Efficient Key Management for Enforcing Access Control in Outsourced Scenarios

C. Blundo and S. Cimato and S. De Capitani di Vimercati and A. De Santis and S. Foresti and S. Paraboschi and P. Samarati

Abstract Data outsourcing is emerging today as a successful paradigm allowing individuals and organizations to exploit external servers for storing and distributing data. While trusted to properly manage the data, external servers are often not authorized to read them, therefore requiring data to be encrypted. In such a context, the application of an access control policy requires different data to be encrypted with different keys so to allow the external server to directly enforce access control and support selective dissemination and access.

The problem therefore emerges of designing solutions for the efficient management of the encryption policy enforcing access control, with the goal of minimizing the number of keys to be maintained by the system and distributed to users. Since such a problem is NP-hard, we propose a heuristic approach to its solution based on a key derivation graph exploiting the relationships among user groups. We experimentally evaluate the performance of our heuristic solution, comparing it with previous approaches.

1 Introduction

Data outsourcing has become increasingly popular in recent years. The main advantage of data outsourcing is that it promises higher availability and more effective disaster protection than in-house operations. However, since data owners physically release their information to external servers that are not under their control, data

C. Blundo, A. De Santis
Università di Salerno, 84084 Fisciano - Italy, e-mail: {carblu,ads}@dia.unisa.it

S. Cimato, S. De Capitani di Vimercati, S. Foresti, P. Samarati
Università di Milano, 26013 Crema - Italy, e-mail: {cimato,decapita,foresti,samarati}@dti.unimi.it

S. Paraboschi
Università di Bergamo, 24044 Dalmine - Italy, e-mail: parabosc@unibg.it

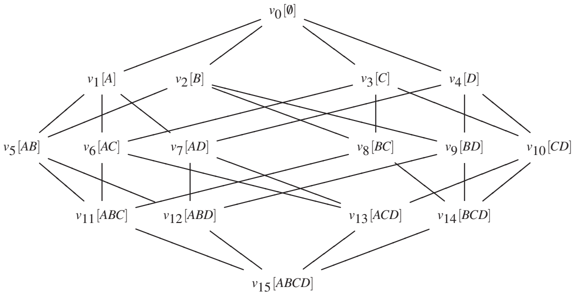
confidentiality and even integrity may be put at risk. As a matter of fact, sensitive data (or data that can be exploited for linking with sensitive data) are stored on external servers. Besides protecting such data from attackers and unauthorized users, there is the need to protect the privacy of the data from the so called *honest-but-curious* servers: the server to whom data are outsourced, while trustworthy to properly manage the data, may not be trusted by the data owner to read their content. The problem of protecting data when outsourcing them to an external honest-but-curious server has emerged to the attention of researchers very recently. Existing proposals (e.g., [4, 8, 13]) in the data outsourcing area typically resort to store the data in encrypted form, while associating with the encrypted data additional indexing information that is used by the external DBMS to select the data to be returned in response to a query. Also, existing works typically assume that the data owner is a single organization that encrypts the data with a single key and that all users have complete visibility of the whole database. Such approaches clearly are limiting in today's scenarios, where remotely stored data may need to be accessible in a selective way, that is, different users may be authorized to access different views of the data.

There is therefore an increasing interest in the definition of security solutions that allow the enforcement of access control policies on outsourced data. A promising solution in this direction consists in integrating access control and encryption. Combining cryptography with access control essentially requires that resources should be encrypted differently depending on the access authorizations holding on them, so to allow their decryption only to authorized users [5, 6]. The application of this approach in data outsourcing scenarios allows owners: *1)* to encrypt data, according to an encryption policy regulated by authorizations, *2)* outsource the data to the external servers, and *3)* distribute to users the proper encryption keys. Proper encryption and key distribution automatically ensure obedience of the access control policy, while not requiring the data owner to maintain control on the data storage and on accesses to the data. In the literature, there are different proposals exploiting encryption for access control [5, 6, 10]. In [5], the authors address the problem of access control enforcement in the database outsourcing context, by exploiting selective encryption and hierarchical key assignment schemes on trees. Since a crucial aspect for the success of such a solution is the efficacy, efficiency, and scalability of the key management and distribution activities, the authors propose an algorithm that minimizes the number of secret keys in users' key rings. In [6], the authors address the problem of policy updates. Here, two layers of encryption are imposed on data: the inner layer is imposed by the owner for providing initial protection, the outer layer is imposed by the server to reflect policy modifications (i.e., grant/revoke of authorizations). In [10], the authors introduce a framework for enforcing access control on published XML documents by using different cryptographic keys over different portions of the XML tree and by introducing special metadata nodes in the structure.

In this paper we propose a novel heuristic approach minimizing the number of keys to be maintained by the system and distributed to users. Consistently with other proposals in the literature [5, 6], we base our solution on *key derivation* exploiting

$$\begin{array}{c|cccccc|} & \mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} & \mathbf{r_4} & \mathbf{r_5} & \mathbf{r_6} \\ \hline A & 1 & 1 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 & 1 \\ C & 0 & 1 & 0 & 1 & 0 & 1 \\ D & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

(a)



(b)

Fig. 1 An example of access matrix (a) and of user graph over $\mathscr{U}$=$\{A,B,C,D\}$ (b)

a *key derivation graph* that allows users to derive new keys by combining other keys and public tokens. As we will show, compared with previous proposals, our heuristics prove efficient and effective in the computation of a key derivation graph.

2 Basic concepts

We assume that the data owner defines a discretionary access control policy to regulate access to the distributed resources. Consistently with other approaches for data outsourcing, we assume access by users to the outsourced resources to be read-only. Given a set $\mathscr{U}$ of users and a set $\mathscr{R}$ of resources, an *authorization policy* over $\mathscr{U}$ and $\mathscr{R}$ is a set of pairs $\langle u,r \rangle$, where $u \in \mathscr{U}$ and $r \in \mathscr{R}$, meaning that user u can access resource r. An authorization policy can be modeled via an *access matrix* $\mathscr{A}$, with a row for each user $u \in \mathscr{U}$, a column for each resource $r \in \mathscr{R}$, and $\mathscr{A}[u,r]$ set to 1 (0, resp.) if u has (does not have, resp.) authorization to access r. Given an access matrix $\mathscr{A}$, $acl(r)$ denotes the *access control list* of r (i.e., the set of users that can access r), and $cap(u)$ denotes the *capability list* of u (i.e., the set of resources that u can access). Figure 1(a) illustrates an example of access matrix with four users (A, B, C, D) and six resources (r_1, …, r_6), where, for example, $acl(r_2)$=$\{A,C\}$ and $cap(C)$=$\{r_2,r_4,r_6\}$.

In the data outsourcing scenario, the enforcement of the authorization policy cannot be delegated to the remote server, which is trusted neither for accessing data content nor for enforcing the authorization policy. Consequently, the data owner has to be involved in the access control enforcement. To avoid the owner's involvement in managing access and enforcing authorizations, recently *selective encryption* techniques have been proposed [5,6,10]. Selective encryption means that the *encryption policy* (i.e., which data are encrypted with which key) is dictated by the authorizations to be enforced on the data. The basic idea is to use different keys for encrypting data and to release to each user the set of keys necessary to decrypt all and only the resources the user is authorized to access. For efficiency reasons, selective encryption is realized through symmetric keys.

A straightforward solution for implementing selective encryption associates a key with each resource r and communicates to each user u the keys used to encrypt the resources in $cap(u)$. It is easy to see that this solution, while correctly enforcing the authorization policy, is too expensive to manage, due to the high number of keys each user has to keep. Indeed, any user $u \in \mathscr{U}$ would need to hold as many keys as the number of resources she is authorized to access.

To avoid users having to store and manage a huge number of (secret) keys, consistently with other proposals in the literature [5, 6], we exploit a *key derivation method* that allows the derivation of a key starting from another key and some public information [1–3, 7, 9, 11]. In our scenario, the derivation relationship between keys can be represented through a graph with a vertex v for each possible set of users and an edge (v_i,v_j) for all pairs of vertices such that the set of users represented by v_i is a subset of the set of users represented by v_j. In the following, we use $v.acl$ to denote the set of users represented by vertex v and $v.key$ to denote the key associated with v. Formally, a user graph is defined as follows.

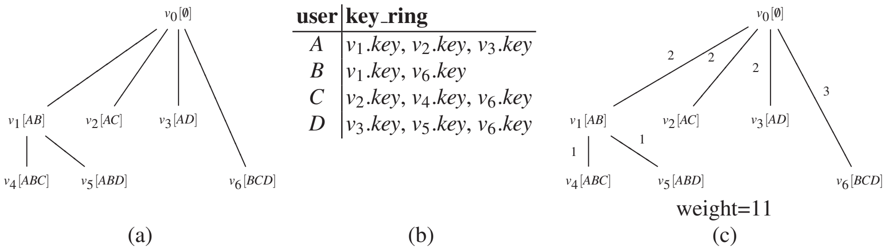
Definition 1 (User Graph). Given a set $\mathscr{U}$ of users, a *user graph* over $\mathscr{U}$, denoted $G_{\mathscr{U}}$, is a graph $\langle V_{\mathscr{U}},E_{\mathscr{U}}\rangle$, where $V_{\mathscr{U}}=P(\mathscr{U})$ is the power set of $\mathscr{U}$, and $E_{\mathscr{U}}=\{(v_i,v_j) \mid v_i.acl \subset v_j.acl\}$.

As an example, consider the set of users $\mathscr{U}=\{A,B,C,D\}$. Figure 1(b) reports the user graph, where, for each vertex v_i, the users in the square brackets represent $v_i.acl$ and, for clearness of the picture, edges that are implied by other edges (relationships between sets differing for more than one user) are not reported.

By exploiting the user graph defined above, the authorization policy can be enforced: *i)* by encrypting each resource with the key of the vertex corresponding to its access control list (e.g., resource r_4 should be encrypted with $v_{11}.key$ since $acl(r_4)=v_{11}.acl=\{A,B,C\}$), and *ii)* by assigning to each user the key associated with the vertex representing the user in the graph. Since edges represent the possible key derivations, each user u, starting from her own key, can directly compute the keys of all vertices v such that $u \in v.acl$. It is easy to see that this approach to design the encryption policy *correctly enforces* the authorization policy represented by matrix $\mathscr{A}$, meaning that each user u can only derive the keys for decrypting the resources she is authorized to access. For instance, with reference to the user graph in Fig. 1(b), user A knows the key associated with vertex v_1 from which she can derive, following the edges outgoing from v_1, the set of keys of vertices v_5, v_6, v_7, v_{11}, v_{12}, v_{13}, and v_{15}.

3 Problem formulation

The key derivation methods working on trees are in general more convenient and simpler than those working on DAGs and require a lower amount of publicly available information. Indeed, given two keys k_i and k_j in $\mathscr{K}$, where $\mathscr{K}$ is the set of



user	key_ring
A	$v_1.key$, $v_2.key$, $v_3.key$
B	$v_1.key$, $v_6.key$
C	$v_2.key$, $v_4.key$, $v_6.key$
D	$v_3.key$, $v_5.key$, $v_6.key$

Fig. 2 A user tree (a), the corresponding key rings (b), and its weighted version (c)

symmetric encryption keys in the system, such that k_j can be directly derived from k_i, then k_j=h(k_i,l_j), where l_j is a publicly available label associated with k_j and h is a deterministic cryptographic function. We then transform, according with the proposal in [5], the user graph $G_{\mathscr{U}}$ in a *user tree*, denoted T, enforcing the authorization policy in $\mathscr{A}$. Since each resource r is encrypted with the key associated with the vertex representing $acl(r)$, the user tree must include the set, denoted $\mathscr{M}$, of all vertices, called *material vertices*, representing acl values and the empty set of users (i.e., $\mathscr{M} = \{v \in V_{\mathscr{U}} \mid v.acl{=}\emptyset \vee \exists\, r \in \mathscr{R}$ with $v.acl = acl(r)\}$), as formally defined in the following.

Definition 2 (User tree). Let $\mathscr{A}$ be an access matrix over a set $\mathscr{U}$ of users and a set $\mathscr{R}$ of resources, and $G_{\mathscr{U}} = \langle V_{\mathscr{U}}, E_{\mathscr{U}} \rangle$ be the user graph over $\mathscr{U}$. A *user tree*, denoted T, is a tree $T = \langle V,E \rangle$, subgraph of $G_{\mathscr{U}}$, rooted at vertex v_0, with $v_0.acl{=}\emptyset$, where $\mathscr{M}{\subseteq}V{\subseteq}V_{\mathscr{U}}$, and $E{\subseteq}E_{\mathscr{U}}$.

In other words, a user tree is a tree, rooted at the vertex representing the empty user group $\emptyset$, subgraph of $G_{\mathscr{U}}$, and spanning all vertices in $\mathscr{M}$.

To grant the correct enforcement of the authorization policy, each user u has a key ring, denoted $key\_ring_T(u)$, containing all the keys necessary to derive the keys of all vertices v such that $u{\in}v.acl$. The key ring of each user u must then include the keys associated with all vertices v such that $u{\in}v.acl$ and $u{\notin}v_p.acl$, where v_p is the parent of v. If $u{\in}v_p.acl$, u must already have access to the key in v_p and must be able to derive $v.key$ through the key of v_p, which she knows either by derivation or by direct communication.

Clearly, given a set of users and an authorization policy $\mathscr{A}$, more user trees may exist. Among all possible user trees, we are interested in determining a *minimum user tree*, correctly enforcing a given authorization policy and minimizing the number of keys in users' key rings.

Definition 3 (Minimum user tree). Let $\mathscr{A}$ be an access matrix and T be a user tree correctly enforcing $\mathscr{A}$. T is *minimum* with respect to $\mathscr{A}$ iff $\nexists T'$ such that T' correctly enforces $\mathscr{A}$ and $\sum\limits_{u \in \mathscr{U}} |\text{key\_ring}_{T'}(u)\,| < \sum\limits_{u \in \mathscr{U}} |\text{key\_ring}_{T}(u)\,|$.

Figure 2(a) illustrates an example of user tree and Fig. 2(b) reports the corresponding user key rings.

We observe that the keys in the key ring could be managed with the use of *tokens*, public pieces of information that allow the reconstruction of a secret from another one [2, 3]. The minimality of the user tree implies a minimization in the number of tokens, making the approach presented in this paper applicable to scenarios using tokens.

Given an access matrix $\mathscr{A}$, different minimum user trees may exist and our goal is to compute one of them, as stated by the following problem definition.

Problem 1. Let $\mathscr{A}$ be an access matrix. Determine a minimum user tree T.

Since Problem 1 is NP-hard, in [5] we proposed a heuristic algorithm working as follows: *1)* the algorithm initially computes the closure of $\mathscr{M}$ with respect to the intersection operator; *2)* the algorithm selects, for each vertex, a parent choosing first the vertices representing larger sets of users, and then material vertices; finally *3)* the algorithm prunes non necessary vertices.

4 Minimum spanning tree heuristics

Our solution is based on a reformulation of Problem 1 in terms of a weight minimization problem. We start by introducing the concept of weight in association with a user tree.

Definition 4 (Weight function). Let T=$\langle V,E \rangle$ be a user tree.

- w:$E \rightarrow \mathbb{N}$ is a weight function such that $\forall(v_i, v_j) \in E$, $w(v_i, v_j) = |\, v_j.acl \setminus v_i.acl\,|$
- $\texttt{weight}(T) = \sum\limits_{(v_i,v_j)\in E} w(v_i, v_j)$.

According to this definition, the weight $w(v_i,v_j)$ of edge (v_i,v_j) in E is the number of users in $v_j.acl \setminus v_i.acl$. The weight $\texttt{weight}(T)$ of user tree T is then defined as the sum of the weights of its edges. Problem 1 can be reformulated as the problem of finding a *minimum weight* user tree. In fact, the presence of an edge $(v_i,v_j) \in E$ implies that users in $v_i.acl$ should know both keys $v_i.key$ and $v_j.key$ while users in $v_j.acl \setminus v_i.acl$ need only to know $v_j.key$. It is then sufficient to include key $v_i.key$ in the key rings of all users in $v_i.acl$, since $v_j.key$ can be derived from $v_i.key$, and to include key $v_j.key$ in the key rings of users in $v_j.acl \setminus v_i.acl$. This is equivalent to say that $w(v_i, v_j)$ corresponds to the number of users whose key ring must include key $v_j.key$. Generalizing, it is immediate to conclude that $\texttt{weight}(T)$ is equal to the sum of the total number of keys stored in users' key rings (i.e., $\texttt{weight}(T) = \sum\limits_{u\in\mathscr{U}} |key\_ring_T(u)\,|$).

The problem of computing a user tree with minimum weight is NP-hard since the Vertex Cover problem can be reduced to it (for space reason, we do not report the proof of this reduction). We therefore propose a heuristic algorithm for solving such a problem that consists first in computing a minimum spanning tree (MST) over a

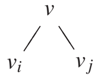
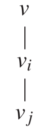
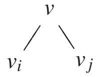
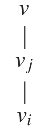
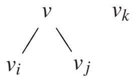
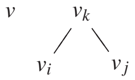
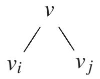
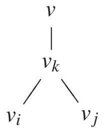
Case (v_k**.**acl**=**v_i**.**$acl\cap v_j$**.**acl)	**Initial configuration**	**Final configuration**	**weight_red**(v,v_i,v_j)
1 $v_k.acl=v_i.acl$			$\|v_i.acl\| - \|v.acl\|$
$v_k.acl=v_j.acl$			$\|v_j.acl\| - \|v.acl\|$
2 $v_k \in V$ and $v_k \neq v_i$ and $v_k \neq v_j$			$2(\|v_k.acl\| - \|v.acl\|)$
3 $v_k \notin V$			$\|v_k.acl\| - \|v.acl\|$

Fig. 3 Possible updates to the user tree

graph $G = \langle V, E', w\rangle$, with $V = \mathscr{M}$, $E' = \{(v_i, v_j) \mid v_i, v_j \in V \wedge v_i.acl \subset v_j.acl\}$, and w the weight function defined in Definition 4, rooted at v_0. It is immediate to see that the MST over G is a user tree whose weight can be further reduced with the addition of vertices obtained from the intersection of at least two vertices already in the MST. The insertion of a new vertex v as a parent of at least two vertices, say v_i and v_j, can reduce the weight of the tree since the key ring of users in $v.acl$ should only include $v.key$ instead of both $v_i.key$ and $v_j.key$.

The basic idea behind our approach is that for each internal vertex v of the minimum spanning tree (i.e., for each vertex with at least one child) and for each pair $\langle v_i, v_j\rangle$ of children of v, we first compute the set U of users in $v_i.acl$ and $v_j.acl$, that is, $U = v_i.acl \cap v_j.acl$. If $U \neq v.acl$, we then evaluate if the insertion in T of vertex v_k representing U can reduce `weight`(T). Among all possible pairs of children of v, we then choose the pair $\langle v_i, v_j\rangle$ such that, when v_k is possibly inserted in the tree (or it becomes the parent of at least one of two vertices v_i and v_j), we obtain the highest reduction in the weight of the tree. Such a weight reduction, formally defined by function *weight_red*:$V \times V \times V \rightarrow \mathbb{N}$, depends on whether v_k exists in T or it needs to be inserted. The following three cases, represented in Fig. 3, may occur.

Case 1 v_k=v_i (or v_k=v_j), that is, one of the two children represents a subset of the users represented by the other child. The user tree can be updated by removing the edge connecting vertex v with v_j (v_i, resp.) and by inserting the edge connecting v_i with v_j (v_j with v_i, resp.). As a consequence, the weight of the tree is reduced by $w(v, v_j) - w(v_i, v_j)$, which is equal to $|v_i.acl| - |v.acl|$.

```
INPUT
set $\mathscr{U}$ of users
set $\mathscr{R}$ of resources
access matrix $\mathscr{A}$
criterion ($I_{\max}$, $I_{\min}$, or $I_{\mathrm{rnd}}$) to adopt
OUTPUT
user tree $T = \langle V, E \rangle$
MAIN
$V := \emptyset$
$E := \emptyset$
/* Phase 1: select material vertices */
$Acl_{\mathscr{M}} := \{acl(r) | r \in \mathscr{R}\} \cup \{\emptyset\}$
for each $acl \in Acl_{\mathscr{M}}$ do
    create vertex $v$
    $v.acl := acl$
    $V := V \cup \{v\}$
/* Phase 2: compute a minimum spanning tree */
$E' := \{(v_i, v_j) \mid v_i, v_j \in V \wedge v_i.acl \subset v_j.acl\}$
let $w$ be a weight function such that
$\forall (v_i, v_j) \in E'$, $w(v_i, v_j) = |v_j.acl \setminus v_i.acl|$
$G := (V, E', w)$
let $v_0$ be the vertex in $V$ with $v_0.acl = \emptyset$
$T :=$ Minimum_Spanning_Tree($G, v_0$)
/* Phase 3: insert non-material vertices */
$T :=$ Factorize_Internal_Vertices($T$, criterion)
return($T$)
```

```
FACTORIZE_INTERNAL_VERTICES($ST$, criterion)
let $ST$ be $\langle V, E \rangle$
for each $v \in \{v_i \mid v_i \in V \wedge \exists (v_i, v_j) \in E\}$ do
  $CC_v := \{\langle v_i, v_j \rangle \mid (v, v_i), (v, v_j) \in E \wedge v_i.acl \cap v_j.acl \neq v.acl\}$
  $max\_red :=$ max$\{weight\_red(v, v_i, v_j) \mid \langle v_i, v_j \rangle \in CC_v\}$
  while $CC_v \neq \emptyset$ do
    $MC_v := \{\langle v_i, v_j \rangle \mid \langle v_i, v_j \rangle \in CC_v \wedge weight\_red(v, v_i, v_j) = max\_red\}$
    case criterion of
      $I_{\mathrm{rnd}}$: choose $\langle v_i, v_j \rangle \in MC_v$ randomly
      $I_{\max}$: choose $\langle v_i, v_j \rangle \in MC_v : |v_i.acl| + |v_j.acl|$ is maximum
      $I_{\min}$: choose $\langle v_i, v_j \rangle \in MC_v : |v_i.acl| + |v_j.acl|$ is minimum
    $U := v_i.acl \cap v_j.acl$
    find $v_k \in V : v_k.acl = U$
    case $v_k$ of
      /* case 1 */
      $= v_i$:          $E := E \setminus \{(v, v_j)\} \cup \{(v_i, v_j)\}$
      $= v_j$:          $E := E \setminus \{(v, v_i)\} \cup \{(v_j, v_i)\}$
      /* case 2 */
      $\neq v_i \wedge \neq v_j$: $E := E \setminus \{(v, v_i), (v, v_j)\} \cup \{(v_k, v_i), (v_k, v_j)\}$
      /* case 3 */
      UNDEF:    create a vertex $v_k$
                $v_k.acl := U$
                $V := V \cup \{v_k\}$
                $E := E \setminus \{(v, v_i), (v, v_j)\} \cup \{(v, v_k), (v_k, v_i), (v_k, v_j)\}$
    $CC_v := \{\langle v_i, v_j \rangle \mid (v, v_i), (v, v_j) \in E \wedge v_i.acl \cap v_j.acl \neq v.acl\}$
    $max\_red :=$ max$\{weight\_red(v, v_i, v_j) \mid \langle v_i, v_j \rangle \in CC_v\}$
return($ST$)
```

Fig. 4 Heuristic algorithm for computing a minimal user tree

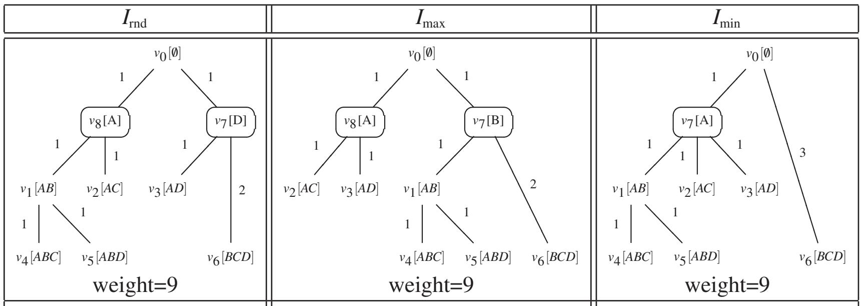
Case 2 $v_k \in V$ and $v_k \neq v_i$ and $v_k \neq v_j$, that is, there is a vertex in the tree representing U. The user tree can be updated by removing the edges connecting vertex v with both v_i and v_j, and by inserting two new edges, connecting v_k with v_i and v_j, respectively. As a consequence, the weight of the tree is reduced by $w(v, v_i) + w(v, v_j) - (w(v_k, v_i) + w(v_k, v_j))$, which is equal to $2(|v_k.acl| - |v.acl|)$.

Case 3 $v_k \notin V$, that is, there is no vertex representing U in the tree.[1] The user tree can be updated by: creating a new vertex v_k with $v_k.acl = U$; removing the edges connecting v with both v_i and v_j; and inserting three new edges connecting respectively: *1)* v with v_k, *2)* v_k with v_i, and *3)* v_k with v_j. As a consequence, the weight of the tree is reduced by $w(v, v_i) + w(v, v_j) - (w(v, v_k) + w(v_k, v_i) + w(v_k, v_j))$, which is equal to $|v_k.acl| - |v.acl|$.

As an example, consider the weighted user tree in Fig. 2(c) and suppose to compute the intersection between the pairs of children of the root vertex v_0. In this case, all possible intersections correspond to singleton sets of users that are not already represented in the tree and therefore each intersection requires the addition of a new vertex in the tree as child of v_0 and parent of the considered pair of children.

Formally, for each internal vertex v of the minimum spanning tree $ST = \langle V, E \rangle$, we first compute the set CC_v of pairs of *candidate children* as follows: $CC_v = \{\langle v_i, v_j \rangle \mid (v, v_i), (v, v_j) \in E \wedge v_i.acl \cap v_j.acl \neq v.acl\}$. Among all possible pairs in CC_v, we then choose a pair $\langle v_i, v_j \rangle$ that maximizes *weight_red*. Note that different pairs of

[1] Note that this is the only case that can occur if both v_i and v_j belong to $\mathscr{M}$, since T is initially obtained as a minimum spanning tree over G.



user	key_ring
A	v_3.*key*, v_8.*key*
B	v_1.*key*, v_6.*key*
C	v_2.*key*, v_4.*key*, v_6.*key*
D	v_5.*key*, v_7.*key*

user	key_ring
A	v_1.*key*, v_8.*key*
B	v_7.*key*
C	v_2.*key*, v_4.*key*, v_6.*key*
D	v_3.*key*, v_5.*key*, v_6.*key*

user	key_ring
A	v_7.*key*
B	v_1.*key*, v_6.*key*
C	v_2.*key*, v_4.*key*, v_6.*key*
D	v_3.*key*, v_5.*key*, v_6.*key*

Fig. 5 User trees and key rings computed by our heuristics over the MST of Fig. 2(c)

vertices in CC_v may provide the same maximum weight reduction. In this case, different preference criteria may be applied for choosing a pair, thus obtaining different heuristics. In particular, we propose the following three criteria:

- I_{rnd}: at random;
- I_{max}: in such a way that $|v_i.acl| + |v_j.acl|$ is maximum, ties are broken randomly;
- I_{min}: in such a way that $|v_i.acl| + |v_j.acl|$ is minimum, ties are broken randomly.

Any of these three preference criteria can be used to compute an approximation of the minimum user tree.

Figure 4 illustrates our algorithm that, given an authorization policy represented through an access matrix $\mathscr{A}$, creates a user tree correctly enforcing the policy. The algorithm creates the set V of material vertices and builds a graph G, where the set of vertices coincides with the set V of material vertices and the set E' of edges includes an edge (v_i,v_j) for each pair of vertices $v_i,v_j \in V$ such that $v_i.acl \subset v_j.acl$. The algorithm then calls function **Minimum_Spanning_Tree**[2] on G and vertex v_0, with $v_0.acl = \emptyset$, and returns a minimum spanning tree of G rooted at v_0. On such a minimum spanning tree, the algorithm calls function **Factorize_Internal_Vertices**. Function **Factorize_Internal_Vertices** takes a minimum spanning tree ST and a selection criterion as input and returns a minimal user tree. For each internal vertex v in ST (first **for** loop in the function), the function first computes the set CC_v of pairs of candidate children of v and determines the maximum reduction *max_red* of the weight of the tree that any of these pairs can cause. At each iteration of the **while** loop, the function selects, according to the given criterion, a pair $\langle v_i,v_j \rangle$ in CC_v such that *weight_red*(v,v_i,v_j) is equal to *max_red*. The tree is then updated as illustrated in Fig. 3. Then, both CC_v and *max_red* are re-evaluated on the basis of

[2] This function may correspond to any algorithm commonly used for computing a minimum spanning tree. Our implementation is based on Prim's algorithm.

Number of resources	**5 users**				**6 users**				**10 users**			
	I_{tot}	$I_{\min}$	$I_{\max}$	I_{rnd}	I_{tot}	$I_{\min}$	$I_{\max}$	I_{rnd}	I_{tot}	$I_{\min}$	$I_{\max}$	I_{rnd}
5	937	932	924	927	865	863	830	834	828	802	692	709
10	879	872	849	849	778	693	648	657	709	633	219	269
15	947	946	936	936	735	720	637	634	729	685	168	205
20	987	983	979	982	780	751	671	685	717	626	118	120
25	1000	998	998	998	781	763	705	714	694	598	90	131
30	1000	1000	1000	1000	846	835	808	815	626	543	77	131
35					891	886	853	858	554	484	64	104
40					943	940	924	928	570	538	59	85
45					981	978	966	973	501	488	57	68
50					993	992	989	991	501	478	55	67

Fig. 6 Number of times that our heuristics are better than the heuristic in [5]

the new topology of the tree. Note that CC_v does not need to be recomputed at each iteration of the **while** loop, since it can be simply updated by removing the pairs involving v_i and/or v_j and possibly adding the pairs resulting from the new vertex v_k. The process is repeated until CC_v becomes empty. The function terminates when all internal vertices have been evaluated (i.e., when the **for** loop has iterated on all internal vertices). As an example, consider the authorization policy $\mathscr{A}$ in Fig. 1(a). The table in Fig. 5 is composed of three columns, one for each of the preference criteria defined for our heuristic (i.e., I_{rnd}, $I_{\max}$, and $I_{\min}$). Each column represents the user tree and the user key rings computed by our heuristic, following one of the three preference criteria. Note that the vertices inserted by the algorithm are circled in Fig. 5, to distinguish material from non-material vertices.

5 Experimental results

A correct evaluation of the performance of the proposed heuristics is requested to provide the system designer with a valid set of tools she can use for the selection of the right strategy to implement a given authorization policy. In large scale access control systems, where the number of users and resources is large, the time needed to set the right key assignment scheme can be considerably large. So, the analysis we provide can help the designer to select the right trade-off between the quality of the solution returned by the selected heuristic and the amount of time invested on obtaining such a result. The heuristics have then been implemented by using Scilab [12] Version 4-1 on Windows XP operating system on a computer equipped with Centrino 1,7 Mhz CPU. We ran the experiments on randomly generated access matrices, considering different numbers of users and resources in the system.

A first set of experiments, whose results are reported in Fig. 6, has been devoted to compare the quality of the solutions returned by the different heuristics. For a fixed number of users and resources, we generated 1000 access matrices for each trial and applied to the resulting access matrix the heuristic proposed in [5] and our heuristics, considering all the possible choices (i.e., $I_{\min}$, $I_{\max}$, or I_{rnd}) for the selection of a candidate pair among the pairs maximizing the *weight_red* function. Columns

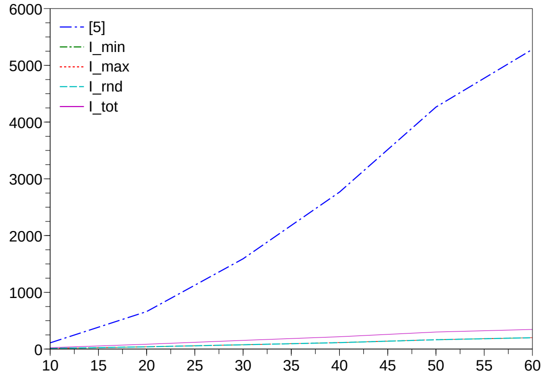


Fig. 7 Execution time (in seconds) for the heuristics for 10 users (1000 runs)

I_{min}, I_{max}, and I_{rnd} list the number of times the selected heuristic computes a user tree better than the user tree obtained by running the heuristic in [5], meaning that the total number of keys, in the key rings of the users, computed by our heuristic is less than or equal to the total number of keys in the key rings of the users obtained with the heuristic in [5]. Column I_{tot} lists the number of times that any of our heuristics returns a better solution than one returned by the heuristic in [5]. Note that while the I_{min} heuristic returns a better solution in most of the cases, there are cases where I_{max} or I_{rnd} perform better. On the basis of the data reported in Fig. 6, it is possible to observe the good behavior of our heuristics in the sense that they compute a solution that, in many cases, is better than the one returned by the heuristic in [5].

Figure 7 reports the sum of the execution times for all the considered instances. Note that the lines representing the I_{max}, I_{min}, and I_{rnd} heuristics are overlapping. For each instance (i.e., each randomly generated access matrix), the execution time is composed of the time for the construction of the graph G (see Sect. 4), the time for the construction of the minimum spanning tree on G, and the time for the execution of the selected heuristic. As shown in the figure, our heuristics are very efficient compared with the heuristic in [5]. Considering that in many cases, such heuristics return a better solution than the one computed by the heuristic in [5], we can conclude that they represent a good trade-off between quality and execution time. Also, since our heuristics are fast to execute, after graph G and the corresponding minimum spanning tree have been generated, it should be also possible to execute all our heuristics to select the best of the three returned results (without need of generating the graph and the MST again).

6 Conclusions

There is an emerging trend towards scenarios where resource management is outsourced to an external service providing storage capabilities and high-bandwidth distribution channels. In this context, selective dissemination of data requires enforcing measures to protect the resource confidentiality from both unauthorized users as well as honest-but-curious servers. Current solutions provide protection by exploiting encryption in conjunction with proper indexing capabilities, and by exploiting selective encryption for access control enforcement. In this paper we proposed a heuristic algorithm for building a key derivation graph that minimizes the total number of keys to be distributed to users in the system. The experimental results obtained by the implementation of the algorithm prove its efficiency with respect to previous solutions.

Acknowledgements This work was supported in part by the EU, within the 7FP project, under grant agreement 216483 “PrimeLife” and by the Italian MIUR, within PRIN 2006, under project 2006099978 “Basi di dati crittografate”.

References

1. Akl S, Taylor P (1983) Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer System*, 1(3):239–248
2. Atallah MJ, Frikken KB, Blanton M (2005) Dynamic and efficient key management for access hierarchies. In *Proc. of the ACM CCS05*, Alexandria, VA
3. Ateniese G, De Santis A, Ferrara AL, Masucci B (2006) Provably-secure time-bound hierarchical key assignment schemes. In *Proc. of ACM CCS06*, Alexandria, VA
4. Ceselli A, Damiani E, De Capitani di Vimercati S, Jajodia S, Paraboschi S, Samarati P (2005) Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152
5. Damiani E, De Capitani di Vimercati S, Foresti S, Jajodia S, Paraboschi S, Samarati P (2006) Selective data encryption in outsourced dynamic environments. In *Proc. of VODCA 2006*, Bertinoro, Italy
6. De Capitani di Vimercati S, Foresti S, Jajodia S, Paraboschi S, Samarati P (2007) Over-encryption: Management of access control evolution on outsourced data. In *Proc. of VLDB 2007*, Vienna, Austria
7. De Santis A, Ferrara AL, Masucci B (2004) Cryptographic key assignment schemes for any access control policy. *Information Processing Letters*, 92(4):199–205
8. Hacigümüs H, Iyer B, Mehrotra S, Li C (2002) Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD 2002*, Madison, WI
9. MacKinnon S, Taylor P, Meijer H, Akl S (1985) An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE TC*, 34(9):797–802
10. Miklau G, Suciu D (2003) Controlling access to published data using cryptography. In *Proc. of VLDB 2003*, Berlin, Germany
11. Sandhu RS (1988) Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98
12. Scilab Consortium Scilab, the open source platform for numerical computation. http://www.scilab.org, V. 4-1
13. Wang H, Lakshmanan LVS (2006) Efficient secure query evaluation over encrypted XML databases. In *Proc. of VLDB 2006*, Seoul, Korea