Knowledge-Based Systems 35 (2012) 332–348

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Executing SQL queries over encrypted character strings in the Database-As-Service model

ZongDa Wu [a,c], GuanDong Xu [d], Zong Yu [b,c,*], Xun Yi [d], EnHong Chen [c], YanChun Zhang [d]

[a] Oujiang College, Wenzhou University, Wenzhou 325035, China
[b] West Anhui University, Luan 237012, China
[c] University of Science and Technology of China, Hefei 230016, China
[d] School of Engineering and Science, Victoria University, Melbourne, Australia

ARTICLE INFO

Article history:
Received 11 January 2012
Received in revised form 17 May 2012
Accepted 19 May 2012
Available online 6 June 2012

Keywords:
Database-As-Service
Data privacy and security
Encryption
Two-phase query
Characteristic index

ABSTRACT

Rapid advances in the networking technologies have prompted the emergence of the "software as service" model for enterprise computing, moreover, which is becoming one of the key industries quickly. "Database as service" model provides users power to store, modify and retrieve data from anywhere in the world, as long as they have access to the Internet, thus, being increasingly popular in current enterprise data management systems. However, this model introduces several challenges, an essential issue being how to implement SQL queries over encrypted data efficiently. To ensure data security, this model generally encrypts sensitive data at the trusted client's site, before storing them into the non-trusted database service provider's site, which, unfortunately, results in that SQL queries cannot be executed over the encrypted data immediately at the database service provider.

In this paper we only focus on how to query encrypted character strings efficiently. Our strategy is that when storing character strings to the database service provider, we not only store the encrypted character strings themselves, but also generate some characteristic index values for these character strings, and store them in an additional field; and when querying the encrypted character strings, we first execute a coarse query over the characteristic index fields at the database service provider, in order to filter out most of tuples not related to the querying conditions, and then, we decrypt the rest tuples and execute a refined query over them again at the client site. In our strategy, we define an n-phase reachability matrix for a character string and use it as the characteristic index values, and based on such a definition, we present some theorems to split a SQL 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. Finally, experimental results validate the functionality and effectiveness of our strategy.

1. Introduction

Rapid advance in enterprise informatization makes efficient data processing in enterprise routine activities more and more important, and thus most enterprises end up installing and maintaining database management systems to satisfy diversified data processing requirements. With the rapid increase of enterprise data, however, enterprise databases are becoming larger and more complicated, consequently, making the maintenance cost of database systems increasingly expensive. To decrease maintenance cost, a great number of enterprises begin to consider hiring specialized servers supplied by database service providers (that provide database storage and file access as services) and authorizing the providers to manage their databases. Such application requirement has fueled the emergence of a new database management model called "Database-As-Service" (or called DAS for short) [1]. In the DAS model, enterprises are allowed to store themselves databases into Database Service Providers (or called DSP for short), and to access the databases in the DSP through the Internet. In other words, the DAS model provides enterprises power to store, modify and retrieve information from anywhere in the world, as long as they have access to the Internet. Therefore, the DAS model allows enterprises to leverage hardware and software solutions that are provided by the DSP for data management, without having to develop them on their own. Perhaps more importantly, just as pointed out in [1], the DAS model provides a way for many enterprises to share the expertise of database professionals, thereby

* Corresponding author at: West Anhui University, Luan 237012, China.
E-mail addresses: zongda1983@163.com (Z. Wu), nick.zongy@gmail.com (Z. Yu).

0950-7051/$ - see front matter Crown Copyright © 2012 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2012.05.009

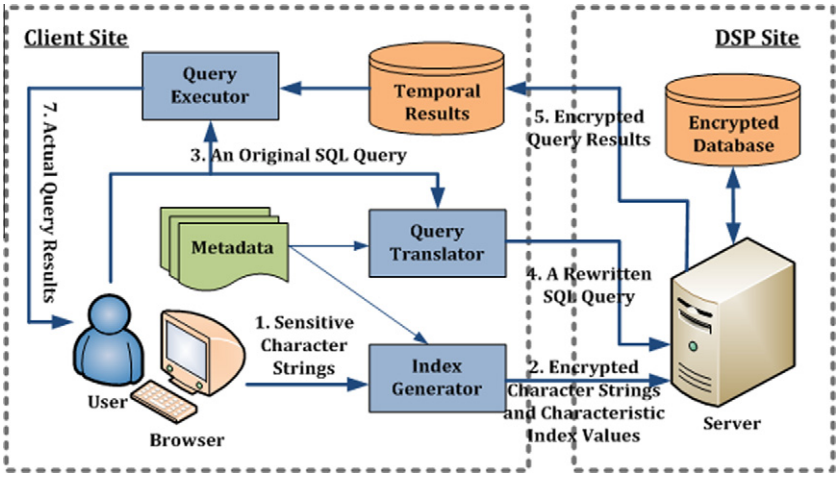

Fig. 1. Prototype architecture for executing SQL query over encrypted character strings in the Database-As-Service model.

decreasing the people cost of enterprises managing their database systems. All the advantages make the DAS model increasingly popular in current enterprise data management systems.

As described in [1–4], although cutting the cost of managing databases, from the technological angle, the DAS model poses many significant challenges foremost of which is the problem of data security. In the DAS model, the DSP is not trusted, where some DSP workers (e.g., database administrator) can access data stored in enterprise databases without any constraint, thus, making sensitive enterprise data possible to be stolen. Hence, the DAS model needs to provide more sufficient security measures to guard data privacy. It not only includes the protection from theft of sensitive enterprise data from external hackers to break into the DSP site and scan disks, but also includes the protection from the DSP itself. However, traditional database access strategies, such as user checking and access control, obviously no longer can satisfy such requirements of guarding sensitive data security in the DAS model. In order to guard data security in the DAS model, a straightforward way is to encrypt sensitive data in the DSP database, such that the encrypted data even if being stolen unfortunately cannot be understood [2,5,6]. However, if encrypted by using traditional encryption techniques [7,8], data may lose some important characteristics which the data themselves own originally, such as well-ordering and comparability.[1] In other words, to execute a SQL query over a database encrypted by using traditional encryption techniques, we have to decrypt all the encrypted data (which may be an encrypted table or database) and then execute the query over the decrypted data. Because the decryption operation of an entire table or database is time-consuming, the approach will extremely decrease the query performance in the DAS environment. Therefore, how to efficiently implement a SQL query over encrypted data in the DAS model is an important problem.

In this paper, we only concentrate on how to query encrypted character strings efficiently in the DAS model, because the query issue for other numerical data types has been discussed in [1–4] and solved successfully. Our proposed system, whose basic architecture and control flow are shown in Fig. 1, is comprised of some fundamental components. A **user** through a Web browser poses a SQL query to the client. A **server** is hosted by the DSP that stores the encrypted database. Before being submitted by a user to insert into the database at the DSP site, a character string has to be encrypted by an **index generator** and augmented with some additional information (called the **characteristic index**), which allows certain amount of query processing to occur at the DSP server without jeopardizing data privacy. A **client** stores the enterprise data into the DSP server, maintains **metadata** used for translating an original SQL query given by user to the representation over the encrypted database at the DSP site, and performs post-processing on temporal query results that are returned from the server. Based on the auxiliary metadata, we develop techniques to split an original SQL query defined over character strings into two parts: (1) a server-side query defined over the corresponding characteristic index to run on the DSP server, and (2) a client-side query for post-processing results of the server query. We achieve this goal by a **query translator**. A **query executor** at the client site is used first to decrypt the temporal results returned by the server-side query, and then to execute the client-side query (actually, which is the original query, shown as Fig. 1) over the decrypted data to return actual results of the user query.

From what mentioned above, we can find that, the primary issue in our work is how to generate good characteristic index values for sensitive character strings. Generally, a good characteristic index should satisfy the following three requirements. (1) **Good security**. The characteristic index values are stored into the DSP database as additional information on encrypted character strings, consequently, making them visible to the DSP workers. So, the characteristic index values should ensure themselves security, i.e., it is difficult to infer the original plaintexts from the index values. (2) **Good effectiveness**. Using the characteristic index, a query with any common type over character strings should be able to be translated to a new server-side query that can be executed immediately over the encrypted database at the DSP site, without needing to decrypt data. (3) **Good efficiency**. The temporal results returned by the server-side query should be as close to the real results as possible, to lighten the computation at the client site, consequently, improving the query performance in the DAS environment.

For the above three requirements, the first one could conflict with the last two ones to a certain extent. On the one hand, good security generally requires the characteristic index to store as little characteristic information on character strings as possible, making attackers impossible to obtain plaintexts based on the index values. On the other hand, good effectiveness and efficiency require that as much information as possible on character strings can be reflected out by the index values. So, a good characteristic index

[1] Although there are some data encryption techniques which can make the encrypted data still keep original ordering and comparability, they are generally of weak security, i.e., the data encrypted by these techniques are easy to be decrypted by statistics attack, as pointed out in [9].

may be required to make a compromise between security and effectiveness, i.e., it may be not able to perform perfectly over all the three requirements. However, a good characteristic index should better satisfy the first one, due to that, if unable to ensure security, it is not meaningful to introduce the characteristic index at all.

In this paper, we use a data structure called n-phase reachability matrix to generate the characteristic index values for sensitive character strings. In such an n-phase reachability matrix, we only store the key information on a character string, but discard the non-key information. By using the key information stored, many relationships (such as similar to each other) between character strings can be reflected out by their characteristic index values to a certain content, consequently, making that, many queries over character strings could be translated to server-side queries over the index values, i.e., making the index of good effectiveness. Besides, discarding the non-key information makes attackers difficult to infer plaintexts from the index values, i.e., making the index of good security. In this paper, we detail how to generate the n-phase reachability matrix for a given character string, and then, by using the n-phase reachability matrix as the characteristic index, we detail how to translate a query to its server-side representation for partitioning the computation of the query across the client and the server and thus improving query performance. Last, we also detail the experimental results of leveraging our approach to query encrypted character strings in the DAS model. The experimental results show that, the performance of executing a SQL query over the encrypted database in the DAS environment can be improved effectively by our approach, i.e., it is of good efficiency.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 shows how a character string is encrypted and stored into a database at the DSP site. Section 4 discusses how each type of SQL conditions over character strings is mapped into a new condition over the characteristic index values at the encrypted database. Section 5 presents a two-phase query algorithm that rewrites an original SQL query by splitting it into a server-side query and a client-side query, such that the computation at the client is decreased, and then analyzes the security of our proposed approach. Section 6 presents the experimental results of executing our approach over a randomly generated database. We last conclude this paper in Section 7.

2. Related work

Hakan et al. in [1] aiming at the issue of how to query encrypted numerical data in the DAS model, for the first time proposed to create the index for encrypted numerical data at the DSP database, and then translate each query to a new query defined over these index values. This approach first maps the domain of values of each sensitive attribute into several partitions and calls a hash function to assign a unique identifier to each partition, such that each value in the attribute, based on partition which this value belongs to, could be mapped to an identifier (called an index value); and then this approach redefines each query over these index values instead of original attribute values, such that the redefined query can be executed in the encrypted database at the DSP site immediately, without the need to decrypt the stored data. Later, the authors in [3] proposed to use the homomorphism techniques to enhance their approach, thus, making it become allowable to conduct an aggregation query operation over encrypted data. And, the authors in [4] further discussed the query optimization techniques about their approach, i.e., how to utilize multiple communications between the DSP server and the client to decrease the workload of the client as much as possible. The work given by Hakan et al. is very significant, which presented a basic framework of how to ensure data security in the DAS model. Especially, it presented two useful techniques to minimize the computation at the client, i.e., an algebraic framework to split the original query, and the two phase query algorithm, which have been widely referenced by other researchers in their related works [11,12]. In this paper, we also reference the two techniques (Sections 5.1 and 5.2). However, Hakan et al. did not analyze the security of their approach, and unfortunately, their approach is valid only for numerical data, i.e., it cannot solve the issue of querying character strings, a very important data type for enterprise computing.

To better implement range queries over encrypted data, Hore et al. in [10] presented a new approach to partitioning value domain, based on the solutions in [2]. By increasing the computation at the server as much as possible, so as to decrease the computation at the client, this approach can well improve the performance of querying encrypted data in the DAS model. Although maximizing the precision of range queries, this approach is also valid only for numerical data, and is useless for character strings.

Aiming to the problem of querying encrypted character strings in a database, Wang et al. in [11] proposed to turn a character string into characteristic values by using a characteristic function (or called a pairs coding function), so as to support similarity queries over encrypted character strings. The basic idea of this approach is as follows: when encrypting and storing a character string, based on the pairs information inherent in the character string itself, this approach constructs characteristic functions to turn the character string into the characteristic values and then stores them in an additional field; and when querying the encrypted character string, this approach firstly executes a coarse query over the additional field to filter out some irrelevant tuples, and then decrypts the rest tuples and executes a refined query over them again to generate actual results. This approach can reduce the scope of data decryption, and thus improve the querying performance. In this approach, however, there still are the following disadvantages: (1) the approach due to only considering the pairs information of a character string, makes it necessary to decrypt the entire table when querying the character strings only containing one character, resulting in decreasing the query performance; (2) the approach cannot well solve the similarity queries in the form of "LIKE '%s'" and "LIKE 's%'"; (3) the approach cannot solve the similarity queries over Chinese character strings; (4) the approach cannot solve the range queries over encrypted character strings; (5) owing to using only one pairs coding function to map each pair of a character string into one characteristic value, in the characteristic value there exists strong tendentiousness to '1' for each bit, making the approach difficult to withstand statistics attack or inference attack; (6) the approach did not present the way for computing the length of characteristic values; and (7) the approach did not present specific characteristic functions.

Furthermore, Wang et al. in [12] subsequently presented a new map approach based on flattening-disturbing function, which can make the '1' in each bit of characteristic value of better equality, thus improving data privacy and security. However, this approach may lead to time-consuming and space-consuming computation. This makes database update operations become expensive, resulting in a bad feasibility in real applications. By analyzing the traditional order-preserving encryption approach to numerical data, a fuzzy matching encryption approach aiming at character strings was proposed in [13]. In this approach, a character string is first transformed to numerical values, and an order-preserving encryption technique in [9] 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 [11,12], Cui et al. in [14] proposed to split index values into two parts: partition values and characteristic values, and use the partition values to support range queries. Essentially, this approach is to apply the numerical

data indexing approach given by Hakan et al. into the character strings, and combine with the approach given by Wang et al. Hence, this approach has the disadvantages of both Hakan and Wang, i.e., having worse data security.

In [15], Houmani et al. presented a novel approach to verify the secrecy property of cryptographic protocols under equational theories. In [16], inspired by the theory of artificial immune systems, Yang et al. presented a novel model of Agents of Network Danger Evaluation, which have been verified experimentally to have the features of real-time processing that provide a good solution for network surveillance. In [17], to improve security protocol analysis, Chen et al. proposed a formal framework to deal with the inconsistency in secure messages by weighting majority, consequently, making that people can verify protocols in an intuitive way and guarantees correct verification results. In [18], Yuan proposed an asymptotic secrecy model to secure communications between and within knowledge based systems over public channels. The proposed model is applied to solve some problems in wireless sensor networks as examples to show how the model can be applied for knowledge based systems in general. In addition, there are also other related encryption techniques for outsourced databases [19–21] or systems [22–24,31].

From what discussed above, we have the following two conclusions. On the one hand, although there are a number of research results about database encryption, most of which are not designed for the DAS model, making that, when querying encrypted data, we need first to decrypt the data and then query over the decrypted data, thereby not satisfying the requirement of querying performance in the DAS model. On the other hand, for existing approaches to querying encrypted data in the DAS model, there are many disadvantages, for example, they cannot well support to query over encrypted character strings. Therefore, how to execute SQL queries over encrypted character strings efficiently and effectively in the DAS environment is a problem urgent to be solved.

3. Encrypting character string

Before we discuss techniques for encrypting character strings, let us first show how the encrypted character strings and their additional characteristic index values are stored into the DSP database. The scheme we below used is similar to that mentioned in [2,11].

For each relation $R(A_1, A_2, \ldots, A_t, \ldots, A_m)$, where A_t is a sensitive field with character string type, which needs to be encrypted, we store at the DSP database an encrypted relation as follows:

$$R^S\left(A_1, A_2, \ldots, A_t^E, \ldots, A_m, A_t^S\right)$$

where the attribute A_t^E stores an encrypted string (we will explain how A_t^E is defined in Section 3.3) that corresponds to the sensitive attribute A_t in the original relation R; while the attribute A_t^S corresponds to the characteristic index for A_t that will be used for query processing at the DSP server. The remaining fields are identical to those of the original relation. For example, let us consider a relation *std* that stores information about students, shown as Table 1, where the field marked with shadow is sensitive and needs to be encrypted. The relation *std* is mapped to a corresponding encrypted relation stored at the DSP database, given as follows:

$$std^S(id, name, sex, age, addr^E, cid, addr^S)$$

where the attribute $addr^E$ stores the encrypted bit string corresponding to the field *addr*, while the attribute $addr^S$ stores corresponding characteristic index values.

In this section, we mainly discuss how to create the characteristic index values for a character string, to make that (1) the characteristic index can support various queries over character strings efficiently, and (2) the index itself is safe, namely, it is difficult for attackers to obtain original character strings from the index values. Specifically, we first introduce a concept called *n*-phase reachability matrix; second, we explain how a character string is mapped into an *n*-phase reachability matrix; and last, we present the detailed scheme about how to use the *n*-phase reachability matrix as the characteristic index to store sensitive character strings into the DSP database. Moreover, to simplify presentations, we assume that there exists only one sensitive field $R.A_t$ of character string type in the relation R.

Table 1
A relation *std* that contains a sensitive filed of character string type.

Id	Name	Sex	Age	Addr	cid
01092125	John	Male	24	China	12
01092126	Mary	Female	23	California	13
01092127	Tony	Male	26	Florida	10
01092128	Lucy	Female	22	Massachusetts	11

3.1. Reachability matrix

The goal of creating characteristic index values for every sensitive character string is to make each SQL query defined over the character strings transformed into a new SQL query defined over the characteristic index values, such that the new query can be executed immediately at the DSP server, without having to decrypt encrypted character strings. For this purpose, the characteristic index values should reflect some useful information of their corresponding character strings. However, these characteristic index values themselves should be as safe as possible, for preventing attackers from obtaining original character strings based on the index values. To meet the two purposes, we introduce an *n*-phase reachability matrix, used as the characteristic index of character strings. In this section, we first present the definition of an *n*-phase reachability matrix. In subsequent sections, we will step by step show the advantages of leveraging the *n*-phase reachability matrixes as the characteristic index, i.e., based on the *n*-phase reachability matrix, how various queries about character strings are implemented at the DSP server (Section 4), and how data security is guaranteed (Section 5.3).

Definition 1. Let U:$\{e_1, e_2, \ldots, e_m\}$ be a set consisting of m elements ($1 \leqslant m$), and Q:$q_1q_2\cdots q_t$ be a sequence ($1 \leqslant t$) defined over U (i.e.,$\forall i(1 \leqslant i \leqslant t,\ q_i \in U)$). Then, the **n-phase connected element pairs** ($1 \leqslant n$) over the sequence Q are defined as follows:

$$P^n(Q)[U] = \begin{cases} \{(q_i, q_{n+i}) | i = 1, 2, \ldots, t-n\}, & \textbf{if } t \geqslant (n+1) \\ \emptyset, & \textbf{otherwise} \end{cases}$$

and, let f_1:$U \to \{1, 2, \ldots, m\}$, f_2:$U \to \{1, 2, \ldots, m\}$ and f_3:$U \to \{1, 2, \ldots, m\}$ be three bijections. Then, the **n-phase reachability matrix** over the sequence Q is defined as follows:

$$M^n(Q)[U] = (a_{ij})_{(m+1)\times(m)}$$

where (1) for $1 \leqslant i,\ j \leqslant m$, $a_{ij} = 1$ if $(f_1^{-1}(i), f_2^{-1}(j)) \in (P^1(Q)[U] \cup P^2(Q)[U] \cup \ldots \cup P^n(Q)[U])$; (2) for $(1 + m) = i$ and $1 \leqslant j \leqslant m$, $a_{ij} = 1$ if $f_3^{-1}(j) = q_1$ or $f_3^{-1}(j) = q_t$; and (3) $a_{ij} = 0$, otherwise.

From Definition 1, we can obtain the following properties: (1) each *n*-phase reachability matrix is a Boolean matrix; (2) for a sequence, the size of its *n*-phase reachability matrix is determined by the set over which the sequence defined, instead of being determined by the sequence itself (i.e., for any two sequences over the same set, their reachability matrixes are of the same size); (3) each *n*-phase reachability matrix uses its first m rows to store the

information about the connected element pairs contained in its corresponding sequence; and (4) each n-phase reachability matrix uses its last row to point out the front element and the tail element of its corresponding sequence. We below illustrate the construction of the n-phase reachability matrix over a sequence.

Example 1. Assume that there are U:{5, 7, 18, 9}, a set consisting of four elements, and Q: 5 5 9 18, a sequence over U. And, for simplifying presentations, we let $f_1 = f_2 = f_3$; $f_1(5) = 1$; $f_1(7) = 2$; $f_1(18) = 3$; and $f_1(9) = 4$. Then, based on Definition 1, we can construct the 1-phase reachability matrix, the 2-phase reachability matrix and the 3-phase reachability matrix, over the sequence Q, respectively given as follows:

$$M^1(Q)[U] = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad M^2(Q)[U] = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$M^3(Q)[U] = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

In Definition 1, we use the three bijections (i.e., f_1, f_2, and f_3) to generate the matrix, which not only aims to make the definition more general, but also aims to enhance data security. If without the bijections, the information about the connected element pairs in a sequence would be reflected out immediately by the n-phase reachability matrix over the sequence, consequently, increasing the probability of sensitive data leakage. For example, assume without using the bijections, i.e., $a_{ij} = 1$, if $(e_i, e_j) \in (P^1(Q)[U] \cup P^2(Q)[U] \cup \cdots \cup P^n(Q)[U])$. If $a_{ij} = 1$ in the 1-phase reachability matrix over a sequence, then we can conclude that the 1-phase connected element pair (e_i, e_j) should be contained in the sequence, and we can further infer all the 1-phase element pairs contained in the sequence. However, such a problem can be solved by the bijections to a certain extent. Specifically, the bijections are first generated through using a collision-free hash function or a random way, and second, the metadata about these bijections is stored into the trusted client site (see Fig. 1), namely, which is difficult to be obtained by other persons. Such a way makes attackers difficult to infer the element pairs contained in an n-phase reachability matrix.

Besides, the attackers even if having known all the element pairs contained in a sequence, are difficult to infer the original sequence based on these element pairs, because the order of the element pairs is not reflected out by the reachability matrix. For example, if having known the bijections used in Example 1, the attackers based on the 1-phase reachability matrix can infer that the following 1-phase element pairs: (5, 5), (5, 9) and (9, 18) should be contained in the sequence, but they do not know how many times each element pair appears in the sequence, and how the element pairs are arranged, so they cannot infer the original sequence. Therefore, the n-phase reachability matrix has better security. A more detailed security analysis about an n-phase reachability matrix is presented in Section 5.3.

3.2. Mapping character string to reacability matrix

We observe that any character string can be viewed as a sequence defined over a set consisting of corresponding characters, so based on Definition 1, we can construct the n-phase reachability matrix for a character string, and then use the matrix as the characteristic index values of the character string. However, the set of characters over which a string is defined, generally is relatively large, consequently, making the corresponding reachability matrix space-consuming. For example, if a character string is defined over the ASCII set that is of size 2^8, then any n-phase reachability matrix constructed based on the character string would be of size $(2^8 + 1) \times 2^8$; and furthermore, if defined over the Unicode set of size 2^{16}, then it would lead to the larger reachability matrix that is of size $(2^{16} + 1) \times 2^{16}$. Hence, it is not suitable to use the reachability matrix that is generated immediately from a string over a larger set of characters, as the characteristic index.

To solve this, for any given string $Q_C = q_1, q_2, \ldots, q_t$, defined over a set U_C of characters, we consider mapping the set U_C into a new set U_D with smaller size, noted as: $U_D = map_{set}(U_C)$; and then mapping the string Q_C into a new sequence Q_D defined over U_D, noted as: $Q_D = map_{seq}(Q_C)$. In this mapping process for generating a new sequence, we use a method which is similar to what mentioned in [2] for indexing numerical data. Specifically, we first map the set U_C into many smaller partitions: $\{u_1, u_2, \ldots, u_k\}$, such that, (1) these partitions taken together cover the whole set U_C, i.e., $U_C = (u_1 \cup u_2 \cup \cdots \cup u_k)$; and (2) any two partitions do not overlap, i.e., $(u_i \cap u_j) = \emptyset$, for $1 \leqslant i, j \leqslant k$ and $i \neq j$. Second, we assign each partition $u_i (i = 1, 2, \ldots, k)$ with an integer identifier d_i which is different from that of any other partition, i.e., $d_i \neq d_j$, for $1 \leqslant i, j \leqslant k$ and $i \neq j$, thereby, forming a new set: $U_D = map_{set}(U_C) = \{d_1, d_2, \ldots, d_k\}$. Third, we map each character q_i in the string Q_C, based on the partition that the character belongs to, into an integer $g(q_i) (g(q_i) \in U_D$, i.e., the partition identifier), thereby, generating a new sequence: $Q_D = map_{seq}(Q_C) = g(q_1)g(q_2)\cdots g(q_t)$, defined over the set U_D. Based on the sequence Q_D, we construct the n-phase reachability matrix as follows:

$$M^n(Q_D)[U_D] = M^n(map_{seq}(Q_C))[map_{set}(U_C)]$$

Last, we use the above matrix as the character index values of the character string Q_C, instead of the original reachability matrix constructed from the character string itself immediately.

In the above process, if we let the size of each partition greater than 1, i.e., $|u_i| > 1$, for $i = 1, 2, \ldots, k$, then $|U_C| > |U_D|$, and $|M^n(Q_C)[U_C]| > |M^n(Q_D)[U_D]|$. Furthermore, the greater size each partition has, the smaller size the corresponding reachability matrix has. Therefore, through setting a greater size for each partition, we can generate the n-phase reachability matrix of smaller size for any character string, resulting in consuming less space. We below use a simple example to explain the detailed mapping process from a character string to such a new n-phase reachability matrix.

Example 2. For the ASCII set U_C, consisting of 2^8 elements, and Q_C: China, a character string over the set U_C, we below show how to create the n-phase reachability matrix for the character string Q_C. First, we divide the set U_C into five smaller partitions as follows:

$$u_1 = [a, f]; \quad u_2 = [g, j]; \quad u_3 = [k, p]; \quad u_4 = [q, z]; \text{ and}$$
$$u_5 = U_1 - (u_1 \cup u_2 \cup u_3 \cup u_4)$$

In this process, we use a random way to partition the set of characters; and of course, other partitioning techniques such as Equi-Width and Equi-Depth [21] can also be used. In our evaluation experiments discussed in Section 7, we use the Equi-Depth to partition the set of characters, making that, randomly given a character string, the sum of probability values for each character in a partition appearing in the character string, is approximately equal to that of any other partition.

Second, we assign each partition $u_i (i = 1, 2, \ldots, 5)$ with an integer identifier i, i.e., $(u_1 \to 1)$; $(u_2 \to 2)$; $(u_3 \to 3)$; $(u_4 \to 4)$ and

($u_5 \to 5$), resulting in forming a new set $U_D = map_{set}(U_C) = \{1, \ldots, 5\}$. In this process, any two partitions should be assigned with different identifiers. Here, we use a simple way to assign partition identifiers. However, in real application, we can use a collision-free hash function mentioned in [2], and store the corresponding metadata into the trusted client site, so as to enhance data security. Third, we map each character in the string Q_C: China, to an integer (equal to the identifier of the partition at which the character is located), thereby, generating a new sequence : $Q_D = map_{seq}(Q_C) = 5\ 2\ 2\ 3\ 1$. Fourth, let $f_1 = f_2 = f_3$; $f_1(1) = 1$; $f_1(2) = 2$; $f_1(3) = 3$; $f_1(4) = 4$; and $f_1(5) = 5$. Based on Definition 1, we construct the 1-phase reachability matrix, the 2-phase matrix, the 3-phase matrix and the 4-phase matrix, for the new sequence Q_D over the new set U_D, given as follows:

$$M^1(Q_D)[U_D] = M^1(map_{seq}(Q_C))[map_{set}(U_C)] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M^2(Q_D)[U_D] = M^2(map_{seq}(Q_C))[map_{set}(U_C)] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M^3(Q_D)[U_D] = M^3(map_{seq}(Q_C))[map_{set}(U_C)] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M^4(Q_D)[U_D] = M^4(map_{seq}(Q_C))[map_{set}(U_C)] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Last, the above four reachability matrixes are used as the new matrixes of the character string Q_C over the set U_C. We choose any of them as the characteristic index of the string Q_C, instead of the n-phase reachability matrixes constructed from the character string immediately. In real application, it is important to choose an appropriate n-phase reachability matrix as the index for a character string. Generally, the greater "n" would lead to the better data security (see Section 5.3 for detail), but the greater false rate (reference Definition 4). In our system, "n" is a parameter that can be set by users according to their real application.

For a sensitive character string in the relation R, after using the above approach to generate the n-phase reachability matrix (i.e., the characteristic index) of the string, we need to store the matrix into the corresponding additional attribute $R^S.A_t^S$ at the DSP database. To simplify such storage, we map the reachability matrix into a string of bits, through using a way of prior mapping matrix rows or a way of prior mapping matrix columns. Here, prior mapping matrix rows denotes that we first map the first row of the matrix into a string of bits, then map the second row, the third row and so forth, and last we combine these bit strings. For example, if prior mapping matrix rows, we can obtain a bit string from the 1-phase reachability matrix in Example 2 as follows: "00000 01100 10000 00000 01000 10001"; and similarly, if prior mapping matrix columns, we obtain another bit string as follows: "001001 010010 010000 000000 000001". Therefore, in order to store the reachability matrix in Example 2, we need to spend 30 bits (i.e., 4 bytes) space. More generally, given a set U of characters, and any character string Q defined over U, in order to store the n-phase reachability matrix constructed based on this string Q, the space that needs to be consumed is $(|map_{set}(U)| + 1) \times (|map_{set}(U)|)$ bits or $(|map_{set}(U)| + 1) \times (|map_{set}(U)|)/8$ bytes. This shows that (1) for all the character strings over the same set U, their index values would have the same size; and (2) if the set U is divided into many partitions, resulting in the set $map_{set}(U)$ of greater size, then the index values will consume a larger storage. In real application, we allow users to set the number of divided partitions, although we recommend 128 as the default value. We below use $str_{bit}(M)$ to denote the bit string generated from the n-phase reachability matrix M through using the way of prior mapping matrix rows.

3.3. Storing encrypted character strings

We now have enough notations to describe how to store the encrypted relation into the DSP database. Given any tuple $t = (a_1, a_2, \ldots, a_t, \ldots, a_m)$ over the relation: $R(A_1, A_2, \ldots, A_t, \ldots, A_m)$, where a_t is a sensitive character string defined over a character set U, the corresponding encrypted relation: $R^S\left(A_1, A_2, \ldots, A_t^E, \ldots, A_m, A_t^S\right)$, in the DSP database stores an encrypted tuple t^S as follows:

$$t^S = (a_1, a_2, \ldots, \textbf{\textit{encrypt}}(a_t), \ldots, a_m, str_{bit}(M^n(map_{seq}(a_t))[map_{set}(U)]))$$

where ***encrypt*** is a function used to encrypt a character string. We treat the encryption function as a black box. Some traditional data encryption techniques such as AES [26], RSA [27], Blowfish [28], and DES [29] can be used to encrypt sensitive character strings. In Table 2, we show the encrypted relation std^S that corresponds to the relation std in Table 1, and is stored at the DSP database.

In Table 2, the column $std^S.addr^E$ stores the encrypted bit strings corresponding to the sensitive attribute $std.addr$, which is generated by the encryption function. For instance, the character string "China" is encrypted to "11011000…" (as shown at the first row and the fifth column in Table 2) that is equal to ***encrypt***("China"). Moreover, the column $std^S.addr^S$ corresponds to the index strings over $std.addr$, and we here use the approach identical to that in

Table 2
An encrypted relation std^S that contains encrypted strings and index values.

Id	Name	Sex	Age	$Addr^E$	cid	$Addr^S$
01092125	John	Male	24	*11011000…*	12	*00000 01100 10000 00000 01000 11101*
01092126	Mary	female	23	*00101010…*	13	*01000 11100 00010 01000 10000 11101*
01092127	Tony	Male	26	*10110110…*	10	*10000 11000 00000 01000 01000 11101*
01092128	Lucy	female	22	*11001000…*	11	*11010 00010 00000 10010 10000 11101*

Example 2 to construct the 1-phase reachability matrix as the characteristic index. For instance, the index values "00000 01100 10000 00000 01000 11101" (as shown at the first row and the seventh column in Table 2) corresponding to "*China*" is equal to the 1-phase reachability matrix given in Example 2.

In our system, the task about encrypting sensitive character strings and generating their characteristic index values are all completed at the trusted client site by the index generator component; and then the encrypted character strings and their index values are transmitted through the Internet and stored into the DSP database (see Fig. 1).

4. Mapping query conditions

We in this section study how to translate specific query conditions over sensitive attributes of character string type into the conditions over the corresponding characteristic index attributes in the DSP database. Once we know how query conditions are translated, we will in Section 5 discuss how to implement efficiently queries over encrypted character strings, i.e., how to split an original SQL query posted by user and over unencrypted character strings into (1) a server-side query that can be executed in the DSP server immediately, without having to decrypt tuples, and (2) a client-side query for post-processing results returned by the server query.

For each relation, the DSP server stores the encrypted tuples, along with the characteristic index values for sensitive attributes of character string type. Meanwhile, the client stores the metadata about the specific characteristic indices, such as the information about the partitioning of character set, the mapping functions and so on (see Sections 3.1 and 3.2). Then, the client uses this information to translate a SQL query posted by user into its server-side representation (i.e., the server-side query, executed by the DSP server). We mainly consider two types of query conditions defined over unencrypted character strings, i.e., similarity conditions (e.g., $R.A_t$ *LIKE* "%abc%") and range conditions (e.g., $R.A_t >$ "abc"). Besides, we assume that all the character strings are defined over the same character set U, so for simplicity, we below use the notation $M^n(Q)$, instead of $M^n(Q)[U]$, to denote the n-phase reachability matrix constructed based on the character string Q.

In this section, we first in Section 4.1 present related definitions and theorems; and through using them as a theoretical basis, we then in Section 4.2 introduce how a query condition related to character strings is translated into its server-side representation.

4.1. Definitions and theorems

In the encrypted relation, each characteristic index value (i.e., n-phase reachability matrix) is generated based on a character string, such that the index value could reflect some useful information about the original character string itself. Therefore, for any two character strings between which there is a relationship (e.g., similar to each other), such a relationship may be also reflected by their corresponding n-phase reachability matrixes. In other words, the judgment of whether there is a relationship between two character strings (e.g., a character string whether similar with another), may be achieved through checking the reachability matrixes of the two character strings. We below introduce some related definitions and theorems to confirm the above supposition.

Definition 2. Let U be a set consisting of m elements ($1 \leqslant m$), $Q{:}q_1q_2\cdots q_t$ be a sequence ($1 \leqslant t$) over U, and $f_1{:}U \to \{1, 2, \ldots, m\}$, $f_2{:}U \to \{1, 2, \ldots, m\}$, and $f_3{:}U \to \{1, 2, \ldots, m\}$ be three bijections. Then, three new **n-phase reachability matrixes** over the sequence Q are defined as follows:

$M^n_{\prec}(Q)[U] = (a_{ij})_{(m+1)\times(m)}$ $M^n_{\succ}(Q)[U] = (b_{ij})_{(m+1)\times(m)}$
$M^n_{+}(Q)[U] = (c_{ij})_{(m+1)\times(m)}$

where (1) when $1 \leqslant i,\ j \leqslant m,\ a_{ij} = b_{ij} = c_{ij} = 1$, if $(f_1^{-1}(i), f_2^{-1}(j)) \in (P^1(Q)[U] \cup P^2(Q)[U] \cup \ldots \cup P^n(Q)[U])$; (2) when $(1 + m) = i$ and $1 \leqslant j \leqslant m,\ a_{ij} = 1$, if $f_3^{-1}(j) = q_1$, and $b_{ij} = 1$, if $f_3^{-1}(j) = q_t$; and (3) $a_{ij} = 0$, $b_{ij} = 0$, $c_{ij} = 0$, otherwise.

From Definition 2, we see that the three new defined n-phase reachability matrixes are similar to that in Definition 1, except having different last rows, where: (1) the first matrix uses its last row to point out the front element in the sequence Q, which is called an **n-phase reachability matrix with front information**; (2) the second one uses its last row to point out the tail element, which is called an **n-phase reachability matrix with tail information**; and (3) in the third one, each element of its last row is equal to '0', so it is called an **n-phase reachability matrix without front and tail information**. We uniformly call these three matrixes as **incomplete n-phase reachability matrixes** over the sequence Q. In Example 3, we show the construction of the three incomplete n-phase reachability matrixes over a sequence.

Example 3. Assume that there are U:{5, 7, 18, 9}, a set consisting of four elements, and Q: 5 5 9 18, a sequence over U. Let $f_1 = f_2 = f_3$; $f_1(5) = 1$; $f_1(7) = 2$; $f_1(18) = 3$; and $f_1(9) = 4$. Then, based on Definition 2, we construct the three incomplete 1-phase reachability matrixes over the sequence Q, respectively given as follows:

$$M^1_{\prec}(Q)[U] = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad M^1_{\succ}(Q)[U] = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$M^1_{+}(Q)[U] = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Theorem 1. *Given two character strings $Q_A{:}A_1A_2\cdots A_a (1 \leqslant a)$ and $Q_B{:}B_1B_2\cdots B_b$ $(1 \leqslant b)$ over the same set, assuming that $\exists i (1 \leqslant i \leqslant b, A_1 = B_i, A_2 = B_{i+1}, \ldots, A_a = B_{i+a-1})$, we conclude that*

(1) *If $1 = i$, i.e., Q_A appears in the front of Q_B, then: (where $\wedge$ represents a "bit and" operation)*

$$str_{bit}\left(M^n_{\prec}(map_{seq}(Q_A))\right) = str_{bit}\left(M^n_{\prec}(map_{seq}(Q_A))\right) \wedge str_{bit}(M^n(map_{seq}(Q_B))) \quad (1)$$

(2) *If $b - a + 1 = i$, i.e., Q_A appears in the tail of Q_B, then:*

$$str_{bit}\left(M^n_{\succ}(map_{seq}(Q_A))\right) = str_{bit}\left(M^n_{\succ}(map_{seq}(Q_A))\right) \wedge\ str_{bit}(M^n(map_{seq}(Q_B))) \quad (2)$$

(3) *If $b - a + 1 > i > 1$, i.e., Q_A appears in the middle of Q_B, then:*

$$str_{bit}\left(M^n_{+}(map_{seq}(Q_A))\right) = str_{bit}\left(M^n_{+}(map_{seq}(Q_A))\right) \wedge str_{bit}(M^n(map_{seq}(Q_B))) \quad (3)$$

Proof. We here only prove Eq. (1), and the other two can be proven similarly. Because Q_A is a substring of Q_B, and they are defined over the same set U, based on what introduced in Section 3.2, we conclude that, $map_{seq}(Q_A)$ is also a subsequence of $map_{seq}(Q_B)$, and both are defined over the same set map_{set} (U). Then, based

on the description about n-phase connected element pairs in Definition 1, we conclude that, $P^n(map_{seq}(Q_A))[map_{set}(U)] \subseteq P^n(map_{seq}(Q_B))[map_{set}(U)]$; and based on the description about n-phase reachability matrix in Definition 1, we can further conclude that for any element in the matrix $M^n_{\prec}(map_{seq}(Q_A))[map_{set}(U)]$, if it is equal to '1', then its corresponding element (namely, located at the same row and the same column) in the reachability matrix $M^n(map_{seq}(Q_B))[map_{set}(U)]$ is also equal to '1'. Therefore, the Eq. (1) is correct. □

Actually, Theorem 1 presents an insufficient but necessary condition for that a character string is contained in another. Therefore, the judgment of whether a character string is contained in another can be achieved by checking the n-phase reachability matrixes, namely (1) a character string is not contained in another, if their reachability matrixes cannot meet Eqs. (1), (2) or (3); and (2) it is likely that a character string is contained in another, otherwise.

In SQL [30], a character string match pattern (such as "ABC[D-F]") represents a series of character strings with similar feature, which is important to execute similarity queries over the character strings. Below, we present a definition on the matrix (called **match matrix**) over a character string match pattern in the form of "ABC[D-F]" or "[A-B]CDE". Then, we present a theorem to show that, it can be reflected out to some extent by the match matrix and the reachability matrix that a character string whether being an instance of a match pattern.

Definition 3. Let be U a set consisting of m elements ($1 \leqslant m$), $Q{:}q_1q_2\cdots q_t$ be a sequence over U ($1 \leqslant t$), $S \subseteq U$, QS be a sequence match pattern over U, each of whose instances consists of Q connecting an element in S, and SQ be another sequence match pattern, each of whose instances consists of an element in S connecting Q. And, let $f_1{:}U \to \{1,2,\ldots,m\}$, $f_2{:}U \to \{1,2,\ldots,m\}$; and $f_3{:}U \to \{1,2,\ldots,m\}$ be three bijections. Then, the two **match matrixes** over the match pattern QS and SQ respectively, are defined as follows:

$$M(QS)[U] = (a_{ij})_{(m+1)\times(m)} \quad M(SQ)[U] = (b_{ij})_{(m+1)\times(m)}$$

where (1) for $1 \leqslant i, j \leqslant m$, $a_{ij} = 1$ if and only if $(f_1^{-1}(i), f_2^{-1}(j)) \in \{(q_t, q) | q \in S\}$, and $b_{ij} = 1$ if and only if $(f_1^{-1}(i), f_2^{-1}(j)) \in \{(q, q_1) | q \in S\}$; (2) for $(1+m) = i$ and $1 \leqslant j \leqslant m$, $a_{ij} = 1$ if and only if $f_3^{-1}(j) = q_1$, and $b_{ij} = 1$ if and only if $f_3^{-1}(j) = q_t$; and (3) $a_{ij} = 0$, $b_{ij} = 0$, otherwise.

From Definition 3, we can conclude that, (1) for any sequence match pattern, the size of its match matrix is determined by the set, which the match pattern is defined over, so the size of the match matrix is identical to that of the reachability matrix of a sequence over the same set; and (2) for all the sequences that can be generated from the match pattern QS, all their last element pairs (i.e., the element pairs appearing in the tail of a sequence) are stored into the match matrix $M(QS)[U]$. The above two observations are important for Theorem 2 given below.

In order to generate the match matrix for a character string match pattern, we use the same method mentioned in Section 3.2. For a given character set U and a character string match pattern QS (or SQ) over U (where Q is a character string over U and $S \subseteq U$), we first map U into a new integer set $map_{set}(U)$ through set partitioning; then, we map each character in S, based on the partition that the character belongs to, into an integer (namely, the partition identifier), consequently, generating a new set S' (obviously, $S' \subseteq map_{set}(U)$); and last, we map the character string Q into a new integer sequence $map_{seq}(Q)$. Finally, we obtain a new sequence match pattern, which is comprised of $map_{seq}(Q)$ and S', and is defined over the new set $map_{set}(U)$. We use $map_{prn}(QS)$ to denote the new generated sequence match pattern.

Example 4. For the ASCII set U_1, and, QS (where Q = China and $S = [a, q)$) and SQ, two match patterns defined over U_1, if using the same partitioning method in Example 2, then we can obtain two new sequence match patterns: $Q'S'$, where $Q' = 5\ 2\ 2\ 3\ 1$ and $S' = [1, 2]$, and $S'Q'$, defined over the new integer set $U_2 = map_{set}(U_1)$. Then, from the new generated two match patterns, we construct their match matrixes, respectively given as follows:

$$M(Q'S')[U_2] = M(map_{prn}(QS))[map_{set}(U_1)] = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M(S'Q')[U_2] = M(map_{prn}(SQ))[map_{set}(U_1)] = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Theorem 2. *Given two character strings Q_1 and Q_2 defined over the same character set U, and a character set S ($S \subseteq U$), we conclude that*

(1) *If Q_2 is an instance of the character string match pattern Q_1S, then*

$$str_{bit}(M(map_{seq}(Q_1S)) \wedge str_{bit}(M^n(map_{seq}(Q_2))) \neq 0 \textbf{ AND}$$
$$str_{bit}(M^n_{\prec}(map_{seq}(Q_1))) = str_{bit}(M^n_{\prec}(map_{seq}(Q_1))) \wedge str_{bit}(M^n(map_{seq}(Q_2))) \quad (4)$$

(2) *If Q_2 is an instance of the character string match pattern SQ_1, then*

$$str_{bit}(M(map_{seq}(SQ_1))) \wedge str_{bit}(M^n(map_{seq}(Q_2))) \neq 0 \textbf{ AND}$$
$$str_{bit}(M^n_{\succ}(map_{seq}(Q_1))) = str_{bit}(M^n_{\succ}(map_{seq}(Q_1))) \wedge str_{bit}(M^n(map_{seq}(Q_2))) \quad (5)$$

Proof. We here only prove Eq. (4), and the other one can be proven similarly. In Eq. (4) there are two equations: (1) For the second equation, because Q_2 is an instance of the match pattern Q_1S, we conclude that Q_1 is a substring of Q_2, and Q_1 appears in the front of Q_2; and then based on Theorem 1, we further conclude that the equation is correct. (2) For the first equation, we assume that it is not correct. From such an assumption, we conclude that there is not any character string in the set: {AB | A is the last character in Q_1, and B belongs to S.}, appearing in Q_2, which conflicts with the known condition that Q_2 is an instance of the match pattern Q_1S. Therefore, the assumption is not tenable, i.e., the first equation is correct. □

Actually, Theorem 2 presents an insufficient but necessary condition for that a character string is an instance of a match pattern, so the judgment of a character string whether or not being an instance of a character string match pattern can be achieved through checking some related n-phase reachability matrixes and related match matrixes, namely (1) a character string cannot be generated from a character string match pattern, if their related matrixes cannot meet Eq. (4) or (5); and (2) it is likely that the character string is an instance of the match pattern, otherwise.

4.2. Mapping query conditions

Based on the definitions and theorems given in Section 4.1, we below discuss how query conditions over character string attributes are mapped to their server-side representations, which are defined over the corresponding characteristic index attributes in the server-side encrypted relation, and thus can be performed by the DSP database immediately. Below, we call the process of mapping a query condition to its server-side representation as condition mapping for short, and use $trans_c$ to denote such a condition mapping.

Mapping 1. $R_1 \cdot A_t = R_2 \cdot A_d$: Such a query condition, which is related to two attributes in two relations, arises in a join operation, where $R_1.A_t$ and $R_2.A_d$ are both sensitive attributes of character string type. The condition mapping is defined as follows:

$$trans_c(R_1.A_t = R_2.A_d) \Rightarrow R_1^S.A_t^S = R_2^S.A_d^S$$

In our system, all the characteristic index values (which may belong to different index attributes or different encrypted relations) are generated by using the same mapping approach (see Section 3), consequently, making that, any two character strings, which are identical to each other, would be mapped to the same n-phase reachability matrix, namely, they would have the same characteristic index value.

Mapping 2. $R.A_t = Q$: Such a query condition, which is related to only one attribute, arises in a selection operation, where Q is a character string constant, and $R.A_t$ is a sensitive attribute of character string type. Its condition mapping is defined as follows:

$$trans_c(R.A_t = Q) \Rightarrow str_{bit}(M^n(map_{seq}(Q))) = R^S.A_t^S$$

The above mapping is relatively simple. We only need to generate the characteristic index value for the character string constant contained in the query condition, using the approach identical to that used to generate the index values for the sensitive attribute of character string type.

Mapping 3. $R.A_t LIKE \%Q\%$: Such a query condition appears in the LIKE clause, where "%" is a wildcard [30] which represents to match a character string of any size. For example, "%ABC" can match any character strings ended with "ABC". From Theorem 1, we have known that, the judgment of any character string whether being contained in another can be achieved by checking the n-phase reachability matrixes of the two character strings. Therefore, for three LIKE query conditions in the form of "$R.A_tLIKE\%Q\%$", "$R.A_tLIKE\ Q\%$" and "$R.A_tLIKE\%Q$", we can define their condition mappings, respectively given as follows:

$$trans_c(R.A_tLIKE\%Q\%) \Rightarrow str_{bit}\left(M_+^n(map_{seq}(Q))\right) = str_{bit}\left(M_+^n(map_{seq}(Q))\right) \wedge R^S.A_t^S$$

$$trans_c(R.A_tLIKEQ\%) \Rightarrow str_{bit}\left(M_\prec^n(map_{seq}(Q))\right) = str_{bit}\left(M_\prec^n(map_{seq}(Q))\right) \wedge R^S.A_t^S$$

$$trans_c(R.A_tLIKE\%Q) \Rightarrow str_{bit}\left(M_\succ^n(map_{seq}(Q))\right) = str_{bit}\left(M_\succ^n(map_{seq}(Q))\right) \wedge R^S.A_t^S$$

Besides, for similar LIKE conditions described by using another wildcard "_", which represents to match only one character (e.g., "_ABC" can match "AABC"), we can define their condition mappings identical to the three ones mentioned above.

Mapping 4. $R.A_tLIKE\ Q[A_1 - A_2]$: Such a condition appears in the LIKE clause, where A_1 and A_2 are both characters and $A_1 \leqslant A_2$, and "[]" is a wildcard which is used to match only one character located in a given domain. For example, "ABC[D-E]" can match the following two character strings: "*ABCD*" and "*ABCE*". This query condition is mainly used to check whether a character string can be generated by a given match pattern. Therefore, for two given LIKE query conditions in the form of "$R.A_tLIKEQ[A_1 - A_2]$" and "$R.A_tLIKE[A_1 - A_2]Q$", based on Theorem 2, we can define their condition mappings, respectively given as follows:

$$trans_c(R.A_tLIKEQ[A_1 - A_2]) \Rightarrow 0 \neq str_{bit}(M(map_{seq}(Q[A_1 - A_2]))) \wedge R^S.A_t^S \textbf{ AND } str_{bit}\left(M_\prec^n(map_{seq}(Q))\right) = str_{bit}\left(M_\prec^n(map_{seq}(Q))\right) \wedge R^S.A_t^S$$

$$trans_c(R.A_tLIKE[A_1 - A_2]Q) \Rightarrow 0 \neq str_{bit}(M(map_{seq}([A_1 - A_2]Q))) \wedge R^S.A_t^S \textbf{ AND } str_{bit}\left(M_\succ^n(map_{seq}(Q))\right) = str_{bit}\left(M_\succ^n(map_{seq}(Q))\right) \wedge R^S.A_t^S$$

Moreover, for similar two LIKE query conditions described through another wildcard "[∧]" which is used to match one character that is not located in a given domain (e.g., "ABC[∧B-Z]" only can match "ABCA"), we define their condition mappings, similar to the two ones mentioned above.

Mapping 5. $R.A_tLIKE\ Q_1\#Q_2\#...\#Q_k$: It presents a more general *LIKE* condition, where "#" represents a wildcard (i.e., it can be equal to "%", "_" "[]" or "[∧]"). For such a general character string match pattern "$Q_1\#Q_2\#...\#Q_k$", if any character string is an instance of the match pattern, then it would contain all the substrings $Q_1, Q_2, \ldots$, and Q_k. Based on such an observation, we define the mapping for the general *LIKE* condition as follows:

$$trans_c(R.A_t\ LIKE\ Q_1\#Q_2\#\ldots\#Q_t) \Rightarrow trans_c(R.A_tLIKEQ_1\%) \textbf{ AND } trans_c(R.A_t\ LIKE\%Q_2\%) \textbf{ AND } \ldots \textbf{ AND } trans_c(R.A_tLIKE\%Q_k)$$

Moreover, we can present similar condition mappings for the *LIKE* query conditions in the forms of "$R.A_tLIKE\#Q_1\#Q_2\#\cdots\#Q_k$", "$R.A_tLIKE\ Q_1\#Q_2\#\cdots\#Q_k\#$" and "$R.A_tLIKE\#Q_1\#Q_2\#\cdots\#Q_k\#$".

Mapping 6. $R.A_t > Q$: Such a query condition presents a range query operation over character strings. Without loss of generality, we assume that $Q = A_1A_2\ldots A_r$ $(1 \leqslant r)$ and A is a character of the greatest value in the character set. Then, any character string $Q' = A_1'A_2'\cdots A_h'(1 \leqslant h)$ is greater than Q, if and only if it satisfying that: $(A_1 < A_1')$; or $(A_1 = A_1'$ and $A_2 < A_2')$; or $\ldots$; or $(A_1 = A_1', A_2 = A_2', \ldots, A_{k-1} = A_{k-1}'$ and $A_k < A_k')$, where k is equal to r (if $r \leqslant h$) or h (if $r > h$). Based on such an observation, the range condition mapping is defined as follows:

$$trans_c(R.A_t > Q) \Rightarrow trans_c(R.A_t LIKE[B_1 - A])\ \textbf{OR}$$
$$trans_c(R.A_t LIKE\ A_1[B_2 - A])\ \textbf{OR} \ldots\ \textbf{OR}$$
$$trans_c(R.A_t LIKE\ A_1A_2 \ldots A_{r-1}[B_r - A])$$

where $B_1 = A_1 + 1, B_2 = A_2 + 1, \ldots$, and $B_r = A_r + 1$. Moreover, we can define the similar condition mapping for another range condition "$R \cdot A_t < Q$". In addition, the range conditions "$R \cdot A_t \geqslant Q$" and "$R \cdot A_t \leqslant Q$" are equivalent with "$R \cdot A_t < Q$" and "$R \cdot A_t > Q$", respectively, and thus, their condition mappings are also identical to the two ones.

Above, we present the condition mappings for three main types of query conditions about character strings, i.e., equivalent conditions (Mappings 1–2); similarity conditions (Mappings 3–5); and range conditions (Mapping 6). All the server-side condition representations in Mappings 1–6 are described using the computation between two related matrixes, and can be executed in the DSP database immediately. Below, we analyze the performance of executing these server-side condition representations.

From the approach to constructing reachability matrixes mentioned in Section 4, we have known that, all the matrixes appearing in Mappings 1–6 are with the same size, i.e., equal to $(m^2 + m)$, where $m = |map_{set}(U)|$ and U is the set which all sensitive character strings defined over. Therefore, the computation between any two matrixes has to conduct m bit operations, i.e., its time complexity is $O(m^2 + m)$. If viewing each bit operation in the matrix computation as a basic unit, then we conclude that, (1) for each server-side condition representation generated from Mappings 1 to 3, it contains only one matrix computation, and thus its time complexity is $O(m^2 + m)$,i.e.,$O(m^2)$; (2) for each server-side condition from Mapping 4, its time complexity is $O(2m^2 + 2m)$, i.e., $O(m^2)$, due to having two matrix computations; (3) for each server-side condition from Mapping 5, its time complexity is $O(km^2 + km)$, i.e., $O(km^2)$, due to containing k matrix computations; and (4) for each server-side condition from Mapping 6, its time complexity is $O(rm^2 + rm)$, i.e., $O(rm^2)$. Last, it should be pointed out that, the bit computation between two n-phase reachability matrixes can be completed efficiently, and thus the server-side condition representations generated from Mappings 1 to 6 have better computation performance.

5. Querying encrypted character string

In this section, we describe how individual query operations (such as selections and joins) are implemented in our database architecture. The basic idea of our strategy is similar to that given in [2,11], i.e., to partition the computation of a query operation across the user client and the DSP server, so as to improve the querying performance. Specifically, we attempt to use a redefined query operation over the characteristic index attribute at the DSP database, to compute a superset of answers; and then the temporal answers are decrypted and filtered at the client, to generate the actual results. Using such a two-phase query approach, we attempt to minimize the query processing work done at the client as much as possible; consequently, making that, the client-side query operations can be implemented efficiently.

Below, we first discuss how individual selection or join operations are partitioned into two parts that would be executed at the client and the DSP server, respectively. Next, we based on such a partition, present a two-phase query algorithm, to efficiently execute SQL queries over encrypted character strings in the DAS environment. Last, we analyze the security of our proposed approach.

5.1. Partitioning query operations

Here, we only consider two types of essential query operations, i.e., selections and joins. We mainly consider how to partition a selection or join operation into its server-side representation and its client-side representation, so as to minimize the work done at the client as much as possible and to improve the query performance. Actually, the idea of this part is originated from the work given by Hakan et al. in [2]. We acknowledge their contribution. However, for the integrity of this paper, we here introduce this work.

(1) **Selection operation:** Let us consider a basic selection operation $\sigma_{COND}(R)$ defined on a relation R, where $COND$ is a basic condition specified on the sensitive attribute $R{\cdot}A_t$ of character string type in the relation R. A straightforward implementation of such an operation in the DAS environment is to transmit the entire encrypted relation R^S from the DSP server to the client. And, then the client decrypts R^S and executes the selection operation over the decrypted data. However, the running performance of such a straightforward strategy would be decreased, due to the two following factors. On the one hand, the encrypted relation needs to be transmitted from the server to the client through the Internet, resulting in a large data transmission quantity. On the other hand, all the encrypted character strings need to be decrypted at the client, resulting in a time-consuming data decryption operation. An alternative mechanism is to partially compute the selection operation at the DSP database through using the characteristic index related to the sensitive attribute $R{\cdot}A_t$, so as to decrease the number of encrypted tuples which need to be transmitted from the server to the client, and to improve the running performance effectively. Based on such a consideration, the selection operation can be rewritten as follows:

$$\boldsymbol{\sigma}_{COND}(R) = \boldsymbol{\sigma}_{COND}(\textbf{\textit{decrypt}}(\boldsymbol{\sigma}^S_{trans_c(COND)}(R^S)))$$

where ***decrypt*** is a decryption function used to decrypt the encrypted temporal results transmitted from the server. Besides, in the above notations, the selection operation, which being executed at the server, is adorned with a superscript "S" ($\boldsymbol{\sigma}^S$), namely, it is a server-side selection operation; and the non-adorned operation ($\boldsymbol{\sigma}$) would be executed at the client, namely, it is a client-side selection operation.

(2) **Join operation**: Let us consider a basic join operation $R_1 \bowtie_{COND} R_2$ over two relations R_1 and R_2, where $COND$ should be an equality condition defined on a sensitive attribute of character string type in R_1 and a sensitive attribute of character string type in R_2. Because a join can be viewed as a selection operation over the Cartesian product of two relations, such a basic join operation can be implemented by using an approach similar to that of a selection operation. Specifically, the join operation can be rewritten as follows:

$$R_1 \bowtie_{COND} R_2 = \boldsymbol{\sigma}_{COND}\left(\textbf{\textit{decrypt}}\left(R_1^S \bowtie^S_{trans_c(COND)} R_2^S\right)\right)$$

where as before, the "S" adornment on the join operation is to emphasize the fact that, the join operation would be executed at the DSP server. However, different from selection operations, such a join partition cannot guarantee that, the rewritten query operations have a better running performance than the original operation (i.e., through transmitting the encrypted relations to the client, and then executing the join operation at the client). In real system, we use a cost estimation formula to decide whether the computation of a join operation is partitioned across the server and the client. In addition, for other non-equality join operation (i.e., theta-join), we use the original strategy immediately to push the entire work of executing the join operation to be done at the client. Of course, other query operations related to non-sensitive fields would be executed at the DSP server in

advance, to decrease the number of the encrypted tuples required to be transmitted.

Below, we analyze the rationality of the above approach to partitioning query operations. From what mentioned in Section 4, we can conclude that, the server-side selection or join operations can guarantee to return a superset of actual results. Consequently, this guarantees that, the client-side query operations can select the actual results from the temporal ones transmitted from the server. Therefore, the partition approach can guarantee to return the actual query results.

5.2. Two-phase query algorithm

Algorithm 1. A two-phase query over encrypted character strings

Phase 0: Original query being translated at the client
Step 1. The **query translator** partitions the original SQL query posted by user into two parts: server-side query related to the characteristic index and client-side query identical to the original query;
Phase 1: Coarse query being executed at the DSP server
Step 2. The **server** executes the server-side query at the encrypted database, and then transmits the encrypted tuples returned by the server-side query to the client through the Internet, after discarding related characteristic index fields.
Phase 2: Refined query being executed at the client
Step 3. The **query executor** decrypts the encrypted tuples (i.e., decrypts the encrypted attributes corresponding to encrypted character strings), which are generated in the first coarse query phase and transmitted from the server;
Step 4. The **query executor** executes the client-side query over the decrypted tuples, as a result, obtaining actual query results and then returning to the user.

Based on the above approach to partitioning query operations, we present a two-phase query algorithm for querying encrypted character strings in the DAS environment efficiently, which is shown as Algorithm 1. In Algorithm 1, its input is an original SQL query, which is posted by user and defined over unencrypted relations, and its output is the true query results. As shown in Algorithm 1, the steps about querying sensitive character strings are mainly implemented by the query translator and the query executor. The particular workflow also can be seen in Fig. 1.

Example 5. Let us use the relation in Table 1 and its encrypted relation in Table 2 as an example. First, we assume that the original SQL query posted by user is as follows:

Q1. SELECT std.id, std.name, std.addr **FROM** std **WHERE** std.addr **LIKE** "%if%"
Then, in the first phase of Algorithm 1, this query is partitioned into two parts, noted as Q1 and Q2, respectively, where Q1 denotes the client-side query which is identical to the original query; and Q2 denotes the server-side query, given as follows:
Q2. SELECT std^S.id, std^S.name, std^S.addr^S **FROM** std^S **WHERE** $trans_C$(std.addr **LIKE** "%if%")
After execution of the sever-side query Q2, two encrypted tuples will be returned, which are the second record and the third record in Table 2. Next, in the second phase of Algorithm 2, the client decrypts the returned tuples and executes the client-side query Q1 (i.e., the original query) over the unencrypted tuples, as a result returning the second record in Table 1.

5.3. Security analysis

In the encrypted relation R^S at the DSP database, A_t^E is used to store the encrypted values of sensitive character strings and thus is in the form of ciphertext, so we think its values are safe as long as the encryption algorithm and the key are secure. However, it is out of scope of this paper to discuss the security of the encryption algorithm and the key. Here, we only analyze the security of the additional characteristic index field A_t^S.

From Section 3, we have known that, the characteristic index values are generated through mapping each sensitive character string Q into its n-phase reachability matrix $M^n(map_{seq}(Q))$. In this section, we mainly discuss the security problem of the characteristic index value $M^n(map_{seq}(Q))$. Specifically, we analyze whether there is probability that the attackers infer the plaintext character string based on the characteristic index value, through using some attack method. For comparison, we also consider other two schemes of (1) using $map_{seq}(Q)$ as the characteristic index value, and (2) using $M^n(Q)$ as the characteristic index value. Obviously, it is difficult for the attackers to infer the character strings from the characteristic index values directly, due to that the mappings such as g, f_1, f_2 and f_3 (referenced in Sections 3.1 and 3.2) are used. Therefore, we below analyze the security of the characteristic index values under two types of familiar attack methods, i.e., statistical attack and known-plaintext attack.

(1) **Statistical attack**. First, let us consider the simple scheme, which uses the 1-phase reachability matrix $M^1(Q)$, generated based on the method mentioned in Section 3.1, as the characteristic index value of the character string Q. In this scheme, different 1-phase connected character pairs always would be mapped into the different positions in a 1-phase reachability matrix, consequently, making it possible for the attackers to infer the mapping from character pairs to matrix positions by using a statistical method, and then, further to infer the plaintext character strings, i.e., this scheme may be helpless to prevent the statistical attack. In general, the statistical attack is built on the following two preconditions.

Precondition 1. The attackers have in advance known the detailed process of how to generate the characteristic index values for plaintext character strings, but they have not known the related metadata, due to which is stored at the trusted client site.
Precondition 2. Let G_O be a set of sensitive character strings, G_E another set of character strings, which are randomly chosen from G_O and need to be encrypted and stored into the DSP database, G_C a set of the characteristics index values, corresponding to the character strings in G_E. The attackers have in advance known G_O and G_C, but have not known G_E.

Based on the above two preconditions, the attackers can infer the position in a matrix, into which each character pair would be mapped, using the statistical method. Let U_C be the set over which all character strings are defined. The attack process is presented as follows. First, for each character pair $p_i = (a,b)$, which is defined over the set U_C, i.e., $a \in U_C$, and $b \in U_C$, the attackers compute the value $\mathbf{num}(p_i)[G_O]$, that is equal to the number of the character

strings in G_O, which contain the character pair p_i. Thus, the attackers can obtain a series of values: $\mathbf{num}(p_1)[G_O]$, $\mathbf{num}(p_2)[G_O]$, $\cdots$, $\mathbf{num}(p_h)[G_O]$, where h is equal to the number of all character pairs over U_C, i.e., $h = |U_C|^2$. Second, the attackers compute the frequencies for all the character pairs appearing in G_O as follows:

$$\boldsymbol{freq}(p_1,p_2,\ldots,p_h)[G_O] = \left(\frac{\mathbf{num}(p_1)[G_O]}{|U_C|},\frac{\mathbf{num}(p_2)[G_O]}{|U_C|},\ldots,\frac{\mathbf{num}(p_h)[G_O]}{|U_C|}\right)$$

Third, for each position c_i(except for the last row) in a 1-phase reachability matrix defined over the set U_C, the attackers count the value $\mathbf{num}(c_i)[G_C]$ that is equal to the number of the reachability matrixes in G_C, satisfying the value of the position c_i is equal to '1'. As a result, the attackers obtain a series of values: $\mathbf{num}(c_1)[G_C]$, $\mathbf{num}(c_2)[G_C]$, …, $\mathbf{num}(c_h)[G_C]$, where $h = |U_C|^2$. Fourth, the attackers compute the frequencies for all the matrix positions over G_C as follows:

$$\boldsymbol{freq}(c_1,c_2,\ldots,c_h)[G_C] = \left(\frac{\mathbf{num}(c_1)[G_C]}{|U_C|},\frac{\mathbf{num}(c_2)[G_C]}{|U_C|},\ldots,\frac{\mathbf{num}(c_h)[G_C]}{|U_C|}\right)$$

In a 1-phase reachability matrix, one position corresponds to one character pair defined over U, i.e., between which there is a one-to-one mapping. Now, the attackers try to infer the mapping. Assume that the character pair corresponding to the position c_i is $p'_i (i = 1, 2, \ldots, |U_C|^2)$. Obviously, have $freq(p'_1, p'_2, \ldots, p'_h)[G_E] = freq(c_1, c_2, \ldots, c_h)[G_C]$. Moreover, because G_E are randomly chosen from G_O, both of them should present the similar feature distribution, i.e.,$freq(p_1, p_2, \ldots, p_h)[G_E] \approx freq(p_1, p_2, \ldots, p_h)[G_O]$. Finally, through comparing the two sequences $freq(p'_1, p'_2, \ldots, p'_h)[G_E]$ and $freq(p_1, p_2, \ldots, p_h)[G_E]$, i.e.,$freq(c_1, c_2, \ldots, c_h)[G_C]$ and $freq(p_1, p_2, \ldots, p_h)[G_O]$, the attackers infer the character pair, which each position in the 1-phase reachability matrix corresponds to. Once the attackers know the mapping from matrix position to character pair, they can generate the 1-phase reachability matrixes G_M for all the character strings in G_O. Then, for any given characteristic index value, the attackers can find out all of its possible plaintext character strings based on G_M and G_O.

Now, let us consider scheme, which uses $map_{seq}(Q)$ generated based on what mentioned in Section 3.2 as the characteristic index value of the character string Q. In general, the statistical attack is built on the precondition that, there should be some one-to-one relationship between plaintext and ciphertext, e.g., the one-to-one mapping from matrix positions to character pairs, mentioned above. However, in this scheme, we use an Equi-Depth way to divide the set of characters, and then, map each character in a character string into an integer, resulting in the character string being mapped into an integer sequence (see Section 3.2 for detail). This makes many characters would be mapped into the same integer, i.e., forming a multiple-to-one mapping from character strings to integer sequences. In other words, it is difficult for the attackers to find out some one-to-one relationship between the character strings and the characteristics index values generated by this scheme, i.e., it is difficult to use a statistical method to infer the plaintext character strings from the characteristic index values. So, we conclude that, the scheme of using $map_{seq}(Q)$ as the characteristic index value can well oppose the statistical attack.

In our approach, we use $M^n(map_{seq}(Q))$ as the characteristic index value of the character string Q, which first maps the character string into an integer sequence, and then, maps the integer sequence into an n-phase reachability matrix. It can be viewed as a composite mapping, i.e., the combination of $map_{seq}(Q)$ and $M^n(Q)$, consequently, making it difficult to determine some one-to-one relationship between the character strings and the characteristics index values. Therefore, our approach also can well oppose the statistical attack.

(2) **Known-plaintext attack**. Known-plaintext attack means that the attackers have several samples of the plaintext and its ciphertext, and then make use of them to further infer other plaintexts. In general, except for the preconditions of the statistical attack, the following precondition should be added in order to conduct the known-plaintext attack.

Precondition 3. The attackers have in advance known a set of plaintext character strings S_C and their corresponding characteristic index values S_D.

First, let us consider the scheme, which uses the integer sequence $map_{seq}(Q)$ as the characteristic index value of the character string Q. We now show how the attackers infer the multiple-to-one mapping $g: U_C \to U_D$ (mentioned in Section 3.2), from the set U_C of characters to the set U_D of integers, and then, infer the plaintext character string from a characteristic index value in the form of integer sequence.

Let $Q_C = q_1q_2\cdots q_t (q_1, q_2, \ldots, q_t \in U_C, Q_C \in S_C)$ represent a plaintext character string, and $Q_D = d_1d_2\cdots d_t (d_1, d_2, \ldots, d_t \in U_D, Q_D \in S_D)$ be its corresponding characteristic index value. Due to having in advance known Q_C and Q_D, as well as the detailed process of how to generate the characteristic index value, the attackers can infer that: $d_1 = g(q_1)$, $d_2 = g(q_2)$, …, $d_t = g(q_t)$. Therefore, the attackers can further infer the mapping $g: U_C \to U_D$, as long as the character strings in S_C can cover all the characters in U_C. Once knowing the mapping from characters to integers, actually, the attackers know the mapping from character strings to their index values (i.e., integer sequences), so they can generate the integer sequences S_O for all the character strings in G_O. Then, for any given characteristic index value (in the form of integer sequence), the attackers can determine all of its possible plaintext character strings based on S_O and G_O.

Now, let us consider another scheme, which uses $M^n(Q)$ as the characteristic index value of the character string Q. In an n-phase reachability matrix generated by this scheme based on a character string, we only store the information about the n-phase connected character pairs contained in the character string, but do not store the order and frequency information about these character pairs. This makes that, given a character string and its n-phase reachability matrix, it become too difficult for the attackers to determine the n-phase connected character pair, to which each '1' in the matrix corresponds. Moreover, the use of three bijections f_1, f_2, and f_3 (referenced in Definition 1) also further enhance such difficulty. In other words, this scheme makes the attackers difficult to determine the mapping from character string to its n-phase reachability matrix, consequently, making it difficult to use a known-plaintext method to infer the plaintext character strings from the characteristic index values. In conclusion, the scheme that uses $M^n(Q)$ as the characteristic index value can well oppose the known-plaintext attack. In our proposed approach, we also leverage n-phase reachability matrixes as the characteristic index values of character strings. So, our approach also can well oppose the known-plaintext attack.

As mentioned above, the two schemes of using $map_{seq}(Q)$ as the characteristic index value and using $M^n(Q)$ as the index value, both cannot ensure the data security of their generated index values, where the first one cannot prevent the known-plaintext attack, and the other one cannot prevent the statistical attack. In our approach, we combine the two schemes. This makes the characteristic index values generated by our approach can well present both of the known-plaintext attack and the statistical attack, resulting in good security. However, just as pointed by Sergei et al. in [25], it is impossible to design an absolutely secure system that can well support queries over encrypted database. In other words, the in-

crease of data security certainly results in the decrease of the query effectiveness; and vice versa. In our approach, we make a good compromise between data security and query effectiveness.

Last, let us review the security of the characteristic index values generated using the approach mentioned in [11,12]. As pointed out by the authors, with the increase of the size of index values, the probability of different character pairs being mapped to different bits in the index values would be increased, such that the attackers can infer all the character pairs using the statistical attack or the plaintext attack. However, once all the character pairs have been known, the attackers can infer the original character strings easily, due to that the character pairs are well-ordered in their index values. Hence, our approach has a better security than that mentioned in [11,12].

6. Evaluation experiments

In order to evaluate the effectiveness and efficiency of our proposed approach, we have performed a number of experiments. In this section, we present our experimental results.

First, we constructed a database that only contains one table, whose schema is shown as Table 1. Table 3 presents the information about the attribute that is considered sensitive and needs to be encrypted. Second, for the database, we randomly generated a million of tuples, where *addr* values were constructed over a set consisting of 80 characters, i.e., they were determined by the regular expression given in the 5th column of Table 3. Last, the experiments were conducted over two Lenovo personal computers with Intel Core2 Duo 2.93 GHz CPU and 2 GB RAM. One of the computers performed as the DSP server, and the other performed as the client. In addition, we used Microsoft Windows NT as the operating system, and Microsoft SQL Server 2000 as the database management system.

6.1. Effectiveness evaluations

In the first group of experiments, we aimed to evaluate the effectiveness of our proposed approach, i.e., to evaluate the effectiveness for filtering non-targeted tuples in the first phase of the two-phase query algorithm. To do this, we used the following two measure factors: FIE and FAE.

Definition 4. Assuming that there are N tuples in a relation, the number of tuples returned in the first phase of the two-phase query algorithm is N_1, and the number of tuples in actual query results is N_2, then, we define the **filter rate** (FIE) and the **false rate** (FAE), as follows:

$$FIE = \frac{N - N_1}{N - N_2} \quad FAE = \frac{N_1 - N_2}{N_1}$$

where $N - N_1$ denotes the number of non-targeted tuples filtered in the first phase; $N_1 - N_2$ denotes the number of non-targeted tuples returned in the first phase; and $N - N_2$ denotes the total number of non-targeted tuples in the relation.

Table 4 presents all the query conditions used in our experiments, which, actually, presents the general cases for basic similarity queries and basic range queries. Other more complex similarity or range queries can be generated by using these basic queries. Aiming to similarity queries, we conducted experiments over character strings with the same length (equal to 20) but with characteristic index values of increasing length. The experimental results are shown in Figs. 2 and 3, where each data point was obtained by performing twenty experiments and computing average value of their results. In our experiments, we increased the length of index values, through partitioning the set that sensitive character strings are defined over, into more subsets. Hence, in Figs. 2 and 3, as well as the following Figs. 6, 7 and 10, each value m of the horizontal ordinate denotes the number of subsets partitioned, i.e., the length of characteristic index values is equal to $(m^2 + m)$.

From Figs. 2 and 3, we can see that, with the increase of the size of characteristic index values, on one hand, the FAE values would become smaller and smaller, i.e., the rates of non-targeted tuples returned in the coarse-query phase (i.e., Phase 1 in Algorithm 1) become smaller; and on the other hand, the FIE values would become greater and greater, i.e., the effects of the coarse query to filter non-targeted tuples are improved increasingly. This is because, with the increase of the length of characteristic index values, the probability of different character strings corresponding to different index values would become greater, i.e., the probability of non-targeted tuples being filtered would become greater. From Figs. 2 and 3, we also see that, by using the characteristic index values generated by our proposed approach, most of non-targeted tuples will be filtered at the coarse-query phase, and the filter rates to non-targeted tuples are generally greater than 0.8, even if the characteristic index values are assigned with a small length. Especially, for the similarity query condition F1 (i.e., addr **like** 'A_1%'), when the size of characteristic index is increased to 80^2+80, the FAE value would be equal to 0, and the FIE value equal to 1, i.e., all the non-targeted tuples have been filtered out in the coarse-query phase. This would reduce the number of encrypted tuples transmitted from the server to the client, and thus improve similarity query performance.

Aiming to similarity queries, we also have conducted another group of experiments over character strings with decreasing length but with characteristic index values of the same length (that is equal to $40^2 + 40$). The experimental results are shown in Figs. 4 and 5. From Fig. 4, we can see that, although the length of sensitive character strings is decreased, the FAE values are not changed obviously, which shows that the rates of non-targeted tuples returned by the coarse queries are relatively steady. This is due to that the decrease speed of the number of tuples returned by the coarse query is approximately accordant to that returned by the refined query, with the decrease of the length of sensitive character strings. From Fig. 5, we can see that, with the decrease of the length of character strings, the effects of the coarse query to filter non-targeted tuples can be improved to a certain extent, but such improvement is not obvious. In other words, the good effectiveness of the coarse query to filter non-targeted tuples will not be decreased seriously with the change of the length of sensitive character strings.

From Figs. 2 and 4, we see that, different similarity query conditions lead to the different change trends for FAE values, and, with the increase of quantity of information contained by the matching patterns (M1–M3 and F1–F3), the FAE values are also increased. This is due to that, the increase of quantity of information in the matching patterns would decrease the number (i.e., the N_1 value in Definition 4) of tuples returned by the coarse query phase in Algorithm 1, resulting in the increase of the FAE values. Besides, from Figs. 3 and 5, we also see the similar cases for the FIE values, which is also caused by the similar reason.

Regarding range queries, we conducted experiments over character strings with the same length (that is equal to 20) but with characteristic index values of increasing length. The experimental results are shown in Figs. 6 and 7. From Figs. 6 and 7, we see that,

Table 3
The information about sensitive field that needs to be encrypted.

Table name	Field name	Data type	Tuple number	Regular expression
Std	Addr	Varchar (20)	1,000,000	[,–|]{20}

Table 4
The similarity and range query conditions used in our experiments, where A_1, A_2, and A_3 denote three characters.

Name	LIKE condition	Name	LIKE condition	Name	Range condition
M1	Addr like '%A_1%'	F1	Addr like 'A_1%'	R1	Addr > 'A_1'
M2	Addr like '%A_1A_2%'	F2	Addr like 'A_1A_2%'	R2	Addr > 'A_1A_2'
M3	Addr like '%$A_1A_2A_3$%'	F3	Addr like '$A_1A_2A_3$%'	R3	Addr > '$A_1A_2A_3$'

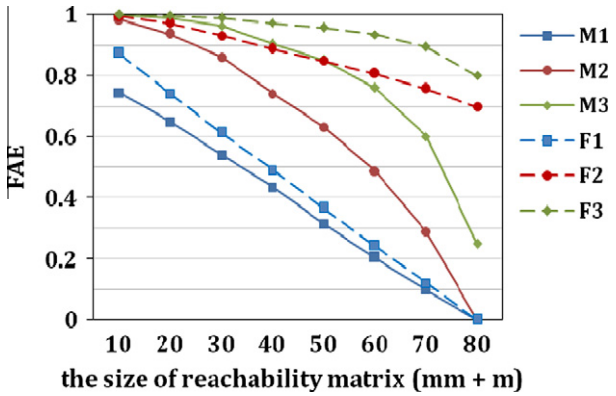

Fig. 2. FAE values for different LIKE query conditions over characteristic index values with increasing size.

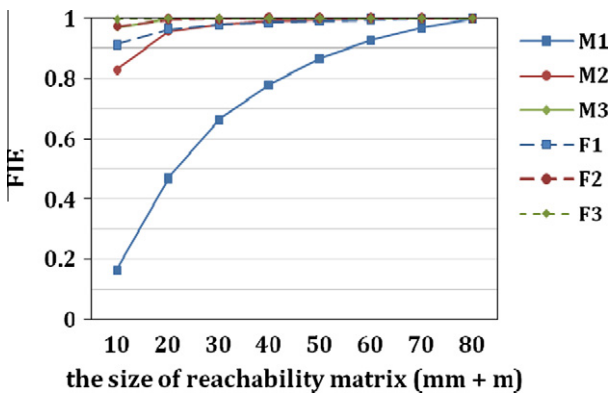

Fig. 3. FIE values for different LIKE query conditions over characteristic index values with increasing size.

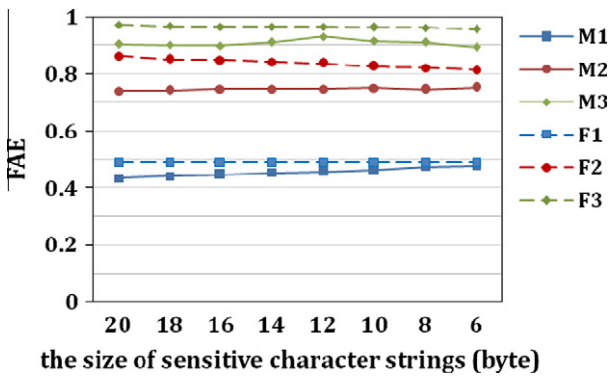

Fig. 4. FAE values for different LIKE query conditions over sensitive field of character string type with decreasing size.

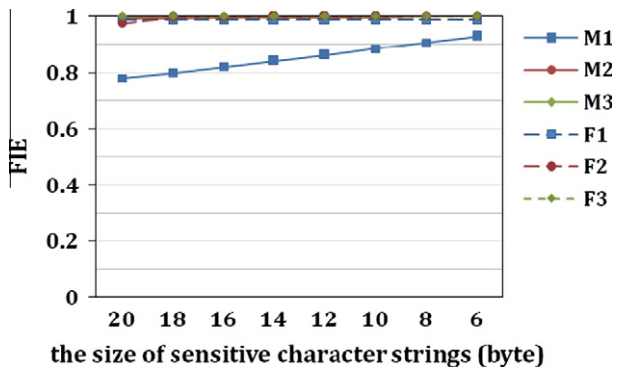

Fig. 5. FIE values for different LIKE query conditions over sensitive field of character string type with decreasing size.

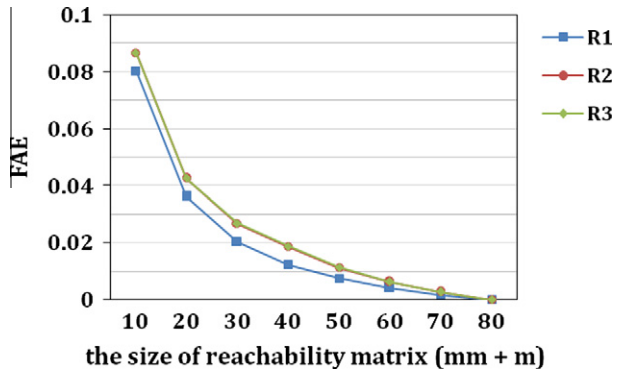

Fig. 6. FAE values for different range query conditions over characteristic index values with increasing size.

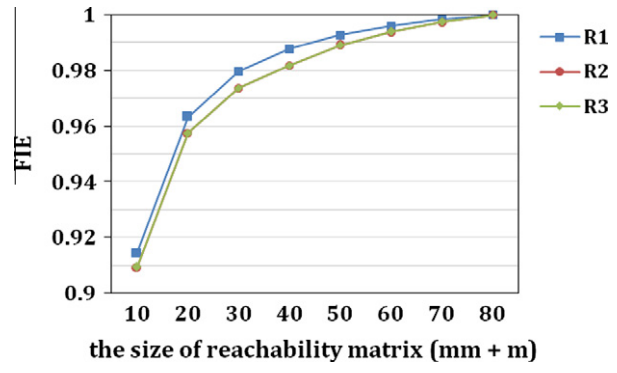

Fig. 7. FIE values for different range query conditions over characteristic index values with increasing size.

the range queries have smaller FAE values (less than 0.1) and greater FIE values (greater than 0.9), compared to the similarity queries; and, with the increase of the size of characteristic index values, both FAE and FIE values would be further improved to a certain extent. This is due to that, the actual result of a range query generally consists of too many tuples, thereby reducing the impact of the non-targeted tuples returned by the coarse query to the FAE and

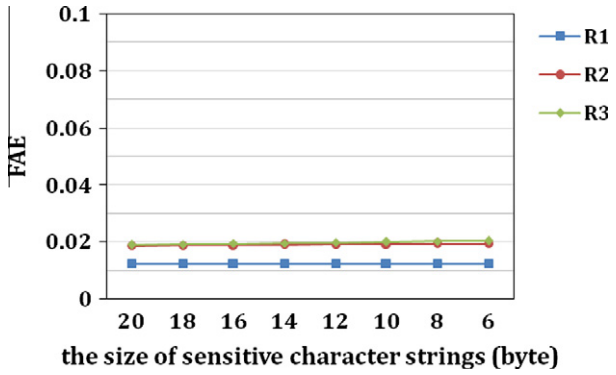

Fig. 8. FAE values for different range query conditions over sensitive field of character string type with decreasing size.

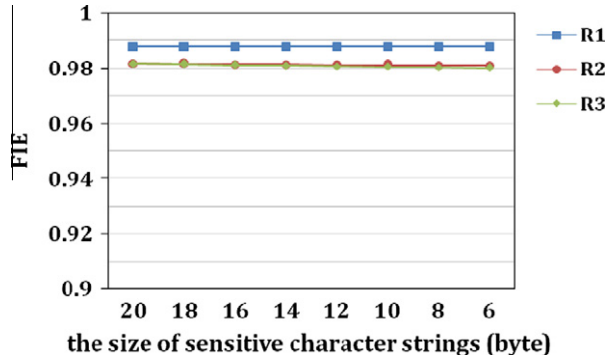

Fig. 9. FIE values for different range query conditions over sensitive field of character string type with decreasing size.

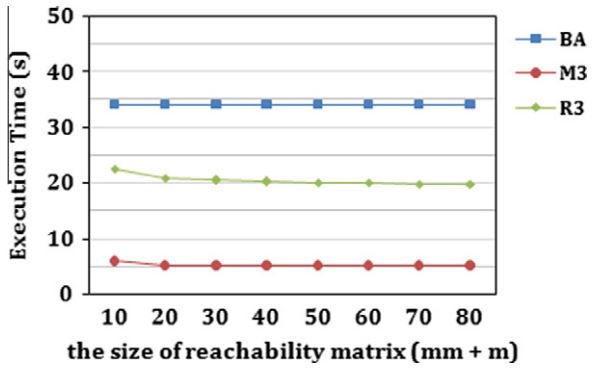

Fig. 10. Execution times for performing LIKE and range queries over characteristic index values with increasing size.

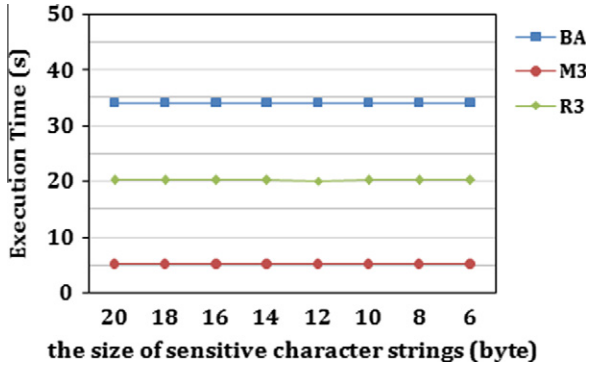

Fig. 11. Execution times for performing LIKE and range queries over character strings with decreasing size.

FIE values. From Fig. 7, we also see that, using the characteristic index values, most of non-targeted tuples would be filtered by the coarse query (the filter rates greater than 0.9, even if the index values are assigned with a small length). Therefore, our approach can well reduce the number of encrypted tuples transmitted from the server to the client, and improve the range query performance.

Aiming to the range queries, we also have conducted experiments over character strings with decreasing length but with characteristic index values of the same length (equal to $40^2 + 40$). The experimental results are shown in Figs. 8 and 9, which are similar to those of similarity queries except having steadier FAE and FIE values.

6.2. Efficiency evaluations

In the second group of experiments, we aimed to evaluate the efficiency of our approach, through testing the execution time of the two-phase query algorithm over encrypted tables, and comparing the results to the execution time in the traditional way that is to decrypt all encrypted data before querying them. In the experiment, the execution time of the two-phase algorithm over a query condition is computed by adding (1) the time of executing the coarse-query at the DSP database, (2) the time of transmitting the encrypted data returned by the coarse-query from the server to the client; (3) the time of decrypting the data; and (4) the time of executing the original query at the client.

The experimental results are shown in Figs. 10 and 11, where "BA" denotes the traditional way that transmits all encrypted tuples from the server to the client and executes queries at the client after decrypting the tuples. From Figs. 10 and 11, we see that, using the characteristic index values generated by our approach, the execution times for performing all kinds of queries over sensitive character strings in the DAS model can be improved effectively: using our approach, the execution time of a similarity query is decreased to about 0.2, and the execution time of a range query is decreased to about 0.6, compared with those in the traditional way.

6.3. Comparisons with other approaches

From Section 2, we know that although there have been many approaches to database encryption, most of them were not designed for querying encrypted character strings in the DAS model, consequently, making them difficult to be applied into the DAS model. Therefore, in this part, we compare our approach only with three existing ones, which were proposed in [11,13,14], respectively, and designed for the problem of how to query encrypted sensitive character strings. It should be pointed out that all the three approaches were not designed for the DAS model. For comparison, we have re-implemented the approaches over our prototype experimental system.

Table 5
The effectiveness comparison results among the approaches to encrypting and querying character strings, where "low" denotes non-support, "medium" denotes unable to support all the familiar query operations on character strings, and "high" denotes good support.

Approach	In [11]	In [13]	In [14]	**Of ours**
Similarity queries	Medium	Medium	Medium	**High**
Range queries	Low	Low	High	**High**
Security	Medium	Medium	Medium	**High**

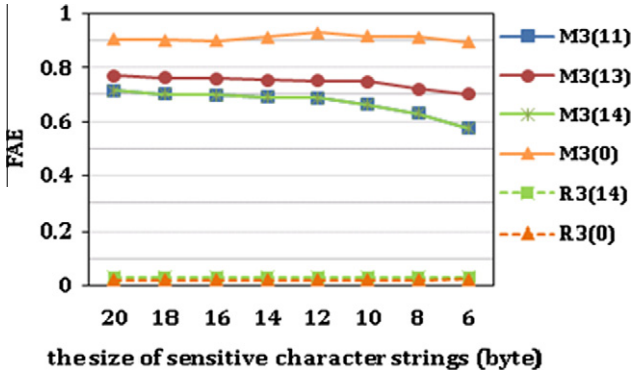

Fig. 12. FAE values for different LIKE and range query conditions over sensitive field of character string type with decreasing size.

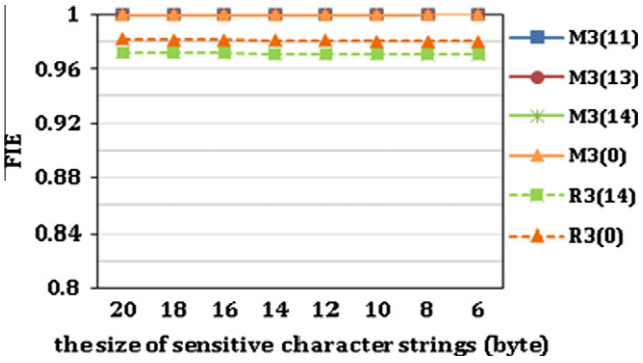

Fig. 13. FIE values for different LIKE and range query conditions over sensitive field of character string type with decreasing size.

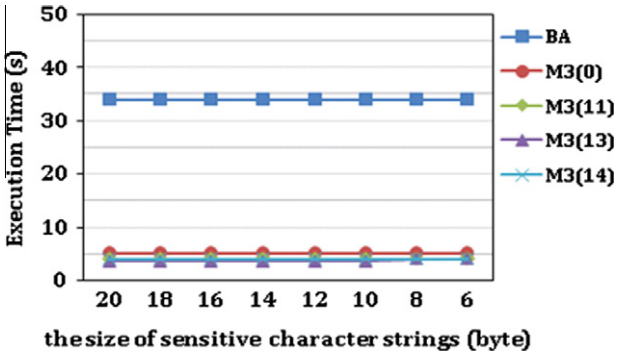

Fig. 14. Execution times for performing LIKE queries over characteristic index values with increasing size.

First, based on the evaluation results mentioned in [11,13,14], we make an overall comparison in terms of effectiveness and security. The comparison results are shown in Table 5. From Table 5, we can see that, compared to the other three approaches, our approach can not only better support various operations about querying encrypted character strings (including similarity queries and range queries), i.e., having better effectiveness, but also have better data security, enabling us to prevent attackers from attacking, thus, ensuring the data security and privacy better in the DAS environment. Overall, our approach has a good balance between security and effectiveness.

Second, experimentally, we make a more detailed comparison in terms of query effectiveness. In the experiments, (1) for our approach, the length of characteristic index values is set to $(40^2 + 40)$ (i.e., $m = 40$); (2) for the approach in [11], the number of bits of characteristic index field is set to 32 (that is recommended by the authors); (3) for the approach in [14], the number of bits of index is set to 32, and the number of partitions is set to 200. The experimental results are shown in Figs. 12 and 13, where "M3(O)" and "R3(O)" denote the implementation of our approach over the query conditions M3 and R3, respectively; "M3(11)" denotes the approach in [11]; "M3(13)" denotes the approach in [13]; and "M3(14)" and "R3(14)" denote the approach in [14]. From Figs. 12 and 13, we see that, for LIKE queries, our approach performs slightly worse than the other three ones in terms of both FAE values and FIE values. This is because in our approach, only the critical information contained in character strings would be retained into the characteristic index values, so as to improve the security. In addition, for range queries, our approach performs slightly better than the approach in [14].

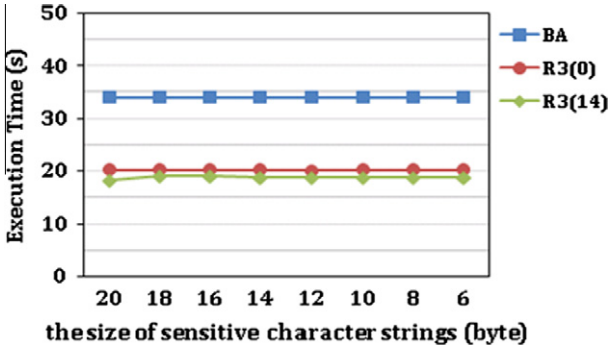

Fig. 15. Execution times for performing range queries over character strings with decreasing size.

Third, experimentally, we make a comparison in terms of query efficiency. The experimental results are shown in Figs. 14 and 15. From Figs. 14 and 15, we see that, although performing slightly worse than the other three approaches in terms of FAE and FIE values, our approach has nearly the same running efficiency to the other three ones. In Fig. 15, the approach in [14] has slightly better range query efficiency than our approach, which is because in our approach, a range query is generally completed based on several LIKE queries (see Mapping 6 in Section 4.2), resulting in the decrease of running efficiency of coarse query operations in the DSP site.

7. Conclusions

The "software as service" model for enterprise computing has emerged with the rise of the Internet technologies, in which, a service provider can provide software as a service to many clients over the Internet. However, unlike other services, databases are special, because data is a precious resource of an enterprise. This results in a higher request on data privacy and security for "database as service" model. To solve this, the "database as service" model stores sensitive data into the service provider after encrypting them, as a result, leading to a new issue, i.e., how to execute queries over encrypted data efficiently.

In this paper, aiming to the issue of how to query encrypted character strings efficiently in the "database as service" environment, we have addressed a new approach. In this approach, we use the n-phase reachability matrix to generate the characteristic index values for sensitive character strings, and we then store the characteristic index values in an additional field, when storing the encrypted character strings at the server site. Next, when querying the character strings, we first execute a coarse query over

the characteristic index at the server to filter out most non-targeted tuples, and we then decrypt the rest tuples and execute a refined query over them again at the client. Consequently, the query performance of executing queries over encrypted character strings is improved effectively. By using the *n*-phase reachability matrix as the characteristic index, our approach can not only support diversified operations about querying encrypted character strings, including equivalent query, similarity query, range query, etc., but also can well ensure data privacy and security, consequently, making our approach superior to other existing ones. Therefore, our proposed approach is applicable to query encrypted character strings in the "database as service" environment. Last, the evaluation experiments have validated the effectiveness and feasibility of our approach.

Acknowledgement

We thank anonymous reviewers for their valuable comments. This material is based upon work funded by the Scientific Research Fund of Zhejiang Provincial Education Department of China under Grant No. Y201016197; the Scientific Research Fund of Wenzhou Science and Technology Department of China under Grant Nos. 2009G0339 and S20100055; the Nature Science Foundation of Anhui Education Department of China under Grant Nos. KJ2012A273 and KJ 2012A74; and the Nature Science Research of Anhui under Grant No. 1208085MF95.

References

[1] H. Hakan, L. Bala, M. Sharad, Providing database as a service, in: Proceedings of ICDE' 02, San Jose, Canada, 26 February–1 March 2002, IEEE Computer Society, Los Alamitos, CA, 2002, pp. 29–38.
[2] H. Hakan, L. Bala, L. Chen, Executing SQL over encrypted data in the database service provider model, in: Proceedings of SIGMOD' 02, Madison, Wisconsin, 3–6 June 2002, ACM Press, New York, 2002, pp. 216–227.
[3] H. Hakan, L. Bala, M. Sharad, Efficient execution of aggregation queries over encrypted relational databases, in: Proceedings of DASFAA' 04, Jeju Island, South Korea, 17–19 March 2004, Springer-Verlag, Berlin, 2004, pp. 125–136.
[4] H. Hakan, L. Bala, L. Chen, Query optimization in encrypted database systems, in: Proceedings of DASFAA' 05, Beijing, China, 17–20 April 2005, Springer-Verlag, Berlin, 2005, pp. 43–55.
[5] B. Henry, Considerations in implementing a database management system encryption security solution, A Research Report in the Department of Computer Science at the University of Cape Town, 2003.
[6] J.M. He, M. Wang, Cryptography and relational database management system, in: Proceedings of IDEAS, Grenoble, France, 16–18 July 2001, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 273–284.
[7] R.L. Rivest, L. Adleman, M.L. Dertouzos, On Data Banks and Privacy Homomorphism, Foundation of Secure Computation, Academic Press, New York, 1978.
[8] J. Domingo-Ferrer, New privacy homomorphism and applications, Information Processing Letters 60 (5) (1996) 277–282.
[9] R. Agarwal, J. Kiernan, R. Srikant, Order preserving encryption for numeric data, in: Proceedings of SIGMOD' 04, Paris, France, 15–18 June 2004, ACM Press, New York, 2004, pp. 563–574.
[10] B. Shore, M. Sharad, T. Gene, A privacy-preserving index for range queries, in: Proceedings of VLDB' 07, Toronto, Canada, 23–27 September 2007, ACM Press, New York, 2007, pp. 720–731.
[11] Z.-F. Wang, J. Dai, W. Wang, Fast query over encrypted character data in database, Communications in Information and Systems 4 (4) (2004) 289–300.
[12] Z.-F. Wang, W. Wang, B.-L. Shi, Storage and query over encrypted character and numerical data in database, in: Proceedings of CIT' 05, Shanghai, China, 21–23 September 2005, IEEE Computer Society, Los Alamitos, CA, 2005, pp. 77–81.
[13] Y.-X. Li, G.-H. Liu, Encryption method for character data in the database, Computer Engineering 33 (6) (2007) 120–124.
[14] B.-G. Cui, D.-X. Liu, T. Wang, Practical techniques for fast searches on encrypted string data in databases, Computer Science 33 (6) (2006) 115–120.
[15] H. Houmani, M. Mejri, H. Fujita, Secrecy of cryptographic protocols under equational theory, Knowledge-Based Systems 22 (3) (2009) 160–173.
[16] J. Yang, X. Liu, T. Li, G. Liang, S. Liu, Distributed agents model for intrusion detection based on AIS, Knowledge-Based Systems 22 (2) (2009) 115–119.
[17] Q. Chen, S. Zhang, Z. Zhong, Dealing with inconsistent secure messages by weighting majority, Knowledge-Based Systems 24 (6) (2011) 731–739.
[18] B. Yuan, Secure communications with an asymptotic secrecy model, Knowledge-Based Systems 20 (5) (2007) 478–484.
[19] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, Combining fragmentation and encryption to protect privacy in data storagem ACM Transactions on Information and System Security (2010).
[20] W. Cong, W. Qian, R. Kui, Towards secure and effective utilization over encrypted cloud data, in: Proceedings of ICDCS-SPCC, 2011, pp. 282–286.
[21] S. Grau, T. Allen, N. Sherkat, Silog: speech input logon, Knowledge-Based Systems 22 (7) (2009) 535–539.
[22] M. Croitoru, L. Xiao, D. Dupplaw, P. Lewis, Expressive security policy rules using layered conceptual graphs, Knowledge-Based Systems 21 (3) (2008) 209–216.
[23] B. Elisa, C.S. Anna, P. Ivan, M. Lorenzo, Ws-AC: a fine grained access control system for web services, World Wide Web 9 (2) (2006) 143–171.
[24] V. Varadharajan, D. Foster, A security architecture for mobile agent based applications, World Wide Web 6 (1) (2003) 93–122.
[25] P.S. Gregory, C. Charles, Accurate estimation of the number of tuples satisfying a condition, ACM SIGMOD Record 14 (2) (1984) 256–276.
[26] AES, Advanced encryption standard, National Institute of Science and Technology FIPS 197, 2001.
[27] T.H. Cormen, E.L. Charles, L.R. Ronald, S. Clifford, Introduction to Algorithms, second ed., MIT Press, 2001.
[28] A. Silberschatz, Description of a new variable-length key, 64-Bit Block Cipher (Blowfish), Cambridge Security Workshop Proceedings, 1994.
[29] DES, Data encryption standard, FIPS PUB 46, Federal Information Processing Standards Publication, 1977.
[30] SQL, ISO/IEC 9075-1:2003, SQL/Framework. ISO/IEC, 2003.;
E. Sergei, F. Matthias, G. Oliver, , in: , pp. 117–117 (lif.).
[31] Z. Wu, G. Xu, Y. Zhang, Z. Cao, G. Li, Z. Hu, GMQL: a graphical multimedia query language, Knowledge-Based Systems 26 (2012) 135–143.