AS5: A Secure Searchable Secret Sharing Scheme for Privacy Preserving Database Outsourcing

Mohammad Ali Hadavi[1], Ernesto Damiani[2], Rasool Jalili[1], Stelvio Cimato[2], and Zeinab Ganjei[1]

[1] Data and Network Security Laboratory (DNSL), Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
{mhadavi@ce.,jalili@,zganjei@ce.}sharif.edu

[2] SErvice-oriented Software Architecture Research (SESAR) Lab, Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy
{ernesto.damiani,stelvio.cimato}@unimi.it

Abstract. Researchers have been studying security challenges of database outsourcing for almost a decade. Privacy of outsourced data is one of the main challenges when the "*Database As a Service*" model is adopted in the service oriented trend of the cloud computing paradigm. This is due to the insecurity of the network environment or even the untrustworthiness of the service providers. This paper proposes a method to preserve privacy of outsourced data based on Shamir's secret sharing scheme. We split attribute values into several parts and distribute them among untrusted servers. The problem of using secret sharing in data outsourcing scenario is how to search efficiently within the randomly generated pool of shares. In this paper, at first, we customize Shamir's scheme to have *A Searchable Secret Sharing Scheme* (AS4) that enables the efficient execution of different kinds of queries over distributed shares. Then, we extend our method for sharing values to *A Secure Searchable Secret Sharing Scheme* (AS5) to tolerate statistical attacks based on adversary's knowledge about outsourced data distribution. In AS5 data shares are generated uniformly across a domain to prevent information leakage about the outsourced data.

Keywords: Secure database outsourcing, data confidentiality, secret sharing, query processing.

1 Introduction

Nowadays cloud computing environments provide infrastructure as a service, software as a service, and even database as a service to reduce information technology related burden of organizations businesses. Rapid improvements in the area of network and software technology increase the motivation of companies to outsource their supplementary services, the supporting services of their core business, to third party service providers. Nevertheless, outsourcing data and its management raises security challenges regarding confidentiality of outsourced data due to the insecurity of network environment, or even to the untrustworthiness of service providers. Security challenges of database outsourcing are an obstacle to the success of "*Database As a Service*" model in operation compared to infrastructure or software as a service.

A naïve solution for confidentiality of outsourced data is to encrypt data before its outsourcing [1-5]. In encryption based solutions the service provider does not have the decryption key. It must be able to execute submitted queries without decryption. The vast majority of research on secure data outsourcing has focused on query execution over encrypted outsourced data. Generally, encryption based solutions suffer from key management overheads and either inefficiency or vulnerability to statistical inferences over encrypted values [6, 7].

Departing from encryption, fragmentation is another approach for outsourced data confidentiality. It tries to hide sensitive associations of values defined by a set of confidentiality constraints over attributes/tuples of a relation. The data owner partitions a relation, vertically [8, 9] or horizontally [10, 11], into several parts and outsources them to different servers. In this approach, query execution is generally more efficient compared to encryption based solutions. However, encryption is sometimes unavoidable because fragmentation cannot preserve confidentiality of a single sensitive attribute [12, 13]. The vulnerability to statistical database attacks in the case of collusion between servers, in which different partitions are hosted, as well as servers inferences on data updates are two major disadvantages of fragmentation-based solutions.

Agrawal *et al.* [14] use secret sharing to preserve outsourced data confidentiality. They utilize hash functions to reproduce distribution polynomials and data shares. While their solution efficiently supports different kinds of queries, it is susceptible to statistical inferences in the case that the untrusted servers have *a priori* knowledge of data distribution or frequency. Hadavi *et al.* [15, 16] use secret sharing as well to distribute data among untrusted servers. They use a B^+ index tree on order preserving encrypted values of attributes to be able to search within data shares.

Utilizing the secret sharing concept, in this paper we introduce a solution for confidentiality of outsourced data. We fragment a relation into several parts, neither vertically nor horizontally, by splitting attribute values into randomized shares, and distribute each share to a server. Observing data shares, the servers cannot improve their knowledge about outsourced data, even if they collude. The main problem of sharing attribute values among servers is how to efficiently search within the pool of randomly generated shares. Focusing on this problem, we propose *A Searchable Secret Sharing Scheme* (AS4) for data outsourcing that supports efficient query processing. Then, we extend it to *A Secure Searchable Secret Sharing Scheme* (AS5) to tolerate an intensified threat model in which servers have *a priori* knowledge about data distribution.

This paper is organized as follows. Section 2 contains background information on using secret sharing in data outsourcing scenario, our problem definition, and the threat model. Section 3 introduces AS4 as our basic scheme for efficient search on distributed shares. Then, in Section 4 we introduce a security extension to AS4, namely AS5, to make the secret sharing scheme secure under statistical analysis based on adversary's prior knowledge about data distribution. Section 5 includes query processing scenarios in our proposed approach. In Section 6 we discuss some security issues in AS5. Finally, Section 7 concludes the paper and expresses some directions of future work.

2 Secret Sharing in Outsourcing Scenario

A Secret sharing scheme is a method of sharing a secret s among a set of participants $U = \{u_1, u_2, ..., u_n\}$ such that only authorized subsets of U, called access structure, can reconstruct s. We use threshold (k, n) secret sharing scheme, proposed by Shamir [17], where the access structure is a subset A of 2^U such that $\forall\ B \in A, |B| \geq k$. That is, every k or more participants can reconstruct s while less than k participants cannot.

We follow a database outsourcing model in which an owner outsources his confidential data to honest but curious servers. Then, the owner and authorized users/clients, given required credentials, submit their queries to the servers.

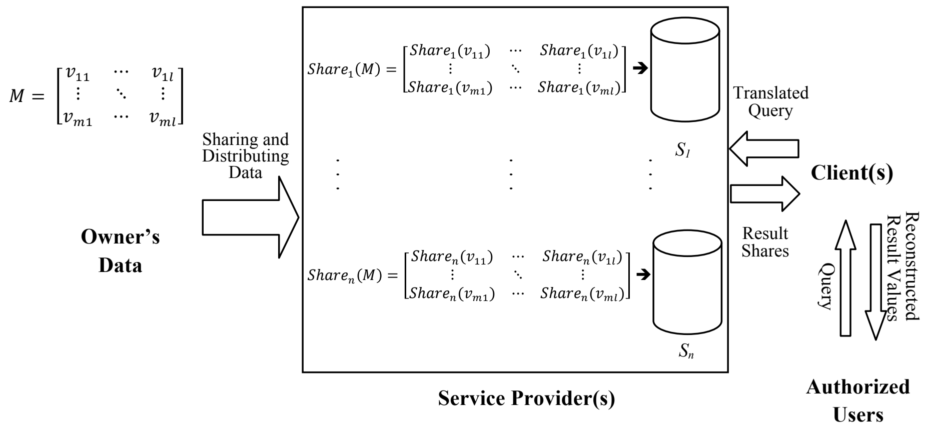
Using threshold (k, n) secret sharing in data outsourcing scenario, the data owner becomes the distributor of shares among n servers, $S_1, S_2,..., S_n$, that are the participants. Each attribute value v is a secret that is split into n shares, $share_i(v)$ $(1 \leq i \leq n)$. To compute the share values of attribute value v, the data owner produces a polynomial of order $k\text{-}1$, $p(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} +...+ a_1x + a_0$, where $a_0 = v$ and other coefficients are chosen randomly from $GF(P)$. Having a secret vector $X\ (x_1, x_2, ..., x_n)$, x_i corresponds to S_i, the owner computes $share_i(v) = p(x_i)$ and stores it on $S_i\ (1 \leq i \leq n)$. In fact, for each attribute value, there are n points $(x_i, p(x_i))$ through which the polynomial $p(x)$ passes. k distinct points are enough to uniquely reconstruct a polynomial of order $k - 1$. Therefore, when a trusted party, who knows the distribution vector X, receives at least k shares of the secret v, the secret can be reconstructed. On the other hand, the servers infer nothing about v even if they collude and pool their shares because they do not have the distribution vector X.

Let us model a database table with m rows and l columns as a matrix $M_m\text{x}_l$. Fig. 1 shows outsourcing a relation, as an $m\text{x}l$ matrix M to n servers, each of them is given an mxl matrix $Share_i(M)$, where v_{xy} as a cell of M, is mapped onto $share_i(v_{xy})$ of matrix $Share_i(M)$ $(1 \leq i \leq n)$.

2.1 Problem Definition

The theoretic security of Shamir's scheme guarantees outsourced data confidentiality, keeping in mind that the distribution vector X is hidden from the untrusted servers. This solution provides the highest level of data confidentiality because the data shares are produced using random coefficients and do not leak any information about the distribution or the frequency of original values. This is caused by generating random shares for values which results in different polynomials and consequently different shares for two equal values.

In outsourcing scenario, authorized users are supposed to query outsourced data and request the servers for retrieving the appropriate shares. While the owner or authorized users do not store the random coefficients, data retrieval becomes a problem. Obviously it is not efficient to send back all shares in response to a query, and execute the query over the reconstructed values at client side. For this purpose, a method is required to identify the shares, stored on the servers, that satisfy the query condition.



$M = \begin{bmatrix} v_{11} & \cdots & v_{1l} \\ \vdots & \ddots & \vdots \\ v_{m1} & \cdots & v_{ml} \end{bmatrix}$
Owner's Data
Sharing and Distributing Data
$Share_1(M) = \begin{bmatrix} Share_1(v_{11}) & \cdots & Share_1(v_{1l}) \\ \vdots & \ddots & \vdots \\ Share_1(v_{m1}) & \cdots & Share_1(v_{ml}) \end{bmatrix}$
S_1
$Share_n(M) = \begin{bmatrix} Share_n(v_{11}) & \cdots & Share_n(v_{1l}) \\ \vdots & \ddots & \vdots \\ Share_n(v_{m1}) & \cdots & Share_n(v_{ml}) \end{bmatrix}$
S_n
Service Provider(s)
Translated Query
Result Shares
Client(s)
Query
Reconstructed Result Values
Authorized Users

Fig. 1. Sharing a database relation, as a matrix, among *n* servers

To solve the above problem we propose a solution in which searching a subset of shares is possible, while the confidentiality of outsourced data is preserved. At first, we redefine Shamir's secret sharing scheme to preserve the order of values in their corresponding shares and at the same time to perturbate the distribution of values in their shares. The obtained scheme, AS4, is still vulnerable to statistical analysis since the order of attribute values is preserved in their shares. We extend AS4 in terms of security to AS5, a solution that tolerates statistical analysis on data shares based on adversary's prior knowledge about outsourced data. We also examine query processing support for different kinds of queries in our approach and discuss about security achievement considering possible attack scenarios on data confidentiality.

2.2 Threat Model

We have the following assumptions for our threat model:

- Service providers are honest but curious. They execute submitted queries honestly on outsourced data and send complete and authenticated results. They are curious and try to increase their knowledge about confidential data by observing data shares and submitted queries
- Service providers have *a priori* knowledge about outsourced data. For example, they might know the domain of values, the minimum or maximum values, or some information about the frequency of values.
- The servers can communicate and collude with each other to extract knowledge about outsourced data.
- An adversary who has compromised the servers can observe data shares and submitted queries as well as the service providers.

Also we assume that clients, as users' machines that mediate user requests to service providers, are trusted. Moreover, users are given appropriate authorities and credentials to submit a query through a client and see the result.

3 A Searchable Secret Sharing Scheme (AS4)

To have a searchable secret sharing scheme with threshold (k, n) we define a scheme, called AS4, in which the order of attribute values is preserved in their corresponding shares but not the frequency of values. This scheme is required to satisfy two following properties:

1. Order preserving: Let V be the domain of attribute values. The order of values must be preserved in their corresponding shares. That is:
$$\forall\ v, v' \in V: \quad if \ \ v < v' \ \Rightarrow share_i(v) < share_i(v') \quad (1 \leq i \leq n)$$
2. Distribution perturbation: The original data distribution must be perturbated by changing the frequency of attribute values in their corresponding shares. That is, two equal values have different shares with a high probability:
$\forall\ v, v' \in V$ (v and v' are attribute values in different database tuples):
$$\text{if} \quad v = v' \Rightarrow \Pr[share_i(v) = share_i(v')] < \varepsilon \quad (1 \leq i \leq n)$$

The first property is aimed at making query execution efficient. The second property indicates that the sharing is not deterministic. It is aimed at concealing the frequency of original values. This property improves the robustness against statistical analysis. In the above formula, ε is a small value, dependent on the domain size of shares.

Redefining the Shamir's threshold scheme, now we introduce AS4 as "a searchable secret sharing scheme" whose shares, while preserving the ordering of original values, substantially change the original data distribution.

Choosing values of k and n for a threshold secret sharing (k, n) has some influences on the availability and fault tolerance aspects of the system that are not the focus of this work. In our system, choosing large k increases the communication cost while it does not offer more security in terms of confidentiality. Therefore, in the remaining parts of this paper we assume a threshold ($2, 3$) secret sharing scheme with the polynomial of the general form $p(x) = ax + v$ where "a" is a random coefficient of a specified domain. The domain of a searchable attribute in a relation is a set of values $V = \{v_1, v_2, \ldots, v_t\}$ where $v_1 < v_2 < \ldots < v_t$.

To obtain AS4 we follow two steps: first, we partition the coefficient domain, and second, we assign each attribute value to a partition from which the coefficient a is randomly chosen. Let us have a detailed view for these two steps.

1. **Partitioning the domain of the coefficient:** The first step is to partition the coefficient's domain D (of real numbers) considering the following definition for partitioning.

   Definition- *Partitioning* a domain D of values is defined as dividing D into t parts d_i ($1 \leq i \leq t$) where

   1. $d_i \subseteq D$ ($1 \leq i \leq t$) is a range of values in D
   2. $\bigcup_{i=1}^{t} d_i = D$
   3. $\forall\ d_i, d_j \subseteq D\ (i \neq j): \quad d_i \cap d_j = \emptyset$. ■

   A possible way of partitioning a domain D is to divide it into a sequence of an arbitrary number of equal partitions. With D as values in the range $[d_s .. d_e]$ being the coefficient's domain, we divide it into $t = |V|$ equal consecutive partitions including $[d_s\ ..\ d_s+\frac{|D|}{t})$, $[d_s+\frac{|D|}{t}\ ..\ d_s+2\frac{|D|}{t})$, $[d_s+2\frac{|D|}{t}\ ..\ d_s+3\frac{|D|}{t})$, …, and $[d_s+(t-1)\frac{|D|}{t}\ ..\ d_e]$.

2. **Value-Partition Assignment.** The next step is to define a function to map a value $v \in V$ onto a partition $d \subseteq D$.

   Definition- *Mapping function F* is a function that maps a value $v \in V$ onto a partition $d \subseteq D$ where

   $$\forall\ v, v' \in V,\ F(v) = d,\ F(v') = d'\ (d, d' \subseteq D) : v \neq v' \Rightarrow d \cap d' = \emptyset$$

   ■

   For AS4, we define F as a total order function such that maps $v \in V$ onto $d = [d_s+(v-v_1)\frac{|D|}{t}\ ..\ d_s+(v-v_1+1)\frac{|D|}{t})$. For sharing attribute value v among three servers, the owner choose a randomly from the above range and construct the polynomial $p(x_i) = ax_i + v$ to compute the data share of S_i $(1 \leq i \leq 3)$.

Fig. 2 exemplifies sharing Age values (integer values of the range [1..100]) of Employee relation (Table 1) among three servers with a sample distribution vector $X = \{x_1=9,\ x_2=14,\ x_3=2\}$ and integer random coefficients of the range [1..1000]. For simplicity, in this example we only show sharing Age values. All searchable attributes should be distributed in an order preserving manner to be efficiently searchable in the future.

For query processing, when a user submits her query via a client, the client translates the query into a query over data shares and sends it to the servers. In our example of Employee relation (Table 1) and sharing Age values (Fig. 2), the query "`SELECT Salary FROM Employee WHERE Age = 45`" is translated into "`SELECT` *`Share`*$_1$`(Salary) FROM Employee WHERE (9*441 + 45) ≤` *`Share`*$_1$`(Age) ≤ (9*450 + 45)`" and sent to S_1. The client does similar translations for S_2 and S_3 with respect to the distribution vector X and the mapping function F. It is worth mentioning that if the mapping function F is defined in such a way, the owner or users/clients do not need to store F to find the association of values to partitions for query translation. In Section 5 we elaborate more on the query processing scenario for different kinds of queries.

Now we verify the satisfaction of our two desired properties, i.e., order preserving and distribution perturbation, in two following lemmas.

Table 1. The Employee relation

ID	Age	Salary
1	45	100
2	84	200
3	78	150
4	46	350
5	45	200
6	80	210
7	45	175
8	57	200

Share$_1$(Age)
$9a_{45} + 45$ $441 \le a_{45} \le 450$
$9a_{84} + 84$ $831 \le a_{84} \le 840$
$9a_{78} + 78$ $771 \le a_{78} \le 780$
$9a_{46} + 46$ $451 \le a_{46} \le 460$
$9a_{45} + 45$ $441 \le a_{45} \le 450$
$9a_{80} + 80$ $791 \le a_{80} \le 800$
$9a_{45} + 45$ $441 \le a_{45} \le 450$
$9a_{57} + 57$ $561 \le a_{57} \le 570$

S_1

Share$_2$(Age)
$14a_{45} + 45$ $441 \le a_{45} \le 450$
$14a_{84} + 84$ $831 \le a_{84} \le 840$
$14a_{78} + 78$ $771 \le a_{78} \le 780$
$14a_{46} + 46$ $451 \le a_{46} \le 460$
$14a_{45} + 45$ $441 \le a_{45} \le 450$
$14a_{80} + 80$ $791 \le a_{80} \le 800$
$14a_{45} + 45$ $441 \le a_{45} \le 450$
$14a_{57} + 57$ $561 \le a_{57} \le 570$

S_2

Share$_3$(Age)
$2a_{45} + 45$ $441 \le a_{45} \le 450$
$2a_{84} + 84$ $831 \le a_{84} \le 840$
$2a_{78} + 78$ $771 \le a_{78} \le 780$
$2a_{46} + 46$ $451 \le a_{46} \le 460$
$2a_{45} + 45$ $441 \le a_{45} \le 450$
$2a_{80} + 80$ $791 \le a_{80} \le 800$
$2a_{45} + 45$ $441 \le a_{45} \le 450$
$2a_{57} + 57$ $561 \le a_{57} \le 570$

S_3

Fig. 2. Sharing Age values of Employee relation (Table 1) among three servers

Lemma 1- *AS4 preserves the order of attribute values in their corresponding shares.*

Proof: $\forall\, v, v' \in V: v < v' \Rightarrow$

$$a_v \in [d_s + (v - v_1)\frac{|D|}{t} \,..\, d_s + (v - v_1 + 1)\frac{|D|}{t}) < a_{v'} \in [d_s + (v' - v_1)\frac{|D|}{t} \,..\, d_s + (v' - v_1 + 1)\frac{|D|}{t}) \Rightarrow$$

$$a_v x_i + v < a_{v'} x_i + v' \quad \Rightarrow \quad share_i(v) < share_i(v') \qquad (1 \le i \le 3) \qquad \Box$$

Lemma 2– *AS4 perturbates the distribution of attribute values in their corresponding shares.*

Proof: $\forall\, v, v' \in V$, $v = v'$ and v, v' are in different database tuples:

$$\Pr[share_i(v) = share_i(v')] = \Pr[a_v x_i + v = a_{v'} x_i + v'] \quad (1 \le i \le 3) \quad \Rightarrow$$

$$\Pr[share_i(v) = share_i(v')] = \Pr[a_v = a_{v'}]$$

Considering that a_v and $a_{v'}$ are random numbers, independently chosen from the corresponding partition of v (or v'), $\Pr[a_v = a_{v'}] < \varepsilon$ where $\varepsilon < 1$ and its value is affected by the size of the partition. $\Box$

In the next Section, we extend our scheme in terms of security so that data shares reveal neither the order nor the frequency of outsourced data.

4 A Secure Searchable Secret Sharing Scheme (AS5)

Statistical inferences in our basic scheme (AS4) are possible since the order of attribute values is preserved in their shares. That is, observing data shares, the untrusted servers may statistically deduce some information about original values. Consider an outsourced relation of employees of an organization and assume that the adversary knows that the minimum and maximum ages of employment are 25 and 75, respectively. So, the maximum share value of Age for each server may correspond to the maximum possible Age value, which is 75 in our example. These kinds of information help the adversary to make inferences about original values that may result in revealing the mapping function or even the secret distribution vector X. Such inferences are originated by the two following characteristics in AS4:

1. having equal-length partitions of the coefficient's domain and
2. the definition of mapping function F in a way that preserves the order of values in their corresponding shares.

A generic countermeasure solution is to introduce a weighted partitioning as well as an order-obfuscated mapping function to break the two above characteristics of AS4. Therefore, the first line of AS4 extension focuses on the partitioning method of the coefficient domain. It is intuitively acceptable that when data shares are uniformly distributed across their domain they reveal as least information as possible for an adversary aimed at making inferences on outsourced data. So, the first goal in AS5 is to generate data shares, which have been distributed uniformly across their domain, while query processing is still efficient.

The second line of AS4 extension is related to the mapping function F (by which attribute values are mapped onto partitions) and tries to hide the ordering relation of values in their corresponding shares. Therefore, the goal is to define a mapping function so that the ordering relation between values is obfuscated in their corresponding shares.

Given an original data distribution for a searchable attribute, the owner follows four following steps to generate uniformly distributed - order obfuscated data shares:

1. partitioning the domain of shares
2. obfuscating the ordering relation between values and their corresponding shares
3. partitioning the domain of random coefficient in the distribution polynomial
4. sharing attribute values

Let us have a more detailed view to these steps.

1. **Partitioning the domain of shares**: The owner calculates the length of each partition based on the given data distribution. Let us model the data distribution (for a searchable attribute value) by pairs (v_i, f_i) where $v_i \in V$ is an attribute value and f_i is v_i's frequency in the relation. It is clear that $|V|$ is the required number of partitions and $N_{tuple} = \sum_{i=1}^{|V|} f_i$ is the number of database tuples. Each $v \in V$ is associated with $d \subseteq D$ where the length of d, denoted by $|d|$, is calculated by $|d| = \frac{|D|f_i}{N_{tuple}}$. We have a weighted partitioning based on the values' frequencies in the relation.
2. **Defining an order-obfuscated mapping function**: the owner defines the mapping function F such that the ordering relation between attribute values is not preserved in their corresponding partitions. To this end, a random permutation of d_is ($1 \le i \le |V|$), among $|V|!$ possible permutations, is chosen by the owner. Fig. 3 illustrates partitioning a domain of shares based on values' frequencies and an order preserving (a) and an order-obfuscated (b) mapping function.

3. **Partitioning the domain of random coefficient**: Considering that the owner uses $Share_i(v) = ax_i + v$ to compute S_i's share of v, the next step is to partition the domain of coefficient a such that it generates a set of shares which have been distributed uniformly across their domain. It is simply done by putting minimum and maximum values of $Share_i(v)$ into $a = \frac{Share_i(v) - v}{x_i}$ to find the range boundaries of a partition in the coefficient domain corresponding to v.
4. **Sharing attribute values**: For the final step, the data owner uses the calculated ranges of the coefficient domain to compute data shares of each value. He chooses a randomly from the specified range (previous step) and put it into the equation $Share_i(v) = ax_i + v$. Then, the computed share $Share_i(v)$ is outsourced to S_i.

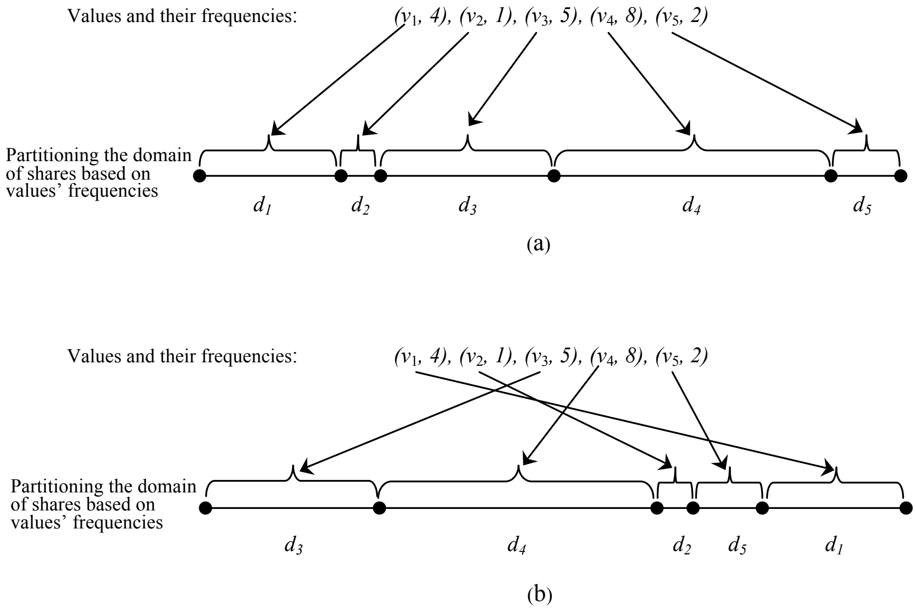

Values and their frequencies:
(v1, 4), (v2, 1), (v3, 5), (v4, 8), (v5, 2)
Partitioning the domain of shares based on values' frequencies
d1
d2
d3
d4
d5
(a)
Values and their frequencies:
(v1, 4), (v2, 1), (v3, 5), (v4, 8), (v5, 2)
Partitioning the domain of shares based on values' frequencies
d3
d4
d2
d5
d1
(b)

Fig. 3. An order preserving partition assignment (a), and an obfuscated partition assignment (b) of an attribute with a domain of five values.

Obviously, the mapping function F must be kept hidden from an adversary who may know some information about original data distribution. The owner informs authorized users/clients about the mapping function F for further use in query translation.

Compared to AS4, the extra imposed cost in AS5 is to store F at owner and client sides for query translation. To store F, the owner requires the storage cost of the order O(|V|). This is reasonable for a database size of order O(|N|) where N is the number of database tuples. For example, for the attribute Age with the domain size |V| = 100 and 4-Byte share values, the owner needs eight bytes to specify a partition. While the size of a database with one million 64-Byte tuples is almost 64MB, less than 1KB (800 Bytes) is enough to store the mapping function at owner and client sides.

5 Query Processing

In this section we examine the query execution scenario of typical equality, range, projection, join, aggregation, and update queries. The query processing scenarios in AS4 and AS5 are almost similar considering that in AS5 case, clients have the mapping function F to translate submitted queries into server side queries over shares. The only difference is that the execution of range queries in AS5 is less convenient than in AS4 because in AS5 the ordering relation is obfuscated in data shares.

Equality Queries. The condition "$att = v$", where *att* is a searchable attribute and v is a value, is translated by a client into "$min(Share_i(v)) \leq att \leq max(Share_i(v))$" and sent to S_i ($1 \leq i \leq 3$). The minimum and maximum values of the query condition predicate are obtained by following equations

$$min(Share_i(v)) = (min(a)).x_i + v \quad , \\ max(Share_i(v)) = (max(a)).x_i + v$$

where $min(a)$ and $max(a)$ are computed according to the mapping function F, which is known for authorized users/clients.

Consider a simple equality query "`SELECT * FROM Employee WHERE Age = 20`". The client translates the query into a range query of the form "`SELECT * FROM Employee WHERE` $min(Share_i(20)) \leq Age \leq max(Share_i(20))$", and submits it to S_i ($1 \leq i \leq 3$). Receiving shares of satisfying tuples from two servers, the client interpolate original values.

This method guarantees that all tuples with `Age = 20` are returned exclusively as query results. Let $Res_i = \{Share_i(v) \mid min(Share_i(v)) \leq Share_i(v) \leq max(Share_i(v))\}$ be a set of satisfying shares in S_i ($1 \leq i \leq 3$) based on a query condition. The execution of equality queries in our scheme generates a *sound* and *complete* result set.

Lemma 3- *The returned result of an equality query is sound.*

Proof: The result set is not sound if there is a $share_i(v') \in Res_i$ where $v' \neq v$. According to the threat model that assumes honesty of servers, they send back all appropriate shares based on the selection predicate of a submitted query, whose condition predicate has been translated.

According to the definition of partitioning which states that partitions are disjoint and with respect to our mapping function in which each value v is mapped to a different partition we have:

$\forall v, v' \in V, v' \neq v$:

$[min(Share_i(v)) .. max(Share_i(v))] \cap [min(Share_i(v')) .. max(Share_i(v'))] = \emptyset \Rightarrow$

$Share_i(v') \notin [min(Share_i(v)) .. max(Share_i(v))] \Rightarrow share_i(v') \notin Res_i \qquad (1 \leq i \leq 3)$ □

The Soundness of query result means that there is not any false hit in returned results of the servers, bringing less client side computation (for result pruning) as well as less communication cost.

Lemma 4- T*he returned result of an equality query is complete.*

Proof: The result set is complete if $\forall\ v' \in V,\ v' = v:\quad Share_i(v') \in Res_i\ (1 \leq i \leq 3)$.
Suppose that there is a $v' = v$ for which $Share_i(v') \notin Res_i$. Then we have:

$$\exists\ v, v' \in V, v = v', Share_i(v) \in Res_i\,, Share_i(v') \notin Res_i$$

$$Share_i(v') \notin Res_i \quad \Rightarrow \quad Share_i(v') \notin [min(Share_i(v)) .. max(Share_i(v))] \xrightarrow{(v=v')} Share_i(v') \notin [min(Share_i(v')) .. max(Share_i(v'))]$$

which is a contradiction with respect to our secret sharing scheme. □

Simply it is possible to have conjunctive or disjunctive conditions on searchable attributes. In such a case, the servers return a set of shares so that the whole condition, as the composition of conditions, is satisfied.

Range Queries. In AS4, range queries are processed similar to equality queries except the change in the range boundaries. That is, for the lower bound of the range the minimum possible value and for the upper bound of the range the maximum possible value are used. Consider a simple range query "`SELECT * FROM Employee WHERE 50 ≤ Age ≤ 80`". The submitted query to S_i $(1 \leq i \leq 3)$ is: "`SELECT * FROM Employee WHERE` *min*(*Share$_i$*(50)) ≤ `Age` ≤ *max*(*Share$_i$*(80))".

In AS5 the translation of range queries is not as straightforward as in AS4 caused by the order-obfuscated mapping function in AS5. In AS5, a range of attribute values is usually mapped onto several ranges of shares. Consider the sample range query "`SELECT Salary FROM Employee WHERE 50 < Age < 55`". The query is translated into the following query and sent to S_i $(1 \leq i \leq 3)$:

```
"SELECT Salary FROM Employee WHERE
                    min(Sharei(51)) ≤ Age ≤ max(Sharei(51)) OR
                    min(Sharei(52)) ≤ Age ≤ max(Sharei(52)) OR
                    min(Sharei(53)) ≤ Age ≤ max(Sharei(53)) OR
                    min(Sharei(54)) ≤ Age ≤ max(Sharei(54))"
```

Range queries generate a complete set of results without any false hits as well as equality queries.

Lemma 5- T*he returned result of a range query is sound and complete.*
Proof: We can consider the result set of a range query as the union of the result sets of a sequence of equality queries. Each equality query, for which we have a sound and complete result set, is executed independently over a partition. Consequently, the result set of a range query is sound and complete regarding that each value is assigned a distinct partition and partitions are disjoint. □

Projection Queries. Projection is supported in AS5 as well as AS4 without extra overheads, since the granularity of sharing is attribute value. For a query such as "`SELECT Salary FROM Employee WHERE 20 ≤ Age ≤ 25`", each S_i $(1 \leq i \leq 3)$ finds satisfying share values of Age and then, returns the corresponding Salary shares to the client. The client can interpolate original Salary values after receiving Salary shares of two servers.

Aggregate Queries. Queries with SUM aggregation function are supported in both AS4 and AS5 thanks to the additive homomorphism property of the secret sharing scheme. Consider a query with SUM aggregation function such as "`SELECT SUM(Salary) FROM Employee WHERE 20 ≤ Age ≤ 25`". The client translates it and sends the query "`SELECT Salary FROM Employee WHERE` $min(Share_i(20)) \le$ `Age` $\le max(Share_i(25))$" to S_i $(1 \le i \le 3)$. Then, S_i locally calculates the summation of satisfying Salary values. Thanks to the additive homomorphism property of Shamir's scheme, the client can compute the total summation value when it receives two values from the servers.

Queries containing CONUT function such as "`SELECT COUNT(Salary) FROM Employee WHERE Age = 20`" are executed simply by reforming as "`SELECT COUNT(Salary) FROM Employee WHERE` $min(Share_i(20)) \le$ `Age` $\le max(Share_i(20))$" for S_i $(1 \le i \le 3)$. Since the servers are honest but curious, COUNT queries of the above form can be sent to only one S_i instead of sending to all, incurring less communication overhead.

Executing MIN/MAX queries in both AS4 and AS5 is not as straightforward as COUNT and SUM queries and may need several rounds of client mediation to have the final result. Consider a sample MIN/MAX query such as "`SELECT Salary FROM Employee WHERE Age = MIN(Age)`". This query is translated into "`SELECT` $Share_i$`(Salary) FROM Employee WHERE` $min(Share_i(v_1)) \le$ `Age` $\le max(Share_i(v_1))$" and sent to S_i $(1 \le i \le 3)$. v_1 is the minimum value of the Age domain. If the query returns no shares, the client continues with the next possible minimum, e.g. v_2, to reach the result finally. In our example, the client continues to query the servers until it sends "`SELECT` $Share_i$`(Salary) FROM Employee WHERE` $min(Share_i(45)) \le$ `Age` $\le max(Share_i(45))$" to S_i $(1 \le i \le 3)$ and receives the result shares.

Some mechanisms such as client side storage of minimum and maximum values of a searchable attribute or using an auxiliary table to maintain the ordering of values [16] can also be used to tackle the problem of MIN/MAX queries in our approach.

For more complex queries with aggregation functions in their selection predicates such as "`SELECT MIN(Salary) FROM Employee where Age = 50`" or "`SELECT MAX(Salary) FROM Employee WHERE Age = MIN(Age)`" the final result is computed at client side after receiving the satisfying values from at least two servers.

Join Queries. Join queries are performed on two tables with an attribute in common. Consider two simple relations *T_1(ID, Dep, Salary)* and *T_2(ID, Name, Age)* and the sample join query "`SELECT Salary FROM T1, T2 WHERE T1.ID = T2.ID`". This query is rewritten as a parameterized query "`SELECT Salary FROM` T_1, T_2 `WHERE` $(min(\text{share}(v_i)) \le T_1.\text{ID} \le max(\text{share}(v_i)))$ `AND` $(min(\text{share}(v_i)) \le T_2.\text{ID} \le max(\text{share}(v_i)))$" for $(1 \le i \le t)$ where t is the maximum value of the attribute domain. To have a complete result, the client submits $|V|$ queries to the servers and performs a union operation on the received results.

Update Queries. While some existing solutions suffer from frequent updates in dynamic environment, our approach supports update queries. The execution of update queries including INSERT, DELETE, and UPDATE is straightforward in AS4. A typical deletion query such as "`DELETE FROM Employee WHERE Age = 80`" is translated into "`DELETE FROM Employee WHERE` $min(Share_i(80)) \le$ `Age` $\le max(Share_i(80))$" and sent to $S_i (1 \le i \le 3)$. Since equality and range queries return a complete result set without false hits, S_i removes exclusively a complete set of satisfying tuples from the target relation. To insert a new tuple, shares of attribute values in the tuple are computed and inserted into S_i $(1 \le i \le 3)$. For an update query, satisfying tuples based on the condition predicate of the query are selected and sent to the owner. The owner constructs a new polynomial and computes the values in SET predicate of the update query to substitute new shares with the old ones.

One can say that database update is a challenge in AS5. Because update queries change the distribution of a searchable attribute and consequently, the uniform distribution of shares is disorganized. In such a case it is required to repartition, redefine the mapping function, and redistribute data which is considerably costly for outsourcing scenario. To let AS5 be adopted in dynamic environments, a practical suggestion is to use standard distributions of attribute values instead of real distributions. Using such standard distributions, data updates do not deteriorate the distribution of attribute values when sharing the values of a new tuple. In other words, we will have almost the same distribution in different snapshots of the system. From a practical point of view, it may be acceptable to refer to an expected distribution of Age values stored in an organizational database. We plan to quantify the possible performance degradation of AS5 due to using standard data distribution instead of real ones.

6 Security Discussion

In this section we discuss about the security of our approach, focusing on AS5 as it provides more security guarantees. As an obvious assumption, the distribution vector X must be kept hidden from the untrusted servers. Therefore, even if the servers collude with each other and pool their shares, they cannot reconstruct the original values. Attribute values can be reconstructed by the trusted parties who know the distribution vector X. Moreover, the mapping function F in AS5 must be kept secret from the untrusted parties to prevent them from inferring the ordering relation between values.

Arithmetic calculations for sharing and reconstructing secrets in Shamir's scheme in defined over $GF(P)$ where P is a prime number. In AS5, Instead of modular arithmetic calculations, we uniformly distribute share values across a domain using random coefficients that belong to partitions of a specified domain. Having a uniform distribution of shares minimizes the information leakage for the adversary who observes the outsourced data shares. The imposed overhead is to have the share size bigger than the secret size. Theoretically, we need the domain D of share values with the size of $|D| \ge N_{tuple}$ to uniformly distribute N_{tuple} distinct shares across D. N_{tuple} is the maximum number of tuples in the relation (total frequency of values). In practice, having 4-Byte shares provides the possibility of more than four billion distinct values for shares which is enough for many applications. If the server side storage is an important concern, choosing an appropriate domain size for data shares is a trade-off between security and storage cost of the scheme. We plan to investigate more on the size of domain to have a minimum storage cost in addition to achieve desired level of security.

Although the original data distribution is perturbated in AS4, it is vulnerable to statistical analysis as it preserves the ordering relation of attribute values in their corresponding shares. We proposed AS5 as a security extension to AS4 where we change any given distribution of original values to a uniform distribution of shares across a domain. From the information theory viewpoint, uniform distribution has the highest entropy and reveals at least information as possible for the untrusted servers. However, there is an inference possibility in AS5 for the untrusted servers while executing queries. They are able to make inferences about frequency of attribute values using the interval width of satisfying shares. In other words, a group of shares for which the query condition is satisfied, form a partition in the domain of shares that can be mapped onto an attribute value. The untrusted servers can gradually find out the partitions and use their knowledge about outsourced data to find the hidden mapping function. To prevent this kind of inference, the client must frequently submit fake queries without taking care of their results. These queries target invalid intervals of shares (with respect to the mapping function) to mislead the servers of making valid inferences about associating requested partitions of shares to original values. Submitting fake queries together with the order obfuscated mapping function is a countermeasure against share alignment, which is an attack described in [18] as a security limitation of using secret sharing for outsourcing scenario.

There is another attack scenario considering the knowledge of system's query workload. While AS5 defends the confidentiality of outsourced data against adversaries powered by *a priori* knowledge of data distribution, it is susceptible to statistical inferences if we assume that adversaries are powered by *a priori* knowledge of system's query workload. Although the aforementioned fake queries can be considered as a countermeasure for this attack, it requires more investigation on the way of generating queries and their submission frequencies. To have a secure solution under the assumption of adversaries with prior knowledge about submitted queries, we should extend our approach to a solution with private information retrieval in which access confidentiality is preserved.

7 Conclusion

In this paper we proposed an approach for outsourcing confidential data to honest but curious servers. We used the idea of secret sharing to split a database relation into several parts, in the granularity of attribute values, and outsource each part to an honest but curious server.

We proposed a searchable secret sharing scheme in which the ordering relation between values is preserved in their corresponding shares, while the distribution of shares is different from the original data distribution. However, the order preserving property makes the scheme vulnerable to statistical analysis. We extended our searchable secret sharing scheme to be secure against adversaries powered by *a priori* knowledge of outsourced data to tolerate statistical analysis on data shares.

Our approach is promising in terms of provided security, as it uses secret sharing with strong theoretic background, and supporting different kinds of queries. Less computation time of secret sharing and reconstruction compared to encryption and decryption operations in encryption-based approaches, the additive homomorphism property of Shamir secret sharing scheme, and not to having false hits in returned results of the servers are of the main reasons for its efficient query processing support. Nevertheless, we should extend our solution to support character data considering pattern matching queries over string attributes.

While our approach targets data confidentiality with honest but curious servers, dealing with the correctness of query results is another issue for active adversaries who can manipulate outsourced data shares. We plan to investigate on share integrity and query result correctness using redundant shares in our threshold scheme for data outsourcing scenario.

References

[1] Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over Encrypted Data in the Database-Service-Provider Model. In: ACM SIGMOD International Conference on Management of Data, SIGMOD 2002, New York, USA, pp. 216–227 (2002)

[2] Agrawal, R., Kiernan, G.G.: System and method for fast querying of encrypted databases. US Patent 7,395,437: Google Patents (2008)

[3] Canim, M., Kantarcioglu, M.: Design and analysis of querying encrypted data in relational databases. In: The 21st Annual IFIP WG 11.3 working Conference on Data and Applications Security, pp. 177–194 (2007)

[4] Zhang, Y., Li, W.-X., Niu, X.-M.: Secure cipher index over encrypted character data in database. In: 2008 International Conference on Machine Learning and Cybernetics, pp. 1111–1116 (2008)

[5] Zhu, H., Cheng, J., Jin, R., Lu, K.: Executing Query over Encrypted Character Strings in Databases. In: 2007 Japan-China Joint Workshop on Frontier of Computer Science and Technology (FCST 2007), pp. 90–97 (2007)

[6] Damiani, E., Vimercati, S.D.C., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In: Proceedings of the 10th ACM Conference on Computer and Communication Security, CCS 2003, New York, USA, pp. 93–102 (2003)

[7] Li, J., Omiecinski, E.R.: Efficiency and security trade-off in supporting range queries on encrypted databases. In: 19th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, pp. 69–83 (2005)

[8] Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two Can Keep a Secret: A Distributed Architecture for Secure Database Services. In: 2nd Biennial Conference on Innovative Data Systems Research (2005)

[9] Samarati, P., Ciriani, V., Foresti, S.: Keep a Few: Outsourcing Data While Maintaining Confidentiality. In: 14th European Conference on Research in Computer Security, pp. 440–455 (2009)

[10] Wiese, L.: Horizontal Fragmentation for Data Outsourcing with Formula-Based Confidentiality Constraints. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) IWSEC 2010. LNCS, vol. 6434, pp. 101–116. Springer, Heidelberg (2010)

[11] Soodejani, A.T., Hadavi, M.A., Jalili, R.: *k*-Anonymity-Based Horizontal Fragmentation to Preserve Privacy in Data Outsourcing. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 263–273. Springer, Heidelberg (2012)
[12] Ciriani, V., Vimercati, S.D., Foresti, S., Jajodia, S.: Combining Fragmentation and Encryption to Protect Privacy in Data Storage. ACM Transactions on Information and System Security (TISSEC) 13, 1094–9224 (2010)
[13] Ciriani, V., Vimercati, S.D.C.D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and Encryption to Enforce Privacy in Data Storage. In: 12th European Symposium on Research in Computer Security, pp. 171–186 (2007)
[14] Agrawal, D., Abbadi, A.E., Emekci, F., Metwally, A.: Database Management as a Service: Challenges and Opportunities. In: 2009 IEEE 25th International Conference on Data Engineering, pp. 1709–1716 (2009)
[15] Hadavi, M.A., Jalili, R.: Secure Data Outsourcing Based on Threshold Secret Sharing: Towards a More Practical Solution. In: VLDB 2010 PhD Workshop, Singapore, pp. 54–59 (2010)
[16] Hadavi, M.A., Noferesti, M., Jalili, R., Damiani, E.: Database as a Service: Towards a Unified Solution for Security Requirements. In: 36th International Conference on Computer Software and Applications, The 4th IEEE International Workshop on Security Aspects in Processes and Services Engineering, Izmir, Turkey, pp. 415–420 (2012)
[17] Shamir, A.: How to Share a Secret. Communications of the ACM 22, 612–613 (1979)
[18] Dautrich, J.L., Ravishankar, C.V.: Security Limitations of Using Secret Sharing for Data Outsourcing. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 145–160. Springer, Heidelberg (2012)