2009 First International Conference on Advances in Databases, Knowledge, and Data Applications

Distributed Key Management in Dynamic Outsourced Databases: a Trie-Based Approach

V. El-khoury, N.Bennani
Lyon University, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
{vanessa.el-khoury, nadia.bennani}@insa-lyon.fr

A. M. Ouksel
The University of Illinois
Dept. of Information and Decision Sciences
Chicago, IL, USA
aris@uic.edu

Abstract

The decision to outsource databases is strategic in many organizations due to the increasing costs of internally managing large volumes of information. The sensitive nature of this information raises the need for powerful mechanisms to protect it against unauthorized disclosure. Centralized encryption to access control at the data owner level has been proposed as one way of handling this issue. However, its prohibitive costs renders it impractical and inflexible. A distributed cryptographic approach has been suggested as a promising alternative, where keys are distributed to users on the basis of their assigned privileges. But in this case, key management becomes problematic in the face of frequent database updates and remains an open issue.

In this paper, we present a novel approach based on Binary Tries[1]*. By exploiting the intrinsic properties of these data structures, key management complexity, and thus its cost, is significantly reduced. Changes to the Binary Trie structure remain limited in the face of frequent updates. Preliminary experimental analysis demonstrates the validity and the effectiveness of our approach.*

1. Introduction

Outsourcing databases is becoming very popular due to the dramatic increase in the size of the databases and the costs incurred by their management. The databases are hosted by a third party [9], who then provides a "service" to clients to seamless access them. Data owners can now concentrate on their core competencies while expecting the outsourced databases to be managed by the best experts using the latest innovative solutions at lower costs. This approach, it is hoped, leads to an increase in productivity as well as cost savings.

Nonetheless, outsourcing databases is beset with new challenges. Foremost is the issue of data privacy in the presence of sensitive information. Most corporations view their data as very valuable assets. Therefore, it is paramount to protect these data against unauthorized access, including by the provider. Database encryption was seen as a solution to prevent exposure of sensitive information even in situations where the database server is compromised. The data will be encrypted at the server side allowing only the authorized persons to access the plaintext form of the databases. This solution however is not satisfactory as it does not allow access the database through ad-hoc queries. More flexible techniques have been proposed [2, 10, 9] based on storing additional indexing information with the encrypted database. These indexes are employed by the DBMS to enable posing queries over the encrypted data without revealing either the query or the data results. Figure 1 describes this mechanism. First, the user sends the query to the owner who maintains the metadata needed to translate it to the appropriate representation on the server (1). Then, the transformed query is executed on the encrypted database at the server side (2). Once executed, the results are sent encrypted to the owner who decrypts them and filters out those tuples not satisfying the user's assigned rights (3). Finally, the results are sent to the user in plaintext (4).

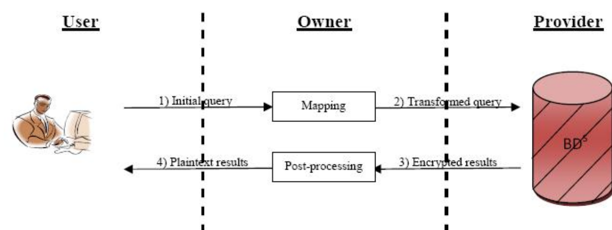


Figure 1: The service-provider architecture

[1]The Trie structure was introduced and implemented by Fredkin in 1960. The etymology of "trie" is the middle part of the term "Retrieval" and we pronounced it "try" in order to distinguish it from the word "tree".

978-0-7695-3550-0/09 $25.00 © 2009 IEEE
DOI 10.1109/DBKDA.2009.31

IEEE computer society

In such a scenario, the result of a query will contain false hits that must be removed in a post-processing step after decrypting the tuples returned by the query. This filtering can be quite complex especially for complex queries involving joins, subqueries, aggregations, etc... Clearly, the techniques based on this architecture [2, 6, 10] are impractical as the presence of the data owner is required to enforce access control.

Other query processing approaches [12, 5, 7] shift access control to the user. One example is the effective mechanism by Damiani et al. [5], which combines cryptography with authorizations, thus enforcing access control via selective encryption. Access control policies are modeled using an access matrix where rows represent users and columns tuples. The method consists of grouping users with the same access privileges, and encrypting each tuple accessed by the group with the key associated with it. A hierarchical key derivation method based on a User Tree Hierarchy (UTH [2][5]) is used to reduce the number of keys held by a user. But the complexity of the algorithm to build this UTH is exponential and the resulting structure does not flexibly handle frequent modifications to access control policies. In many cases, the whole structure may need to be reconstructed as it loses its properties and needs to be rebuilt to maintain the derivation efficiency.

In this paper, a new key management scheme based on the Binary Trie structure is proposed. As in [5], access control policies are modeled via an access matrix where rows represent group of users and columns the tuples to be accessed. An algorithm is described to generate the keys ring for each group of users based on the construction of the Binary Trie from the access matrix. This structure is characterized by its low complexity and high flexibility in the face of frequent access control modifications. Preliminary experimental analysis shows the validity and the effectiveness of our approach.

The remainder of the paper is organized as follows. Section 2 presents an overview of the related work and positions our approach within existing solutions. Section 3 introduces necessary basic definitions and describes the Binary Trie construction algorithm. Section 4 illustrates the management of the access control policies in a dynamic scenario. Section 5 presents and discusses the results obtained using our approach while section 6 analyzes them. Finally, Section 7 summarizes the work and gives some perspectives.

[2]Given a set U of users, a set T of tuples, and an access matrix A, the **U**ser **T**ree **H**ierarchy, denoted UTH, is a pair $(N, \preceq)$, such that:

- $N \subseteq P(U)$;
- $\forall t \in T, Acl_{\mathbf{t}} \in N$;
- $\forall X, Y \in N$, $X \preceq Y$ iff $Y \subseteq X$;
- $\forall X, Y, Z \in N$, $X \preceq Y$ and $X \preceq Z \Rightarrow Z \preceq Y$ or $Y \preceq Z$

2. Related works

Early approaches to database outsourcing [2, 10, 9] enforce access control at the owner side in a centralized manner. As a result, a bottleneck may occur, which in turn will reduce the benefits anticipated from outsourcing. Recent approaches [7, 12, 5] shift access control from the owner to the user. Such solutions require an off-line preprocessing stage where data sets accessed by each user, or more accurately, a user profile, are identified. An appropriate key is then used to encrypt each data set before its outsourcing. Finally, a set of keys are provided to each user accessed by, for its data sets according to his/her privileges.

Enforcing access control at the user level faces still two main problems: the determination of the key set and their efficient generation. Each user may be accessing several data sets, which themselves may be intersecting with those of other users, and thus may be require subdivisions into non-intersecting data sets, which adds enormous complexity to the method. Further the key generation algorithm must be as simple as possible to be scalable in practice, and flexible to adapt to frequent changes in a user's access rights.

All related works [7, 12, 5] model the problem using an access matrix where a row represents users or group of users, columns the data sets accessed by users or groups of users, and each matrix entry gives the access privileges of a user or a group of users to a data set. In [5], the notion of access configuration is defined for a data set ***d*** that the set of users, or groups of users, having access privileges on ***d***. A key derivation tree (UTH) is built from the access matrix, labelled heretofore A, to support the enforcement of the access control policy. Starting from an empty root vertex, the algorithm adds vertices to the tree structure based, each one represents an access configuration of a data set. The vertices satisfy a partial order where each vertex ***v*** is pointed by an edge from a parent vertex ***p*** whose access configuration $AC_{\mathbf{p}}$ is included in $AC_{\mathbf{v}}$. Each vertex is then assigned a secret key such that a key for vertex ***v*** is given to a user **u** if and only if **u** belongs to a ***v***'s access configuration and **u** does not belong to ***v***'s parent access configuration. All the other keys for which user **u** has rights are obtained by a derivation mechanism using one-way hash functions from the assigned keys. To make the derivation mechanism more efficient , the UTH construction algorithm adds other vertices which do not correspond to an access configuration but still corresponds to a set of users or user groups combinations called non-material vertices. The construction of UTH is exponential, $O(2^{|\mathbf{U}|})$ [5], where U is the set of users or user groups implied by the access control policy. The flexibility of the structure to adapt to frequent modifications in

access control policies, especially for the insertion and deletion of users is extremely limited. After several updates, the edges between vertices are so modified that UTH is enable to maintain key derivation efficiency. Consequently, UTH must be completely rebuilt, access configuration rekeyed, and data deciphered and re-ciphered.

In [12], an alternative key management scheme is presented based on a partial oder access configuration directed hierarchy. The leaves of such a structure are the access configurations for single users and while the edges are constructed bottom-up such that each intermediate vertex is pointed to by exactly two edges, and its access configuration is the union of those of its two children. The resulting hierarchy is then run the Diffie-Hellman [1] key generation algorithm which assigns secret keys to vertices in a bottom-up fashion such that a user with a single key could derive all the secret keys assigned to its data sets. However, the construction algorithm as well as the generation scheme are very complex and costly. Again, access rights updates cause a major rebuilding of the hierarchy and mandatory key generation leading in most cases to data re-keying.

In our Binary-Trie-based key generation and management approach, each category is considered as a binary string representing the path from the root of the trie to a leaf. The management of structure does not suggest a partial order over vertices, even though, the idea of key derivation to minimize the number of distributed keys is preserved. Key management complexity is reduced using the intrinsic properties of the Binary Trie. As keys are not tightly coupled to access configurations, most of the data access policy updates do not require significant changes to the structure, which reduces key regeneration, and thus data re-keying.

3. The Binary Trie building algorithm

In real databases, the data is represented by a set of views **V** that overlap. In our approach, we assume that there is a process used to partition these views in separate categories, so that $c_\mathbf{i} \bigcap c_\mathbf{j} = \phi$ if $\mathbf{i} \neq \mathbf{j}$. Furthermore, the users having the same privileges on categories are gathered in a single group. The process to obtain disjoined categories from views is out of the scope of this paper. Following [5], given a system with a set **G** of groups and a set **C** of categories, we assume that access control policies are represented in a matrix **A** having |**G**| rows and |**C**| columns. An entry A[g,c] of this matrix is a read operation of a group **g** over a category **c**, which can assume two values: A[g,c] = 1 if **g** has the privilege to read **c** and 0 otherwise. Table 1 represents an example of an access matrix of four groups and five categories, where the access control list of the category $\mathbf{c_1}$ is $Acl_{\mathbf{c1}} = \{g_2, g_3, g_4\}$ and the capability list of the group $\mathbf{g_4}$ is $Cap_{\mathbf{g4}} = \{c_1, c_2, c_3, c_4\}$.

Let us now introduce the Binary Trie, and then describe

Table 1: Example of an access matrix

	$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{c_3}$	$\mathbf{c_4}$	$\mathbf{c_5}$
$\mathbf{g_1}$	0	0	1	1	0
$\mathbf{g_2}$	1	1	0	1	1
$\mathbf{g_3}$	1	0	1	0	0
$\mathbf{g_4}$	1	1	1	1	0

the algorithm to construct it according to the desiderata discussed. Note that this is the first time that such structure is used to manage access control policies in outsourced databases. A Trie [8] is an ordered tree data structure that has been used for various applications, such as the construction of natural language dictionaries, the research of words to a compiler, database systems, networking [11], etc. In this structure, all the descendants of any one node have a common prefix associated with that node, while the root is represented by an empty node. This structure has many properties that make it particularly suitable to access control as discussed above. It allows fast retrieval time by finding the longest match of a given string. Moreover, the time complexity for the construction of strings is easy to compute and it is flexible towards modifications. The structure of a node is adapted as indicated below to handle the requirements of our specific application:

- *Content* is used to sequentially store the prefix of the category.
- *AccessNb* is the number of time that the node is visited during the insertion of categories. This value will be useful while modifications are done in the access matrix, particularly when a category is deleted.
- *LeftChild* will be represented by an edge labeled 0.
- *RightChild* will be represented by an edge labeled 1.

Below, the algorithm 1 illustrates the algorithm we developed that takes the access matrix **A** as an input and returns the Binary Trie as output.

Given the access matrix A, $\forall i, j \in \mathbb{N}$ for j:= 1;|c| the algorithm scans A[g,j] by category and for i:=1; |g| each category is scaned bit per bit. Then for each entry A[i,j], we applies the function InsertValue(value, position) to insert the bit value at the current position in the Binary Trie. During this operation, two cases can be represented. For the bit value 1 (respectively 0); if the right (respectively left) child node exists, the field AccessNb will be incremented by 1. Otherwise, a new right (respectively left) node is created having AccessNb equal 1 and his content will be the concatenation of the content value of his parent and the bit 1(respectively 0).

For example, let us insert the category $\mathbf{c_1}$ having the bit sequence 0111. At the beginning, the children of the root

Algorithm 1 Binary Trie construction algorithm.

```
PROCEDURE BuildTRie(Mat : Matrix)
CreateNode (root,'')
FOR j ← 1 TO |C| DO
    position ← root
    FOR i ← 1 TO |G| DO
        position ← InsertValue(ValMatrix(i,j), position)
    END FOR
END FOR
END PROCEDURE

FUNCTION InsertValue(value, position)
IF value = 1
    IF(Rightchild(position) = nil)
        CreateNode(RightChild(position), value)
        Content(RightChild(position)) ←
        Content(position) + '1'
    ELSE
        IncrAccessNb(RightChild(position))
    END IF
    InsertValue ← RightChild(position)
ELSE
    IF(LeftChild(position) = nil)
        CreateNode(LeftChild(position), value)
        Content(LeftChild(position)) ←
        Content(position) + '0'
    ELSE
        IncrAccessNb(LeftChild(position))
    END IF
    InsertValue ← LeftChild(position)
END IF
END FUNCTION
```

node point to null. Starting with the first bit '0', a left child node of the root will be created having 0 as content and 1 for the AccessNb value. Then, the second bit '1' will be scanned and the right child of the left node will be created having 01 as content and 1 for the AccessNb value. We proceed with the same reasoning till the leaves level. Let us now consider the insertion of the category c_2 having the bit sequence 0101. Since the left node for the bit '0' already exists, all that we have to do is to increment by 1 the AccessNb value and so on. The Figure 2 is an illustrative example of the Binary Trie according to the access matrix in Table1.

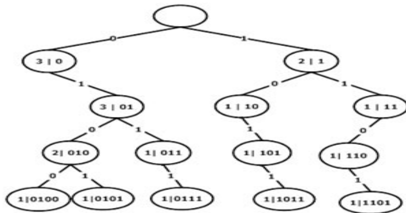


Figure 2: Binary Trie example

An observation of the Binary Trie shows that the transformation algorithm retains all the properties of the matrix. Each group of users is represented by a level in this structure according to their order in the matrix, while categories appear in the leaf level preserving the same sequence of bits. In fact, for a given category **c,** $\forall i \in \mathbb{N}$ and $i := 1; |g|$ an entry A[i,c] in the access matrix is represented by the level **i** in the Binary Trie. Besides, the internal nodes store the prefix of their descendants which is the least upper bound. Therefore, the content of the nodes at the leaf level is nothing but the bit sequence of the category c. Furthermore, for a given group **g,** $\forall j \in \mathbb{N}$ and $j := 1; |c|$, the entries A[g,j] are represented in the level **g**. For example, consider the group $g_1$of the matrix in Table 1, for $j := 1; 5$ the values of A[g_1,j] correspond to the level 1 in the Binary Trie. In the following section, we describe the algorithm of key distribution as well as the key management towards the changes in the access control policies.

4. Key management in case of dynamic access control policies

Once the Binary Trie structure is created, the next step consists in affecting the appropriate key ring to each group of users. A group can just access the nodes having their edges labeled by the bit 1, that is, the right nodes (for a level i = 0, . . . ,|G| there is a maximum of 2^{i-1} right nodes). Therefore, an encryption key is assigned to each right node, thus to the associated group. Then, we apply the one-way-hash function to derive the real keys that will encrypt the data itself in the outsourced database. Consider Φ a function that attributes the appropriate set of keys to the corresponding group. With respect to the trie in Figure 1,$\Phi(g_1) = \{k_1\}$, $\Phi(g_2) = \{k_{01}, k_{11}\}$, and so on. For example, the group g_1 is authorized to access c_3 and c_4 respectively encrypted with $\mathbf{k_{1011}}$ and $\mathbf{k_{1101}}$. Since the group g1 holds the key $\mathbf{k_1}$, then he will be able to derive these two keys and to decrypt the corresponding categories.

The Binary Trie structure is built and each group of users holds his keys ring according to his privileges as defined in the access matrix. In the following, we discuss the five operations that are triggered whenever a change occurs in the access matrix.

4.1. Insert/Delete category

When the owner inserts new tuples into the database, two cases may arise. If these tuples belong to an existing category in the access matrix, they will be encrypted by the key assigned to this category (the key associated with the node of the corresponding category). Otherwise, the owner creates a new category c_j based on Acl_j. The category will be scanned bit per bit in the matrix leading to some modification in the Binary Trie (Algorithm 1). If the node exists, it is sufficient to increment its *AccessNb* attribute, otherwise

we must create a new node and assign a new key if it's a right node. Once inserted, this key will be associated with the groups having the same level as that node.

For example, let us insert a new category c6 having$Acl_{\mathbf{c6}} = \{g_{\mathbf{1}}, g_{\mathbf{4}}\}$ which is equivalent to {1001} sequence of bits (Figure 2), their *AccessNb* are updated, when two new vertices are created in levels 3 and 4. A new key $\mathbf{k_{1001}}$ is generated and assigned to the group $\mathbf{g_4}$.

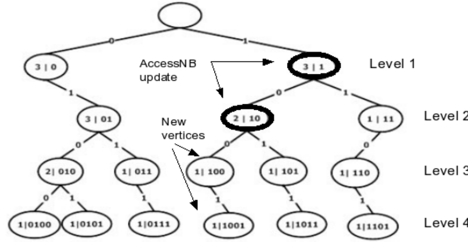


Figure 3: Insert category $\mathbf{c_6}$

Similarly, when a tuple is removed from the database, this requires an update on the Binary Trie. If the tuple covers a whole category, it will be removed from the access matrix as well from the structure. The idea is to scan all the node and decrement by 1 their *AccessNb*. Once we get a node with *AccessNb*=1, we delete this node and all its descendants.

4.2. Insert/Delete group

When a new user u is inserted in the system, the owner examine his $\text{Cap}_{\mathbf{u}}$. If there is a group $g \in \mathbf{G}$with a similar capability $\text{Cap}_{\mathbf{u}}$, the user will be added to group g. Otherwise, a new group is created in the matrix and inserted at the leaf level of the Binary Trie. Inversely, the deletion of user u from a group g with Card(g)=1 leads to the group deletion. As a consequence, a category $\mathbf{c} \in \mathbf{Cap_g}$ will be decrypted and re-encrypted with a new key. Moreover, the deletion of user u from a group g with Card(g)$\neq$1, implies keys ring regeneration and redistribution to the remaining members of the group.

4.3. Grant/revoke authorization

When the owner grants/revokes to a group $\mathbf{g} \in \mathbf{G}$ the authorization on a category $\mathbf{c} \in \mathbf{C}$, the entry **A[g,c]** in the access matrix will change to 1 (respectively 0). Therefore, the category will be encrypted with a new Key and modifications are needed for the Binary Trie. In fact, a partial deletion is done for the category **c** ; we scanned the category till the $\text{bit}_{\mathbf{i}}$ that should be modified ; the node at the level **i** will be deleted and the trie will be fixed according to the rest of the binary sequence.

5. Theoretical and Experimental Evaluation

5.1. Theoretical evaluation

Let us have now a theoritical evaluation of our algorithm. Consider S a set of string $S_{\mathbf{1}},\ldots, S_{\mathbf{n}}$, that is, $S_{\mathbf{i}} \neq S_{\mathbf{j}}$ for $i \neq j$. For an alphabet of k elements, the Trie of the set S is a tree of k-ary such that each node represents a distinct prefix in S. Thus, the time complexity to build this Trie is O($|S_{\mathbf{1}}|$+...+$|S_{\mathbf{n}}|$).

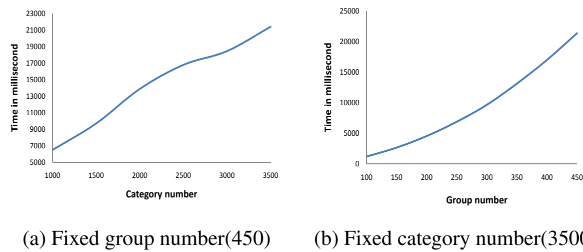
Let |C| and |G| be respectively the number of categories and groups in an access matrix. Inserting a category of |G| bits to a Binary Trie requires adding a sequence of |G| nodes, thus the time insertion complexity is O(|G|). Since the construction of this structure is nothing else a succession of insert operations where the categories may have a common prefix, then in the worst case, the time complexity of the Binary Trie construction algorithm is O(|C|*|G|). In [5], the time complexity of the algorithm for the UTH building is polynomial with respect to the number of vertices selected to build the tree hierarchy. However, according to the proof presented in the appendix of this article, the complexity is exponential with respect to the number of users in the system when the UTH is a complete tree.

5.2. Experimental evaluation

In this section, we present our experimental results validating the above theory. The experiments were ran on a Toshiba Intel-based with 1.66Ghz Core2 Duo processor with 2 GB of RAM. We developed our program in Java version 1.6 and ran it on Windows Vista as an operating system. This program consists of three phases. First, we developed an algorithm to generate an access matrix policies to be as close as possible to reality. Doing so, we control the cardinality of the access control list of the groups. In the second phase, an algorithm is executed to build the Binary Trie from this access matrix. Finally, a third algorithm is executed to generate the set of keys for each group of users based on this structure.

In the first set of experiments, we conduct tests on couples [group, category] for a group value 450 with increasing number of categories from 1000 to 3500 with steps size 500 (Figure 4.a). For each couple, we launch 20 simulations and then calculate the average of time construction. Though we conduct a second set of experiments by fixing the category value to 3500 and increasing the number of groups from 100 to 450 with steps size 50 (Figure 4.b).

According to the graphs, the experimental results conforms with the theoretical complexity of O(|C|*|G|). The curve for a different value of categories (respectively groups) grows almost linear O(K*|G|) where $K \in \mathbb{R}$ and



(a) Fixed group number(450) (b) Fixed category number(3500)

Figure 4: Binary trie time building evolution

is proportional to |C|. The reason is due to the categories having same prefix in the access matrix.

6. Discussion

In this paper, we provided an efficient mechanism for key managment based on the Binary Trie structure. As we notice, this structure is flexible towards access rights granted dynamically to the users and the growth of the accessed data set. It efficiently handles all these changes without need of a systematic reconstruction. Also, a derivation key algorithm is represented so that the Binary Trie is scanned from left to right in-depth first manner. Nevertheless, as the Binary Trie depth grows with the increase of the groups number, this could penalize the key generation time. For this purpose, a flat representation of the Binary Trie [3, 4] could help to reduce the time of key generation by an immediate access to the vertices. Moreover, our approach offers a good flexibility in the case of access rights update. The only case where categories should be rekeyed is the user revoke case. Even though, the effects on the structure of management are tiny and do not require a complete reorganization of the Binary Trie, but just some bonds update in the structure which is less costly. However, our approach doesn't reduce the number of keys held by a group. In fact, when examining the Binary Trie, we noticed that the groups at the low-level of the structure(espacially at the leaf level) hold a large number of keys; but the more the groups are in the bottom of the structure, the fastest are the derivation to obtain the keys.

7. Conclusion and Perspectives

In this paper, we present a novel mechanism based on the Binary Trie structure, for the enforcement of the access control at the client side for the outsourced database. This mechanism allows an off-line key generation and exploits hierarchical key derivation methods using the one-way hash function. Our approach has the advantage of efficiently generating the groups keys ring from the Binary Trie structure according to their privileges and reducing the cost of the key management in the case of access control policies modifications. Also, we proved theoretically and in experiments that our solution is of reduced complexity and is likely to be scalable. However, our approach does not completely resolve the well-known data rekeying problem due to users rights revocation. As a perspective, we plan to investigate a remote storage of the users' keys. In fact, since the keys are attributed to a group of users, a single copy of them will be stored rather than distributing the same one to each user belonging to the group. Furthermore, this allows the users to query the database regardless of their machine.

References

[1] Rsa laboratories.pkcs number three: Diffie-hellman key agreement standard. *http://www.rsa.com/rsalabs/node.asp?id=2126*.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[3] M. Ai-Suwaiyel and E. Horowitz. Algorithms for trie compaction. *ACM Transactions on Database Systems*, 9:243–263, 1984.

[4] J. Aoe, K. Morimoto, and M. Shishibori. A trie compaction algorithm for a large set of keys. *Knowledge and Data Engineering, IEEE Transactions on*, 8:476–491, 1996.

[5] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective data encryption in outsourced dynamic environments. *Electronic Notes in Theoretical Computer Science*, 168:127–142, 2007.

[6] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. *Proceedings of the 10th ACM conference on Computer and communications security*, pages 93–102, 2003.

[7] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: management of access control evolution on outsourced data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 123–134. VLDB Endowment, 2007.

[8] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[9] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 29–38, 2002.

[10] H. Hacigumus and S. Mehrotra. Performance-conscious key management in encrypted databases. *Research Directions In Data And Applications Security XVIII: IFIP TC 11/WG 11.3 Eighteenth Annual Conference On Data And Applications Security, July 25-28, 2004, Sitges, Catalonia, Spain*, 2004.

[11] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Academic Press, 2007.

[12] A. Zych, M. Petković, and W. Jonker. Efficient key management for cryptographically enforced access control. *Comput. Stand. Interfaces*, 30(6):410–417, 2008.