COMMUNICATIONS IN INFORMATION AND SYSTEMS
Vol. 4, No. 4, pp. 289-300, 2004

002

FAST QUERY OVER ENCRYPTED CHARACTER DATA IN DATABASE*

ZHENG-FEI WANG[‡†], JING DAI[‡], WEI WANG[‡], AND BAI-LE SHI[‡]

Abstract. There are a lot of very important data in database, which need to be protected from attacking. Cryptographic support is an important mechanism of securing them. People, however, must tradeoff performance to ensure the security because the operation of encryption and decryption greatly degrades query performance. To solve such a problem, an approach is proposed that can implement SQL query on the encrypted character data. When the character data are stored in the form of cipher, we not only store the encrypted character data, but also turn the character data into the characteristic values via a characteristic function, and store them in an additional field. When querying the encrypted character data, we apply the principle of two-phase query. Firstly, we implement a coarse query over the encrypted data in order to filter the records not related to the querying conditions. Secondly, we decrypt the rest records and implement a refined query over them again. Results of a set of experiments validate the functionality and usability of our approach.

Key words: database security, characteristic values, coarse query, refined query

1. Introduction. Traditionally, database security has been provided by physical security and operating system security. As far as we know, neither of these methods sufficiently provides a secure support on storing and processing the sensitive data. Cryptographic support is another important dimension of database security. It is complementary to access control and both of them should be used to guide the storage and access of confidential data in a database system. In [1, 2, 3, 4], database encryption mechanism could provide the following security.

(1) Encryption mechanism can prevent users from obtaining data in an unauthorized manner. For example, illegal users can not obtain the readable data without the proper key to decrypt it, even if they evade the access control of operating system or database management system.

(2) Encryption mechanism can verify the authentic origin of a data item. An attacker without knowing how to encrypt will be unable to create legal records which can be accepted by the database.

(3) Encryption mechanism also prevents from leaking information in a database when storage mediums, such as disks, CD-ROM, and tapes, are lost. Because the data are not in a readable format, the person obtaining the data will be of no advantage without the proper key to decrypt it.

*(Eds.) Wai Lam, Rui-song Ye, Haiying Wang, and Jun Zhang. This research was supported in part by the National Natural Science Foundation under Grant No.69933010

†Department of Computer and Electron Engineer, Hunan Business College, 410205, Changsha, China, E-mail: zhengfwang@sohu.com

‡Department of Computer and Information Technology, Fudan University, 200433, Shanghai, China, E-mail: {Daijing, weiwang1}@fudan.edu.cn

However, how to query efficiently the encrypted data becomes a challenge. This usually implies that the system has to sacrifice the performance to obtain the security. When data are stored in the form of cipher, we have to decrypt all the encrypted data before querying them. It is impractical because the cost of decryption over all the encrypted data is very expensive [5].

It is very interesting to develop a method that directly deals with the encrypted data without decrypting them [7, 8], in which data are encrypted by using the algorithm based the privacy homomorphism. The method can reduce the cost of the encryption operations and improve the performance. However, there are some disadvantages. Firstly, it does not possess the tough ability against attacks. As far as we know, there is still not a perfect method that ensures security of the encrypted data. Secondly, it is very difficulty to construct a privacy homomorphism function in practice. Song [9] presents a new encryption schema that will allow searching the encrypted data without decryption. But, the encryption algorithm used in their approach is not adapted for database. Hankan Hacijumus [10] proposes a way of executing SQL over the encrypted data in the database-service-provider model. However, the way is valid only for the numerical data, and is useless for the character data. Another weakness of the method is that it will output a large number of false joining records when querying over multi-tables, which leads to greatly the increase in the cost of decrypting records, so that the way enormously degrades the performance.

In this paper, we propose a framework that can conduct fast query over the encrypted character data in database. While storing the character data, we not only encrypt the character data themselves, but also turn the character data into the characteristic values via the characteristic function, and store them in an additional field (which we call index field) in the database. Therefore, the encrypted database is augmented with index field. When querying data, we apply the principle of two-phase query. In the first phase query (called Coarse Query); we filter the portion of the records not related to the query conditions by checking the index field. In the second phase query (called Refined Query); we decrypt the rest of records, and querying them again. Results of a set of experiments validate the functionality and usability of our approach.

Following the convention, E denotes the encryption function, D denotes the decryption function. The granularity of encryption is the field level, that is, the sensitive fields need to be encrypted to protect the sensitive information from exposing to unauthorized users.

The rest of the paper is organized as follows: Section 2 presents the architecture of storage and query over the encrypted character data. In section 3 we discuss how to store and query the encrypted character data in database. Section 4 analyzes security, storage space and the efficiency of filtering, and gives the relation among them. Section 5 gives our experimental results of querying over the encrypted table

from TPC-H benchmark. Section 6 concludes the paper.

2. The Architecture of Storage and Query. There are a number of approaches to implement the encrypted operation in database. e.g. application-based, DBMS-based, OS-based, and even by the way of collaboration among them. Our proposed system, whose basic architecture is shown in figure 1, is to add an encryption/decryption layer between the application and DBMS. The purpose of such design is to implement encrypted storage and efficiently query over character data without changing the internal architecture of the present DBMS and applications.

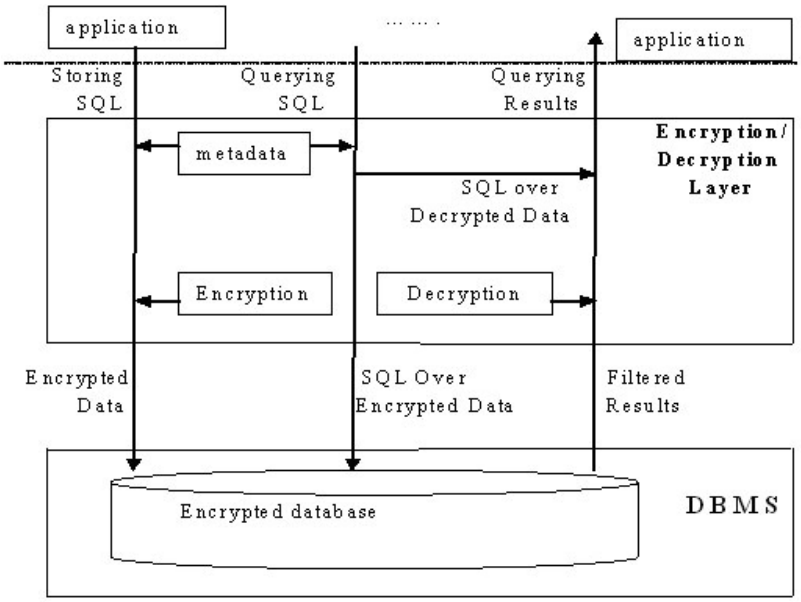


FIG. 1. *The architecture of encrypted storage and query over character data*

In the encryption/decryption layer of figure 1, metadata module contains some mapping functions and transformation rules. While storing data, metadata is used to transform querying SQL in order to store the characteristic value of the encrypted data together with the encrypted data themselves; while querying the encrypted data, metadata is used to transform querying SQL into appropriate SQL executed on the encrypted data. Encryption and decryption module contains encryption functions and decryption functions, which encrypt and decrypt the sensitive fields, respectively.

3. Storage and Query over Encrypted Character Data. In this section, we firstly design a characteristic function, which extracts the characteristic values from the character data. Then extend the storage schema in encrypted database in order to store the characteristic values. Lastly we give how to translate the querying conditions and present the query algorithm over the encrypted character data.

3.1. Characteristic Function: Pairs Coding Function.

DEFINITION 1. If there is a function PC: $s_1 \rightarrow s_2$, where s_1 denotes a string of characters $c_1c_2 \ldots c_n$, s_2 is a string of bits $b_0b_1 \ldots b_{m-1}$, b_i=0, $0 \leq i \leq m-1, n < m$. H denotes a hash function which encodes each connected character pairs $c_1c_2, c_2c_3, \ldots, c_{n-1}c_n$ of s_1 into a number between 0 and $m-1$, then the "signature" of the line $c_1c_2 \ldots c_n$ is the string of m bits $b_0b_1 \ldots b_{m-1}$, where $b_i = 1$ if and only if $H(c_jc_{j+1}) = i$ for some j. We call PC the Pairs Coding Function.

For examplestring s_1 is the word "abcehklst", a hash function maps $ab, bc, \ldots, st$ into a integer between 0 and 15, then $s_2 = PC(s_1) = PC(abcehklst) = (001010001010\ 1001)_2$, where there are six bits whose values are 1 in s_2. The reason is that some of eight pairs have the same hash value.

DEFINITION 2. For each relation schema $R(X_1, \ldots, X_r, \ldots X_n)$, where X_r field need to be encrypted, the corresponding encrypted relation schema is $R^E(X_1, \ldots, X_r^E, \ldots X_n, X_r^S)$, where, X_r^E is the encrypted field, $X_r^S = PC(X_r)$ and PC is the pairs coding function. We call X_r^S Pairs Coding Field of X_r, which is also called Index Field.

3.2. Encrypted Storage. For each relation schema $R(X_1, \ldots, X_r, \ldots X_n)$ in relational database, where X_r field is a sensitive field and need to be encrypted, we store an encrypted relation:

$$R^E(X_1, \ldots, X_r^E, \ldots X_n, X_r^S)$$

where, X_r^E in the encrypted relation R^E stores the encrypted value of X_r in relation $R, viz.\ X_r^E = E(X_r)$, index Field X_r^S stores the characteristic value of X_r in relation $R, viz.\ X_r^S = PC(X_r)$.

3.3. Query over Encrypted data. According to the extended storage schema of the encrypted relation, we use two-phase query over the encrypted character data. In the first phase, the original query is translated to the appropriate query over the binary bits in the corresponding index field before query. Once the translated query is executed, we can filter many records not related to the querying conditions. However, some records, which are not satisfied with the original querying conditions, can still in the returned record sets, because they can satisfy with the translated conditions. These records are false, and should be removed from the real result set. Hence, it is necessary to further process the record sets returned from the first phase again. In the second phase, after decrypting the returned records, we use a refined query over the decrypted data to obtain the accurate result set.

3.3.1. Translating Conditions of Query. The essential issue of query is how to translate the normal query conditions appeared in 'where' clause into the corresponding conditions over the index field of the encrypted table. This translation function is denoted as Tran(). We now analyze how to translate the query conditions

based on the different query categories. In general, we consider query conditions as the following three categories.

Simple Query. It gives a specific string value of a specific attribute. viz. attribute=value. The translation function is defined as follows:

DEFINITION 3. Tran$(a_i v) \Rightarrow a_i^s = PC(v)$

where field a_i has been encrypted, string v is the value of the query condition, a_i^s is the corresponding index field of a_i, PC is the pairs coding function.

For example, Tran$(did = davids) \Rightarrow did^s = PC(davids)(0010100010100001)_2$.

(2) Contain Query. It gives that a specific attribute which contains (or does not contain) a specific string value. viz. attribute like value, or attribute not like value. The translation function is defined as follows:

DEFINITION 4. Tran$(a_i$ like $c_1c_2 \ldots c_k) \Rightarrow ((a_i^s)_{H(c1c2)} = 1)$ AND $((a_i^s)_{H(c2c3)} = 1)$ AND $\ldots ((a_i^s)_{H(Ck-1ck)} = 1)$;

DEFINITION 5. Tran$(a_i$ not like $c_1c_2 \ldots c_k) \Rightarrow ((a_i^s)_{H(c1c2)} = 0)$ AND $((a_i^s)_{H(c2c3)} = 0)$ AND $\ldots ((a_i^s)_{H(c_{k-1}c_k)} = 0)$

where H is a hash function of the pairs coding function, $c_1c_2 \ldots c_k$ is the value of the query condition, $(a_i^s)_{H(c_{i-1}c_i))}$ denotes the $H(c_{i-1}c_i)$ bit.

For example, Tran(did like vid)$\Rightarrow$ $((did^s)_{H(vi)}$=1) AND $((did^s)_{H(id)}$=1).

(3) Boolean Query. It consists of the previous two types of queries combined with operation AND, OR, NOT, viz. (attribute=value 1) OR (attribute=value 2), (attribute like value 1) AND (attribute like value 2), (attribute like value 1) AND NOT (attribute like value 2). More complex query conditions result from the combination of these operations. The translation function is defined as follows:

DEFINITION 6. Tran$((a_i = v1)$ OR $(a_i = v2)) \Rightarrow$ Tran$(a_i = (PC(v1)))$ OR Tran $(a_i = PC(v2)))$.

DEFINITION 7. Tran$((a_i$ like $v1)$ AND $(a_i$ like $v2)) \Rightarrow$ Tran$(a_i$ like $(PC(v1))$ AND Tran$(a_i$ like $PC(v2))$.

DEFINITION 8. Tran$((a_i$ like $v1)$ or $(a_i$ like $v2)) \Rightarrow$ Tran$(a_i$ like $(PC(v1))$ OR Tran$(a_i$like $PC(v2))$.

3.3.2. Query Algorithm.

ALGORITHM 1: two phase query over the encrypted data

First Phase: Coarse Query Phase

(1) Translating the query conditions of SQL using the rules of metadata.

(2) Executing the translated SQL query, returning the records satisfying the translated query conditions and discarding the index field.

Second Phase: Refined Query Phase

(1) Decrypting the records returned in the first phase.

(2) Executing the original query SQL over the decrypted records and obtaining actual results.

In fact, the first phase query in algorithm 1 is used to filter some records not related with the query conditions in order to reduce the number of records needed to be decrypted in the second phase. Generally speaking, as we know, the cost of database decryption operation is far higher than that of query operations. Algorithm 1 improves the query performance through reducing the cost of decryption operation.

For example, consider a relation employee below table 1, in which the field 'did' is sensitive and need to be encrypted. After encrypted, the relation is shown in table 2.

TABLE 1
employee

eid	did	age	Sex
021021	Chessbasketball	24	M
021094	basketballcook	30	F
021095	Languageschat	26	M
021096	programnetwork	21	M

TABLE 2
employee^E

eid	did^E	age	sex	did^s
021021	1001010110010010010 11...	24	M	1011001011001011
021094	1001111001100001101 01...	30	F	1001100001101011
021095	0110101101000111001 01...	26	M	0110100011100100
021096	1111100011101011100 11...	21	M	0001110101110010

Assuming the original SQL is as follows:

select eid, age from employee where did like 'chess'.

In the first phase of algorithm 1, SQL is transferred into the follow:

select * from employee^E,

where ($\text{did}^s_{H(ch)}$ =1) and ($\text{did}^s_{H(he)}$ =1) and ($\text{did}^s_{H(es)}$ =1) and ($\text{did}^s_{H(ss)}$ =1).

After execution of the transferred SQL, two records will be returned, that is the first record and the third record.

In the second phase of algorithm 1, it firstly decrypts the returned records and executes a query over the unencrypted records again, the first record will be returned in the end.

4. Analyses. In this section, we analyze security, storage space and the efficiency of filtering, and give the relation among them.

4.1. Security Analysis. In the encrypted relation schema, the values of the sensitive fields are stored in the form of encryption, so we think that they are safe as

long as the cipher algorithm and the key are secure. Discussion about the security of cipher algorithm and the key is out of scope of this paper. We only analyze the security of the additional index field. The values of the sensitive field are mapped into the values of the index field via the pairs coding function. In general, it is very hard for attackers to directly infer the values of the sensitive field from the values of the index field due to use the hash function in the process of mapping. Take Birthday Attack as a example, although attackers can perform the collision attack, he does not infer the plaintext from the value of the index field. However, it is likely to suffer from the following two attacks in the environment of database.

(1) Statistical attacks. Assuming that the hash value is evenly distributed, when the number of bits m of the index field is increasing, the probability that different character pairs correspond to the same binary values in the corresponding index field is decreasing. That means, it is very possible for different character pairs to be mapped to the different values in the index field. In this case, attackers can infer the values of the sensitive field using statistical methods.

For example, there are 100 thousand of records in an encrypted relational table and m is 32. The attackers can obtain the accumulating occurrence of 1 for each bit in the index field, viz. $n_1, n_2, \ldots, n_{32}$, and compute the probabilities of each n_i, i.e. $n_1/100,000, n_2/100,000, \ldots, n_{32}/100,000$. Moreover, the attackers could also find out the probabilities of each pair of characters appeared in English. Comparing the two sets of probabilities, the attackers can infer the values of the sensitive field from the values of the index field.

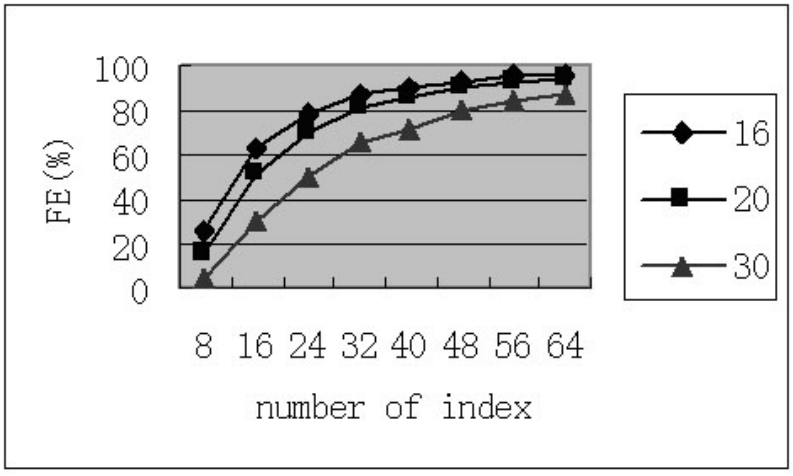
This kind of attack is based on the case that the number of bits m of the index field is great enough. If m is small, different character pairs will probably correspond to the same bit of the index field. For instance, assuming that there are 26 characters in English we are going to use, the number of different character pairs is 26^2, if m is 32 bits, there will be $26^2/32$=21.1 character pairs corresponding to the same bit of the index field in average. Thus it is very difficulty for the attackers to figure out the values of the sensitive index in this way.

(2) Plaintext Attacks. Similarly, when the number of bits of the index field is increasing, different character pairs probably correspond to the different values in the index field, so that the attackers can infer some sensitive values using plaintext attack. The reason is that the same (close) values of the sensitive field correspond to the same (close) values of the index field. For example, assuming that the attacker has known e_i as the value of the index field and the corresponding plaintext p_i, if the value of the index field is also e_i in other records, they can infer that the corresponding value of the sensitive field is p_i. When the number of bits of the index field is smaller, different character pairs will probably correspond to the same bits of the index field. Consequently, it increases the difficulty of such an attack.

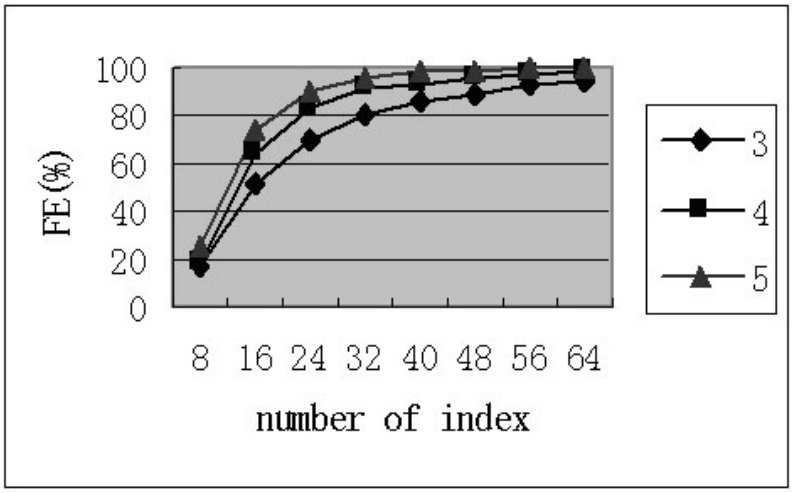
4.2. Filtering Efficiency Analysis. Result sets returned in the first phase of algorithm 1 may include some false records, which are not satisfied with the query conditions. The number of false records is closely related to the number of bits m of the index field. Assuming that the hash value is evenly distributed, when m is increasing, the probability, that different character pairs have the same binary value in the corresponding index field, will decrease. Therefore, the number of false records returned in the first phase becomes smaller and the filtering efficiency becomes better. When m is decreasing, it is contrary to the former.

4.3. Storage Space Analysis. In the encrypted table, additional index field leads to additional storage space. Additional space is related to the number of bits m of the index field. When m increases, storage space accordingly increases, otherwise, storage space decreases. Assuming that there are n records in a relation and m bits in the index field, our proposed schema only need to extend n*m bits.

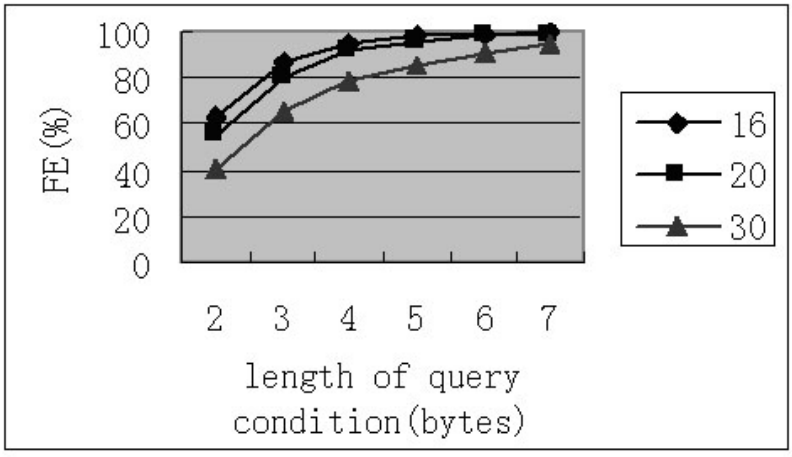
5. Experiments and Analyses of Performance. The purpose of the experiments is to show the validity and the efficiency of our proposed approach. According to TPC-H benchmark [11], the 10MB database is automatically created at scale factor 0.01 by utilizing the tool dbgen. TPC-H database include eight tables, of which the three tables used in our experiment are lineitem, customer and orders tables. The field comment in the three tables is considered as the sensitive field, which needs to be encrypted. To encrypt the field comment of the tables, safer++ encryption algorithm implemented in C is used. In safer++, the number of bits of each block and key are all 128. The experiments are conducted on a personal computer with Pentium IV 2.5GHz processor and 512 MB RAM. Relevant software components used are Windows NT as the operating system and SQL Server as the database server.



(a)



(b)



(c)

FIG. 2. *Effect of filtering efficiency.*

5.1. Experiment 1: Filtering Efficiency of Index Field. In the first set of experiments, we test the filtering efficiency of the first phase in algorithm 1. We conduct these tests with the increasing number of bits of the index field, the increasing length of the character string in query conditions and the increasing length of the character string need to be encrypted. Figure 2(a) (b) (c) shows the relation between filtering efficiency and these changing parameters. In order to precisely express the

filtering efficiency, we define it as follows:

DEFINITION 9. assuming that there are N records in the relation, the number of records returned in the first phase is n_1, and the number of records in actual results is n_2, then the filtering efficiency FE is defined as

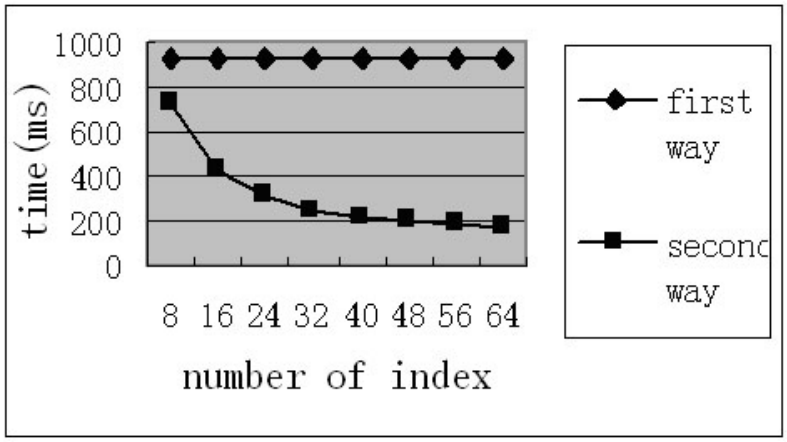
$$\mathrm{FE} = \frac{N - n1}{N - n2} \tag{1}$$

where $N - n_1$ and $N - n_2$ in formula (1) denote the number of filtered records in the first phase and the number of records not satisfied with the query conditions respectively.

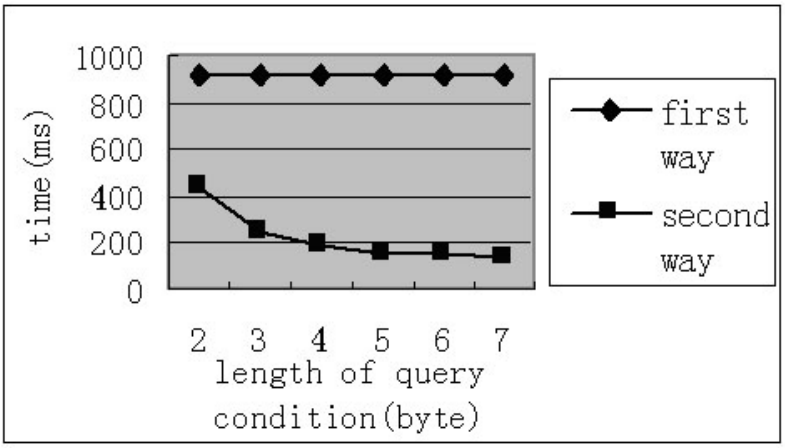
In figure 2(a), the length of the query value is 3 bytes, and different curves denote different length of the character strings in the sensitive field. In figure 2(b), the length of the character strings in the sensitive field is 16 bytes, and different curves denote different length of querying value. SQL is as follows:

select * from lineitem where comment like "query value".

We can find out from figure 2(a) and 2(b): (1) with the number of bits m of the index field increasing, filtering efficiency accordingly improves. The reason is that it is more possible for different character strings to correspond to different values in the index field when m increases. (2) With the length of the character strings in the sensitive field decreasing, filtering efficiency gets better. The reason is that it is more possible for different character strings to correspond to different values in the index field when length of the character strings in the sensitive field decreases. (3) Filtering efficiency increases slowly when the number m of bits of the index field is large than 32 bits. That is, it is not obvious to enhance the filtering efficiency by continuously increasing m when m increases to 32 bits. As we analyzed above, storage space will increase and security will degrade as m increases.



(a)



(b)

FIG. 3. *Effect of query-execution time.*

In figure 2(c), the number of bits of the index field is 32 bits, and different curves denote different length of the character strings in the sensitive field. SQL is as follows:

select * from lineitem where comment like 'query value'.

We can find out from figure 2(c): (1) with the length of query value increasing, filtering efficiency improves. (2) Filtering efficiency can reach about 80% when the number m of bits of the index field is 32 bits in acceptable circumstance. It indicates that we can filter out about 80% useless records in first phase of query.

5.2. Experiment 2: Performance of Querying Single Table. In the second set of experiments, we test the query-execution time on single table in algorithm 1, and compare the result to the query-execution time in the traditional way that is to decrypt all encrypted data before querying them. Where, SQL is as follows:

select * from lineitem where comment like 'query value'.

Figure 3(a) (b) show the cost of query-execution time in the two kinds of querying methods when the number of bits m of the index field and the length of the query value change respectively. Where, the axis of X denotes the length of encrypted character strings and query value respectively, the axis of Y denotes the time cost, and different curves denote different querying methods.

In the figure 3(a), the length of the query value is 3 bytes. It shows that the time cost of query-execution varies with m. We can find that the time cost of query-execution is decreasing with the increasing m. The main reason is due to the decreasing number of records needed to decrypt in the algorithm 1. The time cost of decryption operation is 12 times more than that of querying data. However, it seems that the performance will not significantly improve if m is larger than 32. The reason is that the filtering efficiency increases very slowly when m is larger than 32.

Figure 3(b) shows that the time cost of query-execution varies with the length of the query value. We can find that the time cost of query-execution is decreasing while increasing the length of the query value. Obviously the filtering efficiency will be better if the length of the query value is greater, accordingly, the cost of query time decreases. We can also find that the query time cost in the algorithm 1 decreases about 75% compared with that in the traditional way.

6. Conclusions. We present the architecture of storage over the encrypted character data, the corresponding query algorithm, and implement this proposed approach. Our solution has a number of advantages. It is so simple and practical that only need to add a module of encryption and decryption between applications and Database Management System. Therefore, we can easily integrate this solution into DBMS without much change. Secondly, this approach uses bit as data type of the index field, so it only need a little extra storage space. Thirdly, it is quite secure as long as the number of bits in the index field is not very large. Furthermore, it is so fast that it can decrease about 75% time cost compared with traditional query method.

REFERENCES

[1] Henry Brown, *Considerations in implementing a Database Management System Encryption Security solution*, A Research Report presented to The Department of Computer Science at the University of Cape Town, 2003.

[2] George l Davida, David L Wells, and John B Kam,*A Database Encryption System with Subkeys*, ACM Transactions on Database Systems, 6:2(1981), pp. 312–328.

[3] Hakan Hacigumus, Bala Lyer, and Sharad Mehrotra, *Providing Database as a Service*, in: Proc. of ICDE 2002, pp. 29–38.

[4] Jingmin He, Min Wang, *Cryptography and Relational Database Management System*, IDEAS, 2001, pp. 273–284

[5] Oracle, *Oracle9i Database Security for eBusiness*, An Oracle White Paper. June 2001.

[6] Thomas Fanghanel, *Using Encryption for Secure Data Storage in Mobile Database Systems[Ph D dissertation]*, September 2002.

[7] Ahitub N, Lapid C, and Neumann S, *Processing Encrypted Data*, Communications of the ACM. September 1987, pp. 777–780

[8] Rivest R L, Adleman L M, and Dertouzos M L, *On Data Banks and Privacy Homomorphisms*, in: Foundations of Secure Computation, 1978, pp. 169–178

[9] Dawn Xiaodong Song, David Wagner, and Adrian Perring, *Practical Techniques for Searches on Encrypted Data*, IEEE Symposium on Security and Privacy. 2000, pp. 44–55.

[10] Hakan Hacigumus, Bala Lyer, Chen Li, and Sharad Mehrotra, *Executing SQL over Encrypted Data in the Database-Server-Provider Model*, in: Proc of ACM SIGMOD, 2002, pp. 216–227.

[11] TPC-H, *Benchmark Specification*, http://www.tpc.org.