World Wide Web (2018) 21:915–938
DOI 10.1007/s11280-017-0491-8

An effective approach for the protection of privacy text data in the CloudDB

Zongda Wu[1,2] · Guandong Xu[3] · Chenglang Lu[1] · Enhong Chen[2] · Fang Jiang[1] · Guiling Li[4]

Received: 23 April 2017 / Revised: 3 August 2017 / Accepted: 10 August 2017 /
Published online: 18 August 2017

Abstract Due to the advantages of pay-on-demand, expand-on-demand and high availability, cloud databases (CloudDB) have been widely used in information systems. However, since a CloudDB is distributed on an untrusted cloud side, it is an important problem how to effectively protect massive private information in the CloudDB. Although traditional security strategies (such as identity authentication and access control) can prevent illegal users from accessing unauthorized data, they cannot prevent internal users at the cloud side from accessing and exposing personal privacy information. In this paper, we propose a client-based approach to protect personal privacy in a CloudDB. In the approach, privacy data before being stored into the cloud side, would be encrypted using a traditional encryption algorithm, so as to ensure the security of privacy data. To execute various kinds of query

✉ Guiling Li
guiling@cug.edu.cn

Zongda Wu
zongda1983@163.com

Guandong Xu
guandong.xu@uts.edu.au

Chenglang Lu
playnet107@163.com

Enhong Chen
enhc@ustc.edu.cn

Fang Jiang
fang.jiang@wzu.edu.cn

1 Oujiang College, Wenzhou University, Wenzhou, Zhejiang, China

2 School of Computer Science, University of Science and Technology of China, Hefei, Anhui, China

3 Faculty of Engineering and IT, University of Technology, Sydney, Australia

4 School of Computer Science, China University of Geosciences, Wuhan, China

operations over the encrypted data efficiently, the encrypted data would be also augmented with additional feature index, so that as much of each query operation as possible can be processed on the cloud side without the need to decrypt the data. To this end, we explore how the feature index of privacy data is constructed, and how a query operation over privacy data is transformed into a new query operation over the index data so that it can be executed on the cloud side correctly. The effectiveness of the approach is demonstrated by theoretical analysis and experimental evaluation. The results show that the approach has good performance in terms of security, usability and efficiency, thus effective to protect personal privacy in the CloudDB.

Keywords CloudDB · Information system · Privacy protection · Feature index

1 Introduction

A cloud database (CloudDB) refers to a database deployed on an Internet-based virtual computing environment, which allows users to store, modify and retrieve data anywhere in the world, as long as they have access to the Internet [20]. Due to the advantages of pay-on-demand, expand-on-demand and high availability, CloudDB has been widely used in information systems [10]. However, since a CloudDB is distributed on the cloud side instead of a local server, it is important how to effectively protect massive information about personal privacy (such as phone number and personal name) stored in the CloudDB [11]. To ensure the security of personal privacy information, many strategies have been used in information systems, such as identity authentication, and authorization and access control [3, 30, 32]. These strategies can prevent illegal users from accessing unauthorized data, consequently, ensuring the security of personal information to a large extent. However, almost all the strategies are targeted only for external illegal users of an information system, and they cannot prevent internal users (such as administrators) at the cloud side from accessing personal information stored in the CloudDB.

A general framework of a CloudDB information system is shown in Figure 1, where (1) **external users**, who generally work at client sides, store their data into the CloudDB and use data services supplied by the CloudDB; and (2) **internal users**, who work at the cloud side, manage the CloudDB and a large amount of external users' personal data stored in the CloudDB. In a CloudDB, the client sides are considered to be trusted because existing security strategies used by database systems can prevent users from accessing unauthorized data. However, the cloud side is considered to be untrusted [31]. For example, an administrator or a attacker who has broken into the cloud side can access private data stored in the CloudDB easily. In other words, it is possible for internal users at the cloud side to access

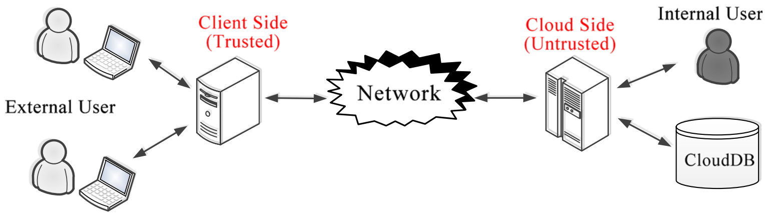


Figure 1 A general framework of a CloudDB information system

and expose the personal data in the CloudDB, driven by economic interests, thereby leading to the disclosure of personal information.

1.1 Motivation

It has been reported by iResearch[1] that the frequent occurrence of disclosure of personal privacy is making people become "transparent", and more than half of the disclosure events are caused by the internal users of a network system. Therefore, it is very important to supply an effective approach to ensure the security of personal privacy in a CloudDB, which should be able to prevent the disclosure of personal information caused by internal users at the cloud side, not just by external users at the client sides. To protect personal privacy in a CloudDB, a straightforward way is to encrypt personal data, so that even if the encrypted data are exposed, they are difficult to be decrypted [22]. However, in an information system, generally, there are a large number of database query operations, which are relevant to personal data (i.e., defined over personal data). Once the private data are encrypted using a traditional encryption algorithm (e.g., those in [2, 17]), most of the database query operations (such as text similarity queries) will no longer be able to be executed correctly over the encrypted data in the CloudDB.

To solve the above problem on querying encrypted data, we can transmit the encrypted data (which may be a whole table) from the cloud side, decrypt the encrypted data and then execute the query operations over the decrypted data. However, as the cost of transmitting and decrypting an encrypted table is expensive, such a way (i.e., decrypting before querying) will greatly reduce the execution efficiency of database query operations, resulting in its inapplicability to a CloudDB. Although the homomorphic encryption techniques [17] can maintain the original order and comparability of the encrypted data so that a number of database query operations can be executed correctly over the encrypted data, they are generally of weak security, e.g., the encrypted data are easy to be decrypted by statistical attack, as pointed out in [13, 33]. Although there are a number of studies on data encryption [16], most of them require to first decrypt the encrypted data and then execute queries over the decrypted data, consequently, making them difficult to satisfy the efficiency requirement of a CloudDB. Although there are a small number of data encryption algorithms (see the related work section for detail) that allow users to query encrypted data directly without the need to decrypt data, they generally have the disadvantages of weak security or inability to fully support query operations, thereby, making them difficult to solve the problem on querying encrypted personal data in a CloudDB.

1.2 Contribution

In this paper, we propose a client-based approach to protect personal privacy in a CloudDB. In the approach, before being submitted to the cloud side, personal data have to be encrypted on a trusted client side using a traditional encryption algorithm, so as to ensure the security of personal data on the untrusted cloud side. Meanwhile, to execute various kinds of query operations over the encrypted data efficiently, the approach generates additional feature information (called feature index) for the encrypted data, which allows a certain amount of query processing to occur on the cloud side without the need to decrypt the data. Thus, the approach mainly explores how the feature index of personal data is constructed, and how

[1]iResearch, a well-known consulting company in China - http://report.iresearch.cn/.

each query operation over personal data is transformed into a new query operation over the feature index so that it can be executed correctly on the cloud side. Specifically, the contributions of this paper are threefold.

(1) We present a scheme to generate the feature index for personal data, which has not only good security (i.e., it is difficult to infer the original personal data from the feature index), but also good usability (i.e., it can support various kinds of database query operations).
(2) We present a scheme to transform each user query relevant to personal data into a cloud-side query relevant to the feature index, so that the new query can be executed on the cloud side correctly, consequently, improving the execution efficiency of database query operations.
(3) We demonstrate the effectiveness of our approach by theoretical analysis and experimental evaluation. The results show that the approach has good performance in terms of security, usability and efficiency, thus it is applicable to effectively protect personal privacy in a CloudDB.

The rest of this paper is organized as follows. Section 2 surveys related work. Section 3 presents the system model and the problem studied in this paper, i.e., formally describing what requirements should be satisfied so as to protect personal privacy effectively in a CloudDB. Section 4 presents a scheme to generate the feature index for personal data, and analyzes the security of the scheme. Section 5 presents a scheme to map each query over personal data into a cloud side query over the feature index, and analyzes the usability of the scheme. Section 6 presents the experimental evaluation to demonstrate the efficiency of our approach. Finally, we conclude this paper in Section 7.

2 Related work

In this section, we briefly describe some research related to querying encrypted data in outsourced databases. In [13], Hacigumus et al. first proposed the bucket partitioning idea for querying encrypted data in the database-as-service model. The basic idea is to divide the attribute domains into multiple buckets and then map bucket identifiers to random numbers, thereby, protecting the security of sensitive data. Moreover, this makes that much of a query operation over encrypted data can be processed at the database service provider, thereby, improving query performance. Later, in [14], the authors proposed to use the homomorphism encryption techniques to enhance their approach, so as to support aggregation queries over encrypted data, and in [15], the authors further discussed an optimization technique for their approach, i.e., how to use multiple communications between the server and the client to decrease the workload of the client. In order to better support range queries over encrypted data, Hore et al. [7] explored an optimal approach to partitioning data domain, thereby, improving the precision of range queries. The work presented by Hacigumus et al. is significant, which presented a basic framework to ensure data security in the database-as-service model. However, the work did not analyze the security formally for the approach. Besides, the work is valid only for numerical data without considering text data. Since personal privacy data in an information system are generally of text type, it is not suitable to apply the above approach to protect personal privacy in a CloudDB.

Recently, many studies on the data security in cloud databases have been presented. Li et al. [19] discussed the problem about privacy preserving range query processing on clouds, and presented a fast range query processing scheme by organizing indexing elements in a complete binary tree. Wai et al. [28] addressed security issues in a cloud database system, and proposed a secure query processing scheme on relational tables and a set of elementary operators on encrypted data, which allows a wide range of database queries to be processed by the server on encrypted data. Chen et al. [9] proposed an efficient privacy and integrity preserving scheme for multi-dimensional range queries over cloud computing. Luca et al. [23] proposed an architecture that integrates cloud database services with data confidentiality and the possibility of executing concurrent operations on encrypted data. This is the first solution supporting geographically distributed clients to connect directly to an encrypted cloud database, and to execute concurrent and independent operations. Recent work [6] proposed a general framework for boolean queries of disjunctive normal form queries on encrypted data. Although all the approaches are proposed for cloud databases, most of them are targeted for building a secure cloud database. As mentioned above, they are not proposed for a CloudDB, so based on them, we cannot build an effective CloudDB that can support a variety of database query operations over privacy data (such as text similarity queries and range queries).

Some researchers also proposed to split sensitive data among multiple servers to ensure data security. In [12], a scheme for vertical partitioning of relations among multiple untrusted servers was employed, whose privacy goal is to prevent access of a subset of attributes by any single server. Aggarwal et al. [1] also used a similar vertical partitioning scheme which has the same privacy goal but different partitioning and optimization algorithms. Wang et al. [24] used a salted version of IDA scheme to split encrypted tuple data among multiple servers. In [26], a novel l-diversity privacy model was proposed for privacy preservation in the release of data for mining purposes. Recently, some researchers also proposed to use a hardware approach to ensure data security, such as TrustedDB [25], MONOMI [27] and Cipherbase [4]. The advantages of the hardware approach are that it can provide strong security protection, and it does not limit query expressiveness. However, the hardware approach needs to reconstruct the system structure of a CloudDB. In addition, there are also other related encryption techniques for spatial data [8, 21, 34].

From the above, we see that most of existing approaches to data security protection in outsourced databases focus on constructing a secure framework, without fully taking into consideration the structure and type of sensitive data. As a result, if we apply these approaches immediately into a CloudDB, it is difficult to support a variety of similarity queries and range queries over encrypted privacy data. Actually, aiming at the problem of querying encrypted textual data in a database, there are some related studies. Wang et al. [29] proposed to turn a character string into characteristic values, so as to support similarity queries. This approach can reduce the scope of data decryption, and thus improve query performance. However, the approach cannot well solve the similarity queries in the form of "LIKE '%s'" and "LIKE 's%'", and cannot support range queries. Besides, owing to using only one characteristic function, the approach is difficult to withstand statistics attack or inference attack. By analyzing the traditional order-preserving encryption approach to numerical data, a fuzzy matching encryption approach aiming at character strings was proposed in [18]. In this approach, a character string is first transformed to numerical values, and an order-preserving encryption technique in [14] for numerical data, is then used to encrypt the transformed numerical values. To solve the problem of not supporting range queries for the approach in [29], Wu et al. [33] defined a structure called n-phase reachability matrix for a character string and used it as the characteristic index values, and then presented split a database query into its server-side representation and client-side representation for partitioning the computation of a query across the client and the server and thus improving query performance. However, it is space-consuming to store a complete reachability matrix.

3 Problem statement

3.1 System model

The system model used by our approach is presented in Figure 2. As shown in Figure 2, the system model consists of an index generator, a query translator and a query executor, whose processing flows can be briefly described as follows.

(1) Before being submitted to the cloud side, privacy data u has to be handled by the **index generator**, so as to generate the ciphertext $\boldsymbol{E}(u)$ and the feature index $\boldsymbol{X}(u)$, where $\boldsymbol{E}$ and $\boldsymbol{X}$ denote an encryption function and an index function, respectively.

(2) Each query operation q_u relevant to privacy data, before being submitted to the cloud side, has to be transformed into a new cloud-side query operation q_x, which is defined over the feature index so can be executed by the encrypted CloudDB correctly. This process is completed by the **query translator**.

(3) The **query executor** decrypts the temporal result $R(q_x)$ returned from the cloud side, which is obtained by executing the cloud-side query operation q_x over the CloudDB; and then executes the user query operation q_u over the decrypted data $\boldsymbol{D}(R(q_x))$ (where $\boldsymbol{D}$ denotes a decryption function), thereby, obtaining the accurate result $R(q_u)$ of q_u.

(4) Meanwhile, each client side of a CloudDB also maintains an internal **metadata** structure that is used to store all kinds of parameter information for the above components.

It can be seen that the system model is located on a client side (i.e., it is client-based), but it is transparent to the client side, i.e., it requires no change to existing softwares on the client side. In the system model, a cloud side is deemed untrusted, i.e., the adversary is located on the cloud side, who has full access to not only the entire CloudDB, but also all the database query operations from client sides. Thus, the adversary is deemed powerful, who can master a large quantity of plaintext, ciphertext and feature index information.

3.2 Problem analysis

In the system model, the encryption function $\boldsymbol{E}$ is developed based on an existing data encryption algorithm (e.g., AES [5]), so it is almost impossible for the adversary to infer the plaintext u from the encrypted data $\boldsymbol{E}(u)$, i.e., the security of encrypted data can be well protected in the untrusted cloud side. Thus, this paper non longer pays attention to the security of the encrypted data. From Figure 2, we know that the feature index is the key to protecting personal privacy effectively. In general, a good feature index scheme should satisfy the following requirements.

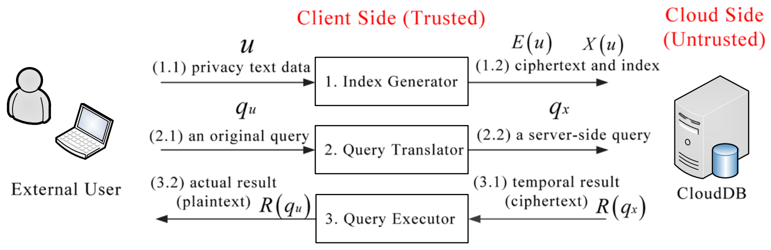


Figure 2 The system model used in our approach, where the *arrows* denote data processing flows

(1) Good **security**. On the cloud side, the indexes are visible to the adversary, thus the feature indexes should ensure their own security, i.e., it should be difficult for the adversary to infer the plaintext u from the index $\boldsymbol{X}(u)$.
(2) Good **usability**. Based on the feature index, each familiar user query q_u over privacy data should be able to be transformed into a cloud-side query q_x that can be executed over the encrypted CloudDB. It is required that the result returned by q_x should be a superset of the accurate result of q_u, i.e., $R(q_u) \subseteq \boldsymbol{D}(R(q_x))$.
(3) Good **efficiency**. On the cloud side, the query q_x should be able to filter as many of the non-target tuples (i.e., which do not satisfy q_u) as possible, making the temporal result $R(q_x)$ as close to the accurate result $R(q_u)$ as possible, so as to lighten the computation on a client side, and thus improve the execution efficiency of q_u.

However, it is difficult to meet the above requirements simultaneously. On the one hand, good security generally requires the feature index to describe as little feature information on privacy data as possible, so as to make it difficult for the adversary to obtain the original plaintext based on the index. On the other hand, good usability and efficiency require that as much feature information as possible on privacy data can be reflected by the index. Thus, good feature index should be a reasonable compromise among security, usability and efficiency.

3.3 Problem definition

To simplify the presentation, below we use a symbol Θ to represent a privacy protection approach that runs on the system model in Section 3.1, and use $\boldsymbol{X}_\Theta$ to represent an index function used by the approach Θ. Based on the analysis given in Section 3.2, we formulate the requirements that the approach Θ has to satisfy so as to effectively protect personal privacy.

Let $\mathcal{U}$ denote the domain of privacy data, and $\mathcal{X}_\Theta$ the domain of index data generated by the approach Θ. Then, we have $\mathcal{X}_\Theta = \{x \mid u \in \mathcal{U} \wedge x = \boldsymbol{X}_\Theta(u)\}$. As we mentioned above, the adversary can master a large quantity of plaintext and the corresponding feature index. Thus, (1) the **prior knowledge** that the adversary has mastered can be defined as a set of two-tuples from privacy data u $(u \in \mathcal{U})$ to index data $\boldsymbol{X}_\Theta(u)$; (2) the limit k_Θ^* of prior knowledge of the adversary can be represented as: $k_\Theta^* = \{(u, x) \mid u \in \mathcal{U} \wedge x \in \mathcal{X}_\Theta \wedge x = \boldsymbol{X}_\Theta(u)\}$; and (3) the domain $\mathcal{K}_\Theta$ of prior knowledge can be represented as: $\mathcal{K}_\Theta = 2^{k_\Theta^*}$, i.e., the prior knowledge that an adversary has mastered is a subset of k_Θ^*. From experience, we know that, (1) given any index data x $(x \in \mathcal{X}_\Theta)$, the **probability** that the adversary infers the plaintext from the index data x mainly depends on the prior knowledge k_Θ $(k_\Theta \in \mathcal{K}_\Theta)$ that an adversary has mastered, so we denote it as: $Pr(k_\Theta)$ $(0 < Pr(k_\Theta) \leq 1)$; and (2) $Pr(k_\Theta) \propto |k_\Theta|$, i.e., the probability of inferring the plaintext is proportional to the amount of the prior knowledge mastered by the adversary. Now, the security of the approach Θ can be defined as follows.

Definition 1 Given a threshold λ $(0 < \lambda \leq 1)$, an approach Θ meets λ-**security**, if and only if the probability that an adversary infers the plaintext from any index data established by Θ is always less than λ, regardless of the prior knowledge mastered by the adversary. Formally, an approach Θ meets λ-**security**, if and only if $\forall k_\Theta$ $(k_\Theta \in \mathcal{K}_\Theta \rightarrow Pr(k_\Theta) \leq \lambda)$, i.e., $Pr(k_\Theta^*) \leq \lambda$ (since $Pr(k_\Theta) \propto |k_\Theta|$).

Let $\mathcal{Q}_u$ denote the domain of user query operations relevant to privacy data. As mentioned in Section 3.2, good usability requires that each query q_u $(q_u \in \mathcal{Q}_u)$ can be transformed into a cloud-side query q_x, so that $R(q_u) \subseteq \boldsymbol{D}(R(q_x))$. However, it can be seen that $\mathcal{Q}_u$ is an infinite set. Thus, we first define a core set of user query operations, and then define the usability of the approach Θ.

Definition 2 $\mathcal{Q}_u^*$ is a **core set** of user queries relevant to privacy data if it meets: (1) $\mathcal{Q}_u^* \subseteq \mathcal{Q}_u$; (2) $\forall q_1 \exists q_2 (q_1 \in \mathcal{Q}_u \wedge q_2 \in \mathcal{Q}_u^* \rightarrow R(q_1) = R(q_2))$; and (3) $\forall q_1 \forall q_2 (q_1 \in \mathcal{Q}_u^* \wedge q_2 \in \mathcal{Q}_u^* \wedge q_1 \neq q_2 \rightarrow R(q_1) \neq R(q_2))$.

Definition 3 An approach Θ meets **usability**, if and only if any user query operation q_u $(q_u \in \mathcal{Q}_u^*)$ can be transformed into a cloud-side query operation q_x, which is defined over the feature index $\mathcal{X}_\Theta$, and the actual result $R(q_u)$ of q_u is contained in the temporal result $R(q_x)$ returned by q_x, i.e., $R(q_u) \subseteq \boldsymbol{D}(R(q_x))$.

Let $T(q_u)$ denote all the tuples in the table related to a query operation q_u $(q_u \in \mathcal{Q}_u)$ (i.e., the table presented in the FROM clause of q_u). Then, $|T(q_u) - R(q_u)|$ denotes the number of non-target tuples of q_u; and $|T(q_u) - \boldsymbol{D}(R(q_x))|$ (where q_x is a cloud-side query transformed from q_u by the approach Θ) denotes the number of non-target tuples filtered out on the cloud side by q_x. As mentioned above, good efficiency requires that as many of non-target tuples as possible can be filtered out by q_x, i.e., $|T(q_u) - \boldsymbol{D}(R(q_x))|$ is as close as possible to $|T(q_u) - R(q_u)|$. Below, we first define filtering rate, and then define the efficiency of the approach Θ.

Definition 4 For any user query operation q_u $(q_u \in \mathcal{Q}_u^*)$, we use q_x to denote its cloud-side query operation generated by the approach Θ. Then, the **filtering rate** $Fr_\Theta(q_u)$ of Θ to non-target tuples of q_u is defined as: $Fr_\Theta(q_u) = \frac{|T(q_u) - \boldsymbol{D}(R(q_x))|}{|T(q_u) - R(q_u)|}$.

Definition 5 Given a threshold μ $(0 \leq \mu \leq 1)$, an approach Θ meets μ**-efficiency** if and only if the mathematical expectation $\sum_{q_u \in \mathcal{Q}_u^*} Pr(q_u) \cdot Fr_\Theta(q_u) \geq \mu$, wherein, $Pr(q_u)$ denotes the probability of a query q_u $(q_u \in \mathcal{Q}_u^*)$ issued by external users, and $\sum_{q_u \in \mathcal{Q}_u^*} Pr(q_u) = 1$.

Now, based on Definitions 1, 3 and 5, we define the requirements that the approach Θ has to satisfy so as to protect personal privacy effectively.

Definition 6 Given two thresholds λ $(0 < \lambda \leq 1)$ and μ $(0 \leq \mu < 1)$, if an approach Θ meets λ-security, usability and μ-efficiency, then Θ is **effective** to protect personal privacy in a CloudDB.

4 Privacy protection scheme

In this section, before introducing the approach of encrypting and indexing privacy data, we first show how the encrypted data and their feature indexes are stored into the CloudDB. Note that in a CloudDB, privacy data such as identification number, phone number, personal name and bank account are generally stored as a text field (i.e., whose field type is CHAR or VARCHAR), so in our work privacy data of any type are treated as text uniformly (i.e., we take no account of the privacy data of numeric type, which is out of the scope of this paper). We suppose that there exists one relational table $R(A_1, A_2, ..., A_r, ...)$ in the CloudDB, where A_r is a field used to store privacy data thus needs to be encrypted (A_r is called a

private field. To simplify presentation, we assume that there is only one private field A_r in the table R. Then, in the encrypted CloudDB, we will store an **encrypted relational table** $R^E\left(A^E, A_1, A_2, ..., A_r^X, ...\right)$ instead of R, wherein,

(1) The field A^E (called an **encrypted field**) stores an encrypted binary string (i.e., ciphertext) that corresponds to a tuple in the table R (we will explain how the encrypted field A^E is constructed in Section 4.2).
(2) The field A_r^X (called an **index field**) corresponds to the feature index for the private field A_r, and the type of A_r^X is identical to that of A_r, i.e., whose type is also CHAR or VARCHAR.
(3) The remaining fields in the encrypted table R^E are all consistent with those in the original table R.

Below, we study a privacy protection scheme used in our approach, i.e., study how privacy data are encrypted and indexed so as to ensure the security of privacy data in the untrusted cloud side. Specifically, we first show how to construct a feature index function $\boldsymbol{X}$ for privacy data. Second, we show how privacy data are encrypted and indexed, and then stored into the encrypted CloudDB. Finally, we analyze the security of the privacy protection scheme.

4.1 Feature index function

For any value u in the domain of the private field A_r of R, this subsection explains how it is mapped to the feature index value $\boldsymbol{X}(u)$, so that it can be stored into the index field A_r^X of R^E, i.e., how the feature index function $\boldsymbol{X}$ is constructed. For the private field A_r of R, suppose that each value in its domain contains no more than n ($n \in \mathbb{N}$) characters. Then, the private field A_r consists of n **character units**. Below, we use P_i ($i = 1, 2, ..., n$) to denote each character unit of A_r, and $\boldsymbol{dom}(P_i)$ to denote the domain of values of P_i. In order to construct a feature index function over the private field A_r, we need the following three steps: (1) automatically assigning a number $\boldsymbol{num}(P_i)$ for each character unit P_i; (2) automatically dividing the domain $\boldsymbol{dom}(P_i)$ into $\boldsymbol{num}(P_i)$ partitions; and (3) automatically assigning a character identification for each partition of P_i. Below, we detail the steps and their implementations.

Algorithm 1 Assigning a group of partition numbers for a given private field.

```
Input: (1) the character units $P_1, P_2, ..., P_n$ of a private field $A_r$; and (2) a security
       factor $\rho$.
1 begin
2     foreach $P_i$ ($i = 1, 2, ..., n$) do $num(P_i) \leftarrow |dom(P_i)|$;
3     while $\rho \prod_{i=1}^{n} num(P_i) > \prod_{i=1}^{n} |dom(P_i)|$ do
4         from the set $\mathcal{P} = \{P_1, P_2, ..., P_n\}$, select a subset $\mathcal{P}^*$ of character units of the
          biggest current partition number, i.e.,
          $\mathcal{P}^* = \left\{P_i^* \mid P_i^* \in \mathcal{P} \wedge \forall P_j \left(P_j \in \mathcal{P} \rightarrow num\left(P_j\right) \leq num\left(P_i^*\right)\right)\right\}$;
5         from the set $\mathcal{P}^*$, randomly select a character unit $P_i^*$, and reset its current
          partition number as: $num(P_i^*) \leftarrow \left\lfloor \frac{1}{2} num(P_i^*) \right\rfloor$;
6     return $\{num(P_i) \mid i = 1, 2, ..., n\}$;
```

Step 1 Automatically assign a number $\boldsymbol{num}(P_i)$ (called a **partition number**) for each character unit P_i ($i = 1, 2, ..., n$) of the private field A_r, which has to meet the following requirements:

(1) The partition number $\boldsymbol{num}(P_i)$ has to be a positive integer and less than the size of the domain $\boldsymbol{dom}(P_i)$, i.e., $\boldsymbol{num}(P_i) \in \mathbb{N} \wedge \boldsymbol{num}(P_i) \leq |\boldsymbol{dom}(P_i)|$.
(2) All the partition numbers from the private field A_r have to meet $\rho \prod_{i=1}^{n} \boldsymbol{num}(P_i) \leq \prod_{i=1}^{n} |\boldsymbol{dom}(P_i)|$, where ρ is a given factor.

In Step 1, ρ is called a **security factor** and $\rho \in \mathbb{N} \wedge \rho \leq \prod_{i=1}^{n} |\boldsymbol{dom}(P_i)|$, which is preset for the private field A_r, and used to control the security of the generated feature index function. In general, the greater the value of ρ, the safer the generated feature index function $\boldsymbol{X}$. The detailed analysis of ρ on how to impact the security of $\boldsymbol{X}$ will be presented in Section 4.3. It can be found that generally there are a large number of solutions that satisfy the requirements mentioned in Step 1. In our approach, we use Algorithm 1 to perform Step 1, so as to automatically assign a group of partition numbers for all the character units of the private field A_r. From Algorithm 1, we can see that the time complexity of Lines 4 and 5 is $O(n)$ and the loop will terminate after $\log \rho$ operations, so the time complexity of Algorithm 1 is $O(n \cdot \log \rho)$.

Example 1 Consider a private field of phone number. Because a phone number in China generally consists of 11 numeric characters, the private field created for storing phone numbers also consists of 11 character units ($n = 11$). Besides, the first two numeric characters of a phone number can only be '13', '15' or '18', so the domain of each character unit of the phone field is given as follows:

$$\boldsymbol{dom}(P_1) = \{\text{'1'}\}; \boldsymbol{dom}(P_2) = \{\text{'3', '5', '8'}\}; \boldsymbol{dom}(P_3) = ... = \boldsymbol{dom}(P_{11}) = \{\text{'0', '1', '2', ..., '9'}\}$$

If the security factor ρ is set to 30, then using Algorithm 1, the partition number for each character unit of the phone field is assigned as follows:

$$\boldsymbol{num}(P_1) = 1; \boldsymbol{num}(P_2) = 3; \boldsymbol{num}(P_3) = ... = \boldsymbol{num}(P_6) = 10; \boldsymbol{num}(P_7) = ... = \boldsymbol{num}(P_{11}) = 5$$

Algorithm 2 Constructing partitions for each character unit of a private field.

```
Input: (1) the character units $P_1, P_2, ..., P_n$ of a given private field $A_r$; (2) a set
       $\{\boldsymbol{num}(P_i) \mid i = 1, 2, ..., n\}$ of partition numbers.
1  begin
2      foreach $P_i$ $(i = 1, 2, ..., n)$ do
3          $n_1 \leftarrow \lfloor |\boldsymbol{dom}(P_i)|/\boldsymbol{num}(P_i) \rfloor; n_2 \leftarrow \lceil |\boldsymbol{dom}(P_i)|/\boldsymbol{num}(P_i) \rceil$;
4          obtain a solution $(x_1, x_2)$ of the equation $\begin{cases} x_1 \in \mathbb{N}, x_2 \in \mathbb{N} \\ x_1 + x_2 = \boldsymbol{num}(P_i) \\ n_1 x_1 + n_2 x_2 = |\boldsymbol{dom}(P_i)| \end{cases}$
5          for $(k \leftarrow 1; x_1 \neq 0 \vee x_2 \neq 0; k \leftarrow k + 1)$ do
6              randomly set a variable $t$ as 1 or 2;
7              if $(x_1 \neq 0 \wedge t = 1 \vee x_1 \neq 0 \wedge x_2 = 0)$ then $x_1 \leftarrow x_1 - 1, n^* \leftarrow n_1$;
8              if $(x_2 \neq 0 \wedge t = 2 \vee x_2 \neq 0 \wedge x_1 = 0)$ then $x_2 \leftarrow x_2 - 1, n^* \leftarrow n_2$;
9              from the set $\boldsymbol{dom}(P_i)$, select the smallest $n^*$ elements to constitute a
               partition $B_k^{(i)}$, and update: $\boldsymbol{dom}(P_i) \leftarrow \boldsymbol{dom}(P_i) - B_k^{(i)}$;
10     return $\{\{B_k^{(i)} \mid k = 1, 2, ..., \boldsymbol{num}(P_i)\} \mid i = 1, 2, ..., n\}$;
```

Step 2 Based on $\boldsymbol{num}(P_i)$ assigned by Step 1 for each character unit of A_r, we use some strategy (e.g., Equi-width or Equi-depth) to divide the domain $\boldsymbol{dom}(P_i)$ of P_i into $\boldsymbol{num}(P_i)$ subsets (called **partitions**). Let $B_k^{(i)}$ ($k = 1, 2, ..., \boldsymbol{num}(P_i)$) denote a partition of P_i. The partitions of P_i have to meet the following requirements:

(1) Each partition $B_k^{(i)}$ is nonempty, i.e., $\forall k(k \in \mathbb{N} \wedge k \leq \boldsymbol{num}(P_i) \rightarrow B_k^{(i)} \neq \oslash)$.

(2) Each partition $B_k^{(i)}$ is mutually disjoint with another partition $B_j^{(i)}$, i.e., $\forall k \forall j(k, j \in \mathbb{N} \wedge k, j \leq \boldsymbol{num}(P_i) \wedge k \neq j \rightarrow B_k^{(i)} \cap B_j^{(i)} = \oslash)$.

(3) The union of all the partitions of P_i is $\boldsymbol{dom}(P_i)$, i.e., $\bigcup_{k=1}^{\boldsymbol{num}(P_i)} B_k^{(i)} = \boldsymbol{dom}(P_i)$.

(4) Each element in the partition $B_k^{(i)}$ is greater than each element in $B_{k-1}^{(i)}$, that is, $\forall k \forall a \forall b(k \in \mathbb{N} \wedge 2 \leq k \leq \boldsymbol{num}(P_i) \wedge a \in B_k^{(i)} \wedge b \in B_{k-1}^{(i)} \rightarrow a > b)$.

In our approach, we use Algorithm 2, which is developed based on an Equi-width strategy, to perform Step 2, so as to automatically divide the domain $\boldsymbol{dom}(P_i)$ of each character unit P_i of the private field A_r into $\boldsymbol{num}(P_i)$ partitions. It can be seen that Line 9 of the inner loop needs to scan all the elements in $B_k^{(i)}$, so the time complexity of the inner loop is $O\ (|\boldsymbol{dom}(P_i)|)$, and hence the time complexity of Algorithm 2 is $O\ (n \cdot \alpha)$ (where α denotes the averaged domain size of each character unit).

Example 2 Using the result of Example 1 as input, Algorithm 2 obtains a group of partitions for each character unit of the telephone number field, which are shown as the columns "partition" in Figure 3. It can be seen that the units P_1 and P_2 are divided into 1 and 3 partitions, respectively, and the units P_3 to P_6 and P_7 to P_{11} are divided into 10 and 5 partitions, respectively.

P_{11}		P_{10}		P_9		P_8		P_7		P_6		P_5		P_4		P_3		P_2		P_1	
partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier	partition	identifier
0,1	A	0,1	J	0,1	H	0,1	J	0,1	H	0	A	0	J	0	C	0	H	3	B	1	H
2,3	F	2,3	B	2,3	G	2,3	A	2,3	F	1	B	1	I	1	D	1	I	5	F		
4,5	G	4,5	C	4,5	B	4,5	F	4,5	D	2	C	2	H	2	E	2	J	8	G		
6,7	C	6,7	F	6,7	A	6,7	H	6,7	A	3	D	3	G	3	F	3	A				
8,9	D	8,9	I	8,9	J	8,9	I	8,9	J	4	E	4	F	4	G	4	B				
										5	F	5	E	5	H	5	C				
										6	G	6	D	6	I	6	D				
										7	H	7	C	7	J	7	E				
										8	I	8	B	8	B	8	F				
										9	J	9	A	9	A	9	G				

Figure 3 The partitions and identifiers for a private field of telephone number (each number in columns "partitions" denotes a numeric character)

Algorithm 3 Assigning the identifier for each partition of each character unit of a private field.

```
Input: (1) the partition numbers $\{\boldsymbol{num}(P_i) \mid i = 1, 2, ..., n\}$ of all the character units
       of a private field $A_r$; and (2) the partitions $\{B_k^{(i)} \mid k = 1, 2, ..., \boldsymbol{num}(P_i)\}$ of
       each character unit $P_i$ ($i = 1, 2, ..., n$).
1 begin
2     generate a random character $\theta$, and store it as metadata;
3     foreach $P_i$ ($i = 1, 2, ..., n$) do
4         based on the character $\theta$, construct a set of characters as follows:
          $H' = \left\{\theta, \theta + 1, \theta + 2, ..., \theta + \max_{t=1}^{n} (\boldsymbol{num}(P_t) - 1)\right\}$;
5         foreach $B_k^{(i)}$ ($k = 1, 2, ..., \boldsymbol{num}(P_i)$) do
6             from the set $H'$, randomly select a character $h$, and then set:
              $\boldsymbol{id}\left(B_k^{(i)}\right) \leftarrow h$ and update the set: $H' \leftarrow H' - \{h\}$;
7     return $\left\{\left\{\boldsymbol{id}\left(B_k^{(i)}\right) \mid k = 1, 2, ..., \boldsymbol{num}(P_i)\right\} \mid i = 1, 2, ..., n\right\}$;
```

Step 3 For each partition $B_k^{(i)}$ constructed by Step 2 for P_i, we determine a character $\boldsymbol{id}\left(B_k^{(i)}\right)$ as the **identifier** of $B_k^{(i)}$. It has to meet the following requirements:

(1) The identifiers of any two partitions of each character unit P_i are not equal to each other, that is, $\forall k \forall j \left(1 \leq k, j \leq \boldsymbol{num}(P_i) \wedge k \neq j \rightarrow \boldsymbol{id}\left(B_k^{(i)}\right) \neq \boldsymbol{id}\left(B_j^{(i)}\right)\right)$.

(2) The identifer of any partition of each P_i belongs to the same range of values (where θ is a randomly generated character): $\forall i \forall k \left(1 \leq i \leq n \wedge 1 \leq k \leq \boldsymbol{num}(P_i) \rightarrow 0 \leq \boldsymbol{id}\left(B_k^{(i)}\right) - \theta \leq \max_{j=1}^{n} \left(\boldsymbol{num}(P_j) - 1\right)\right)$.

We use Algorithm 3 to perform Step 3, so as to automatically assign a character identifier $\boldsymbol{id}\left(B_k^{(i)}\right)$ for each $B_k^{(i)}$ of each P_i of A_r. It can be seen that the time complexity of the inner loop (Lines 5 to 6) is equal to $O\ (\boldsymbol{num}(P_i))$, so the time complexity of Algorithm 3 is $O\ (n \cdot \beta)$ (where β denotes the averaged partition number of each character unit).

Example 3 Using the results of Examples 1 and 2 as input, Algorithm 3 assigns the identifier for each partition of each character unit of the telephone field, and the output results are shown as the columns "identifier" in Figure 3 (where θ is set to 'A'). It can be seen that each partition is set to an identifier within 'A' to 'J'.

Now, based on the above partitions and identifiers presented in Steps 1 to 3, we define n mapping functions: $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, ..., \boldsymbol{X}^{(n)}$. Given any character $u_i \in \boldsymbol{dom}(P_i)$, the function $\boldsymbol{X}^{(i)}$ would map u_i to the character identifier of the partition which u_i belongs to, i.e., $\boldsymbol{X}^{(i)}(u_i) = \boldsymbol{id}\left(B_k^{(i)}\right)$, where $B_k^{(i)}$ is the partition which contains u_i. Furthermore, given any character string $u = u_1 u_2 ... u_m\ (m \leq n)$ in the domain of the private field A_r, we can define a mapping function to map u to a new character string as: $\boldsymbol{X}(u) = \boldsymbol{X}^{(1)}(u_1)\boldsymbol{X}^{(2)}(u_2)...\boldsymbol{X}^{(m)}(u_m)$.

Example 4 Based on the partitions and identifiers shown in Figure 3, we can generate a feature index function $\boldsymbol{X}$ for the telephone number field. Now, given a telephone number

u = '13587898721', we can use $\boldsymbol{X}$ to map u to an index data $\boldsymbol{X}(u)$ = 'HBCBCIJIABA'. It can be seen that based on our index construction scheme, the index data will be always of the same type and length with its plaintext data.

4.2 Encryption storage

Now, we describe how encrypted data and indexed data are stored into the CloudDB. Given any tuple $t = \langle a_1, a_2, ..., a_r, ...\rangle$ over the relational table $R\,(A_1, A_2, ..., A_r, ...)$ (where a_r is a privacy data over the private field A_r), the corresponding encrypted relational table $R^E\left(A^E, A_1, A_2, ..., A_r^X, ...\right)$ in the CloudDB stores an encrypted tuple $t^E = \langle \boldsymbol{E}\left(\langle a_1, a_2, ..., a_r, ...\rangle\right), a_1, a_2, ..., \boldsymbol{X}(a_r), ...\rangle$, where $\boldsymbol{E}$ is a function used to encrypt a tuple of the relational table R. We treat the encryption function as a black box, thus any well-known data encryption technique (e.g., AES [5]) can be used.

Example 5 Let us consider a relation as: persons (no, name, phone, ...) (see the left in Figure 4). Then, the CloudDB stores an encrypted relation as: persons^E (tuple^E, no, name, phone^X, ...) (see the right in Figure 4), where the first column "tuple^E" stores the binary strings corresponding to the encrypted tuples. For example, the first tuple in "persons" is encrypted to "110111000011...", which is obtained by $\boldsymbol{E}$ (⟨1, 'Ada', '13587898721', ... ⟩). Moreover, the column "phone^X" in "persons^E" denotes the index field corresponding to the private field "phone" in "persons".

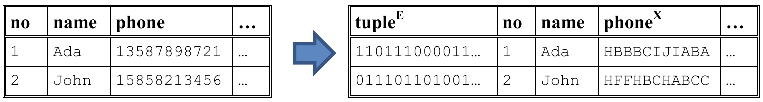
Note that in our approach, the tasks such as encrypting privacy data and generating their feature index values are all completed on a trusted client side by the index generator; and then the encrypted data and their index data are transmitted through network and stored into the CloudDB (see Figure 2).

4.3 Security analysis

In this subsection, we firstly demonstrate that our approach can meet λ-**security**, and then briefly analyze the security of the index function generated by our approach in terms of two types of common attacks: statistical attack and known-plaintext attack.

Observation 1 Given a threshold λ $(0 < \lambda \leq 1)$, after the **security factor** ρ of the privacy protection scheme is set to $\lfloor 1/\lambda \rfloor$, our approach can meet λ-**security**.

Rationale Based on the index function $\boldsymbol{X}$ constructed in Section 4.1, we know that the index function $\boldsymbol{X}$ is a many-to-one mapping from privacy domain $\mathcal{U}$ to index domain $\mathcal{X}$. Moreover, each index value in $\mathcal{X}$ would correspond to ρ privacy values in $\mathcal{U}$. Thus, we conclude that if the security factor ρ of the privacy protection scheme is set to $\lfloor 1/\lambda \rfloor$, each index value in $\mathcal{X}$ corresponds to $\lfloor 1/\lambda \rfloor$ privacy values in $\mathcal{U}$, i.e., the probability of inferring the plaintext from any index value is always less than λ, even if an adversary has mastered



no	name	phone	...
1	Ada	13587898721	...
2	John	15858213456	...

tuple^E	no	name	phone^X	...
110111000011...	1	Ada	HBBBCIJIABA	...
011101101001...	2	John	HFFHBCHABCC	...

Figure 4 A translation from a relational table and its encrypted relational table

the index function $\boldsymbol{X}$. After combining Definition 1, we know that our approach meets λ-security if the security factor ρ is set to $\lfloor 1/\lambda \rfloor$.

The precondition of statistical attack is that an attacker has known a set $\mathcal{U}^*$ of privacy data and a set $\mathcal{X}^*$ of corresponding index data. Then, the attacker attempts to establish a set of two-tuples from $\mathcal{U}^*$ to $\mathcal{X}^*$, so as to reconstruct the index function $\boldsymbol{X}$. The implementation of statistical attack is based on an observation that the probability of occurrences of each u in $\mathcal{U}^*$ is basically consistent with $\boldsymbol{X}(u)$ in $\mathcal{X}^*$. However, in our approach, each unit of privacy data is divided into several partitions by some strategy (e.g., Equi-width or Equi-depth), so that many privacy values would be mapped into the same index value, as a result, lightening the consistency of probability distribution between privacy data and index data.

The precondition of known-plaintext attack is that an attacker has known a small set of two-tuples from privacy data $\mathcal{U}^*$ to index data $\mathcal{X}^*$. Then, the attacker attempts to reconstruct the index function $\boldsymbol{X}$. From Section 4.1, we know that to reconstruct $\boldsymbol{X}$, we need to know the partitions of each unit of privacy data. For the unit P_i of privacy data, we at least need to know $|\boldsymbol{dom}(P_i)|$ two-tuples from privacy data to index data to reconstruct $\boldsymbol{X}^{(i)}$. Therefore, we probably need to know at least $(\sum_{i=1}^{n} |\boldsymbol{dom}(P_i)|)$ two-tuples from privacy data to index data to reconstruct $\boldsymbol{X}$. Finally, it should be pointed out that, it is possible for an attacker to guess the function $\boldsymbol{X}$ using statistical or known-plaintext attack, especially when the security factor μ is set to a greater value (e.g., equal to 1.0)

However, even if an attacker has completely mastered the index function $\boldsymbol{X}$ based on statistical attack or known-plaintext attack, it is still difficult for the attacker to guess the corresponding plaintext u from a given index value x. This is because our approach can meet λ-**security**, making that the attacker only has a $\lfloor 1/\lambda \rfloor$ probability to guess the corresponding plaintext u from the index value x. Besides, based on the index function $\boldsymbol{X}$ constructed in Section 4.1, we know that although our approach can meet λ-**security**, the index values generated by our approach still might reveal some sensitive information to the cloud-side, e.g., the length of the private field since the index field is of the same length as the corresponding private field. It is our next work how to improve the feature index scheme so as to make the index safer (not just to meet the λ-**security**).

5 Privacy query scheme

In our approach, privacy data will be encrypted before being stored into the CloudDB, so as to ensure the security. However, this leads to that a number of user query operations defined over the private field will be no longer able to be executed correctly on the encrypted CloudDB. In this section, we discuss the privacy query scheme used in our approach, i.e., how each database query q_u over the private field A_r is mapped into a new cloud-side query q_x over the corresponding index field A_r^X, so that the query q_x can be executed on the CloudDB correctly. To this end, we first discuss how each type of basic query conditions over the private field is mapped to the cloud-side representation over the index field. Second, based on the condition mappings, we discuss how a database query q_u is transformed to its cloud-side query q_x. Finally, we analyze the usability and efficiency of the proposed privacy query scheme.

5.1 Mapping query conditions

A database query operation consists of several basic query conditions. Thus, once we know how each type of basic query conditions over the private field is mapped correctly into its cloud-side representation, we can know how a database query operation is mapped into its cloud-side query operation. In this subsection, we consider three main types of basic query conditions over the private field A_r: (1) **equivalent conditions**, e.g., $R.A_r$ = '123'; (2) **similarity conditions**, e.g., $R.A_r$ LIKE '%123%'; and (3) **range conditions**, e.g., $R.A_r >$ '123'. Below, we call the process of mapping a basic query condition to its cloud-side representation as condition mapping for short, and use ***map*** to denote such a condition mapping. Besides, we use the table in Example 5 and the index function generated in Example 4 to illustrate the condition mapping.

Mapping 1 $R.A_r = u$: this is the basic form of an equivalent condition, where u denotes a character string constant, and $R.A_r$ denotes a private field of a relational table R. If $u = u_1u_2...u_m (m \leq n)$, then the condition mapping is defined as follows:

$$\boldsymbol{map}(R.A_r = u) \Rightarrow R^E.A_r^X = \boldsymbol{X}(u) \Rightarrow R^E.A_r^X = \boldsymbol{X}^{(1)}(u_1)\boldsymbol{X}^{(2)}(u_2)...\boldsymbol{X}^{(m)}(u_m).$$

For example, since '13587898721' would be mapped into 'HBCBCIJIABA' by the index function generated in Section 4, we have a condition mapping as follows:

$$\boldsymbol{map}\ (\text{persons.phone} = \text{'13587898721'}) \Rightarrow \text{persons}^E.\text{phone}^X = \text{'HBCBCIJIABA'}$$

A similarity query condition generally contains some wildcards, and the similarity wildcards include: (1) '%', it denotes to match one or more characters; (2) '_', it denotes to match only one character; and (3) '[CharList]', it denotes to match any character described in 'CharList'. Thus, we below present the mappings for three main types of similarity conditions.

Mapping 2 $R.A_r$ LIKE u_v: this is the basic form of a similarity condition based on the wildcard '_', where u and v represent two character string constants. If $u = u_1u_2...u_m$ and $v = v_1v_2...v_k$ $(m + k \leq n - 1; 0 \leq m; 0 \leq k)$, then the condition mapping is defined as follows:

$$\boldsymbol{map}(R.A_r \text{ LIKE } u\_v) \Rightarrow R^E.A_r^X \text{ LIKE } u^X\_v^X, \text{where } \begin{cases} u^X = \boldsymbol{X}^{(1)}(u_1)...\boldsymbol{X}^{(m)}(u_m) \\ v^X = \boldsymbol{X}^{(m+2)}(v_1)...\boldsymbol{X}^{(m+k+1)}(v_k) \end{cases}$$

For example, we have a mapping of similarity condition as follows:

$$\boldsymbol{map}\ (\text{persons.phone LIKE '1358789\_721'}) \Rightarrow \text{persons}^E.\text{phone}^X\text{LIKE 'HBCBCIJ\_ABA'}$$

Mapping 3 $R.A_r$ LIKE $u[l]v$: this is the basic form of a similarity condition based on the wildcard '[]', where u and v denote two character string constants, and l denotes a character list. If $l = l_1l_2...l_t$, $u = u_1u_2...u_m$ and $v = v_1v_2...v_k$ $(m+k \leq n-1; 0 \leq m; 0 \leq k; 0 \leq t)$, then the condition mapping is defined as follows:

$$\boldsymbol{map}\ (R.A_r \text{ LIKE } u[l]v) \Rightarrow R^E.A_r^X \text{ LIKE } u^X[l^X]v^X, \text{where } \begin{cases} u^X = \boldsymbol{X}^{(1)}(u_1)...\boldsymbol{X}^{(m)}(u_m) \\ l^X = \boldsymbol{X}^{(m+1)}(l_1)...\boldsymbol{X}^{(m+1)}(l_t) \\ v^X = \boldsymbol{X}^{(m+2)}(v_1)...\boldsymbol{X}^{(m+k+1)}(v_k) \end{cases}$$

For example, since each character in the list '[789]' is respectively mapped to 'H', 'I' and 'I' by the index function $\boldsymbol{X}$, we have a condition mapping as follows:

$$\boldsymbol{map}\ (\text{persons.phone LIKE '1358789[789]721'}) \Rightarrow \text{persons}^E.\text{phone}^X\text{LIKE 'HBCBCIJ[HII]ABA'}$$

Mapping 4 $R.A_r$ LIKE $u\%v$: this is the basic form of a similarity condition based on the wildcard '%', where $u = u_1u_2...u_m$ and $v = v_1v_2...v_k$ $(m + k \leq n - 1; 0 \leq m; 0 \leq k)$. Since the wildcard '%' represents to match one or more characters, it is equivalent to '_' (i.e., one character), ‘__’ (i.e., two characters) etc., and the maximum number of ‘_’ is not more than $(n - m - k)$. Based on such an observation, with the help of Mapping 3, the condition mapping is defined as follows:

$$\boldsymbol{map}\,(R.A_r \text{ LIKE } u\%v) \Rightarrow \text{OR}_{i=1}^{n-m-k}\, R^E.A_r^X \text{ LIKE}\, u^X \overbrace{\_\ldots\_}^{i} v_i^X, \text{ where } \begin{cases} u^X = \boldsymbol{X}^{(1)}(u_1)\ldots\boldsymbol{X}^{(m)}(u_m) \\ v_i^X = \boldsymbol{X}^{(m+i+1)}(v_1)\ldots\boldsymbol{X}^{(m+i+k)}(v_k) \end{cases}$$

Specially, if $k = 0$, then the condition mapping can be defined as follows:

$$\boldsymbol{map}\,(R.A_r \text{ LIKE}\, u\%) \Rightarrow R^E.A_r^X \text{ LIKE}\, \boldsymbol{X}^{(1)}(u_1)\ldots\boldsymbol{X}^{(m)}(u_m)\%$$

Specially, if $m = 0$, then the condition mapping can be defined as follows:

$$\boldsymbol{map}\,(R.A_r \text{ LIKE}\, \%v) \Rightarrow R^E.A_r^X \text{ LIKE}\, \%\boldsymbol{X}^{(n-k+1)}(v_1)\ldots\boldsymbol{X}^{(n)}(v_k)$$

For example, we have three mappings about similarity condition as follows:

map (persons.phone LIKE ‘135%’) ⇒ persons^E.phone^{X}LIKE ‘HBC%’
map (persons.phone LIKE ‘%721’) ⇒ persons^E.phone^{X}LIKE ‘%ABA’
map (persons.phone LIKE ‘135%898721’) ⇒ persons^E.phone^{X}LIKE‘HBC_BJJHGJ’ OR
persons^E.phone^{X}LIKE ‘HBC__IJIABA’

Mapping 5 $R.A_r \geq u$: this is the basic form of a range query condition. Without loss of generality, we assume that $u = u_1u_2...u_m(m \leq n)$ and note v_i as a character of the greatest value in the character unit $P_i(i = 1, 2, ..., n)$, i.e., $\forall v^*(v^* \in \boldsymbol{dom}(P_i) \rightarrow v^* \leq v_i)$. Then, any character string $u^* = u_1^*u_2^*...u_h^*(1 \leq h)$ is greater than u, if and only if it satisfying that: $(u_1 = u_1^*, u_2 = u_2^*, ..., u_m = u_m^*, m \leq h)$; or $(u_1 \leq u_1^* + 1)$; or $(u_1 = u_1^*, u_2 \leq u_2^* + 1$; or ...; or $(u_1 = u_1^*, u_2 = u_2^*, ..., u_{k-1} = u_{k-1}^*, u_k \leq u_k^* + 1)$, where k is equal to m (if $m \leq h$) or h (if $m > h$). Based on such an observation, the range condition mapping is defined as follows:

$$\boldsymbol{map}\,(R.A_r \geq u) \Rightarrow \text{OR}_{i=1}^{m}\, \boldsymbol{map}\,(R.A_r \text{ LIKE } u_1u_2...u_{i-1}[(u_i + 1) - v_i])\,\%\,\text{OR}\, \boldsymbol{map}\,(R.A_r \text{ LIKE } u)\,\%$$

For example, we have a mapping about range query condition as follows:

map (persons.phone ≥ ‘13587’) ⇒ persons^E.phone^{X}LIKE ‘HBCBC%’ OR
persons^E.phone^{X}LIKE ‘H[FG]%’ OR
persons^E.phone^{X}LIKE ‘HB[DEFG]%’ OR
persons^E.phone^{X}LIKE ‘HBC[A]%’ OR
persons^E.phone^{X}LIKE ‘HBCB[BA]%’

Besides, we can define a similar mapping for another range condition: $R.A_r < u$.

Above, we describe the condition mappings for three main types of basic query conditions defined over privacy data. It can be seen that all the cloud-side condition representations are defined over the index field $R^E.A_r^X$, thus can be executed on the encrypted CloudDB. Besides, it can be noted that each cloud-side representation p_x is a sufficient condition of the corresponding query representation p_u over privacy data, i.e., the result of executing p_x on the CloudDB would be a superset of that of p_u (i.e., $R(p_u) \subseteq \boldsymbol{D}(R(p_x))$).

5.2 Privacy query processing

For any SELECT query operation from a client side, from the WHERE clause of the query, we can first obtain all the relevant basic query conditions defined over the privacy field. Second, based on the condition mappings (i.e., Mappings 1 to 5) mentioned above, we map each basic query condition into a new condition representation defined over the corresponding index field of the CloudDB. Finally, we combine all the new condition representations to form a new cloud-side query operation. Note that all the works are completed by the query translator running on a trusted client side (see Figure 2).

Example 6 Consider the relation "persons" and the encrypted relation "persons^E" mentioned in Example 5. First, we present a SQL query operation (q_u.) as follows:

SELECT p.no, p.name FROM persons p WHERE p.phone = '15858707069' OR p.phone LIKE '1358789_721'.

Then, based on the condition mappings, the query operation q_u can be mapped into a new cloud-side query operation (q_x) defined over the index field "persons^E.phone^X":

SELECT p.tuple^E FROM persons^E p WHERE p.phone^X = 'HFFHBCHABCC' OR p.phone^X LIKE 'HBCBCIJ_ABA'.

Finally, the cloud-side query q_x will be submitted to the cloud side, instead of the query q_u. After the query q_x is executed by the CloudDB, a set $R(q_x)$ of encrypted tuples will be returned to the client side, which is a superset of the result $R(q_u)$ of q_u. Then, on the client side, the query executor will decrypt the set $R(q_x)$ of encrypted tuples, and execute the original query q_u over the decrypted tuples again, so as to obtain the accurate query result $R(q_u)$.

From above, we can see that, for a user query q_u, its corresponding cloud-side query q_x and the intermediate query result $R(q_x)$ are both revealed to the untrusted cloud-side. However, the cloud-side query q_x and the query result $R(q_x)$ are not plaintext, where the query result $R(q_x)$ are in the form of ciphertext, and the conditions of q_x are defined over the index fields. Therefore, although $R(q_x)$ and q_x are visible to the cloud-side, it is difficult for the cloud-side to guess the private information from them.

5.3 Usability and efficiency analysis

In this subsection, based on Definitions 2 and 3, we use some observations to demonstrate the usability and efficiency of our proposed approach.

Observation 2 Let $\mathcal{P}_u$ denote a set of all the basic query conditions defined over the private field $R.A_r$. Then, any query requirement that is relevant to the private field $R.A_r$ can be described using a logical operation (i.e., an expression connected by NOT, AND and OR) among the basic query conditions in $\mathcal{P}_u$.

Observation 3 Let p_1 and p_2 denote two basic query conditions over the private field $R.A_r$, and p_1^* and p_2^* two cloud-side query representations corresponding to p_1 and p_2. Then, the result of executing the AND condition "p_1^* AND p_2^*" on the CloudDB will be a superset of that of "p_1 AND p_2" (i.e., $R(p_1 \text{ AND } p_2) \subseteq \boldsymbol{D}(R(p_1^* \text{ AND } p_2^*))$). Similarly, we have $R(p_1 \text{ OR } p_2) \subseteq \boldsymbol{D}(R(p_1^* \text{ OR } p_2^*))$.

Rationale The two observations mentioned above can be easily demonstrated by the fundamentals of the logic algebra and the relation algebra.

In Observation 3, we have not mentioned the NOT condition. Actually, based on Mappings 1 to 5, we know that $R(\text{NOT } p_1) \nsubseteq \boldsymbol{D}(R(\text{NOT } p_1^*))$, that is, our approach cannot support NOT logical operations. However, for any NOT condition, we generally can generate an equivalent positive condition. For example, "NOT p.phone > 13587898721" is equivalent to "p.phone <= 13587898721". Thus, based on Observation 2 and 3, we have an observation as follows.

Observation 4 The privacy query scheme used in our approach can meet usability, i.e., any user query operation q_u over private data can be transformed into a cloud-side query operation q_x, which is defined over the corresponding index data, and the result $R(q_u)$ of q_u is contained in the result $R(q_x)$ returned by q_x, i.e., $R(q_u) \subseteq \boldsymbol{D}(R(q_x))$.

Observation 5 Given any threshold μ ($0 < \mu \leq 1$), after setting a suitable value for the security factor ρ of the privacy protection scheme, our approach can meet μ-**efficiency**.

Rationale Let us consider an extreme case where the security factor $\rho = 1$. At this time, the index function become an one-to-one mapping, and thus all the non-target tuples can be filtered out by cloud-side query operations, i.e., at this time, our approach can meet 1.0-**efficiency**. Thus, for any given threshold μ ($0 \leq \mu \leq 1$), our approach can meet μ-**efficiency**.

Based on Observation 1 and Observation 5, we can conclude that the security factor ρ is proportional to the security of our approach, but is inversely proportional to the efficiency of our approach.

6 Experiment evaluation

In Section 4.3, we have demonstrated the efficiency of our approach by theoretical analysis. In this section, we evaluate the efficiency of our approach by experiments, i.e., to evaluate the filtering rate (refer to Definition 4) of cloud-side query operations generated by our approach to filter out non-target tuples in the CloudDB.

6.1 Experimental setup

Before the experimental evaluation,we briefly describe the experimental setup, including the dataset preparation, user queries and system configuration.

(1) **Dataset preparation**. To perform the experiments, we in advance constructed a database, which only contains one relational table "persons". The schema of the table "persons" is similar to that shown as Example 5, but it contains two private fields "phone" and "name". Table 1 presents some information related to the two private fields. Then, we randomly generated a million of tuples for the table "persons" (i.e., the database size is about one million orders of magnitude), where the privacy field values were generated based on the two regular expressions presented in the fifth column of Table 1. As shown in Table 1, each value of the field "name" is defined over a set of 100 Chinese characters, and consists of at least three but up to five characters.

Table 1 The information about the private field that needs to be encrypted

Table name	Private field	Data type	Tuple number	Regular expression
Persons	phone	CHAR (11)	1,000,000	[1][358][0-9]{9}
Persons	name	CHAR (5)	1,000,000	[\u4e00-\u5200]{3,5}

Table 2 The similarity and range conditions used in the experiments, where A_1, A_2 and A_3 denote three characters

Symbols	LIKE conditions	Symbols	Range conditions
PL1	phone LIKE "%A_1"	PR1	phone $\geq$ "15A_1"
PL2	phone LIKE "%$A_1 A_2$"	PR2	phone $\geq$ "15$A_1 A_2$"
PL3	phone LIKE "%$A_1 A_2 A_3$"	PR3	phone $\geq$ "15$A_1 A_2 A_3$"
NL1	name LIKE "%A_1"	NR1	name $\geq$ "A_1"
NL2	name LIKE "%$A_1 A_2$"	NR2	name $\geq$ "$A_1 A_2$"
NL3	name LIKE "%$A_1 A_2 A_3$"	NR3	name $\geq$ "$A_1 A_2 A_3$"

(2) **User query operations**. Table 2 presents the basic query conditions used in our experiments. Table 2 shows the general cases for two main types of basic queries (i.e., basic similarity queries and basic range queries) over the private field "phone" or "name". It should be noted that equivalent queries can be considered as a special kind of similarity queries. In addition, other more complex similarity queries or range queries can be generated based on these basic query operations.

(3) **System configuration**. The experiments were conducted over two Lenovo personal computers with an Intel (R) Core (TM) I7-4510U CPU and 8 GB RAM, where one of the two computers performed as the cloud-side, and the other as the client side. The network speed between computers is about 2.0 MB/s, and the disk speed is about 200 MB/s. In addition, we used Microsoft Windows 7 as the operating system, and MySQL (version 5.7.17) as the database system.

6.2 Efficiency evaluation and analysis

In the experiments, we used the metric *Fr* (i.e., the **filtering rate** defined in Section 3.3) to evaluate the efficiency of the approach. Aiming at similarity queries, we conducted two groups of experiments over the private fields "phone" and "name", respectively, by setting different values for the security factor ρ. The experimental results are shown in Table 3, where each value was obtained by performing 10 experiments and then computing their average value.

From Table 3, we have the following four observations. First, with the increasing of the security factor ρ, the *Fr* value decreases, i.e., the effectiveness of the cloud-side query operations to filter non-target tuples is reduced, and thus the efficiency of the approach is also

Table 3 The *Fr* values for different similarity query conditions over the private field "phone" or "name" (ρ is set to $2^9 - 2^{21}$)

Factor (ρ)	$\rho = 2^9$	$\rho = 2^{11}$	$\rho = 2^{13}$	$\rho = 2^{15}$	$\rho = 2^{17}$	$\rho = 2^{19}$	$\rho = 2^{21}$
PL1	0.88889	0.83951	0.79012	0.74074	0.69136	0.64198	0.59259
PL2	0.96970	0.94974	0.92580	0.89787	0.86594	0.83003	0.79012
PL3	0.99299	0.98638	0.97687	0.96393	0.94703	0.92566	0.89927
NL1	0.97374	0.96566	0.95758	0.94949	0.94141	0.93333	0.92525
NL2	0.99880	0.99816	0.99740	0.99650	0.99548	0.99432	0.99304
NL3	0.99995	0.99992	0.99986	0.99978	0.99969	0.99956	0.99941

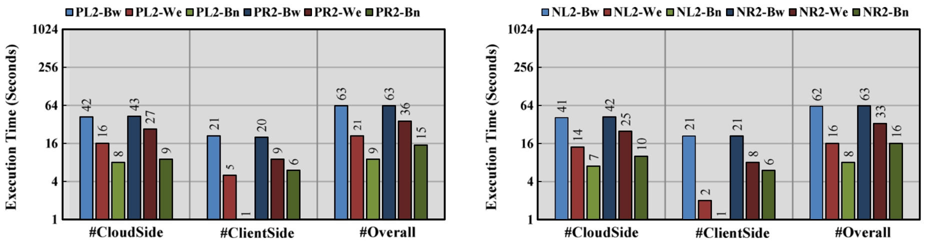
Table 4 The *Fr* values for different range query conditions over the private field "phone" or "name" (ρ is set to $2^9 - 2^{21}$)

Factor (ρ)	$\rho = 2^9$	$\rho = 2^{11}$	$\rho = 2^{13}$	$\rho = 2^{15}$	$\rho = 2^{17}$	$\rho = 2^{19}$	$\rho = 2^{21}$
PR1	0.80000	0.75556	0.71111	0.66667	0.62222	0.57778	0.53333
PR2	0.81818	0.79798	0.77778	0.75758	0.73737	0.71717	0.69697
PR3	0.85586	0.84585	0.83584	0.82583	0.81582	0.80581	0.79580
NR1	0.96400	0.95600	0.94800	0.94000	0.93200	0.92400	0.91600
NR2	0.98090	0.97630	0.97158	0.96673	0.96176	0.95666	0.95143
NR3	0.99521	0.99283	0.99036	0.98779	0.98511	0.98231	0.97939

reduced accordingly. The reason is that with the increasing of ρ, it would increase the number of different privacy field values mapped into the same index field value by the index function, consequently, decreasing the probability of non-target tuples being filtered. Second, different similarity conditions lead to the different change trends of *Fr* values, and the *Fr* values would increase with the increasing of quantity of information contained by the similarity matching conditions (PL1−PL3 and NL1−NL3). This is because the increasing of quantity of information in the matching conditions would decrease the number of tuples returned by the cloud-side query (i.e., $R(q_x)$ in Definition 4), resulting in the increasing of the *Fr* values. Third, each *Fr* value related to "name" is generally greater than that related to "phone", which is caused by the larger value domain of the private field "name". Finally, we find that the mathematical expectation of the *Fr* values for the similarity query operations over the private field "phone" is equal to 0.86222 (we assume the same probability of occurrence of each similarity query operation); and the mathematical expectation over "name" is equal to 0.98183. As a result, this would reduce the number of encrypted tuples transmitted from the cloud side to the client, thereby, improving the efficiency of similarity query operations.

Aiming at range queries, we also conducted two groups of experiments over the private fields "phone" and "name", respectively. The experimental results are presented in Table 4. In general, the experimental results are similar to those of similarity queries: (1) the increasing of the security factor ρ decreases the effectiveness of the cloud-side query operations to filter non-target tuples of the CloudDB, thereby, decreasing the efficiency of the approach; (2) the greater value domain of the private field "name" makes that each *Fr* value of "name" is generally greater than that of "phone"; and (3) the mathematical expectations over "phone" and "name" are respectively equal to 0.75003 and 0.96468, i.e., most of non-target tuples generally can be filtered by the cloud-side query operations, consequently, improving the efficiency of range query operations.

In addition, we have also conducted experiments to evaluate the actual execution performance of our approach. In the experiments, we compared our approach (below, denoted by "We") with the following two ways: (1) decrypting the encrypted data before querying them (denoted by "Bw"); and (2) querying data without encryption (denoted by "Bn", i.e., directly storing plaintext into the CloudDB). In the experiments, the execution performance of our approach is computed by adding: (1) the time of executing a cloud-side query on the CloudDB, and transmitting the encrypted data from the cloud-side to the client, i.e., the time consumed on the cloud-side; and (2) the time of decrypting and querying the data on the client, i.e., the time consumed on the client-side. The experiments were performed based on the basic similarity queries "PL2" and "NL2", and the basic range queries "PR2"



(a) The execution times over the private field "phone". (b) The execution times over the private field "name".

Figure 5 The execution times for performing similarity and range queries over the private field "phone" or "name"

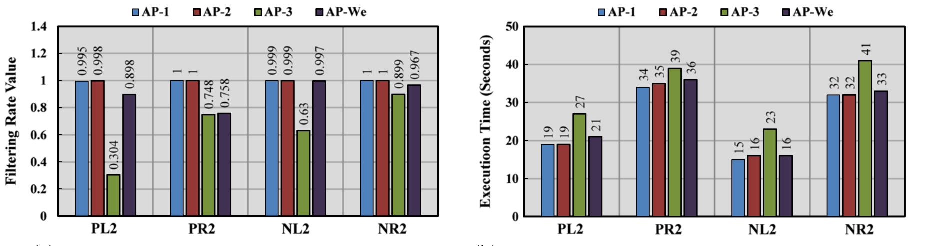
and "NR2". The experimental results are shown in Figure 5, where the security factor ρ is set to 2^{15}. From the two subfigures, we see that based on the feature index generated by our approach, the overall execution performance of similarity and range queries over the private fields can be improved effectively: compared with those of "Bw", the overall execution time of a basic similarity query is decreased to about 0.3, and the execution time of a range query is decreased to about 0.6. In addition, we also see that the execution performance of our approach is almost twice that of "Bn", which is mainly because the volume of the encrypted tuples is greater than the volume of the original tuples (i.e., the tuples without encryption).

Finally, based on the above experimental results, we conclude that the increasing of the security factor ρ would decrease the efficiency of the proposed approach, i.e., it would decrease the effectiveness of the cloud-side query operations generated by our approach to filter non-target tuples of the CloudDB.

6.3 Effectiveness comparison and analysis

From the related work section, we know that there have been many approaches to database encryption, but most of them were not designed for personal privacy protection in a CloudDB, thereby, making them difficult to be applied into a CloudDB. In this subsection, we compare our approach with three existing ones proposed in [18], [29], and [33], respectively. It should be pointed out that all the approaches were not designed for a CloudDB. For comparison, we have re-implemented the approaches over our prototype experimental system.

First, we make an effectiveness comparison in terms of efficiency (i.e., Definition 4). In the experiments, (1) for our approach, the security factor ρ is set to 2^{15}; (2) for the approach



(a) The *Fr* values over the private field "phone"/"name". (b) The execution times over the private field "phone"/"name".

Figure 6 The efficiency comparisons between our approach and other existing ones

Table 5 The effectiveness comparison, where "low" denotes non-support, "high" denotes good support, and "medium" denote some support

Approach	in [18]	in [29]	in [33]	Of ours
Similarity queries	Medium	Medium	High	High
Range queries	Low	Low	High	High
Security	Medium	Medium	High	High
Efficiency	High	High	Medium	High

in [29], the number of bits of characteristic index field is set to 32 (that is recommended by the authors); and (3) for the approach in [33], the size of an index matrix is set to 8 (for the private field "phone") or 20 (for the private field "name"). The experiments were performed based on the basic similarity queries "PL2" and "NL2", and the basic range queries "PR2" and "NR2". The experimental results are shown in Figure 6, where "AP-1", "AP-2" and "AP-3" denotes the approaches presented in [18, 29], and [33], respectively, and "AP-We" denotes our approach. From Figure 6, we see that, our approach has nearly the same running efficiency to the approaches presented in [18] and [29], and has better running efficiency than that presented in [33].

Second, based on the above results and the results mentioned in [18, 29], and [33], we make an overall effectiveness comparison in terms of security (i.e., Definition 1), usability (i.e., Definition 3) and efficiency (i.e., Definition 4). The comparison results are shown in Table 5. From Table 5, we can see that, compared to the other approaches, our approach not only has better usability better, namely, which can support all kinds of query operations over text private fields (including similarity queries and range queries), but also has better security, enabling us to prevent attackers from attacking, thus, better ensuring the security of personal privacy in a CloudDB. Overall, our approach has a better overall effectiveness in terms of security, usability and efficiency than the other approaches.

7 Conclusion

In this paper, we proposed a client-based approach to protect personal privacy in a CloudDB. The approach presents a privacy protection scheme and a privacy query scheme, which can ensure not only good security of privacy data, but also good efficiency of query operations over privacy data. Moreover, we demonstrated the effectiveness of the approach by theoretical analysis and experimental evaluation. The results show that: (1) the feature index constructed by the approach has good security, i.e., it is difficult to infer the plaintext from the feature index data; (2) the approach has good usability, i.e., with the help of the feature index, each type of familiar query operations over privacy data can be transformed into a cloud-side query operation that can be performed correctly at the cloud-side; and (3) the approach has good efficiency, i.e., with the help of a cloud-side query operation, most of non-target tuples can be filtered out at the cloud-side, consequently, improving the execution efficiency of a client-side query operation over privacy data.

However, the approach proposed in this paper is not the end of our work. As the future work, we will try to further study some problems, e.g., (1) how to establish a solution to automatically determine the security factor ρ based on the characteristics of users' privacy data, instead of being preset by users; (2) how to improve the approach, so as to support more privacy data types, not just text data type; and (3) the practical implementation of this approach in a CloudDB. In addition, in this work, we only focus on the protection of users' privacy data; however, in a CloudDB, users' behaviour may potentially pose a threat to personal privacy. Therefore, it is also our future work of how to protect the privacy behind users' behaviour.

Acknowledgements We would like to thank anonymous reviewers for their valuable comments. The work is supported by the National Social Science Fund of China (No. 17CTQ011).

References

1. 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: Proc. of the CIDR (2005)
2. Ahituv, N., Lapid, Y., Neumann, S.: Processing encrypted data. Commun. ACM **30**(9), 777–780 (1987)
3. Alfred, B., Melissa, Z.: Database Security. Delmar Cengage Learning (2011)
4. Arvind, A., Spyros, B., Ken, E., Manas, J., Raghav, K., Donald, K., Ravi, R., Prasang, U.: Secure database-as-a-service with cipherbase. In: Proc. of the SIGMOD (2013)
5. Ashwini, M.D., Mangesh, S.D., Devendra, N.K.: Fpga implementation of aes encryption and decryption. In: Proc. of the 2009 International Conference on Control, Automation, Communication and Energy Conservation (2009)
6. Bharath, S., Wei, J., Elisa, B.: Privacy-preserving complex query evaluation over semantically secure encrypted data. In: Proc. of the ESORICS (2014)
7. Bijit, H., Sharad, M., Gene, T.: A privacy-preserving index for range queries. In: Proc. of the VLDB (2007)
8. Boyang, W., Ming, L., Haitao, W., Hui, L.: Circular range search on encrypted spatial data. In: Proc. of the ICDCS (2015)
9. Chen, F., Liu, A.X.: Privacy and integrity preserving multi-dimensional range queries for cloud computing. In: Proc. of the IFIP (2014)
10. Chen, K., Weimin, Z.: Cloud computing: System instances and current research. J. Softw. **20**(5), 1137–1148 (2010)
11. Feng, D., Zhang, M., Zhang, Y., Xu, Z.: Study on cloud computing security. J. Softw. **22**(1), 71–83 (2011)
12. Ganapathy, V., Thomas, D., Feder, T., Garcia-Molina, H., Motwani, R.: Distributing data for secure database services. In: Proceedings of the 4th International Workshop on Privacy and Anonymity in the Information Society. ACM (2011)
13. Hacigümüṡ, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proc. of the ACM SIGMOD (2002)
14. Hacigümüṡ, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Proc. of the DASFAA (2004)
15. Hacigümüṡ, H., Iyer, B., Mehrotra, S.: Query optimization in encrypted database systems. In: Proc. of the DASFAA (2005)
16. Huang, L., Tian, M., Huang, H.: Preserving privacy in big data: A survey from the cryptographic perspective. J. Softw. **26**(4), 777–780 (2015)
17. Josep, D.F.: A new privacy homomorphism and applications. Inf. Process. Lett. **60**(5), 227–282 (1996)
18. Li, Y., Liu, G.: Encryption method for character data in the database. Comput. Eng. **33**(6), 120–124 (2007)
19. Li, R., Liu, A.X., Wang, A.L.: Fast range query processing with strong privacy protection for cloud computing. Proc. VLDB Endow. **7**(14), 1953–1964 (2014)
20. Lin, Z., Lai, Y., Lin, C., Xie, Y., Quan, Z.: Research on cloud databases. J. Softw. **23**(5), 1148–1166 (2012)
21. Liu, A., Zheng, K., Li, L., Liu, G., Zhou, X.: Efficient secure similarity computation on encrypted trajectory data. In: Proc. of the ICDE (2015)
22. Luc, B., Philippe, P.: Chip-secured data access: Confidential data on untrusted servers. In: Proc. of the VLDB (2002)
23. Luca, F., Michele, C., Mirco, M.: Distributed, concurrent, and independent access to encrypted cloud databases. IEEE Trans. Parallel Distrib. Syst. **25**(2), 437–450 (2014)

24. Shiyuan, W., Divyakant, A., Amr, E.A.: A comprehensive framework for secure query processing on relational data in the cloud. In: Proc. of the VLDB Workshop on Secure Data Management (2011)
25. Sumeet, B., Radu, S.: Trusteddb: A trusted hardware-based database with privacy and data confidentiality. IEEE Trans. Knowl. Data Eng. **26**(3), 752–768 (2014)
26. Sun, X., Li, M., Wang, H.: A family of enhanced (ℓ, α)-diversity models for privacy preserving data publishing. Futur. Gener. Comput. Syst. **27**, 348–356 (2011)
27. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. Proc. VLDB Endow. **6**(5), 289–300 (2013)
28. Wai, K.W., Ben, K., David, W.L.C., Rongbin, L., Siu, M.Y.: Secure query processing with data interoperability in a cloud database environment. In: Proc. of the SIGMOD (2014)
29. Wang, Z., Wang, W., Shi, B.: Fast query over encrypted character data in database. Commun. Inf. Syst. **4**(4), 289–300 (2004)
30. Wang, H., Cao, J., Zhang, Y.: A flexible payment scheme and its role-based access control. IEEE Trans. Knowl. Data Eng. **27**, 332–348 (2005)
31. Wang, H., Zhang, Y., Cao, J.: Effective collaboration with information sharing in virtual universities. IEEE Trans. Knowl. Data Eng. **21**, 840–853 (2009)
32. William, S.: Cryptography and Network Security: Principles and Practice, 6th edn. Pearson Education Limited (2013)
33. Wu, Z., Xu, G., Zong, Y., Yi, X., Chen, E., Zhang, Y.: Executing sql queries over encrypted character strings in the database-as-service model. Knowl.-Based Syst. **35**, 332–348 (2012)
34. Xu, H., Guo, S., Chen, K.: Building confidential and efficient query services in the cloud with rasp data perturbation. IEEE Trans. Knowl. Data Eng. **26**(2), 232–246 (2014)