International Journal of Database Theory and Application
Vol.7, No.3 (2014), pp.63-72
http://dx.doi.org/10.14257/ijdta.2014.7.3.07

Enabling Access Control in Partially Honest Outsourced Databases

Lanju Kong, Qingzhong Li and Lin Li

School of Computer Science and Technology, Shandong University
{klj,lqz,lilin@sdu.edu.cn}

Abstract

With the growing popularity of outsourced databases (ODBs), access control for multiple users with different privileges in outsourced environments is required in more and more applications. Under the assumption that ODBs may be interested in the original data value, or delay the update operations when end users cannot verify the results, this paper attempts to enhance ODBs with finegrained access control for multiple users with less impact on their other functionalities. Our work can be divided into two parts. In the first part, we propose a method to enforce the access control rules by encrypting the original table and using the keys to distinguish various rights. In addition to read/non-read rights, read/update rights can be distinguished in our encrypted table. We also implement validation-only rights for ODBs and oblige them to fulfill any update validation without knowing the original data. In the second part, we study the query evaluation over the encrypted table. Two kinds of B+ tree indexes on each column are designed, which can accelerate the selection in ODBs.

Keywords: *Access Control, Outsourced databases, Query evaluation*

1. Introduction

ODBs have drawn extensive attention from both academia and industry. In this environment, customers outsource their data to a third-party service provider, which not only offers scalable and stable service at a low price, but also migrates tedious administrator tasks from them. Nowadays commercial ODBs are available [1, 2]. These products are mature enough to satisfy the application requirements. Hence, it becomes an important trend for end users to build their enterprise information management systems on ODBs.

ODBs bring many benefits to end users, but also raise new security issues. On the one hand, to fully exploit the functionalities of OBDs, such as query evaluation and data backup/restore, and the like, all the data including sensitive information need be outsourced. On the other hand, end users have little control over the outsourced data. The administrator of the ODB can easily access the data they manage without the data owner's awareness. In order to cope with such a dilemma, many methods have been proposed. When ODBs are assumed curious, the data is always encrypted before being outsourced. Hence the query friendly encryption method or index strategy over the encrypted data has been studied in [3, 6, 9]. When ODB is assumed not fully honest, the correctness and completeness of query results from ODB are studied in [8, 11, 14, 15].

This paper assumes that ODBs are "curious" and "partially honest". We make the same assumption as existing work that ODBs are "curious" about the original data. The measure of the honesty of ODBs is whether they perform instructions correctly. Rather than assuming that ODBs are "honest" in [16], we assume ODBs are "partially honest". Since ODBs promise service quality in their contract with users, they face a penalty if users can find evidences to prove that ODBs work incorrectly. Consequently, ODBs are trusted when they

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2014 SERSC

are aware that the results can be verified by end users. Otherwise, ODBs may delay or even deny the operations.

The access control is an important component of database and is greatly desirable in most applications. It is not trivial work to implement access control in "curious" and "partially honest" ODBs. First, various privileges, not only read/non-read right, but also the read/update right and validate-only right, need be distinguished. The users with read-only right can access the original data. If they write the modified data back to the ODBs, the next reading is required to discover the unauthorized write. In addition, even if the next reading can find the error, the database has been damaged. Hence, we need to support a validate-only privilege for ODBs and force them to check the unauthorized update, even if they cannot access the original data. Second, key management becomes more complex when different privileges are considered. In order to support specified privileges for different users, data items will inevitably be encrypted with different keys. To free users from the burden of key management, a simple but effective key-derivation mechanism is desired. Last but not least, the implementation of the access control cannot seriously degrade the query performance over ODBs. The query evaluation should consider both the query predicates and the user's access rights. Certain kinds of indexes to speed up the processing are therefore highly desirable.

In order to overcome these challenges, this paper proposes an approach to enforce access control for multiple users. Specifically, our contributions can be summarized as follows:

- We propose a method to support the access control mechanism in an outsourcing environment. The access control rules can be specified at the granularity of table cells. For each table cell, symmetric encryption is used to prevent the unauthorized read, and the asymmetric encryption technique is introduced to distinguish the read and update privileges. ODBs are also assigned keys for unauthorized update checking. In order to lower the encryption overhead on each table cell, we devise a cell grouping optimization strategy. (See Section 3)
- We discuss the query evaluation over the encrypted table. Specifically, we propose two kinds of B+ tree indexes on each column in the table to accelerate query evaluation. One tree for encrypted values, named EVT, is to support the evaluation of query predicates. Another tree for tuple ID, named IT, is to locate the data item via tuple ID. (See Section 4)

The remainder of the paper is organized as follows: Section 2 reviews preliminary knowledge and shows the framework of our method. Section 3 presents the method to enforce the access control rules. Section 4 discusses the query evaluation over the encrypted table. Section 5 reviews the related work and Section 6 concludes the paper.

2. Preliminary Knowledge

In this section, we review the access control rules and the attack model, and finally sketch out the framework of our method.

2.1. Access Control Rules

The access control rule in this paper is specified at the granularity of table cells. This is an extension to that in the traditional relational DBMS, where the access control can only be assigned on specific columns. The access control rule can be described as follows [7]:

Definition 1 Access Control Rule. *An access control rule is a 5-tuple of the form (subject, object, condition, right, sign), where subject is the user to whom the authorization is granted*

 Copyright ⓒ 2014 SERSC

or revoked, object is a table, condition is expressed by a SQL selection statement, $right \in \{read, update\}$, $_{right}$ *and* sign *is* 1 *(for grant statement) or* 0 *(for revoke statement).*

The subject in our paper represents the user. In order to simplify the expression, the user with the update privilege is referred as the writer, and the user who can only read is referred as the reader. Apart from these, we refer to the user with no privilege on the cell as the intruder. In addition, the data owner is a special kind of user. The data owner is responsible not only for the initial allocation of the data blocks under the access control rules, but also for the maintenance of the shared data structure for other users used in ODBs.

Then we use $\mathrm{right}_{u,R}(\mathrm{cell}_{i,j})$for the right (read or update) on $\mathrm{cell}_{i,j}$of a user u under the rules R . The evaluation of the access rules will transform the original table to an encrypted table. The transformation is correct when the cells which users can access from the encrypted table is the same as the cells which are specified by rules.

Definition 2 Accessible Cells Under Rules. *Let R be the access control rules, U be the users, p be a right, for each* $u \in U$, $Cell_{p,R}(u) = \{c|c \ is \ in \ T, Cell_{u,R}(c) = p\}$.

2.2. Attack Model

The enforcement of the access control rules should protect the data from any unauthorized read or update attack. An unauthorized read attack occurs when an intruder attempts to read the original data they are not authorized to see. Recall the assumption that ODB is partially honest; it might return data blocks to intruders without validating their privileges. To counter such an attack, we should make sure that even the intruders receive these blocks, and they are unable to view the data.

The second kind of attack, unauthorized update attack, is issued by readers or intruders. They may attempt to modify the data blocks they have no right to read or can only read. As we know, each user can obtain the (original) block via get operation, modify the data and send the modified data back to ODB using put operation. Even if the user can detect that the data has been modified incorrectly at the next visiting, the original valid data may have been overwritten. A promising method to defeat against this kind of attack is to empower ODB to validate each update operation without knowing the original data, and oblige ODB to fulfill the validation.

2.3. Framework of Access Control

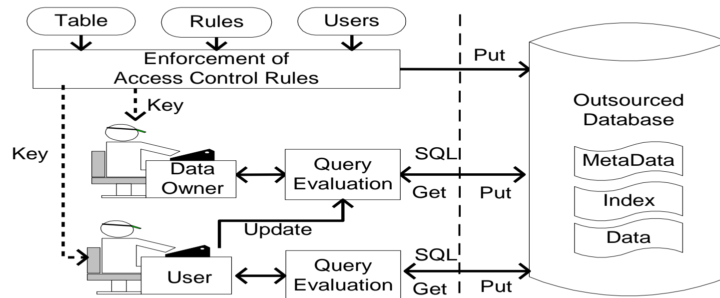

Figure 1. The Framework of Access Control Implementation in ODB

The framework of our method is illustrated in Figure 1. The client in the left part can communicate with the ODB in the right part via get/put commands and SQL statements. ODB stores all the data and responds to clients' requests. The data stored in ODB are organized

Copyright ⓒ 2014 SERSC

into three layers, namely the meta data layer, the index layer and the encrypted data layer. The metadata layer and encrypted layer are discussed in Section 3. The index layer is introduced in Section 4.

Initially, the data owner specifies access control rules for other users on one table, and evaluates the rules over the table. The encrypted blocks are put into the server, and each user u is assigned with a distinct key $u.key$. It can be used to derive the other nested keys outsourced in ODB. Users can issue privilege-related statements and select/update statement against the ODB. The query evaluation may involve multiple interactions with ODB. A user should first get the meta data block, decrypt it, and then visit the corresponding index with get command to locate the data blocks required. Finally, he can get the encrypted data blocks. By performing decryption with his initial key and the nested keys derived, he can obtain the final results. When a writer attempts to update the data item, he need not only change the data, but also update the validation information.

3. Enforcement of Access Control Rules

In this section, we propose a method to enforce access control rules over a table. We first discuss the encryption form for each table cell, then extend it to the cluster with the same accessibility for all users, and finally introduce the encrypted table.

3.1. Encrypted Cell for Multiple Users

The encrypted form of the cell is the basis of the encrypted table, since the access rules are specified at the granularity of the cell. The encrypted cell should allow authorized access and prevent the unauthorized access both. The users here include readers, writers, intruders, and ODBs. Although neither can view the original data, the former will be forced to check the unauthorized update on the data item. Hence, we need to distinguish four kinds of privileges, namely the update, the read, the validation and no right.

As in the existing work, the symmetric encryption function can be used to prevent any unauthorized read. The data is encrypted with a symmetric encryption function before being uploaded into the ODB. The key will be distributed to the reader and writer. The other two roles cannot access the data without the key. Formally, the encrypted cell for multiple users is defined as follows:

Definition 3 Encrypted cell for Multiple Users. *Let c for the content of $cell_{i,j}$ in a table T, U be users, each user $u \in U$ with one key $u.key$, O be ODB, R be the access control rules, $hash$ be a hash function known to users and ODB, we generate a public/private key pair pub_c/pri_c for the update verification by users, a key k_o for update verification by ODB, a key k_c for the encryption of the content. O is assigned with k_o. The encrypted form $enc_{U,R}(c)$ takes a 4-tuple form $\{e(c), v_u(c), v_o(c), K_U(c)\}$,*

(1) $e(c) = E^s_{k_c}(c)$;

(2) $v_u(c) = E^a_{pri_c}(hash(e(c)))$;

(3) $v_o(c) = E^s_{k_o}(hash(e(c)))$;

(4) $K_U(c) = \{k_u(c) | u \in U\}$. $k_u(c) = E^s_{u.key}(k_c, pub_c)$ $when\ right_{u,R}(c) = read$;
$k_u(c) = E^s_{u.key}(k_c, pub_c, pri_c, k_o)$ $when\ right_{u,R}(c) = update$;

We illustrate relationships among components in an encrypted cell in Figure 2. Basically, the original cell c is encrypted into $e(c)$ by a symmetric function with a key k_c. In order to distinguish the read/update rights, $v_u(c)$ is introduced for update verification, in which asymmetric encryption runs on a hash result over $e(c)$. A writer can update $v_u(c)$ and a reader

 Copyright ⓒ 2014 SERSC

can verify $v_u(c)$. $v_o(c)$ is introduced to allow ODB to detect any unauthorized update without knowing the original value. Notice that hash is also known to ODB. In order to reduce the burden of key management for each user, an encrypted cell takes the nested key strategy for each user. Writers can get 4 nested keys and readers can get 2 nested keys from $K_U(c)$ with their initial keys.

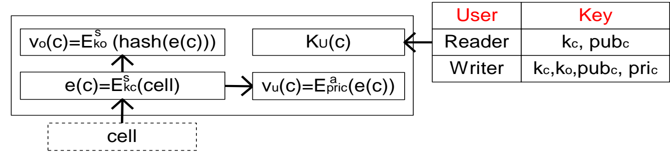

Figure 2. Relationships between Keys

We further explain the interaction between different components in an encrypted cell during reading and updating operation in Figure 3. Suppose a user u attempts to read a cell c in the left part of Figure 3. When u has no right, $right_{u,R}(c)$ returns empty and $K_U(c)$ does not contain the nested key for u, so u cannot access the data. When $right_{u,R}(c)$ returns read, u can obtain k_c and pub_c from $K_U(c)$, and then u can further decrypte(c) with k_c . After the original content of c is reproduced, u detects whether $D^a_{pub_c}(v_u(c))$ equals $hash(e(c))$. If it does, the data decrypted from $e(c)$ is a correct value. Otherwise, u will know ODB allowed an unauthorized update, which is an evidence to show that the ODB works incorrectly.

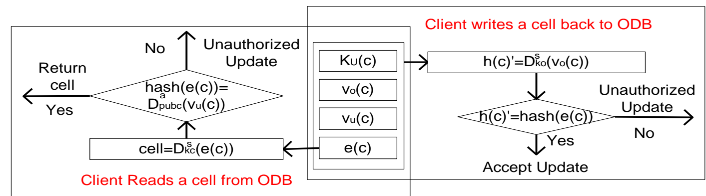

Figure 3. Read and Update of an Encrypted Cell

The right part of Figure 3 describes the updating operation. Suppose that u is a writer, u will get 4 keys by decrypting $K_U(c)$. u can then use kc to view the content of the data. If u wants to update c, u needs to regenerate both $v_u(c)$ and $v_o(c)$ according to the new $e(c)$. Each time the data is written back to the outsourced database, ODB accepts the update operation only when $D^s_{k_o}(v_o(c))$ equals $hash(e(c))$. Since readers have no key k_o, they cannot produce a valid $v_o(c)$and consequently ODB will find an unauthorized update. Notice the reason that ODB checks the unauthorized update is that the next reading can also verify the consistency of data if ODB does not, as illustrated in the left part of Figure 3.

3.2. Cells Grouping and Partition

Basically, an encrypted table can be the union of all encrypted cells. However, this straightforward method will incur high time and space overhead. The encryption itself is a time consuming operation, and the size of the encrypted data is always larger than that of the plain text due to the padding strategy used in the encryption function.

An important observation is that some cells share the same accessibility for all users. These cells provide an opportunity to implement the bulk encryption. That is, we can merge them together, generate one public/private key pair and two symmetric encryption keys, and encrypt them as one big cell. The validation in the reading and the verification in the updating

Copyright ⓒ 2014 SERSC

are the same as those in the cell operations. In this way, the space cost and computation cost incurred by the asymmetric encryption can be greatly reduced. Formally, these cells are described as follows:

Definition 4 Equivalent Access Class. Let $U = < u_0, \ldots, u_n >$ be an ordered set of users, R be the access control rules, C be cell set, each $c \in C$ is annotated with a n-dimensional vector $\phi(c) = < right_{u_0,R}(c), \ldots, right_{u_n,R}(c) >$. An equivalent access class $C_e \subseteq C$ meets the following two requirements:

(1) For any cell $c \in C_e$ and $c' \in C_e$, $\phi(c) = \phi(c')$;

(2) There is no cell $c'' \notin C_e$, where $\phi(c'')$ is the same to $\phi(c)$, $c \in C_e$.

The next key problem is how to place cells from one equivalent access class into different blocks. Since the number of the blocks transferred is an important measure of query performance, the minimization of blocks transferred for multiple queries is the objective of the cell placement. The problem can be formulated as follows. Let Q be a query workload, C be an equivalent access class, for each $q \in Q$, $freq(q)$ be the frequency of a query $q \in Q$, $cost(q)$ be the sum of the blocks B sent back to the client side, where all cells required in the evaluation of q are in B. The problem is how to lace the cell c into the blocks, so as to minimize the total evaluation cost $\sum_{q \in Q} cost(q) * freq(q)$. Since the block required by query q may contain cells unrelated to q, the placement of cells greatly impacts on the total evaluation cost of the query workload. This problem can be proven NP-hard, which can be reduced from a set cover problem.

In this paper, we take a greedy method to implement the partition. We expect that the cells in one block can be used in the same queries as often as possible. Let C be an equivalent access class, for each $c \in C$, $c.usedIn$ be the queries whose evaluations require c. We then place each c into blocks. When the current block is full, a new empty block b is allocated and the cell $c \in C$ with maximal $|c.usedIn|$ is placed into b. Otherwise, the benefit $ben(c, b)$ of a cell $c \in C$ into b can be defined as $\sum_{c' \in b} \sum_{q \in c.usedIn \cap c'.usedIn} freq(q)$. In other words, if c is not put into b, $ben(c, b)$ indicates the maximal extra blocks required in the evaluation of the query workload. After c is selected and added into one block, c is removed from C. Such processing continues until C is empty.

3.3. Encrypted Table

After the cells are put into different blocks, the encrypted table is produced with all encrypted blocks. In addition, the encrypted table also contains a metadata block for each user to record the location of the blocks in ODB. The metadata block is stored in ODB, and its location is stored at the client side.

Definition 5 Metadata Block for Each User. *Let u be a user, C be all cells, R be all accessrules, the metadata block $meta(u)$ for u is an encrypted block $E^s_{u.key}(L)$, where L is a sequence of* $\{loc(b) | loc(b)$ is the location of encrypted block b, b contains at least one cell in $Cell_{read,R}(u)$ or $Cell_{update,R}(u)\}$.

4. Query Evaluation

In this section, we first discuss the straightforward method to evaluate a query, and then propose two kinds of indexes used in the query evaluation.

 Copyright ⓒ 2014 SERSC

4.1. Basic Method to Evaluate Query

End users can evaluate SQL queries directly against the encrypted table. Suppose a user u issues a query SQL over an encrypted table T, u will get the meta data block meta(u) first. With u's initial key u.key, meta(u) can be decrypted and the locations of all accessible blocks are recovered. Consequently, u applies get operations to request all blocks return to the client side. These blocks can be decrypted with the nested keys. Since each cell is annotated with the column name and a unique tuple ID, the subset of table which u can access is then reconstructed. The SQL query runs on this subset of the table and u gets the final results.

Although the meta data block for end users can prune the inaccessible blocks, the basic method still incurs expensive time and space overheads, since all accessible cells, whether used in the query evaluation or not, will be sent back to the client side and decrypted. In this method, ODBs only provide a simple storage service and end users carry out all tasks in the query evaluation.

4.2. EVT and IT Index over Encrypted Table

In order to overcome the limitations of the basic method, we try to enhance ODBs to provide more query support. In the following, we design indexes in ODBs to reduce the number of encrypted blocks required in the query evaluation, so to reduce both network transfer and decryption overheads.

We first discuss what kinds of indexes are required. An SQL query mainly consists of the query predicates and target list. The query predicate specifies the conditions on one or multiple columns. The target list contains the columns returned to end users. The columns in the target list may not be in the predicates. Since the cells in each tuple may not be in the same block, the evaluation of a query firstly gets the tuple ID set *I* each of which is for a tuple satisfying the predicates, and then locates the values of the target columns whose tuple IDs are also in *I*. We observe that the location of ID set from predicates as well as the location of data value set from ID set can be sped up by indexes. Due to arbitrary combinations of columns in the query predicates or in the target list, it had better build two indexes for each column.

The next key problem is how these indexes are organized for different users. We build one index shared by all users and maintained by the data owner. Each user can access the index in a read-only mode with the key assigned by the data owner. When the cell is updated by user u, u need send a request to the data owner, who is responsible for the index adjustment. Since the index is shared by all users, the verification on each index node is also needed to defend against the unauthorized update on the internal nodes of the index.

Specifically, we introduce an encrypted B+ tree on each column to support an efficient predicate evaluation. Intuitively, the encrypted B+ tree is the encryption version of the B+ tree on the plain data.

Definition 6 Encrypted Value B+ Tree. *Let col be a column in a table,* $\mathrm{B_t}$ *be the B+ tree for the plain values of col, $hash$ be a hash function known to all users and ODB. We generate a public/private key pair pub_i/pri_i, symmetric encryption function keys k_i and k_o. The data owner has all keys, and other users only have pub_i and k_i. The encrypted value B+ tree $EVT(col)$ can be converted from B_t as follows:*

(1) For each internal node $b \in B_t$, the encrypted $enc(b)$ takes the form of $\{\mathrm{e(b)}, \mathrm{v_u(b)}, \mathrm{V_O(b)}\}$, where $\mathrm{e(b)} = E^s_{k_i}(b), \mathrm{v_u(b)} = E^a_{pri_i}(\mathrm{hash(e(b))}), v_o(b) = E^s_{k_o}\ (hash(e(b)))$;

(2) For each leaf node $b \in B_t$, b consists of a value sequence $< v_0,\dots,v_k >$. The leaf node in EVT takes the form of $< M\ (v_0),\dots,M\ (v_k)\ >$. Each $\mathrm{M(v_i)}(0 \le \mathrm{i} \le \mathrm{k})$ *is*

Copyright ⓒ 2014 SERSC

a 2-dimensional matrix, with the row as the cell c *whose value equals* $\mathrm{v_i}$ *and the column as the user* u *whocan access a cell with the value* $\mathrm{v_i}$. $k_u(c)$ *containing the nested key of* c *for* u *and its location is stored in the matrix correspondingly.*

The encryption method and the verification action on the B+ tree node are similar to those used in the encrypted cell or block. We omit any discussion due to space limitations. EVT(salary) is illustrated in Figure 4. The internal node is actually protected by the key known to all users. Since the values in the column for B+ tree may be not unique, there exist multiple cells with the same value. In addition, there may be multiple users to access the cell with this value. Therefore, we use a matrix to store these relationships. The user ID sequence in the leaf node can be open to ODB, from which ODB can return the blocks selectivity.

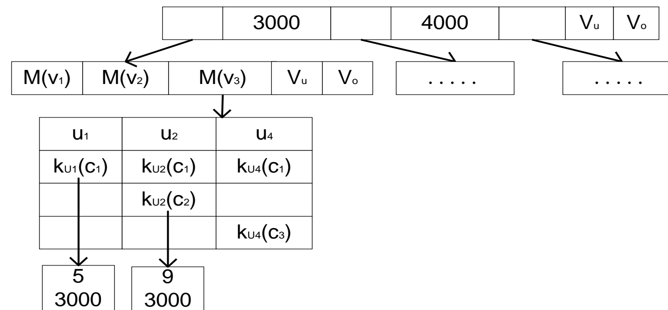

Figure 4. EVT Tree

EVT index can be used to accelerate the query predicate evaluation. Suppose a user u with keys k_i and pub_i attempts to evaluate a query with a predicate on column col, u locates the root node of EVT(col) first, decrypts the root node with k_i, verifies the consistency of the node with pub_i, and then obtains the location for the next internal node according to the predicate. This process repeats itself until the leaf node is reached. u then locates $k_u(\mathrm{c})$ in the matrix. When $k_u(\mathrm{c})$ is decrypted, u can obtain the location and the nested key for the encrypted form of e(c), and further get the cell value. When the value of the cell meets the requirement of the predicate, the tuple ID for the cell is recorded. Compared with the straightforward method, only a small number of blocks need be processed in the presence of the *EVT* index.

In order to locate the target data items after the tuple *IDs* are determined, we also introduce another B+ tree on the tuple *IDs* of each column, named *IT*. Since tuple *IDs* are meaningless, we need not encrypt them and can rely on the functionality of ODB to build the *IT* index.

Definition 7 ID B+ Tree. *Let* col *be a column in a table, the B+ tree*$IT(col)$ *is built with the tuple IDs as keys. For each value*v*in the leaf node in* $IT(col)$,v*is linked to* $K_U(\mathrm{c})$*, where the tuple ID for cell*c*is the same to*v*and the column of*c*is* col. $K_U(\mathrm{c})$ *contains the location of* c *and the nested keys for* $e(c)$.

A user u can request ODB directly to return $K_U(\mathrm{c})$ from IT(col) with a tuple IDi. u can then get the nested keys for the encrypted cell after the decryption of $K_U(\mathrm{c})$ with u.key. Then $u$locates the encrypted form of cell$e(c)$, decrypts it and produces the final results.

With the introduction of EVT andIT, the metadata block for each user needs be extended to record the root nodes of different index trees. Let$u$be a user, $Cols$ be the column set$u$can access (read/update), the meta data block $\mathrm{meta}(u)$ is an encrypted block $E^s_{u.key}(B)$, where *B* is a sequence of *{(*loc_{evt}, pub_i, k_i, loc_{it}*)|*loc_{evt} is the location of $EVT(col)$, pub_i is the

public key to verify the content in $EVT(col)$, k_i is the key to decrypt the node in $EVT(col)$, loc_{it} *is* the location of the root node for $IT(col)$, $col \in Cols\}$.

5. Related Work

The query processing over the encrypted data in ODBs receives a great deal of attention. Sensitive data is always encrypted before uploaded into ODBs, and the straightforward method to query encrypted data is expensive. In order to overcome this limitation, a kind of method is to use some specific encryption methods, such as order preserving encryption [3] to let ODBs evaluate queries directly on the encrypted data. An encrypted B+ tree is proposed in [6], with which the data can be accessed in an interactive way. A privacy-preserving index based on the partition of sensitive attributes is proposed by [9]. Different from existing works, the query processing in our paper considers the access control rules, where different kinds of privileges need be distinguished.

Another hot research line on ODBs is how to ensure the integrity of query results. Since ODBs are assumed not to be fully trustworthy, the client side has to combine the verification object along with the original data, and check the verification object when the results come from ODBs. According to different forms of verification objects, current methods can be classified into the MHT-based approach [8], the probabilistic approach [13], and the chain-based approach [14, 15].

The access control over the relational database has been extensively studied and incorporated into the commercial database. Discretionary access control is used to restrict access to objects based on the identity of subjects and their rights [10]. The distributed access control is studied in [5]. In our paper, we cannot rely on ODBs to check the rights since ODBs are not fully trustworthy. Our access control is also implemented in a distributed way due to the lack of the central support to the access control in ODBs.

The access control on resources in the outsourced server is studied [16].They proposed a novel two-layer encryption, one performed by the data owner to enforce the initial policy and another performed by server provider to enforce the dynamic changes over the policy. The purpose of two-layer encryption is to avoid re-encrypting the original data when the rights for others have been changed. Different from our work, it assumes that ODBs can honestly carry out the instructions. In addition, our paper discusses the fine-grained access control over the table cell and provides SQL query support with two kinds of indexes.

6. Conclusion

In this paper, we study the problem of access control for multiple users in ODBs. The access control rules are specified at the granularity of the table cell. The basic idea is to compile the access control rules on each table cell inside the encrypted data, and rely on different encryption techniques to distinguish the various rights. The cell grouping strategy is used to reduce the impact of the encryptions. Two kinds of indexes, one for predicate evaluation and another for target item location, are proposed to speed up the query processing.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant No.61272241,No.61303085; Science and Technology Development Plan Project of Shandong Province No. 2012GGX10134; Independent Innovation Foundation of Shandong University under Grant No.2012TS075, No.2012TS074.

Copyright ⓒ 2014 SERSC

References

[1] Microsoft sql server data service. http://www.microsoft.com/azure/data.mspx.
[2] Simple database in amazon. http://aws.amazon.com/simpledb/ .
[3] R. Agrawal,J. Kiernan,R. Srikant, andY. Xu. Order preserving encryption for numeric data. In Proc. of SIGMOD, pages 563–574, 2004.
[4] Luc Bouganim, Francois Dang Ngoc, and Philippe Pucheral. Dynamic access-control policies on XML encrypted data. ACMTrans. Inf. Syst. Secur, 10(4):1094–9224, 2008.
[5] BogdanCautis. Distributed access control: a privacy-conscious approach. In Proc. of SACMAT, pages 61–70, 2007.
[6] E. Damiani, S.D.C.Vimercati,andJajodiaS. et.al. Balancing confidentialityandefficiencyin untrusted relational DBMSs. In Proc. of CCS, pages 93–102, 2003.
[7] E.Damiani, S.Vimercati, S.araboschi, and P.amarati. Afine-grained access control system for XML documents. ACMTrans. Inf. Syst. Secur., 5(2):169–202, 2002.
[8] L.Feifei,M. Hadjieleftheriou,G.Kollios,andL.Reyzin. Dynamic authenticated index structures for outsourced databases. In Proc. of SIGMOD, pages 121–132, 2006.
[9] B. Hore, S. Mehrotra, and G. Tsudik. Aprivacy-preserving index for range queries. In Proc. of VLDB, pages 720–731. VLDB Endowment, 2004.
[10] TeresaF. Lunt and EduardoB. Fernandez. Database security. SIGMOD Record, 19(4):90-97, 1990.
[11] Y. Man, Lin. Yimin, and K. Mouratidis. Efficient verification of shortest path search via authenticated hints. In Proc. of ICDE, pages 237–248, 2010.
[12] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In Proc. of VLDB, pages 898–909, 2003.
[13] X. Min,W. Haixun,Y. Jian, andM. Xiaofeng. Integrity auditingof outsourced data. In Proc. of VLDB, pages 782–793, 2007.
[14] E.Mykletun, M.Narasimha, and G.Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In Proc. of ESORICS, pages 160–176, 2004.
[15] M.Narasimha and G.Tsudik. Authentication of outsourced databases using signature aggregation and chaining. InProc. OfDASFAA, pages 420–436, 2006.
[16] S.D.Vimercati, S.Foresti, S. Jajodia, S.Paraboschi, andP. Samarati. Over-encryption management of access control evolution on outsourced data. In Proc. of VLDB, pages 123–134, 2007.

Authors

Lanju Kong, born in 1978, Ph.D. She is an assistant professorin Shandong University and Shandong Provincial Key Laboratory of Software Engineering. Her main research interests include computer software and theory, software and data engineering, XML query and access.

Li Qingzhong, born in 1965, professor, Ph.D. supervisor. His research interests include large-scale network data management and web data integration.

Lilin, born in 1980, master. Her research interests include database.

Copyright ⓒ 2014 SERSC