CERIAS Tech Report 2007-23

EFFICIENT TECHNIQUES FOR REALIZING GEO-SPATIAL ACCESS CONTROL

by Mikhail Atallah, Marina Blanton, and Keith Frikken

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Efficient Techniques for Realizing Geo-Spatial Access Control

Mikhail J. Atallah[*]
Department of Computer Science
Purdue University
mja@cs.purdue.edu

Marina Blanton[†]
Department of Computer Science
Purdue University
mbykova@cs.purdue.edu

Keith B. Frikken
Computer Science and Systems Analysis
Miami University
frikkekb@muohio.edu

ABSTRACT

The problem of key management for access control systems has been well-studied, and the literature contains several schemes for hierarchy-based and temporal-based access control. The problem of key management in such systems is how to assign keys to users such that each user is able to compute and have access to the appropriate resources while minimizing computation and storage requirements. In the current paper, we consider key management schemes for geo-spatial access control. That is, the access control policy assigns to a user a specific geographic area, and the user consequently obtains access to her area or information about it.

In this work, the geography is modeled as an $m \times n$ grid of cells (let $m \geq n$). Each cell has its own key associated with it, and a user who wants to access the content of a cell needs to obtain its key. Each user obtains access to a rectangular area (or a finite collection of such rectangles) and is able compute keys corresponding to the cells that comprise her area.

Our main result is an efficient scheme with the following properties: (i) each user obtains a small constant number of secret keys that permit access to an arbitrary rectangular sub-grid, (ii) computation to derive the key of a specific cell in that rectangle consists of a constant number of efficient operations, and (iii) the server needs to maintain $O(mn(\log\log m)^2 \log^* m)$ public information accessible to all users. The public storage requirement is the worst-case bound and can be improved if the grid is partitioned into regions where the cells of a region share the same key.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; E.1 [**Data**]: Data Structures—*graphs and networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems.

General Terms

Security, Algorithms, Design.

Keywords

Geo-spatial access control, key management, key assignment and derivation.

[*]Supported in part by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

[†]Supported by Intel Ph.D. fellowship.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASIACCS'07, March 20-22, 2007, Singapore.
Copyright 2007 ACM 1-59593-574-6/07/0003 ...$5.00.

1. INTRODUCTION

The problem of key management for access control systems has been well-studied. Previously, the main focus of key management research has been key management for hierarchical access control systems (see, e.g., [1, 3, 5, 10, 12, 13, 19, 28], for a small subset of this extensive literature), and its extension that allows to support time-based policies [6, 11, 14, 18, 23, 24, 26, 27]. In hierarchical access control, users are divided into disjoint access classes, and access rights of a class are a superset of the access rights of every descendant of that class in the hierarchy (as an example, consider hierarchically organized roles in RBAC models). In systems that additionally support temporal constraints, each user is given access rights for a specific time interval, in addition to having users organized in a hierarchy of classes.

A naive but simple solution to the key management problem in a hierarchy is to assign a key to every access class (i.e., each node in the hierarchy or each node in the hierarchy at each time unit), and to give a user the keys to all access classes that the user is entitled to access. Unfortunately, this solution requires each user to store a prohibitively large set of keys. To reduce this set of keys, the literature suggests using key derivation mechanisms. In schemes that use key derivation, each user is given a set of private keys, and the key of an access class can be derived from a user's private keys and public information (accessible to all users) if and only if that user is entitled to access that class. These key derivation schemes are benchmarked by several metrics, including: (i) the number of private keys each user must store, (ii) the number of keys assigned to each access class, (iii) the amount of public storage the server must maintain, and (iv) amount of computation it takes to derive a target key by a user.

In this paper, we introduce key assignment and derivation techniques for geo-spatial access control systems. We consider systems where each user is granted access rights to a specific area (or a set of areas). As this paper is a first step in addressing this problem, we consider the case where the user has access rights to a rectangular section of a larger grid. If a user's region is not rectangular, then it can be partitioned into a number of rectangles to each of which our technique is applicable.

We envision many applications of this work, including (but not limited to) the following scenarios:

1. Consider a physical facility that houses projects with different degrees of sensitivity/confidentiality, with each project assigned its own area. A specific employee might have access to certain areas of the building, but not to others. In this case, the users could be given a smartcard (or some other device) that can derive the access keys for the areas to which the user has access.

2. Consider a GIS that contains information (e.g., demographic, marketing, etc.) about specific locations. This information may be interesting to researchers, commercial firms, and other entities. Thus key management could be used to provide a subscription-based service where users purchase access rights to the information about a specific geographic area.

3. There could be a hybrid access control system based on not only location information, but also on role hierarchies, temporal constraints, or both. As an example, re-consider the first scenario above. It is a reasonable access control policy that a senior researcher might be able to access a specific room all of the time, but a consultant might be able to access the same room for two days. As our scheme extends to higher dimensions (see Section 10), it can be used in such hybrid frameworks (with time as an additional dimension).

The key derivation scheme we introduce in this paper for geo-spatial grids uses a novel data structure and achieves the following characteristics for an grid composed of $m \times n$ cells:

1. To obtain access to an arbitrary rectangular subsection, a user is required to store a constant number of keys.

2. Key derivation within the authorized rectangle involves a constant number of operations (including cryptographic operations).

3. The public storage space at the server due to our solution is only $O(mn(\log\log m)^2 \log^* m)$ with a small constant involved in the "$O(\cdot)$" notation.

4. All cryptographic operations are very efficient, and no expensive public-key cryptography is required.

The rest of this paper is organized as follows. Section 2 provides a brief review of related work. In Section 3, we give a formal problem definition, describe building blocks that are used later in this paper, and also provide a summary of our results. Section 4 introduces a basic (and inefficient) solution, which is later used as a part of our main scheme. In Section 5 we present efficient schemes for special cases, and we then describe, in Section 6, a significantly improved scheme for the general case. Section 7 shows how the space requirements our solution imposes can further be improved, and Section 8 states the security of our solution. Finally, a description of how dynamic updates are handled in given in Section 9, and Section 10 concludes the paper with extensions to higher dimensions.

2. RELATED WORK

Using location information for access control, i.e., location-based access control (LBAC), is not a new concept. The major challenges in geo-spatial computing were covered in the summary of the recent NRC's *IT roadmap to a geo-spatial future* [21]. One of the issues mentioned as a future challenge are "fine-grained access control mechanisms permitting the precise release of location information to just the right parties under the right circumstances." Atluri and Chun [7] propose a new model that supports privilege modes specific to geo-spatial data and includes geometric considerations (such at the region of overlap between an authorization and an access request). Similarly Bertino et al. [9] extend the RBAC model to GEO-RBAC, a model that can deal with geo-spatial information. Other previous work includes efficiently tracking the location of a user [17], other of models for representing and evaluating LBAC conditions [2], answering database queries based on location [20], the introduction of architectures for supporting location-aware applications [25], and many other important problems. However, we are not aware of any key management schemes that implement geo-spatial access control policies.

There has been a significant amount of work in key derivation for user hierarchies, and a thorough survey of this literature is beyond the scope of this paper. For an overview of such publications, see, e.g., [5] and [13]. In what follows, we give a brief summary of the results that are used as building blocks in this paper (a more detailed version can be found in Section 3.1). Atallah et al. [3, 5] use efficient key derivation techniques for a user hierarchy (in [3] the hierarchy is assumed to be a tree, and in [5] the techniques are extended to non-trees), where user classes are organized by a partial order into a hierarchy represented as a directed acyclic graph (DAG) G. No interaction with the server is assumed after the user obtains her secret key, which means that all necessary access keys are computed independently by the user. In the work of [3, 5], each user receives a single key, the server stores public information associated with the graph to aid the key derivation process, and computation of the access key for a class below the user class in the hierarchy consists of traversing the path from the user class to the target class. Very efficient key derivation is achieved through the addition of extra, so-called *shortcut*, edges to the hierarchy[1], such that the distance between any two nodes in the graph is minimized.

There are also many schemes that have been introduced for key management in temporal-based access control systems [6, 11, 14, 18, 23, 24, 26, 27]. In these temporal schemes a user is allowed certain access rights during a specific time interval. It is worth noting that the geo-spatial problem considered in this paper can be viewed as a higher-dimensional version of this temporal problem, with the possibility of having different granularity for individual cells. Several such temporal schemes [23, 18, 11, 27] are unsatisfactory and have been shown to be insecure with respect to collusion. How-

[1]While a shortcut edge is not in the original graph G, it is in the transitive closure of G.

ever, a very recent paper of Ateniese et al. [6] is the first work that formalizes the notion of security for time-based hierarchical key assignment schemes and presents provably secure solutions. Another recent work [4] introduces an alternative scheme that is provably secure under the same definitions of security as [6]. As we use the temporal schemes as a building block, we present more details about these schemes in Section 3.1.

3. BACKGROUND AND OVERVIEW OF RESULTS

In this section we review previous results that are used by the scheme we present in this paper. We also give a formal problem definition and outline the major contributions of this paper.

3.1 Background

3.1.1 Key derivation for graphs

In this work we use the key derivation of technique of [3], which we review next. This techniques will be used in conjunction with a novel data structure to achieve efficient key derivation in the geo-spatial grid setting. Such a key derivation technique works for any directed acyclic graph (DAG) $G = (V, E)$, where V is the set of nodes, and E is the set of edges. It consists of two algorithms: an algorithm to setup the system, Set, and an algorithm to derive a key, Derive. The Set algorithm assigns secret keys to the nodes of the graph and computes public information associated with its edges. The Derive algorithm, given a node v of G, its secret key, and another node w, computes the key of w using the public information about G as long as w is a descendant of v in the graph.

In what follows, $F : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$ denotes a family of pseudo-random functions (PRFs) that, on input a κ-bit key and a string, outputs a κ-bit string that is indistinguishable from random. Note that a PRF can be efficiently implemented using HMAC [8] or CBC MAC constructions.

The Set and Derive algorithms are then as follows:

Set$(1^\kappa, G)$: For each node $v \in V$, select a random secret key $k_v \in \{0,1\}^\kappa$. For each node $v \in V$, select a unique label $\ell_v \in \{0,1\}^\kappa$ and make it publicly available. For each edge $(v, w) \in E$, compute $y_{v,w} = k_w \oplus F(k_v, \ell_w)$, where $\oplus$ denotes bitwise XOR, and make it publicly available.

Derive(v, w, k_v): Let $(v, w) \in E$, i.e., v is a parent node of w. Then given k_v and the above-mentioned public information, derivation of the key k_w can be performed as $k_w = F(k_v, \ell_w) \oplus y_{v,w}$, where ℓ_w and $y_{v,w}$ are publicly available. More generally, if there is a directed path between nodes v and u in G, then u's key can be derived from v's key by considering each edge on the path.

In other words, there is a public label associated with every node in the graph, and there is a public information associated with every edge in the graph. This public information is what allows users to derive appropriate keys.

To avoid changing user keys when the hierarchy changes or when a certain class needs to be re-keyed (e.g., a user with certain privileges leaves the system), a slightly different version of the scheme should be used. We refer the reader to [3] for more details.

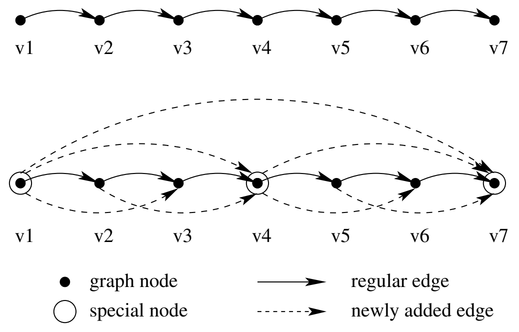

Figure 1: Illustration of building efficient data structure for one-dimensional graphs (original and modified graphs).

3.1.2 Efficient data structures for key derivation

In the current work we utilize an efficient data structure for key derivation in a partial order of dimension one or higher. Before reviewing prior results, we briefly define the notion of dimension in a hierarchy. 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., [15, 22]). That is, it is possible to associate with every node 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. and (iii) node v is ancestor of node w in G if and only if $x_{v,j} > x_{w,j}$ for every $1 \leq j \leq t$.

In [5], a data structure was introduced that allowed key derivation for such a one-dimensional partial order (i.e., total order) with n nodes that facilitated $O(n \log^* n)$ public storage, one key per user, and $O(1)$ key derivation time. For the purpose of the current discussion, assume that we are given a one-dimensional graph G, where the nodes are numbered according to their relationship with node v_i being the parent of node v_{i+1}. The main idea behind this construction is then that the (ordered) n nodes are partitioned into blocks of the same size, with the nodes on the boundary of blocks being "special" nodes. Within each block, each node has a direct edge to the first special node after it, and a direct edge from the nearest preceding special node to that node. The special nodes are well-connected, so that going from one block to another consists of following one or a small number of edges. Then reaching a node v_j from another node v_i in the graph (assuming that $j > i$) consists of reaching a special node from v_i, jumping to a special node close to v_j, and reaching v_j from that special node. Figure 1 shows a specific example of adding extra edges to a one-dimensional graph of 7 nodes.

This construction can be recursive, allowing for several levels of partitioning which affect the performance of the data structure. Note, however, that any edges that get inserted into the graphs (of dimension one or otherwise) do not modify the original relationship between the nodes. They simply allow to shorten the length of the path between nodes in the graph. The characteristics of the data structure described above (and which we use on one-dimensional graphs in this work) are: there is a path of 4 edges between any node and any of its descendants in the graph, and the space complexity of the data structure is $O(n \log^* n)$ for an n-node graph. Then the key derivation mechanism of the previous section can be applied to the data structure to achieve the performance of one private key per user, constant key derivation time, and $O(n \log^* n)$ public storage.

For graphs of dimension $d > 1$, an efficient data structure can also be built using dimension reduction techniques. That is, [5] shows that, given a solution to the one-dimensional graph, a data structure for a graph of dimension d will have an extra factor of $O((\log n)^{d-1})$ in the space complexity of the data structure, and extra $2(d-1)$ edges in the maximum distance between any nodes (between which a path exists) in the data structure. For instance, using the above solution of 4 edges between nodes and $O(n \log^* n)$ space complexity for one-dimensional graphs, for graphs of dimension $d = 4$ we obtain the distance of 10 edges between any node and its descendant and space complexity of $O(n(\log n)^3 \log^* n)$. Once again, applying the key derivation technique to this result gives us a key management mechanism with the performance corresponding to the data structure.

Finally, we also utilize the key derivation scheme of [4] for a contiguous set of time intervals for a single resource. In time-based key assignment schemes, the time is partitioned into short time intervals, and user keys change during each interval. A user who is entitled to obtain access to some resource during a period of time obtains secret keys corresponding to that resource for the duration of the time interval. Once again, to achieve efficiency, key derivation techniques are used. Specifically, the scheme of [4] allows a user to obtain access to an arbitrary contiguous set of intervals by storing a constant number of keys and performing a constant number of operations during key derivation. The public storage space requirement of the scheme is $O(n \log \log n \log^* n)$. This result is achieved through a non-trivial construction, which consists of building a half of a grid on the time intervals of the system and applying key derivation techniques for one-dimensional graphs to parts of it. Then this data structure is built for selected sets of time intervals at different levels of granularity through a recursive algorithm.

3.2 Problem description

The space consists of N cells in an $m \times n$ grid, $N = mn$. Without loss of generality, we assume that $m \geq n$ and that the grid has m rows and n columns. A user is permitted to subscribe and gain access to any sub-area within the grid. In general, user rights might permit access to areas of arbitrary shape (subject to the cell partitioning), which can be represented as a set of rectangles. Since the number of such rectangles in user access rights will be small in most applications, we will assume, for simplicity, that the user is given access to a single rectangular area R.

Then a grid cell will have an access key that permits access to the resources associated with that cell. This means that during the system initialization the grid cells will be assigned certain keys. Note that it will not always be the case that each cell has a unique key, because in some systems access to certain cells will always be granted in an all or none fashion (i.e., if a user is allowed access one cell in the group then that user can access all cells in the group), in which case such cells can share the same key.

When a user joins the system and obtains access to an area R, she will be given a key (or a set of keys) that permits access to every single cell within the area (through a key derivation process). The above means that the operation of the system requires algorithms to (i) setup the system, (ii) assign keys to users, and (iii) perform key derivation. Thus, from a user perspective, the interaction with the system consists of two phases:

(i) At the time of signing up, the user obtains secret keys that correspond to the area R to which access is being granted.

(ii) When the user would like to obtain access to a certain cell within R, she will use her secret keys (in combination with the public data made available by the server) to independently derive the access key for that cell. It is assumed that access to that key will permit her to either access the area or access information about the area, based on the context.

The security of a geo-spatial key assignment scheme is defined in a standard way. That is, we require the properties of completeness and soundness to hold, as defined below:

Completeness A user with access privileges to a rectangular area R is able to compute the access key for each cell within R.

Soundness Any coalition of users with access to rectangle areas $R_1, \ldots, R_k$ is unable to obtain access to any cell other than those contained in $R_1 \cup \cdots \cup R_k$.

As was mentioned above, the key assignment can be such that each cell obtains a unique key, but for efficiency reasons it might be advisable to assign the same key to multiple cells (when access to a certain area is always granted as whole and not at the level of individual cells). The fact that parts of the grid might have different access granularity (i.e., the level of individual cells versus the level of blocks of cells) will allow us to achieve significant savings in the data structure and key derivation time in certain systems. However, we initially consider the case where the mn cells have distinct access rights (hence there needs to be a separate key for each cell). Later in the paper, we discuss the case when groups of grid cells share a key (i.e., there are disjoint, arbitrarily shaped regions of the grid, and each region has its own key).

We also would like to note that this problem formulation easily supports multi-level security where access levels form a military-style ladder or a hierarchy. That is, suppose that, for instance, access to the Secret level permits access to a certain area R_S. Also suppose that access to the Top Secret level permit access to the area R_{TS}, such that R_S is contained within R_{TS}. In our problem formulation this will mean that a user with access rights to the Secret level obtains secret keys that allow her to obtain access to every cell of R_S. Similarly, a user with access rights to the Top Secret level obtains keys that allow her to obtain access to all cell comprising R_{TS} including every cell of R_S. Since the user with access to region R_{TS} will be able to derive keys of all cells of region R_S with the converse being false, a solution to our problem naturally provides a solution to the multi-level access control system.

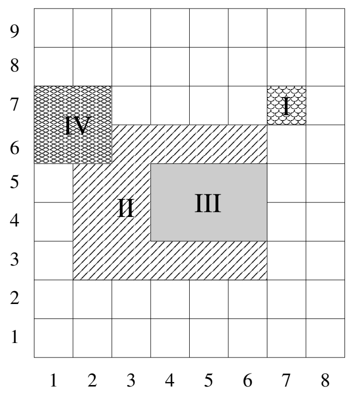

Figure 2: Illustration of regions on the grid with $m = 9$ and $n = 8$.

3.3 Our result

For an $m \times n$ grid of cells that have distinct keys, a user who is entitled to access a rectangular sub-grid R of such cells is given $O(1)$ private keys from which the key of any cell in R can be derived in $O(1)$ time. Key derivation uses only inexpensive cryptographic operations (pseudo-random functions). Moreover, given any such R, it is possible to compute the private keys for it in $O(1)$ time. The public storage space that the server needs to maintain is $O(mn(\log\log m)^2 \log^* m)$.

4. AN INEFFICIENT SCHEME

The papers [3, 5] describe schemes that are excellent for general graphs, but that yield inefficient solutions for spatial access control. This is because they fail to appropriately exploit the spatial structure. This section examines the solution that follows from [5], as it will be needed in the later more efficient schemes,

The result of [5] is an efficient key derivation mechanism (through building a data structure) for a graph of dimension d. In the geo-spatial domain, we can represent each rectangular area on the grid by the coordinates of its four corners. For each corner, we can use its coordinates to form a total order relationship, which for all of them results in four total orders. By incorporating all possible rectangular sub-areas of the grid, we obtain a graph of dimension $d = 4$ to which the techniques of [5] can be applied.

We denote a cell by its x and y coordinate. A rectangular region R within the grid is described by two x coordinates $a \le b$ and two y coordinates $c \le d$, i.e., by a 4-tuple representation (a, b, c, d) where (a, c) is R's bottom-left corner, and (b, d) is R's top-right corner. If R is a single cell, then $a = b$ and $c = d$. Such an R typically represents the subset of the $m \times n$ grid that a user is entitled to access (i.e., the user must be able to derive the key of every individual cell in R).

For instance, in Figure 2 Region I has coordinates (7, 7, 7, 7), Region II has coordinates (2, 6, 3, 6), Region III has coordinates (4, 6, 4, 5), etc.

We create an N^2 $(= m^2n^2)$ node graph G whose vertices correspond to all possible rectangles of the grid. That is, each vertex v in G is associated with a rectangle $R(v)$ whose bottom-left corner is $(a(v), c(v))$ and whose top-right corner is $(b(v), d(v))$. For reasons that will become apparent soon, we associate with such a vertex v the 4-tuple:

$$\tau(v) = (n - a(v), b(v), m - c(v), d(v)).$$

To illustrate this function on an example, we go back to Figure 2. Here if we associate node v_1 with Rectangle I, node v_2 with Rectangle II, and node v_3 with Rectangle III, then we have $\tau(v_1) = (1, 7, 2, 7)$, $\tau(v_2) = (6, 6, 6, 6)$ and $\tau(v_3) = (4, 6, 5, 5)$. Note that we have nodes in G and the corresponding values of the τ function for all possible rectangles on the grid.

Now observe that rectangle $R(v)$ contains rectangle $R(w)$ if and only if all 4 of the following inequalities hold:

$$a(v) \le a(w),\ b(w) \le b(v),\ c(v) \le c(w),\ d(w) \le d(v)$$

This is equivalent to:

$$m - a(v) \ge m - a(w),\ b(v) \ge b(w),$$
$$n - c(v) \ge n - c(w),\ d(v) \ge d(w)$$

which is the same as $\tau(v) \ge \tau(w)$. Hence rectangle $R(v)$ contains rectangle $R(w)$ if and only if $\tau(v) \ge \tau(w)$. This is also true when $R(w)$ is a single cell, i.e., when $a(w) = b(w)$ and $c(w) = d(w)$.

We now describe the edge set of G. There is an edge in G from vertex v to vertex w if and only if $\tau(v) \ge \tau(w)$, i.e., every one of the 4 components of the 4-tuple $\tau(v)$ is $\ge$ the corresponding component of $\tau(w)$. Using rectangles from Figure 2, there will be an edge from $R(v_2)$ to $R(v_3)$, but no edges between any other ordered pair of the four rectangles depicted in the figure.

Now we have a 4-dimensional partial order G in which it is desired that v can derive the key of w if and only if v precedes w in G. Thus, we can use the solution in [5] to solve the problem with performance of: 1 key, constant key derivation time, and $O(N^2(\log N)^3 \log^* N)$ space.

The following sections improve the performance of this data structure and culminate in a scheme that gets rid of the quadratic space while maintaining the constant number of keys and the constant key derivation performance.

5. SPECIAL CASES

Before presenting our scheme for the general case, we cover special cases. Solutions to these special cases will be used in the overall construction for the general case. The special cases considered in this section are rectangles that have less than 4 degrees of freedom, i.e., each of them shares one or more of its 4 sides with the boundary of the grid.

5.1 Rectangles that span the grid

Let us now consider rectangles that span the whole width or whole height of the $m \times n$ grid. This happens in one of two ways:

- *Vertical spanning*: The top and bottom boundaries of the rectangle are at rows 1 and (respectively) m.
- *Horizontal spanning*: The left and right boundaries of the rectangle are at columns 1 and (respectively) n.

Figure 3 illustrates such rectangles.

Given that users can only obtain access to rectangles that span the grid, our goal is to build a data structure that will permit a user to possess a small number of secret keys and derive access key to each cell of her region R. Without loss of generality, we give a solution for the case of horizontal

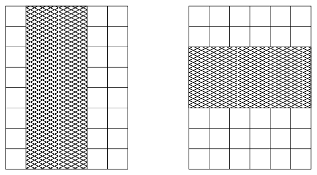
Figure 3: Illustration of rectangles that span the grid vertically and horizontally.

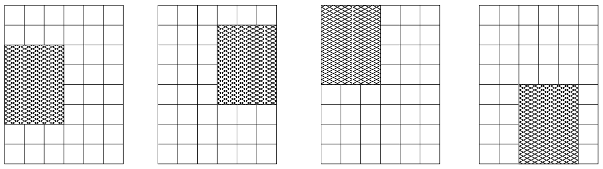
Figure 4: Illustration of rectangles that touch the grid's boundary.

spanning, and the case of vertical spanning can be addressed analogously.

In this special case, we can treat every row as a single "super-cell," ignoring the fact that it consists of many cells. Thus, we assign a key to each row, and this turns the problem into a problem with a single parameter. That is, now the only parameter that can change is the number of rows in user rectangle R. This makes it possible to apply the techniques of [4] to solve the problem. Recall that in [4] is user is allowed to obtain access to a contiguous set of time intervals with each interval having a different key. In the current discussion, we allow a user to obtain access to a contiguous set of rows with each row having a different key. In other words, now the m rows play the role the n time units played in [4]. Then a user with access to a rectangle that spans the grid obtains secret keys created according to the solution of [4].

The above solution allows us to obtain the data structure of size $O(m \log\log m \log^* m)$ with the distance between any node and its descendant being at most a constant number of edges. This translates into the following result: each user will need to store a constant number of secret keys, key derivation can be performed in constant time, and the server must maintain $O(m \log\log m \log^* m)$ public space. Note that none of the above characteristics depend on n, even though the grid is $m \times n$.

The case of vertical (rather than horizontal) spanning is treated similarly to the above, except that the roles of rows and columns are interchanged, as are the roles of m and n. Thus, the space complexity for vertical spanning is $O(n \log\log n \log^* n)$.

5.2 Rectangles that share a grid boundary

Next, we consider rectangles that share at least one of their boundaries with a grid's boundary. This means that at least one of the bounding four coordinates of each rectangle of this type is in the set $\{1, m, n\}$. Figure 4 shows examples of such rectangles.

We use "row" as an abbreviation for "y coordinate" and "column" as an abbreviation for "x coordinate." Without loss of generality, assume that the rectangle touches the right boundary of the grid, i.e., its right side is at column n as in the second grid from the left in Figure 4. We call such a problem *right-anchored*, and the other three cases are analogously referred to as *left-*, *top-*, or *bottom-anchored*.

Our solution to the right-anchored case consists of applying a solution to one-dimensional graphs from [5] to cells in each row (because key derivation is unidirectional in this case, i.e., from a certain point to the boundary). And we also apply the solution of [4] to each column (in this case, key derivation must be bounded by two points and key derivation is permitted within that interval only, just like in [4] for an interval of time). The algorithm for building the data structure is then as follows:

1. For each individual row i of the grid, we create a one-dimensional structure H_i that allows any position j in that row to have a short path to any other position j' of that same row iff $j < j'$.

2. For each individual column j of the grid, we use the single-parameter structure, call it V_j, of [4] to permit key derivation between any two positions i and i' ($i < i'$) in that column.

User Key Assignment and Derivation. Let a user be given access rights to a right-anchored rectangle R such that the leftmost column of R is j. Then the user is given the keys that correspond to the set of rows of R from V_j. Derivation of the key of the cell at location (i', j') within R consists of using the user's secret keys and the V_j structure to obtain the key for the cell (i', j) at the same column, and then $H_{i'}$ structure of that row to derive the target key for cell (i', j').

Performance. The data structure built in Step 1 of the above algorithm for a single row has the characteristics of one secret key per user, constant key derivation time, and the size of the data structure (public storage) of $O(n \log^* n)$ space. The total space for all rows is $O(mn \log^* n)$. The data structure built in Step 2 of the algorithm achieves, for a single column, a constant number of secret user keys, constant key derivation time, and $O(m \log\log m \log^* m)$ storage space for the data structure. The total public storage space for all columns is then $O(mn \log\log m \log^* m)$. This gives us the overall performance of a constant number of secret keys per user, constant key derivation time, and $O(mn \log\log m \log^* m)$ public storage space.

The case of left-anchored rectangles is handled very similar to the case of right-anchored rectangles. The only difference is in Step 1 of the algorithm, where the data structure built should permit key derivation from a cell at position j to a cell at position j' iff $j > j'$. For top-anchored and bottom-anchored rectangles, the solution will consists of the roles of rows and columns reversed in the original algorithm. That is, we apply the solution of [5] to each column, and the solution of [4] to each row.

6. THE GENERAL CASE: A PRELIMINARY SOLUTION

This section describes our preliminary scheme for the general case of rectangles R with no assumptions other than that they must be contained within the $m \times n$ grid (which we denote by S). In Section 7 we show how to lower the space complexity associated with the storage required for the data structure.

In this section, we first (in Section 6.1) describe how to build the data structure that permits efficient key derivation. This computation must be performed when the geo-spatial system is being setup. Then we show in Section 6.2 how, given a rectangle area R access to which is to be granted to a user, user secret keys are generated. This procedure is performed at the time a user joins the system and grants access privileges to access area R. Lastly, given access to R, we show in Section 6.3 how a user can obtain keys for a cell contained in R. It is assumed that access to such a key enables the user to obtain access to the area or information about the area, depending on the context.

6.1 The data structure

This section describes how, for a grid S, to build the public tree data structure, which we will use for key management. How it is used is covered in the next sections.

Without loss of generality, we assume $m = 2^{2^s}$ and $n = 2^{2^q}$, where $s \geq q$.[2] The algorithm for building the data structure takes, as inputs, a node v and an $m \times n$ grid S. It builds a tree for S rooted at v, using a recursive construction.

The idea behind it is to partition the grid into tiles of size $\sqrt{m} \times \sqrt{n}$ each, and apply the (inefficient) scheme of Section 4 on them treating each tile as a giant cell. Such a data structure will be able to handle key assignment and derivation at the granularity of tiles: now only rectangles that consist of a whole number of tiles are supported. Then the algorithm builds support for the special cases of Section 5 on the grid. In more detail, it builds data structures to support rectangles that border the boundary of the grid or span (vertically or horizontally) across one or more tiles. Finally, the algorithm is invoked recursively on each of the tiles to build the equivalent data structures at finer levels of granularity.

`Data_Struct_Build`(v, S)

1. If $n = 2$ (i.e., $q = 0$) then S consists of two columns of length m each. For each of these columns, create and store at node v the data structure for the m cells described in [4]. That is, we build a data structure that will permit key assignment and derivation for any contiguous set of cells within those m cells.

2. Partition S into a $\sqrt{m} \times \sqrt{n}$ array of *tiles* $S_{i,j}$, $1 \leq i \leq \sqrt{m}$ and $1 \leq j \leq \sqrt{n}$, where each tile is itself a $\sqrt{m} \times \sqrt{n}$ grid. That is, $S_{i,j}$ consists of the cells of S whose row number is in the interval $[(i-1)\sqrt{m} + 1, i\sqrt{m}]$ and whose column number is in the interval $[(j-1)\sqrt{n} + 1, j\sqrt{n}]$. Create a node $v_{i,j}$ for each $S_{i,j}$, and make $v_{i,j}$ a child of v.

3. Generate a grid $C(v)$, derived from S by treating each $S_{i,j}$ as a single cell (i.e., "merging" the constituents of cells of $S_{i,j}$ into a single cell). Note that $C(v)$ is $\sqrt{m} \times \sqrt{n}$.

4. Store at node v the scheme of Section 4 for $C(v)$, which we denote by $D(v)$. $D(v)$ will allow for user key assignment and derivation *at the granularity of tiles*. This means that $D(v)$ can process a rectangle only if that rectangle is the union of a subset of the $S_{i,j}$'s (i.e., it cannot handle rectangles whose corners are inside the $S_{i,j}$'s, as it cannot "see" inside an $S_{i,j}$).

5. Also store at node v a solution for each of the 4 "anchored" special cases (rectangles that have at least 1 side along a boundary for S). Call these structures $AL(v)$ for rectangles anchored at the left, $AR(v)$ for rectangles anchored at the right, $AT(v)$ for rectangles anchored at the top, and $AB(v)$ for rectangles anchored at the bottom. Having these structures enables the handling of anchored rectangles.

6. Let $HS_{i,j',j''}$, where $1 \leq i \leq \sqrt{m}$ and $1 \leq j' \leq j'' \leq \sqrt{n}$, be the horizontal slab consisting of the union of all of the tiles $S_{i,j'}, S_{i,j'+1}, \ldots, S_{i,j''}$. For every such $HS_{i,j',j''}$, we store at v a "horizontal spanning" structure for processing rectangles that horizontally span it. Since there are $\sqrt{m}$ choices for i and $\sqrt{n}$ choices for each of j', j'', the total number of such slabs is $n\sqrt{m}$. We denote by $H(v)$ the information stored at v for all of these $O(n\sqrt{m})$ horizontal slabs. The $H(v)$ can handle any rectangle that horizontally spans any one of those horizontal slabs.

7. Similarly, let $VS_{i',i'',j}$, $1 \leq i' \leq i'' \leq \sqrt{n}$, be the vertical slab consisting of the union of all of the tiles $S_{i',j}, S_{i'+1,j}, \ldots, S_{i'',j}$. For every such $VS_{i',i'',j}$, we store at v a "vertical spanning" structure for processing rectangles that vertically span it. The number of such slabs is $m\sqrt{n}$. We denote by $V(v)$ the information stored at v for all of these vertical slabs. The $V(v)$ can handle any rectangle that vertically spans any one of those vertical slabs.

8. Recursively apply the scheme to each child of v, that is, call `Data_Struct_Build`$(v_{i,j}, S_{i,j})$ for all $1 \leq i \leq \sqrt{m}$ and $1 \leq j \leq \sqrt{n}$.

[2]Note that these assumptions are for presentation purposes only, and our data structures can easily be generalized to other grids that are not of this form. That is, it is not necessary to increase the actual size of the grid to obtain m and n of the above form, but instead rounding can be used in computing partitions of the grid.

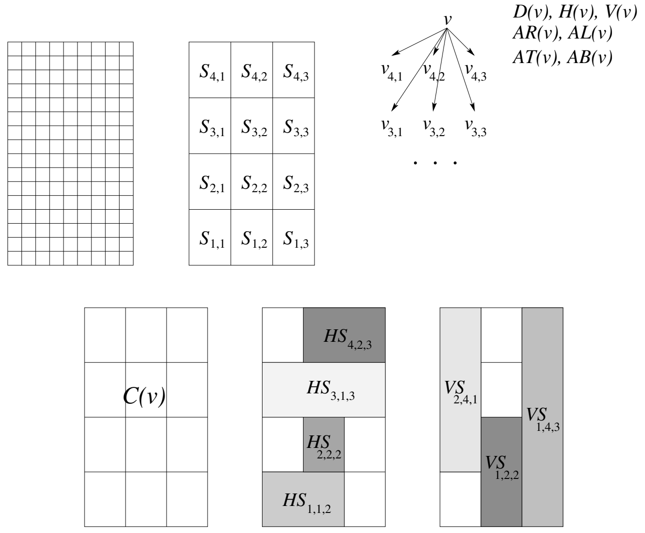
$S_{4,1}$ $S_{4,2}$ $S_{4,3}$
$S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$
v
$v_{4,1}$ $v_{4,2}$ $v_{4,3}$
$v_{3,1}$ $v_{3,2}$ $v_{3,3}$
. . .
D(v), H(v), V(v)
AR(v), AL(v)
AT(v), AB(v)
C(v)
$HS_{4,2,3}$
$HS_{3,1,3}$
$HS_{2,2,2}$
$HS_{1,1,2}$
$VS_{2,4,1}$
$VS_{1,4,3}$
$VS_{1,2,2}$

Figure 5: Illustration of building the data structure for a grid 16×9, the first level of recursion.

Now we analyze the performance of the data structure built. In Step 1, we obtain a construction of $O(m \log\log m \log^* m)$ space, constant distance between nodes, and a constant number of keys per user. In Step 4, the data structure built has the space complexity of $O(mn(\log m)^3 \log^* m)$ with constant distance between nodes and one key per user. In Step 5, the construction gives us the space complexity of $O(mn \log\log m \log^* m)$ with a constant distance between nodes and a constant number of user secret keys.

In Step 6, we have that each structure $HS_{i,j',j''}$ has space complexity of $O(\sqrt{m} \log\log m \log^* m)$, a constant number of keys per user, and a constant distance between nodes. Since there are $\sqrt{m}$ choices for i and $\sqrt{n}$ choices for each of j', j'', the total space for all such structures is $O(mn \log\log m \log^* m)$. Finally, in Step 7, each structure $VS_{i',i'',j}$ has space complexity of $O(\sqrt{n} \log\log n \log^* n)$, a constant distance between nodes, and a constant number of keys per user. Since there are $\sqrt{n}$ choices for j and $\sqrt{m}$ choices for each of i', i'', the total space for all such structures is $O(nm \log\log n \log^* n)$.

Since Step 1 is at the bottom of the recursion, the total space satisfies the following recurrence:

$$f(m,n) \leq \sqrt{mn} f(\sqrt{m}, \sqrt{n}) + c_1 mn(\log m)^3 \log^* m$$

if $n > 2$, and

$$f(m,2) = c_2 m \log\log m \log^* m$$

where c_1 and c_2 are constants. The solution is $f(m,n) = O(mn(\log m)^3 \log\log m \log^* m)$.

The above "augmented tree" data structure (call it G) is used to set up the system, i.e., we associate with each node of G a key, and we create public information associated with each edge (v,w) in G that allows someone with v's key to derive w's key in one simple step.

In addition to the above data structure, the server will need to maintain public information associated with another, simple graph that maps keys corresponding to tiles to cell keys. A description of such an auxiliary data structure is given in Section 6.3 where we explain how key derivation is performed.

6.2 Key assignment

We now turn our attention to describing how keys are assigned to a user who is being granted access to a rectangle area of cells R.

In what follows, v is the root node of the above tree data structure, S is the $m \times n$ grid associated with v (with $m \geq n$), and R is an arbitrary rectangle in S. Although our ultimate goal is to achieve constant time in computing the key assignment of any such R, we begin with describing a key assignment algorithm that does so in $O(\log\log n)$ time.

6.2.1 *Sub-optimal key assignment*

Given a user's rectangle R, the recursive procedure below returns a constant-size set of secret user keys that will permit access to R. The algorithm majorly follows the data structure build using `Data_Struct_Build` to find the largest blocks of cells, keys for which are encoded in the data structure.

`Assign_Keys`(R, v, S)

1. If v is a leaf node, it stores data structures for a small sub-area of size $m \times n$ where $n = 2$. More precisely, it stores two solutions to the single-parameter problem of m cells (one for each of the two columns). If R consists of a single column, we retrieve from the data structure corresponding to that column the keys that permit access to the range of R's rows. If R consists of two columns, we retrieve such keys from both data structures. Return the (constant number of) keys computed.

   If v is not a leaf, continue with the next step.

2. Recall from the above data structure construction algorithm that the $v_{i,j}$'s are the children of v, and that $S_{i,j}$ is the $\sqrt{m} \times \sqrt{n}$ tile associated with node $v_{i,j}$. We distinguish different cases, based on how R overlaps with the $S_{i,j}$'s. They are also depicted in Figure 6.

   (a) If R overlaps with only one $S_{i,j}$, then we recursively call `Assign_Keys`$(R, v_{i,j}, S_{i,j})$ and return the keys returned by that recursive call. Otherwise, we continue with the next steps, in which different pieces of R are handled separately depending on how they intersect with the $S_{i,j}$'s.

   (b) Let R_1 be the maximal sub-rectangle of R that consists of the union of one or more $S_{i,j}$'s. If no such R_1 exists, then continue to the next step. Otherwise, obtain the key for R_1 from the $D(v)$ structure stored at v, in constant time (by indexing into the $D(v)$ structure).

   (c) Let R_2 be any of the (at most 4) maximal sub-rectangles of R that (i) are disjoint from R_1, and (ii) horizontally or vertically span one of the slabs $HS_{i,j',j''}$ or $VS_{i',i'',j}$. There can be at most 2 such horizontally spanning R_2's (a top one and a bottom one), and at most 2 vertically spanning R_2's (a left one and a right one). For each of (at most 4) such R_2's we obtain the $O(1)$ keys by using the $H(v)$ or $V(v)$ structure stored at v, in constant time.

   This and the previous steps have considered all but the "corners" of R, each of which could lie

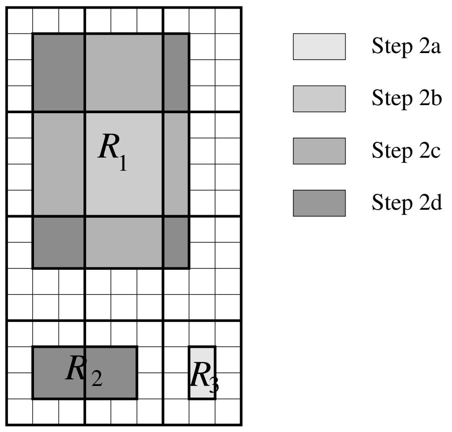

Figure 6: Illustration of the key assignment procedure for various user rectangles. The color of an area indicates where in the `Assign_Keys` algorithm the area is addressed.

inside an $S_{i,j}$. These are considered in the next step.

(d) Let R_3 be any of the (at most 4) maximal subrectangles of R that contain a corner of R and are disjoint from R_1 and the R_2's of the previous two steps. Note that there could be fewer than four such R_3's, as such an R_3 could contain 2 corners of R (when both corners lie within the same $S_{i,j}$). Each such R_3 is surely "anchored", therefore the $O(1)$ keys for it can be obtained in constant time from one of the four structures $AL(v)$, $AR(v)$, $AT(v)$, $AB(v)$. That is, we first index to the appropriate V_j data structure for left- and right-anchored rectangles (and H_j for top- and bottom-anchored rectangles) and then retrieve the right keys from it in constant time (as in [4]).

(e) Return the $O(1)$ keys computed in the previous three steps.

The above procedure assigns a constant number of keys per rectangle R and does so in $O(\log\log n)$ time. In the next subsection we sketch a modification that brings the time down to $O(1)$.

As can be seen from above, Step 2b of the `Assign_Keys` procedure returns from $D(v)$ keys associated with tiles, but not with individual cells. When users, however, want to obtain access to cells, they will need to have keys associated with those cells. For that reason, the server must maintain a mapping from tile keys to the keys that compose the corresponding tiles. The data structure that allows such mapping is explained in Section 6.3 along with the key derivation process.

6.2.2 *Constant-time key assignment*

What is preventing the above `Assign_Keys` procedure from working in constant time is the fact that we are going down the tree G (working with blocks of finer granularity) until we find the node u at which R overlaps with more than one $S_{i,j}$. To achieve $O(1)$ time performance for key assignment, we need to find this u is constant time. This can be done as follows.

1. Let cells $\alpha, \beta, \gamma, \theta$ be the four corners of R. For every $\lambda \in \{\alpha, \beta, \gamma, \theta\}$, let $\ell(\lambda)$ denote the leaf of G that contains λ.

2. Use the constant-time algorithm for computing nearest common ancestors (NCA) in a tree [16] to compute the lowest (i.e., farthest from the root) node of G that is ancestor of all of $\ell(\alpha)$, $\ell(\beta)$, $\ell(\gamma)$, $\ell(\theta)$. That node is the u we seek.

The above computation of u clearly takes $O(1)$ time. We prove its correctness by contradiction. Suppose, to the contrary, that the u returned does not have the desired property, i.e., that at that node u the rectangle R overlaps with only one of the $S_{i,j}$ (Step 2a of `Assign_Keys`). In that case, the corresponding child $v_{i,j}$ is an ancestor of all of $\ell(\alpha)$, $\ell(\beta)$, $\ell(\gamma)$, $\ell(\theta)$, thereby contradicting the fact that u is their NCA.

6.3 Constant-time key derivation

The above key assignment process could also be used to guide the processing of a key derivation request. But before we proceed with sketching it, we describe an auxiliary data structure, G', that will allow users to map keys associated with tiles to cell keys. Given the augmented tree G, G' is constructed as follows:

1. Starting with the root node v of G and going down the tree, add to the set of nodes of G' a node associated with each tile in $C(v)$.

2. Add to the set of nodes of G' a node for each cell of S.

3. For each node of G' that corresponds to a tile, insert an edge from it to every cell contained in that tile.

Given the above graph G', we assign a fresh unique public label to every node of it as in the key derivation method given in Section 3.1.1. We then use the corresponding secret keys from G to compute public information associated with G'. Now each user who obtains a key for a tile in S will be able to use the public information for G' to obtain the key for any cell within that tile.

The space complexity of G' is $O(mn \log\log m)$, which is lower than that of G.

Going back to the key derivation procedure at large, we have that a user with secret keys for an area R should be able to obtain the key of a cell within R using the public information associated with the augmented tree G and the additional graph G' above. To do so, the user locates (in constant time using the NCA algorithm, as in the previous sub-section) the node u at which R overlaps with more than one $S_{i,j}$. Having u, the user derives the key depending on whether the target grid cell is in an R_1 (Step 2b), an R_2 (Step 2c), or an R_3 (Step 2d). All that is needed to carry out the constant-time derivation of the target cell's key is the local information stored at node u, i.e., $AL(u)$, $AR(u)$, $AT(u)$, $AB(u)$, $H(u)$, $V(u)$, or data structures stored at leaves. The only exception is the case when the key is returned from $D(u)$ in Step 2b. In this case, such a key will correspond to a tile, but not to an individual cell. This means that the user will need to refer to G' to compute the cell's key from the tile key obtained above (by following one edge in G').

An alternative way to the use of NCA computations in locating u would be to provide the user with access rights to R with a pointer to the node u, thereby allowing constant access to that node whenever that user needs to do key derivation.

7. IMPROVING THE SPACE COMPLEXITY

We now give an improvement in the space complexity of the scheme. The improved scheme looks just like the preliminary scheme of Section 6, except that in Step 4 of `Data_Struct_Build`(v, S), instead of using the inefficient scheme of Section 4 for $C(v)$, it uses the better scheme of Section 6. This implies that the space for Step 4 of `Data_Struct_Build` goes down to $\sqrt{m}\sqrt{n}(\log m)^3 \log\log m \log^* m$, which is dominated by the $O(mn \log\log m \log^* m)$ space for other structures $AL(v)$, $AR(v)$, $AT(v)$, $AB(v)$, $H(v)$, and $V(v)$. The recurrence for the total space thus becomes:

$$f(m,n) \le \sqrt{mn} f(\sqrt{m}, \sqrt{n}) + c_1 mn \log\log m \log^* m$$

if $n > 2$, and

$$f(m,2) = c_2 m \log\log m \log^* m$$

where c_1 and c_2 are constants. The solution is $f(m, n) = O(mn(\log\log m)^2 \log^* m)$.

As was mentioned earlier, in practice it is quite likely that not every cell has its own distinct access key, and that groups of cells may have the same key. The easiest way to exploit this structure is for the recursive construction of the G structure to stop as soon as its corresponding sub-grid consists of cells that all have the same key. That is, Step 1 of `Data_Struct_Build` needs to contain a termination test for when all of the mn cells share the same key (in which case it stops even if $n > 2$). This is likely to result in less space that the worst-case theoretical bound we proved, especially when the cells that share a key tend to be contiguous (but not when they form a checkered pattern). We can quantify the improvement if we make assumptions about the shapes of those sub-regions of cells that share the same key (e.g., assume a rectangular shape), but we do not include these analyses in this document (they will be given in the journal version of this paper).

8. SECURITY

To show the security of our schemes, we must show that, given a user's keys, the user: (i) can generate all keys for his designated region (completeness) and (ii) cannot generate a key outside of his region (soundness). Given the security properties of the key derivation method of [3] (which can be applied to any DAG and proven to be secure) and the above properties, this scheme is secure even against collusion of multiple users. We omit the detailed proof (which is more tedious than difficult) of this claim, but it will be given in the full version of the paper.

9. HANDLING UPDATES

It was already described earlier how keys are issued to a new user, therefore this section focuses on what happens when: (i) a cell's key is modified or (ii) a user's access rights are revoked. These issues were considered in [3] for hierarchical access control systems, and the same basic techniques work for the geo-spatial problem considered in this paper.

One concept used in [3] is to use a level of indirection for the keys. That is, the system creates two keys for every key in the system. The first key, a system key, is assigned as before and the data structure is built exactly as the scheme described in this work. The second key, a user key, is assigned a random value, and the public information is amended so that when given a user key it is possible to derive the corresponding system key (i.e., by adding an edge between the two keys' nodes). Furthermore, the system keys are the "used keys" in the system (e.g., the encryption keys for content). The advantage of this approach is that when the system wants to change a specific key, it needs only to change the system key and the public information associated with that key, and thus does not need to re-key any users.

This approach avoids having to re-key every user who shares access to a cell with the revoked user: Only if they share a key is there a need for re-keying. On the other hand, in some environments, re-keying even a single user is expensive (or is simply not possible). In such environments it is possible (again using techniques of [3]) to not require rekeying of any user for revocation. The basic idea of this approach is that each user is given their own node in the public structure, and an edge is added in the public structure from the user's node to the nodes containing keys for that user. Now one can change the keys for the underlying structure and can update the user's keys by modifying only public information. An added benefit of this approach is that each user needs to store only a single key. Note that this benefit comes as a cost: the public structure now grows linearly as the number of users increases.

10. EXTENSIONS

The scheme we gave extends to higher dimensions: Every additional dimension causes an additional $\log\log N$ factor in the space complexity, an extra constant number steps in key derivation, and a multiplicative constant factor in the number of keys. Therefore for a dimension d problem, we obtain:

(i) The number of keys is $O(c^d)$ for some constant c (and thus is only efficiently applicable when the number of dimensions is small);

(ii) The key derivation time becomes $O(d)$;

(iii) The space complexity becomes $O(N(\log\log N)^d \log^* N)$.

11. 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, Sept. 1983.

[2] C. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Supporting location-based conditions in access control policies. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'06)*, pages 212–222, 2006.

[3] M. Atallah, M. Blanton, N. Fazio, and K. Frikken. Dynamic and efficient key management for access hierarchies. Preliminary version appeared in *ACM Conference on Computer and Communications Security (CCS'05)*, full version is available as *Technical Report TR 2006-09, CERIAS, Purdue University*, 2006.

[4] M. Atallah, M. Blanton, and K. Frikken. An efficient and provably-secure time-based key assignment scheme. Under submission, 2006.

[5] M. Atallah, M. Blanton, and K. Frikken. Key management for non-tree access hierarchies. In *ACM Symposium on Access Control Models and Technologies (SACMAT'06), Full version available at* `http://www.cs.purdue.edu/homes/mbykova/papers/key-derivation.pdf`, pages 11–18, 2006.

[6] G. Ateniese, A. De Santis, A. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. In *ACM Conference on Computer and Communications Security (CCS'06)*, 2006. Full version is available as Cryptology ePrint Archive Report 2006/255, `http://eprint.iacr.org/2006/225`.

[7] V. Atluri and S. Chun. An authorization model for geospatial data. *IEEE Transactions on Dependable and Secure Computing*, 1(4):238–254, 2004.

[8] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO'96*, volume 1109, 1996.

[9] E. Bertino, B. Catania, M. Damiani, and P. Perlasca. GEO-RBAC: a spatially aware RBAC. In *ACM Symposium on Access Control Models and Technologies (SACMAT'06)*, pages 29–37, 2005.

[10] 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, Sept. 2004.

[11] H. Chien. Efficient time-bound hierarchical key assignment scheme. *IEEE Transactions of Knowledge and Data Engineering (TKDE)*, 16(10):1301–1304, 2004.

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

[13] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.

[14] A. De Santis, A. Ferrara, and B. Masucci. Enforcing the security of a time-bound hierarchical key assignment scheme. *Information Sciences*, 176(12):1684–1694, 2006.

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

[16] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338–355, 1984.

[17] H. Hu and D. L. Lee. Energy-efficient monitoring of spatial predicates over moving objects. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 28(3):19–26, 2005.

[18] H. Huang and C. Chang. A new cryptographic key assignment scheme with time-constraint access control in a hierarchy. *Computer Standards & Interfaces*, 26:159–166, 2004.

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

[20] M. Mokbel, W. Aref, S. Hambrusch, and S. Prabhakar. Towards scalable location-aware services: requirements and research issues. In *ACM International Symposium on Advances in Geographic Information Systems (GIS'03)*, pages 110–117, 2003.

[21] C. Patterson, R. Muntz, and C. Pancake. Challenges in location-aware computing. *IEEE Pervasive Computing*, 2(2):80–89, 2003.

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

[23] W. Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(1):182–188, 2002.

[24] W. Tzeng. A secure system for data access based on anonymous authentication and time-dependent hierarchical keys. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'06)*, pages 223–230, 2006.

[25] U. Varshney. Location management for mobile commerce applications in wireless internet environment. *ACM Transactions on Internet Technology (TOIT)*, 3(3):236–255, 2003.

[26] S.-Y. Wang and C.-S. Laih. Merging: an efficient solution for a time-bound hierarchical key assignment scheme. *IEEE Transactions on Dependable and Secure Computing*, 3(1):91–100, 2006.

[27] J. Yeh. An RSA-based time-bound hierarchical key assignment scheme for electronic article subscription. In *ACM International Conference on Information and Knowledge Management (CIKM'05)*, pages 285–286, 2005.

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