Key Management for Non-Tree Access Hierarchies*

Mikhail J. Atallah, Marina Blanton, Keith B. Frikken
Department of Computer Science
Purdue University
{mja,mbykova,kbf}@cs.purdue.edu

ABSTRACT

Access hierarchies are useful in many applications and are modeled as a set of access classes organized by a partial order. A user who obtains access to a class in such a hierarchy is entitled to access objects stored at that class, as well as objects stored at its descendant classes. Efficient schemes for this framework assign only one key to a class and use key derivation to permit access to descendant classes. Ideally, the key derivation uses simple primitives such as cryptographic hash computations and modular additions. A straightforward key derivation time is then linear in the length of the path between the user's class and the class of the object that the user wants to access.

Recently, work presented in [2] has given an efficient solution that significantly lowers this key derivation time, while using only hash functions and modular additions. Two fast-key-derivation techniques in that paper were given for trees, achieving $O(\log\log n)$ and $O(1)$ key derivation times, respectively, where n is the number of access classes. The present paper presents efficient key derivation techniques for hierarchies that are not trees, using a scheme that is very different from the above-mentioned paper. The construction we give in the present paper is recursive and uses the one-dimensional case solution as its base. It makes a novel use of the notion of the *dimension* d of an access graph, and provides a solution through which no key derivation requires more than $2d+1$ hash function computations, even for "unbalanced" hierarchies whose depth is linear in their number of access classes n.

The significance of this result is strengthened by the fact that many access graphs have a low d value (e.g., trees correspond to the case $d = 2$). Our scheme has the desirable property (as did [2] for trees) that addition and deletion of edges and nodes in the access hierarchy can be "contained" in the node and do not result in modification of keys at other nodes (no wholesale re-keying as changes are made to the access hierarchy).

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*graphs and networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems.; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous

General Terms

Algorithms.

Keywords

Access hierarchy, fast key derivation, dimension of a graph.

1. INTRODUCTION

The problem of key management for hierarchical access control is important for many applications and has received significant attention in the research literature. In this framework, all users are divided into disjoint access classes and each access class inherits the privileges of its descendants. Access is based on key derivation where users receive keys that allow them to obtain access to the authorized objects without interaction with the server, through a key derivation process. That is, for any given user, the key issued to that user allows her to access objects stored at her access class as well as objects stored at all descendant classes in the hierarchy.

Applications where such access hierarchies are useful include the Role-Based Access Control (RBAC) models, which are naturally modeled as hierarchies. Whereas an organization's role hierarchy tends to be shallow rather than deep, in a number of contexts the access hierarchies have a large size and depth. These include hierarchically organized distributed control structures such as physical plants or power grids (involving thousands of networked devices such as sensors, actuators, etc.); hierarchically organized hardware where the hierarchy is based on functional, control, and trust considerations; hierarchical design structures of large complex systems such as aircrafts and VLSI circuits; subscription services where the set of included programs or media depends on the subscription package type; task graphs where descendant tasks are known only to their ancestor tasks. Deep access hierarchies can also arise in simple databases where the hierarchical complexity can come

*Portions of this work were supported by Grants IIS-0325345, IIS-0219560, IIS-0312357, and IIS-0242421 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

SACMAT'06, June 7–9, 2006, Lake Tahoe, California, USA.
Copyright 2006 ACM 1-59593-354-9/06/0006 ...$5.00.

from super-imposed classifications on the database that are based on functional or structural features of the database. See also [18, 23] for other examples of deep hierarchies.

Normally, an access hierarchy can be modeled as a directed access graph G, where each node corresponds to an access class and edges preserve relations between the nodes. Recent key management schemes achieve the following properties:

- each node in the access graph has a single secret key associated with it;
- the amount of public information for the key assignment scheme is asymptotically the same as that needed to represent the graph itself;
- key derivation involves only the usage of efficient cryptographic primitives such as one-way hash functions;
- given a key for node v, the key derivation for its descendant node w takes ℓ steps, where ℓ is the length of the path between v and w;
- it is impossible for dishonest colluding users to obtain access to more objects than what they can already legitimately access, ensuring collusion resilience of the scheme.

While the above is computation- and space-efficient, for large and deep hierarchies the key derivation process can require up to $O(n)$ steps, where n is the number of nodes in the graph. This operation must also be performed in real time. Thus, if such key derivation is performed by a device with computation and space limitations (such as cheap and possibly disposable smartcards), this key derivation computation will be larger than what the device can handle. Atallah et al. [2] recently provided techniques for reducing the key derivation time for tree hierarchies to $O(\log\log n)$ and $O(1)$ steps using two different schemes. The present paper significantly extends that work by showing how key derivation can be greatly reduced for more general access graphs, and gives techniques for achieving no more than $2d+1$ key derivation steps, where d is the dimension of the graph (and many practical access graphs are of low dimension, typically less than 4). The scheme has the desirable property (as did [2] for trees) that the addition and deletion of nodes in the access hierarchy can be "contained" in the node and do not result in the modification of keys at other nodes (similarly for edge additions and deletions); hence no wholesale rekeying is done as changes are made to the access hierarchy.

The rest of this paper is organized as follows: Section 2 provides a review of related literature. Section 3 gives a brief overview of the new technique. In section 4 we present background information and definitions. Section 5 gives the details of our approach, while section 6 supplements it with comments on how dynamic changes to hierarchies can be addressed. Finally, section 7 concludes this work.

2. RELATED WORK

Prior research literature on hierarchical access control is very extensive (see, e.g., [1, 3, 4, 6, 9, 10, 13, 14, 16, 17, 19, 22, 24, 25, 26, 29, 32]), and its overview is beyond the scope of this paper. In this section, we focus on the publications most closely related to this work. We only note that a rather large number of schemes in the prior literature (e.g., [1, 17, 10, 3, 12, 11, 6, 19, 13, 22, 16, 26]) seem to derive any key in a single step, but that single step is significantly more expensive than a chain of key derivations in our and similar schemes. More precisely, they operate on large numbers computed as a product of up to $O(n)$ coprime numbers or, alternatively, up to $O(n)$ large numbers, where n is the number of nodes in the graph. Such numbers will grow to n bits long and are prohibitively large for most hierarchies. While in many of these approaches key derivation might consist of one division and one modular exponentiation operation, in practice, division of two numbers $O(n)$ bits long involves $O(n^2)$ operations, in addition to their use of expensive public-key crypto operations. Our key derivation, on the other hand, even in the base scheme is bounded by the depth of the access hierarchy and does $O(n)$ hash operations in the worst case. And the improved scheme described in the present paper exhibits significantly better key derivation time.

A number of recent schemes [2, 5, 7, 15, 33] give more efficient solutions to the key assignment and management in access hierarchies. All of these schemes have similar overall structures with the following properties: (i) each node in the access graph has a single secret key; (ii) public information associated with the scheme is asymptotically the same as that needed to represent the access graph itself; (iii) key derivation involves only efficient operations such as one-way hash functions and bitwise XOR[1]; and (iv) the number of steps in key derivation is the same as the length of the path between the user's node and the target node (the node whose key is being derived). These constructions differ, resulting in some of them being more efficient and simpler than others, but the general structure remains the same (not all the properties of these approaches were stated in the above form, but for the sake of comparison, here we unify the notation to describe their capabilities).

Dynamic changes to the hierarchy are handled differently in these schemes: in [5, 7, 15, 33] insertions and re-keying are local, but deletions affect secret keys of all descendant nodes; in [2], however, all changes to the graph are local. In addition, only the scheme of [2] has a formal proof of security (in the presence of an active adversary who can adaptively corrupt nodes), while other schemes provide only informal discussion of attacks. Results of [15, 7] also assume tamper-resistance of clients.

Since the inspiration of the results reported in this work comes from the shortcut techniques of [2] (which allow to reduce the key derivation time for trees), here we give a brief description of the ideas underlying the shortcut techniques of [2]. As was mentioned earlier, for trees (whose dimension is $d = 2$), [2] reports two results: (i) addition of $O(n)$ shortcut edges[2] results in key derivation time being no more than $O(\log\log n)$ for n-node hierarchies, and (ii) addition of $O(n\log\log n)$ shortcut edges results in key derivation time being no more than 3 steps (each step is a computation of the form $\mathsf{H}(x,y) - z \bmod 2^\rho$, i.e., requires one application of a one-way function and one subtraction operation for a security parameter ρ). One idea in [2] is to use the notion of a *centroid*: a centroid of an n-node tree T is a node whose re-

[1] The scheme of [5] additionally utilizes symmetric key encryption.

[2] A shortcut edge is an additional new edge that, while not in the original graph G, is in the transitive closure of G.

moval from T leaves no connected component of size greater than $n/2$. In addition, a modified variant of *centroid decomposition* is used: to compute a centroid decomposition, compute the centroid of the tree, remove it, and then recursively repeat this process with the remaining trees. The modified variant used in that paper is the so-called "prematurely terminated centroid decomposition," which is similar to the above centroid decomposition, except that the recursion stops not when the tree becomes a single node, but rather when the tree size becomes $\leq \sqrt{n}$. The centroids used in this type of tree decomposition and the root of the tree are called the "special nodes." These special nodes are caused to become "well connected" with the addition of shortcuts, and the nodes of each small residual subtree of size $\sqrt{n}$ are also caused to become well connected with the addition of shortcuts. Then to reach any node from another node, we first need to reach the special node of the current residual tree, then jump to the special node of the target residual tree, and finally reach the target node within that small tree.

As non-tree hierarchies have no centroids, the above scheme does not readily extend to non-tree access hierarchies. The extension turns out to require a new, different technique. We briefly sketch it (at a high level) next.

3. OVERVIEW OF THE NEW TECHNIQUE

The essence of the new technique consists of three ingredients:

1. The judicious addition of new "dummy" vertices that make it possible to add a remarkably small number of shortcuts to achieve the desired fast-key-derivation performance. Note that the dummy vertices and their associated keys are internal to the system (used purely for performance reasons) and that no access classes correspond to them (e.g., they are not roles in the RBAC sense). Unlike [2], where shortcut edges were *in addition* to the original edges of the hierarchy, in the present paper the only explicit (i.e., "key-derivation") edges that remain are the shortcut edges (some of them may of course coincidentally correspond to edges in the original graph, but this need not happen and it is indeed conceptually possible for all the original access graph's edges to "vanish"). The addition of dummy vertices and shortcut edges is a novel technique in this area, and we believe it has much promise beyond enabling the specific performance bounds that we claim in the present paper.

2. As it is cumbersome to carry out the computation of the dummy vertices and of the shortcuts using the original graph, and also difficult to subsequently use them, we first "transform" the graph into a d-tuple-representation of the vertices where d is the dimension of the partial order represented by the graph; this transformation step is not needed if the graph is already specified in the d-tuple form, e.g.,

   "node v is ancestor of node w iff v has both a higher value than w and is also less vulnerable than w".

   We believe this representation of the access graph will have uses other than the present framework of key-derivation.

3. In the above-mentioned representation, it becomes possible to carry out the desired computation of the dummy vertices and of the shortcut edges with the claimed performance – the algorithm for doing this is an important contribution of our paper. We prove precise bounds for that algorithm, both in terms of its consumption of resources (time and space), and in terms of the key-derivation performance that is made possible by the data structure that it produces.

4. BACKGROUND

4.1 User hierarchy

In our model, each user belongs to one of the disjoint access classes $C_1, C_2, \ldots, C_n$. The classes are organized into a hierarchy described by a directed acyclic graph G, i.e., a partial order relationship. The users at access class C_j can access information at access class C_i if and only if there is a path in G from node C_j to node C_i.

The above-mentioned directed access graph $G = (V, E, O)$ is such that V is a set of vertices $V = \{v_1, \ldots, v_n\}$, E is a set of edges $E = \{e_1, \ldots, e_m\}$, and O is a set of objects $O = \{o_1, \ldots, o_k\}$. Each vertex v_i represents a class in the access hierarchy and has a set of objects $O(v_i) \subset O$ associated with it.

4.2 The shortcut technique

Here we review the shortcut technique of [2] that results in $O(n \log \log n)$ additional edges and key derivation being at most 3 steps for a n-node tree T, as this technique could be used as the basis of the recursion for the techniques given in this work.

As was mentioned earlier, the tree T is decomposed into $\sqrt{n}$ residual trees of $\sqrt{n}$ nodes each using centroid decomposition. The centroids used in the decomposition are called the special nodes, and the residual trees are denoted as $T_1, \ldots, T_k$. In addition, there is a notion of a reduced tree, denoted $\hat{T}$, that consists of the special nodes and edges that satisfy the following condition: there is an edge from node v to node w in $\hat{T}$ iff (i) v is an ancestor of w in T, and (ii) there is no other node of $\hat{T}$ on the v-to-w path in T. Then the following procedure adds shortcut edges to the tree T. In what follows, $|T|$ denotes the number of nodes in T.

AddShortcuts(T):

1. If $|T| \leq 4$ then return an empty set of shortcuts. Otherwise continue with the next step.

2. Compute the special nodes of T in linear time. Initialize the set of shortcuts S to be empty.

3. Create, from T, the reduced tree $\hat{T}$ and add to S a shortcut edge between every ancestor-descendant pair in $\hat{T}$ (unless the ancestor is a parent of the descendant, in which case there is already such an edge in T). Because $\hat{T}$ has $O(\sqrt{|T|})$ vertices, the size of S is $O(|T|)$.

4. For every residual tree T_i in turn ($i = 1, \ldots, k$), add to S a shortcut edge from the root of T_i to every node in T_i that is not a child of that root. This increases the size of S by no more than $\sum_{i=1}^{k} |T_i|$, which is $\leq |T|$.

5. For every residual tree T_i in turn ($i = 1, \ldots, k$), add to S a shortcut edge from each node N in T_i (other than the root) to all nodes in $\hat{T}$ that are both: (i) descendants of N and (ii) children of the root of T_i in $\hat{T}$. This adds at most $O(|T|)$ edges to the shortcut set: For each node SN in $\hat{T}$, all of the new edges that point to SN come from at most one tree (as SN has at most one parent in $\hat{T}$). Furthermore, since each tree has at most $O(\sqrt{|T|})$ nodes, there are at most $O(\sqrt{|T|})$ edges pointing to SN that are added during this step. Finally, there are only $O(\sqrt{|T|})$ nodes in $\hat{T}$, and so there are at most $O(|T|)$ edges added during this step.

6. For every residual tree T_i in turn ($i = 1, \ldots, k$), recursively call $\mathsf{AddShortcuts}(T_i)$ and, if we let S_i be the set of shortcuts returned by that call, then we update S by doing $S = S \cup S_i$.

7. Return S.

There is a corresponding $\mathsf{FindPath}()$ procedure that, given two nodes v and w, finds a path from node v to node w of length at most 3 edges in the graph. We omit its description here.

4.3 Dimension of an access hierarchy

An n-vertex access hierarchy G is a partial order, and it is well known that any partial order can be represented as the intersection of t total orders, with the smallest t for which this is possible being the *dimension* of the partial order (see, e.g., [8, 28]). That is, it is possible to associate with every vertex v of G a t-tuple $(x_{v,1}, \ldots, x_{v,t})$ such that:

1. Every $x_{v,j}$ is an integer between 1 and n.

2. If $v \neq w$ then $x_{v,j} \neq x_{w,j}$ for every $1 \leq j \leq t$.

3. Node v is ancestor of node w in G if and only if $x_{v,j} > x_{w,j}$ for every $1 \leq j \leq t$.

We denote the dimension of G by $d(G)$, or by d when G is understood. While computing the dimension of an arbitrary partial order is NP-complete [31], and even approximating it to within a constant factor is not known to be in P, the dimension of many access hierarchies is small: The dimension of a tree is 2 (trivially so), and it was also shown in [27] that a G whose transitive reduction is planar has dimension ≤ 3 and the 3-tuples representing it are computable in linear time. If the transitive reduction of G is 4-colorable then its dimension is ≤ 4 [27]. Many access hierarchies are 4-colorable; especially those for organizational hierarchies.

There are, however, some hierarchies with higher dimension; for example, in the Bell-LaPadula model with k categories (denoted by $s_1, \ldots, s_k$) and ℓ classifications (denoted by $c_1, \ldots, c_\ell$), the dimension of the lattice is $k + 1$. However, computing the tuple representation for this model is straightforward: The access level c_i with categories in the set S is converted into a tuple $(i, x_1, \ldots, x_k)$ where $x_i = 1$ if and only if $s_i \in S$ and is 0 otherwise. It can be easily verified that this conversion faithfully implements the access control policy.

We may actually not need to compute the dimension, but rather *any* d'-tuple representation of the graph with a small enough d'. Moreover, some access graphs can naturally be specified in such a tuple representation, when the "ancestor" relationship is the conjunction of a number of total-order conditions such as "v has higher security clearance than w", "v is a higher-priority asset than w", "v is more vulnerable than w", "v is a higher-paying class of subscribers than w", etc. In summary, the techniques of this paper generalize the shortcut technique to any access hierarchy where a tuple-based representation (of reasonable dimension) can be found. This is a significant improvement over the previous work that supported only trees.

5. OUR SCHEME

We give a solution that achieves key derivation in no more than (and typically less than) $2d + 1$ hash function computations (each of which corresponds to following one shortcut edge). The public space used in this scheme is $O(n(\log n)^{d-1})$.

5.1 The base case of $d = 1$

The case of $d = 1$ was covered in [2], and we could use it as the base case in describing the solution for higher dimensions.[3] That solution, however, results in $O(n \log \log n)$ edges to be added to the graph (see section 4.2), while for $d = 1$ a more efficient solution with $O(n)$ additional edges and the same 3-hop worst case key derivation time is possible. We describe such a solution next. In what follows, let the nodes in a one-dimensional graph be numbered v_1 through v_n such that for each $v_1 \leq v < v_n$ there is an edge from node v_i to v_{i+1}, i.e., node v_i is a parent of node v_{i+1}.

$\mathsf{AddShortcuts}(G)$:

1. Create a set of special nodes S consisting of every $\sqrt{n}$th node in the graph. That is, initialize S with $\{v_1\}$ and then add nodes $v_{j\sqrt{n}+1}$ for all j such that $j\sqrt{n} \leq n$. If $v_n \notin S$, set $S = S \cup \{v_n\}$.

2. Insert new edges between the nodes in S to form the transitive closure of the set.

3. For each node $v_i \notin S$, find $v_j \in S$ such that $j < i$ and $i < j + \sqrt{n}$. Insert an edge (v_j, v_i) if it is not already present.

4. For each node $v_i \notin S$, find $v_j \in S$ such that $i < j$ and $j - \sqrt{n} < i$. Insert an edge v_i, v_j if it is not already present.

Figure 1 depicts different stages of the above algorithm. It is clear that this procedure results in additional $O(n)$ edges and there is a path of length at most three between any two nodes.

Rather than immediately giving the solution for arbitrary d, for expository reasons we choose to first present the solution for $d = 2$, because the two dimensional case is easier to grasp intuitively than the higher-dimensional one. Once the basic idea has been presented (with good intuition) for $d = 2$, we give the general construction for arbitrary d.

[3] The techniques of [2] work for trees with dimension $d = 2$. We, however, cannot use these techniques for the case of $d = 2$, because not all graphs of dimension two are trees.

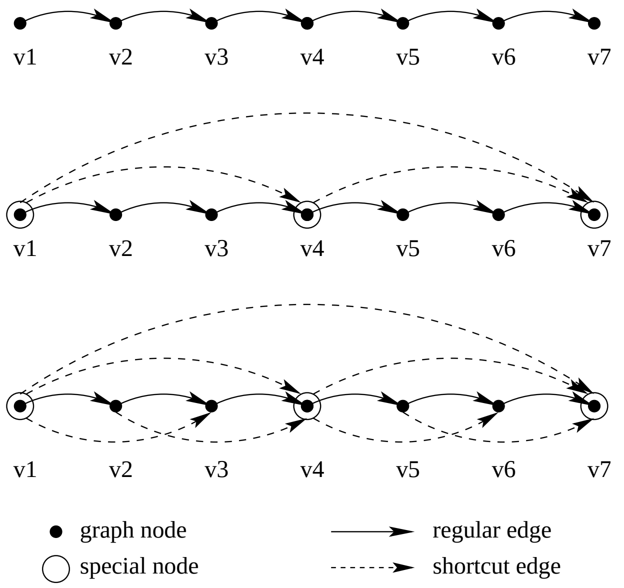

Figure 1: Addition of shortcut edges for dimension $d = 1$: (a) the original hierarchy, (b) the hierarchy after selection of special nodes and constructing their transitive closure, and (c) the final hierarchy.

5.2 The case $d = 2$

The fact that the graph G has dimension 2 implies that every vertex v can be replaced by a pair of numbers $(x(v), y(v))$, such that w is ancestor of v in G if and only if w *dominates* v, i.e., $x(w) \geq x(v)$ and $y(w) \geq y(v)$. From now on, for convenience, we refer to "points" rather than "vertices". A *shortcut* is an ordered pair of points w, v describing an extra "key-derivation edge" that will be added from point w to point v, and that allows the key for v to be derived from the key of w in one step consisting of a hash computation and an addition.[4]

The input is a set V of n points in 2-dimensional space, and the desired output includes a set S of shortcuts between pairs of points (some of which may not belong to V) such that (i) $|S| = O(n \log n)$, and (ii) given any pair of points $v, w \in V$ such that w dominates v, there is a path of at most 5 shortcut edges from w to v. The output also includes the set P that contains V as well as the additional dummy points (i.e., points not in V but that are touched by edges in S).

The solution steps are as follows.

1. Initialize $P = V$, and initialize S to be empty.

2. If $|V| = 1$ then return P and S.

3. If $|V| > 1$ then compute a median line M that is perpendicular to the y axis and partitions V into two equal sets V_1 and V_2 where V_1 (V_2) is left (right) of M. Let V_1' (V_2') be the projection of V_1 (V_2) on line M.

4. Add to S the following shortcut edges:
   - a shortcut edge from every point of V_1' to its corresponding point of V_1;
   - a shortcut edge from every point of V_2 to its corresponding point of V_2'.

5. Recursively build the shortcut edges and dummy points for the set V_1. Let that recursive call return P_1 as the set of points (including dummies) and S_1 as the set of shortcut edges within P_1. Update S and P as follows: $S = S \cup S_1$ and $P = P \cup P_1$.

6. Recursively build the shortcut edges and dummy points for the set V_2. Let that recursive call return P_2 as the set of points (including dummies) and S_2 as the set of shortcut edges within P_2. Update S and P as follows: $S = S \cup S_2$, and $P = P \cup P_2$.

7. Solve the 1-dimensional problem consisting of $V_1' \cup V_2'$ using the result of section 5.1. Let this return a set of edges S_3 (note that the solution of [2] returns only a set of edges, i.e., it does not add any dummy points). Update (i.e., augment) S as follows: $S = S \cup S_3$. (P stays the same.).

According to the previous subsection, the base case of $d = 1$ is solvable with no more than 3 shortcut edges and $O(n)$ space complexity. We use this in the analysis below.

The space complexity (i.e., the number of shortcut edges and dummy points) of the above-described scheme obeys a recurrence of the form $f(n) \leq 2f(n/2) + c_1 n$ if $n > 1$, and $f(n) = O(1)$ if $n = 1$, whose solution is $O(n \log n)$. Note that this recurrence follows from step 4 (which recursively solves the problem for (n/2) points), step 5 (which recursively solves the problem for (n/2) points), step 7 which adds $O(n)$ edges, and step 1 which adds $O(n)$ points.

That any w-to-v number of shortcut edges is at most 5 is proved by induction on n (the base case being trivial): If $w \in V_2$ and $v \in V_1$, then the path of length at most 5 consists of following one edge from $w \in V_2$ to its projection $w' \in V_2'$, at most 3 edges from w' to the point $v' \in V_1'$ that is the projection of v on M, and one edge from v' to v. When both points v and w are in V_1 or both are in V_2, the claim follows from the induction hypothesis.

5.2.1 *An example*

To help clarify our shortcut technique, we give an example of the recursive step in the previous section. Figure 2 shows a tree access hierarchy that will be used for this example.

Figure 3 contains a set of points in two dimensions that represents a tree's access structure. Note that if a point dominates another point in this figure, then the dominating point must have a path to the dominated point in the final structure.

Figure 4 shows the shadow points (added in step 3 and denoted by open circles) for the previous figure. Note that the shadow points are on a one dimensional plane (i.e., a line). This figure also shows the transitions from normal points to shadow points and vice versa (as described in step 4). Also note that the shadow points will be linked in step 7.

[4] The results reported imply usage of the key derivation scheme of [2], which is provably secure against key recovery in the presence of an active adversary. Usage of a different scheme with the same properties or a scheme with stronger security properties (e.g., security in the key indistinguishability setting) will imply different, normally higher, computational assumptions.

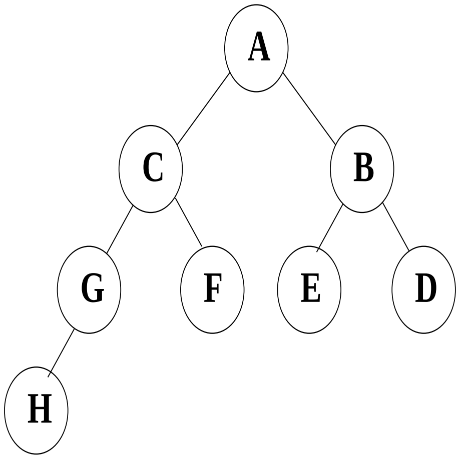

Figure 2: Two dimensional access hierarchy (original).

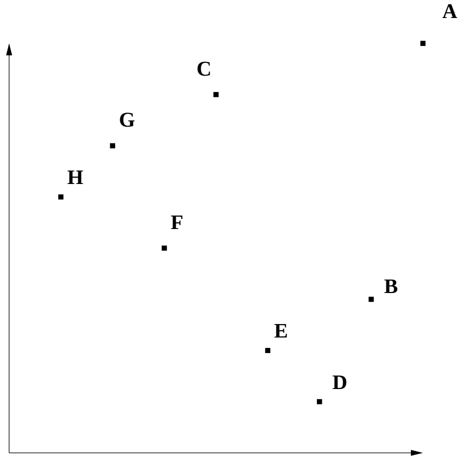

Figure 3: Two dimensional access hierarchy (converted to tuple form).

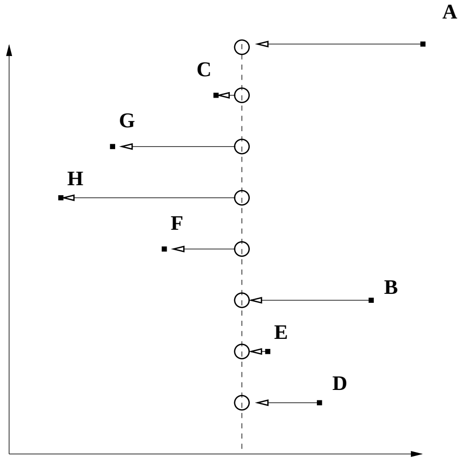

Figure 4: Hierarchy with shadow points.

5.3 The case $d \geq 3$

The fact that the graph G has dimension d implies that every vertex v can be replaced by a d-tuple of numbers $(x_1(v), \ldots, x_d(v))$, such that w is ancestor of v in G if and only if w *dominates* v, i.e., $x_i(w) \geq x_i(v)$ for all $i \in \{1, \ldots, d\}$.

The input is a set V of n d-dimensional points, and the desired output includes a set S of shortcuts between pairs of points (some of which may not belong to V) such that (i) $|S| = O(n(\log n)^{d-1})$, and (ii) given any pair of points $v, w \in V$ such that w dominates v, there is a path of $O(d)$ shortcut edges from w to v. The output also includes the set P that contains V as well as the additional dummy points (i.e., points not in V but that are touched by edges in S).

As we did for the 2-dimensional case, the construction we use is recursive. Specifically, we inductively assume that the $d-1$ dimensional problem can be solved with $O(n(\log n)^{d-2})$ edges and with a key derivation path of $2d+1$ (note that this holds for $d = 1$ and for $d = 2$ by the previous subsections).

The solution steps are as follows:

1. Initialize $P = V$, and initialize S to be empty.

2. If $|V| = 1$, then return P and S.

3. If $|V| > 1$, then compute a $d - 1$ dimensional hyperplane M, perpendicular to the dth dimension, that partitions V into two equal sets V_1 and V_2, where V_1 is the set of points that are on the smaller side of the hyperplane (according to their dth coordinate). Let V_1' be the projection along dimension d of V_1 on hyperplane M. Let V_2' be the projection of V_2, along dimension d, on hyperplane M.

4. Add to S the following shortcut edges:
   - a shortcut edge from every point of V_1' to its corresponding point of V_1;
   - a shortcut edge from every point of V_2 to its corresponding point of V_2'.

5. Recursively build the shortcut edges and dummy points for the set V_1. Let that recursive call return P_1 as the set of points (including dummies) and S_1 as the set of shortcut edges within P_1. Update S and P as follows: $S = S \cup S_1$ and $P = P \cup P_1$.

6. Recursively build the shortcut edges and dummy points for the set V_2. Let that recursive call return P_2 as the set of points (including dummies) and S_2 as the set of shortcut edges within P_2. Update S and P as follows: $S = S \cup S_2$ and $P = P \cup P_2$.

7. Solve the $d-1$ dimensional problem consisting of $V_1' \cup V_2'$, using the solution for dimension $d-1$, and update P and S according to what this solution returns: If it returns S_3 and P_3 then the updates are $S = S \cup S_3$ and $P = P \cup P_3$.

The space complexity (number of shortcut edges and dummy points) obeys the following recurrence. If $n > 1$ then:

$$f(n, 2) \leq c_1 n \log n$$

and, if $d > 2$, then

$$f(n, d) \leq 2f(n/2, d) + f(n, d-1) + c_2 dn$$

Note that this recurrence follows from steps 5 and 6 (which each recursively solve the problem for $n/2$ points in d dimensions), step 7 (which recursively solves a problem for n

points in $d-1$ dimension), and the other steps add at most $O(n)$ points and edges.

Now, if $n=1$ then $f(1,d)=c_3d$. Thus, the solution to the above recurrence is:

$$f(n,d)=O(dn(\log n)^{d-1}).$$

The w-to-v number of shortcut edges obeys the following recurrence. If $n>1$ then:

$$g(n,2)\leq 5$$

and, if $d>2$, then

$$g(n,d)\leq 2+g(n,d-1)$$

Note that the above recurrence follows from the following number of edges: one hop from V_2 to a shadow point, $g(n,d-1)$ hops on the $d-1$ dimensional hyperplane in step 7, and one hop from the shadow point to the destination point.

Now, if $n=1$ then $g(1,d)=1$. Thus, the solution to the above recurrence is:

$$g(n,d)\leq 2d+1.$$

6. EXTENSIONS: DYNAMIC HIERARCHIES

Dynamic changes to the hierarchy (such as addition and deletion of nodes and edges, as well as a node's key replacement) do not require wholesale re-keying, rather, only the nodes directly affected by the change need re-keying. To achieve this, the ideas of [2] can be applied.

At a high level, we use two ideas from [2]. First, each subject is given their own key (and a separate new node), and the public storage is modified to include an edge from that subject's node to its access class v_i. The other idea is that the actual key of a node is computed as a function of a label and a secret key. And it follows that, by changing the label, one can change a node's key. For more information, we refer the reader to [2].

However, while individual nodes do not need to be rekeyed using the above techniques, the public information (dummy nodes and shortcut edges) does need recomputing after the access graph is modified. Because of the divide and conquer recursive nature of the algorithm, this re-computation looks amenable (at least for the case $d=2$) to the techniques of dynamization of van Leeuwen and Overmars (see, e.g., [30, 20, 21]). This looks like a promising direction for future work.

7. CONCLUSIONS AND FUTURE RESEARCH

We gave the first scheme for key derivation in a non-tree hierarchy that performs any derivation in a small number of hash function computations and modular additions. This performance bound holds even for key derivations between nodes separated by a path of linear length.

Promising directions for future work include reducing the space complexity and improving the performance of dynamic changes to the graph.

8. REFERENCES

[1] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, September 1983.

[2] M. Atallah, K. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *ACM Conference on Computer and Communications Security (CCS'05)*, pages 190–201, 2005.

[3] C. Chang and D. Buehrer. Access control in a hierarchy using a one-way trapdoor function. *Computers and Mathematics with Applications*, 26(5):71–76, 1993.

[4] C. Chang, I. Lin, H. Tsai, H. Wang, and T. Taichung. A key assignment scheme for controlling access in partially ordered user hierarchies. In *International Conference on Advanced Information Networking and Application (AINA'04)*, 2004.

[5] T. Chen, Y. Chung, and C. Tian. A novel key management scheme for dynamic access control in a user hierarchy. In *IEEE Annual International Computer Software and Applications Conference (COMPSAC'04)*, pages 396–401, September 2004.

[6] G. Chick and S. Tavares. Flexible access control with master keys. In *Advances in Cryptology – CRYPTO'89*, volume 435 of *LNCS*, pages 316–322, 1990.

[7] H. Chien and J. Jan. New hierarchical assignment without public key cryptography. *Computers & Security*, 22(6):523–526, 2003.

[8] B. Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63:600–610, 1941.

[9] A. Ferrara and B. Masucci. An information-theoretic approach to the access control problem. In *Italian Conference on Theoretical Computer Science (ICTCS'03)*, volume 2841, pages 342–354, October 2003.

[10] L. Harn and H. Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, October 1990.

[11] M. He, P. Fan, F. Kaderali, and D. Yuan. Access key distribution scheme for level-based hierarchy. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'03)*, pages 942–945, August 2003.

[12] M. Hwang. An improvement of novel cryptographic key assignment scheme for dynamic access control in a hierarchy. *IEICE Trans. Fundamentals*, E82–A(2):548–550, March 1999.

[13] M. Hwang and W. Yang. Controlling access in large partially ordered hierarchies using cryptographic keys. *Journal of Systems and Software*, 67(2):99–107, August 2003.

[14] H. Liaw, S. Wang, and C. Lei. A dynamic cryptographic key assignment scheme in a tree structure. *Computers and Mathematics with Applications*, 25(6):109–114, 1993.

[15] C. Lin. Hierarchical key assignment without public-key cryptography. *Computers & Security*, 20(7):612–619, 2001.

[16] I. Lin, M. Hwang, and C. Chang. A new key assignment scheme for enforcing complicated access control policies in hierarchy. *Future Generation Computer Systems*, 19(4):457–462, 2003.

[17] S. MacKinnon, P. Taylor, H. Meijer, and S. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on*

Computers, 34(9):797–802, September 1985.

[18] P. Maheshwari. Enterprise application integration using a component-based architecture. In *IEEE Annual International Computer Software and Applications Conference (COMSAC'03)*, pages 557–563, 2003.

[19] K. Ohta, T. Okamoto, and K. Koyama. Membership authentication for hierarchical multigroups using the extended fiat-shamir scheme. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, pages 446–457, February 1991.

[20] M.H. Overmars and J. van Leeuwen. Dynamization of order decomposable set problems. *Journal of Algorithms*, 2(3):245–260, 1981.

[21] M.H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and Systems Science*, 23(2):166–204, 1981.

[22] I. Ray, I. Ray, and N. Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. In *ACM Symposium on Access Control Models and Technologies*, June 2002.

[23] J. Rose and J. Gasteiger. Hierarchical classification as an aid to database and hit-list browsing. In *International Conference on Information and Knowledge Management*, pages 408–414, 1994.

[24] R. Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 405–410, December 1987.

[25] R. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, January 1988.

[26] A. De Santis, A. Ferrara, and B. Masucci. Cryptographic key assignment schemes for any access control policy. *Information Processing Letters (IPL)*, 92(4):199–205, November 2004.

[27] W. Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.

[28] W.T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins University Press, Baltimore, MD, 1992.

[29] H. Tsai and C. Chang. A cryptographic implementation for dynamic access control in a user hierarchy. *Computers & Security*, 14(2):159–166, 1995.

[30] J. van Leeuwen and M.H. Overmars. The art of dynamizing. *Mathematical Foundations of Computer Science*, pages 121–131, 1981.

[31] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods*, 3:351–358, 1982.

[32] Y. Zheng, T. Hardjono, and J. Pieprzyk. Sibling intractable function families and their applications. In *Advances in Cryptology – AsiaCrypt'91*, LNCS, 1992.

[33] S. Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.