Efficient Keyword Query in Encrypted Cloud Databases

He-qun XIAN, Shu-guang ZHANG, Man ZHANG,
Jun-zheng XIANG and Min LI
College of Computer Science and Technology, Qingdao University, China

Keywords: Bloom filter, Encrypted database query, SQL function, Cloud database.

Abstract. Encryption is an efficient way of protecting database security. Traditional keyword based queries cannot be executed directly over encrypted databases. An encrypted keyword search method is presented based on the bloom filter mechanism. Keywords are mapped into a bloom filter array, which are attached to database tables and stored as string fields in the tables. When processing queries, the bit-array and the encrypted index are compared, then the query can be carried out against encrypted fields. The principle of bloom filter is introduced together with its error analysis. We also implement multiple hash functions as the building block of our design. Finally, we make an extension of the current SQL in our system by creating user-defined functions and CRL assemblies. The prototype of our design is implemented on a SQL Server database. Experiment results show that our method is efficient and highly applicable.

Introduction

The Cloud hosted database technology has been extensively developed and widely accepted in recent years. Massive volume of application data have been stored in cloud databases, which may be gathered from websites, mobile APPs, web TV, vehicle carried devices et al. These datasets inevitably contain some sensitive data or data that may cause privacy issues. As the data owner and the cloud database service provider are no longer in the same trusted domain, the security of these sensitive data is highly at stake. A number of database leakage accidents happened in the past few years, some of which strike the nerve of the public and the government, causing huge economic damages at the same time. In December 2014, the famous Chinese railroad ticket distribution website 12306.cn was attacked, millions of entries of data concerning registered user accounts information were disclosed. In the year that followed, a former employee stole 20 Gigabyte of user information from the largest e-commerce corporation in China, and then sold them out on the Internet. These accidents all have the same primary cause that sensitive data are stored in databases with little protection or no protection at all. As soon as the adversary breaches the defense via SQL injection or other techniques, they may acquire direct access to the sensitive data, and the information security is severely compromised.

Over the years, database encryption has been proven an effective way of protecting sensitive data against various attacks. Encryption is widely adopted in security critical applications. However, in the cloud database scenario, encryption baffles traditional database query processing. Keyword based queries cannot be executed directly over encrypted database tables. There is a rich literature of study on how to carry out those queries because people need both security and efficiency. The problem of carrying out queries over encrypted databases has received much attention from the academic and industrial society.

In this paper, we focus on keyword-based queries over encrypted databases. We harness the bloom filter mechanism and devise a novel scheme that can handle almost all of the data types in relational databases. We enumerate all the keywords and process them as string type variables. Keywords are mapped into bloom array elements via multiple hash functions. When carrying out user queries, keywords related bit values are compared with the pre-computed array stored with each tuple. Tuples containing the matching bit index values are selected and decrypted, and then they are returned as the query result. Our method improves the efficiency of query execution over...
encrypted databases, and is independent of encryption scheme. It is compatible with any encryption algorithm and is of great application value.

Related Works

One of the key issues in the area of database security is how to perform query and update operation in an encrypted database. The first solution addressing this problem is privacy homomorphism mechanism proposed by Rives et al., which defined multiple encryption functions[1]. Encrypted data can be operated on without decryption. In 2009 Gentry proposed the first fully homomorphic encryption scheme based on ideal lattices[2]. Theoretically, any encrypted data can be arbitrarily calculated. However, the computational complexity of Gentry’s scheme is too heavy, it cannot be used in real word applications. Agrawal et al. proposed an order preserving encryption scheme in SIGMOD, they devised a special encryption algorithm for numerical data[3]. Ciphers can be arithmetically compared to one another with respect to their original values, encryption and decryption operations can be performed with satisfying speed. Effective as it is, the security of the special encryption algorithm is relatively weaker than standard encryption algorithms. An encrypted index method is introduced in [4] which uses the encrypted data to build the index on the database server. Buckets are defined with respect to the original values, each data tuple is assigned with several bucket index values which are stored on the server. User queries are converted into queries that can be directly executed on these index values. Following their previous work, Hakan et al. discussed the key updating problem in the encrypted database. A key update transaction was defined which updates keys and re-encrypts data[5]. Shi et al. developed a new method for storing and querying textual data and numerical data. Most queries for character data involve precise match conditions or fuzzy match conditions, while queries for numerical data often include quantity and range conditions. So, it is much easy to handle encrypted query processing corresponsive for the two different type of data[6]. A method for aggregation query processing over encrypted databases was proposed in [7] which was based on the privacy homomorphism principle. Zhu et al. devised a novel method for querying encrypted character data by converting sensitive data into matrixes[8]. Their method can handle both English and Chinese keywords, it can also reduce the false rate in fuzzy and range queries. Alhanjouri et al. made use of hash map, and constituted another method of querying encrypted databases. Selection conditions were substituted so that queries can be carried out more efficiently [9].

Most related works focus on database encryption mechanisms, some of which do not meet the security requirement for certain sensitive data applications. Our scheme mainly solve the problem of query processing over encrypted data, it is algorithm independent which makes the scheme compatible with almost all data encryption standards. Hence, the proposed scheme in this paper is more applicable, it can be deployed for real world database applications.

Problem Definition

In a relational database table \( R \), any attribute \( A \) could be considered sensitive, which needs encryption protection. We use \( R^E \) to represent the partly encrypted table stored on the server. To carry out a keyword search query against encrypted attributes, we need encrypted secure indices on the database server. When a query is issued by a user or an application, the conditions are defined against \( R \). The query cannot be executed directly on \( R^E \), it should be rewritten so that original conditions on plaintext attributes are mapped into conditions on the secure indices. The rewritten queries can be executed on the database server directly. Those matching tuples are returned to the client side with their sensitive attributes encrypted throughout the whole process. We illustrate the procedure with Figure 1. The relation on secure index items and original keyword should be established in advance.

We define client query \( Q \) as a triple \( < T,R,P > \), \( T \) is the target attribute set of the query, which could contain simple attribute names, aggregation functions on attributes or valid expressions. \( R \) represents the involving relational table or table set. We simplify the problem by treating views and
table-value SQL functions as ordinary tables. In most application systems, queries against views and table-value SQL functions are converted into queries against ordinary tables by the database server. We use $P$ to represent the predicate expression in the query, which constitute the query condition. $P$ usually takes the form of conjunctive normal form of predicates, which may very likely contain some keyword search predicates on encrypted attributes. During the query rewriting process, those predicate keyword search predicates on encrypted attributes are substituted by their counterpart predicates on secure indices. Then a new query $Q' < T, R, P'>$ is generated from the original query $Q < T, R, P>$ with reference to the predefined keyword mapping rules.

![Figure 1. Query rewriting procedure in encrypted databases.](image)

In most commercial database management systems, only precise match and fuzzy match predicates are supported for traditional keyword search SQL query. So, in our design, we not only need to guarantee the correctness and efficiency of our method, but also need to minimize the cost for implementing our scheme on common and popular database management systems. Comparison operators in common SQL grammar cannot be directly used to construct predicates on secure indices. So, it is necessary to extend current SQL grammar, and maintain high performance of SQL execution. In our work, we define new SQL functions which are used as the comparison operation for secure index match. Another challenge is the data type compatibility. In most real world applications, sensitive data may be formatted in various data types. When encryption is applied to multiple types of data, it is necessary to take into consideration the data type compatibility issue.

### Secure Index

We use a hash function that takes input with arbitrary length, and produces fixed length output [10] defined as in (1),

$$f : \{0,1\}^{+} \rightarrow \{0,1\}^{n} (t \geq 1)$$  \hspace{1cm} (1)

The hash function is an essential ingredient of the Bloom filter, which is an array of $m$ bits, initialized to zero. It requires a set of $k$ independent hash functions that produce uniformly distributed output in the range $[0, m-1]$ over all possible inputs [11]. It is a very efficient way to store information about the existence of a record in a database. It is susceptible to false positives, but the probability of a false positive can be made as small as desired.

To use the Bloom filter, initialize the $m$ bits in array $A$ to zero. For a set with $n$ elements $S = \{s_1, s_2, ..., s_n\}$, each element $s_j (1 \leq j \leq n)$ is mapped into $k$ values $\{g_1, g_2, ..., g_k\}$ by $k$ independent mapping hash functions $\{f_1, f_2, ..., f_k\}$.

$$g_1 = f_1(s), \quad g_2 = f_2(s), \ldots, \quad g_k = f_k(s) \quad \text{for all} \quad j, 1 \leq j \leq n$$

Then set the bit values in the $k$ bit positions denoted by $\{g_1, g_2, ..., g_k\}$ to one.
\[ A[g_1] = A[g_2] = \ldots = A[g_k] = 1 \]

To check if an element \( r \) is in the set \( S \), map \( r \) into \( k \) values \( \{g_1, g_2, \ldots, g_k\} \) and check if the bit values in bit positions \( A[g_1], A[g_2], \ldots, A[g_k] \) are all one. If all the bit values are one, the element \( r \) probably does exist in the set \( S \). Otherwise, the element \( r \) is surely not in the set \( S \).

Firstly, we generate the keyword set of sensitive data in the original database with no encryption applied. We store the keywords in a keyword set field which has character type values and will be eliminated later. Then, the \( m \)-length array is initialized to be all zero. Every keyword is mapped into index values by the mapping hash functions. The number of the index values is determined by number of mapping hash functions, which is described as parameter \( k \). Each index value describes a bit position in the bloom filter array, the bit value at that position is supposed to be set to one. The array is formatted into a string and stored in the character type field. We have a mapping procedure which takes keyword set as input and outputs the corresponding bloom filter string. Figure 2 shows the data processing setup before the encryption.

![Data processing before encryption.](image)

To implement the mapping operation, the following SQL statement is executed on the database server:

```sql
UPDATE [TableName] SET [IndexFieldName] = [MapFunction]('KeyWordSet')
```

The mapping functions are defined by executing the following SQL statement:

```sql
CREATE FUNCTION [dbo].[MapFunction] ( @KeyWordSet nvarchar(4000) )
RETURNS nvarchar(4000)
AS EXTERNAL NAME [Database].[UserDefinedFunctions].[MapFunction]
```

After completing the above operations, we delete values in the keyword set field. We then encrypt data in all field except for the index field acquired from the mapping functions.

When a client query arrives, each keyword in the keyword set in the query is mapped into its corresponding index values. Each keyword is mapped into \( k \) index values by \( k \) mapping functions. According to these index values, check the bit value in \( k \) bit positions designated by the \( k \) index values. If all the bit values are ones, it is implied that the keyword is contained in the record, which is returned as one record in the query result. Otherwise the current record is excluded from the result record set.

The examination of bit positions and bit values during the query process is handled by the SQL user-define function, which takes as input the keyword set in plaintext and returns a flag. The flag is set to 0 if the record doesn’t contain the searched keyword, otherwise the flag is set to 1.

To implement the operation, the following SQL statement is executed:

```sql
SELECT * FROM [TableName] WHERE [dbo].[SearchFunName]( 'KeyWordSet', 'bloom' ) = 1
```

The search function is defined by executing the following SQL statement.
CREATE FUNCTION [dbo].[SearchFunName]
    (@KeyWordSet nvarchar(4000),
     @bloomfilter nvarchar(4000))
RETURNS int
AS EXTERNAL NAME
    [Database].[UserDefinedFunctions].[SearchFunName]

The original client query is rewritten by substituting the original condition by the condition with
the search function above. After executing the rewritten query, the query result is returned to the
client, which needs decryption to get the final result of the query.

Experiment Results

We evaluate the performance of our scheme through a series of experiments. We follow the TPC-H
criterion and populate the database with 1G data using the DBGEN program provided by TPC-
H[12]. We build the experiment database on a Microsoft SQL Server 2008 server with Windows
Server 2008 as the operating system. The server is equipped with 8GB memory and an Intel
2.40GHz CPU. The programming tool we use is Visual Studio 2012. The functions are
implemented in C# language, compiled into a CRL program assembly in the Microsoft .NET
framework. We create two user-defined functions in the database system via the SQL Server’s
extension mechanism. One function is used for the mapping operation, the other is used as the
search operation. The two functions are associated with the CRL assembly so that the operations of
the SQL function are actually handled by the CRL binary.

We apply different query parameters to acquire different size of query result. Respectively, we
select 100, 1000, 2000 and 10000 tuples out of 10000 tuples. We also use different number of
keyword to construct the queries. The number of keyword we use varies from 1 to 6. As for the
parameter of the bloom filter, we set $k=5$ and $m=32$. We measure several key performance index
such as the number of result record, the index generation time, the query execution time and the
false positive rate. Figure III shows the results with encryption applied to numerical values. Figure
IV shows the results with encryption applied to character values.

![Graph showing results](image-url)
Figure 4. Results of character values.

Figure 5. False positive rates.

As shown in the figures above, with the increasing of number of result records, the index generation time and the query execution time increase in a linearly. The index generation time is more than the query execution time, which implies that after the system setup and the data preprocessing, the query execution doesn’t take much system resource and can be carried out efficiently.

Although the false positive does exits, the probability is relatively small and no more than 0.01. Figure 5 shows the actual false positive rate for different type of values, which we note as the false hit ratio in the figure.

With the false positive small enough, our scheme can notably improve the efficiency of query execution over encrypted database as no more data decryption is needed before the query.

Conclusions

We examine the problem of keyword based query execution in encrypted databases. We present a new method based on the bloom filter mechanism, it allows the database to apply any encryption algorithm to encrypt and protect sensitive data. Keywords are mapped into a bloom filter array, which is stored as the secure index for query processing. When querying the database, the original client queries are rewritten into queries that can be executed against secure indices. Conditions in the original query about encrypted attributes are substituted by conditions on secure indices. Query results are determined by checking if the mapping bit positions of the given keyword are all 1s in the bloom array. As we known so far, our work is the first to construct an efficient keyword search scheme in encrypted databases based on the bloom filter mechanism. Our method is encryption algorithm independent. Experiment results show that the method is efficient in query execution and can be applied to real-word applications.
Acknowledgment
This research is supported by the National Natural Science Foundation of China (Grant No. 61303197), Natural Science Foundation of Shandong Province (Grant No. ZR2013FQ005).

References
[6] Shi B L. Storage and Query over Encrypted Character and Numerical Data in Database [C] The Fifth International Conference on Computer and Information Technology, pp.77-81, 2005