CERIAS Tech Report 2007-99

Incorporating Temporal Capabilities in Existing Key Management Schemes

by Mikhail J. Atallah
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Incorporating Temporal Capabilities in Existing Key Management Schemes

Mikhail J. Atallah[1], Marina Blanton[2], and Keith B. Frikken[3]

[1] Department of Computer Science, Purdue University
mja@cs.purdue.edu
[2] Department of Computer Science and Engineering, University of Notre Dame
mblanton@cse.nd.edu
[3] Department of Computer Science and Systems Analysis, Miami University
frikkekb@muohio.edu

Abstract. The problem of key management in access hierarchies studies ways to assign keys to users and classes such that each user, after receiving her secret key(s), is able to *independently* compute access keys for (and thus obtain access to) the appropriate resources defined by the hierarchical structure. If user privileges additionally are time-based, the key(s) a user receives should permit access to the resources only at the appropriate times. This paper presents a new, provably secure, and efficient solution that can be used to add time-based capabilities to existing hierarchical schemes. It achieves the following performance bounds: (i) to be able to obtain access to an arbitrary contiguous set of time intervals, a user is required to store at most 3 keys; (ii) the keys for a user can be computed by the system in constant time; (iii) key derivation by the user within the authorized time intervals involves a small constant number of inexpensive cryptographic operations; and (iv) if the total number of time intervals in the system is n, then the server needs to maintain public storage larger than n by only a small asymptotic factor, e.g., $O(\log^* n \log\log n)$ with a small constant.

1 Introduction

This work addresses the problem of key management in access control systems, with the emphasis on time-based access control policies. Consider a system where all users are divided into a set of disjoint classes, and a user is granted access to a specific access class for a period of time specified by its beginning and end. In such systems, it is common for the access classes to be organized in a hierarchy, and a user obtains access to the resources at her own class and the resources associated with all descendant classes in the hierarchy. When a user joins the system and is granted access to a certain class for a specific duration of time, she is given a key (or a set of keys) which allows her to *independently* derive access keys for all resources she is entitled to have access during her time interval. For hierarchically organized user classes this means that the key allows the user to access objects at her access class and all descendant classes in the hierarchy during the time interval specified. Note that the time interval is user-specific and might be different for each user in the system.

There is a wide range of applications that follow this model and which would benefit from automatic enforcement of access policies through efficient key management.

Such applications include (among others) role-based access control (RBAC) models, subscription-based services, content distribution, and cryptographic directories or file systems. In all of these examples we use the current time to enforce time-based policies. Additionally, instead of being based on the current time, access control policies can be based on the time in the past and permit access to historical data. For example, a user might buy access to data such as historical transactions, prices, legal records, etc. for a specified time interval in the past, e.g., the year of 1920. These different notions of time can be combined, e.g., a user buys access to 1920 data and is entitled to access it for two weeks starting from today.

If we let the lifetime of a system be partitioned into n short time intervals, the existence of time-based access control policies requires the access keys to be changed during each time interval. In this work, we concentrate on applications where the system is setup to support a large number of such time intervals. For example, access key to a video stream might change at least once a day (thus, permitting users to subscribe on any given day). If the system is setup for a few years, this results in n being in thousands. Likewise, if the application of interest is access to historical data, say, for the last century, the number of time intervals will tend to be even higher. Thus, a small number of keys per user and efficient access with large n's is the goal of this work.

The notion of security for time-based hierarchical key assignment (KA) schemes was formalized only recently by Ateniese et al. [5]. Thus, in the current paper we use their security definitions and provide a new efficient solution to the problem of key management in systems with time-based access control policies. The approach we propose is provably secure and relies only on the security of pseudo-random functions (PRFs). In addition, our solution does not impose any requirements or constraints on the mechanisms used to enforce policies in systems where access control is not time-based (e.g., for a hierarchy of user classes). This means that our solution can be built on top of an existing scheme to make it capable of handling time. In the rest of this paper, we refer to a scheme without the support for temporal access control as a *time-invariant* scheme, and we refer to a scheme that supports temporal access control policies as *time-based*.

Existing efficient time-invariant key management schemes for user hierarchies are based on the notion of key derivation: a user receives a single key, and all other access keys a user might need to possess according to her privileges can be derived from that key. In the most general formulation of the problem, inheritance of privileges is modeled through the use of a directed graph, where a node corresponds to a class and a parent node can derive the keys of its descendants. In this paper we follow the same model, but, unlike previous work, apply key derivation techniques to time.

In a setup with n time intervals, the server is likely to maintain information linear in n. By building a novel data structure, we only slightly increase the storage space at the server beyond the necessary $O(n)$ and at the same time are able to achieve other attractive characteristics. In more detail, our solution enjoys the following properties:

- To be able to obtain access to an arbitrary contiguous set of time intervals, a user is required to store at most 3 keys.
- The above-mentioned keys to be given to a user can be computed in constant time from that user's authorized set of contiguous time intervals.

– Key derivation within the authorized time intervals involves a small constant number of cryptographic operations and thus is independent of the number of time intervals in the systems or the number of time intervals in the user's access rights.
– If the total number of time intervals in the system is n, then the increase of the public storage space at the server due to our solution is only by a small asymptotic factor, e.g., $O(\log^* n \log\log n)$ with a small constant.
– All operations are very efficient, and no expensive public-key cryptography is used.

We provide several solutions with slightly different characteristics, where the difference is due to the building blocks used in our construction. These solutions are summarized in Table 3. An extension of our techniques also allows to support access rights that can be stated as periodic expressions.

While the results given above correspond to a time-based key assignment scheme with a single resource or user class, we can use them to construct a time-based key assignment scheme for a user hierarchy. We show that our construction favorably compares to existing schemes and provides an efficient solution to the problem (the comparison is given at the end of the paper in Section 7). Additionally, our scheme is balanced in the sense that all resource consumption such as the client's private storage, computation to derive keys, and the server public storage are minimized with tradeoffs being possible. This allows the scheme to work even with very weak clients and not to burden the server with excessive storage. Furthermore, our scheme is provably secure under standard complexity assumptions.

In the rest of the paper, we first review related literature in Section 2. In Section 3 we define the model and give some preliminaries. Section 4 gives a preliminary data structure, which we use in Section 5 to build our improved scheme. Thus, the core of our solution lies in Section 5 along with its analysis. In Section 6 we show how to use the scheme to build a time-based key assignment scheme for a user hierarchy. Finally, Section 7 compares our solution with other existing schemes and concludes. Several extensions of our scheme and security proofs can be found in [4].

2 Related Work

The literature on time-invariant key assignment (KA) schemes in a user hierarchy is extensive, and its survey is beyond the scope of this paper. For an overview of such publications, see, e.g., [2] and [11].

While the list of publications on time-invariant KA schemes is very large, the number of publications that consider time-based policies and provide schemes for them is rather modest. The time-based setting and the first scheme was introduced by Tzeng [17]. The scheme, however, was later shown to be insecure against collusion of multiple users [22]. Subsequent work of Huang and Chang [12], Chien [10], and Yeh [20] was also shown to be insecure against collusion (in [16], [21, 14], and [5], respectively).

Among very recent publications, Wang and Laih [19] present a time-based hierarchical KA scheme. While their scheme is shown to be collusion-resilient, the notion of security, however, is not formalized and no clear adversarial model is given in that work. Tzeng [18] also describes a time-based hierarchical key assignment scheme, which is used as a part of an anonymous subscription system. The scheme is proven to resist collusion attacks; however, no formal model of adversarial behavior is provided. The work of Ateniese at el. [5] is the first result that provides a formal framework for time-based hierarchical KA schemes and gives provably secure solutions, both secure against key recovery and with pseudo-random keys. Concurrently with and independently from this work, time-based solutions have been developed by De Santis et al. [15]. Section 7 compares all solutions.

There is extensive literature on broadcast encryption and multicast security, which might be considered applicable here. There are, however, crucial differences in the models, which prevent us from using solutions from those domains. First, broadcast encryption and multicast security schemes permit access to a single resource instead of a hierarchy and cannot be composed in an obvious way to solve our problem. More importantly, they assume that each client obtains key updates for each time interval, which is impossible in our model: no private channels between the server and a client after the initial issuance of the user keys is assumed, the client is allowed to remain off-line, and can access the resources at her own discretion. The only exception from the above online requirement that we are aware of is the work of Briscoe on multicast key management [9]. That solution builds a binary tree from the time intervals, thus achieving $O(\log n)$ secret keys and $O(\log n)$ key derivation time.

Finally, the access control literature has a large body of work on temporal access control models (see, e.g., [7, 8]). These models, however, concentrate on policy specification and not on key assignment and derivation mechanisms.

3 Problem Description and Preliminaries

3.1 The model

While the motivation for this work comes from the need to support access control policies with temporal constraints in user hierarchies, the problem does not need to be limited to this particular setting. That is, an efficient solution to the key management problem in temporal access control can find use in other domains. Therefore, we provide a very general formulation of the problem, without any assumptions on the environment in which it is used. Of course, access control in user hierarchies remains the most immediate and important application of our techniques. Thus, in Section 6 we will show how our solution can be used to realize temporal access control for user hierarchies.

Now let us assume that we are given a resource, and the owner of this resource would like to control user access to that resource using time-based policies. For that purpose, the lifetime of the system is partitioned into short time intervals (normally, of a length of a day or shorter), and the access key for that resource changes every time interval. Let n denote the number of time intervals in the system, $T = \{t_1, \ldots, t_n\}$ denote the intervals, and $K = \{k_{t_1}, \ldots, k_{t_n}\}$ denote the corresponding access keys.

Now assume that a user $\mathcal{U}$ is authorized to access that resource during a contiguous set of time intervals $T_{\mathcal{U}} \subseteq T$, where $T_{\mathcal{U}} = \{t_{start}, \ldots, t_{end}\}$. Following the notation of [5], we use the *interval-set* over T, denoted by $\mathcal{P}$, which is the set of all non-empty contiguous subsequences of T, i.e., $T_{\mathcal{U}} \in \mathcal{P}$ for any $T_{\mathcal{U}}$. With such access rights, $\mathcal{U}$ should receive or should be able to compute the keys $K_{T_{\mathcal{U}}} \subseteq K$, where for each $t \in$

$T_{\mathcal{U}}$ the key $k_t \in K_{T_{\mathcal{U}}}$. We denote the private information that $\mathcal{U}$ receives by $S_{T_{\mathcal{U}}}$. Obviously, storing $|T_{\mathcal{U}}|$ keys at the user end is not always practical, and significantly more efficient solutions are possible. Then a *time-based key assignment scheme* assigns keys to the time intervals and users, so that time-based access control is enforced in a correct and efficient manner. Such key generation is assumed to be performed by a central authority CA, but once a user is issued the keys, there is no interaction with other entities. More formally, we define a time-based KA scheme as follows:

Definition 1. *Let T be a set of distinct time intervals and $\mathcal{P}$ be the interval-set over T. A time-based key assignment scheme consists of algorithms* $(\mathsf{Gen}, \mathsf{Assign}, \mathsf{Derive})$ *s.t.:*

Gen *is a probabilistic algorithm, which, on input a security parameter 1^κ and the set of time intervals T, outputs (i) a key k_t for any $t \in T$; (ii) secret information* Sec *associated with the system; and (iii) public information* Pub*. Let $(K, \mathsf{Sec}, \mathsf{Pub})$ denote the output of this algorithm, where K is the set of all keys.*

Assign *is a deterministic algorithm, which, on input a time sequence $T_{\mathcal{U}} \in \mathcal{P}$ and secret information* Sec*, outputs private information $S_{T_{\mathcal{U}}}$ for $T_{\mathcal{U}}$.*

Derive *is a deterministic algorithm, which, on input a time sequence $T_{\mathcal{U}}$, time interval $t \in T_{\mathcal{U}}$, private information $S_{T_{\mathcal{U}}}$, and public information* Pub*, outputs the key k_t for time interval t. The correctness requirement is such that, for each time sequence $T_{\mathcal{U}} \in \mathcal{P}$, each time interval $t \in T_{\mathcal{U}}$, each private information $S_{T_{\mathcal{U}}}$, each key $k_t \in K$, and each public information* Pub *that* $\mathsf{Gen}(1^\kappa, T)$ *and* $\mathsf{Assign}(T_{\mathcal{U}}, \mathsf{Sec})$ *can output,* $\Pr[\mathsf{Derive}(T_{\mathcal{U}}, t, S_{T_{\mathcal{U}}}, \mathsf{Pub}) = k_t] = 1$.

Note that in many cases the Assign algorithm can be a part of the Gen algorithm, i.e., private values $S_{T_{\mathcal{U}}}$ for every $T_{\mathcal{U}} \in \mathcal{P}$ are generated at the system initialization time. We, however, separate these algorithms to account for cases where retrieving $S_{T_{\mathcal{U}}}$ from Sec is not straightforward (which is the case in our scheme). In such cases, merging these two algorithms together will needlessly complicate Gen.

Also note that since a user accesses the server's public storage for key derivation purposes, there is no need for additional time synchronization mechanisms between the user and the server: the current time interval can be stored as a part of the public information the server maintains.

We distinguish between two different notions of security for a time-based KA scheme: security against *key recovery* and security with respect to *key indistinguishability* (i.e., schemes with pseudo-random keys). A time-based KA scheme can also be secure against static or adaptive adversaries. In [5], however, it was shown that the security of a time-based hierarchical KA scheme against a static adversary is polynomial-time equivalent to the security of that scheme against an adaptive adversary for both security goals (key recovery and key indistinguishability). While in the current discussion we are not concerned with hierarchical schemes, our setting can be considered to be a special case of a hierarchy with a single class. Thus, in this work we only provide definitions of a time-based KA scheme secure against a static adversary; and a proof of security under such definitions will imply security against an adaptive adversary.

In our definition of a scheme secure against static adversary, let adversary $\mathcal{A}_{st}$ attack the security of the scheme at time $t \in T$. $\mathcal{A}_{st}$ is allowed to corrupt all users with no access to k_t and, when finished, is asked to guess k_t. We consider a scheme to be secure only if $\mathcal{A}_{st}$ has at most negligible probability in outputting the correct key.

In addition to the security requirements, an efficient KA scheme is evaluated by the following criteria: (i) The size of the private data a user must store; (ii) The time it takes the system to assign a user its keys; (iii) The amount of computation necessary for a user to generate an access key for the target time interval; and (iv) The amount of information the service provider must maintain for public access.

3.2 Key derivation

Our approach relies heavily on the notion of key derivation. In our solution, we use the same key derivation techniques that were used in [1]. The crucial difference, however, is that in [1] key derivation was used between user classes (to provide a time-invariant scheme for a user hierarchy), while in this work we use key derivation for the data structures that we build. This is possible because the techniques of [1] work for an arbitrary directed acyclic graph (DAG), and we review them next.

Assume that we are given a DAG denoted by $G = (V, E)$, where V is the set of nodes and E is the set of edges. Let $Anc(v, G)$ denote the set of ancestors of node v in G including v itself, and let $Desc(v, G)$ denote the set of descendants of v in G including v itself. Let $F^\kappa : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$, for a security parameter κ, be a family of pseudo-random functions (PRFs) that, on input of a κ-bit key and a string, outputs a κ-bit string that is indistinguishable from a random string (note that a PRF can be implemented very efficiently as HMAC [6] or CBC MAC). For brevity, instead of $F^\kappa(k, x)$, we may write $F_k(x)$. Also, when the graph G is clear from the context, we may omit it in the ancestry functions and use $Anc(v)$ and $Desc(v)$.

To be able to derive keys, we need two algorithms:

- Set is an algorithm for assigning keys to the graph which takes as input a security parameter 1^κ and a DAG $G = (V, E)$ and outputs (i) an access key k_v for each $v \in V$, (ii) secret information S_v for each $v \in V$, and (iii) public information Pub.
- Derive is an algorithm for deriving keys which takes as input nodes $v, w \in V$, secret information S_v for v, and public information Pub. It outputs the access key k_w for w, if $w \in Desc(v, G)$.

The derivation method we use is from [1], and is sufficient to achieve security against key recovery:

- $\mathsf{Set}(1^\kappa, G)$: For each node $v \in V$, select a random secret key $k_v \in \{0,1\}^\kappa$ and set $S_v = k_v$. For each node $v \in V$, select a unique public label $\ell_v \in \{0,1\}^\kappa$ and store it in Pub. For each edge $(v, w) \in E$, compute public information $y_{v,w} = k_w \oplus F_{k_v}(\ell_w)$, where $\oplus$ denotes bitwise XOR, and store it in Pub.
- $\mathsf{Derive}(v, w, S_v, \mathsf{Pub})$: Let $(v, w) \in E$. Given $S_v = k_v$ and Pub, derivation of 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 in Pub. More generally, if there is a directed path between nodes v and u in G, u's key can be derived from v's key by considering each edge on the path.

3.3 Shortcut techniques

Our constructions use the so-called shortcut edges: a *shortcut edge* is an edge that is not in the original graph G but is in the transitive closure of G. Such edges are added to

Scheme	Private storage	Key derivation	Public storage
2HS [2]	1	2 op.	$O(n \log n)$
3HS [1]	1	3 op.	$O(n \log \log n)$
4HS [2]	1	4 op.	$O(n \log^* n)$
$\log^*$HS [2]	1	$O(\log^* n)$ op.	$O(n)$

Table 1. Performance of shortcut schemes for one-dimensional graphs.

G for performance reasons. Note that addition of shortcut edges does not affect partial order relationship between the nodes, i.e., we may add a shortcut edge (v, w) to the graph only if there is already a directed path from node v to w in the original graph.

In this work we rely on efficient shortcut techniques from prior literature for a graph of dimension 1 (i.e., a total order), reviewed in [4]. Here we only summarize the performance of existing schemes, any of which can be used as a building block in our constructions. Consider a directed graph of dimension 1 consisting of n vertices. The performance of known solutions for such graphs is given in Table 1. In the table, we denote by sHS a solution where the distance between any two nodes (i.e., the diameter of the graph) is at most s, i.e., a so-called s-Hop Scheme.

Throughout this work we may use $\mathcal{S}1(n)$ to denote any shortcut scheme for graphs of dimension 1 applied to a total order of size n. We also use $space(\mathcal{S}1(n))$ and $time(\mathcal{S}1(n))$ to denote its public storage and key derivation complexity, respectively.

4 Building Basic Data Structure

As was mentioned above, all of our constructions are based on the notion of key derivation in a graph. Throughout the rest of the paper, when we say that there is a directed edge from v to w in G, it implies that v is capable of deriving w's key using its own key. This means that, for the data structures that we build (all of which are DAGs), there will be a public and secret information associated with each node, and there will be public information corresponding to each edge.

Our preliminary data structure is rather simple and consists of two main steps: building a grid of size $n \times n$ (where n is the number of time intervals in the system) and applying one-dimensional shortcut techniques to parts of the grid. A more detailed description follows.

1. Build half of a grid of dimension $n \times n$ with the time intervals $t_1, \ldots, t_n$ being on its diagonal (see Figure 1). In the grid, we denote by $v_{1,1}$ the root node; node $v_{i,j}$ is located at the row i and column j (i.e., $v_{2,1}$ is "below" $v_{1,1}$ and $v_{1,2}$ is "on the left" of $v_{1,1}$). There is a directed edge from each $v_{i,j}$ to $v_{i+1,j}$, and from each $v_{i,j}$ to $v_{i,j+1}$. The time interval t_i corresponds to the node $v_{i,n-i}$.
   From this data structure it should be clear that, given a key for $v_{i,j}$, all keys for time intervals $t_i, \ldots, t_{n-j+1}$ can be derived from it (in the worst-case $O(n)$ time).
2. Next, we apply a one-dimensional shortcut scheme $\mathcal{S}1$ to each row and column of the grid (see Figure 2). More precisely, we add shortcuts to the data structure to be able to derive $v_{i,x}$'s key from $v_{i,y}$'s key for any $x > y$ (and similarly $v_{x,j}$'s key from $v_{y,j}$'s key for any $x > y$) in a small number of steps instead of previous $O(n)$

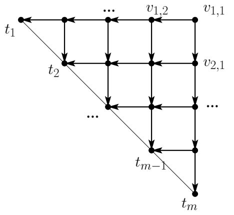


Fig. 1. Building a grid for the basic scheme.

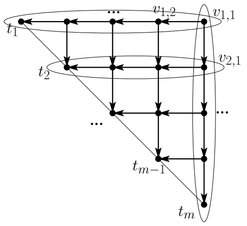


Fig. 2. Adding shortcuts to the grid.

Underlying scheme	Private storage	Key derivation	Public storage
2HS	1	≤ 4 op.	$O(n^2 \log n)$
3HS	1	≤ 6 op.	$O(n^2 \log \log n)$
4HS	1	≤ 8 op.	$O(n^2 \log^* n)$
$\log^*$HS	1	$O(\log^* n)$ op.	$O(n^2)$

Table 2. Performance of the basic (and preliminary) scheme.

time. This is done at the expense of $O(space(\mathcal{S}1(n)))$ additional shortcuts per row or column and therefore $O(n \cdot space(\mathcal{S}1(n)))$ total shortcuts.

Having this, now a user entitled to have access during time intervals $T_{\mathcal{U}} = \{t_x, \ldots, t_y\} \in \mathcal{P}$ can receive a single key corresponding to node $v_{x,n-y+1}$. Key derivation of the key corresponding to the current time interval $t_i \in T_{\mathcal{U}}$ now consists of at most $2 \cdot time(\mathcal{S}1(n))$ steps: at most $time(\mathcal{S}1(n))$ steps are needed to derive $v_{i,n-y+1}$'s key from that of $v_{x,n-y+1}$, and then at most $time(\mathcal{S}1(n))$ steps are needed to derive $v_{i,n-i+1}$'s key (which corresponds to t_i) from that of $v_{i,n-y+1}$.

Table 2 summarizes the performance of the basic scheme, when used with various one-dimensional schemes.

5 An Improved Scheme

This section describes a solution that achieves significantly better performance than the previous scheme. We first present a new data structure and then fill other parts in to provide a full-fledged time-based KA scheme.

At a high level, to build a new data structure, we partition all time intervals in the system into coarse "chunks" ($\sqrt{n}$ chunks of $\sqrt{n}$ time intervals each) and apply the basic scheme to the chunks. If access is to be granted to a large time interval that spans across boundaries of these chunks, we can use this level of granularity to assign keys. If, on the other hand, the interval to which the user should obtain access is contained within a chunk, we recursively apply this procedure to the time intervals within each chunk to support time-based access control of finer granularity. If a time interval spans across different chunks, but contains partial chunks at the beginning and at the end of the user's sequence of time intervals, then we utilize the coarse chunk's keys along with two new types of keys that are introduced later.

5.1 Reducing storage space

This section describes the tree data structure we build; how it is used is covered in the next sections. For the purposes of presentation of this work, we let $n = 2^{2^q}$ for some integer q. This allows us to avoid using rounding notation $\lfloor x \rfloor$ and $\lceil x \rceil$ throughout the algorithms and results in a cleaner presentation (note that this assumption is purely to make the presentation cleaner, and the solution will work without this assumption). Our procedure for building the data structure takes as inputs a node v and the set $T = \{t_1, \ldots, t_n\}$, and then recursively builds a tree for the set rooted at v. Due to the recursive nature of this function, we use $\hat{T}$ to denote the working set of the current function invocation and $|\hat{T}|$ to denote the size of $\hat{T}$. Then the data structure is constructed as described below:

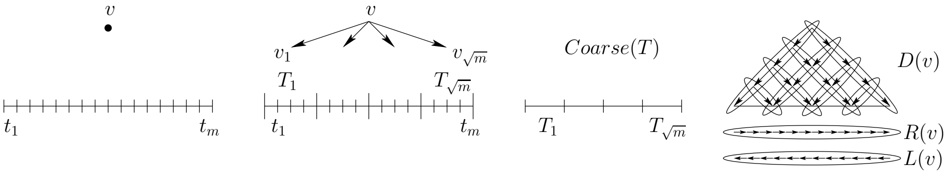
Algorithm $\mathsf{DataStructBuild}(v, \hat{T})$:

1. If $|\hat{T}| = 2$ (i.e., $q = 0$), then return. Otherwise, continue with the steps below.
2. Partition $\hat{T}$ into $\sqrt{|\hat{T}|}$ sets of $\sqrt{|\hat{T}|}$ contiguous time intervals each, call these $\hat{T}_1, \ldots, \hat{T}_{\sqrt{|\hat{T}|}}$. That is, if $\hat{T} = \{t_1, \ldots, t_{|\hat{T}|}\}$, then $\hat{T}_i = \{t_{i\sqrt{|\hat{T}|}+1}, \ldots, t_{i\sqrt{|\hat{T}|}+\sqrt{|\hat{T}|}}\}$. Create a node v_i for each $\hat{T}_i$, and make v_i a child of v.
3. Generate a problem $Coarse(\hat{T})$, derived from $\hat{T}$ by treating each $\hat{T}_i$ as a black box (i.e., "merging" the constituents of $\hat{T}_i$ into a single item). Note that the size of set $Coarse(\hat{T})$ is $\sqrt{|\hat{T}|}$.
4. Store at node v an instance of the basic scheme for $Coarse(\hat{T})$, denoted $D(v)$. $D(v)$ supports performance of: 1 key, $O(time(\mathcal{S}1(|\hat{T}|)))$ key derivation, and $O(space(\mathcal{S}1(|\hat{T}|)))$ space; but $D(v)$ can only process an interval if it is the union of a contiguous subset of $Coarse(\hat{T})$ (i.e., it cannot handle intervals whose endpoints are inside the $\hat{T}_i$'s, as it cannot "see" inside a $\hat{T}_i$).
5. Also store at node v two solutions of one-dimensional problems on $\hat{T}$: One is for intervals all of which start at the right boundary of $\hat{T}$ and end inside $\hat{T}$ (we call this the *right-anchored* problem and denote the one-dimensional structure for it by $R(v)$); another is for intervals all of which start at the left boundary of $\hat{T}$ and end inside $\hat{T}$ (we call this the *left-anchored* problem and denote the one-dimensional structure for it by $L(v)$). Note that having $R(v)$ and $L(v)$ enables the handling of an interval that lies within $\hat{T}$ and also has its left or right endpoint at a boundary of $\hat{T}$, with performance of: 1 key, $O(time(\mathcal{S}1(|\hat{T}|)))$ steps per key derivation, and $O(space(\mathcal{S}1(|\hat{T}|)))$ space.
6. Recursively apply the scheme to each child of $\hat{T}$; that is, call $\mathsf{DataStructBuild}(v_i, \hat{T}_i)$ in turn for each $i = 1, 2, \ldots, \sqrt{|\hat{T}|}$.

Figure 3 gives an illustration of how the data structure is built. The total space of the data structure satisfies the recurrence $S(n) \leq \sqrt{n}S(\sqrt{n}) + c_1 \cdot space(\mathcal{S}1(n))$ if $n > 2$ and $S(2) = c_2$, where c_1 and c_2 are constants. Thus, $S(n) = O(space(\mathcal{S}1(n)) \log\log n)$.

5.2 Key assignment

We now turn our attention to which keys are given to a user with access to an arbitrary $T_{\mathcal{U}} \in \mathcal{P}$. In what follows, v is a node of the above tree data structure, $\hat{T}$ is the set of



(a) Initial state. (b) State after Step 2. (c) State after Step 3. (d) State after Step 5.

Fig. 3. Construction of the data structure for the improved scheme (first level of recursion).

time intervals associated with v, and I is a sequence of time intervals for which the keys must be given. The recursive procedure below, when invoked on any $T_{\mathcal{U}}$ and our data structure, returns a set of (at most 3) keys associated with $T_{\mathcal{U}}$.

```
Algorithm AssignKeys($I, v, \hat{T}$):
1. If $v$ is a leaf, then return a key for each of the (at most two) time intervals in $I$.
   Otherwise, continue with the next step.
2. Let $v_1, \ldots, v_{\sqrt{|\hat{T}|}}$ be the children of $v$, and let $\hat{T}_1, \ldots, \hat{T}_{\sqrt{|\hat{T}|}}$ be the respective sets
   of times associated with these children. We distinguish two cases:
   (a) $I$ overlaps with only one set $\hat{T}_i$. Then we return the keys from the recursive call
       AssignKeys($I, v_i, \hat{T}_i$).
   (b) $I$ overlaps with all of $\hat{T}_k, \hat{T}_{k+1}, \ldots, \hat{T}_{k+\ell}$, where $\ell \geq 1$. These $\ell + 1$ intervals
       are handled in 3 different ways: Those completely contained in $I$ are collectively
       processed using the $D(v)$ structure, resulting in one key. If $\hat{T}_k$ overlaps
       with $I$, but is not contained in $I$, then it is right-anchored and is processed using
       $R(v_k)$, resulting in one key. If $\hat{T}_{k+\ell}$ overlaps with $I$, but is not contained in
       $I$, then it is left-anchored and is processed using $L(v_{k+\ell})$, resulting in one key.
       Those (at most) 3 keys are returned.
```

One can also lower the time complexity of the above algorithm to $O(time(\mathcal{S}1(n)))$ (e.g., it can be constant). We show how to achieve this in [4].

All keys given to users must be labeled with the level at which they were retrieved in the data structure, i.e., the distance from the root node. This is necessary for achieving constant-time computation of access keys, which will be explained in the next section. To make key derivation simpler, we also label user keys with their type; namely: D, R, or L. In addition, if a user receives more than a single key for her time sequence $T_{\mathcal{U}}$, each key is labeled with a range of time intervals to which it permits access.

To summarize, we assume that a key given to a user will be labeled with four values $(lev, type, t_a, t_b)$, where $0 \leq lev \leq \log\log n$, $type \in \{R, L, D\}$, and $t_a, t_b \in T$ such that $t_a < t_b$. For example, if a user with access rights to $T_{\mathcal{U}} = \{t_{start}, \ldots, t_{end}\}$ is given private information consisting of three keys $S_{T_{\mathcal{U}}} = \{k_1, k_2, k_3\}$, then k_1 could be labeled with (l, R, t_{start}, t_a), k_2 with $(l-1, D, t_{a+1}, t_b)$, and k_3 with (l, L, t_{b+1}, t_{end}).

5.3 Content distribution

At time $t \in T$, the service provider wants to make certain content (possibly very voluminous) available to the users with access rights at time interval t. To do so, the content is encrypted with the access key k_t using a symmetric encryption scheme and is made available to all users in the encrypted form (by placing it in a public location, broadcasting it to the users, or by other means). In our scheme the server also needs to ensure that the keys that users derive for t allow them to derive k_t. There are $O(\log\log n)$ such keys for t in the data structure access to which should allow access to k_t. Since the data structure has $(\log\log n + 1)$ levels, such keys are:

- Keys from $R(v)$, for some v in the data structure, one from each level.
- Keys from $L(v)$, similarly, for a single v per level.
- Keys corresponding to $D(v)$, one from each level l, $0 \leq l \leq \log\log n - 1$.

We refer to these keys as *enabling keys*. The server places in the public domain information that permits derivation of k_t from any of the enabling keys above. Additionally, the server labels the public derivation information associated with each of the enabling keys with the level and the type (i.e., R, L, or D) of the corresponding enabling key. This is needed to permit fast constant-time derivation of the access key.

5.4 Key derivation

A user $\mathcal{U}$ with access to the sequence of time intervals $T_{\mathcal{U}} = \{t_{start}, \ldots, t_{end}\} \in \mathcal{P}$ receives private information $S_{T_{\mathcal{U}}}$ consisting of 1, 2, or 3 keys that permit her to derive enabling keys for each $t \in T_{\mathcal{U}}$. In the most general (and common) case, such private information consists of 3 keys – denoted by k_1, k_2, and k_3 – labeled as (l, R, t_{start}, t_a), $(l-1, D, t_{a+1}, t_b)$, and (l, L, t_{b+1}, t_{end}), respectively, for some l, a, and b. Let us assume, without loss of generality, that if the number of keys is less than 3, then the missing keys are set to empty strings with k_1 remaining of type R, key k_2 of type D, and key k_3 of type L. Then to obtain the enabling key for a time interval $t_i \in T_{\mathcal{U}}$, $\mathcal{U}$ executes a derivation algorithm which we sketch here:

```
Algorithm DeriveKey($T_{\mathcal{U}}, t_i, S_{T_{\mathcal{U}}}$, Pub):
 1. Parse $S_{T_{\mathcal{U}}}$ as $k_1(l, R, t_{start}, t_a)$, $k_2(l-1, D, t_{a+1}, t_b)$, $k_3(l, L, t_{b+1}, t_{end})$.
 2. If $t_i \in \{t_{start}, \ldots, t_a\}$, find the node $v$ at level $l$ such that $R(v)$ permits access to $t_i$
    (note that such node $v$ can be computed in constant time using index $i$ of the time
    interval $t_i$). Use $k_1$ and the public information about the edges in Pub to derive the
    key corresponding to $t_i$ and return that enabling key.
 3. Similarly, if $t_i \in \{t_{b+1}, \ldots, t_{end}\}$, locate the node $v$ at level $l$ s.t. $L(v)$ permits
    access to $t_i$. Use $k_3$ and Pub to derive an enabling key for $t_i$ and return that key.
 4. Finally, if $t_i \in \{t_{a+1}, t_b\}$, locate $v$ at level $l-1$ such that $D(v)$ permits access to
    $t_i$; use $k_2$ and Pub to derive an enabling key for $t_i$ and return it.
```

Key derivation complexity in all of the above cases is $O(time(\mathcal{S}1(n)))$.

5.5 Putting everything together

In this section we summarize our construction and show its performance. All proofs corresponding to our security theorems can be found in [4]. Figure 4 gives a complete description of our time-based KA scheme. In addition to the algorithms given in previous sections, we specify how they are used. Table 3 summarizes performance of our

```
Algorithm Gen$(1^\kappa, T)$:
  1. Create a root node $root$ for the data structure and run DataStuctBuild$(root, T)$. Let $G = (V, E)$ denote the tree structure returned.
  2. For each $v \in V$, randomly choose a secret key $k_w \in \{0,1\}^\kappa$ and a unique public label $\ell_w \in \{0,1\}^\kappa$ associated with each node $w$ in $D(v)$, $R(v)$, and $L(v)$.
  3. For each $v \in V$, construct public information for each edge in $D(v)$, $R(v)$, and $L(v)$ using the key derivation method, e.g., for an edge $(w, u)$, its public value is $y_{w,u} \in \{0,1\}^\kappa$.
  4. For each $t \in T$, randomly choose a secret key $k_t \in \{0,1\}^\kappa$ and a unique public label $\ell_t \in \{0,1\}^\kappa$.
  5. For each $t \in T$, let $V_t \subset V$ denote the set of nodes in $G$ access to which implies access to $t$. Then for each $V_t$, for each $v \in V_t$:
     (a) find in $D(v)$ the node corresponding to the time interval $t$; call it $w$.
     (b) create an edge from $w$ to $t$ by computing public information using enabling key $k_w$, $t$'s secret key $k_t$, public label $\ell_t$, and the key derivation method. Mark such an edge with the level of $v$ and type $D$.
     (c) repeat (a) and (b) for $R(v)$ and $L(v)$, using types $R$ and $L$, respectively.
  6. Let $K$ consist of the secret keys $k_t$ for each $t \in T$ and Sec consist of the remaining secret keys $k_w$. Also let Pub consist of $G$, all public labels (of the form $\ell_w$ and $\ell_t$), and public information about all edges generated above.

Algorithm Assign$(T_\mathcal{U}, \mathsf{Sec})$:
  1. Execute AssignKeys$(T_\mathcal{U}, root, T)$, where $root$ is the root node of $G$.
  2. Set $S_{T_\mathcal{U}}$ to the keys computed and return $S_{T_\mathcal{U}}$.

Algorithm Derive$(T_\mathcal{U}, t, S_{T_\mathcal{U}}, \mathsf{Pub})$:
  1. If $t \notin T_\mathcal{U}$, return a special rejection symbol $\perp$.
  2. Execute DeriveKey$(T_\mathcal{U}, t, S_{T_\mathcal{U}}, \mathsf{Pub})$ to compute an enabling key for $t$; call it $k'_t$.
  3. Use $k'_t$ along with its (level-type) label and Pub to derive key $k_t$.
```

Fig. 4. Proposed time-based key assignment scheme.

solution. The security of our solution comes from the way key derivation is performed in a DAG and is not due to the details of the data structures built.

Theorem 1. *Assuming the security of the family of PRFs F^κ, the time-based key assignment scheme given in Figure 4 is both complete and sound with respect to key recovery in the presence of a static adversary.*

To achieve a stronger notion of key indistinguishability, our solution will require a slightly different key derivation method. Intuitively, we decouple the keys used in the public information from the actual access keys, so that now it is not feasible to test access keys using the public information. The separation is performed using an additional invocation of a PRF, where the keys to be used in Pub are computed as $F(0||k)$ and the access keys are computed as $F(1||k)$. This key derivation method is described in [1] (full version only).

Then in our scheme of Figure 4, we use this enhanced key derivation method in Step 3 of the Gen algorithm (i.e., in data structures $D(v)$, $R(v)$, and $L(v)$). This means that now someone with access to a certain key in, for instance, $R(v)$ and who guesses an unauthorized key correctly, cannot use the public information for that data structure to test the key. This change implies the corresponding change in the Derive algorithm.

Underlying scheme	Private storage	Key derivation	Public storage
2HS	≤ 3	≤ 5 op.	$O(n \log n \log \log n)$
3HS	≤ 3	≤ 7 op.	$O(n(\log \log n)^2)$
4HS	≤ 3	≤ 9 op.	$O(n \log^* n \log \log n)$
$\log^*$HS	≤ 3	$O(\log^* n)$ op.	$O(n \log \log n)$

Table 3. Performance of the improved scheme.

So far we devised a solution to support access rights that span across a contiguous sequence of intervals. It is also possible to support periodic access rights that span across a contiguous set of time periods but the time intervals themselves might be discontinuous within a period. If we treat time as a single dimension and the solution presented in this work as a solution to one-dimensional problem, it is possible to extend our approach to higher dimensions. An extension to dimension 2, which is useful in the geo-spatial context, is presented in [3]. This two-dimensional solution can be used to conveniently address the problem of periodic access rights with a small number of keys per user: we use one dimension to specify periods in user access rights and the other dimension to specify individual time slots within a period. We omit further details here.

Full version [4] gives extensions to this solution. In particular, we show how to extend the lifetime of the system beyond the original n time intervals and how to generalize the scheme to further decrease the public space using a key-space tradeoff.

6 Temporal Access Control for a User Hierarchy

In systems with hierarchically organized access classes, such a hierarchy is normally modeled as a directed acyclic access graph which we denote by G_U. In such a graph, each node corresponds to an access class and the edges form a partial order relationship between the classes. An edge from node v to node w means that the parent node v inherits privileges of the node w (while the converse is not true). This implies that a user with access to a specific class obtains access to the resources at that class and the resources at all of the descendant classes in the hierarchy. With this setup, it is possible to assign each class a single secret key and let users obtain keys of their descendant classes through a key derivation process. Similar to a general graph, in an access graph G_U a directed path from node v to w means that w's keys are derivable from v's key.

Now if we equip the model with time-based policies, in addition to computing keys of descendant classes, a user should be able to compute keys based on time. That is, a user $\mathcal{U}$ entitled to access class $v \in V_U$ during a sequence of time intervals $T_\mathcal{U} \in \mathcal{P}$ obtains private information that permits her to compute keys $k_{v,t}$ for her access class v and each $t \in T_\mathcal{U}$ (time-based key derivation). In addition, the private information allows $\mathcal{U}$ to compute, for each $t \in T_\mathcal{U}$, keys $k_{w,t}$ for each descendant access class w in the user hierarchy (class-based key derivation). Thus, key derivation now consists of two dimensions, which can potentially be performed using drastically different techniques. We give details on how to extend out current scheme to this hierarchically-temporal based model in the full version [4].

Scheme	Public information	Private information	Key derivation	Operation type	Complexity assumption
Encryption-based [5]	$O(\|V_U\|^2\|T\|^3)$	1	1	decryption	one-way functions
Pairing-based [5]	$O(\|V_U\|^2)$	$O(\|T\|)$	1	pairing evaluation	Bilinear Diffie-Hellman
Binary tree	$O(\|E_U\|\|T\|)$	$O(\log\|T\|)$	$O(\log\|T\| + diam(G_U))$	PRF	one-way functions
ISPIT+(3,1)-CSBT +EBC [15]	$O(\|E_U\|\|T\| + \|V_U\|\|T\| \cdot \log\|T\|(\log\log\|T\|)^2)$	≤ 3	$O(diam(G_U))$	decryption	IND-P1-CO encryption [13]
Our 4HS-based	$O(\|E_U\|\|T\| + \|V_U\|\|T\| \cdot \log^* n \log\log\|T\|)$	≤ 3	$O(diam(G_U))$	PRF	one-way functions
ISPIT+(3,1)-CSBT +EBC [15]	$O(\|E_U\|\|T\| + \|V_U\|\|T\| \cdot \log\|T\|\log\log\|T\|)$	≤ 3	$O(\log^*\|T\| + diam(G_U))$	decryption	IND-P1-CO encryption [13]
Our $\log^*$HS-based	$O(\|E_U\|\|T\| + \|V_U\|\|T\| \cdot \log\log\|T\|)$	≤ 3	$O(\log^*\|T\| + diam(G_U))$	PRF	one-way functions

Table 4. Comparison of time-based hierarchical KA schemes.

7 Comparison with Existing Solutions

Table 4 compares performance of our scheme with other existing solutions; only security against recovery was considered. In the table, $diam(G_U)$ denotes the diameter of the graph (i.e., maximum distance between nodes) that bounds the number of operations necessary to derive a descendant class's key in the user hierarchy G_U. Also, $|E_U|$ denotes the number of edges in G_U. The table does not list private storage at the server since it is equivalent for all solutions. Before proceeding with comparing existing results, we briefly explain what these parameters mean.

In the great majority of cases, the depth of user hierarchies is a small constant, resulting in small constant $diam(G_U)$. In cases where the depth of the original graph G_U is fairly large and it is unacceptable to have the user perform $diam(G_U)$ operations, the graph can be modified to significantly reduce $diam(G_U)$. This is done by inserting shortcut edges at random (if $diam(G_U) = O(V_U)$) or using the techniques of [1] and [2] that reduce $diam(G_U)$ to a small constant at the expense of small increase in the public storage associated with the hierarchy[4]. Thus, in this case $diam(G_U)$ is also a small constant, and parameter $|E_U|$ will need to be replaced with a slightly larger value.

We also would like to mention that the schemes [19, 18] are not listed in the table due to the difference in the expressive power. These solutions allow a user to obtain access to an arbitrary subsequence of time intervals, but require significantly slower key derivation of $O(|V_U| \cdot |T|)$ modular exponentiations.

Considering that small private user storage and fast key derivation, followed by reasonable server storage are the main evaluation criteria, we can analyze the solutions as follows. The Pairing-based scheme of [5] will have the slowest key derivation time among all of the schemes listed, as it uses pairing evaluation rather than fast encryption or PRF operations. Additionally, the number of secret keys a user has to maintain

[4] The techniques of [1] and [2] may fail on hierarchies of high dimensions, but we believe that such cases are very rare for the applications we consider in this work.

is large. Compared to the Encryption-based scheme of [5], our key derivation time is higher by a constant factor, private storage is similar (i.e., three keys instead of one), but the amount of public information the server must maintain in our scheme is much lower than in that scheme.

While the simple binary-tree approach has asymptotically higher performance, for small values of $|T|$ it will be preferred due to its simplicity. However, for the applications we envision, other solutions exhibit better performance. Thus, our recommendation is to use the simplest approach suitable for a particular setup.

The work of De Santis et al. [15] lists solutions with different performance parameters, and we include only selected two here. We chose two schemes that require a user to store 3 private keys (like in our solutions) and where time-based key derivation involves $O(1)$ and $O(\log^* n)$ decryptions, respectively. This allows us to directly compare the schemes of [15] with our schemes. As can be seen from the table, the solutions exhibit very similar performance with CSBT-based constructions having an additional factor of $\log |T|$ in the public storage space. Moreover, they do not discuss key assignment, but it does not look like their key assignment can be done in constant time, whereas our scheme allows constant time key assignment.

To summarize, our solution offers very attractive characteristics and superior performance compared to other existing solutions: each user in the system receives a small (≤ 3) number of keys, constant-time key assignment to a user, (off-line) computation of any access key involves a small number of very efficient operations, and the public storage required by our solution is only slightly higher than the number of access keys that the system must maintain.

Acknowledgments

The authors would like to thank Michael Rabinovich for his excellent suggestion of using the geo-spatial key assignment scheme to address temporal key assignment for periodic expressions. Mikhail Atallah is 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. Marina Blanton was supported by Intel Ph.D. fellowship, work was performed while at Purdue University.

References

1. 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.
2. 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)*, pages 11–18, 2006. Full version is available as *Technical Report TR 2007-30, CERIAS, Purdue University*.
3. M. Atallah, M. Blanton, and K. Frikken. Efficient techniques for realizing geo-spatial access control. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'07)*, pages 82–92, 2007.

4. M. Atallah, M. Blanton, and K. Frikken. Incorporating temporal capabilities in existing key management schemes. Full version, available as *Cryptology ePrint Archive Report 2007/245*, `http://eprint.iacr.org/2007/245`, 2007.
5. 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.
6. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO'96*, volume 1109, 1996.
7. E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems (TODS)*, 23(3):231–285, 1998.
8. E. Bertino, P. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. In *ACM Symposium on Access Control Models and Technologies (SACMAT'00)*, pages 21–30, 2000.
9. B. Briscoe. MARKS: Zero side effect multicast key management using arbitrarily revealed key sequences. In *First International Workshop on Networked Group Communication (NGC'99)*, volume 1736 of *LNCS*, pages 301–320, 1999.
10. H. Chien. Efficient time-bound hierarchical key assignment scheme. *IEEE Transactions of Knowledge and Data Engineering (TKDE)*, 16(10):1301–1304, 2004.
11. J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.
12. 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.
13. J. Katz and M. Yung. Characterization of security notions for probabilistic private-key encryption. *Journal of Cryptology*, 19:67–95, 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. A. De Santis, A. Ferrara, and B. Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. In *ACM Symposium on Access Control Models and Technologies (SACMAT'07)*, 2007.
16. Q. Tang and C. Mitchell. Comments on a cryptographic key assignment scheme for access control in a hierarchy. *Computer Standards & Interfaces*, 27:323–326, 2005.
17. 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.
18. 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.
19. Shyh-Yih Wang and Chi-Sung 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.
20. 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.
21. X. Yi. Security of Chien's efficient time-bound hierarchical key assignment scheme. *IEEE Transactions of Knowledge and Data Engineering (TKDE)*, 17(9):1298–1299, 2005.
22. X. Yi and Y. Ye. Security of Tzeng's time-bound key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(4):1054–1055, 2003.