U)$ is embedded into the p.o. $(V, \preceq)$ by a one-to-one map $f_U : U \to V$, such that for all $u_i, u_j \in U$: $u_i \leq_U u_j$ if and only if $f_U(u_i) \preceq f_U(u_j)$.

(2) Similarly, the resource hierarchy is a sub-p.o. of the unified hierarchy; this means: $(R, \leq_R)$ is embedded into the p.o. $(V, \preceq)$ by a one-to-one map $f_R : R \to V$, such that for all $r_i, r_j \in R$: $r_i \leq_R r_j$ if and only if $f_R(r_i) \preceq f_R(r_j)$.

(3) A user $u \in U$ has access to a resource $r \in R$ if and only if u is above r in the unified hierarchy; thus, u has access to r if and only if $f_R(r) \preceq f_U(u)$.

(4) The p.o. $(V, \preceq)$ is the smallest p.o. (regarding the size of V), satisfying (1), (2), (3).

The following theorem shows that a unified hierarchy, as just defined, exists and is not larger than the combined size of the two original user and resource hierarchies.

We will use the notation $x \succeq y$ to mean $y \preceq x$.

Theorem 1. *For any user hierarchy $(U, \leq_U)$ (a p.o.) and any resource hierarchy $(R, \leq_R)$ (a p.o.) there* **exists** *a unified hierarchy $(V, \preceq)$ (a p.o. as defined above), and this p.o. is* **unique** *up to 'isomorphism' (i.e., up to renaming the elements of V). For the size $|V|$ of V we have: $|V| \leq |U| + |R|$.*

Moreover, $(V, \preceq)$ can be constructed from $(U, \leq_U)$ and $(R, \leq_R)$ in polynomial time.

Proof. We use the classical notation 2^R for the set of all subsets of R. We will construct the unified hierarchy $(V, \preceq)$ as a sub-p.o. of the p.o. $(2^R, \subseteq)$. (Analogously, we could have based the construction on 2^U, which would have been quite similar.)

(1) We embed the user hierarchy $(U, \leq_U)$ into $(2^R, \subseteq)$ by the map $f_U : u_i \in U \rightarrow f_U(u_i) = R(u_i) \in 2^R$.
(So, $f_U(u_i)$ consists of the resources that u_i can access.) Then f_U is one-to-one (since we merged equivalent users), and $u_i \leq_U u_j$ if and only if $f_U(u_i) \subseteq f_U(u_j)$ (by the very definition of $\leq_U$).

(2) We embed the resource hierarchy $(R, \leq_R)$ into the p.o. $(2^R, \subseteq)$ by the map $f_R : r_i \in R \rightarrow f_R(r_i) = \{r_j \in R : r_j \leq_R r_i\}$.
Then f_R is one-to-one (because $\leq_R$ is a p.o.), and $r_i \leq_R r_j$ if and only if $f_R(r_i) \subseteq f_R(r_j)$ (again because $\leq_R$ is a p.o.).

(3) The third condition of the definition then holds: u_i can access r_j if and only if $r_j \in R(u_i)$ if and only if $(\forall r_k \leq_R r_j)\ r_k \in R(u_i)$ if and only if $f_R(r_j) \subseteq f_U(u_i)$.

Let $V = \{f_U(u) : u \in U\} \cup \{f_R(r) : r \in R\} \subseteq 2^R$. Then the p.o. $(V, \subseteq)$ satisfies conditions (1), (2), (3) of the definition (with $\subseteq$ playing the role of $\preceq$).

Also, clearly $|V| \leq |U| + |R|$.

It is easy to implement the construction of $(V, \preceq)$ in polynomial time; note that we need not consider all of 2^R in the construction.

Minimality of $|V|$ and uniqueness of the minimal unified hierarchy will follow from the following lemma.

Lemma: For any minimal unified hierarchy $(V, \preceq)$ obtained from $(U, \leq_U)$ and $(R, \leq_R)$, with embedding maps f_U and f_R we have:
$f_U(u_i) = f_R(r_j)$ if and only if $R(u_i) = \{r_k : r_k \leq_R r_j\}$.

Proof of the Lemma: By the definition of the unified hierarchy, $f_U(u_i) = f_R(r_j)$ if and only if u_i can access r_j (and hence the descendants of r_j), and no other resources (if u_i could access another r_k, then $f_R(r_j) = f_U(u_i) \succeq f_R(r_k)$ hence $r_j \geq_R r_k$). This proves the Lemma.

Minimality of our construction then follows: Indeed, from the definition, V must contain U and R (via one-to-one maps). To make $|V|$ smaller than $|U|+|R|$, the embedding maps must identify some u_i's with some r_j's. But the Lemma tells us a necessary and sufficient condition for this to happen. In our construction based on 2^R, all these possible identifications do happen, hence $|V|$ is minimal.

Now we can also prove that our construction has minimum size: Let V' be any minimum-size unified hierarchy containing copies of U and R (via embedding maps f'_U, f'_R), according to Def. 2. For $|V'|$ to be smaller than $|U|+|R|$, the embedding maps must identify some $u \in U$ with some $r \in R$: $f'_U(u) = f'_R(r) (\in V')$.
Since $f'_U(u) = f'_R(r)$ we conclude that u can access r (by (3) of Def. 2), and hence $\{r_j \in R : r_j \leq_R r\} \subseteq R(u)$.
On the other hand, if u can access some resource r_j then (again by point (3) of Def. 2), $f'_R(r_j) \preceq f'_U(u) (= f'_R(r))$. Hence (by point (2) of Def. 2), $r_j \leq_R r$. Therefore,

$R(u) \subseteq \{r_j \in R : r_j \leq_R r\}$.

The two set-inclusions imply:

If $f'_U(u) = f'_R(r)$ then $R(u) = \{r_j \in R : r_j \leq_R r\}$.

Then it follows from the Lemma above, that u and r are also identified in our construction of V above. So, our construction of V makes every identification that any minimum-size unified hierarchy V' will do, so our construction is of minimum-size too.

Uniqueness also follows: different minimal unified hierarchies can only differ in the way u_i's are identified with r_j's. But the Lemma tells us that this can only be done in one way.

This proves the Theorem. □

The definition of the unified hierarchy does not tell us explicitly what it means for a resource to be above a user $u \preceq r$). From the construction one can derive the following (and the proof is straightforward):

Proposition In the unified hierarchy the following are equivalent (where u is a user and r is a resource):
$u \preceq r$;
every resource accessible by u is $\leq_R r$;
every user of r is $\geq_U u$;
for every user u_i of r and every resource r_j accessible by u we have: u_i can access r_j;
the Cartesian product $U(r) \times R(u)$ is a subset of **A**.

Example: For the example of the Introduction, Figure 3 represents the unified hierarchy. Note that some users have been merged with resources.

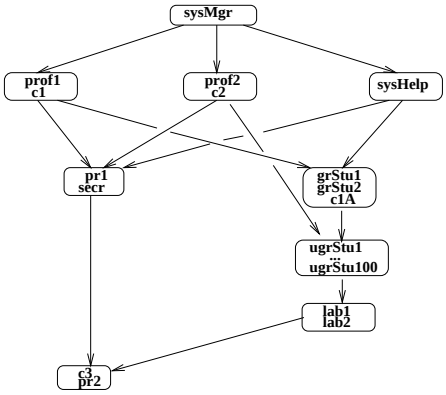

Figure 3: Unified hierarchy

3. IMPLEMENTATION OF ACCESS CONTROL

In this section we use the hierarchies that we introduced in order to develop secure access control schemes. Hierarchies can be used for both the specification and the enforcement of access control. In our enforcement schemes, a user u_i has to prove to a resource r_j that u_i has the right to access r_j, and this should be possible if and only if $r_j \preceq u_i$ with respect to the unified hierarchy. We will present three

basic means to enforce access control: certificates, unconditionally secure keying schemes, and computationally secure keying schemes (based on one-way functions).

3.1. Specification of access control

An access control relation can be given explicitly, by an *access matrix*, which can be useful for theoretical reasonings but is wasteful for space. A more compact description of the access control relation can be given by *adjacency lists*: user-dominant adjacency lists or resource-dominant adjacency lists.

The *unified hierarchy* can also be used to describe the access relation. In this case, the hierarchy is given as a graph in which every vertex is labeled by the set of equivalent users or resources (or some of both) that are represented by this vertex. Because of the merger of equivalent users or resources, and the merger of some users with some resources, the unified hierarchy is a representation which is as compact as (and usually more compact than) the adjacency list representation. Moreover, the unified hierarchy has the advantage that certain queries are more efficient: Given a user u_i or a resource r_j it is easy to find the adjacency list of u_i or r_j (namely, pick all the resources that are $\preceq u_i$, respectively, all the users that are $\succeq r_j$). In the user-dominant adjacency list representation, it is tedious to find a resource's adjacency list; on the other hand, if both user- and resource-dominant adjacency lists are explicitly given, storage is wasted. In the rare cases when no mergers occur, the unified hierarchy loses its compactness advantage (however, one still has to consider the concepts and go through some of the construction of the unified hierarchy, in order to find out that no users are merged with resources). In any case, the unified hierarchy keeps an advantage regarding queries. Thus, the unified hierarchy could serve as a representation of the access relation, which is both compact and efficient for queries.

Various mixed representations of the access relation are also possible: we might be given partial information about adjacency lists, about the user and resource hierarchies, or information about equivalence of some users or some resources. This may arise in specifications, and one could be asked to reconstruct the entire unified hierarchy from these data.

In a distributed environment, partial information about the access relation or the unified hierarchy will be distributed among the users and the resources; no central authority is needed (except may be at the set-up of the system or for occasional maintenance and updates).

In the next three subsections we give schemes for enforcing an access relation.

3.2. Certificate-based schemes

In these schemes a trusted certificate authority (CA) distributes certificates to users. When a user accesses a resource the protocol is as follows: the user provides an access request along with a certificate. The resource then verifies the user's access right based on this certificate (without consulting the CA).

It is natural to assume that users know which resources they can access. A user u_i may have a certificate of the following form for each resource r_j that u_i can access:

[u_i's ID, (u_i, r_j), cert.-valid-time, CA-sig.].

Here, u_i's ID identifies the user, (u_i, r_j) indicates the access right, and the CA's digital signature certifies to the resource that the information in the certificate is correct. We refer to books on cryptography for more information on certificates and digital signatures (e.g., [9]).

Alternatively, instead of having a different certificate for each resource that u_i can access, u_i might have just one certificate that lists all of $R(u_i)$ (i.e., all the resources accessible to u_i). This approach may be simpler when the number of resources is small; but it gives more information to a resource than this resource needs to know.

In any case, no information about the access relation or the hierarchies needs to be stored in the resources.

A disadvantage of this scheme comes from a general problem with certificates: it is hard to keep certificates up to date when the system changes (the revocation problem – see [9]).

3.3. Unconditionally secure keying schemes

In this approach every vertex v in the unified hierarchy has a key k_v. Moreover, depending on the unified hierarchy order $\preceq$, each user at v knows a set of keys $U_v \subseteq \{k_w : w \preceq v\}$ (i.e., the user knows some keys of lower-ranking resources), and each resource at vertex v knows a set of keys $R_v \subseteq \{k_w : v \preceq w\}$ (i.e., some keys of higher-ranking users); the sets U_v and R_v should be chosen in such a way that

$$v_i \preceq v_j \text{ if and only if } U_{v_j} \cap R_{v_i} \neq \emptyset.$$

Moreover, the sets U_v and R_v should be such that one cannot guess any key contained in any of these sets. In particular, we assume that the keys k_v are long enough so that they cannot be guessed.

When a user u_j (at vertex v_j) requests a resource r_i (at vertex v_i) he presents his set U_{v_j} to the resource; the resource then checks whether $U_{v_j} \cap R_{v_i} \neq \emptyset$, which holds if and only if $v_i \preceq v_j$, i.e., if and only if u_j has the right to access r_i. (In this protocol, a resource can get information about the keying material held by users; this could however be avoided by adding 'challenge-response' methods into the protocol – see [9].)

We will illustrate this approach by simple special cases, namely 'user multiple keying', 'resource multiple keying', and 'mixed keying'.

User multiple keying:

In this scheme we have for every vertex v: $R_v = \{k_v\}$ and $U_v = \{k_{v_j} : v_j \preceq v\}$.

This scheme can also be implemented by directly using the access relation: Then for every resource r and every user u we have $R_r = \{k_r\}$ and $U_u = \{k_{r_j} : u \text{ can access } r_j\}$.

A user u requesting access to a resource r presents U_u to r. The resource r verifies u's access right by checking whether $k_r \in U_u$.

For example (see Figure 4), $U_{v_2} = \{k_2, k_4, k_5\}$, $R_{v_2} = \{k_2\}$, $R_{v_4} = \{k_4\}$, $R_{v_5} = \{k_5\}$, therefore user u_2 can access resources r_2, r_4, and r_5.

Resource multiple keying:

This scheme is similar to User multiple keying, with the roles of user and resources switched.

Mixed keying:

This is the general case. For example (see Figure 5), $U_{v_1} = \{k_1, k_5\}$, and $R_{v_2}, R_{v_3}, R_{v_4}, R_{v_6}$ contain the key k_1, so the user at vertex v_1 can access the resources at v_2, v_3, v_4, v_6, and he can also access the resource at v_5 because they share k_5. One can check from the graphs that $v_i \leq v_j$ if and only if $R_{v_i} \cap U_{v_j} \neq \emptyset$

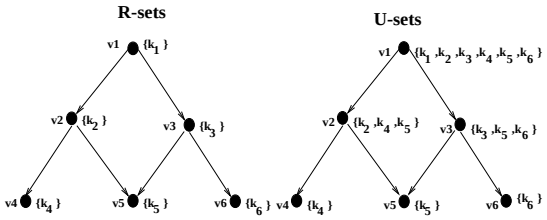

Figure 4: User multiple keying

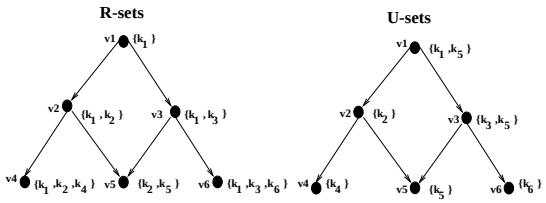

Figure 5: Mixed-Keying

3.4. One-way function based keying schemes

A drawback of the unconditionally secure schemes presented above is that the total amount of secret information that the users or the resources have to store can be quite large (proportional to the number of vertices in the unified hierarchy). One-way functions can drastically improve this. We present one way of using one-way functions, namely a generalization of Lin's scheme [8]. Other methods could be generalized as well by using the the unified hierarchy (e.g., the scheme of Akl and Taylor [1]).

A generalization of Lin's scheme

Lin's original scheme [8] assumed that the user hierarchy and the resource hierarchy were the same. However, we do not need this assumption; we will simply apply Lin's method to the unified hierarchy $(V, \preceq)$. Moreover, we will use any one-way function $F : \mathbf{K} \times \mathbf{I} \rightarrow \mathbf{K}$, where $\mathbf{I}$ is the space of vertex identifiers ('IDs'), and $\mathbf{K}$ is a large key space.

Every vertex $v \in V$ is assigned its own independent key $k_v \in \mathbf{K}$. Only v is explicitly given k_v.

For all vertices v, w such that $v \preceq w$, let $r_{vw} = F(k_w, v) \oplus k_v$, where $\oplus$ is bitwise exclusive OR. If $v \not\preceq w$, we choose r_{vw} to be a random element of $\mathbf{K}$. In any case, the elements (r_{vw}, v, w) (as v and w range over V) are made public. The one-way function F is also made public.

Now, if $v \preceq w$ then w can compute k_v, using k_w (which w knows) and r_{vw} (which is public): $k_v = F(k_w, v) \oplus r_{vw}$. On the other hand, if $v \not\preceq w$, the element r_{vw} is random and carries no information (and the relations $r_{vw} = F(k_w, v) \oplus k_v$ and $k_v = F(k_w, v) \oplus r_{vw}$ do not hold, with high probability). A user associated with vertex w accesses a resource associated with vertex v by presenting k_v to v. (Lin used a more special one-way function, which required some set-up work, namely $F(k_w, v) = g^{k_w + N_v} \ mod \ p$, where p is a large prime number, g is a primitive element $mod\ p$, and N_v is a numerical identifier of v.)

The advantages of this scheme are that every vertex can select its own key and a vertex does not need to remember the entire hierarchy.

4. CONCLUSION

We showed that three hierarchies can be extracted from an access relation: a user hierarchy, a resource hierarchy, and a unified hierarchy. These hierarchies allow compact specifications of access control, and are useful for schemes that enforce an access relation. Cryptographic key-based hierarchical schemes can be designed to effectively enforce and implement access control in distributed environments. Other issues such as general dynamic access control and specification of negative access relations are challenging problems which we plan to investigate in the future.

REFERENCES

[1] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–247, March 1983.

[2] E. Bertino, S. Jajodia, and P. Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2):101–140, April 1999.

[3] G. C. Chick and S. E. Tavares. Flexible access control with master keys. *Advances in Cryptology: CRYPTO '89 LNCS*, 435:316–322, 1990.

[4] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[5] H. M. Gladney. Access control for large collections. *ACM Transactions on Information Systems*, 15(2):154–194, April 1997.

[6] S. J. Greenwald. A new policy for distributed resource management and access control. *Proceedings of the UCLA Conference on New Security Paradigms Workshops*, pages 74–86, 1996.

[7] T. Jaeger, A. Prakash, J. Liedtke, and N. Islam. Flexible control of downloaded executable content. *ACM Transactions on Information and Systems Security*, 2(2):177–228, May 1999.

[8] C. H. Lin. Dynamic key management schemes for access control in a hierarchy. *Computer Communications*, 20:1381–1385, 1997.

[9] A. Menezes, P. V. Ooschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, Inc., Boca Raton, Florida, 1996.

[10] R. S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27:95–98, 1988.

[11] G. Thomas and G. R. Thompson et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, September 1990.

[12] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the TAOS operating system. *ACM SIGOPS*, pages 256–269, December 1993.

[13] X. Zou. Secure group communications and hierarchical access control. *PhD. Thesis, University of Nebraska-Lincoln, USA*, December 2000.