Int. J. Inf. Secur. (2015) 14:513–529
DOI 10.1007/s10207-015-0277-x

REGULAR CONTRIBUTION

Security and searchability in secret sharing-based data outsourcing

Mohammad Ali Hadavi · Rasool Jalili · Ernesto Damiani · Stelvio Cimato

Published online: 21 February 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract A major challenge organizations face when hosting or moving their data to the Cloud is how to support complex queries over outsourced data while preserving their confidentiality. In principle, encryption-based systems can support querying encrypted data, but their high complexity has severely limited their practical use. In this paper, we propose an efficient yet secure secret sharing-based approach for outsourcing relational data to honest-but-curious data servers. The problem with using secret sharing in a data outsourcing scenario is how to efficiently search within randomly generated shares. We present multiple partitioning methods that enable clients to efficiently search among shared secrets while preventing inference attacks on the part of data servers, even if they can observe shares and queries. Also, we prove that with some of our partitioning methods the probability of finding a correspondence between a set of shares and their original values is almost equal to that of a random guess. We discuss query processing for different types of queries including equality, range, aggregation, projection, join, and update queries. Our extensive experimentation confirms the practicality and efficiency of our approach in terms of query execution time, storage, and communication overheads.

Keywords Database outsourcing · Data confidentiality · Searchable secret sharing · Partitioning · Query processing

M. A. Hadavi · R. Jalili (✉)
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
e-mail: jalili@sharif.edu

M. A. Hadavi
e-mail: mhadavi@ce.sharif.edu

E. Damiani · S. Cimato
Department of Computer Science, Università degli Studi di Milano, Crema, Italy
e-mail: ernesto.damiani@unimi.it

S. Cimato
e-mail: stelvio.cimato@unimi.it

E. Damiani
Information Security Group, Khalifa University, Abu Dhabi, United Arab Emirates (UAE)

1 Introduction

Cloud computing allows organizations to radically outsource their ICT operations, deploying all applications and data on a virtualized infrastructure managed by an external supplier. However, concerns over potential disclosure of business data to other cloud tenants or to the cloud platform itself are still a major barrier to data outsourcing. At first sight, there seems to be a straightforward way to handle data privacy: storing and processing all data in encrypted form. In principle, this can be achieved via an encryption scheme that allows evaluating conditions on encrypted data, i.e., a *homomorphic encryption* scheme. Unfortunately, however, the practical applicability of homomorphic schemes to the cloud has been limited by their *complexity*.

Homomorphic encryption arithmetics are highly complex, and practically feasible queries are limited to equalities, or simple statistical functions such as mean and standard deviation [24]. Some researchers have tried to improve homomorphic encryption performance, e.g., by exploiting special hardware features such as Intel SIMD (single instruction, multiple data) extensions or vector processing capabilities of graphic processing units (GPUs) [33]; but even if they will become available, such techniques cannot be deployed on low-cost commodity hardware.

We would come to terms with the complexity issues of using homomorphic encryption if it totally resolved confi-

dentiality concerns of outsourced data. However, recent literature has shown that data outsourcing based on homomorphic encryption can be vulnerable to reaction attacks [35]. Indeed, homomorphic encryption is secure against attackers who may read outsourced data but have no access to query distribution and context. The cloud platform, however, is in the best conceivable position to monitor queries and the users' reaction toward their results. While "noise queries" can in principle be used to reduce observability, successful attacks against full homomorphic schemes based on observing query results and users' behavior have been described in the literature [35].

Considering these limitations of homomorphic encryption, almost a decade of research has been done to support data confidentiality in data outsourcing. Architecturally, this includes a wide range of solutions, from single server approaches [5,17,25] to more recent approaches relying on multiple cloud servers [3,15,18] that collaboratively store and manage data. Another line of research relies on secure multiparty computation (SMC) to perform data processing on the cloud without actually sharing the data [22]. However, SMC protocols scale linearly with data size and require minutes to join even small-sized databases [23]. Such performance may eventually allow privacy-preserving data mining but makes privacy-preserving data processing unfeasible in practice.

Extending our initial idea published in [18], in this paper, we adopt the concept of secret sharing to develop a secure solution for data outsourcing. Our approach, in addition to preserving confidentiality of outsourced data, is totally compatible with existing database technologies and efficiently supports server side execution of a wide range of queries.

In Shamir's classic secret sharing scheme, a secret is mapped onto a set of random shares using a randomized distribution polynomial. The main problem of splitting attribute values into shares is how to efficiently search within the pool of randomly generated shares. To support querying on shared data, authorized clients must be able to reconstruct the distribution polynomial associated with the searched values in the query. While Agrawal et al. [3], Wang et al. [31], and Emekci et al. [15] used hash functions to generate random coefficients, their approach is vulnerable to inference attacks when the honest-but-curious server has a priori knowledge about outsourced data distribution [13]. Moreover, Emekci et al. [15] assume that the servers, which host data shares, are non-communicating.

Considering above limitations, in this paper, we introduce *searchable sharing schemes* appropriate for data outsourcing that are robust against statistical analysis in spite of servers' collusion to pool their shares. As a result, it is possible in our approach to store all shares of a secret on a single database server. The main idea is to use client-aware partitioning of the domain of shares for searchable attributes. We propose three different schemes with different levels of security and search efficiency. We also discuss how our approach supports query processing for different kinds of queries. Utilizing a real data set, we implement our approach and evaluate the imposed communication, computation, and storage overheads at both client and server sides, finding them to be suitable for practical applications.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 covers background information on using secret sharing for data outsourcing scenario. Section 4 introduces our searchable sharing schemes and elaborates on their security. Section 5 explains the details of query execution scenarios. Section 6 summarizes the results of our empirical studies. Finally, Sect. 7 concludes the paper and outlines our future work.

2 Related work

Outsourced data confidentiality can be categorized into content, access, and pattern confidentiality [14]. *Content confidentiality* refers to the confidentiality of the outsourced data values. *Access confidentiality* is aimed at concealing the relation between a single query and data being returned. *Pattern confidentiality* is to hide that different accesses target the same data. When instead of a tuple, as the actual result of a query, a set of tuples including the result tuple, is returned back to the user, we have some degrees of access confidentiality for the server as the actual requested tuple is concealed among the returned ones. When a user poses the same query once more and receives the same set of result, the server can recognize that the two queries are identical while the actual accessed tuple remains confidential. Most research efforts on data outsourcing deal with content confidentiality, handling scenarios where data are outsourced to honest-but-curious servers. Few research works (see, for instance, [14]) deal with all aspects of confidentiality in their solution. In this section, we review techniques for preserving content confidentiality.

2.1 Encryption for single server solutions

The main problem with storing data in encrypted form is server-side query execution. Since external storage providers are not trusted, they cannot hold the decryption keys and are not allowed to decrypt data for processing queries.

Different techniques are available for querying encrypted data [4,5,17,21,25,28]. A popular solution is to use an auxiliary metadata structure, called index, to facilitate search over encrypted data. Bucket-based [17], hash-based, and B^+-tree indexing [11] are among some popular index-based methods. Index-based methods are usually susceptible to partial information exposure due to statistical analysis on the outsourced data and its indices [7].

Other proposals suggest homomorphic [17,32] and order-preserving [4] encryption schemes to support special queries, namely aggregation and range queries, respectively. Generalizing this idea, Popa et al. [25] jointly used different encryption schemes and proposed multilayer encryptions to support a subset of the SQL query language.

2.2 Fragmentation for multi-server solutions

With the aim of moving away from encryption and its complexities, some researchers have proposed to substitute encryption with data fragmentation. This line of research gave rise to the idea of providing confidentiality by hiding associations among values. Usually, the associations are defined by a set of confidentiality constraints over attributes or tuples of a relation. The constraints over attributes of a relation can be translated into vertical fragmentation [10], while horizontal fragmentation is used when the constraints are defined over tuples of the relation [29]. Data fragments are then distributed among multiple servers, which collaboratively process queries [8,10,29].

In general, fragmentation-based approaches are more efficient than encryption-based solutions. However, they are vulnerable to statistical attacks when servers collude, as well as to untrusted servers' inferences on data updates. In some situations, such as the one of a single sensitive attribute, neither horizontal nor vertical fragmentation can preserve the confidentiality of outsourced data. Ciriani et al. [9] proposed a hybrid technique using both encryption and fragmentation to provide content confidentiality for data outsourcing.

Secret sharing-based solutions Using secret sharing on data values has been introduced as a technique for preserving the confidentiality of outsourced data [2,3,6,15,18,19,30]. For outsourcing XML data, Brinkman et al. [6] suggest translating XML elements into polynomials. Each polynomial is then split into two parts, a random polynomial for the client and the difference between the original polynomial and the client polynomial for the server. However, their approach relies on the tree-based XML data model and is therefore targeted to search over outsourced XML data.

Hadavi and Jalili [19] and Tian et al. [30] use secret sharing to outsource relational data. For efficient search over outsourced data shares, they construct an order-preserving B^+ tree index structure with different techniques. While secret sharing is suggested to depart from encryption, nevertheless both schemes still use encryption to construct their index structures. More importantly, Tian et al.'s scheme reveals the order as well as the frequency of values for the untrusted server. Therefore, it is vulnerable to statistical analysis if the servers have a priori knowledge about original data distribution.

Agrawal et al. [2,3] as well as Emekci et al. [15] use hash functions to generate distribution polynomials of secret sharing. While their approach support different query types, it reflects the distribution of original dataset on its corresponding data shares and is consequently vulnerable to statistical analysis. Moreover, it cannot tolerate colluding servers and the security proofs in [15] rely upon the fact that the data servers cannot communicate to collude and pool their shares. Indeed, it is not farfetched that the servers collude and pool their shares to gain valuable information. It is also plausible that an adversary knows domains of attributes such as Age and Salary and uses them for statistical analysis over outsourced data shares.

Dautrich and Ravishankar [13] have identified some security limitations of using a (k, n) threshold secret sharing scheme for outsourcing scenario. Their analysis has shown that data confidentiality can be violated provided that honest-but-curious data servers find $k + 2$ secrets and their corresponding sets of shares even if the distribution vector of Shamir's scheme is kept hidden from the servers. The idea behind Dautrich and Ravishankar's attack is to align shares based on their ordering and then to map the aligned shares onto actual secret values relying upon servers' prior knowledge of original data distribution. They showed that in schemes such as [2] and [30], in which the order of shares is revealed to the servers, content confidentiality can be violated.

Departing from encryption, the sharing scheme described in this paper addresses the Dautrich and Ravishankar attack, even in spite of servers collusion, by perturbating the data distribution. Despite these security improvements over existing secret sharing-based methods, our approach preserves the searchability of shares and well-known efficiency of secret sharing solutions. In Sect. 6, we have experimentally examined the efficiency of query execution in our approach and compared it with another secret sharing-based solution.

3 Secret sharing for outsourcing scenario

Let us now recall some basic notions on secret sharing. A *secret sharing scheme* is a method of sharing a secret s among a set of participants $U = \{u_1, u_2, \ldots, u_n\}$ such that only authorized subsets of U, called access structure, can reconstruct s. Shamir [27] proposed a threshold (k, n) secret sharing scheme $(n \geq k)$ in which the access structure is a subset A of 2^U where $\forall B \in A, |B| \geq k$. That is, every k $(k \geq 1)$ or more participants can reconstruct s, while less than k participants cannot.

In order to apply a secret sharing scheme to data outsourcing, attribute values, data owner, and (honest-but-curious) data servers of the outsourcing scenario are identified, respectively, with the secrets, distributor, and participants to the secret sharing scheme. Using a (k, n) threshold secret sharing

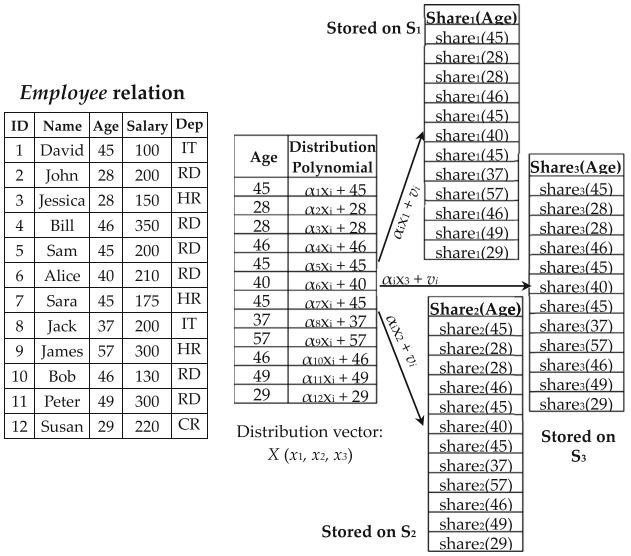

Fig. 1 Sharing *Age* values among three servers

scheme, each attribute value v is split into n shares, $share_i(v)$ $(1 \leq i \leq n)$ and stored on the correspondent server S_i among n servers, $S_1, S_2, \ldots, S_n$.

While in conventional secret sharing schemes the participants are trusted to pool their shares and collaboratively reconstruct secret values, in data outsourcing scenarios the servers hosting data shares are not trusted to obtain the secrets. Therefore, we define a *private vector* as the distribution/reconstruction key, which can be accessed just by the owner and authorized users. The data owner as well as authorized users collect shares from the servers and perform the reconstruction process using the reconstruction key.

Secret distribution To compute the shares of an attribute value v, the data owner chooses a prime P and produces a $k-1$ degree polynomial $p(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1x + a_0$, where $a_0 = v$ and other coefficients are chosen randomly from GF(P). Having a secret vector $X = (x_1, x_2, \ldots, x_n)$, x_i corresponds to S_i $(1 \leq i \leq n)$, the owner computes $share_i(v) = p(x_i)$ and stores it on S_i. Figure 1 depicts sharing *Age* values of a simple Employee relation through a (2, 3) secret sharing scheme and the distribution vector $X = (x_1, x_2, x_3)$. In the figure, for each value v, a first degree polynomial is constructed with a random coefficient α_i and the shares are calculated and stored on the corresponding servers.

Secret reconstruction For each attribute value, there are n points $(x_i\ ,\ p(x_i))$ through which the polynomial $p(x)$ passes. Therefore, k distinct points are enough to uniquely reconstruct a polynomial of order $k-1$. When a trusted party, who knows the distribution vector X, receives at least k shares of secret v, the secret can be reconstructed. That is, the distribution vector X is the only storage overhead, imposed on the clients for reconstructing secrets.

3.1 Security definition

When the naïve method of Shamir's secret sharing scheme is used to share attribute values, the generated set of shares for each data server does not reveal any information about the distribution or the frequency of original values. However, a major problem is searching within attribute values before reconstruction of shares. Since neither the owner nor authorized users store the random coefficients, data retrieval according to user query 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. For this purpose, a method is required to identify the shares that satisfy the query condition before reconstruction.

Dautrich and Ravishankar [13] highlighted some security limitations of using (k, n) secret sharing for the data outsourcing scenario. They assume that if data servers have access to at least $k+2$ attribute values and their corresponding sets of shares, the schemes in [2,19,30] cannot provide data confidentiality. While they use shares ordering to find some valid associations between shares and secret values, a mitigation of their attack is to prevent revealing such associations. To this end, we shall define our notion of security on the basis of their attack and propose secret sharing schemes in which associations among original values and their corresponding shares are not revealed to honest-but-curious servers. Indeed, such schemes are weaker with respect to the basic Shamir's scheme in terms of data protection, trading off perfect security for the capability of efficiently searching on the shares.

Definition 1 A secret sharing scheme for data outsourcing is secure if the probability of a passive attacker holding a share to find the corresponding original value is almost equal to that of a random guess.

According to above definition, deterministic sharing schemes for secure data outsourcing like the ones described in [2,3,15] are not secure. It is because equal values are split into a set of equal shares. An adversary who has prior information about original data distribution can then make inferences on share-value associations. In our sharing schemes (see Sect. 4), we perturbate original data distribution in their shares and distribute shares across their domain using a managed-random generation of coefficients for distribution polynomials. This way, the resulting distribution of shares can get arbitrarily closer to a uniform distribution as the domain of shares gets larger. Therefore, given a data distribution and any $\varepsilon > 0$, it is possible to determine the size of shares domain so that the difference between the probability of finding a mapping and that of a random guess is less than ε. In the next section, we introduce our searchable sharing schemes and discuss in detail their efficiency and security with respect to Definition 1.

4 Searchable sharing schemes

To solve the problem of efficient query processing over data shares, we propose secret sharing schemes where searching a subset of shares is possible, while the security is improved scheme by scheme to satisfy Definition 1.

4.1 Assumptions and threat model

We follow a database outsourcing model in which a data owner outsources her relational data $R(attr_1, attr_2, \ldots, attr_m)$ to honest-but-curious server(s). Then, the owner and authorized clients submit queries to the servers continuously. The queries are issued over any searchable attribute $attr_j$ of domain D_v. The notion of searchable attribute simply means that the users can pose queries regarding that attribute. That is, the attribute can appear in WHERE predicate of queries. Our focus is on searching numeric values, though the approach is immediately extendable to character data upon which exact match queries can be executed.

With the aim of providing content confidentiality of the outsourced data, we use a (2, 2) secret sharing scheme. Thus, $p(x_i) = ax_i + v$ $(i = 1, 2)$ is the general form of the distribution polynomial, where v is the value to be distributed, a is the random coefficient from a specified domain, x_i is the ith member of the hidden distribution vector $X = \{x_1, x_2\}$, and $p(x_i)$ is the S_i's corresponding share of v, i.e., $share_i(v)$ (Fig. 1). We are well aware that the choice of k $(k > 1)$ and n $(n \geq k)$ parameters for a threshold (k, n) scheme affects the availability and fault tolerance aspects of the system, which were not the aim of this work. From the confidentiality viewpoint, choosing a large k does not offer more security while it increases the communication cost as well as the client-side computation cost for interpolating original values.

We assume a real domain of values for shares and coefficients rather than arithmetic calculations over GF(P).

Let D_v and D_s be domains of the searchable attribute and its shares, respectively. We assume $|D_v| \leq |D_s|$, that is, the domain of shares is big enough to prevent mapping two different values onto equal shares. So, we have the following assumption in our sharing schemes:

$$\forall v, v' \in D_v : v \neq v' \Rightarrow share_i(v) \neq share_i(v');\ i = 1, 2$$

We consider the following threat model:

- Data servers are honest-but-curious. They execute submitted queries honestly on outsourced data and send complete and authenticated result sets. However, they are curious to increase their knowledge about confidential data.
- Data servers have a priori knowledge of outsourced data distribution. They might know the domain of values, the minimum or maximum values, and some information about the frequency of values. However, they do not have a priori knowledge about the system query workload.
- Data servers can communicate and collude with each other to extract knowledge about outsourced data. With this assumption, it is possible to store shares of secrets either on different servers, which can communicate and collude, or on a single server.

Also, we assume that clients, i.e., the users' machines that mediate user requests to service providers, are trusted and do not disclose confidential information such as the distribution vector X. Moreover, they have the same authorities to access the outsourced data by submitting queries through clients. When users have selective access to the outsourced data, our solution can be extended using proposals such as [34] to adopt multi-user environments with selective accesses.

4.2 Solution overview

The general idea behind searchable sharing schemes is to share values using a distribution polynomial, which can be generated later on by clients in order to translate user queries into the queries over shares. Agrawal et al. [3,15] use hash functions for generating polynomials so that their coefficients are outputs of defined hash functions on secret values. Instead of adopting such a deterministic sharing scheme in which the distribution of shares can reveal information about original values, we use the idea of partitioning the domain of shares through partitioning the domain of random coefficients in the distribution polynomial of the sharing scheme. To distribute each value v, the coefficient a of the distribution polynomial $ax + v$ is assigned a random value from a particular partition of a's domain. Thus, equal secret values are more likely to be mapped onto different shares. Our sharing techniques perturbate the original values' distribution provided that the domain of shares is large enough compared with the domain of the searchable attribute. This perturbation improves the robustness against statistical analysis. There are two basic steps for our partitioning-based secret sharing:

1. Partitioning the domain of coefficients (choosing a partitioning method) and
2. Mapping partitions onto secret values (choosing a mapping function).

At first, partitioning is performed with respect to the following definition.

Definition 2 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. $\forall\, d_i, d_j \in D\ (i \neq j) : d_i \cap d_j = \emptyset$

The next step is to define a function to map values of attribute domain onto the partitions with respect to the following definitions.

Definition 3 The mapping function $F : V \rightarrow D$ is a function that maps value $v \in V$ onto partition(s) $d \subseteq D$ (denoted by $v \mapsto d$)

Definition 4 A mapping function is an order-preserving mapping function if it preserves the ordering relation of original values of a domain D_v in their shares of a domain D_s:

$$\forall v, v' \in D_v, v \mapsto d, v' \mapsto d' :$$
$$v < v' \Rightarrow share_i(v) < share_i(v') \ ; \ i = 1, 2$$

Given a domain of coefficient for the threshold $(2, 2)$ sharing scheme, the distribution algorithm of the secret sharing scheme calculates $share_i(v)$ of a secret v for each server S_i such that if $v \mapsto d$ then $\text{Min}(d).x_i + v \leq share_i(v) \leq \text{Max}(d).x_i + v$. Obviously, our mapping function must be kept secret from the servers.

Based on the above definitions, different methods of partitioning are possible with differences in terms of query processing efficiency, client-side storage overhead, and security. In the subsequent sections, we introduce three partitioning methods along with mapping functions to have searchable sharing schemes.

4.3 Simple partitioning

A straightforward way of partitioning a domain is to divide it into equal partitions.

Partitioning method Let $D_c = [a_{first} \,..\, a_{last}]$ and D_v be domains of the coefficient and attribute, respectively. We divide D_c into $|D_v|$ equal consecutive partitions $d_1, d_2, \ldots,$ and $d_{|D_v|}$, where $d_1 = [a_{first} \,..\, a_{first} + \frac{|D_c|}{|D_v|}]$, $d_2 = [a_{first} + \frac{|D_c|}{|D_v|} \,..\, a_{first} + 2\frac{|D_c|}{|D_v|}]$, $d_3 = [a_{first} + 2\frac{|D_c|}{|D_v|} \,..\, a_{first} + 3\frac{|D_c|}{|D_v|}]$, $\ldots,$ and $d_{|D_v|} = [a_{first} + (|D_v| - 1)\frac{|D_c|}{|D_v|} \,..\, a_{last}]$.

Now we map the attribute values in D_v onto the above partitions in D_c.

Mapping function The mapping function $F : V \rightarrow D$ maps a value $v \in D_v$ onto a partition $d \subseteq D_c$ (denoted by $v \mapsto d$) where:

1. $\forall\, v, v' \in D_v, v \mapsto d, v' \mapsto d' \ : \ v \neq v' \Leftrightarrow d \cap d' = \emptyset$
2. $\forall\, v, v' \in D_v, v \mapsto d, v' \mapsto d' \ : \ v = v' \Leftrightarrow d = d'$

The first property says that each value is assigned to a separate partition. The second one states that equal values are assigned the same partition.

Suppose that the searchable attribute *Dep* of the *Employee* relation in Fig. 1 is to be shared by our searchable sharing schemes. The domain of its values includes HR (human resource), CR (customer relationship), IT (information technology), and RD (research and development). Figure 2 shows the histogram of *Dep* values in the *Employee* relation in Fig. 1. An outline of simple partitioning with an order-preserving mapping function is also shown in Fig. 3. If the mapping function is order preserving, the owner and query clients do not need to store the information related to the association of partitions to original values, except the partition width. Instead, for an order-obfuscating mapping, clients must store the associations of values and their partitions.

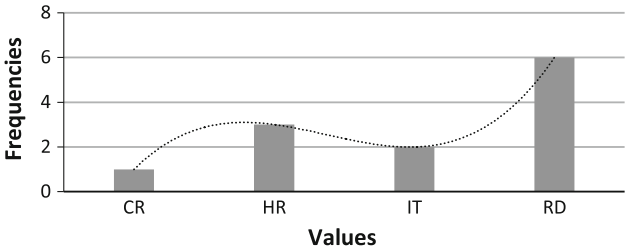

Fig. 2 Histogram of original values

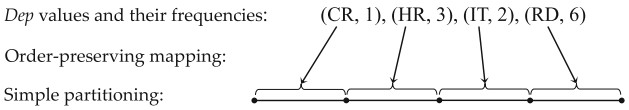

Fig. 3 Schematic view of simple partitioning

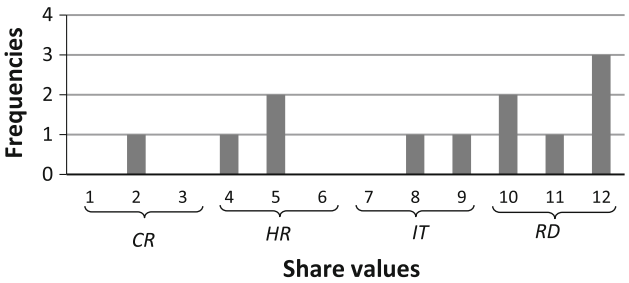

Fig. 4 Histogram of shares, distributed with simple partitioning

If the domain of shares is larger than the domain of attribute values, equal values can potentially be mapped onto different shares. This leads to the perturbation of original data distribution. Figure 4 shows a typical histogram of shares after sharing *Dep* values according to Fig. 3.

4.3.1 Security analysis

As shown in Fig. 4, the distribution of shares is perturbated compared with the distribution of original values. However, assigning partitions with the same width to values having different frequencies can be a source of inference. An adversary can infer that a group of shares with high frequency (more than one) are more probably mapped onto values with higher frequencies. Then, she will be able to partially find out some correspondences between shares and original values.

In Fig. 4, it is shown that the group of shares belonging to RD have higher frequency. Then, an adversary can infer that they are more probably mapped onto the original value RD. This intuitive notion can be formalized as follows:

Theorem 1 *Simple partitioning is not secure with respect to Definition 1.*

Proof sketch If the mapping function preserves the order, an adversary can map the minimum (maximum) share value onto the minimum (maximum) value of the database. So, the security is violated since it is possible to find some correspondence between values and their shares. Even if the mapping function is not order preserving, the distribution of shares still reveals some information about original values. Consider a situation where there are high-frequency values in database. Then, an adversary can find some groups of shares whose frequencies are larger than (or equal to) one. In such a case, the probability of finding a correspondence between a group of highly frequent shares and a highly frequent value is more than that of a random guess for adversaries who have a priori knowledge of data distribution. The inference can also be made on low-frequency values. □

Simple partitioning is only suitable when there are no remarkable differences in the frequency of values. Otherwise, it does not satisfy our security definition even if we use an order-obfuscating mapping function.

4.4 Weighted Partitioning

The problem of simple partitioning is that all partitions have the same width regardless of values' frequency. However, intuition suggests that we can partition the domain of coefficient so that a wider partition is assigned to a value with higher frequency. This way, assuming the size of coefficient domain is large enough ($|D_c| \geq \mathrm{N} =$ database tuples), each attribute value can potentially be mapped onto a non-repetitive share.

Partitioning method Let D_c and D_v be domains of the coefficient and attribute, respectively. We divide D_c into $|D_v|$ partitions $d_1, d_2, \ldots,$ and $d_{|D_v|}$, where $|d_i|$ as the width of the v_i's corresponding partition is computed by $|d_i| = \frac{|D_c| freq(v_i)}{N}$ $(1 \leq i \leq |D_v|, v_i \in D_v)$.

Mapping function The mapping function $F : V \rightarrow D$ maps a value $v \in D_v$ onto a partition $d \subseteq D_c$ (denoted by $v \mapsto d$) where:

1. $\forall\ v \in D_v, d \subseteq D_c, v \mapsto d\ :\ |d| = \frac{|D_c| freq(v)}{N}$
2. $\forall\ v, v' \in D_v, v \mapsto d, v' \mapsto d'\ :\ v \neq v' \Leftrightarrow d \cap d' = \emptyset$
3. $\forall\ v, v' \in D_v, v \mapsto d, v' \mapsto d'\ :\ v = v' \Leftrightarrow d = d'$

As before, each value is assigned to a separate partition and equal values are assigned the same partition. To prevent inference on the ordering, we use an order-obfuscating mapping function. To make sure that the order of shares is totally obfuscated, the owner can relocate partitions across the domain of the coefficient by choosing a random permutation among $|D_v|!$ possible ones. Then, the data owner stores the specified mapping of values and their corresponding partitions (i.e., the widths and starting points of partitions) and announces this information to clients, who are in charge of query translation.

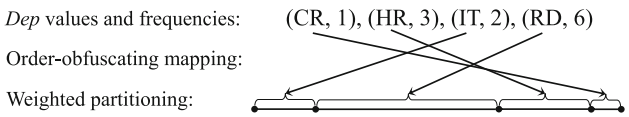

Fig. 5 Schematic view of weighted partitioning

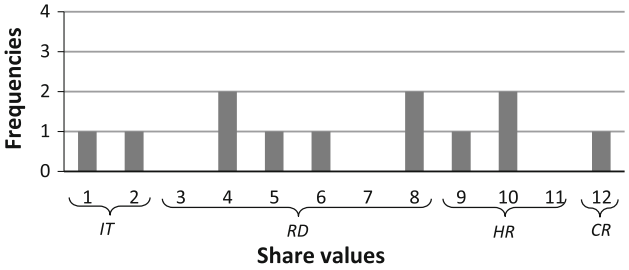

Fig. 6 Histogram of shares, distributed using weighted partitioning

Figure 5 shows a schematic view of weighted partitioning with an order-obfuscating mapping function for the domain of values whose frequencies are shown in Fig. 2. To distribute each value, the owner locates the corresponding partition and chooses randomly a point from the specified range of the coefficient domain in the distribution polynomial. Figure 6 shows the histogram of shares for the given distribution of values using the mapping shown in Fig. 5. As shown in Fig. 6, shares are distributed more uniformly than in simple partitioning (Fig. 4), since the width of a partition is determined by the frequency of its corresponding value.

4.4.1 Security analysis

In this section, we show that an adversary, who observes the shares, cannot infer any valuable information about associations between values and shares. We also prove that by observing stored data shares, servers can guess a mapping between shares and values with a probability that is not better than the one of a wild guess. Finally, we show that this method of partitioning is vulnerable to adversaries who can observe queries.

In weighted partitioning method, the width of a partition corresponding to an attribute value is proportional to the frequency of the value. Therefore, high-frequency values are smoothly mapped onto low-frequency shares that prevent inference based on the frequency of shares. An attribute value

v with the frequency f is independently distributed f times on the server S_i $(i = 1, 2)$. Equation (1) shows the probability of the existence of a v's share on S_i with the frequency f' where $f' > \frac{f}{2}$.

$$Pr(freq(share_i(v)) = f') = \frac{f\binom{f}{f'}(f-1)^{f-f'}}{f^f} \quad (1)$$

Equation (1) says that for a high-frequency value the frequency of its shares on a server is considerably different than the frequency of the value.

Lemma 1 *The probability of mapping freq(v) equal values v onto the same shares from v_i's corresponding partition is*

$$Pr(freq(share_i(v) = freq(v)) = \frac{freq(v)}{freq(v)^{freq(v)}}$$

Proof We obtain the probability of the frequency of a share being equal to the frequency of its corresponding value by replacing both f and f' with $freq(v)$ in (1). □

Lemma 1 states that the probability of the frequency of a share being equal to the frequency of its corresponding data value approaches to zero for highly frequent values.

Lemma 2 *The probability of having the same distribution for values and their corresponding shares is*

$$\prod_{i=1}^{|D_v|} \frac{freq(v_i)}{freq(v_i)^{freq(v_i)}} \quad ; \quad v_i \in D_v$$

Proof The probability of having the same distribution of original values and shares for a server is equal to the one of the situations where all values and their corresponding shares have the same frequency. Each value is distributed randomly and independently from other values. So, the probability is calculated by the multiplication of probabilities for all values in attribute domain. □

Let us consider our sample domain of *Dep* values in the *Employee* relation (Fig. 1). According to Lemma 2, the probability of having the same distribution for values and their corresponding shares is calculated as:

$$\prod_{v\in\{CR,HR,IT,RD\}} \frac{freq(v)}{freq(v)^{freq(v)}} = \frac{1}{1} \times \frac{3}{3^3} \times \frac{2}{2^2} \times \frac{6}{6^6} = 7.14 \times 10^{-6}.$$

This is a very low probability for a small domain of values. This probability gets even closer to zero for larger domains and for database relations containing highly frequent values. Concluding from Lemmas 1 and 2, high-frequency shares as a potential source of inference hardly appear as share values on the server. On the other hand, low-frequency shares, e.g., between 0 and 4, can be mapped onto almost all values in the relation because a share value with frequency f can be associated with an attribute value v with $freq(v) \geq f$. Thus, after the distribution phase of secret sharing scheme, servers cannot estimate associations between values and shares any better than a wild guess.

Theorem 2 *The weighted partitioning method is secure for an adversary who observes the distributed shares on a server.*

Proof Assume that the adversary chooses a share x stored on a server $S_i (i = 1, 2)$ and wants to find its corresponding original value y. We prove that, from the adversary viewpoint, $\Pr(x = share(y))$ is almost equal to $\frac{1}{|D_v|}$, where D_v is the domain of the searchable attribute.

Let us model the frequency of values in the original dataset as f_1, f_2, …, and f_l, where f_i $(1 \leq i \leq l)$ is the number of values in the original dataset with the frequency i, and l is the maximum frequency of values. So, $N = \sum_{i=1}^{l}(i \times f_i)$ is the total number of database tuples and $|D_v| = \sum_{i=1}^{l} f_i$ is the cardinality of the domain of values. Now, we extract $Pr(x = share(y))$ as below:

$$Pr(x = share(y)) = \sum_{i=0}^{l} Pr(x = share(y)|freq(x) = i) \times Pr(freq(x) = i)$$

A share value x with $freq(x) = f$ can be associated with an attribute value v with $freq(v) \geq f$. Therefore,

$$Pr(x = share(y)|freq(x) = f) = \frac{1}{\sum_{i=f}^{l} f_i} \quad (2)$$

On the other hand, to compute $Pr(freq(x) = f)$, we have:

$$\begin{aligned} &Pr(freq(x) = f) \\ &= \sum_{i=1}^{|D_v|} Pr(freq(x) = f | x = share(v_i)) \times Pr(x = share(v_i)) \\ &= \sum_{i=1}^{|D_v|} \left(\binom{freq(v_i)}{f} \left(\frac{1}{freq(v_i)}\right)^f \left(1 - \frac{1}{freq(v_i)}\right)^{freq(v_i)-f} \right) \\ &\quad \times \left(\frac{freq(v_i)}{N}\right) \\ &= \sum_{i=f}^{l} \frac{i \times f_i}{N} \binom{i}{f} \left(\frac{1}{i}\right)^f \left(1 - \frac{1}{i}\right)^{i-f} \end{aligned} \quad (3)$$

From (2) and (3), $Pr(x = share(y))$ is calculated by (4):

$$Pr(x = share(y)) = \frac{1}{N} \sum_{f=0}^{l} \frac{\sum_{i=f}^{l} i \times f_i \binom{i}{f} \left(\frac{1}{i}\right)^f \left(1 - \frac{1}{i}\right)^{i-f}}{\sum_{i=f}^{l} f_i} \quad (4)$$

Expanding (4), we get:

$$
\begin{aligned}
&Pr(x = share(y)) \\
&= \frac{1}{N}\Bigg(f_1\left(\frac{1}{f_1+\cdots+f_l}\right) \\
&\quad + 2f_2\left(\frac{\frac{3}{4}}{f_1+\cdots+f_l}+\frac{\frac{1}{4}}{f_2+\cdots+f_l}\right)+\ldots \\
&\quad + lf_l\Bigg(\frac{\left(\frac{l-1}{l}\right)^l}{f_1+f_2+\cdots+f_l}+\frac{l\left(\frac{1}{l}\right)\left(\frac{l-1}{l}\right)^{l-1}}{f_2+\cdots+f_l} \\
&\quad +\cdots+\frac{\left(\frac{1}{l}\right)^l}{f_l}\Bigg)\Bigg) \\
&= \frac{1}{N}\Bigg(\frac{1}{f_1+f_2+\cdots+f_l} \\
&\quad \times\Bigg(1f_1+2f_2\left(\frac{3}{4}+\frac{1}{4}\left(\frac{f_1+f_2+\cdots+f_l}{f_2+\cdots+f_l}\right)\right)+\ldots \\
&\quad + lf_l\Bigg(\binom{l}{0}\left(\frac{l-1}{l}\right)^l+\binom{l}{1}\frac{1}{l}\left(\frac{1-l}{l}\right)^{l-1} \\
&\quad +\frac{\binom{l}{2}\left(\frac{1}{l}\right)^2\left(\frac{l-1}{l}\right)^{l-2}(f_1+f_2+\cdots+f_l)}{f_2+\cdots+f_l}+\ldots \\
&\quad +\frac{\left(\frac{1}{l}\right)^l(f_1+f_2+\cdots+f_l)}{f_l}\Bigg)\Bigg)\Bigg)
\end{aligned}
$$

According to Lemma 1, the coefficients of $i \times f_i$ in the above formula approach one. Therefore,

$$
\begin{aligned}
&Pr(x = share(y)) \\
&\approx \frac{1}{N}\left(\frac{1}{f_1+f_2+\cdots+f_l}(1f_1+2f_2+\cdots+lf_l)\right) \\
&= \frac{1}{N}\left(\frac{N}{f_1+f_2+\cdots+f_l}\right) = \frac{1}{f_1+f_2+\cdots+f_l} \\
&= \frac{1}{|D_v|}
\end{aligned}
$$

□

For the very small domain Dep = {CR, HR, IT, RD} and its values in the *Employee* relation with just 12 tuples (Fig. 1), the probability of finding a correct association between a share and its corresponding value is equal to 0.274, which is almost equal to $\frac{1}{4}$. This probability would be closer to $\frac{1}{|D_v|}$ for larger domains and also for greater number of relation tuples. For instance, consider another typical domain of five values v_1 to v_5 where the frequencies of values in a relation, consisting of 20 tuples, are 4, 1, 5, 8, and 2. For such a relation, (5) computes the probability of finding a correct association between a share and a value. As we can see in (5), the result is sensibly closer to $\frac{1}{|D_v|}$, i.e., 0.2, compared with the case of Dep domain and the *Employee* relation with 12 tuples.

$$
\begin{aligned}
&Pr(x = share(y)) \\
&= \frac{1}{20}\sum_{f=0}^{8}\frac{\sum_{i=f}^{8} i \times f_i\binom{i}{f}\left(\frac{1}{i}\right)^f\left(1-\frac{1}{i}\right)^{i-f}}{\sum_{i=f}^{8} f_i} \\
&= 0.2019
\end{aligned} \tag{5}
$$

While it is clear that untrusted servers cannot find the correspondence between shares and values merely by observing stored shares, it may be possible for them to infer some information by monitoring query processing. The cloud supplier, or any adversary who has access to communication channels, can observe query answers as a sets of shares for which a query condition is satisfied. A set of shares is potentially a partition with specified width, which can be mapped onto an attribute value. This range of satisfying shares can be a partition whose width reveals the frequency of the corresponding actual value.

Assume that the query SELECT * FROM R WHERE Dep = "RD" is submitted frequently. Even without a priori knowledge on queries, servers can find out that a fixed group of shares, i.e., the range of shares [4..8] in Fig. 6, are always returned for some queries. Thus, they can infer that these shares belong to the same partition, whose length reveals, in turn, the frequency of original value. Therefore, the weighted partitioning method is secure while an adversary can observe distributed shares (in any snapshot of the system in operation), but it is not secure if the adversary can observe query answers. Consequently, for the cloud outsourcing scenario where the cloud platform can observe all queries, the weighted partitioning method is not secure with respect to Definition 1.

4.5 Secure partitioning

Secure partitioning is aimed at concealing the width of partitions assigned to each value using multiple partitions assign-

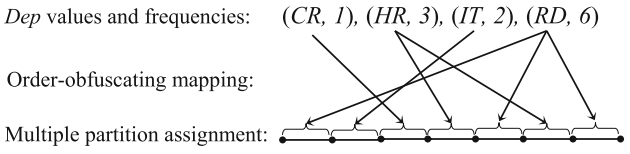

Fig. 7 Schematic view of secure partitioning method

ment, substantially alleviating the possibility of frequency analysis on the part of attackers. Obviously, the partitions assigned to a value need not be consecutive in the coefficient domain of the distribution polynomial.

Partitioning method Let D_c and D_v be domains of the coefficient and attribute, respectively. With t as the width of partitions chosen by the owner, D_c is divided into $\frac{|D_c|}{t}$ equal partitions $d_1, d_2, \ldots$, and $d_{\frac{|D_c|}{t}}$ of the same width.

Mapping function The mapping function $F : V \rightarrow D$ maps a value $v \in D_v$ to a set of partitions of the width t (denoted by $v \mapsto P$) where:

1. $|P| = \left\lceil \frac{freq(v_i)}{t} \right\rceil$
2. $\forall\, v, v' \in D_v, v \mapsto P, v' \mapsto P' : v \neq v' \Leftrightarrow P \cap P' = \emptyset$
3. $\forall\, v, v' \in D_v, v \mapsto P, v' \mapsto P' : v = v' \Leftrightarrow P = P'$

The first property indicates the required number of partitions for a value. The second one says that each value is assigned a disjoint set of partitions, and the third one says that equal values are assigned the same set of partitions.

Again, we prefer using an order-obfuscating mapping. To have a totally order-obfuscating mapping function, a random permutation of partitions, among $\lceil \frac{|D_c|}{t} \rceil !$ possible permutations, is chosen by the owner to determine the location of partitions across the domain of shares. The data owner then stores the mapping function and sends it to authorized clients. Figure 7 shows a schematic view of the secure partitioning method with an order-obfuscating mapping for our example domain. In this case, the distribution of shares across the domain is almost similar to the weighted partitioning method. The only difference is that we "slice" a wide partition to obtain a set of separate smaller ones.

A major issue of our partitioning method is how to determine the partition width t. Consider three scenarios of determining t:

- For $t = 1, v_i \mapsto P, |P| = freq(v_i)$. The owner as well as clients should store $freq(v)$ partition information, which is more than storing the actual values.
- For $|t| \geq \text{Max}(freq(v_i)), v_i \mapsto P, |P| = 1$. Each value is assigned a partition similarly to the weighted partitioning method. So, there is the possibility of inference as there is in the weighted partitioning method.
- For $t = \overline{freq(v)}, v_i \mapsto P, |P| = \frac{freq(v_i)}{\overline{freq(v)}}$. This gives us a trade-off between security and client-side storage overhead.

In our secure partitioning method, given an original dataset, the average of frequencies is computed by the owner and determined as the partition width t. Anyhow, for any partition width $t \neq 1$, there is a limited overhead due to the indivisibility of values' frequencies to the chosen partition width.

4.5.1 Security analysis

Our secure partitioning method is similar to the weighted partitioning in terms of assigned partition width. The assigned partition is either a contiguous one (weighted partitioning) or divided into separate fragments (secure partitioning). Therefore, we obtain same probabilities discussed in Sect. 4.4.1 for secure partitioning method, assuming that values frequencies are a multiply of the partition width t. If the frequencies are not a multiply of t, then required points in the domain of shares increase for a value of frequency f since we need $\lceil \frac{f}{t} \rceil$ partitions. This decreases the probability of finding a correct mapping between values and their shares compared with the weighted partitioning method.

The rationale behind the security of this partitioning method against observing query processing is the indistinguishability of equality and range queries.

Theorem 3 *Our secure partitioning method is secure against adversaries who can observe query answers.*

Proof sketch According to our threat model, adversary is not aware of the pattern of queries submitted to the system (Sect. 4.1). This assumption is not strict as it may look at first sight, because query distribution can be perturbated at will, although at the expense of additional messages, by introducing fake clients/queries [1]. Moreover, it is more difficult to draw inferences from queries and answers when the schema of the outsourced relation is encoded at server side, as proposed in [16]. On the other hand, equality queries in secure partitioning method can be translated into range queries with possibly separate ranges in WHERE predicate as well as a range query. So, equality and range queries are indistinguishable for adversaries who do not have a priori knowledge of system query workload. When honest-but-curious servers observe a group of returned shares, belonging to more than one partition, it is unclear for them that the shares are results of an equality query (can be mapped onto one actual value) or results of a range query (can be mapped onto a set of values). □

5 Query processing

In this section, we discuss processing of equality, range, projection, join, aggregation, and update queries. We apply the same basic idea: a client who knows the mapping function translates a query over a searchable attribute into an equivalent query on a range of shares. A great advantage of our approach is its compatibility with server-side indexing. Inefficient indexes are known to impair the performance and scalability of query processing in large encrypted databases with many concurrent users. Our methods allow servers to build efficient local indexes on outsourced share values.

Depending on the partitioning method, client-side query translation can be less or more convenient. As suggested by intuition, translation is simpler for simple partitioning with order-preserving mapping than it is for secure partitioning with an order-obfuscating mapping.

Equality queries For simple and weighted partitioning methods, the condition $attr = v$ in a query, ($attr$ is a searchable attribute and v is a value) is translated into $\text{Min}(share_i(v)) \leq share_i(v) \leq \text{Max}(share_i(v))$ and sent to S_i $(i = 1, 2)$. For a (2, 2) threshold scheme with the distribution polynomial $share_i(v) = a.x_i + v$, the minimum (maximum) possible ith share of v is obtained by putting the minimum (maximum) possible value of a from the v's corresponding partition in the distribution polynomial:

$$\text{Min}\ (share_i(v)) = (\text{Min}\ (a)).x_i + v,$$
$$\text{Max}\ (share_i(v)) = (\text{Max}\ (a)).x_i + v$$

For secure partitioning method in which the mapping function assigns a set of partitions P to a value v, the condition $attr = v$ is translated into a disjunctive set of range conditions, each range is associated with a partition $p \in P$.

Consider a simple equality query SELECT * FROM Employee WHERE Age = 20 on a typical relation where the values of attribute "Age" have been shared using our sharing schemes. For simple and weighted partitioning methods, the client translates the query into a range query of the form SELECT * FROM Employee WHERE $\text{Min}(share_i(20)) \leq \text{Age} \leq \text{Max}(share_i(20))$ and submits it to S_i $(i = 1, 2)$. Similarly, for the secure partitioning method, it is translated into:

SELECT * FROM Employee
WHERE $\text{Min}(p_1) \leq \text{Age} \leq \text{Max}(p_1)$ OR
$\text{Min}(p_2) \leq \text{Age} \leq \text{Max}(p_2)$ OR . . . OR
$\text{Min}(p_{\lceil \frac{freq(20)}{\overline{f}} \rceil}) \leq \text{Age} \leq \text{Max}(p_{\lceil \frac{freq(20)}{\overline{f}} \rceil})$.

p_i is a partition assigned to value 20, $freq(20)$ is the frequency of the value 20 in the relation, and $\overline{f}$ is the average of frequencies as the partition width. Receiving shares of satisfying tuples from two servers, the client can obtain the original values by a Lagrange interpolation.

Of course, it is possible to combine conditions on searchable attributes, conjoining or disjoining them. In such a case, the servers return a set of shares so that the entire composite condition is satisfied.

Range queries Our approach supports translating range queries into sequences of equality queries, without the need for ad hoc indices [12]. For our sharing schemes with an order-preserving mapping, a range query is translated into a range query with a change in range boundaries. The condition $v \leq attr \leq v'$, where $v \mapsto d$ and $v' \mapsto d'$, is translated into $\text{Min}(d).x + v \leq share(attr) \leq \text{Max}(d').x + v'$. Consider a simple range query SELECT * FROM Employee WHERE $50 \leq \text{Age} \leq 80$. The query submitted to S_i $(i = 1, 2)$ is: SELECT * FROM Employee WHERE $\text{Min}(share_i(50)) \leq \text{Age} \leq \text{MAX}(share_i(80))$.

For simple and weighted partitioning schemes with order-obfuscating mapping, the translation results in having multiple ranges in the translated query for each range in the original query. That is, a range of attribute values is usually mapped onto several ranges of shares. Consider the sample query SELECT Salary FROM Employee WHERE $50 < \text{Age} < 55$. This is translated into the following query and sent to S_i:

SELECT Salary FROM Employee WHERE
$\text{Min}(share_i(51)) \leq \text{Age} \leq \text{Max}(share_i(51))$ OR
$\text{Min}(share_i(52)) \leq \text{Age} \leq \text{Max}(share_i(52))$ OR
$\text{Min}(share_i(53)) \leq \text{Age} \leq \text{Max}(share_i(53))$ OR
$\text{Min}(share_i(54)) \leq \text{Age} \leq \text{Max}(share_i(54))$.

For secure partitioning method, the client-side query translation is a bit more complex. Each value in the queried range is mapped onto several ranges of shares. For example, SELECT Salary FROM Employee WHERE $52 < \text{Age} < 55$ is translated into the following query and sent to S_i:

SELECT Salary FROM Employee WHERE
$\text{Min}(p_1(53)) \leq \text{Age} \leq \text{Max}(p_1(53))$ OR
$\text{Min}(p_2(53)) \leq \text{Age} \leq \text{Max}(p_2(53))$ OR . . . OR
$\text{Min}(p_{\lceil \frac{freq(53)}{\overline{f}} \rceil}) \leq \text{Age} \leq \text{Max}(p_{\lceil \frac{freq(53)}{\overline{f}} \rceil})$ OR
$\text{Min}(p_1(54)) \leq \text{Age} \leq \text{Max}(p_1(54))$ OR
$\text{Min}(p_2(54)) \leq \text{Age} \leq \text{Max}(p_2(54))$ OR . . . OR
$\text{Min}(p_{\lceil \frac{freq(54)}{\overline{f}} \rceil}) \leq \text{Age} \leq \text{Max}(p_{\lceil \frac{freq(54)}{\overline{f}} \rceil})$.

Range queries generate a complete set of results as well as equality queries.

Projection queries Projection is supported by all our sharing schemes independent of partitioning method and mapping function. This is because a relation is shared among servers in the granularity of attribute values. We remark that, while all searchable attributes in the relation should be shared using our sharing schemes, for non-searchable attributes Shamir's naïve scheme can be used. For a query such as SELECT

Salary FROM Employee WHERE $20 \leq$ Age ≤ 25, each S_i ($i = 1, 2$) finds satisfying share values of Age, as discussed for range queries, and finally returns the corresponding shares of Salary to the client. The client can interpolate original Salary values after receiving the corresponding pairs of shares.

Aggregate queries Thanks to the additive homomorphism property of Shamir's scheme, queries with SUM aggregation function are supported in all of our partitioning methods for both order-obfuscating and order-preserving mapping functions. Using the weighted partitioning method, a sample query SELECT SUM(Salary) FROM Employee WHERE $20 \leq$ Age ≤ 25 is translated into SELECT Salary FROM Employee WHERE Min($share_i(20)$) $\leq share(Age) \leq$ Max($share_i(25)$) and sent to S_i. S_i locally calculates the summation of satisfying Salary shares and sends the result. The client computes the total summation value when it receives two sum values from the servers.

Queries containing the COUNT function such as SELECT COUNT(Salary) FROM Employee WHERE Age = 20 are executed simply by reforming as SELECT COUNT(Salary) FROM Employee WHERE Min($share_i(20)$) $\leq$ Age $\leq$ Max ($share_i(20)$). Considering the servers are honest in executing queries, COUNT queries can be sent to only one server instead of sending to all, incurring less communication overhead. The process of SUM and COUNT queries is similar for secure partitioning method except that the query translation is different.

Executing MIN/MAX queries 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). For the weighted partitioning method, it should be translated into SELECT *share*(Salary) FROM Employee WHERE Min($share_i(v_1)$) $\leq$ *share*(Age) $\leq$ Max($share_i(v_1)$), where v_1 is the minimum value of 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. The process is somehow straightforward if we use an order-preserving mapping function. If so, the minimum (maximum) share value is certainly mapped onto the minimum (maximum) original value.

Some mechanisms such as client-side storage of minimum and maximum values of searchable attributes or using an auxiliary table to maintain the ordering of values [20] can also be used to tackle the problem of MIN/MAX queries in our sharing schemes.

For more complex queries with aggregation functions in their selection predicates such as 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 executed over two relations with an attribute in common. Consider two simple relations T1(ID, Dep, Salary) and T2(ID, Name, Age) and a sample join query SELECT Salary FROM T1,T2 WHERE T1.ID = T2.ID. Using the weighted partitioning method, the query is rewritten as a parameterized query SELECT Salary FROM T1,T2 WHERE Min($share(v_i)$) $\leq$ T1.ID $\leq$ Max ($share(v_i)$) AND Min($share(v_i)$) $\leq$ T2.ID $\leq$ Max($share(v_i)$) for $i = 1, 2, \ldots, m$, where m is the maximum value of the domain of ID. To have a complete result, the client submits m queries to the servers and performs a union on the received results. The process is similar for secure partitioning method except the change in query translation, which does not affect the execution of query.

Updates Updating is an important issue for data outsourcing in cloud-based environments. Our sharing schemes efficiently support updates. The execution of updates including INSERT, DELETE, and UPDATE is straightforward regardless of the partitioning method and the mapping function being used. For a DELETE, the satisfying tuples are chosen by each server same as a SELECT query and removed from the relation. To insert a new tuple, shares of attribute values in the tuple are computed and inserted into S_i ($i = 1, 2$). Searchable attributes in a tuple should be shared using our searchable schemes and other attributes can be shared using Shamir's naïve scheme. For an update query, the satisfying tuples are selected based on its condition predicate and sent to the owner. The owner generates polynomials and computes new shares to substitute them with the old ones.

The only problem with updates is that they change the frequency of values, which in turn affects our weighted and secure partitioning methods. Obviously, in the case of recurrent updates, it is not practical to repartition and redistribute data based on new frequencies. The empirical solution we adopted is to use standard distributions of attribute values and use probability of each value in the relation instead of its frequency. Using such standard distributions[1], data updates do not deteriorate the distribution of values and we get almost the same distribution in different snapshots of the system.

6 Experimental results

In this section, the feasibility of our proposal is demonstrated through a prototype implementation of our searchable sharing schemes for database outsourcing to the cloud. We implemented all our sharing schemes and examined computation, communication, and storage overheads. In the figures of this section, "Searchable scheme 1," "Searchable scheme 2," and "Searchable scheme 3" refer to simple,

[1] For instance, one can refer to the distribution of Age values stored in an organizational database.

weighted, and secure partitioning methods, respectively. The mapping function used for weighted and secure partitioning is order obfuscating. We also implemented the bucketing index-based method [17], as a highly cited method of querying encrypted data. This method is denoted by "Encryption- and bucket-based index" in the figures of this section. Moreover, Emekci et al.'s work [15], a secret sharing-based method which uses hash functions to solve the searchability problem, has been implemented. This work is denoted by "Hash-based scheme" in the figures of this section.

For our system prototype, we needed client-side software to be responsible for query transformation and result interpolation of received shares. We implemented such an application in Java with about 300 lines of code supporting all our sharing schemes introduced in Sect. 4. Clients then connect through a LAN to the DBMS servers that host data shares. Our solution is compatible with existing database services since it does not require any changes in database management system internals. We implemented our data share servers using MS SQL Server 2008 on Windows 7 system with a Core i5 2.5 GHz processor and 6 GB of memory. The performance tests were executed on a real dataset of about one million database tuples from IPUMS 2010 ACS data [26] for a relation of different attributes including ID, Age, Income, and Birthplace. We used Age as a searchable attribute in the relation having values between 0 and 90. We implemented the bucketing index-based method to search over Age values with the bucket width 5 and AES encryption algorithm with 128 bits key length. We also implemented the process of initial distribution of shares according to our sharing schemes. This is a lean Java program executed on behalf of the data owner over the original relation.

6.1 Computation overhead

The first series of experiments evaluate computational cost of query processing scenario. We issued equality, range, and aggregation queries to the system and obtained query processing times. Table 1 summarizes the design of our experiments, showing the three query categories used in the experimentation and result sizes for each category. We investigate both client and server times of query processing for the three types of queries, though the client time is more critical in data outsourcing models because the clients are supposed to have a limited computation and storage capacity. In secret sharing-based method, the client time includes query transformation over shares plus secret reconstruction. In bucketing index-based method, it includes query transformation as well, plus decryption and result pruning.

Table 1 Experimental design

Queries	Minimum result size (tuples)	Maximum result size (tuples)
Equality	1000	5000
Range	1000	10000
Aggregation	1000	5000

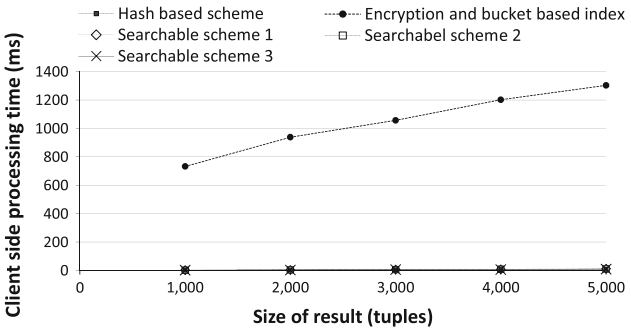

Fig. 8 Client-side processing time for equality queries

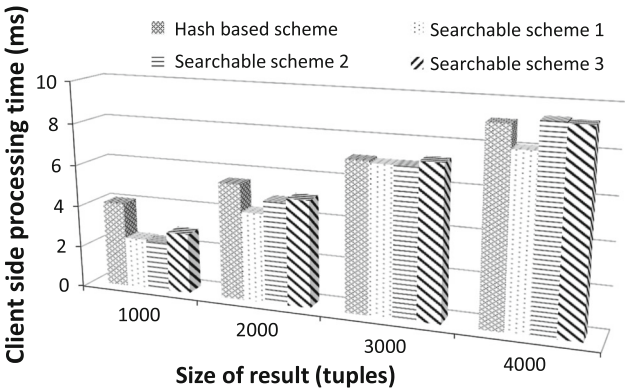

Fig. 9 Detailed view of client-side query processing time for equality queries

Figure 8 compares the client-side processing time of all our searchable schemes, Emekci et al.'s method [15], and the encryption-based approach [17] which uses bucketing to construct an index for search over encrypted values. The figure shows client-side query processing times per different result sizes, expressed as the number of tuples, when an equality query SELECT * FROM Employee WHERE Age = v is executed. As shown in the figure, all secret sharing-based methods completely outperform the bucketing method in terms of client computation overhead. This is due to the considerable number of false hits in the bucketing methods, which needs client-side pruning. Figure 9 is excluded the bucketing method to show a more detailed comparison of Emekci et al.'s method with our three searchable schemes. The figure suggests that there is little difference of client-side computation cost between our sharing schemes and the scheme in [15]. The difference is due to the query transformation phase of execution. Figure 10 shows the client-side

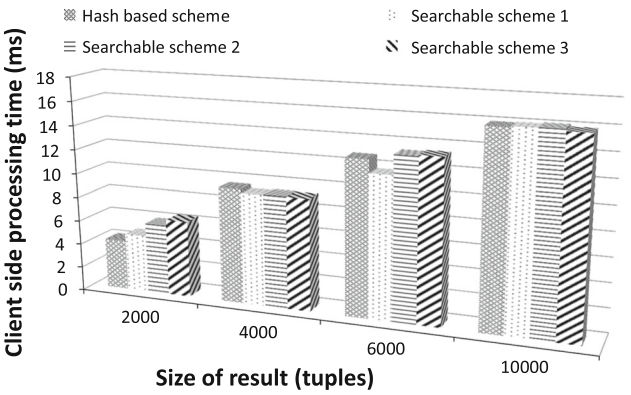

Fig. 10 Detailed view of client-side query processing time for range queries

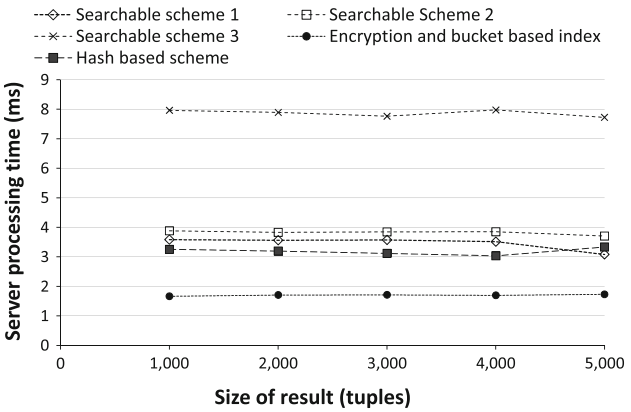

Fig. 11 Server-side query processing time for equality queries

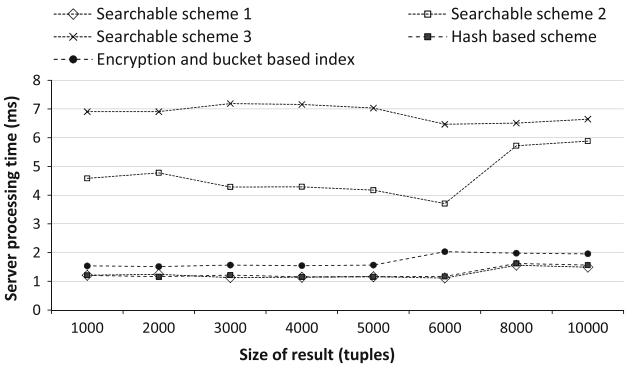

Fig. 12 Server-side query processing time for range queries

execution time of our searchable schemes and Emekci et al.'s method for a range query SELECT * FROM Employee WHERE $v_1 <$ Age $< v_2$. It can be observed in the figure that the query processing times of the methods get closer with large result sets as the secrets interpolation time overwhelms the query transportation time of query processing.

Figures 11 and 12 compare server-side processing time of the methods for the equality and the range query, respectively. The figures suggest no obvious differences in server-side query processing time when the size of result goes up. According to the figures, our weighted and secure partitioning methods due to their order-obfuscating mapping and searching for multiple ranges per a single value (for secure partitioning method) impose more computation costs on the server. However, the extra cost is limited to few milliseconds while our searchable schemes substantially enhance the security against statistical analysis. Moreover, in a real cloud-based data outsourcing scenario, this time is negligible compared with the client-server communication latencies. There is the same deduction from Fig. 13, which compares the server-side query processing times of the secret sharing-based methods for a typical aggregation query SELECT SUM(Income) FROM Employee WHERE $v_1 <$ Age $< v_2$. Also, as shown in Fig. 14, there is less than a millisecond (about 0.2 ms in the figure) difference in client-side processing times of executing the aggregation query between our searchable schemes and Emekci et al.'s method. The little difference is resulted from the process of query transformation. The bucketing index-based method [17] has excluded from Figs. 13 and 14 because the method does not support server-side execution of aggregation queries.

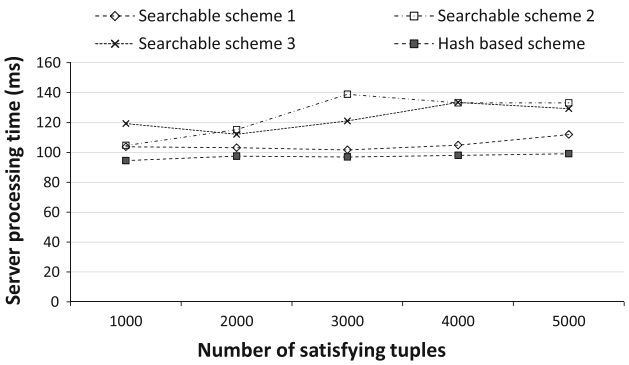

Fig. 13 Server-side query processing time for aggregation queries

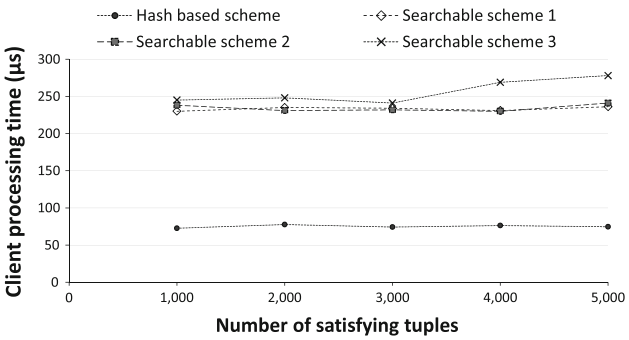

Fig. 14 Client-side query processing time for aggregation queries

Overall, while for our weighted and secure partitioning methods, clients spend just a couple of milliseconds more than that for the hash-based scheme [15], a significant security improvement is achieved. Our schemes slightly increase

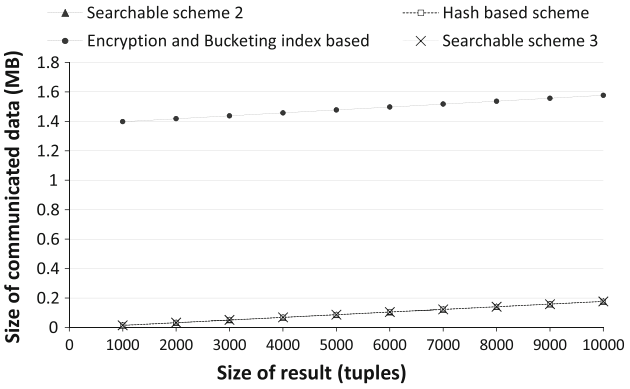

Fig. 15 Communication cost

server-side query execution time whose amount is negligible compared with network latencies in real cloud-based data outsourcing scenarios.

6.2 Communication overhead

Communication cost in our sharing schemes for query processing consists of sending the transformed query to servers and receiving the results from them. The comparison of communication cost is given in Fig. 15. The figure shows that our sharing schemes have the same performance in terms of communication cost with Emekci et al.'s approach [15], as none of them generate false hits in servers' responses. It is worth to mention that this experimentation is performed on a relation with a small tuple size. It is important in communication cost for the bucketing index-based method [17] in which greater tuple size leads to more communication cost.

6.3 Storage overhead

Client-side storage in our schemes requires storing the distribution vector X in addition to the mapping function. Both the distribution vector and the mapping function need to be stored securely, out of the access of unauthorized users. For weighted and secure partitioning methods in which either the number of partitions (in secure partitioning) or their width (in weighted partitioning) varies, the client-side storage cost is of order $O(|D_v|)$, where D_v is the domain of the searchable attribute. This is reasonable for a database size of order $O(|N|)$, where N is the number of tuples. The cost is acceptable compared with the database size especially when the database size goes up with millions of records. Considering Age as the searchable attribute, in our current implementation, about 4KB and 9KB are enough to store partitioning information for weighted and secure partitioning methods, respectively. Figure 16 shows the ratio between client-side storage size and the one of the original relations. The storage

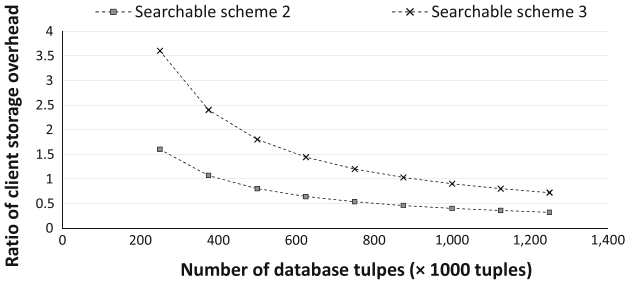

Fig. 16 Client-side storage cost compared to the database size

cost in secure partitioning method is more than weighted partitioning because each value is assigned multiple partitions that need to be stored at client side.

Client-side storage cost is manageable by using a suitable mapping function with respect to the domain size of attribute and the distribution of attribute values in the relation. For example, to reduce the storage cost associated with the large domain of attributes such as Income in a typical Employee relation, we can assign a partition of shares to a range of values instead of to a single value. This method substantially decreases the client-side storage cost, though it causes a limited number of false hits in server-side query processing. For ID values, such as employees' identifiers in a relation, the simple partitioning method can be used since the frequency of values in the relation is nearly equal.

Server-side storage cost is of less importance in the outsourcing scenarios as cloud service providers are supposed to hold considerable storage and computation resources. In our schemes, the imposed server-side storage overhead is due to having share size bigger than the secret size for searchable attribute 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. Nevertheless, 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.

7 Summary and future work

Confidentiality of outsourced data is still a major obstacle toward the adoption of the "Database as a Service" model in cloud computing environments. In this paper, we propose to use secret sharing in order to preserve confidentiality of outsourced data. Our client-aware partitioning approach subdivides the domain of shares so that clients can efficiently search within distributed shares. Also, it prevents untrusted data servers from inferring the original attribute values from distributed shares. We introduced three sharing schemes (each corresponding to a partitioning method) with different levels of security, client-side storage overhead, and query execution efficiency. Among our partitioning methods,

secure partitioning provides data confidentiality for honest-but-curious servers, powered by a priori knowledge of original data distribution. We proved that in our secure partitioning method, the probability of finding the correct association between a share and its corresponding data value is almost equal to that of a random guess. There are three reasons for such an achievement:

1. The distribution of an original dataset is perturbated in the corresponding sets of shares. With large domains of shares, the resulting distribution can be closed to a uniform distribution, which minimizes the servers' inference when observing shares.
2. The ordering relation of values is obfuscated among their corresponding sets of shares.
3. Equality and range queries are indistinguishable from servers' viewpoint. This indistinguishability prevents servers from aligning shares ordering.[2]

Our approach, supporting server-side indexing, allows server-side execution of different queries with acceptable overheads. It does not require any modification of existing database internals and is immediately applicable to existing cloud database service offerings.

Our proposal can be readily extended to improve storage cost at both server and client sides by defining new partitioning methods and mapping functions. As future work, we plan to investigate on the size of shares domain to decrease server-side storage cost in addition to achieving desired level of security. Another extension point to our work is to address the availability and fault tolerance aspects and determine the appropriate values of k and n in a (k, n) sharing scheme with respect to a desired level of availability and fault tolerance.

References

1. Adam, N.R., Worthmann, J.C.: Security control methods for statistical databases: a comparative study. ACM Comput. Surv. **21**(4), 515–556 (1989)
2. Agrawal, D., Abbadi, A.E., Emekci, F., Metwally, A., Wang, S.: Secure data management service on cloud computing infrastructures. In: Agrawal, D., Candan, K.S., Li, W. (eds.) New Frontiers in Information and Software as Services. Lecture Notes in Business Information Processing, vol. 74, pp. 57–80. Springer, Berlin (2011)
3. Agrawal, D., El Abbadi, A., Emekci, F., Metwally, A.: Database management as a service: challenges and opportunities. In: IEEE 25th International Conference on Data Engineering, 2009. ICDE'09, pp. 1709–1716 (2009)
4. Agrawal, R., Kiernan, J., Srikant Ramakrishnan, Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 563–574. ACM (2004)
5. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Proceedings of the 4th Conference on Theory of Cryptography, pp. 535–554. Springer, Berlin (2007)
6. Brinkman, R., Doumen, J., Jonker, W.: Using Secret Sharing for Searching in Encrypted Data. Secure Data Management. Lecture Notes in Computer Science, vol. 3178, pp. 18–27. Springer, Berlin Heidelberg (2004)
7. Ceselli, A., Damiani, E., di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM Trans. Inf. Syst. Secur. (TISSEC) **8**(1), 119–152 (2005)
8. Chow, S.S.M., Lee, J.-H., Subramanian, L.: Two-party computation model for privacy-preserving queries over distributed databases. In: Proceedings of the Network and Distributed System Security Symposium, (NDSS), The Internet Society (2009)
9. Ciriani, V., Capitani, De: Combining fragmentation and encryption to protect privacy in data storage. ACM Trans. Inf. Syst. Secur. (TISSEC) **13**(3), 1–33 (2010)
10. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation design for efficient query execution over sensitive distributed databases. In: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems, ICDCS '09, pp. 32–39. IEEE Computer Society (2009)
11. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 93–102 (2003)
12. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: Computing range queries on obfuscated data. In: Proceedings of the Information Processing and Management of Uncertainty in Knowledge-Based Systems, pp. 1333–1340. IEEE Computer Society (2004)
13. Dautrich, J.L., Ravishanka, C.V.: Security limitations of using secret sharing for data outsourcing. In: Proceedings of DBSec 2012, Lecture Notes in Computer Science, pp. 145–160. Springer, Berlin (2012)
14. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proceedings of IEEE ICDCS 2011, pp. 710–719. IEEE Computer Society (2011)
15. Emekci, F., Methwally, A., Agrawal, D., Abbadi, A.E.: Dividing secrets to secure data outsourcing. Inf. Sci. **263**, 198–210 (2014)
16. Ferretti, L., Colajanni, M., Marchetti, M.: Distributed, concurrent, and independent access to encrypted cloud databases. IEEE Trans. Parallel Distrib. Syst. **25**(2), 437–446 (2014)
17. Hacigümüs, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database service provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 216–227. ACM (2002)
18. Hadavi, M.A., Damiani, E., Jalili, R., Cimato, S., Ganjei, Z.: AS5: A Secure Searchable Secret Sharing Scheme for Privacy Preserving Database Outsourcing. Data Privacy Management and Autonomous Spontaneous Security. Lecture Notes in Computer Science, vol. 7731, pp. 201–216. Springer, Berlin Heidelberg (2013)
19. Hadavi, M.A., Jalili, R.: Secure data outsourcing based on threshold secret sharing: Towards a more practical solution. In: Proceedings of VLDB PhD Workshop, pp. 54–59. VLDB Endowment (2010)
20. Hadavi, M.A., Noferesti, M., Jalili, R., Damiani, E.: Database as a service: towards a unified solution for security requirements. In: Proceedings of 36th IEEE COMPSACW, pp. 415–420. IEEE Computer Society (2012)

[2] Shares ordering has been discussed in [13] as part of the attack scenario on using secret sharing for outsourcing scenario.

21. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proceedings of 30th International Conference on Very Large Database, pp. 720–731. VLDB Endowment (2004)
22. Kerschbaum, F., Schropfer, A., Zilli, A., Pibernik, R., Catrina, O., Hoogh, Sd, Schoenmakers, B., Cimato, S., Damiani, E.: Secure collaborative supply-chain management. Computer **44**(9), 38–43 (2011)
23. Laur, S., Talviste, R., Willemson, J.: From oblivious AES to efficient and secure database join in the multiparty setting. Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 7954, pp. 84–101. Springer, Berlin Heidelberg (2013)
24. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of Computer and Communication Security Workshops 2011, pp. 113–124. ACM (2011)
25. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: processing queries on an encrypted database. Commun. ACM **55**(9), 103–111 (2012)
26. Ruggles, S., Alexander, J.T., Genadek, K., Goeken, R., Schroeder, M.B., Sobek, M.: Integrated Public Use Microdata Series: Version 5.0 [Machine-readable database]. Tech. rep., University of Minnesota, Minneapolis: University of Minnesota (2010)
27. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
28. Steele, A., Frikken, K.B.: An index structure for private data outsourcing. In: Proceedings of DBSec 2011, pp. 247–254. Springer, Berlin (2011)
29. Taheri Soodejani, A., Hadavi, M.A., Jalili, R.: K-Anonymity-based horizontal fragmentation to preserve privacy in data outsourcing. In: Proceedings of the 26th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy, DBSec'12, pp. 263–273. Springer, Berlin (2012)
30. Tian, X., Sha, C., Wang, X., Zhou, A.: Privacy preserving query processing on secret share based data storage. Database Systems for Advanced Applications. Lecture Notes in Computer Science, vol. 6587, pp. 108–122. Springer, Berlin Heidelberg (2011)
31. Wang, S., Agrawal, D., Abbadi, A.: A comprehensive framework for secure query processing on relational data in the cloud. Secure Data Management. Lecture Notes in Computer Science, vol. 6933, pp. 52–69. Springer, Berlin Heidelberg (2011)
32. Wang, S., Agrawal, D., Abbadi, A.E.: Towards practical private processing of database queries over public data with homomorphic encryption. Tech. rep., 2011–06, Department of Computer Science, University of California at Santa Barbara (2011). https://p2p.cs.ucsb.edu/research/tech_reports/reports/2011-06
33. Wang, W., Hu, Y., Chen, L., Huang, X., Sunar, B.: Accelerating fully homomorphic encryption using GPU. In: 2012 IEEE Conference on High Performance Extreme Computing (HPEC), pp. 1–5 (2012)
34. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. Proc. IEEE INFOCOM **2010**, 1–9 (2010)
35. Zhang, Z., Plantard, T., Susilo, W.: Reaction attack on outsourced computing with fully homomorphic encryption schemes. In: Proceedings of ICISC 2011, pp. 419–436. Springer, Berlin (2011)