Kumar and Mathuria *Cybersecurity* (2019) 2:8
https://doi.org/10.1186/s42400-019-0026-y

Cybersecurity

RESEARCH Open Access

Comprehensive evaluation of key management hierarchies for outsourced data

Naveen Kumar[1*] and Anish Mathuria[2]

Abstract

Key management is an essential component of a cryptographic access control system with a large number of resources. It manages the secret keys assigned to the system entities in such a way that only authorized users can access a resource. Read access control allows read access of a resource by the authorized users and disallows others. An important objective of a key management is to reduce the secret key storage with each authorized user. To this end, there exist two prominent types of key management hierarchy with single key storage per user used for read access control in data outsourcing scenario: user-based and resource-based. In this work, we analyze the two types of hierarchy with respect to static hierarchy characteristics and dynamic operations such as adding or revoking user authorization. Our analysis shows that the resource-based hierarchies can be a better candidate which is not given equal emphasis in the literature. A new heuristic for minimizing the key management hierarchy is introduced that makes it practical in use even for a large number of users and resources. The performance evaluation of dynamic operations such as adding or revoking a user's read subscription is shown experimentally to support our analytical results.

Keywords: Key management hierarchy, Data outsourcing, Access control

Introduction

Data outsourcing in the cloud is a cost-effective solution for a resource-constrained IT organization with a significant amount of data to manage. A typical data outsourcing architecture consists of three entities (Wang et al. 2009; di Vimercati et al. 2007): a data owner, a cloud service provider (CSP), and the end users. The data owner creates a service level agreement with the CSP and sends its initial set of data with other necessary information to the service provider. The end users first register with the data owner, receive their authorization information and then (to avoid any bottleneck at the data owner) can directly access the outsourced data from the CSP without interacting with the data owner. The CSP is responsible for initial user authentication, data availability to the authorized users and system scalability.

A major challenge to any data outsourcing is to keep the data confidential from unauthorized entities including the untrusted CSP. We assume "honest-but-curious" CSP which may launch only passive attacks on the stored data (Arapinis et al. 2013). Data encryption provides a straightforward solution to enforce data confidentiality. An access control mechanism allows the authorized users to access the data. The simplest cryptographic solution is to encrypt each set of related data files with a distinct secret key. The decryption keys are then distributed securely to the authorized users by the data owner. In order to reduce the secret key storage requirement (minimum secret key per user's subscription), a *key management hierarchy* (Akl and Taylor 1983; Atallah et al. 2005) or simply a *hierarchy* is generally used. A *hierarchy* is a directed acyclic graph typically composed of many nodes. A key is assigned to each node using an appropriate hierarchical key assignment scheme (Akl and Taylor 1983; Atallah et al. 2005). Data files are associated with the nodes and are encrypted with the respective node's key. The key assignment ensures that a user having a node's key can efficiently compute any descendant node's key in the hierarchy and hence access the associated data files. It also ensures that it is computationally infeasible to derive a key corresponding to a non-descendant node in the hierarchy.

Two other goals of secure data outsourcing setup (other than reducing the secret key storage with each user) are

*Correspondence: naveen_kumar@iiitvadodara.ac.in
[1]Indian Institute of Information Technology, Vadodara, India
Full list of author information is available at the end of the article

to reduce the key derivation cost and public storage cost. Optimizing public storage cost is critical when using a pay-by-use system such as cloud. Although the minimization of secret key storage per user can be addressed using key management hierarchies, other two objectives need further exploration especially when working with the large hierarchies (needed for a system with a large number of resources). The size (the number of nodes and edges) of a hierarchy depends on the number of system resources or the number of users. Therefore, the latter two objectives are more dependent on the construction of the key management hierarchy.

Two types of key management hierarchy have been previously used in the literature for secret data outsourcing: user-based and resource-based (di Vimercati et al. 2008). In a user-based hierarchy, each node represents a group of users having access to that node's key. In contrast, each node of a resource-based hierarchy represents a group of resources such that a user having access to the node's key can access each resource associated with the node.

Motivation

Blundo et al. (2010) formally prove that the problem of minimizing the number of nodes and edges in a key management hierarchy (or the number of system secret keys) required to enforce an authorization policy is NP-Hard. Their proposed heuristic to minimize the hierarchy considers only user-based hierarchies. In particular, a tree hierarchy is used which requires one or more secret keys to be stored at each user (see "User-based hierarchies" section). The heuristic considers static hierarchy and does not consider dynamic operations such as a grant or revoke read authorization, or user revocation.

Prior to the work by Kumar et al. (2015), it was a common belief that resource-based hierarchies require a significantly more public storage (i.e., $2^{|R|}$, where R is the set of resources, di Vimercati et al. (2008)) than the user-based hierarchies (i.e., $2^{|U|}$, in general $|U| << |R|$). The analysis given in Kumar et al. (2015) shows that with comparable public storage, the resource-based hierarchies performs better than the user-based hierarchies when considering very basic and frequent dynamic operation such as extending a user's read authorization.

In this work, we use a resource-based hierarchy solution with single key storage per user per subscription (as compared to the existing tree-based solution with one or more keys storage per user per subscription). The problem of finding minimum cost (sum of nodes and edges) hierarchy can be easily transformed into well-known q-RST problem (Suchý 2016) which is NP-hard (Rothvoß 2011). We prove that finding minimum cost hierarchy and q-RST problems are equivalent. Therefore, if there exists an algorithm to solve the minimum cost hierarchy problem, the algorithm can be used to solve the q-RST problem. We propose a new heuristic for minimizing the number of nodes in a generic resource-based hierarchy. The heuristic called *minimal vertex hierarchy* minimizes the number of nodes in the hierarchy and give a close solution to the minimal hierarchy. The algorithm for building a minimal vertex (resource and user-based) hierarchy is discussed in "Key management hierarchy: definitions and properties" section.

We critically analyze the user and resource-based hierarchies satisfying the proposed heuristic for minimal criteria. The work discusses the dynamic operations considering minimal vertex hierarchy and demonstrates in-depth analysis of both the hierarchy types. Both of the hierarchy types are implemented and the performance of dynamic operations are experimentally evaluated to demonstrate our analytical results. For the sake of confidence, the dynamic operations are performed over the varying size of initial hierarchies and individual results are averaged. A similar kind of implementation work is recently carried out by Hassan and Lounes (2017) to analyze a key tables-based key management scheme. However, the scheme is restricted to linear hierarchies. Similar to Blundo et al. (2010), this work revisited and introduced the definitions of discussed security solutions for the enforcement of access control policies.

Our analysis shows that both types of hierarchy satisfying the minimal heuristic criteria have comparable public storage requirements in practice. The resource-based hierarchies are more efficient in terms of computation and communication costs with respect to the dynamic operations such as extending and revoking a user's read access authorization.

Preliminaries

An authorization policy defines who can access what resource. Access authorizations are generally defined using an Access Control Matrix (ACM). We assume each user has read authorization for some resource. An ACM can be represented in two ways, either as a collection of Access Control Lists (ACLs) or CaPability Lists (CPLs) (Sandhu and Samarati 1994). An ACL corresponding to a resource is the set of users who are authorized to read the resource. On the other hand, a CPL is the set of resources for which a given user has read authorization. Both are dual of each other. For example, consider a system with four users A, B, C, D and four resources a, b, c, d. An example of ACM is shown in Fig. 1. In table (i), each row represents an ACL. $acl[o]$ represents an ACL corresponding to the resource o, i.e., the set of users who are authorized to read o. The entry $acl[a] = ABCD$ or $\{A, B, C, D\}$ means that the resource a can be read by the users A, B, C and D. Similarly, in table (ii), each row represents a CPL. $cpl[u]$ represents a CPL corresponding to user u, i.e., the set of resources for which u has read

o	acl[o]
a	ABCD
b	CD
c	AB
d	AB

(i)

u	cpl[u]
A	acd
B	acd
C	ab
D	ab

(ii)

Fig. 1 An example access control matrix as (**i**) ACLs (**ii**) CPLs

authorization. The entry $cpl[A] = acd$ or $\{a, c, d\}$ means that the user A can access the resources a, c and d.

In general, a resource can be accessed by a group of users. A subset of these users may be authorized to access another resource. For example, resource a can be accessed by users A, B, C, and D. The subsets $\{C, D\}$ and $\{A, B\}$ are authorized to access resources b and c, d, respectively. The relationships between user subsets can be represented using a hierarchy structure as shown in Fig. 2i. In the hierarchy, each node is labeled by a subset of users, hence the name user-based hierarchy (or user hierarchy). For example, user B can access the descendant nodes AB and $ABCD$, and hence can access the associated resources, i.e., c, d and a, respectively.

Consider the hierarchy shown in Fig. 2ii, where the nodes other than the individual user nodes represent resource groupings. This type of hierarchy is called a resource-based hierarchy. In the figure, user A can access all the resources a, b, c, d, whereas user D can only access a and b.

In the following section, we give the definitions and properties of different types of key management hierarchy proposed in the literature for outsourced data. We critically compare the two prominent hierarchy types (user-based and resource-based) with respect to their static structure in "Comparison of static hierarchies" section. "Dynamic access control" section gives the procedures for dynamic operations such as granting and revoking read access permissions. It also compares the two hierarchy types with respect to dynamic characteristics. In "Experimental evaluation" section, operations for both the hierarchy types are experimentally evaluated and compared. "Conclusions" section concludes this work. For the sake of readability, the notations used in this work are listed in Table 1.

Key management hierarchy: definitions and properties

In a key management hierarchy, each user is assigned a fixed number of keys using which it can derive the rest of the authorized keys. The design goals of a key management hierarchy are to minimize the secret key storage per user, system public storage, and key derivation time. In what follows, we describe and compare various resource and user-based hierarchy constructions considering the above design goals.

Resource-based hierarchies

In this section, we describe a key derivation structure called *resource hierarchy* (introduced in di Vimercati et al. (2008)), where nodes are defined based on the resource groupings (i.e., CPLs). In what follows, we first define the most general resource hierarchy structure called *resource graph*. In the definition, $v.cpl$ for a node v is a set of resources that can be accessed using node v's key.

Definition 1 (Resource graph) *A resource graph over a given set of resources R, denoted G_R, is a graph (V_R, E_R), where V_R is the power set of R and $E_R = \{e(v_i, v_j) | v_j.cpl \subset v_i.cpl\}$.*

Figure 3 shows the Hasse diagram of a resource graph for four resources $\{a, b, c, d\}$. In the graph, there is a directed path from each node v_i to node v_j such that $v_j.cpl \subset v_i.cpl$. For example, the node abc with capability list $\{a, b, c\}$ has a path to each of the nodes ab, ac, bc, a, b, and c.

In a resource graph, each user requires to store only one secret key corresponding to its respective node in the graph. For example, knowledge of key assigned to the node abc is sufficient to derive the keys for the nodes a, b, and c. Note that a resource graph is a worst case graph over a

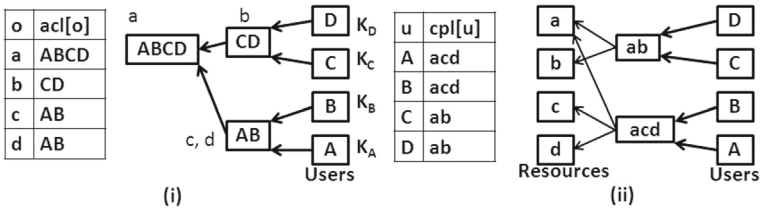

Fig. 2 Example hierarchy structures based on (**i**) ACLs (**ii**) CPLs

Table 1 Notations used

Notation	Description
$a, b, c, ...$	Resources
$A, B, C, ...$	End users
K_i	Random key assigned to node labeled i
$acl[o]$	A set of read authorized users for the resource o
$cpl[u]$	A set of resources authorized for user u
$e(i, j)$	A directed edge from node i to node j
$r_{i,j}$	A public token associated with an edge $e(i, j)$
$E()$	Symmetric encryption function
$\mathcal{E}$ and $\mathcal{D}$	A symmetric encryption and decryption operation
$\mathcal{C}$	A communication between data owner and CSP
$[X]$	It represents a node corresponding to set X of users/resources

set of resources, i.e., it contains a node for every possible grouping of resources in the given resource set and an edge between every related pair of nodes. A resource graph contains $2^{|R|}$ nodes. Considering that $|R| >> |U|$ where U is the set of users, resource graphs are less practical in use. The next key derivation structure we study, namely resource hierarchy, is a sub-graph of the resource graph. We define a material nodes set $\mathcal{M}$ that only contains the nodes used to encrypt a data file. A resource hierarchy is defined as follows.

Definition 2 (Resource hierarchy) *Let $\mathcal{A}$ be a set of CPLs over a set of users U and set of resources R. A resource hierarchy denoted $RH = (V, E)$ for given $\mathcal{A}$ is a sub-graph of $G_R = (V_R, E_R)$ where $\mathcal{M} \bigcup U \subseteq V \subseteq V_R$ and $E = E_1 \bigcup E_2$ where $E_1 = \{e([u], [cpl[u]])|u \in U\}$ and $E_2 = \{e(v_i, v_j)|v_i, v_j \in V, v_j.cpl \subset v_i.cpl\}$.*

The above definition ensures that a resource hierarchy includes root nodes representing the users and leaf nodes representing the resources. The intermediate nodes are corresponding to the given user's CPL. There is an edge from each user (u) node to the node represents its capability list ($cpl[u] \in \mathcal{A}$). Ignoring the user nodes, there is a path from every node x to node y if $y.cpl \subset x.cpl$. An example resource hierarchy is shown in Fig. 4 where (i) represents an example set of CPLs and (ii) gives a corresponding resource hierarchy. In the example hierarchy, there is an edge from user node B to node bcd since $B.cpl =< b, c, d >$ as shown in Figure (i). Similarly, there are edges from node C to node ad, node D to node ab, and node A to node abc.

In general, the public storage is defined as the total number of nodes and the number of edges present in the hierarchy as there is a public value for each node and for each edge (Atallah et al. 2005). In the resource hierarchy, the total number of edges or the nodes can be further reduced to some extent by adding additional nodes or deleting non-material nodes. For example, suppose $v1.acl = bcdex$ and $v2.acl = abcdef$, then a common subset of the two given ACLs is $bcde$. Adding $bcde$ node into the hierarchy may reduce the number of existing edges. If another node $v3.acl = abcdfy$ exists, then it may happen that instead of node $bcde$, node bcd (common to $v1$, $v2$, and $v3$) further reduces the number of edges. Therefore, there are many such possibilities exist. It motivates us to define the notion of minimal hierarchy.

Definition 3 (Minimal hierarchy problem) *To find a hierarchy $H = (V, E)$ for which $|V| + |E|$ is minimum over all $\mathcal{M} \bigcup U \subseteq V$ and $E = \{e(v_i, v_j)|v_i, v_j \in V, v_j.cpl \subset v_i.cpl\}$ is called a minimal hierarchy problem.*

Our objective is to find a hierarchy which optimizes $|V|+|E|$. We call this problem as Minimal Hierarchy Problem (MHP). In what follows, we show that the MHP is a hard problem.

The Steiner Tree Problem (STP, (Hwang and Richards 1992)) on weighted graphs asks for a tree of minimum weight that contains all leaf nodes, but may also include additional nodes. Therefore, when edge weight is fixed to

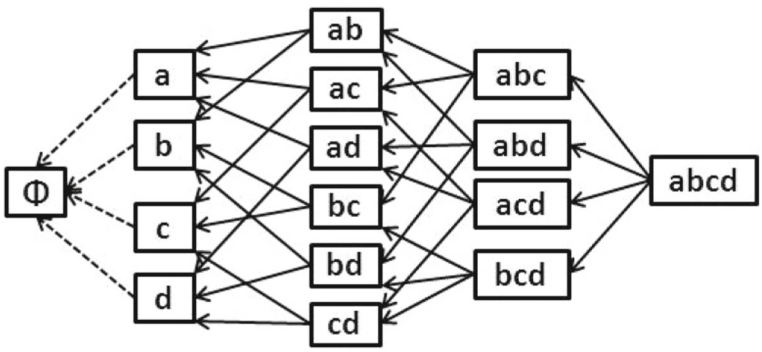

Fig. 3 A resource graph

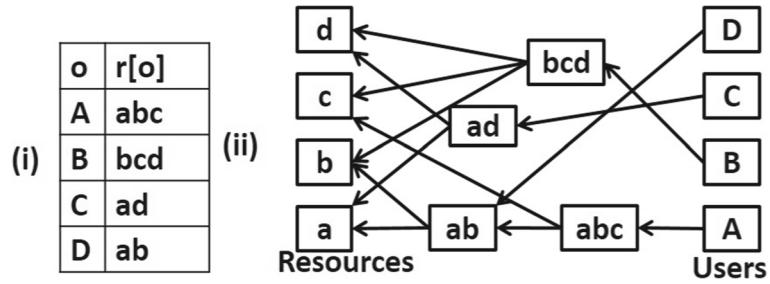

Fig. 4 (**i**) An example CPLs, and (**ii**) A resource hierarchy

1, the problem is the same as minimizing the number of edges and the non-leaf nodes in the graph. It is known that the Steiner tree problem is NP-hard and remains so even in very restricted planar cases (Aho et al. 1977). A variation of STP is directed STP whose goal is to find a minimum cost tree in a directed graph $G = (V, E)$ that connects all leaf nodes $X \in V$ to a given root $r \in V$ (Rothvoß 2011).

A generalization of directed STP is directed STP with multiple roots (or q-Root Steiner Tree, i.e., q-RST problem (Suchý 2016)). The q-RST problem is that given a directed graph $G = (V, E)$, two subsets of its nodes, a set of root nodes Rt of size q and T, the goal is to find a minimum cost subgraph of G that contains a path from each node of Rt to each node of T. The rest of the nodes in set $V \setminus (Rt \cup T)$ can be added to form a minimum cost subgraph. This optimization problem is known to be NP-hard (Suchý 2016; Rothvoß 2011).

Now, consider the q-RST problem with given directed graph $G_U = (V_U, E_U)$ containing unit weight edges, two subsets of its nodes, Rt of size q as user nodes and T the leaf nodes represents the ACLs. The goal is to find a minimum cost subgraph of G_U that contains a path from each node $v1$ of Rt to each node $v2$ in T, where $v1.acl \subseteq v2.acl$ and $v2 \neq \Phi$, i.e., there is at least one target node corresponding to the given root node. This problem is equivalent to the MHP. Therefore, if there exists an algorithm to solve MHP, the algorithm can be used to solve the q-RST problem. Below we show that MHP and q-RST problems are equivalent.

Theorem 1 *MHP and q-RST problems are equivalent.*

Proof To show the equivalence between MHP and q-RST problems, consider an arbitrary instance graph of MHP with unit directed edges, set Rt of size q containing user nodes as root nodes representing the CPLs and T the leaf nodes representing the individual resources. Now, we will show how the MHP instance can be converted into a general weighted graph as in q-RST problem. Consider each chain $C =< x_1, x_2, ..., x_i >$ of nodes in the graph such that each node except x_i in C has only one outgoing edge. Then replace C with one edge chain $C' =< x_1, x_i >$ and weight of the edge is $i - 1$, i.e., the sum of edge weights in C. The updated graph now becomes an instance of q-RST problem which says that the q-RST problem is no harder than the MHP problem. This implies that the two problems are equivalent. □

As an approximation to the MHP problem, we define a new heuristic named *minimal vertex hierarchy*. A minimal vertex hierarchy (V, E) only contains the material nodes (M) and their associated edges. To satisfy the minimality condition if we fix the number of nodes to $|M|$ then the minimum and maximum number of edges required to create a connected hierarchy will be $|M| - 1$ and $|M|(|M| - 1)/2$, respectively. Although, the number of edges may be further reduced by adding more vertices, this introduces an additional complexity of analyzing the relationship between all the vertices and edges in the hierarchy. Therefore, we will use minimal vertex hierarchy as an approximation to the minimal hierarchy (the one where $|V| + |E|$ is minimum). Following minimal vertex hierarchy, a minimal vertex *resource* hierarchy is defined as follows.

Definition 4 (Minimal vertex resource hierarchy) *Let $\mathcal{A}$ be a set of CPLs over a set U of users and set R of resources. A minimal vertex resource hierarchy denoted $RHm = (V, E)$ for given $\mathcal{A}$ is a subgraph of $G_R = (V_R, E_R)$ with $V = U \bigcup R$ and $E = \{e(v_i, v_j) | v_i = [u], u \in U, v_j = [r], r \in R,$ and $r \in cpl[u]\}$*

The above definition ensures that a minimal vertex resource hierarchy includes root nodes representing the users and leaf nodes representing the resources. Since each resource is encrypted with its dedicated leaf node's key, there is no intermediate node needed between user and resource nodes. An algorithm for constructing minimal vertex resource hierarchy corresponding to a given set of CPLs is given in Algorithm 1. There is a direct edge from every user node u to a node corresponding to a resource r if $r \in cpl[u]$. An example minimal vertex resource hierarchy is shown in Fig. 5, where (i) represents

an example set of CPLs and (*ii*) gives a corresponding minimal vertex resource hierarchy. In the example hierarchy, there is a direct edge from node A to the set of nodes $\{[a], [b], [c]\}$ since $cpl[A] = \{a, b, c\}$ as shown in Figure (i). Similarly, there are edges from node B to the set of nodes $\{[b], [c], [d]\}$, node C to the set of nodes $\{[a], [d]\}$ and node D to the set of nodes $\{[a], [b]\}$.

Algorithm 1 *Create_RHier*(ACM, U, R)

Input: An ACM containing a set of CPLs, a set of users U, and a set of resources R.
Output: Create a minimal vertex resource hierarchy corresponding to the given ACM.

1: **for** (each user $u \in U$) **do**
2: Create a node $[u]$ for u
3: **end for**
4: **for** (each user $r \in R$) **do**
5: Create a node $[r]$ for u
6: **end for**
7: **for** (each $CPL[u] \in$ ACM for a user u) **do**
8: **for** (each resource $r \in CPL[u]$) **do**
9: Create an edge $e([u], [r])$
10: **end for**
11: **end for**

Here, each leaf node in a minimal vertex resource hierarchy represents a resource node, i.e., a resource is encrypted with a leaf node's key. There is a direct edge from each user node u to all of her authorized resource nodes, i.e., resources in her capability list ($cpl[u]$).

User-based hierarchies

We review here the user-based key management hierarchies (Blundo et al. 2010; Raykova et al. 2012; Vimercati et al. 2008, 2013), where nodes are defined based on the users grouping (i.e., ACLs), instead of the resource groupings (i.e., CPLs). In what follows, we first define the user graph in a similar fashion to a resource graph and then other related hierarchy constructions. Following (Blundo et al. 2010) and the resource graph, a user graph is defined as follows, where each node represents a group of users. In the definition, notation *v.acl* represents a set of users that can access the node *v*'s key.

Definition 5 (User graph) *A user graph over a given set of users U, denoted* G_U*, is a graph* (V_U, E_U) *rooted at node* v_0*, where* V_U *is the power set of U and* $E_U = \{e(v_i, v_j) | v_i.acl \subset v_j.acl\}$.

It follows from Definition 5 that v_0 is a root node. There is a node corresponding to each subset of users and there is a directed path from each node v_i to node v_j with $v_i.acl \subset v_j.acl$. Also, there is an edge from the root node to each single user node. Figure 6 shows Hasse diagram (Baker et al. 1972) of a user graph with four users $\{A, B, C, D\}$. For simplicity, the edges that are implied by other edges are not shown in the figure.

As the resource graph, in a user graph, each user stores only one secret key corresponding to its respective node in the graph. For example, knowledge of key assigned to node A is sufficient to derive the keys assigned to nodes AB, AC, AD, ABC, ABD and $ABCD$, respectively. It also contains one hop distance to reach any descendant node in the graph but with a significant increase in the number of edges (or the public storage). It requires $O(n^n)$ edges even when excluding those implied by the transitive property, where n is the number of nodes in the hierarchy.

A user tree is a subgraph of user graph, where each node has at most one incoming edge, i.e., allows only one path between two nodes. Every node whose key is used for encrypting a resource is included in the user tree (i.e., $\mathcal{M}$). Formally, for a set of ACLs over a set of resources R, $\mathcal{M} = \{[acl[o]] : o \in R\}$. Following (Blundo et al. 2010), a user tree can be defined as follows.

Definition 6 (User tree) *Let* G_U *be a user graph over a set of users U, with root node* v_0 *and a set of material nodes M. A subgraph* $T = (V, E)$ *of* G_U *with* $\mathcal{M} \bigcup \{v_0\} \subseteq V \subseteq V_U$ *and* $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$ *that satisfies the property of being a tree rooted at* v_0 *is called a user tree.*

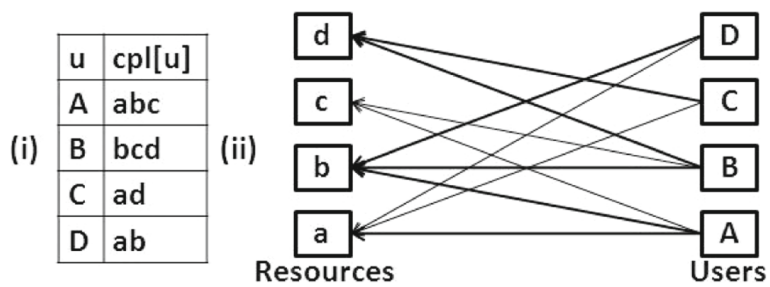

Fig. 5 (**i**) Example CPLs, and (**ii**) A minimal vertex resource hierarchy

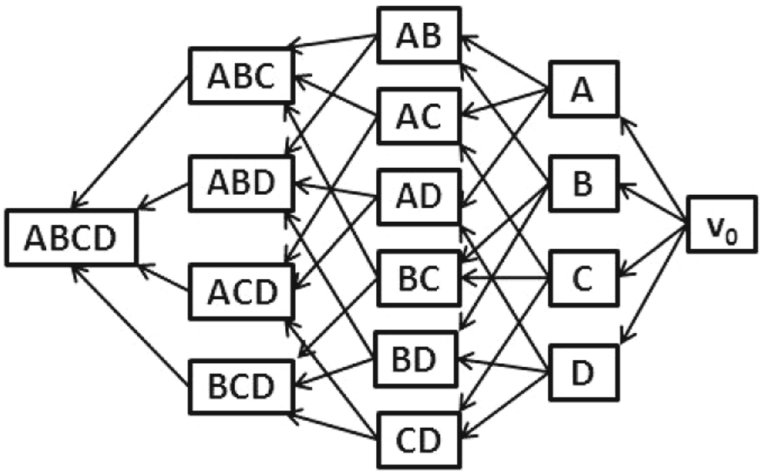

Fig. 6 A user graph over a set $\{A, B, C, D\}$ of four users

For a given set of ACLs, more than one user trees can exist. An example with four users $U = \{A, B, C, D\}$ and four resources $R = \{a, b, c, d\}$ is shown in Fig. 7. Figure 7i represents example ACLs, and Figure (ii) represents one possible user tree corresponding to the given ACLs. Each node in the user tree represents a user grouping, i.e., a set of users that can access the node's key and the associated resources. For example, node ACD represents a group of users A, C and D that can access the key K_{ACD} and hence the associated resource a. We can see in the Figure that there is a node for each ACL, i.e., $acl[o]$ for a resource o. For example, there are nodes $acl[a] = ACD$, $acl[b] = ABD$, $acl[c] = AB$ and $acl[d] = BC$, in the figure.

Although there is a node for each $acl[o]$ in Fig. 7ii, for each node there is no guarantee that its respective ACL exists. For example, there is no ACL for node A. To reduce the public storage, such nodes may be deleted from the tree, resulting in a minimal vertex user tree considering the minimal vertex hierarchy heuristic. A minimal vertex user tree can be defined as follows.

Definition 7 (Minimal vertex user tree) *Let $\mathcal{A}$ be a set of ACLs over a set of users U and set of resources R. A minimal vertex user tree $T_m = (V_m, E_m)$ is a subgraph of $G_U = (V_U, E_U)$, rooted at node v_0 with $v_0.acl = \phi$, where $V_m = \mathcal{M} \bigcup \{v_o\}$ and $E_m = \{e(v_i, v_j) | v_i, v_j \in V_m, v_i.acl \subset v_j.acl\}$.*

A minimal vertex user tree contains exactly the material nodes $\mathcal{M}$ and the root node v_0. An example minimal vertex user tree is shown in Fig. 7iii. The secret storage with each user in the tree is shown in Table 2. From the table, we see that a user may need to store more than one secret key. In the worst case, a user may need to store as many keys as the number of leaf nodes in the tree.

Claim 1 *A minimal vertex user tree is a minimal user graph.*

Proof A minimal vertex user tree contains exactly one node for each ACL. Since each node's key is used to encrypt at least one resource, the number of nodes cannot be reduced. If the number of nodes is n, then the minimum number of edges required to retain connectivity is exactly $n - 1$. Therefore, a minimal vertex user tree is always a minimal graph. □

In comparison to the user graph, a minimal vertex user tree reduces the public storage, while increasing the secret

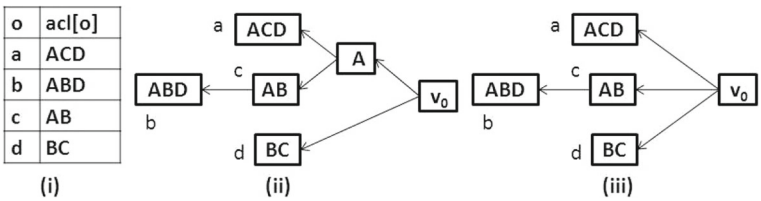

Fig. 7 (**i**) Example *ACL*s with read authorization, (**ii**) A user tree, and (**iii**) Minimal vertex user tree

Table 2 Secret keys with each user

User	Secret keys
A	K_{ACD}, K_{AB}
B	K_{AB}, K_{BC}
C	K_{ACD}, K_{BC}
D	K_{ACD}, K_{ABD}

storage at each user. In contrast to the user trees, a user hierarchy needs to store a single secret key per user and consists of a node for each user. Moreover, a node can have more than one incoming edge. Following (Raykova et al. 2012; Vimercati et al. 2008, 2013), a user hierarchy (can be viewed as a dual of resource hierarchy) can be defined as follows.

Definition 8 (User hierarchy) *Let $\mathcal{A}$ be a set of ACLs over a set U of users and set R of resources. A user hierarchy denoted $UH = (V, E)$ for given $\mathcal{A}$ is a subgraph of $G_U = (V_U, E_U)$ where $\mathcal{M} \bigcup U \subseteq V \subseteq V_U$ and $E = \{e(v_i, v_j) | v_i, v_j \in V, v_i.acl \subset v_j.acl\}$.*

Definition 9 (Minimal vertex user hierarchy) *A minimal vertex user hierarchy $UH_m = (V_m, E_m)$ for a given $UH = (V, E)$ is a subgraph of UH with $V_m = \mathcal{M} \bigcup U$.*

Consider the set of ACLs shown in Fig. 8i. A minimal vertex user hierarchy implementing the given ACLs is shown in Fig. 8ii.

In a minimal vertex user hierarchy, each user requires only one secret key, as in the case of user graph. However, a user hierarchy will take a number of edges, i.e., the public storage, as compared to the corresponding user tree (see in Fig. 7ii). This is because there is a node for each system user in the user hierarchy.

Although the MHP problem is NP-hard, constructing a minimal vertex user hierarchy for a given ACM can be done in polynomial time. A procedure for constructing a minimal vertex user hierarchy for a given ACM is shown in Algorithm 2. In the algorithm, the notation $[x]$ represents a node corresponding to set x of users. A node n is called a out-neighbor of node m if there is a directed edge from m to n.

The Algorithm 2 works as follows. A node is created for each user in set U (Steps 1-3). For each ACL in the given ACM, a corresponding node X is created (Step 5) and inserted into the hierarchy (Steps 6 to 26). For each user u in the given ACL, a node S after which X can be inserted (satisfies the access control relationships) is searched (Steps 7 to 18). Then, outgoing edges from node X corresponding to S and u are updated (Steps 19 to 24). Incoming edge to X is then updated (Step 25). At the end of this algorithm, a user hierarchy is created corresponding to the given ACLs in the ACM. For a given set of resources R, the Algorithm 2 will take a running time cost of $O\left(|R|^2\right)$ in the worst case, considering $|U| << |R|$. It is due to the statement numbers 4 and 11 in the algorithm each of which iterates $O(|R|)$ times. Statement number 6 and 9 will iterate $O(|U|)$ times each.

Comparison of static hierarchies

A hierarchy with a fixed structure is called a static hierarchy. In this section, we compare minimal vertex user and resource hierarchies in a static situation. An ACM is said to be in the worst case if all of its ACLs or CPLs are distinct. We will compare the number of nodes and edges that are required to construct a minimal vertex hierarchy for a worst case ACM. In "Dynamic access control" section, we give algorithms for dynamic operations that guarantee the minimal vertex hierarchy construction.

Let $|U|$ and $|R|$ denote the number of users and resources, respectively. We assume that $|U| << |R|$ but $|R| < 2^{|U|}$. For example, consider that we need to create an electronic health record management system for India, and assume that 1 crore patients receive care every year. Suppose a central database is created to store the patient

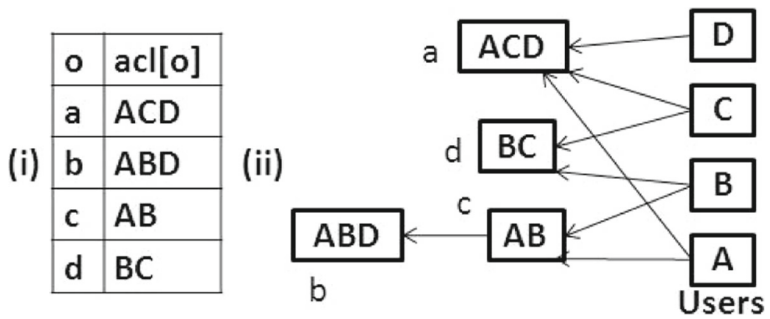

Fig. 8 (**i**) Example *ACLs* with read authorization, and (**ii**) A minimal vertex user hierarchy

Algorithm 2 *Create_UHier*(ACM, U)

Input: An ACM containing a set of ACLs and a set of users U.

Output: Create a user hierarchy corresponding to the given ACM.

```
for (each user u ∈ U) do
    Create a node [u] for u
end for
for (each ACL ∈ ACM) do
    Create a node X for ACL
    for (each user u ∈ ACL) do
        Create and initialize set S = {u}
        Set "Update" = True
        while ( "Update" = True) do
            Set "Update" = False
            for (each out-neighbor n of [S]) do
                if (n.acl ⊂ ACL) then
                    S ← n.acl                    /* Update set S */
                    Set "Update" = True
                    Break        /* Exit from inner for loop */
                end if
            end for
        end while
        for (each out-neighbor n1 of [S]) do
            if (ACL⊂ n1.acl) then
                Create an edge e(X, n1)  /* Update outgoing
                edges from X */
                Delete edge e([S], n1)
            end if
        end for
        Create an edge e([S], X)      /* Update incoming
        edge on X */
    end for
end for
```

records. For 100 years and assuming 20 documents per patient per year, it requires $\sim 10^{10}$ data files to be stored. However, for a set of only 50 users, $2^{|U|} = 2^{50} \sim 10^{15}$ which is a significant number, as compared to the total number of resources in an organization.

Cost of user hierarchy In a user hierarchy, consider a set of ACLs in the worst case, i.e., each resource o has a distinct $acl[\,o]$. As there is a node for each $acl[\,o]$, the maximum number of nodes is $|R|$. In case $|U|$ is small and $2^{|U|} < |R|$ then a maximum number of nodes will be $2^{|U|}$. Therefore, the total number of nodes in the hierarchy will be $min(2^{|U|}, |R|)$. In total $min(2^{|U|}, |R|)$ or $O(|R|)$ nodes are needed assuming $|R| < 2^{|U|}$.

For finding the number of edges required for a given number of nodes, consider user nodes as level 0 nodes, directly connected nodes of the level 0 nodes as level 1 nodes, and so on. In the worst case, the level 0 contains ${}^{|U|}C_1$ nodes, level 1 contains ${}^{|U|}C_2$ nodes, and so on (similar to user graph). Also, the number of incoming edges at each node in level 1 is 1 and in level 2 is 2 and so on. Therefore, the total number of incoming edges at level 1 is $1 \times^{|U|} C_1$, at level 2 is $2 \times^{|U|} C_2$ and so on. Now the total number of edges can be written as follows.

$$1 \times^{|U|} C_1 + 2 \times^{|U|} C_2 + ... + (|U| - 1) \times^{|U|} C_{|U|-1} + (|U|) \times^{|U|} C_{|U|} \quad (1)$$

$$= \frac{|U|}{0!} + \frac{|U|(|U|-1)}{1!} + ... + \frac{|U|(|U|-1)}{1!} + \frac{|U|}{0!} \quad (2)$$

$$= 2\left(\frac{|U|}{0!} + \frac{|U|(|U|-1)}{1!} + ... + \frac{|U|(|U|-1)...(|U|-(|U|/2))}{(|U|/2)!}\right) \quad (3)$$

In total, it comes out as $2\left(\sum_{i=0}^{|U|/2} \frac{|U|!}{(|U|-i-1)!i!}\right)$, i.e., $O\left(|U|^{|U|/2}\right)$ due to the last term in Eq. 3. Also, the number of levels gives the key derivation steps (or time), i.e., $O(|U|)$ (in worst case).

When considering the number of edges in worst case minimal vertex user hierarchy, all the ACLs are distinct of $O|R|$ number of users each and there is no node whose corresponding ACL is a subset of other (i.e., all nodes are at the same level). It creates a hierarchy with two level: user nodes in one level and other nodes in the second level. Now, the total number of edges will be $O(|U||R|)$.

Cost of minimal vertex resource hierarchy In the worst case minimal vertex resource hierarchy, each user has a direct edge to each of its authorization resource node. In total, $|U| + |R|$ nodes and $|U||R|$ edges are needed in the

Table 3 Comparison of storage and key derivation cost

Hierarchy →	User-based hierarchies			Minimal vertex
↓ Attributes	User graph	User tree	Minimal vertex user hierarchy	resource hierarchy
# of keys/users	Single	Multiple	Single	Single
# of nodes	$2^{\|U\|}+1$	$min\left(\|R\|, 2^{\|U\|}\right)+1$	$\|U\| + \|R\|$	$\|U\| + \|R\|$
# of edges	$O\left(\|U\|^{\|U\|/2}\right)$	$O\left(min\left(\|R\|, 2^{\|U\|}\right)\right)$	$O(\|U\|\|R\|)$	$O(\|U\|\|R\|)$
key derivation cost	$O(\|U\|)$	$O(\|U\|)$	$O(\|U\|)$	$O(1)$

worst case. Also, the key derivation cost will be $O(1)$ due to a direct edge from a user to an authorized resource node.

Table 3 compares the minimal vertex resource hierarchy with existing user-based hierarchies (user graph, user tree, and minimal vertex user hierarchy) in the worst case. We can see from the table that, the maximum number of nodes and edges in both minimal vertex user and resource hierarchies are $|U| + |R|$ and $O(|U||R|)$, respectively. The key derivation cost in minimal vertex resource hierarchy is only one edge whereas in minimal vertex user hierarchy is $|U| - 1$ edges in the worst case. This is more in minimal vertex user hierarchy because it may form the longest chain of $O(|U|)$ nodes.

Dynamic access control

Data access authorizations change with time as employees join and leave the organization or the department within the organization. A scheme with dynamic access control would allow granting or revoking access authorizations. In the following, we evaluate the user and resource-based hierarchies in terms of computational and communication costs of the common dynamic operations.

Algorithms for user hierarchy

Grant/revoke read access In user hierarchy, if access authorization is granted (or revoked) for a resource o to a user u then $acl[o]$ will be updated to $acl[o]' = acl[o] \cup \{u\}$ (or $acl[o]' = acl[o] \setminus \{u\}$). Now, since $acl[o] \neq acl[o]'$ (both represent different nodes in the hierarchy), resource o will be now encrypted with the key $K_{[acl[o]']}$ corresponding to $acl[o]'$. To avoid storing multiple copies of the resource encrypted with different keys ($K_{[acl[o]']}$ and $K_{[acl[o]]}$) for security reasons, data owner must delete the old copy from the server. Since granting read access is a frequent operation, associated re-encryption operation to the outsourced resource by the data owner should be avoided, if possible.

Consider Algorithm 3 for granting read access. Running time of the algorithm with respect to the hierarchy manipulation, i.e., excluding encryption, decryption or communication cost will be $O(U + R)$. It is due to the statement number 6 in the algorithm that requires cost $O(U)$ in updating incoming edges to new node v_{new} and $O(R)$ in updating outgoing edges. In the following, $\mathcal{E}$ represents the cost of one symmetric encryption operation, $\mathcal{D}$ the cost of one symmetric decryption operation and $\mathcal{C}$ the cost of one communication between the data owner and the CSP.

In Algorithm 3, granting read access for a resource to a user requires the following steps: (1) downloading the resource from the server ($1\mathcal{C}$), (2) decrypting it using the old key ($1\mathcal{D}$), (3) encrypting it with the new key ($1\mathcal{E}$), and (4) storing it back to the server ($1\mathcal{C}$) (i.e., $total\ cost = 1\mathcal{E} + 1\mathcal{D} + 2\mathcal{C}$). For example, consider the user hierarchy shown in Fig. 9i, granting read access for resource c to users C leads to the modified hierarchy shown in Fig. 9ii. In the modified hierarchy, a new node ABC is inserted and the resource c is encrypted with K_{ABC}.

Algorithm 3 *Grant_Revoke_Read_Access*(UH, o, u)

1: Find node v with $v.acl = acl[o]$ in UH
2: In case of
"Grant operation": $acl[o] \leftarrow acl[o] \bigcup \{u\}$ /* update the ACL of resource o */
"Revoke operation": $acl[o] \leftarrow acl[o] \setminus \{u\}$
3: Find node v_{new} with $v_{new}.acl = acl[o]$
4: **if** (v_{new} does not exist in the UH) **then**
5: Create node v_{new} with $v_{new}.acl = acl[o]$
6: Insert v_{new} into UH
7: **end if**
8: Download the encrypted version o' of o from the server
9: $o \leftarrow D_{K_v}(o')$
10: $o'' \leftarrow E_{K_{v_{new}}}(o)$
11: Outsource o'' to the server
/* Delete v if $v.acl$ is not in the ACL list*/
12: **if** (does not exist $p \in R$ with $acl[p] = v.acl$) **then**
13: Delete v and associated edges from UH /* deleting redundant node */
14: **end if**
15: Publish updated UH to the cloud server

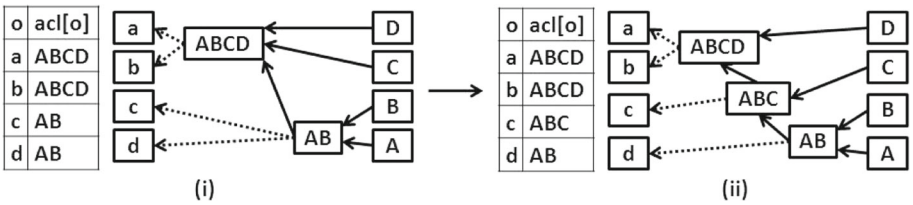

Fig. 9 Modified example of minimal vertex user hierarchy (**i**) before, and (**ii**) after granting read access

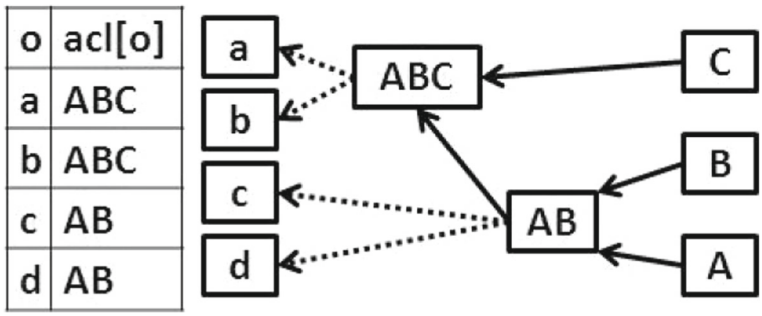

Fig. 10 Modified example of minimal vertex user hierarchy after revoking user *D*

User revocation Since each node in a user hierarchy represents a user grouping, a user revoke operation requires a modification to the hierarchy. Revoking a user requires that each node previously accessible to the revoked user be deleted and replaced by a new node (without revoked user label). For example, consider the minimal vertex user hierarchy given in Fig. 9i. To revoke D we delete the node $ABCD$ and replace it with the new node ABC (by deleting label D). Now, resources a and b are re-encrypted with the new key (K_{ABC}) so that user D will not be able to access the revoked resources. The updated hierarchy is shown in Fig. 10.

Algorithms for resource hierarchy

Grant read access To grant read access for a resource o to a user u, the data owner executes Algorithm 4.

Algorithm 4 *Grant_readAccess*(RH, o, u)

Input: A minimal vertex resource hierarchy RH, a resource o, and a user u.
Output: Grant read access of resource o to user u.

1: $u.cpl = u.cpl \bigcup \{o\}$ /* updating user u's CPL */
2: Create an edge from $[u]$ to $[o]$ by computing a public edge token $r_{[u],[o]}$
3: Publish $r_{[u],[o]}$ (and $E(o, K_{[o]})$, if new resource) at the cloud server to update RH

In the algorithm, $[x]$ represents a node corresponding to set x of users or resources. $K_{[o]}$ is the key used to encrypt resource o. Consider the example hierarchy in Fig. 11i. Initially, user C has read access to the resources a and b. Suppose, read access for resource c is to be granted to the user C. Using Algorithm 4, user C's capability list $C.cpl = \{a, b\}$ is updated by inserting resource c, i.e., $C.cpl = \{a, b, c\}$ (Step 1). An edge is created from node $[u]$ to $[c]$ (Step 2). All updated public information (i.e., $r_{[u],[o]}$ and $E(o, K_{[o]})$ (if o is new resource)) will be now published at the server (Step 3). The modified CPL and the hierarchy are shown in Fig. 11ii.

Revoke read access To revoke read authorization of a resource o for a user u assuming both exists, the data owner executes Algorithm 5. For example, consider the hierarchy in Fig. 11ii, where user B has initially read access for the resources b, c and d. Suppose, read access of resource d is revoked from user B, the algorithm works as follows. Old capability list of user B, i.e., bcd is updated to bc (Step 1). A new key $K'_{[d]}$ is assigned to node d (Step 2). Encrypted resource d is downloaded from the server, decrypted using old key $K_{[d]}$ and then encrypted with new key $K'_{[d]}$ (Steps $3-5$). Edge $r_{B,[d]}$ is deleted (Step 6). Now, for each user node v with $o \subset v.cpl$, compute public token for edge $e(v, o)$ and update it with the stored one (Steps $7-9$). The updated resource hierarchy information is then sent to the server along with encrypted resource $K'_{[d]}$

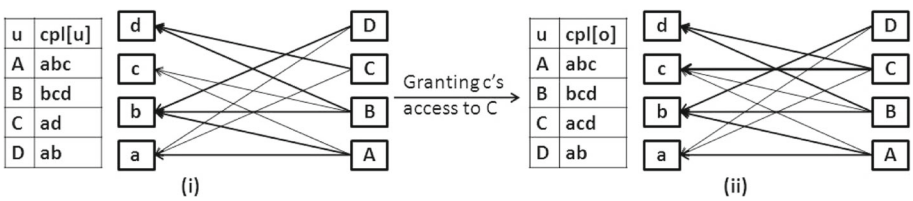

Fig. 11 (**i**) An example minimal vertex resource hierarchy, and (**ii**) Granting read access for resource c to user C

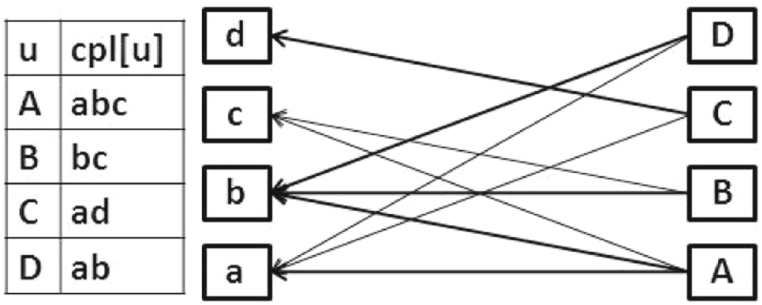

Fig. 12 After revoking read access of resource *d* from user *B*

(Step 10). The updated *CPL* and resource hierarchy are shown in Fig. 12.

User revocation To revoke a user u, the data owner executes the following. For each outgoing edge $e(u, o)$ from u to some resource o, the data owner calls the procedure $Revoke\_readAccess(RH, u, o)$ (Algorithm 5).

Algorithm 5 $Revoke\_readAccess(RH, o, u)$

1: $u.cpl \leftarrow u.cpl \setminus \{o\}$ /* updating user u's CPL */
2: Update node $[o]$'s key to $K'_{[o]}$
3: Download the encrypted version o' of o from the server
4: $o \leftarrow D_{K_{[o]}}(o)$
5: $o'' \leftarrow E_{K'_{[o]}}(o)$
6: Delete edge from $[u]$ to $[o]$ by deleting public edge token $r_{[u],[o]}$
7: **for** (each user v with $o \subset v.cpl$) **do**
8: Compute $r_{[v],[o]}$ and use it to replace the old public edge token in RH
9: **end for**
10: Publish o'' and updated RH information at the cloud server

Comparison of dynamic hierarchies

Table 4 compares the minimal vertex UH and RH. It compares the two with respect to the number of encryption ($\mathcal{E}$) or decryption ($\mathcal{D}$) operations needed by the data owner, communications ($\mathcal{C}$) needed with the CSP to grant one read access, revoke one read access, and whether revoking a user requires modification to the hierarchy structure. An attractive property of the minimal vertex RH is that it does not require any encryption or decryption operation while granting read access of a user. It requires single communication between the data owner and CSP to update the outsourced hierarchy structure while granting read access of a user. Also, it does not require any modification to the hierarchy structure when a user is revoked, unlike the user-based hierarchies. Revoking a user's read access right takes similar cost in both the hierarchy types.

Experimental evaluation

We have implemented the minimal vertex UH and RH for read access control on a local area network. The goal of the experiment is to evaluate the cost of dynamic operations from the perspective of the user and the data owner. We will evaluate the time of user's grant and revoke access right operations, and elapsed time performance of the data owner machine. The elapsed time is the time difference between a start and finishing time for a set of operations.

Setup For testing purposes, we use two machines: a file server and a data owner. Each machine consists of an Intel core 2 quad *Q*8400 processor 2.66 GHz with 3 GB RAM and 7200 RPM, 16 MB Cache, SATA 3.0 Gb/s hard drive. Both systems running windows XP are connected with a 1 Gbps Ethernet link. We choose $AES-128$ as the cipher for file encryption and employ $SHA-1$ as the hash function (found in java.security package). We implement grant and revoke read methods in Java with JDK 1.7. The test

Table 4 Comparison of computation and communication cost

Hierarchy type ↓	Cost of a grant read access	Cost of a revoke read access	Modification of hierarchy due to user revocation
Minimal vertex UH	$1\mathcal{E}+1\mathcal{D}+2\mathcal{C}$	$\|R\|(1\mathcal{E}+1\mathcal{D})+2\mathcal{C}$	Yes
Minimal vertex RH	$1\mathcal{C}$	$\|R\|(1\mathcal{E}+1\mathcal{D})+2\mathcal{C}$	No

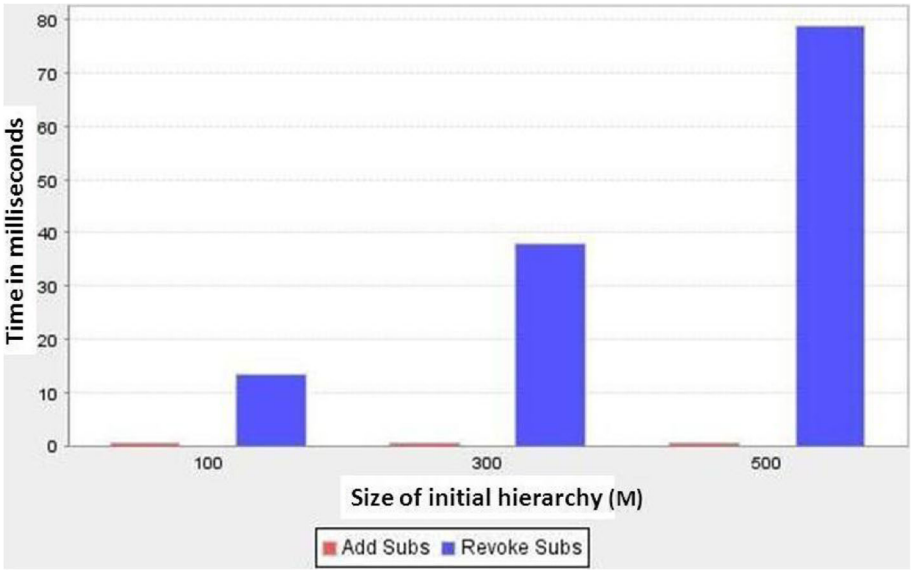

Fig. 13 Permission operation cost

includes a file server that stores 1000 files. The file size varies from 1 MB to 2 MB. The hierarchy is implemented using Hashmap in Java by storing it as an adjacency list. For the test, we fix the number of users to 30 and number of resources to 50. Considering fewer resources will not affect our experimental results since the cost of a grant or revoke operation dependent only on the corresponding resource whose access right is updated. After fixing these, we create different initial hierarchies. We define the size of initial hierarchy in terms of the number M of consecutive grant access right operations. Each grant operation randomly selects a user and a resource from the set of 1000 files.

Minimal vertex RH: Grant and revoke read operations cost

We first evaluate the cost of one grant and revoke operations cost at the data owner. An initial minimal vertex RH is created for a fixed value of M. This defines an initial ACM. Then the grant and revoke permissions are initiated in sequence at the data owner machine for which it updates the respective CPLs and the hierarchy structure. We define a thread containing one grant and one revoke operation that will execute simultaneously to maintain the same size of the initial hierarchy. The thread is executed 100 times. The average cost of each operation in the thread is then computed separately immediately after the corresponding hierarchy is published.

Figure 13 shows the cost of one grant and one revoke operation for different sizes of initial hierarchy, i.e., $M = 100, 300, 500$ and taking an average over 100 operations. Table 5 summarizes the cost (in milliseconds) of one grant or revoke operation along with average number of file re-encryptions needed for different values of M. From the figure, we conclude that the cost of one grant operation is approximately same with different size of initial hierarchy. This is due to the fact that each grant operation adds to at most one node into the hierarchy and updating of corresponding edges. However, the cost of one revoke operation increases almost linearly with the size of initial hierarchy. As the size of hierarchy increases by randomly applying grant permission operations with the same number of users and resources, the user's subscription (subscribed resources) will increase. This will lead to an increase in the number of re-encryption operations at the time of revoke operation and hence the revocation cost.

Figure 14 shows the computation for average cost of revoke operation when considering $M = 100$. We take an average over 100 operations. It requires 296 total file re-encryptions and on average 3 re-encryptions per revoke operation. The average cost of revoke operation is 13.247 ms.

Table 5 Grant and revoke subscription cost in minimal vertex RH

Size of initial hierarchy (M)	Grant operation	Revoke operation	Average file re-encryptions
100	0.477ms	13.2ms	3
300	0.454ms	38.0ms	7
500	0.440ms	78.8ms	11

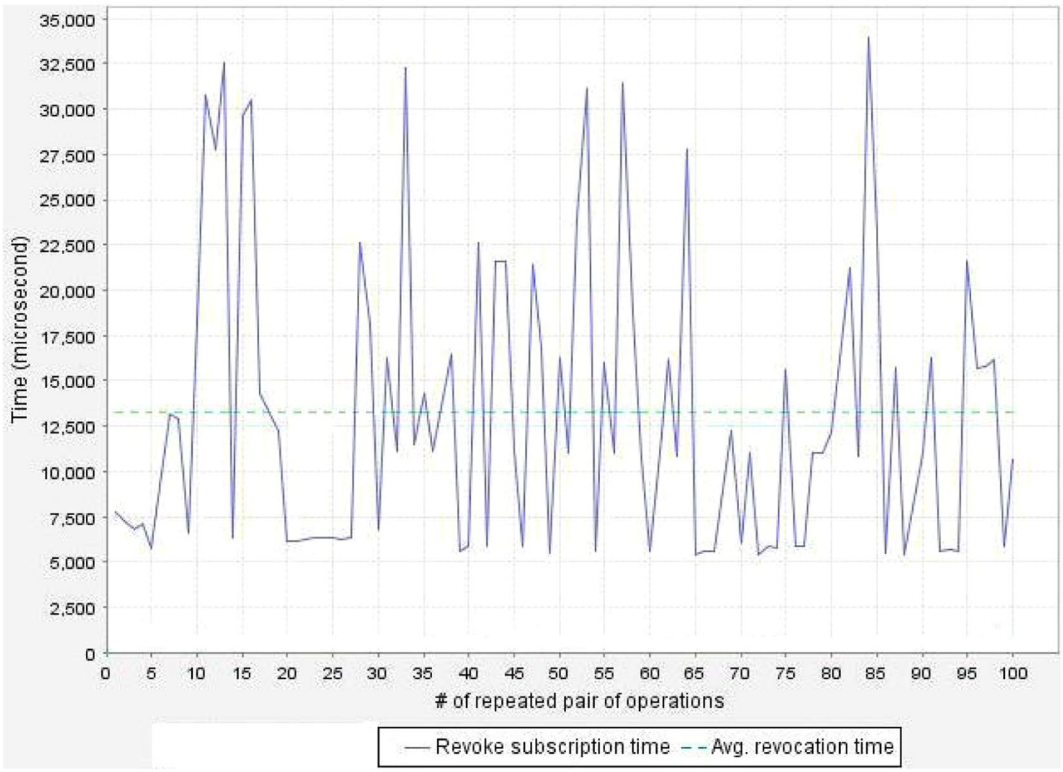

Fig. 14 Average elapse time of one grant/revoke operation

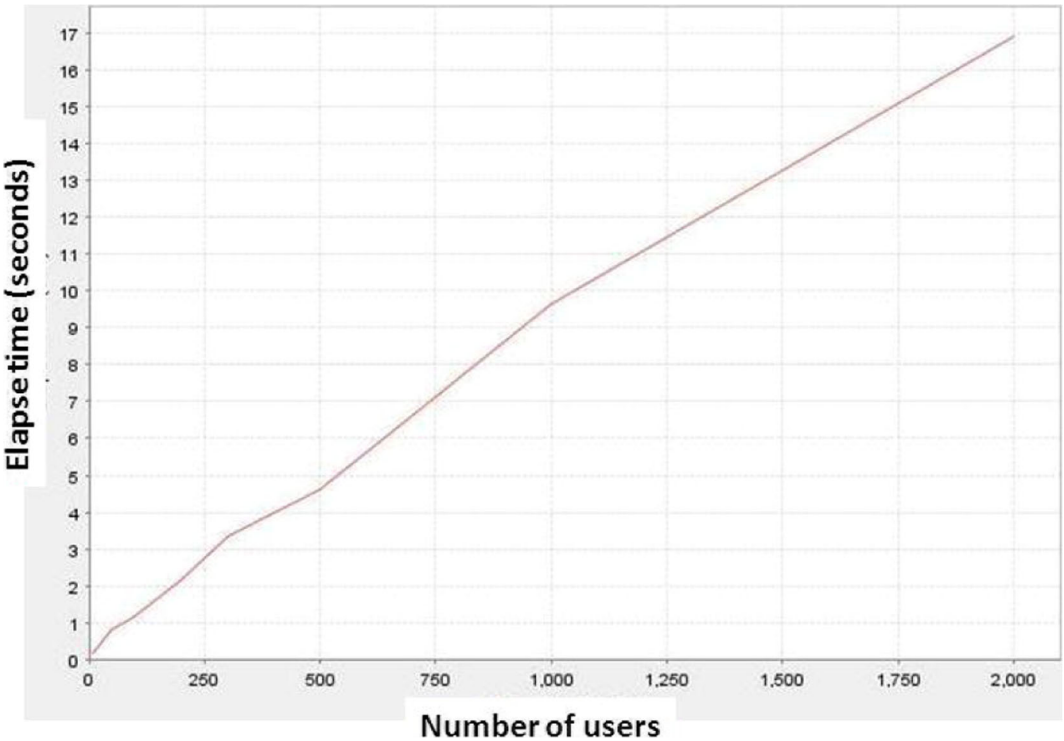

Fig. 15 Elapsed time performance of data owner machine for evaluating user threads

Table 6 Grant and revoke cost in minimal vertex UH

Size of initial hierarchy (M) →		200	500	1000
Operation	Average file size			
Grant	100 KB	12.961 ms	13.923 ms	18.530 ms
	1 MB	146.174 ms	160.385 ms	186.561 ms
Revoke	100 KB	13.511 ms	14.223 ms	17.935 ms
	1 MB	133.274 ms	151.843 ms	172.015 ms

Performance of data owner machine In the above evaluation, we considered only one user. Now, we consider a number of users involve in grant or revoke operations. For each operation, the data owner will update the ACM and corresponding hierarchy. To evaluate the data owner's elapsed time performance for handling a number of user threads, we simulate T simultaneous threads at the data owner. Due to random inputs for each operation, we perform the test 100 times and then the average cost of one batch of T threads is computed. We perform the tests for $T = 10, 50, 100, 200, 300, 500, 700, 1000, 1500, 2000$. M is fixed to 100. Figure 15 shows the results. From the figure, we conclude that there is almost linear relation between the elapsed time and the number of threads T.

Minimal vertex UH: grant and revoke read operations cost

Similar to the minimal vertex RH, the minimal vertex UH is created by fixing M and the corresponding ACM is stored. The grant and revoke operations are initiated in the same way as in the minimal vertex RH. The evaluation cost is shown in Table 6. For a given file size, our results show that the grant and revoke access right operations have a similar cost. This is because each operation requires one re-encryption of an outsourced resource and an addition of at most one node in the hierarchy.

Minimal vertex UH and RH: comparing grant read operation cost

Considering the experimental setup described above, we evaluate the cost of one grant read permission for a user. Figure 16 compares the two hierarchies against grant operation cost. We fixed the initial hierarchy parameter $M = 200, 500$ and 1000. The average file size is 1 MB. This grant operation is executed 100 times. The average cost of one operation is then computed. The results are shown in Table 7. The Fig. 16 shows that in minimal vertex UH the cost of one grant operation is significantly large in comparison to minimal vertex RH. It is due to file encryption and decryption operations needed in the minimal vertex UH when user subscription is granted. These operations are not required in the minimal vertex RH.

Minimal vertex UH and RH: Comparing user revoke operation cost

Figure 17 compares minimal vertex user and resource hierarchies with respect to a user revoke operation cost. In the experiment we only consider the hierarchy modification cost due to user revoke operation, i.e., the cost of resource encryption and decryption is omitted for simplicity. It is to be noted here that the average cost of encryption and decryption operations required per user revocation is same in both the hierarchy types. The graph shows that the hierarchy modification cost significantly

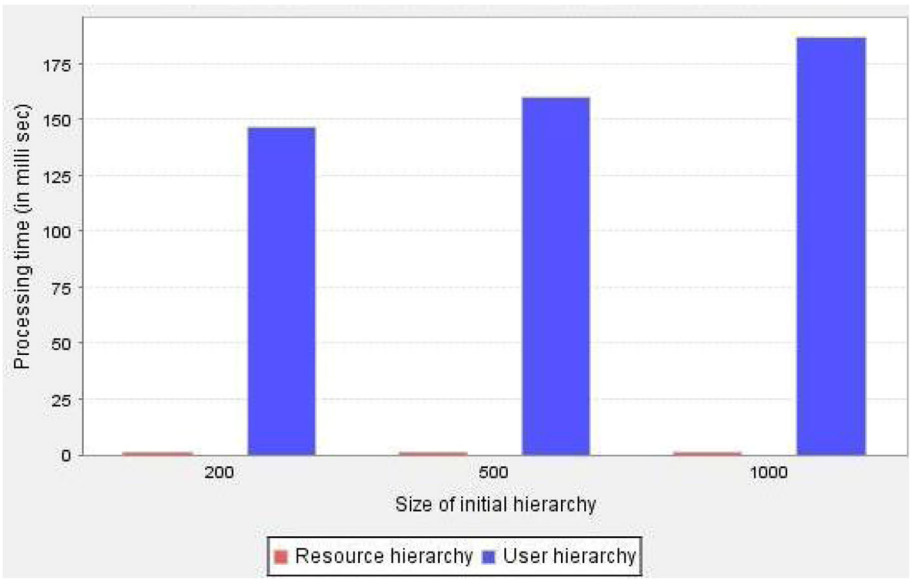

Fig. 16 Elapsed time of one grant read subscription operation

Table 7 Comparison of grant read operation cost

Size of initial hierarchy (M)	Avg. file size	UH grant op.	RH grant op.
200	1MB	146.18 ms	0.481 ms
500	1MB	162.17 ms	0.472 ms
1000	1MB	193.26 ms	0.459 ms

increases in minimal vertex UH with the increase in initial hierarchy size. This is due to the increase in a number of nodes to be modified with the increase in the size of user's ACL. In the minimal vertex RH, the hierarchy modification cost is constant and straightforward as there is an direct edge between a user node and its authorized resource nodes which only needs to be deleted from the hierarchy.

Conclusions

We critically analyzed the types of key management hierarchy used for data outsourcing and based on a new heuristic named minimal vertex hierarchy for optimizing the hierarchy. Such hierarchies require only one secret key per user. Our analysis shows that the storage requirement for minimal vertex resource hierarchies will be same as minimal vertex user hierarchies. The key derivation cost is constant in case of minimal vertex resource hierarchies as compared to the linear cost (i.e., $O(U)$) in minimal vertex user hierarchies. Also, the minimal vertex resource hierarchies perform better in case of dynamic operations such as extending read authorization and revoking a user without affecting other required functionalities. Based on our analysis, we recommend the use of resource-based hierarchies for data access control in a system with a large number of resources. The proposed algorithms for the dynamic operations will be used to maintain the hierarchy size. For the sake of our arguments, we have implemented the two hierarchy types and evaluated the results experimentally. Our results show that the cost of one grant operation is significantly large in user-based hierarchies as compared to resource-based hierarchies. The resource-based hierarchies are also improved over the other when considering user revocation operation.

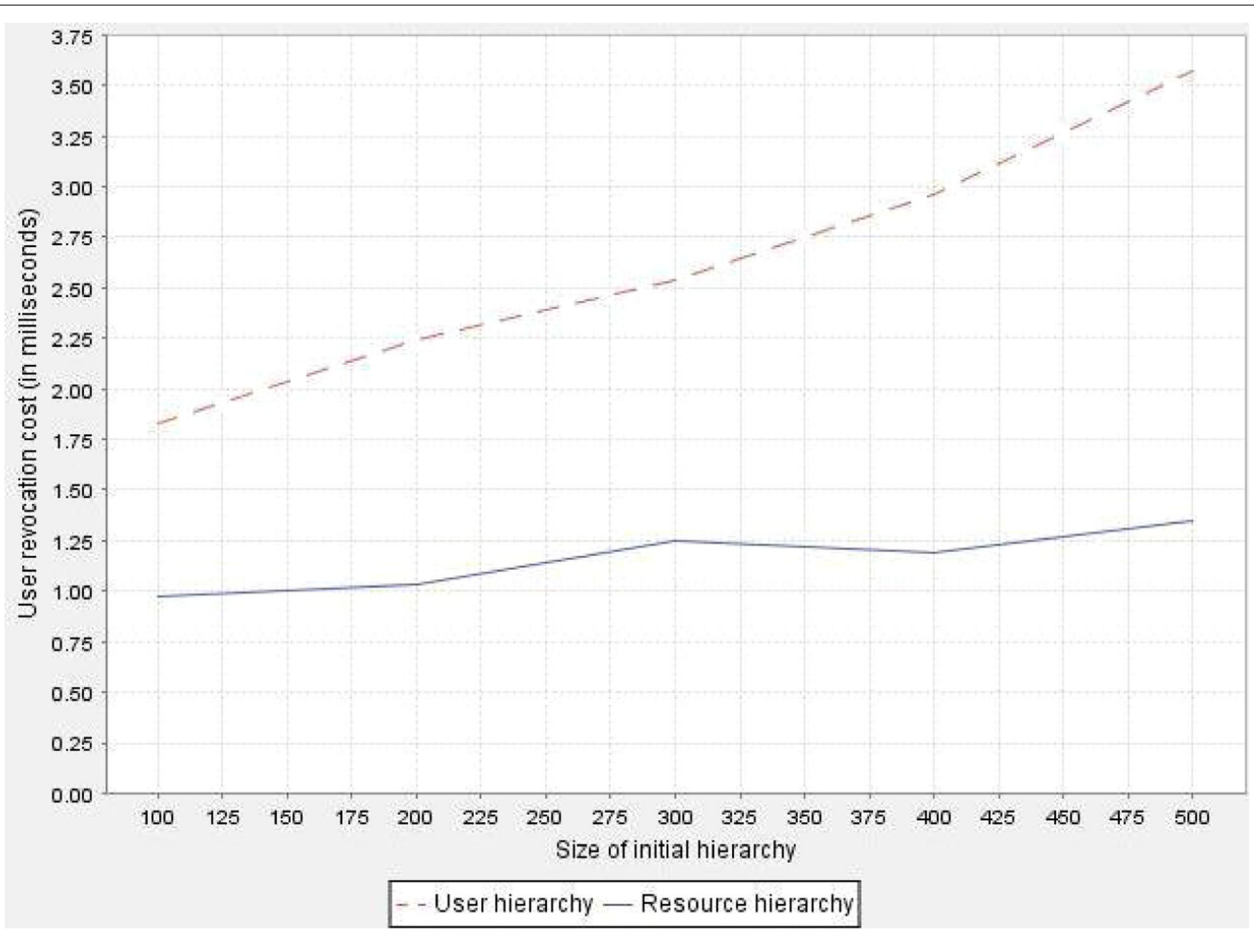

Fig. 17 Average elapsed time of one user revoke operation

Authors' contributions
NK: Initiate the idea, conceptual reasoning, preparation of the manuscript, does experiment evaluation and formulate end results. AM: Coordinate in the initiation of this idea, suggested important conceptual corrections for preparation of the manuscript, participated in drafting or revising it critically and given approval for the final submission. Both authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Indian Institute of Information Technology, Vadodara, India. [2]Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, India.

Received: 6 September 2018 Accepted: 28 January 2019
Published online: 19 February 2019

References

Aho AV, Garey MR, Hwang FK (1977) Rectilinear steiner trees: Efficient special-case algorithms. Networks 7(1):37–58. https://doi.org/10.1002/net.3230070104

Akl SG, Taylor PD (1983) Cryptographic solution to a problem of access control in a hierarchy. ACM Trans Comput Syst 1(3):239–248

Arapinis M, Bursuc S, Ryan M (2013) Privacy-suppor -ting cloud computing by in-browser key translation. J Comput Secur 21(6):847–880. https://doi.org/10.3233/JCS-130489

Atallah MJ, Frikken KB, Blanton M (2005) Dynamic and efficient key management for access hierarchies. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005. pp 190–202. https://doi.org/10.1145/1102120.1102147

Baker KA, Fishburn PC, Roberts FS (1972) Partial orders of dimension 2. Networks 2(1):11–28. https://doi.org/10.1002/net.3230020103

Blundo C, Cimato S, Vimercati SDC, Santis AD, Foresti S, Paraboschi S, Samarati P (2010) Managing key hierarchies for access control enforcement: Heuristic approaches. Comput Secur 29(5):533–547. https://doi.org/10.1016/j.cose.2009.12.006

di Vimercati SDC, Foresti S, Samarati P (2008) Recent advances in access control. In: Handbook of Database Security - Applications and Trends. pp 1–26. https://doi.org/10.1007/978-0-387-48533-1_1

di Vimercati SDC, Foresti S, Jajodia S, Paraboschi S, Samarati P (2007) Over-encryption: Management of access control evolution on outsourced data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007. pp 123–134. http://www.vldb.org/conf/2007/papers/research/p123-decapitani.pdf

Hassen HR, Lounes E (2017) A key management scheme evaluation using markov processes. Int J Inf Sec 16(3):271–280

Hwang FK, Richards DS (1992) Steiner tree problems. Networks 22(1):55–89. https://doi.org/10.1002/net.3230220105

Kumar N, Mathuria A, Das ML (2015) Comparing the efficiency of key management hierarchies for access control in cloud. In: Security in Computing and Communications - Third International Symposium, SSCC 2015, Kochi, India, August 10-13, 2015. Proceedings. Springer. pp 36–44. https://doi.org/10.1007/978-3-319-22915-7_4

Raykova M, Zhao H, Bellovin SM (2012) Privacy enhanced access control for outsourced data sharing. In: Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers. Springer. pp 223–238. https://doi.org/10.1007/978-3-642-32946-3_17

Rothvoß T (2011) Directed steiner tree and the lasserre hierarchy. CoRR abs/1111.5473. 1111.5473

Sandhu RS, Samarati P (1994) Access control: Principle and practice. Comm Mag 32(9):40–48. https://doi.org/10.1109/35.312842

Suchý O (2016) On directed steiner trees with multiple roots. In: Graph-Theoretic Concepts in Computer Science - 42nd International Workshop, WG 2016, Istanbul, Turkey, June 22-24, 2016, Revised Selected Papers. pp 257–268. https://doi.org/10.1007/978-3-662-53536-3_22

Vimercati SDC, Foresti S, Jajodia S, Livraga G, Paraboschi S, Samarati P (2013) Enforcing dynamic write privileges in data outsourcing. Comput Secur 39:47–63

Wang W, Li Z, Owens R, Bhargava BK (2009) Secure and efficient access to outsourced data. In: Proceedings of the First ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009. pp 55–66. https://doi.org/10.1145/1655008.1655016