International Journal of Computer Applications (0975 – 8887)
Volume 41– No.4, March 2012

A New Method of Query over Encrypted Data in Database using Hash Map

Mohammed Alhanjouri
Asst. Prof. at Islamic university of Gaza
Gaza, Palestine

Ayman M. Al Derawi
Islamic university of Gaza
Gaza, Palestine

ABSTRACT

Critical business data in databases is an attractive target for attack. Therefore, ensuring the confidentiality, privacy and integrity of data is a major issue for the security of database systems. High secure data in databases is protected by encryption. When the data is encrypted, query performance decreases. In our paper we propose a new mechanism to query the encrypted data beside make a tradeoff between the performance and the security. Our mechanism will work over many data-types. We implement our work as a layer above the DBMS; this makes our method compatible with any DBMS. Our method based on replacing the select conditions on the encrypted data with another condition which is faster. The new way must have no security weak that is can't show an aspect for the plain data. The results of the experiments validate our approach.

Keywords

Encryption, Hash Map, Querying over Encrypted data, Index.

1. INTRODUCTION

Usually data is stored in databases to process and manage its relations; some data are classified as a high important data that needs to be high secured or on a level of security, the best way to secure such data is to encrypt it. Many encryption algorithms were studied and many designs of databases have prepared to put the considerations of encryption and security of the databases. In [1] the major challenges and design considerations pertaining to database encryption was described. The article first presents an attack model and the main relevant challenges of data security, encryption overhead, key management, and integration footprint. Next, the article reviews related academic work on alternative encryption configurations; indexing encrypted data; and key management. Finally, the article concludes with a benchmark using the following design criteria: encryption configuration, encryption granularity and keys storage. Dawn Xiaodong Song [2] proposes a new encryption method that allows searching the encrypted data without decryption. However, the method is not adapted for database encryption. Hankan Hacijumus [3] proposes a way that has a weakness; it will output false joining records, which leads to the greatly increased cost of decrypting records and degraded performance of query. They propose a schema of executing SQL over encrypted data in the database-service-provider model. Then in [4] the writers proposed a new query method, in which the query is completed on the server side and the client side together, they have proposed bucket index, which support the range query for the numeric data. Then they add a technique that supports arithmetic computation [5]. In [6] Hore optimized the bucket index method on how to partition the bucket to get the trade between the security and query performance. The methods based on index is supported by DBMS (Data Base Management System), and focused on the query performance at the cost of storage space. There are also some researches on the fuzzy query of character string. Zhengfei Wang proposed a function to support fuzzy query over the encrypted character data [7] [8]. Their method named pairing coding method, it encodes every adjacent two characters in sequence and converted original string directly to another characteristic string by a hash function. This method can't deal with some characters, and could perform badly for big character string. Paper [9] had proposed characteristics matrix to express string and the matrix will also be compressed into a binary string as index. Every character string need a matrix size of 259x256, it is large and will lead to much computation; in addition, the length of index has come to more than hundred bits, which is not suitable for storage in database. In [10] the paper works on a group of users that wants to access a secure data on a server. The shared sensitive information requires more security and privacy protection, In that paper, two schemes was proposed which can search the encrypted documents without re-encrypting all documents in a server even if group keys have to be updated. The schemes can support general database normalization for encrypted database. Their experiments show that their schemes are much more efficient than the comparables ones. Paper [11] only encrypts the sensitive field and it is also using bucket index to improve query performance. The order on numeric data is very useful. But on the character data, it has little effect. So the method in [11] is not fit for the character data. [12] Creates a B+ tree index for the data before encrypting them. When querying the encrypted data, firstly, it locates the encrypted records related to the querying predicate based on the B+ tree index; secondly, it decrypts the encrypted records to accomplish the results. Also, it must encrypt the B+ tree itself to protect it from leaking confidential information. According to the structure of the B+ tree, it encrypts each node of the B+ tree separately. The results of experiments in [12] show that the query performance over the encrypted data decreases about 20 percent compared with the plaintext query performance.

The traditional way to search an encrypted data is to decrypt all the data to plain text then find the target records. This way is obviously cost very time and have a bad performance especially with a large number of records.

We are proposing a new method to query encrypted data with many data types (string, character, numeric and date). Our method will have a good comparable response time with the traditional way. We also will use an index over the data, the indexing information should be related with the data well enough to provide an effective query execution mechanism; on the other side, the relationship between indexes and data should not open the door to linking that can comprise the protection. The attackers shouldn't guess the original input value from the output value if using the same function for encryption/decryption.

We have a compatible challenge, we don't know how the current DBMSs work and we can't add changes to its cores, that's needs an open source DBMS. In order to solve this problem we have to make sure that our new method can adapt easily with the DBMS. Our proposed way implemented on a standard database from a universal benchmark, some tests will done to prove the theoretical idea behind our work and this will follows by a comparison with the traditional way.

2. METHODOLOGY

2.1 Layer Technique

In order to implement our work we need an open source database, the drawback for this technique is that our work can adapt only with this type of the database and can't work with the commercial databases like Oracle, MS SQL, MS Access, MySQL, … etc which surely are closed source. To solve this problem, we developed another way to implement our work to adapt with any kind of DBMS. We add a layer above any kind of DBMS, this layer have the responsibility to manage the way to query over encrypted data.

The drawback for adding the new layer is the response time; the results prove that the performance of adding the layer will be much better when working on encrypted data with the traditional way.

The client will work over the layer which will contact with DBMS figure (1).

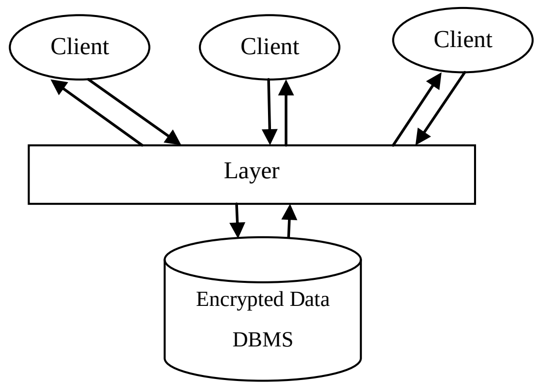

Fig 1: The Layer over the DBMS

The layer will provide the inner needed method for the methodology of process the encrypted data. The layer is better to be placed on the same place with the DBMS for two reasons:

1- Decreases the time of contacting with DBMS
2- Security purpose, the DBMS is usually placed on a safe place from the attackers.

2.2 Architecture of Layer Technique

The architecture of the layer is shown in figure (2). The queries from the client sent to the layer which has a subsystem called the Query Processor to check in the Meta data if there is any query on an encrypted column. The Meta data contains an instance of a data structure object called Hash Map. The Hash Map stores the mapping between the plain text and the encrypted text as KEY: VALUE, in which the KEY is the plain text and the VALUE is the encrypted value of the plain text. The Hash Map contains two main operations, PutValue and GetValue. PutValue(Key , Value), GetValue(Key): Value.

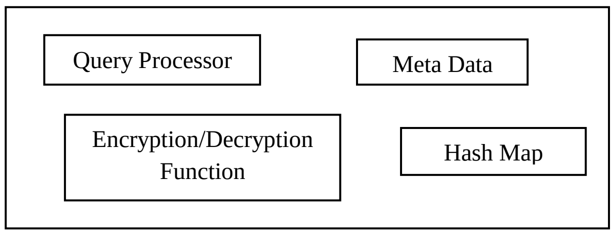

Fig 2: Architecture of the layer

The Query Processor replaces the client query with 'a plain where' clause on encrypted data value (the where clause is a plain text) with another one with an encryption on the plain searched data. For example if table CUSTOMER has an encrypted column C_PHONE and the client query is:

SELECT * FROM CUSTOMER
WHERE C_PHONE = '02 526 544';

By using the tradition way we need to decrypt all the values of C_PHONE then check which one equals '02 526 544', this means a huge response time especially with a large number of records.

By using our technique and using the Hash Map, the query processor will first search the Hash Map for the Key = '02 526 544' and get the Value which will be the ENC_VALUE ('02 526 544'), then replaces the where statement to be

SELECT C_NAME FROM CUSTOMER
WHERE H_C_PHONE = ENC_VALUE('02 526 544');

By using the index over C_PHONE it will be fast and easy to find the row that has the value of '02 526 544' on C_PHONE without needing to decrypt all the values which means a better response time.

2.3 Encryption

In our experiment we used AES-256 to encrypt the pre-selected column that's usually contains a high important data that is needed to be secured, the key of the AES will created according to standards and will kept on the server side.

An index is build over the encrypted column that makes the searching over the values in the encrypted column faster. By finding the needed encrypted value we find the needed plain text. That's done by using the same encryption/decryption algorithm with the same symmetric key which must be kept secret away from the attackers.

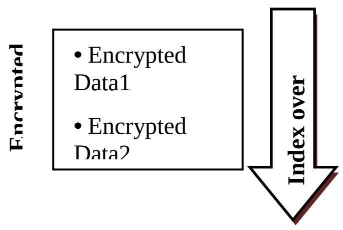

Fig 3: Index over encrypted data

2.4 HASH Map

In computer science, a hash map is a data structure that uses a hash function to map identifying values, known as keys (e.g., a person's name), to their associated values (e.g., their telephone number). Thus, a hash map implements an associative array. The hash function is used to transform the key into the index (the hash) of an array element (the slot or bucket) where the corresponding value is to be sought. In a well-dimensioned hash map, the average cost (number of instructions) for each lookup is independent of the number of elements stored in the table. At the heart of the hash map algorithm is a simple array of items; this is often simply called the hash table. Hash table algorithms calculate an index from the data item's key and use this index to place the data into the array. The implementation of this calculation is the hash function, f [13]:

index = f(key, arrayLength)

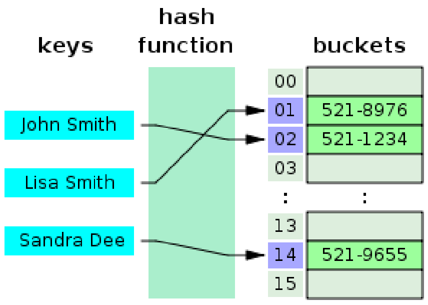

Fig 4: A small phone book as a hash table

The core of our method based on using the hash map. The data is stored on the Hash Map by using the function PutValue(Key, Value), the Key is a an identifier to the Value, we use the function GetValue(Key): Value to get the Value by passing the Key. In our methodology the Key will be the plain text and the Value will be the Encrypted plain text; i.e. encrypted Key.

Value = Enc(Key)

This way will cost more time especially with the insertion and updating on the encrypted column. Any insert or update statement must be followed by an inserting/updating value on the Hash Map. The time complexity for the Hash Map in big O notation is O(1) for the search and O(1) for the insert in average, O(n) for the search and O(1) for the insert in worst case[13].

3. 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, the data in the database is automatically created by using the tool dbgen. TPC-H database include eight tables, of which used in our experiment is customer table. To encrypt data of the tables, AES -256 encryption algorithm implemented in Delphi is used. The experiments are conducted on a personal computer with Intel Core2 Due 2.10 GHz and 2.87 GB RAM. Relevant software components used are Windows 7 as the operating system and Oracle 11g R2 as the database server. The layer is implemented by using the Delphi as a programming language. We test the different methods by measure the response time of the query over the table has a number of records ranging from 100 to 10000 records.

4. DATABASE ENTITIES, RELATION-SHIPS, AND CHARACTERISTICS

The components of the TPC-H database are defined to consist of eight separate and individual tables (the Base Tables). The relationships between columns of these tables are illustrated in Figure 5: The TPC-H Schema.

Table Layouts
The table layout can be finding on TPC-H v2.8.0

Data Generator
The DBGEN program used to generate the executable the data that populate the TPC-H Databases. This program produces flat files that can be used by the test sponsor to implement the benchmark.

Querying over Encrypted data
In the experiment, we test query execution time through comparing two different query approaches. The first way is the traditional way; decrypt all encrypted character data before querying them. The second way, which we propose in this paper, is to decrypt the result records after filtering the records not related to querying conditions.

Query Algorithm: query over encrypted character data
INPUT: a SQL which has a where statement on an encrypted data
OUTPUT: a collection of records satisfying with the query conditions.
METHOD:
(1) Replacing the query conditions of SQL using the rules of metadata.
(2) Executing the new SQL query, returning the records satisfying the translated query conditions by using the index.
(3) If the returning records contain an encrypted column, decrypt the records of the encrypted column and obtaining actual results.

We studied the two cases: the first case when the select query has no selects on an encrypted column(s) and has a where statement on an encrypted column. The second case when the select query has selects on an encrypted column(s) and have a where statement on an encrypted column.

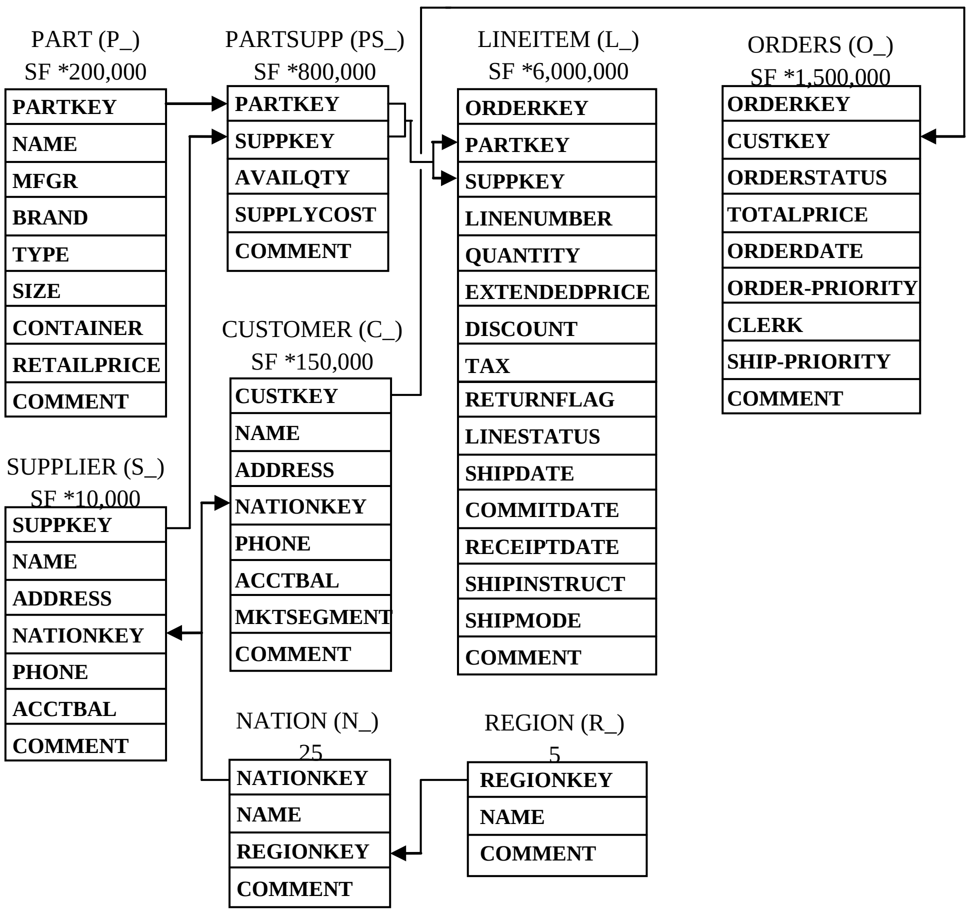

Fig 5: The TPC-H Schema

In each of the cases, we use the following methods:

1- The tradition method: query all the selected data with ignoring the where statement, decrypt the encrypted columns in the where statement, then filter the needed rows that have the values of the where statement.
We marked this method by: DEC_ALL

2- The enhanced method: replace the where statement on the encrypted columns with a where statement on the encrypt value of the searched plain text.
We marked this method by: ENH_HASH_METHOD

The results of each method are listed below in table 1.

Table (1). Query time cost vs. Number of record.

No Of Records	100	500	1000	10000
DEC_ALL*	864	4013	6800	47578
ENH_HASH_METHOD*	9	8	8	9
DEC_ALL	821	4189	6882	47565
ENH_HASH_METHOD	4	6	5	6

*Has selected encrypted columns
*The time is measured in ms

Figure (6) shows the cost of query-execution time of the two kinds of querying methods when the size of the data increased from 100 to 100000 records. We measured the time in mille second. The experiments are done for the two cases; with selected encrypted column and without. We mark the results of the experiments with using a select statement having selection on an encrypted column by *.

We found that DEC_ALL is relatively costly and there is a huge difference between the tradition DEC_ALL method and our method. This difference is obviously due to the number of records needs to decrypt in each of the methods. In the DEC_ALL, first, all the records in the table needs to be decrypt in the advance, then the decrypted records which are now a plain text have to be filtered as the condition in the where statement. The results of DEC_ALL are related to the number on records in the target table.

The results of ENH_HASH_METHOD show that there is a much improvement in the response time in compare with the DEC_ALL method. This improvement due to needing to use the decryption function one time only, the other operations needed (replacements of the where conditions and search the Hash Map) are done in the memory and need very little time in compare with the time needed when using the decryption functions.

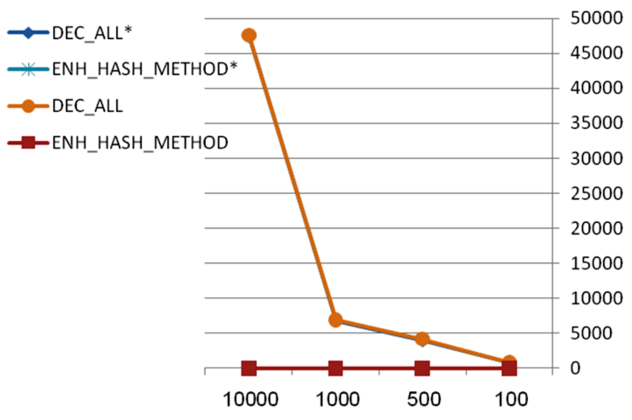

Fig 6: Results of executing the same query using different methods

The number of records in the table does not affect the response time; this is due to using the Hash Map to search the needed record in the database table which column is indexed so the values are ordered.

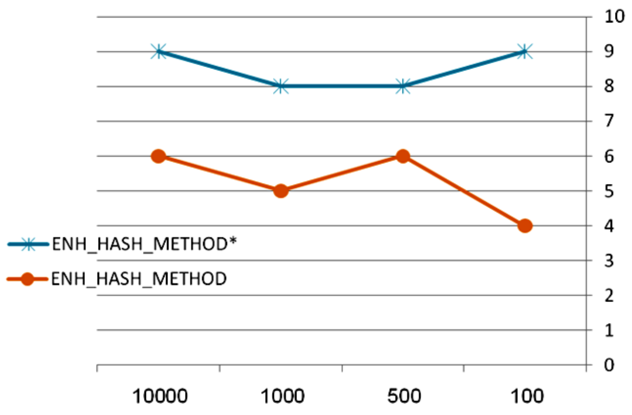

Fig 7: Results of executing the same query using our method

In figure (7), a comparison in made between the ENH_HASH_METHOD and ENH_HASH_METHOD*, we didn’t include the results of DEC_ALL because they are relatively much bigger so the graph will not give us a meaningful view. The results of figure (7) show that in the first case, in which the select statement has a select on an encrypted column that the ENH_HASH_METHOD* have a little more response time due to the time needed to decrypt the encrypted column. Using Hash Map will cost much when there is an insert or update on a value on the encrypted column, but this case (the insert and update statements) are not studied in this paper and we focus here on the select statement.

5. CONCLUSION

We proposed a new method of query over encrypted data in databases that can work with many data types. It doesn’t affect the inner structure of the DBMS because it implemented as a layer above the DBMS. We adapt our method using the hash map. The performance of our method is better than the traditional way to query over encrypted data; we prove this by do experiments that is measure the response time for every method when the number of records in the database changed. We implemented a small database according to TPC-H standard to do our experiments over it. The enhancing of the query performance over the encrypted data is a hot topic that is still under development

6. REFERENCES

[1] Erez Shmueli, Ronen Vaisenberg, Yuval Elovici and Chanan Glezer, “Database Encryption – An Overview of Contemporary Challenges and Design Considerations” SIGMOD Record, September 2009 (Vol. 38, No. 3)

[2] 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.

[3] H. Hacigumus , Bala Iyer and Sharad Mehrotra, "Providing Database as a Service", Data Engineering, 2002. Proceedings. 18th International Conference

[4] H. Hacigumus, B. Iyer, C. Li and S. Mehrotra, “Executing SQL over encrypted data in the database service provider model,” In ACM SIGMOD Conference, 2002, pp. 216-227.

[5] H. Hacigumus, B. Iyer, and S. Mehrotra. “Efficient execution of aggregation queries over encrypted relational databases”. In the proceedings of Database Systems for Advanced Applications (DASFAA), 2004, pp. 125-136

[6] B. Hore, S. Mehrotra and G. Tsudik. “A Privacy-Preserving Index for Range Queries”. In Proceedings of the 30th VLDB Conference, 2004, pp. 720–731.

[7] Z. Wang, J. Dai, W. Wang and B.L. Shi, “Fast Query over Encrypted Character Data in Database”. Communications In Information and Systems, 2004, pp.289-300

[8] Zheng-Fei Wang, Wei Wang and Bai-Le Shi , "Storage and Query over Encrypted Character and Numerical Data in Database", Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference

[9] H. Zhu, J. Cheng and R. Jin, “Execution Query over Encrypted Character Strings in Databases,” Frontier of Computer Science and Technology, 2007, pp. 90-97

[10] H. APark, D. Lee, J. Zhan and G. Blosser, "Efficient Keyword Index Search over Encrypted Documents of Groups" ISI 2008, June 17-20

[11] Yu Han, Zhao Liang Niu Xiamu, “Research on a new method for database encryption and cipher index”. Acta Electronica Sinica, No. 12A 2005

[12] Z. Wang, A. Tang and W. Wang, "Fast Query over Encrypted Data Based on b+ Tree”, International Conference on Apperceiving Computing and Intelligence Analysis (ICACIA), 23-25 Oct. 2009.

[13] Wikipedia, the free encyclopedia that anyone can edit “http://en.wikipedia.org/wiki/Hash_table” , an article on Hash table

[14] Bertino, E.; Sandhu, R., "Database security – concepts, approaches and challenges", IEEETransactions on

Dependable and Secure Computing, VOL. 2, NO. 1, JANUARY-MARCH 2005

[15] S. Sesay, Z. Yang, J. Chen and D. Xu, “A secure Database Encryption Scheme”. Consumer Communications and Networking Conference (CCNC), 2005, pp. 49-53

[16] W. Baohua, M. Xiniang and L. Danning, "A Formal Mutilevel Database Security Model", IEEE International Conference on Computational Intelligence and Security, 13-17 Dec. 2008.

[17] Y. Zhang, W. Li and X. Niu, “A Method of Bucket Index over Encrypted Character Data in Database”. Intelligent Information Hiding and Multimedia Signal Processing, 2007, pp. 186-189

[18] Michael Mitzenmacher , "Compressed Bloom Filters", IEEE/ACM Transactions on Networking, VOL. 10, NO. 5, October 2002

[19] Jehoshua Bruck , Jie Gao and Anxiao (Andrew) Jiang, "Weighted Bloom Filter" ISIT 2006, Seattle, USA, July 9 14, 2006

[20] Yasuhiro Ohtaki, "Partial Disclosure of Searchable Encrypted Data with Support for Boolean Queries, Availability, Reliability and Security”, 2008. ARES 08. Third International Conference

[21] Yong Zhang, Wei-xin Li and Xia-Mu Niu, "A Secure Cipher Index Over Encrypted Character Data in Database", Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming, 12-15 July 2008

[22] Lianzhong Liu and Jingfen Gai, "Bloom Filter Based Index for Query over Encrypted Character Strings in Database", 2009 World Congress on Computer Science and Information Engineering

[23] Yong Soon KIM and Eui Kyeong Hong, "Considerations of Extending SQL on Encrypted Data in UniSQL", Advanced Communication Technology, The 9th International Conference on 12-14 Feb. 2007

[24] Tingjian Ge and Stan Zdonik, "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS", Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference

[25] Premchand B. Ambhore,B.B.Meshram and V.B.Waghmare "A Implementation of Object Oriented Database Security", Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference

[26] Yu Chen and Wesley W. Chu, Fellow "Protection of Database Security via Collaborative Inference Detection", IEEE Transactions on Knowledge and Data Engineering, VOL. 20, NO. 8, August 2008

[27] Zhu Yangqing, Yu Hui and Li Hua, "Design of A New Web Database Security Model", 2009 Second International Symposium on Electronic Commerce and Security

[28] Sohail IMRAN and Irfan Hyder, "Security Issues in Databases", 2009 Second International Conference on Future Information Technology and Management Engineering

[29] Xu Ruzhi, Guo jian and Deng Liwu, "A Database Security Gateway to the Detection of SQL Attacks", 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)

[30]http://www.cecs.csulb.edu/~monge/classes/share/B+TreeIndexes.html.