Key Management for Multi-User Encrypted Databases

Ernesto Damiani
DTI - Università di Milano
26013 Crema - Italy
damiani@dti.unimi.it

S.De Capitani di Vimercati
DTI - Università di Milano
26013 Crema - Italy
decapita@dti.unimi.it

Sara Foresti
DTI - Università di Milano
26013 Crema - Italy
foresti@dti.unimi.it

Sushil Jajodia
George Mason University
Fairfax, VA 22030-4444
jajodia@gmu.edu

Stefano Paraboschi
DIGI - Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

ABSTRACT

Database outsourcing is becoming increasingly popular introducing a new paradigm, called *database-as-a-service* (DAS), where an organization's database is stored at an external service provider. In such a scenario, access control is a very important issue, especially if the data owner wishes to publish her data for external use.

In this paper, we first present our approach for the implementation of access control through selective encryption. The focus of the paper is then the presentation of the experimental results, which demonstrate the applicability of our proposal.

Categories and Subject Descriptors: H.2.1 [Database Management]: Logical Design; H.2.7 [Database Management]: Database Administration

General Terms: Security, Management, Experimentation.

Keywords: Encrypted/indexing databases, selective access, hierarchical key derivation schema.

1. INTRODUCTION

The amount of sensitive information held by organizations' databases is increasing very quickly and these data have to be protected from unauthorized uses. The management of large databases is quite expensive, as it needs not only storage capacity, but also skilled personnel. An emerging solution to this problem is represented by database outsourcing, that is, delegating database management to a third party. In such a solution, called *database as a service* (DAS), an organization's database is stored at an external service provider that should provide mechanisms for clients to access the outsourced databases. The main advantage of the outsourcing solution is twofold. First, it provides significant cost savings and service benefits. Second, it promises higher availability and more effective disaster protection than inhouse operations.

The main problem in outsourcing data to external service providers is that sensitive data become stored on a site that is not under the data owner's direct control. Therefore, data confidentiality and even integrity can be put at risk. In many contexts, confidentiality and integrity are managed by means of encryption [12]. By encrypting the data, the user can be sure that nobody, except her, can read the data. However, a trivial solution that asks the database to store only encrypted information does not work, because it leaves the external service unable to support selective access. Since confidentiality demands that data decryption must be possible only at the client side, techniques are needed to enable external servers to execute queries on encrypted data, otherwise all the relations involved in a query would have to be sent to the client for query execution.

Approaches towards the solution of this problem were presented in [8, 11, 14, 15, 17], where the authors proposed storing, together with the encrypted database, additional indexing information. Such indexes are used by the DBMS to select the data to be returned in response to a query, without need of decrypting the data themselves. Different indexing methods have been proposed, each one suitable for the remote execution of a particular kind of query. In [8, 11] the authors propose a hash-based method for database encryption suitable for selection queries. To execute interval-based queries, the B+-tree structures typically used inside DBMSs are adopted. Privacy homomorphism has also been proposed for allowing the execution of aggregation queries over encrypted data [16, 18]. In this case the server stores an encrypted table with an index for each aggregation attribute (i.e., an attribute on which the aggregate operator can be applied) and obtained from the original attribute with privacy homomorphism. An operation on an aggregation attribute can then be evaluated by computing the aggregation at the server and by decrypting the result at the client side. Other works on privacy homomorphism illustrate techniques for performing arithmetic operations (+, -, ×, /) on encrypted data and do not consider comparison operations [5]. In [1] an order preserving encryption schema (OPES) is presented to support equality and range queries over encrypted data.

StorageSS'05, November 11, 2005, Alexandria, Virginia, USA.
Copyright 2005 ACM 1-59593-223-X/05/0011 ...$5.00.

This approach operates only on integer values and the results of a query posed on an attribute encrypted with OPES is complete and does not contain spurious tuples.

Even if the DAS scenario has been studied in-depth in the last few years, there are new interesting research challenges that have to be investigated. In particular, the problem of guaranteeing an efficient mechanism for implementing selective access to the remote database is still an open issue. As a matter of fact, all the existing proposals for designing and querying encrypted/indexing outsourced databases assume the client has complete access to the query result. However, this assumption does not fit real world applications, where different users may have different access privileges. A trivial solution for implementing access control in the DAS scenario consists in the explicit definition of authorizations. The main drawback of this method is that the data owner has to intercept each reply message from the server to the client, to filter out all the tuples that the final user cannot access. In fact, this work cannot be delegated to the remote server which is not trusted to know the access control policy defined by the data owner. Such an approach may however cause a bottleneck, because it increases the processing and communication load at the data owner site. A promising direction to avoid such a bottleneck is represented by selectively encrypting data so that users (or groups thereof) can decrypt only the data they are authorized to access.

Recently, some approaches have been proposed for searching on encrypted data [3, 13, 25]. Basically, a *secure index data structure* is associated with each document and it allows a requestor with a *trapdoor* for a given keyword x to verify whether the index contains x. The index is computed using the public key of the requestor and the keyword x, and the trapdoor is computed using the private key of the requestor and the keyword x. An important feature of this approach is that it allows the server to retrieve all documents containing the keyword x without revealing any other information. In [3] the authors propose different constructions for implementing this method and one is based on the *Identity Based Encryption* (IBE) [4], which is a public key cryptosystem where public keys can be arbitrary bitstrings, from which a trusted entity can extract the corresponding private keys.

In this paper, after a brief introduction of the structure of the DAS scenario (Section 2), we describe an approach for the implementation of access control through selective encryption (Section 3). Our solution is based on a hierarchical structure, used for key derivation, reflecting the access control policy defined by the data owner. The focus of the paper is then in the presentation of the experimental results which demonstrate the performance of the variants of our approach in a system with a number of subjects and objects hierarchically organized (Section 4).

2. BASIC CONCEPTS AND SCENARIO

The DAS scenario involves mainly four entities (see Figure 1):

- *Data owner*: an organization that produces data to be made available for controlled external release;
- *User*: human entity that presents requests (queries) to the system;
- *Client*: front-end that transforms the user queries into queries on the encrypted data stored on the server;
- *Server*: an organization that receives the encrypted data from a data owner and makes them available for distribution to clients.

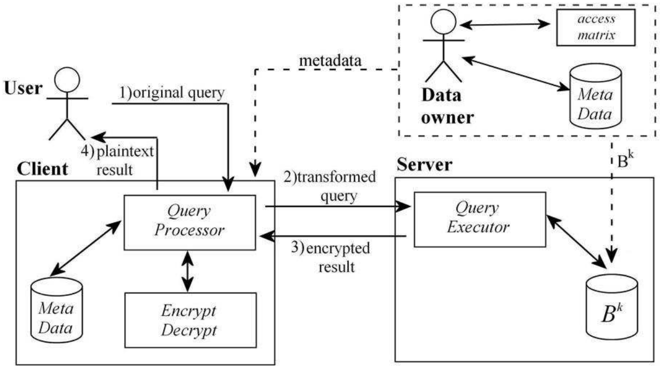

Figure 1: DAS Scenario

TeamNews

IdTeam	Name	Foundation Year	Budget	League
01	team1	1970	15.000	Baseball
02	team2	1986	16.000	Basketball
03	team3	1974	15.000	Baseball
04	team4	1977	16.000	Football
05	team5	1972	18.000	Basketball
06	team6	1981	16.000	Baseball
07	team7	1979	18.000	Football

Figure 2: An example of plaintext relation

TeamNews$^{\mathbf{k}}$

Counter	Etuple	IdKey	$\mathbf{I}_1$	$\mathbf{I}_2$	$\mathbf{I}_3$	$\mathbf{I}_4$	$\mathbf{I}_5$
t_1	r*tso/yui+	BC	α	γ	μ	π	λ
t_2	hai4de-0q1	BD	β	δ	η	ρ	θ
t_3	nag+q8*L	ACD	α	γ	μ	π	λ
t_4	K/ehim*13-	BCD	β	ε	η	ρ	θ
t_5	3gia*ni+aL	C	α	δ	μ	π	θ
t_6	F0/rab1DW*	ABC	β	γ	η	ρ	λ
t_7	Bid2*k1-l0	AB	β	ε	η	π	θ

Figure 3: An example of encrypted relation

Clients and data owners are assumed to trust the server to faithfully maintain outsourced data. Specifically, the server is relied upon for the availability of outsourced databases. However, the server is assumed not to be trusted with the confidentiality of the actual database content. That is, we want to preserve the server from making unauthorized access to the data stored in the database. To this purpose, the data owner encrypts her data and gives the encrypted database to the server. The end users, instead, are trusted to access the database, according to the data owner's policy.

Note that database encryption may be performed at different levels of granularity: relation level, attribute level, tuple level, and element level. To balance the client workload and query execution efficiency, consistently with previous proposals [11, 17], we assume that the database is encrypted at tuple level.

The main effort of current research in this scenario is the design of a mechanism that makes it possible to directly query an encrypted database [14]. The existing proposals are based on the use of indexing information associated with each relation in the encrypted database [11, 17]. Such indexes can be used by the server to select the data to be returned in response to a query. More precisely, the server stores an encrypted table with an index for each attribute on which a query can include a condition. Different types of indexes can be defined, depending on the supported queries. For instance, hash-based methods are suitable for equality queries [17, 20] and B+-tree based methods support range queries [11]. For simplicity, we assume that there is an index for each attribute in each relation. Formally, each relation $\mathtt{r_i}$ over schema $\mathtt{R_i(A_{i1}, A_{i2}, \ldots, A_{in})}$ in a plaintext database $\mathtt{B}$ is mapped onto a relation $\mathtt{r_i^k}$ over schema $\mathtt{R_i^k(\underline{Counter}, Etuple, IdKey, I_1, I_2, \ldots, I_n)}$ in the encrypted database $\mathtt{B^k}$ where: `Counter` is the primary key; `Etuple` is an attribute for the encrypted tuple whose value is obtained using an encryption function E_k (k is the key); $\mathtt{I_i}$ is the index associated with the $\mathtt{i}$-th attribute. For instance, given relation `TeamNews` in Figure 2, the corresponding encrypted relation is represented in Figure 3. (Here, the result of the hash function is represented as a Greek letter. Also, note attribute `IdKey` does not belong to current proposals, but it is inserted for our solution. Its management and semantics will be discussed in Section 3.) As it is visible from this table, the encrypted table has the same number of rows as the original one. The query processing is then performed as follows (see Figure 1): (1) each query is mapped onto a query on encrypted data and (2) it is then sent to the server. The result of this query is a set of encrypted tuples (3) that are then processed by the client front-end to decrypt data and discard spurious tuples that may be part of the result. The final result (4) is then presented to the user. Note that this process is based on catalogs stored at the client side that describe the structure of the remote database [9].

3. ACCESS CONTROL IN THE DAS SCENARIO

The existing proposals for designing and querying encrypted/indexing outsourced databases focus on the challenges posed by protecting data at the server side, and assume the client has complete access to the query result [6, 7, 17, 23]. Therefore, tuples are encrypted using a single key and the knowledge of the key grants complete access to the whole database. Clearly, such an assumption does not fit real world applications, where the data owner often requires to enforce access restrictions to different users, sets of users, or applications. We then propose to exploit data encryption by including authorizations in the encrypted data themselves. While it is in principle advisable to leave authorization-based access control and cryptographic protection separate, in the DAS scenario such a combination can prove successful. The idea is to use different encryption keys for different data. To access such encrypted data, users have to decrypt them, which requires knowledge of the encryption algorithm and of the specific decryption key being used. If the access to the decryption keys is differentiated on the users' identity, different users are given different access rights. In classical terms, the access rights defined by the data owner can be represented by using an *access matrix* $\mathcal{A}$, where rows correspond to subjects, columns correspond to objects, and entry $\mathcal{A}[s, o]$ is set to 1 if s has permission to access o; 0 otherwise.[1]

	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$	$\mathbf{t_7}$
Alice	0	0	1	0	0	1	1
Bob	1	1	0	1	0	1	1
Carol	1	0	1	1	1	0	0
David	0	1	1	1	0	1	0

Figure 4: An example of access matrix

Given an access matrix $\mathcal{A}$, ACL_i denotes the vector corresponding to the i-th column (i.e., the access control list indicating the subjects that can read tuple t_i), and CAP_j denotes the vector corresponding to the j-th row (i.e., the capability list indicating the objects that user u_j can read). Let us consider a situation with four users, namely `Alice`, `Bob`, `Carol`, and `David` (which in the following we abbreviate as A, B, C, and D, respectively); they are supporters of different teams and need to read the tuples of relation `TeamNews`. Figure 4 illustrates an example of access matrix. With a slight abuse of notation, in the following we will use ACL_i (CAP_j, respectively) to denote either the bit vector corresponding to a column (a row, respectively) or the set of users (tuples, respectively) whose entry in the access matrix is 1. With reference to the matrix in Figure 4, ACL_1 denotes both the bit vector [0110] and the set of users {`Bob`, `Carol`}; while CAP_C denotes both the bit vector [1011100] and the set of tuples $\mathtt{t_1}$, $\mathtt{t_3}$, $\mathtt{t_4}$, and $\mathtt{t_5}$.

A straightforward solution for implementing access control through cryptography consists in encrypting each tuple in the outsourced database with a different key and assigning to each user the set of keys associated with the tuples she can access. However, this simple solution is not efficient and requires the management of too many keys. For instance, with respect to the access matrix in Figure 4, user `Carol` should receive the keys used for encrypting tuples $\mathtt{t_1}$, $\mathtt{t_3}$, $\mathtt{t_4}$, and $\mathtt{t_5}$. We propose a different method that consists in grouping users with the same access privileges and in encrypting each tuple (or group) with the key associated with the set of users that can access it. To this purpose, we consider a *user hierarchy* whose elements are all the possible sets of users in the system together with the partial order naturally induced on it by the subset containment relationship. More precisely, the user hierarchy is defined as follows.

Definition 1. (User Hierarchy) *Given a set $\mathcal{U}$ of users, a* user hierarchy*, denoted UH, is a pair* $(\mathrm{P}(\mathcal{U}), \preceq)$*, where* $\mathrm{P}(\mathcal{U})$ *is the power set of $\mathcal{U}$ and $\preceq$ is a partial order on* $\mathrm{P}(\mathcal{U})$ *such that $\forall X,\ Y \in \mathrm{P}(\mathcal{U})$, $X \preceq Y$ iff $Y \subseteq X$.*

We consider each user group has associated the tuples whose ACL, defined in the access matrix, corresponds to the group itself. With respect to our example in Figure 4, tuple $\{\mathtt{t_4}\}$ is associated with group BCD (corresponding to the set of users `Bob`, `Carol`, and `David`), while the set $\{\mathtt{t_1}, \mathtt{t_4}\}$ of tuples are associated with group BC. It is then straightforward to see how the partial order relationship between user groups implies a partial order relationship between the access rights associated with the set of users corresponding to the groups. With respect to the above example, users `Bob`

[1] Generally speaking, the entry should be the list of privileges that s has on o. Since we are only interested in read operations, we assume a boolean value indicates the presence or absence of the permission.

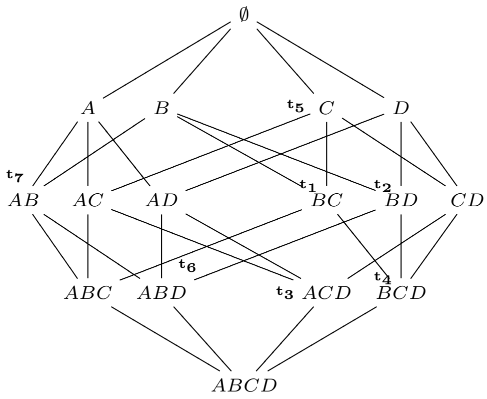

Figure 5: An example of user hierarchy

and Carol can read tuples t_1 and t_4 and users Bob, Carol, and David can read tuple t_4.

The user hierarchy can be graphically represented via a directed acyclic graph (DAG) with a node for each set of users, and an edge from node Y to node X if $X \preceq Y$. Figure 5 illustrates the graphical representation of the user hierarchy constructed on the set {Alice, Bob, Carol, David} of users. The nodes of the hierarchy are labeled with the tuple whose *ACL* coincides with the node, as specified in the access matrix in Figure 4.

By assuming that each node of the hierarchy is associated with a key, the access control problem could be solved by encrypting each tuple with the key of the node corresponding to its *ACL* (e.g., tuple t_1 should be encrypted with key K_{BC}) and by assigning to each user the set of keys of the nodes to which she belongs. Attribute IdKey in the encrypted table contains the label of the node whose key is used to encrypt the specific tuple (e.g., t_1 is associated with BC). The advantage of this solution, with respect to the trivial one above-mentioned, is that potentially a key can be used to encrypt more than one tuple. The disadvantage is that this solution involves the assignment of many keys to each user. For instance, according to the access matrix in Figure 4, user Carol needs to know keys K_C, K_{BC}, K_{ACD}, and K_{BCD}. We therefore propose an alternative solution that exploits *the key derivation methods* operating on DAGs [2, 19, 21, 22]. Basically, these methods operate on the hierarchy computing the keys of lower-level nodes based on the keys of their predecessors. In our context, these methods allow us to reduce the number of keys that need to be directly communicated to each user. For instance, by applying one of these derivation methods, user Carol needs to know only key K_C, from which she can derive K_{BC}, K_{ACD}, and K_{BCD} (these keys are needed to decrypt the tuples in CAP_C). However, the key derivation schemes working on DAGs are complex and require a lot of key storage (whose size grows exponentially with the nodes of the hierarchy). To avoid these problems, we transform the DAG into an equivalent tree structure and adopt simpler key management techniques that work on trees. The method described in [24] is based on a family of *one-way functions* usually implemented through an encryption function (i.e., $f_p(x) = E_x(p)$). This method assumes that each node n_i in the tree has a name, $name(n_i)$, and a key K_i. The value of each K_i is computed as $K_i = f_{name(n_i)}(K_j)$, where n_j is the unique parent of n_i. As the root node has no parent, its key is randomly chosen. Note that with this key derivation method two siblings cannot have the same name because otherwise they would have the same key. A user that knows a key K_i can compute only the keys in the subtree rooted at node n_i. The number of steps necessary to derive a key is therefore equal to the length of the path connecting the node of which the key is known to the node of which the key needs to be computed.

```
Algorithm 1   (DAG to tree transformation).

Transformation($\mathcal{A}$)
Step 1: Identification of candidate nodes
    $M$ = FindMaterialNodes($\mathcal{A}$)
    $NM$ = FindNonMaterialNodes($M$)

Step 2: Identification of edges
    $A$ = SetParent($M \cup NM$) /* Select a parent for each node */
    $T$ = ($M \cup NM$, $A$)

Step 3: Tree pruning
    $T$ = PruneTree($T$)
    return ($T$)
```

Figure 6: DAG to tree transformation algorithm

The DAG to tree transformation process implies the assignment of more than one key to each user: the tree is obtained by removing some paths of the DAG and therefore some keys that the user could derive on the DAG are no more derivable on the tree. For instance, consider the node ABD in the *UH* in Figure 5, which is connected with AB, AD, and BD. Suppose that after the DAG to tree transformation process node ABD is only connected with AB. In this case, David cannot derive key K_{ABD} that is necessary to access tuple t_6 and keys K_D and K_{ABD} have to be directly communicated to David.

On the one hand, the DAG to tree process allows us to work with a more simple key derivation method. On the other hand, more information have to be stored at the client side and the data owner has to spend much more time in communicating and managing the keys. Therefore, the the main objective of the transformation process is the minimization of the number of keys that have to be produced. Unfortunately, transforming a DAG into a tree in such a way to minimize the number of keys to be produced is an NP-hard problem [10]. This result forces us to design a heuristic algorithm. The goal of this paper is to show the design of the heuristic algorithm and investigate its behavior, in terms of the quality of its solution and the time required for its execution. To this end, we define a scenario that allows us to test the amount of resources required to assign keys to every user and the time required to compute the solution. The results of the experiments in Section 4 show that the heuristic algorithm can be applied even in scenarios with a considerable number of users.

3.1 DAG to Tree Transformation Algorithm

As discussed in the previous Section, our algorithm is built with the aim to minimize the number of keys to be communicated to each user. More precisely, the metric we want to minimize is the average number of keys that users have to know to access the tuples for which they have an access right.

At a high level, the algorithm implementing our transformation consists of three main steps (see Figure 6). We now

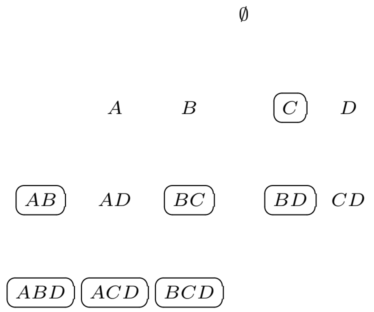

Figure 7: Material and non material nodes corresponding to the DAG in Figure 5

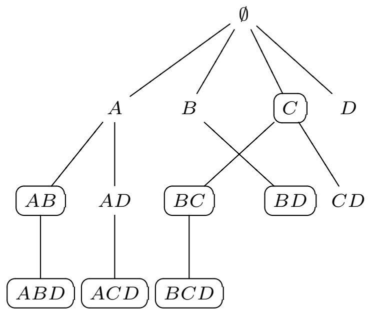

Figure 8: Tree corresponding to the DAG in Figure 5 before the pruning step

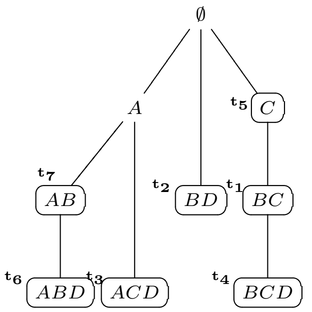

Figure 9: Tree corresponding to the DAG in Figure 5 after the pruning step

discuss the different steps.

Step 1: Identification of candidate nodes

We select a subset of the DAG's nodes, called *material* (*M*) and *non material* (*NM*) nodes. The material nodes correspond to actual *ACLs* and must belong to the tree because their keys are used to encrypt the associated tuples. For instance, with respect to the user hierarchy in Figure 5, the set M of material nodes is $\{C, AB, BC, BD, ABD, ACD, BCD\}$. The non material nodes are the nodes that can be used to reduce the number of keys assigned to users. In our example, the set *NM* of non material nodes is $\{A, B, D, AD, CD\}$. As shown in [10], non material nodes can be computed recursively by considering the fix-point of the computation of the parents of two material or non material nodes. Note that the final set of material and non material nodes is closed under the intersection operation.

Figure 7 illustrates the set $M \cup NM$ of material and non material nodes corresponding to the DAG in Figure 5. To distinguish between the two kinds of nodes, material nodes are circled.

Step 2: Identification of edges

The second step of the algorithm consists in selecting, for each node in $M \cup NM$, one parent, in order to obtain a tree. This step is performed by noting that each node in the DAG appears at a certain *level* and that an edge connects nodes of adjacent levels. A level contains all nodes corresponding to the sets of users with the same cardinality. That is, level 0 contains the empty set, level 1 contains nodes corresponding to single users, level 2 contains nodes corresponding to pairs of users, and so on. For instance, the user hierarchy in Figure 5 has five levels $(0 \dots 4)$ and level 3 contains nodes ABC, ABD, ACD, and BCD. The selection of the parent nodes is performed level-by-level (left to right), starting from the highest level to level 2 (by definition the parent of nodes at level 1 is the empty set). The selection of the parent node can be performed according to different criteria. We will use the experiments to evaluate the contribution that each criterion gives to the quality of the solution, compared with the increase in the execution time required for its application. The criteria are the following.

C1. Choose randomly a candidate parent node located in the lowest level above the node that presents a material or non material ancestor of the node.

C2. Choose a candidate parent node located in the lowest level above the node that presents a material or non material ancestor of the node, giving priority to material nodes. That is, if there are material and non material candidate ancestors, choose randomly among the material nodes, otherwise choose randomly among the non material nodes.

C3. Choose a candidate parent node located in the lowest level above the node that presents a material or non material ancestor of the node, giving priority to material nodes and giving priority, among non material nodes, on the nodes with exactly one child and then on the nodes with the greatest number of children. That is, if there are material and non material candidate ancestors, choose randomly among the material nodes, otherwise choose randomly among the non material nodes with one child and, if such nodes do not exist, choose randomly among the non material nodes with the greatest number of children.

For instance, node BCD in Figure 7 has the following set of candidate parent nodes: $\{B, C, D, BC, BD, CD\}$. The candidate parent node in the lowest level is BC and it is easy to see that this material node is preferable to other material nodes. For instance, if the material node C is selected as parent of node BCD, Bob should directly know keys K_{BC} and K_{BCD}. If node BC is then selected, Bob should directly know only key K_{BC}. As another example, consider node AD for which the candidate parent nodes are the non material nodes A and D. Node A has one child and therefore, according to Criterion 3, it is selected as parent of node AD. Intuitively, the rationale behind the preference of

Criterion 3 to non material nodes with children is that a non material node is useful only if it allows to "factorize" keys, that is, if it allows to assign to a user a higher level key that can permit to derive more than one lower level key, saving on the number of keys that the user needs to receive. In our example, `Alice` with the knowledge of key K_A is able to derive the pair of keys K_{AB} and K_{AD}. Figure 8 illustrates the tree obtained at the end of this second step.

Step 3: Tree pruning

The third step consists in a tree pruning, where (1) non material tree nodes with only one child and (2) non material tree leaves are removed from the tree. In the first case, an alternative parent node needs to be chosen for the child. This selection is performed by applying the following two criteria.

P1. Choose in the level of the removed node a non material node with at least one child.

P2. If there is no alternative candidate parent node in the level of the removed node, connect the child with the parent of the removed node.

For instance, consider the tree in Figure 8. Node AD is removed and node A is selected as parent of node ACD because there are no candidate parent nodes in level 2.

It is important to note that the pruning is performed first by removing the non material nodes with only one child and then by removing any leaf non material node. Figure 9 illustrates the tree obtained after the pruning.

The resulting tree is then used to enforce the access control policy specified in the original access matrix. More precisely, each tuple is encrypted with the key associated with the node in the tree that corresponds to its ACL. The data owner communicates to each user u the key associated with node V in the tree such that user $u \in V$ and $u \notin W$, where W is the parent of V in the tree. For instance, with respect to the tree in Figure 9, `Carol` should know keys $\{K_C, K_{BC}, K_{ACD}, K_{BCD}\}$. However, the keys that must be directly communicated to `Carol` are only $\{K_C, K_{ACD}\}$ because key K_C can be used to derive K_{BC}, which in turn can be used to derive K_{BCD}.

The cost of our transformation algorithm is $O(|NM \cup M|^2)$, where NM is the set of non material nodes initially inserted into the graph and M is the set of material nodes. The cost is therefore at most quadratic in the number of nodes computed in the first step of our algorithm. Since the tree-minimization problem is NP-hard [10], the algorithm that guarantees to identify the optimal tree has an exponential complexity (as usual, if $P \neq NP$).

4. PERFORMANCE EVALUATION

We describe the results of the experiments that we have conducted to evaluate the performance of the DAG to tree transformation algorithm. We consider as parameters of the experiments the number $\mathcal{U}$ of users in the system, the number $\mathcal{T}$ of resources, and a set of authorizations specified in an access matrix $\mathcal{A}$. In particular, we consider a network accessible sport news database, with t *teams* of pt *players* and s *subscribers* (i.e., team supporters). The championship is also followed by a number of *writers*, each working with tw teams. The writers are grouped into sets of wm elements and one manager is assigned to each set. More precisely, the set $\mathcal{U}$ of users is partitioned into the following categories:

- *Players*: $P_1 \ldots P_p$, where $p = t \cdot pt$;
- *Team Managers*: $TM_1 \ldots TM_t$, where t is the number of teams in the system;
- *Writers*: $W_1 \ldots W_w$, where $w = \lceil t/tw \rceil$;
- *Managers of writers*: $WM_1 \ldots WM_m$, where $m = \lceil w/wm \rceil$;
- *Subscribers*: $S_1 \ldots S_s$.

The system is then composed of $|\mathcal{U}| = p + t + w + m + s$ users. Analogously, the set $\mathcal{T}$ of tuples in the database is partitioned into two subsets:

- *Player News*: $PN_1 \ldots PN_p$ (tuples describing players);
- *Team News*: $TN_1 \ldots TN_t$ (tuples describing teams).

The system is then composed of $|\mathcal{T}| = p + t$ resources. Finally, the authorizations defined in the system are the following.

- $\mathcal{A}[P_i, PN_i] = 1, i = 1 \ldots p$: each player can access her player news;
- $\mathcal{A}[TM_i, TN_i] = 1, i = 1 \ldots t$: each team manager can access the team news of her team;
- $\mathcal{A}[TM_i, PN_j] = 1, i = 1 \ldots t, j = ((i-1) \cdot pt + 1) \ldots (i \cdot pt)$: each team manager can access the player news of all the players of her team;
- $\mathcal{A}[TM_{2i}, TN_j] = 1, i = 1 \ldots t/2,\ j = 1 \ldots t$: $t/2$ team managers can access all the team news of the championship;
- $\mathcal{A}[W_i, TN_j] = 1, i = 1 \ldots w, j = ((i-1) \cdot tw + 1) \ldots (i \cdot tw)$: each writer can access the team news for the teams she follows;
- $\mathcal{A}[W_i, PN_l] = 1, i = 1 \ldots w, j = ((i-1) \cdot tw + 1) \ldots (i \cdot tw), l = ((j-1) \cdot pt + 1) \ldots (j \cdot pt)$: each writer can access the player news for the teams she follows;
- $\mathcal{A}[WM_i, TN_l] = 1, i = 1 \ldots m, j = ((i-1) \cdot wm + 1) \ldots (i \cdot wm), l = ((j-1) \cdot tw + 1) \ldots (j \cdot tw)$: each manager of writers can access the team news accessible by the writers she follows;
- $\mathcal{A}[WM_i, PN_h] = 1, i = 1 \ldots m, j = ((i-1) \cdot wm + 1) \ldots (i \cdot wm), l = ((j-1) \cdot tw + 1) \ldots (j \cdot tw),\ h = ((l-1) \cdot pt + 1) \ldots (l \cdot pt)$: each manager of writers can access the player news accessible by the writers she follows;
- subscribers can access team news and the distribution of the authorizations associated with subscribers follow a Zipf distribution, which is shown by a considerable amount of experience to represent well the behavior of many real systems.

The access matrix $\mathcal{A}$ has $|\mathcal{U}|$ rows, one for each user, and $|\mathcal{T}|$ columns, one for each resource. The resulting user hierarchy has $|\mathcal{U}|$ levels and $2^{|\mathcal{U}|}$ nodes.

4.1 Parameter Setting

In our experiments we considered two different scenarios. The first scenario (S1) has been described in the previous section and is characterized by two sets of resources (i.e., *Team News* and *Player News*) and five categories of users (i.e., *Players*, *Team Managers*, *Writers*, *Managers of writers*, and *Subscribers*). In the second scenario (S2), we consider one set of resources (i.e., *Team News*) and four categories of users (i.e., *Team Managers*, *Writers*, *Manager of writers*, and *Subscribers*). Note that in S2 *Players* are not considered because they do not have any access right on *Team News*.

With respect to the DAG to tree transformation algorithm, we implemented three different versions, one for each criterion (**C1**, **C2**, and **C3**) defined for the selection of an appropriate parent node (see Section 3.1). An additional variation that has been considered is related to the tree pruning phase. As described in the previous Section, when a non material node is removed from the tree, a new parent node needs to be chosen for the child of the removed node. This selection has been implemented in two different ways.

Pr1. The new parent coincides with the parent of the removed node;

Pr2. A candidate parent is chosen from the non material nodes with at least one child and at the same level of the removed node; if such a node does not exist, criterion **Pr1** is applied.

The combination of the above-mentioned criteria produces six different configurations for each of the two scenarios S1 and S2. We then ran experiments for the following different cases: 30, 50, 70 number t of teams; 100, 500, 1000, 1500 number s of subscribers; 20 (pt) players; 5 (wm) writers per manager; and 3 (tw) teams followed by each writer.

4.2 Results and Considerations

There are different indicators that may be evaluated in the algorithm testing phase. The most important one is the average number of keys assigned to each user, as this indicates the quality of the tree construction criteria implemented: the lower is the average number of keys each user has to manage, the better is the solution obtained.

Another element that should be considered in evaluating and comparing the different versions of the algorithm proposed is the number of nodes in the final tree. In particular, it is relevant the ratio between material and non material nodes in the structure.

Finally, the last measure evaluated with our experiments is the computational time of the algorithm which is an important parameter for the evaluation of its usability. More precisely, our experiments evaluated the trade-off between computational costs and quality of the solution found. Intuitively, a better solution is obtained when a more refined criterion is adopted and this certainly require a higher computational time.

In the following, we analyze the results of our experiments carried out on a 1500 MHz Pentium IV with 256 Mb of RAM. Note that for clarity and readability of the graphs, we report the results of only four configurations, that is, t=30 and s=100, t=70 and s=100, t=30 and s=1500, and t=70 and s=1500.

Evaluation of the average number of keys.

As the results of our experiments show (Figure 10), the average number of keys decreases if the criterion adopted for the choice of the parent improves (from **C1** to **C3**), as we would expect. In particular, the improvement is greater between Criterion **C2** and Criterion **C3**. In the same way, adopting a more refined procedure for tree pruning, the final result is better than adopting the traditional method. On the basis of these considerations, we can observe that the best solution, using the same parameter setting, is obtained with the most refined Criterion **C3** and adopting the improved pruning method **Pr2**. On the contrary, the worst solution is the one obtained adopting the basic strategies both for tree construction and pruning. Also, it is easy to see that the average number of keys increases with the number of subscribers because it is more difficult to obtain a good solution if the number of users and authorizations increases. By contrast, if the number of teams increases, the average number of keys decreases because we add to the system a set of resources (*Team News*) characterized by similar *ACLs* and a set of users (*Team Manager*) that receive the same set of keys. Comparing the results obtained in the two scenarios S1 and S2 (Figure 10(a) and Figure 10(b)), we observe that the average number of keys is lower in scenario S1 than in scenario S2. The reason is that in S1 each *Player* has exactly one authorization on her news and exactly one key. *Players* therefore contribute to bring down the average number of keys.

Evaluation of the number of nodes.

By definition, each resource in the system has a different ACL and therefore the number of material nodes is equal to the number of resources and, in each scenario, it does not change as the other parameters vary. Also, the number of non material nodes initially inserted in the tree depends only on the set of material nodes. Consequently, after Step 1 of the DAG to tree transformation algorithm, the number of non material nodes is always the same. However, the application of the different criteria (**C1**, **C2**, **C3**, **Pr1**, **Pr2**) may produce a different tree with a different number of non material nodes. Our experiments show that better solutions (i.e., solutions with a low average number of keys) have more non material nodes (see Figure 11). The rationale behind this observation is that, as discussed in Section 3.1, non material nodes allow us to reduce the number of keys assigned to each user. Therefore, in the pruning step the useful non material nodes are not deleted and their number increases. Also, it is easy to see that the ratio between the number of non material and material nodes is higher in scenario S2 than scenario S1 (see Figure 11(a) and Figure 11(b)). The reason is that in scenario S2 there is a low number of material nodes and therefore to construct a tree a large number of non material nodes need to be considered. By contrast, in scenario S1 there are many material nodes and therefore a small number of non material nodes need to be considered.

Evaluation of the performance.

The results obtained demonstrate that the time taken by the DAG to tree transformation algorithm increases with the number of subscribers and/or teams (Figure 12) and is obviously higher in scenario S1 (Figure 12(a) and Figure 12(b)). It is also important to note that the time taken by the algorithm is higher when the choice criterion is more refined

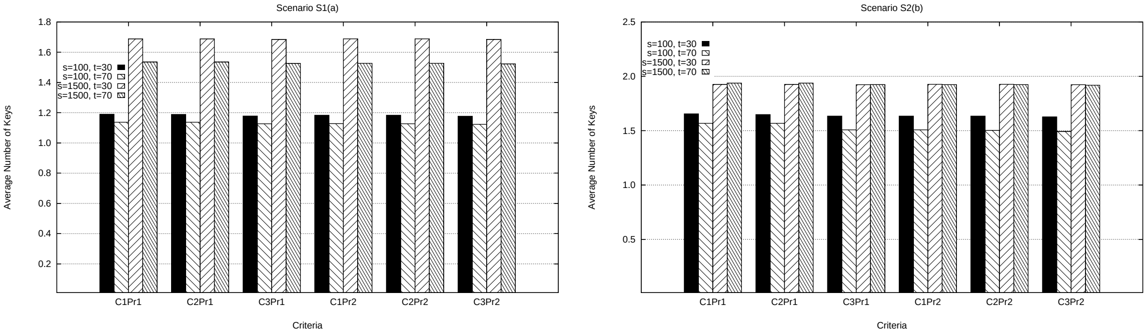

Figure 10: Average number of keys in scenario S1 (a) and scenario S2 (b)

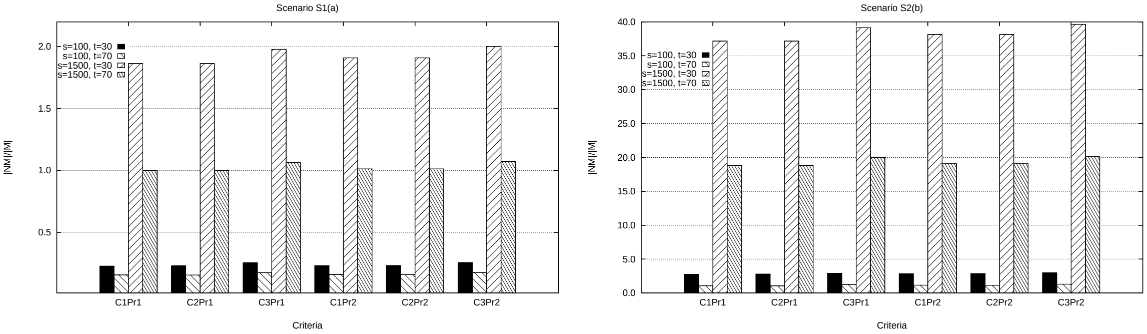

Figure 11: Ratio between non material and material nodes in scenario S1 (a) and scenario S2 (b)

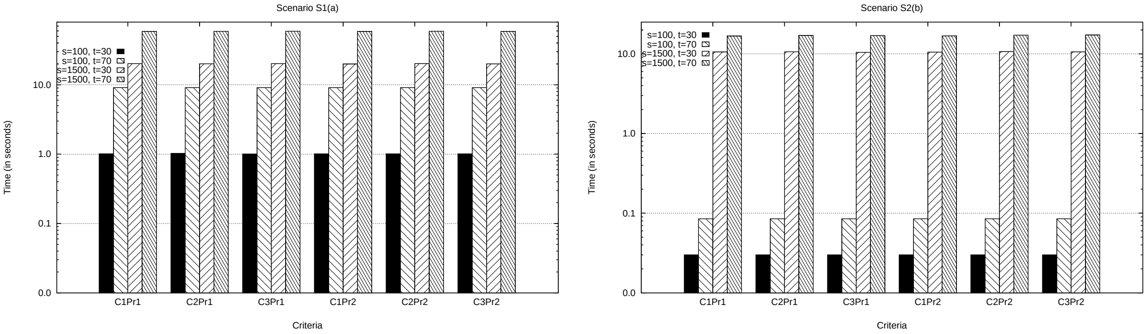

Figure 12: Time in seconds (logarithmic scale) taken by the DAG to Tree transformation algorithm in scenario S1 (a) and scenario S2 (b)

and if the pruning procedure is better, as we expected from the algorithm definition.

4.3 Scalability of the System

To evaluate the scalability of our proposal with respect to increasing access rights, we added the following access rights to the one previously described:

- $\mathcal{A}[P_i, TN_{\lceil i/pt \rceil}] = 1$, $i = 1 \dots p$: each player can access her team news;
- $\mathcal{A}[P_i, PN_j] = 1$, $i = 1 \dots p$, $j = (\lceil i/pt \rceil - 1) \cdot pt + 1 \dots \lceil i/pt \rceil \cdot pt$: each player can access the player news of people playing in her team;
- $\mathcal{A}[TM_{2i-1}, TN_j] = 1$, $i = 1 \dots t/2$, j is randomly chosen among $1, \dots, t$: half team managers (the ones that cannot access all team news) can access the team news of another team of the championship;
- $\mathcal{A}[TM_{2i-1}, PN_l] = 1$, $i = 1 \dots t/2$, $l = (j-1) \cdot pt + 1 \dots j \cdot pt$: these team managers have also access to the player news of the same team;
- as before, the distribution of the authorizations associated with subscribers follow a Zipf distribution but now each subscriber can access both the team and player news for her team.

We thus obtain an enhanced scenario S1 and an enhanced scenario S2 where there are more access rights than the access rights defined in the corresponding base scenario S1 and base scenario S2. As before, we evaluated the average number of keys assigned to each user (Figure 13), the ratio

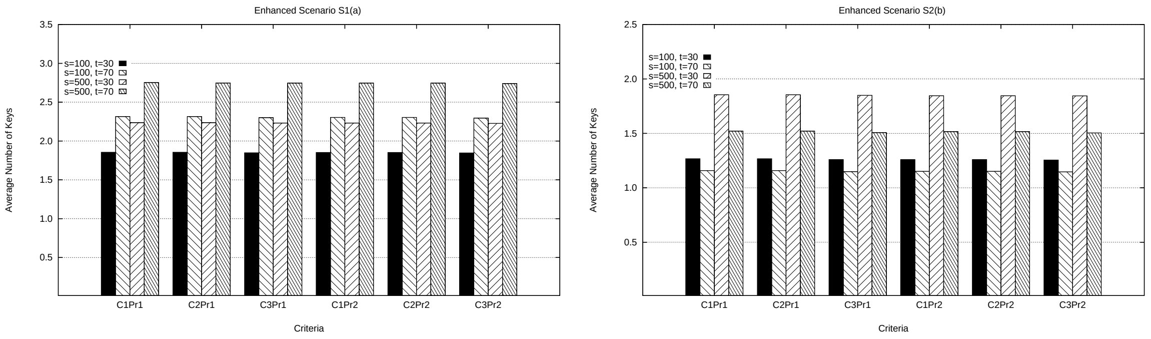

Figure 13: Average number of keys in enhanced scenario S1 (a) and enhanced scenario S2 (b)

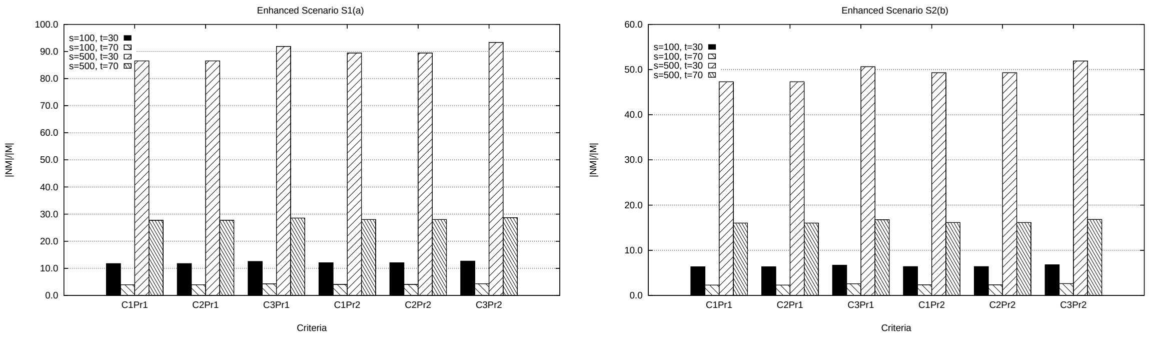

Figure 14: Ratio between non material and material nodes in enhanced scenario S1 (a) and enhanced scenario S2 (b)

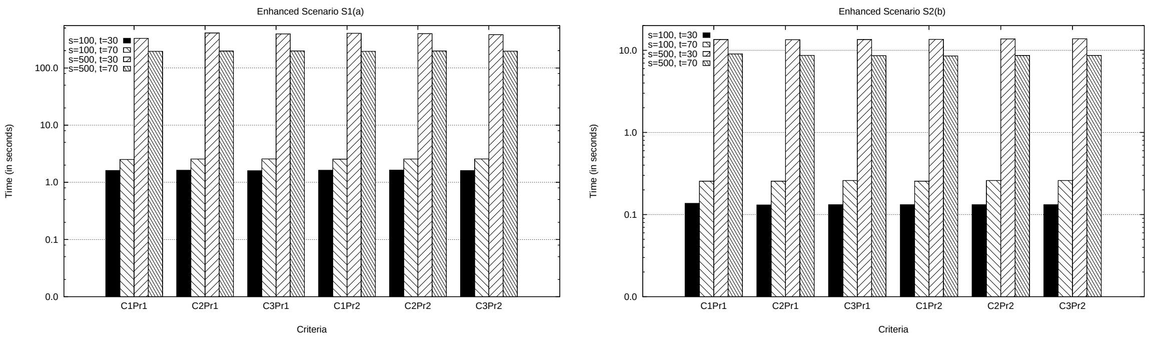

Figure 15: Time in seconds (logarithmic scale) taken by the DAG to Tree transformation algorithm in enhanced scenario S1 (a) and enhanced scenario S2 (b)

between the non material and material nodes (Figure 14), and the time taken by the algorithm (Figure 15).

Figure 13 shows that, as before, the average number of keys decreases with increasing the quality of the criteria adopted. However, by comparing Figure 10 and Figure 13, we can note that the average number of keys assigned to each user in the enhanced scenario is higher than the average number of keys assigned to each user in the base scenario. The reason is that now each player has more access rights and probably more keys. Also, in the base scenario S2 players are not considered because they do not have any access right. By contrast, in the enhanced S2 scenario each player has one access right (the one on her team news) ans consequently the average number of keys in the enhanced scenario S2 is lower than the average number of keys in the base scenario.

With respect to the ratio between the non material and material nodes, it is easy to see that in the enhanced scenario this ratio increases significantly (see Figure 11 and Figure 14). The rationale behind this observation is that in the enhanced scenario, the material node are located in lower levels than the material nodes in the base scenario. Consequently, more non material nodes need to be added in the tree to obtain a set of nodes closed under the intersection operation.

Finally, Figure 15 shows that, as expected, the time taken by the algorithm in the enhanced scenario increases significantly with respect to the time in the base scenario. Also, the considerations formulated for the base scenario are still valid in the enhanced scenario.

5. CONCLUSIONS

Access control is a very important issue in the DAS scenario, especially if the data owner wishes to publish her data for external use. In this paper we investigated a solution for implementing through cryptography a selective access policy. We introduced a method to exploit a tree hierarchy for key management and we performed some experiments for evaluating its efficiency, with respect to both the average number of keys assigned to each user in the system and the computational time. Issues to be investigated will include an analysis of the proposed approach in *dynamic* scenarios, where authorizations, users, and objects can dynamically change [10]. In these case, it may be necessary to re-encrypt data and to update the set of keys kept by each user involved in the changes.

6. ACKNOWLEDGMENTS

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

7. REFERENCES

[1] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of ACM SIGMOND 2004*, Paris, France, June 2004.

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

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public-key encryption with keyword search. In *Proc. of Eurocrypt 2004*, Interlaken, Switzerland, May 2004.

[4] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proc. CRYPTO 01*, pages 213–229, 2001.

[5] C. Boyens and O. Gunter. Using online services in untrusted environments - a privacy-preserving architecture. In *Proc. of the 11th European Conference on Information Systems (ECIS '03)*, Naples, Italy, June 2003.

[6] R. Brinkman, J. Doumen, and W. Jonker. Using secret sharing for searching in encrypted data. In *Proc. of the Secure Data Management Workshop*, Toronto, Canada, August 2004.

[7] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):119–152, February 2005.

[8] E. Damiani, S. De Capitani di Vimercati, M. Finetti, S. Paraboschi, P. Samarati, and S. Jajodia. Implementation of a storage mechanism for untrusted DBMSs. In *Proc. of the Second International IEEE Security in Storage Workshop*, Washington DC, USA, May 2003.

[9] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Metadata management in outsourced encrypted databases. In *Proc. of the 2nd VLDB Workshop on Secure Data Management (SDM'05)*, Trondheim, Norway, September 2005.

[10] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective release of information in outsourced encrypted databases. Technical report, University of Milan, 2005.

[11] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proc. of the 10th ACM Conference on Computer and Communications Security*, Washington, DC, USA, October 27-31 2003.

[12] G. Davida, D. Wells, and J. Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems*, 6(2):312–328, June 1981.

[13] E. Goh. Secure indexes. *http://eprint.iacr.org/2003/216/*.

[14] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of 18th International Conference on Data Engineering*, San Jose, CA, USA, February 2002.

[15] H. Hacigümüs, B. Iyer, and S. Mehrotra. Ensuring the integrity of encrypted databases in the database-as-a-service model. In *DBSec*, pages 61–74, 2003.

[16] H. Hacigümüs, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proc. of the 9th International Conference on Database Systems for Advanced Applications*, Jeju Island, Korea, March 2004.

[17] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the ACM SIGMOD'2002*, Madison, WI, USA, June 2002.

[18] H. Hacigümüs and S. Mehrotra. Performance-conscious key management in encrypted databases. In *DBSec*, pages 95–109, 2004.

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

[20] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proc. of the 30th VLDB Conference*, Toronto, Canada, 2004.

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

[22] S. MacKinnon, P.Taylor, H. Meijer, and S.Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, 34(9):797–802, September 1985.

[23] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced database. In *Proc. of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2004.

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

[25] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *Proc. of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, February 2004.