The ISC Int'l Journal of Information Security

July 2011, Volume 3, Number 2 (pp. 121–133)

http://www.isecure-journal.org

Separating Indexes from Data: A Distributed Scheme for Secure Database Outsourcing

Somayeh Soltani [1,*], Mohammad Ali Hadavi [1], and Rasool Jalili [1]

[1] *Data and Network Security Laboratory, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

ARTICLE INFO.

Article history:
Received: 19 September 2010
Revised: 26 October 2011
Accepted: 29 January 2012
Published Online: 19 May 2012

Keywords:
Database Security, Database Outsourcing, Encrypted Database, Query on Encrypted Data.

ABSTRACT

Database outsourcing is an idea to eliminate the burden of database management from organizations. Since data is a critical asset of organizations, preserving its privacy from outside adversary and untrusted server should be warranted. In this paper, we present a distributed scheme based on storing shares of data on different servers and separating indexes from data on a distinct server. Shamir's secret sharing scheme is used for distributing data to data share servers. A B^+-tree index on the order preserved encrypted values for each searchable attribute is stored in the index server. To process a query, the client receives responses including record numbers from the index server and asks these records from data share servers. The final result is computed by the client using data shares. While the proposed approach is secure against different database attacks, it supports exact match, range, aggregation, and pattern matching queries efficiently. Simulation results show the prominence of our approach in comparison with the bucketing scheme as it imposes lower computation and communication costs on the client.

1 Introduction

Improvements in the network technology and the possibility of connecting all computers around the world have changed many conventional methods of providing services for users. An instance of such a change is outsourcing organizational databases to remote database service providers instead of conventional in-house database management. This results in organizations to concentrate just on their own core business. The *Database as A Service* (DAS) model [1] is a noteworthy solution due to reducing data management costs in addition to more quality services such as database availability.

The concept of *Database As a Service*, has been studied by many researchers since its introduction [1–7]. The DAS model introduces new security challenges. In addition to providing data confidentiality against outside attackers, it is needed to protect data confidentiality against untrusted servers. In this paper, as many other researches, we assume honesty of the server in correctly replying the users' queries, i.e. the complete set of results will be returned without unauthorized manipulations in response to a query. However, the server is curious and tries to enhance its knowledge about the stored data.

The preliminary and straightforward solution to protect data against the untrusted server is data encryption. If so, there is a challenging question: *If the*

* Corresponding author.
Email addresses: s.soltani@srbiau.ac.ir (S. Soltani), mhadavi@ce.sharif.edu (M. A. Hadavi), jalili@sharif.edu (R. Jalili).

ISSN: 2008-2045

server is not allowed to decrypt the data, how it can respond to the users' queries? A naïve solution is to send the whole encrypted database to the user-side in response to their queries. It imposes the whole database decryption on the client-side in addition to the client-side query processing. To overcome the problem, different schemes were proposed for query processing directly over the encrypted data. Some approaches are based on storing some metadata in addition to the encrypted data on the server-side. Other approaches use special kinds of encryption schemes and some of them use fragmentation techniques instead of encryption techniques to conceal the confidential relations between attribute values. We briefly review these approaches in Section 2.

In the data outsourcing scenario, an adversary as well as the untrusted server may know either the distribution of the plaintext database or the whole plaintext database as background information. The latter case may occur if the organization switches from an in-house plaintext database to an outsourced one [8]. If the adversaries obtain some extra information relying on their background information, it is referred to as either *frequency attack* or *known database attack*. Obviously, if an outsourcing scheme is vulnerable to the frequency attack, it is also vulnerable to known database attacks.

A database outsourcing solution should consider practicality issues such as supporting different kinds of query on different data types while respecting security concerns, which threaten confidentiality and integrity of the outsourced data. Whereas some approaches consider practicality aspects of database outsourcing, other approaches concentrate solely on security. In database outsourcing as well as many other contexts, usually there is a tradeoff between security and efficiency. To the best of our knowledge, none of the proposed approaches satisfy all aspects of practicality issues, along with considering the security aspects.

In this paper, we introduce a new approach for secure database outsourcing based on distributing data on several servers and separating indexes from data. Our proposed approach supports different kinds of query including exact match, range, aggregation, and pattern matching queries. Our scheme is secure against known database and database frequency attacks, produces no false hits, is efficiently updatable, and imposes admissible client-side computation and communication overheads. It uses the Shamir secret sharing scheme [9] to securely distribute data among some honest-but-curious servers. Indexes for different searchable attributes are stored on a different index server.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 explains the concept of secret sharing and briefly introduces the Shamir secret sharing scheme. Section 4 explains our proposed approach in detail. Section 5 analyzes security of our approach along with comparison and simulation results. Concluding the paper, we mention some future work in Section 6.

2 Related Works

Majority of the related works in the context of secure data outsourcing focus on data confidentiality against the untrusted server, assuring query result correctness, users' access privacy, or enforcement of access control policies in a multi user environment. As our approach concentrates on data confidentiality, we briefly review confidentiality concerned approaches.

Index-based methods, are the most popular methods for the confidentiality problem in data outsourcing. In these methods, some metadata is stored on the untrusted server as index along with the encrypted outsourced data. The indexes are used for query processing over the encrypted data. There are different index-based schemes based on the index construction methods. The construction method should consider security as well as the efficiency of query processing. In other words, while the curious server should efficiently execute submitted queries on the outsourced data, it must not be able to increase its knowledge about the plain data using the assigned indexes.

Hacigümüs *et al.* [1] introduced the bucketing index-based method. They partition an attribute domain into some buckets and assign a random tag to each bucket as an index. Exact match and range queries are processed in this scheme with considerable false hits on the server-side. Hore *et al.* [4] propounded a bucketing algorithm to minimize the number of false positives over the set of all range queries. SAM [10], as another bucketing algorithm, concentrates on security with the idea of maximizing bucket entropy. Some other approaches propose techniques for modeling the tradeoff between query performance and data secrecy [4, 6, 11].

Damiani *et al.* [8] compare three indexing methods: encrypted field as index, indexing with hash functions, and B^+-tree index made on plaintext values. While the encrypted attribute value as index makes no false hits, it is not secure against the frequency attack. Though using hash functions for index construction is more secure, it partly deteriorates the efficiency and makes range query processing inefficient. On the other hand, the B^+-tree, made over plain-text data, encrypted by the data owner, and saved on the untrusted server, supports range queries. Although the B^+-tree index

structure is secure, it imposes high computation and communication overhead on the client-side.

In addition to the index-based methods, other approaches to support data confidentiality in database outsourcing scenarios have been proposed. Privacy homomorphism and order preserving encryption functions are among the proposed methods used for direct execution of aggregation and range queries over encrypted data, respectively. Privacy Homomorphism (PH) [12–16] is an encryption function which maps a set of operations on plain-text values to another set of operations on cipher-text values. It has been considered by Hacigümüs *et al.* [17] to support aggregation queries. Mykletun *et al.* [5] demonstrated that the encryption scheme in [17] can be broken by a cipher-text-only attack and proposed another solution to support aggregation queries. Ahituv *et al.* [12] illustrated that a PH with addition as one of its cipher-text operations is insecure against chosen-plain-text attack [18]. The anti-tamper database [3] utilizes order preserving encryption functions for processing range queries. Agrawal *et al.* [19] proposed an Order Preserving Encryption Scheme (OPES) in which the distribution of encrypted values is independent from the distribution of plain-text values and follows a user desired distribution. OPES focuses on supporting range queries for numeric data. It works in three stages: model, flatten, and transform. As these encryption schemes preserve the order of plain values in the encrypted ones, they are vulnerable to the frequency attack. Mansouri proposed ROPES [20] by adding a new randomization stage to OPES in order to make it more secure against this kind of attack. Splitting, scaling, and noise [7] are the three stages in another order preserving encryption model in which each plaintext value is mapped to some different encrypted values. This method is secure against the frequency attack.

Another approach to preserve confidentiality of outsourced data is based on fragmentation. The rationale behind fragmentation is that sometimes data confidentiality refers to the associations between attribute values not to the attribute values themselves. For example, in a typical health database, neither of the patient's NAME nor the patient's DISEASE is confidential, but the association between values of NAME with their correspondent values of DISEASE is absolutely confidential. Fragmenting a relation into several parts is aimed at concealing these associations based on predefined security constraints. Vertical fragmentation of a relation was proposed first by Aggarwal *et al.* [21]. In their solution, a relation is partitioned into two fragments based on predefined security constraints over attribute values. Then, two fragments are outsourced to two non-communicating servers. In Aggarwal *et al.*'s solution, queries are executed more efficiently on plain-text values of different partitions rather than on encrypted values. In cases of having sensitive attributes, encryption is used to keep their values secret. Obviously, the fragmentation must be lossless such that the data owner can reconstruct the original relation from its constituent fragments. To complete departure from encryption operations, Ciriani *et al.* [22] proposed to store one fragment on the owner side. They defined an optimized fragmentation so that it minimizes the storage and computational costs for the owner. In Aggarwal *et al.*'s solution, a relation must be divided into two partitions and the servers storing these partitions are not allowed to have any communication, which is a strong limitation in operational environments. Considering these problems, Ciriani *et al.* redefined the optimized fragmentation such that it minimizes the number of fragments and maximizes the number of unencrypted attributes [23]. In [24] they proposed a solution to minimize the query execution cost as another fragmentation criteria.

Agrawal *et al.* [2] used the concept of threshold secret sharing to securely distribute data among untrusted servers. Their method provides an efficient solution for the exact match, range, and aggregation queries. However, it is susceptible to the frequency attack. Brinkman *et al.* [25] used secret sharing to securely store and query the tree structured data, such as XML.

Using the Shamir threshold secret sharing scheme and separating indexes from data, we propose a secure and efficient method to distribute data between untrusted servers. Our method is secure against known database and database frequency attacks while it efficiently supports different kinds of query including exact match, range, pattern matching, and aggregation queries on numeric and string data types.

3 Preliminaries

Secret sharing was proposed separately by Blakley [26] and Shamir [9]. The secret v is shared out to a set of participants $P = \{P_1, \ldots, P_n\}$ so that only authorized sets of participants can compute v. A secret sharing scheme is called a threshold (k, n), if k out of n participants can compute the secret v by pooling their shares.

Shamir secret sharing scheme is a secure threshold (k, n) scheme. Security is achieved if an attacker cannot access k different shares of a secret. To compute share values of a secret v in the Shamir threshold scheme (k, n), a vector $X = \{x_1, \ldots, x_n\}$ for n participants and a polynomial $f(x)$ of order $k - 1$ are selected such that the constant value of $f(x)$ is equal to v. For example, consider the secret $v = 11$ is to be shared among $n = 5$ participants with $k =$

3. We define vector $X = \{x_1 = 3, x_2 = 1, x_3 = 5, x_4 = 2, x_5 = 4\}$ and $f(x) = 3x^2 + x + 11$. Each participant's share is computed by putting x_i $(1 \leq i \leq 5)$ in $f(x)$, e.g. $share(11,1) = f(x_1) = 41$, $share(11,2) = 15$, $share(11,3) = 91$, $share(11,4) = 25$, and $share(11,5) = 63$.

The secret v can be retrieved by the Lagrange interpolation as

$$v = f(0) = \sum_{j=1}^{k} \left(f(x_{i_j}) \prod_{\mu=1,\mu\neq j}^{k} \frac{0 - x_{i_\mu}}{x_{i_j} - x_{i_\mu}} \right).$$

Each of the k participants can use the above equation to compute v. The Shamir secret sharing is additive homomorphic [27], i.e. sum of the secrets can be computed by utilization of the Lagrange interpolation on summation results of each participant shares. This property leads to support aggregation queries with SUM and AVG aggregation functions in our approach.

4 The Proposed Approach

Our proposed approach for secure database outsourcing is based on both distributed databases and secret sharing concepts. Organizational data is distributed to several servers which we call *Data Share Servers* (DSSs), using Shamir secret sharing. Indexes for this data are stored on a separate server called *Index Server* (IS). All the above servers are supposed to be honest-but-curious [28].

Unlike the previous approaches in which the server containing the data was responsible for searching in appropriate indexes and returning the response, in our approach a considerable amount of query processing is done by the IS. In the following subsections, we explain the principal components, the way they interact to process a query, and how the character data is handled to support pattern matching queries over string data types.

4.1 Data Share Servers (DSSs)

In a threshold (k, n) secret sharing, n Data Share Servers (DSSs) store shares of an attribute value (secret) v computed by $f(x)$ of order $k-1$ and vector X with cardinality n. The vector X is selected by the data owner and kept hidden from the untrusted servers. Therefore, DSSs cannot retrieve the secret even if they can collect k or more of their shares. The distribution function $f(x)$ is chosen randomly by the data owner such that the constant value of $f(x)$ is equal to the secret. Share value of each DSS is calculated by $f(x_i)$ where x_i is the corresponding member of the vector X. Consider the Employee relation with

ID	Name	Age	Salary
1	Elvis	45	10
2	John	84	40
3	Chris	78	60
4	Frank	46	20
5	Bob	45	60
6	Alice	80	20
7	Henry	45	80
8	Ian	57	60
9	Gary	57	10
10	Donna	46	40

Figure 1. The Employee Relation

10 tuples in Figure 1, and assume each attribute value is distributed among three DSSs with $n = k = 2$ and $X = \{3, 2\}$. Figure 2 depicts the mapping of the outsourced Employee relation in Figure 1 onto two DSSs. We leave discussion about the string attributes (Name in this example) for Section 4.5.

In this approach, the randomized coefficients of $f(x)$ result in different share values for the same attribute values. For example, although the Salary values of the second and the tenth tuples are equal in the Employee relation, they map onto different share values in DSSs. This property disarranges the attribute values so that the order of plaint-text attributes is not preserved in their shares in DSSs. However, the order of records in all DSSs must be the same, i.e. shares of a secret value on different DSSs are located in the same place in terms of record number. A connection-oriented communication like TCP is assumed between DSSs and the client to preserve the order of response packets. This is not a painful assumption in conventional software implementations.

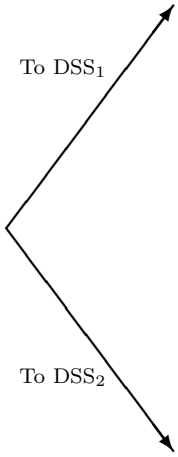
The number of DSSs and the threshold can be chosen by the client. The minimum number of DSSs is 2. While increasing the number of DSSs yields more accessibility, it incurs more computation and communication cost on the client-side.

4.2 Index Server (IS)

The Index Server (IS) is responsible for maintenance of B^+-tree indexes made over the encrypted attribute values. The B^+-tree indexes are made for the attributes that appear in query conditions. To construct a B^+-tree index for an attribute, the attribute values are encrypted by an order preserving encryption function. Then, a B^+-tree is made over the encrypted values. Each leaf pointer in the tree refers to a bucket containing record numbers having the same value of the leaf key. The buckets are encrypted by a common encryption function such as a symmetric encryption method. Finally, the index tree and the encrypted buckets are sent to the IS.

The above process is just for initializing the B^+-tree

ID	Name	Age	Salary
1	Elvis	45	10
2	John	84	40
3	Chris	78	60
4	Frank	46	20
5	Bob	45	60
6	Alice	80	20
7	Henry	45	80
8	Ian	57	60
9	Gary	57	10
10	Donna	46	40



ID	Att_1	Att_2	Att_3
1	E(1,1,Elvis)	57	31
2	E(1,2,John)	93	64
3	E(1,3,Chris)	105	72
4	E(1,4,Frank)	73	26
5	E(1,5,Bob)	63	87
6	E(1,6,Alice)	95	44
7	E(1,7,Henry)	60	83
8	E(1,8,Ian)	75	84
9	E(1,9,Gary)	84	37
10	E(1,10,Donna)	55	55

ID	Att_1	Att_2	Att_3
1	E(2,1,Elvis)	53	24
2	E(2,2,John)	90	56
3	E(2,3,Chris)	96	68
4	E(2,4,Frank)	64	24
5	E(2,5,Bob)	57	78
6	E(2,6,Alice)	90	36
7	E(2,7,Henry)	55	82
8	E(2,8,Ian)	69	76
9	E(2,9,Gary)	75	28
10	E(2,10,Donna)	52	50

Figure 2. Outsourcing the Employee Relation to Two Data Share Servers in the Proposed Scheme

indexes. Thereafter, the IS itself is in charge of any insertion, deletion, or update. In the next subsection, we will examine record insertion, deletion, and update and the IS role in these processes.

Considering the sample Employee relation in Figure 1, Figure 3 shows the B^+-tree made over encrypted values of the Age attribute. E_{OP} in Figure 3 denotes the order preserving encryption and E indicates a common encryption method. Leaves in the tree have pointers to the buckets containing appropriate record numbers. The rightmost pointer of each leaf refers to the next leaf, except the last leaf whose rightmost pointer is null.

4.3 Query Processing

A general scenario of query processing in a typical DAS model is as follows. The user's query is submitted via a client and translated into the server's familiar format. The server processes the encrypted data and sends the results back to the client. Subsequently, the client decrypts the results and probably does some post processing on them to show the final results to the user.

In our approach, when a user issues a query by a client, the query predicate values are encrypted in an order preserving manner and sent to the IS. Subsequently, the IS searches in the relevant index and returns the proper encrypted bucket(s) to the client. The client decrypts the buckets and extracts record numbers satisfying the query conditions. Then, the client sends a query to each DSS in order to retrieve the corresponding records. On receiving at least k DSSs responses, the final result is obtained through applying the Lagrange interpolation. Figure 4 demonstrates the procedure of query processing in the proposed scheme.

In the above scenario, processing a query does not require computing data shares on DSSs. So, it does not need to find coefficients of polynomials making data shares.

4.3.1 Exact Match Query

We explain the process of an exact match query in our approach by an example. Consider a typical query `SELECT Salary FROM Employee WHERE Age=45`. In order to process the query, the value 45 is encrypted by E_{OP} and sent to the IS. The IS searches $E_{OP}(45)$ in the B^+-tree index and returns the corresponding bucket $E(1, 5, 7)$ to the client (Figure 3). Subsequently, the client decrypts the bucket and extracts the record numbers 1, 5, and 7. Now, the client requests DSSs to retrieve the third attribute value (Att_3) of records 1, 5, and 7. In response to this request, DSS_1 and DSS_2 return their share values. In our example, the client receives the values 31, 87, and 83 from DSS_1 and 24, 78, and 82 from DSS_2. Putting the values 31

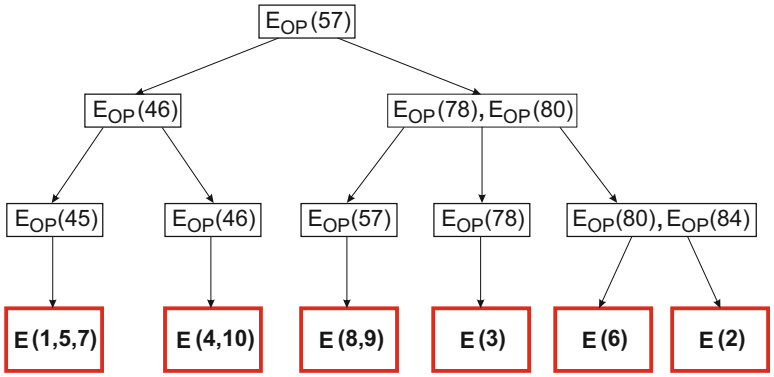


Figure 3. The B^+-tree Index over the Age Attribute

and 24 in the Lagrange interpolation results in the value 10 which is the Salary value of the first record. The other result is computed similarly.

4.3.2 Range Query

While in other schemes the order of data should be preserved on the data server to support range queries, data shares in our approach do not need to have the same order as the original data. This is due to the fact that range queries are processed using B^+-tree indexes on the IS. Accordingly, it does not require preserving neither frequency nor the order of the original data on DSSs.

Processing range queries in our approach is as follows. At first, the IS finds the leaf containing the biggest value smaller or equal to the lower bound of the requested range in the B^+-tree index. Then, it traverses from this leaf to the next leaf as long as the leaf's key is smaller than the upper bound of the range; and sends the proper buckets to the client. The remaining process is similar to the exact match queries.

4.4 Aggregation Query

Our approach supports aggregation queries efficiently. Simple MIN, MAX and COUNT queries do not need communication with DSSs. For example, the query `SELECT COUNT(*) FROM Employee WHERE Age = 57` is processed just by searching in the B^+-tree index. However, more complicated queries like `SELECT Salary FROM Employee WHERE Age = MAX (Age)` need to retrieve responses from DSSs.

Since Shamir secret sharing is additive homomorphic, SUM or AVG queries can be executed on data shares without any decryption. For example, to process the query `SELECT SUM(Salary) FROM Employee WHERE Age=45`, the client requests DSSs to send back sum of Att_3 share values of the appropriate records. In other words, the client sends each DSS a query like `SELECT SUM(`Att_3`) of records 1, 5 and 7`. DSS_1 sends the value 201 to the client as the summation of 31, 87, and 83. Similarly, DSS_2 sends the value 184 to the client as well. The client puts these values in the Lagrange interpolation formula and calculates the final value 150.

4.4.1 Record Insertion, Removal, and Update

To insert a new record into a relation, in addition to insertion of the new record in each DSS, some modifications are necessary in the indexes. For any indexed attribute of the inserted record, the IS finds the leaf into which the new value must be inserted. If the value does not exist in the tree, it is inserted into the tree, a new bucket is created, and the corresponding record number is inserted into the bucket. If the value is repetitive, it exists in the tree and the corresponding inserted record number is appended to the related bucket. Insertion of the new record number into a bucket needs interference of the client. In such a case, the IS sends the encrypted bucket to the client. The client decrypts the bucket, updates it properly, encrypts the bucket again, and finally sends it back to the IS. Finally, the received bucket is replaced by the old one.

A straightforward method for record removal is to remove the record from the DSSs and modify appropriate indexes so that the specified record number is removed from the bucket. In the case of an empty bucket (after this removal), the leaf corresponding to the removed attribute value must be removed from the index tree. As physical record removal leads to change in subsequent record numbers and incurs the indexes to be changed entirely, we apply logical deletion and use a Boolean field for each record to indicate whether it is deleted or not. So, the physical record removal with undesirable consequences is ignored.

To update a value of a searchable attribute in a record, it is required to modify the appropriate index. For each index, the record number should be omitted

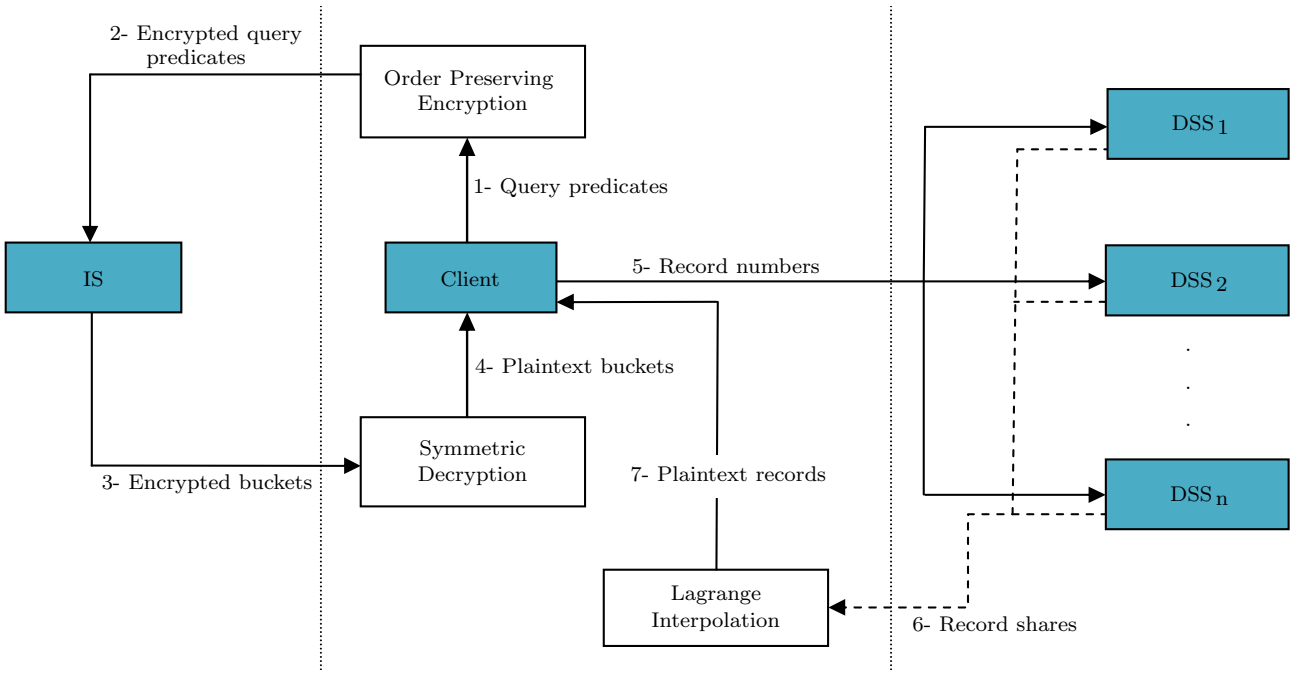


Figure 4. Query Processing Scenario in the Proposed Approach

from the previous containing bucket and inserted to the new proper one.

4.5 Supporting String Data

In our approach, the character data type is handled relatively similar to the numeric one. For a string attribute, an index is made over the encrypted values. A string value is encrypted using an order preserving encryption function on the ASCII codes of its characters. The encrypted value of the string data is stored on DSSs.

Figure 5 shows the B^+-tree index on the Name attribute of the Employee relation of Table 1. The B^+-tree index on the order preserved encrypted values of this attribute is shown in Figure 6. As we can see in Figure 6, for each string value the ASCII code of its characters are encrypted by an order preserving encryption function and a separator (e.g. ',') is inserted between each two encrypted characters.

The order of string values is preserved in the B^+-tree index. Although the alphabetical statistics may reveal some encrypted values, the adversary cannot infer any more information because there is not any explicit correlation between the index values and the other attribute values of a record. However, in order to make alphabetic statistical analysis harder, a different key can be used for each index.

String values are encrypted in a different way from numeric values and stored on DSSs as depicted in Figure 2. To map repetitive data values onto different values, we encrypt record number together with the data itself. To have different values on different DSSs, we can add the server number and encrypt them altogether. However, if availability is not of the main concern, it is sufficient to store the encrypted string attribute on one of the data share servers.

To do exact match or range queries on string attributes, the string value of the query predicate is decomposed to its constituent characters. The ASCII code of each character is encrypted by the order preserving encryption function. Then, the set of encrypted characters is sent to the IS as a single value. Subsequently, the IS searches for this value and returns the proper bucket(s). Finally, the client requests one of the DSSs to retrieve the encrypted string value of the specified records. The final result is obtained by decrypting the returned response from the DSS.

Execution of pattern matching queries is possible in this scheme as well. For example, consider the query `SELECT Name FROM Employee WHERE Name=%an%`. This query can be processed by mapping *an* to *E(97),E(110)* as its encrypted form and sending it to the IS. The IS searches in the keys of the leaves for this encrypted pattern and returns the appropriate bucket(s) if any. Afterwards, the client follows a process similar to exact match or range queries on the string attribute.

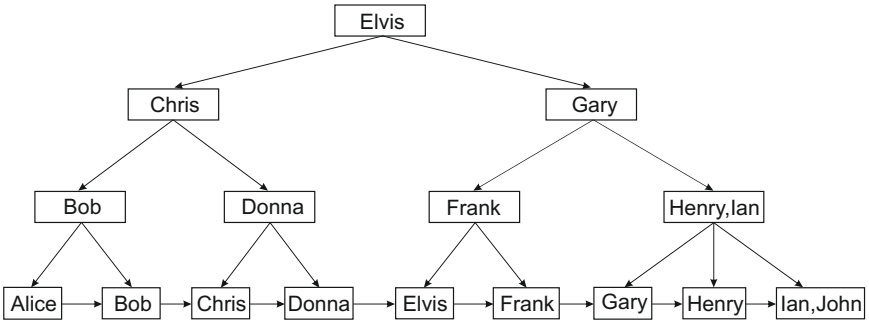


Figure 5. B$^+$-tree Index over the Name Attribute of the Employee Relation

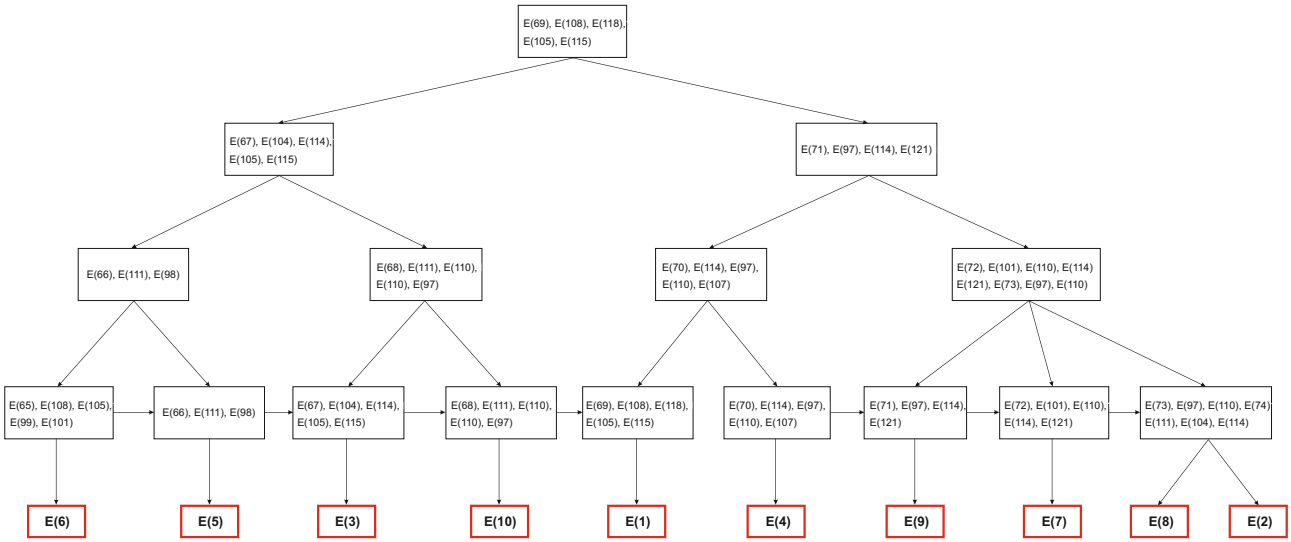


Figure 6. B$^+$-tree Index over Order Preserved Encrypted Values of the Name Attribute of the Employee Relation

5 Analysis and Simulation

In this section we analyze the security of our approach and compare its computation and communication costs in comparison with bucketing index-based approach with respect to the simulation results.

5.1 Security Analysis

In the proposed scheme, data confidentiality against outside attackers and untrusted servers is preserved. Threshold (k, n) Shamir secret sharing scheme provides confidentiality of our approach against the untrusted servers even if they access $k-1$ shares of data or more. This is because the vector X is kept hidden by the owner. Moreover, randomized coefficients of the polynomial $f(x)$, as defined in Section 3, provide protection against statistical attacks in database context based on previous knowledge of an attacker about the outsourced data. This is because the data shares on the DSSs have different frequencies from plain data and also the order of data shares on the DSSs is different from its corresponding plain data. For example, consider the data shares of DSS_1 in Figure 2. The Salary value of 60 is mapped onto three different values 72, 87, and 84. Two of these values are bigger than the share value of 80 in the seventh record, that is, 83. The knowledge of adversaries about the frequency of the value 60 or its order in the plain-text database does not help them to map some data shares onto their corresponding plain values.

To determine the resistance of an outsourcing scheme against frequency-based attacks, Damiani *et al.* [8] define the exposure coefficient ϵ. The inference exposure model in their work is based on the one-to-one correspondence between the RCV-graphs of the plain-text and encrypted databases. To compute the exposure coefficient for an outsourced relation, an equivalence relation with equivalence classes for each attribute is defined. Values with the same number of occurrences are put in the same equivalence classes. In our example, equivalence classes of the outsourced Employee relation on DSS_1 are:

$Att_1.1$ = {E(1,1,Elvis), E(1,2,John), E(1,3,Chris), E(1,4,Frank), E(1,5,Bob), E(1,6,Alice), E(1,7,Henry), E(1,8,Ian), E(1,9,Gary), E(1,10,Donna)}

$Att_2.1$ = {57, 93, 105, 73, 63, 95, 60, 75, 84, 55}

$Att_3.1 = \{31, 64, 72, 26, 87, 44, 83, 84, 37, 55\}$

$Att_1.1$ means the values of Att_1 with occurrence 1. Exposure rate of each value is the inverse of the cardinality of the equivalence class to which the value belongs. For example, an adversary can map the value 57 of Att_2 to the value 45 of the Age attribute with probability $\frac{1}{10}$. Yet it requires the adversary to know the correspondence between Att_2 and Age. For a specific association, the product of the inverses of the cardinalities shows the disclosure probability. Associating the pair (57, 31) of a tuple in the outsourced relation to (45, 10) of a tuple in the original relation is done with probability $\frac{1}{100}$. The exposure coefficient ϵ is the probability of disclosing the whole relation, which is defined by the average exposure rate of its tuples. For a relation with m tuples and l attributes, the exposure coefficient ϵ is:

$$\epsilon = \frac{1}{m} \sum_{i=1}^{m} \prod_{j=1}^{l} e_{i,j}, \quad (1)$$

where $e_{i,j}$ is the exposure rate of each value defined as the inverse of the cardinality of the equivalence class to which the value belongs. In our scheme, since repetitive values of the original relation are mapped to random and different share values, the exposure coefficient becomes $\epsilon = \frac{1}{m^l}$. This relation says that the insertion of a new record decreases the exposure coefficient.

Although the order of attribute values is preserved in the IS, an adversary as well as the untrusted IS cannot infer any relation between the encrypted values on the IS and the data shares on the DSSs due to the encrypted form of the buckets. However, the IS might infer the frequency of attribute values from the volume of the buckets. To solve this problem we can use some implementation-level mechanisms such as padding to equalize the bucket volumes.

In our approach, the DSSs are not aware of the query contents and act just as storage mediums returning requested records by record identifiers. However, they may collude with the IS to discover queries; i.e. for a query, the IS knows the encrypted value of the index and the DSSs know the response buckets. They may collude with each other to know some correlation between encrypted index values and data shares. To prevent the collusion, we simply scatter the order of requests to the IS and the DSSs.

5.2 Cost Analysis

In the cryptography-based approaches, the client-side decryption is the most time consuming part of query processing. So, we consider the number of encryption/decryption operations as a measure to compare client-side computation cost of the proposed approaches.

To execute a query in a bucketing index-based scheme [1], encrypted records that match the query conditional clauses are returned to the client. Client decrypts the records and post-processes the result, if necessary. In the proposed scheme, instead of encrypted records, the encrypted buckets containing the record numbers are sent back to the client. Client decrypts them and finds the record numbers which must be fetched from DSSs. While in the bucketing scheme [1], the received records are decrypted on the client-side, in our approach decryption is performed on the buckets containing record numbers. Consider the query `SELECT Salary FROM Employee WHERE Age=45` in our example. Assuming no server-side false hits, the bucketing scheme returns three encrypted records 1, 5 and 7. But the proposed scheme returns one encrypted bucket which contains $E(1, 5, 7)$.

Consider an attribute $att \in \{v_1, v_2, \ldots, v_k\}$ and frequency $freq \in \{f_1, f_2, \ldots, f_k\}$ which shows the frequency of each attribute value. To process a query with $v_x \leq att \leq v_y$ condition, the bucketing scheme requires $\sum_{i=x}^{y} f_i$ decryption operations on the client-side as the number of returned encrypted records. On the other hand, our scheme requires $\sum_{i=x}^{y} 1$ decryption operations as its number of returned buckets. In our example, the Age attribute contains the values $Age = \{45, 46, 57, 78, 80, 84\}$ with frequencies $freq = \{3, 2, 2, 1, 1, 1\}$. For the query `SELECT Salary FROM Employee WHERE` $45 \leq$ `Age` ≤ 57, an optimized bucketing scheme returns $\sum_{i=1}^{3} f_i = 7$ encrypted records and our scheme returns $\sum_{i=1}^{3} 1 = 3$ encrypted buckets.

The frequency of the values in the database affects the computational cost of our scheme so that more frequent values will result in fewer buckets. This is because of sitting more record numbers in each bucket. The smaller the number of received buckets, the fewer is the decryption overhead on the client-side. In the worst case where the frequencies of all values are one, the computational cost of our scheme is the same as the optimized bucketing scheme.

5.3 Experimental Results

We implemented our approach and also the bucketing index-based method [1] to empirically compare both the computation and communication overheads. Our experiment has been performed on a computer system having Intel Core2 2.5GHz, 3.0GB RAM, and running Windows XP 2008 as its operating system. Development environment consists of C#.Net 2008 and SQL Server 2005. The Sales.SalesOrderHeader

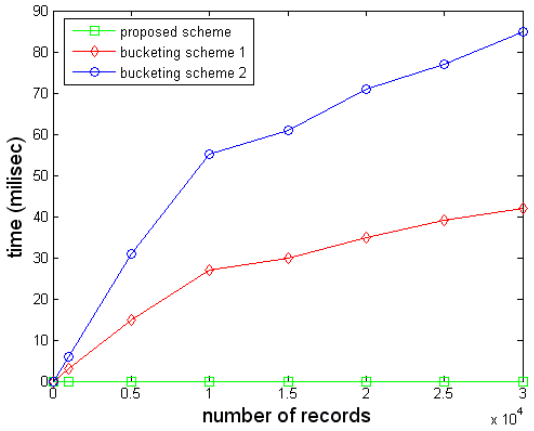


Figure 7. Execution Time of the Sample Exact Match Query

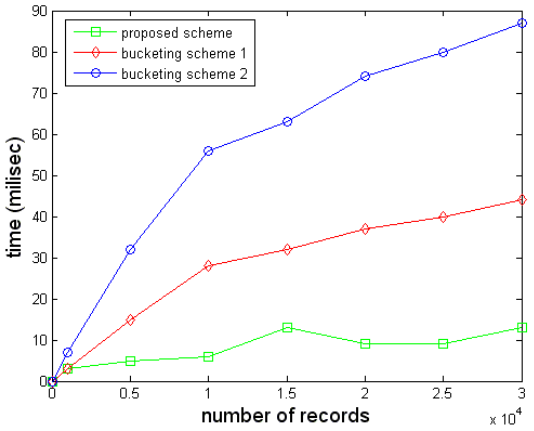


Figure 9. Execution Time of the Sample Aggregation Query

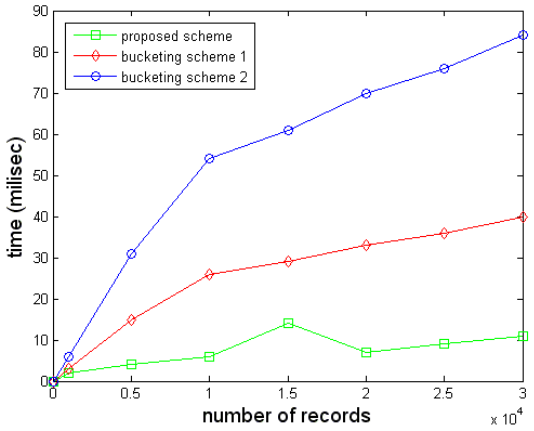


Figure 8. Execution Time of the Sample Range Query

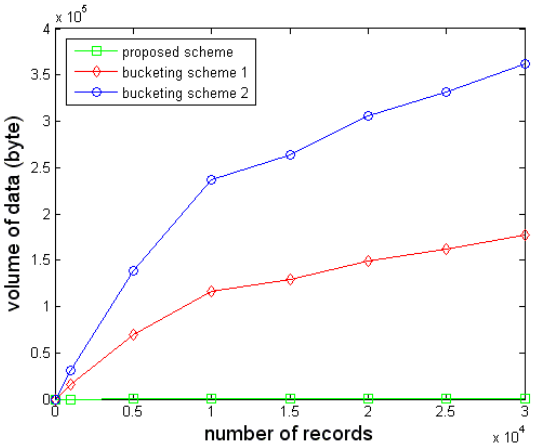


Figure 10. Volume of Transferred Data between the Client and Servers While processing the Sample Exact Match Query

relation of the AdventureWorks [1] database having 31465 records was used. We carried out the insertion of CustomerID and BillToAddressID attributes of the Sales.SalesOrderHeader relation. We also conducted query processing on these attributes. OPES [19] as an order preserving encryption function, DES as a symmetric encryption algorithm to encrypt the buckets, the Lagrange interpolation function, and different functions of a B^+-tree data structure were implemented in our empirical study. We used two different widths 100 and 200 for bucketing CustomerID attribute values as *bucketing scheme 1* and *bucketing scheme 2*, respectively.

To compare our approach with different bucketing schemes in terms of computation and communication overheads, we executed a sample exact match query, range query, and an aggregation query on our data set. Figure 7 shows the execution time of the exact match query `SELECT BillToAddressID FROM Sales.SaleOrderHeader WHERE CustomerID = 430`.

[1] Available at: http://www.codeplex.com (visited on 2009/1/10).

As depicted in this figure, the execution time in our approach is quite less versus the other two bucketing schemes with different bucketing widths per different numbers of records. Figure 7 also certifies that the *bucketing scheme 2*, having greater bucketing width, executes in a longer time compared to the *bucketing scheme 1* due to more false hits generated in the *bucketing scheme 2*.

Figure 8 shows the results of processing a sample range query `SELECT BillToAddressID FROM Sales.SaleOrderHeader WHERE 500` $\leq$ `CustomerID` $\leq$ `600`. The figure confirms that our approach has less computation overhead compared to the two bucketing schemes for a typical range query. As an important point, we consider *bucketing scheme 1* as the optimized bucketing scheme for this query because it generates no false hits. Therefore, we can conclude that the execution time of this query for our approach is less than of bucketing schemes with any bucketing widths.

Consider the sample aggregation query `SELECT`

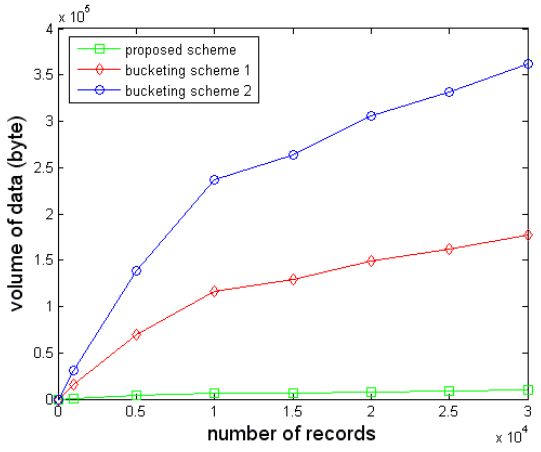


Figure 11. Volume of Transferred Data While Processing the Sample Range Query

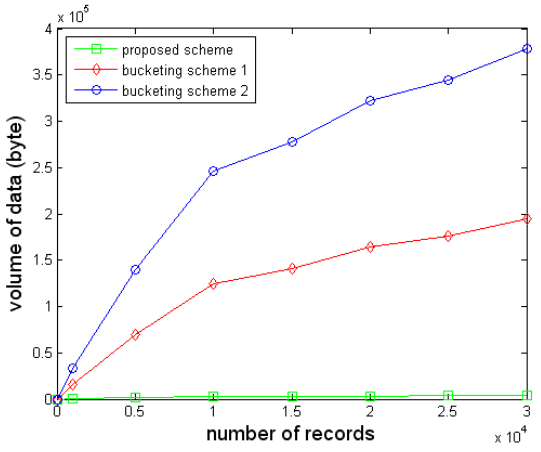


Figure 12. Volume of Transferred Data While Processing the Sample Aggregation Query

`SUM(BillToAddressID) FROM Sales.SaleOrderHeader WHERE 200` $\leq$ `CustomerID` $\leq$ `300`. Figure 9 depicts a similar comparison result in terms of the query execution time. The results of our empirical study approve the correctness of our analysis on computation cost in the previous section.

With $n = 3$ (the number of DSSs) and $k = 3$, Figures 10, 11, and 12 show the volume of transferred data as communication overhead between the client and server(s) while executing the above sample queries. The transferred data in our approach generally is both the encrypted buckets and data shares. These figures confirm that the communication overhead in our approach is significantly less than those of the bucketing schemes, though the client communicates with an IS and three DSSs. With regard to having no false hits, *bucketing scheme 1* has the least transferred data among other bucketing schemes for the typical range and aggregation queries. Since the transferred data in our approach is less than that in *bucketing scheme 1*, it is less than all other bucketing schemes.

5.4 Comparison

Different properties of our approach in comparison with similar confidentiality concerned work in the area of secure database outsourcing are summarized in Table 1. Our approach supports different kinds of queries containing exact match, range, aggregation, and pattern matching on string and numeric data types. There is no post processing on the client-side other than executing an interpolation function on received responses from DSSs without any false hits. Security is an advantage of our approach as well. While other schemes suffer from either statistical attacks on confidentiality of data or having high computation overhead, our approach is protected against such attacks in addition to having a low computation overhead. As another advantage, the outsourced data can be updated without imposing a significant overhead on the client and servers. Computation and communication overheads mentioned in Table 1 indicate the client-side overhead because in data outsourcing scenario usually it is assumed that the client (unlike the server(s)) has limited computational and storage resources.

6 Conclusion and Future Work

We presented a distributed scheme for secure database outsourcing based on the idea of threshold secret sharing and separating indexes from data. Distribution of data among several servers led to supporting different kinds of query on both numeric and string data types with low computation and communication costs. Distributing the database between n servers and retrieving the responses from k ones arise some issues such as availability, communication overhead, and storage cost on the server-side. As future work, defining a tradeoff between these issues helps organizations to choose a desirable model with respect to the values of n and k. Outsourcing the access control to the untrusted server in a multi user environment and improving the proposed approach to assure query result correctness are among our future work in order to put the idea of database outsourcing into practice.

Acknowledgement

This work was supported in part by Iran Telecommunication Research Center (ITRC) for which the authors would like to acknowledge.

Table 1. Comparison of Important Confidentiality Concerned Proposals for Secure Data Outsourcing

			Approaches				
			Bucket-Based Indexing	B^+-Tree Indexing	OPES	Secret Sharing-Based [1]	Our Proposed Approach
Comparison Criteria	Query Support	Exact Match	Yes	Yes	Yes	Yes	Yes
		Range	Yes	Yes	Yes	Yes	Yes
		Aggregation	No	No	No	Yes	Yes
		Pattern Matching*	No	No	No	Yes	Yes
	Data Type Support	String	Yes	Yes	No	Yes	Yes
		Numeric	Yes	Yes	Yes	Yes	Yes
	Overhead Highlights	Computational	High (considerable false hits)	High (number of decryptions)	Low	Low	Low
		Communicational	Reasonable	High (number of communications passes)	Low	Low	Low
	Security		Vulnerable to statistical attacks (almost)	Fairly secure	Vulnerable to statistical attacks	Vulnerable to statistical attacks	Fairly secure
	Update-ability		Moderate	Weak	Good	Good	Moderate

* Only for string data

References

[1] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 216–227, 2002.

[2] D. Agrawal, A. Abbadi, F. Emekci and A. Metwally, "Database management as a service: challenges and opportunities," *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)*, pp. 1709–1716, 2009.

[3] S. Chung, *Anti-Tamper Databases: Querying Encrypted Databases*, Ph.D. Thesis, Case Western Reserve University, Cleveland, Ohio, USA, 2006.

[4] B. Hore, S. Mehrotra and G. Tsudik, "A privacy-preserving index for range queries," *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, pp. 720–731, 2004.

[5] E. Mykletun and G. Tsudik, "Aggregation Queries in the Database-As-a-Service Model," *Data and Applications Security XX, DBSEC'06, LNCS*, vol. 4127, pp. 89–103, 2006.

[6] Y. Tang and J. Yun, "A Method for Reducing False Hits in Querying Encrypted Databases," *Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC-EEE)*, pp. 164–171, 2006.

[7] H. Wang, *Secure Query Answering and Privacy-Preserving Data Publishing*, Ph.D. Thesis, The University of British Columbia, Vancouver, British Columbia, Canada, 2007.

[8] E. Damiani, S. De Capitani di 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 (CCS)*, pp. 93–102, 2003.

[9] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[10] Y. Tang and L. Zhang, "Adaptive Bucket Formation in Encrypted Databases," *Proceedings of the IEEE International Conference on E-Technology, E-Commerce and E-Service (EEE)*, pp. 116–119, 2005.

[11] Y. Tang, J. Yun and Q. Zhou, "A Multi-agent Based Method for Reconstructing Buckets in Encrypted Databases," *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, pp. 546–570, 2006.

[12] N. Ahituv, Y. Lapid, and S. Neumann, "Processing encrypted data," *Communications of the ACM*, vol. 30, no. 9, pp. 777–780, 1987.

[13] E. Brickell and Y. Yacobi, "On privacy homomorphisms," *Advances in Cryptology, Eurocrypt'87, LNCS*, vol. 304, pp. 117–125, 1988.

[14] J. Domingo-Ferrer, "A provably secure additive and multiplicative privacy homomorphism," *Pro-*

ceedings of the 5th International Conference on Information Security (ISC), pp. 471–483, 2002.

[15] J. Domingo-Ferrer, "Privacy homomorphisms for statistical confidentiality," *Quaderns d'Estadística i Investigació Operativa (Qüestiió)*, vol. 20, no. 3, pp. 505–521, 1996.

[16] R. L. Rivest, L. M. Adleman, and M. Dertouzos, "On Data Banks and Privacy Homomorphisms," In *Foundations of Secure Computation,* Academic Press, pp. 169–179, 1978.

[17] H. Hacigümüs, B. Iyer, and S. Mehrotra, "Efficient Execution of Aggregation Queries over Encrypted Relational Databases," *Proceedings of the 9th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp. 125–136, 2004.

[18] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[19] R. Agrawal, J. Kieman, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 563–574, 2004.

[20] F. Mansoori, *A Method for Executing Queries on Encrypted Databases without Firstly Decrypting Them*, M.Sc. Thesis, Sharif University of Technology, Tehran, Tehran, Iran, 2008.

[21] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu, "Two can keep a secret: A distributed architecture for secure database services," *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, pp. 186–192, 2005.

[22] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation and encryption to enforce privacy in data storage," *Proceedings of the 12th European Symposium on Research in Computer Security*, pp. 171–186, 2007.

[23] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragmentation design for efficient query execution over sensitive distributed databases," *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, pp. 32–39, 2009.

[24] P. Samarati, V. Ciriani, and S. Foresti, "Keep a few: outsourcing data while maintaining confidentiality", *Proceedings of the 14th European Conference on Research in Computer Security*, pp. 440–455, 2009.

[25] R. Brinkman, J. M. Doumen, and W. Jonker, "Using secret sharing for searching in encrypted data," *Secure Data Management, VLDB'04, LNCS*, vol. 3178, pp. 18–27, 2004.

[26] G. R. Blakley, "Safeguarding cryptographic keys," *Proceedings of the AFIPS National Computer Conference*, pp. 313–317, 1979.

[27] S. King, *Some Results in Linear Secret Sharing*, Ph.D. Thesis, The University of Wisconsin-Milwaukee, Milwaukee, Wisconsin, USA, 2000.

[28] O. Goldreich, *Secure Multi-Party Computation*, Working Draft, version 1.3, 2001.

Somayeh Soltani received her B.S. Degree in Computer Science from Ferdowsi University, Mashhad, Iran, in 2006 and M.Sc. Degree in computer Science from Islamic Azad University, Science and Research Branch, Tehran, Iran in 2010. Currently, she works as a specialist in Ferdowsi University of Mashhad CERT. Her research interests include database security, digital forensics and ad hoc network security.

Mohammad Ali Hadavi received his B.S. and M.Sc. degrees in software engineering from Ferdowsi University of Mashad, Iran, and Amirkabir University of Tehran, Iran, in 2002 and 2004 respectively. He is now a Ph.D. student at Sharif University of Technology, Tehran, Iran. His research interests include software security, database security, and security aspects of data outsourcing.

Rasool Jalili received his bachelor's degree in Computer Science from Ferdowsi University of Mashhad in 1985, and the master's degree in Computer Engineering from Sharif University of Technology in 1989. He received his Ph.D. in Computer Science from The University of Sydney, Australia, in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, in 1995. He has published more than 130 papers in Computer Security and Pervasive Computing in international journals and conference proceedings. He is now an associate professor, doing research in the areas of access control, vulnerability analysis, and database security in his Data and Network Security Laboratory (DNSL).