*Journal of The Korea Institute of Information Security & Cryptology*
VOL.23, NO.6, Dec. 2013
ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2013.23.6.1259

# 클라우드 환경하에서의 안전한 데이터베이스 구축에 관한 연구*

김 성 용,[†] 김 지 홍[‡]
세명대학교

# A Study on the Secure Database Controlled Under Cloud Environment*

SungYong Kim,[†] Ji-Hong Kim[‡]
Semyung University

요 약

최근 데이터베이스의 크기가 점차 증가하고 있다. 기업에서는 데이터베이스 관리의 어려움으로 인하여 클라우드 형태로 아웃소싱하고 있으며, 이때 클라우드 서비스업체에 의해 관리되는 데이터베이스의 보안은 매우 중요하다. 데이터베이스 내의 중요 정보를 보호하기 위해서는 암호화하는 것이 최선의 방법이지만, 일단 암호화되고 난 후에는 더 이상 검색하기 어려워진다. 암호화된 데이터베이스에서의 검색 성능은 암호화 방법과 검색 방법에 의해 좌우된다. 본 논문에서는 가변길이의 키워드 인덱스와 블룸필터를 사용하여, 클라우드 서비스업체의 관리하의 데이터베이스에 대한 암호화 방법 및 효율적인 검색 방법을 제안한다. 마지막으로 제안된 방법이 데이터베이스 암호화 및 관련 응용분야에 유용하게 사용될 수 있음을 보인다.

ABSTRACT

Nowadays, the databases are getting larger and larger. As the company has difficulty in managing the database, they want to outsource the database to the cloud system. In this case the database security is more important because their database is managed by the cloud service provider. Among database security techniques, the encryption method is a well-certified and established technology for protecting sensitive data. However, once encrypted, the data can no longer be easily queried. The performance of the database depends on how to encrypt the sensitive data, and on the approach for searching, and the retrieval efficiency that is implemented. In this paper we propose the new suitable mechanism to encrypt the database and lookup process on the encrypted database under control of the cloud service provider. This database encryption algorithm uses the bloom filter with the variable keyword based index. Finally, we demonstrate that the proposed algorithm should be useful for database encryption related research and application activities.

**Keywords**: Bloom Filter, Bucket Index, Encrypted Database, Cloud

## I. Introduction

As the volume of the information is increasing very quickly, many organizations decided to outsource their database to an external service provider, namely cloud service provider. Another reason of out-

접수일(2013년 11월 4일), 게재확정일(2013년 12월 15일)
* 본 연구는 한국연구재단 연구과제(No. 2010-0023648) 지원으로 수행하였습니다.
† 주저자, ksy4765@naver.com
‡ 교신저자, jhkim@semyung.ac.kr(Corresponding author)

sourcing is to use the web service. As the database is not under the data owner's control, data confidentiality and integrity are of more concern in outsourced databases [1]. In order to protect the sensitive data, the best way to make it secure is to encrypt the sensitive data. Therefore, the manager of the service provider can execute queries only at the encrypted data.

There are lots of studies to execute queries on encrypted data, which are Bucket based, Privacy homomorphism, and OPES (Order Preserving Encryption Schema) methods and so on. The main idea is to use the additional index in order to search and get the wanted data from the encrypted database effectively. The first proposal was the bucket based indexing method using a number of buckets on the attribute domain [2]. Bucket based indexing methods are used to partition the attribute value range in a number of non-overlapping subsets of values, called buckets, containing contiguous values. The other major method was using homomorphism. This has also been proposed for allowing the execution of aggregation queries over the encrypted data in the database system [3]. Therefore, the operation on an aggregation attribute can be evaluated on the encrypted data at the server site and by decrypting the result at the client side. Most papers that discuss privacy homomorphism focus only on arithmetic operations rather than on comparison operations. An OPES is presented to support equality and range queries over the encrypted data [4]. Because the encrypted data has preserved order, equality and range query can be operated on the encrypted data in the database.

## II. Related Work

### 2.1 Database Encryption Methods

One of the important issues of the database encryption is which part of the database should be encrypted. The granularity level at which the database encryption is performed depends on the data that needs to be accessed. There are table level, attribute level, tuple level, and element level encryption methods [5]. Table level encryption methods are used to encrypt the table as a whole. Attributed level uses each column (attribute) in the plaintext table and represents this by a single encrypted value. Both methods are very simple and fast to implement, but tuples and attributes are indistinguishable in the released data and cannot be specified in a query on the encrypted database.

Tuple level is that each tuple in the plaintext should be represented by a single encrypted value. Element level involves each cell being represented by a single encrypted value. But it would require an excessive workload for data owner and clients.

So it is suitable to distinguish each element in the tuple and process query as long as we use an additional index. Therefore tuple level is suitable in order to balance the client workload and query execution in the server effectively. To summarize, Table 1. shows, for each encryption method, what kind of property is supported.

There are many queries to find the wanted data in the database. Among these queries, we focus on the equality query, range query and aggregation query only. Equality query is to find single matched data. Range query is to find the data included within the given range. Aggregation query is to find the numerical sum or the average value. These queries would not work on both table level and attribute level encryption. If we use element level encryption methods, it is suitable to find the equality value only. It can't be used to find the range value or aggregation value. So, if we use tuple level encryption with an additional index, it is possible to get the wanted value.

Table 1. The comparison on the encryption methods

| Encryption Method | Speed | Query | Size |
|---|---|---|---|
| Table level | Fast | X | Small |
| Attribute level | Fast | X | Medium |
| Tuple level | Medium | O | A little Large |
| Element level | Low | △ | Large |

## 2.2 Bucket Index Algorithm

Originally bucket based index algorithms were designed to execute the numerical data search in a plaintext database. It is very useful to use a bucket index in order to search the encrypted data on the database effectively.

Considering an arbitrary numerical attribute$A_i$ in $R_j$, with domain$D_{i,j}$, bucket-based indexing methods partition domain in a number of non- overlapping subsets of values, called buckets, containing contiguous values. This process, called bucketization, usually generates buckets that are all of the same size [2].

Each plaintext tuple $t(A_1,A_2,..,A_n)$ is mapped onto a tuple $t'(T_k,I_1,I_2,..,I_n)$, where $t'(T_k)=E_k(t)$. $E_k(t)$ is an invertible encryption function over plaintext tuple $t$ with key $k$, and $I_i$ corresponds to the bucket index over some $A_j$.

If we increase the number of buckets, the search speed is fast but not secure. An attacker can infer the range of the exact data from the detailed bucket value. So, the fatal drawback of the bucket index algorithm is the risk of the data exposure although it has good efficiency. If we assume that the attacker is aware of the distribution of plaintext values in the original database, the attacker can infer the plaintext value from the encrypted database. Data exposure rates of the encrypted table can be evaluated by looking at the distinguishable characteristics in the quotient table. Quotient tables show the frequency of the plaintext and IC (Inverse Cardinality) table shows the relative probability of each plaintext element respectively. Then, we can write the exposure rate $\epsilon$ associated with an encrypted relation of IC table as : $\epsilon=\frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{k}IC_{i,j}$. Here, $n$ is the number of the tuples and $i$ ranges over the tuples while $j$ ranges over the columns. This shows that an attacker could guess the plaintext as the number of the buckets is getting larger in the bucket index method [6].

## 2.3 Bloom Filter

A Bloom Filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries [7]. The Bloom Filter is an array of $m$ bits, initially all set to 0. The Bloom Filter uses $k$ independent hash functions with range $m$. The number of the sample space is $n$ elements. After all the elements are inserted in the Bloom Filter, the probability of the false positive error is

$$E_b=(1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k$$

This right-side expression is minimized for $k=\ln(2)\times(\frac{m}{n})$, in case the error rate is $(1/2)^k=(0.6185)^{\frac{m}{n}}$. Thus, the bloom filter is highly effective even for $m=cn$ using a small constant c. For example, if c=8, the false positive error rate is approximately 2%. We can adjust the false positive error rate by selecting proper $m$.

## III. The Proposed method

We use the tuple level encryption algorithm with key word based bloom filters in order to query the database effectively. Fig.1. shows the process of our proposed method. First of all, we extract the key word from each tuple. In word type data, it is easy to get the key word such as "seoul" in the address field. In numerical type data, we can extract the meaningful bytes from the numerical data as a keyword. For example, jumin field in std_info table consists of the birth year, birth month, and birth day. ″920226″ is one example. 92 means that birth year is 1992, 02 means that birth month is February, 26 means that birth date is 26th day. Now, we can use ″9y″ as the keyword in addition that represents the years between 1990 and 1999.

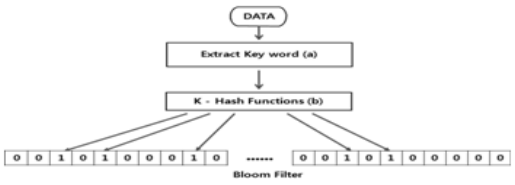


Fig.1. the bloom filter on proposed method

### 3.1 Encryption and Lookup Processes

#### 3.1.1 DB Encryption Process

Each row in the encrypted database has one E_tuple and one bloom filter. Data Encryption Process will be executed row by row.

(1) Create E_tuple: E_tuple is created by encrypting the plaintext tuple row by row. We use the AES algorithm as the encryption algorithm.

(2) Insertion of the bloom filter: The elements of the database are word type data or numerical data.

In the case of the word type data, we extract the keyword and the result of the hash function of the extracted keyword from each field in each row will be set in the bloom filter. In the case of the numerical type data, we extract the variable buckets as the keyword instead using the fixed bucket. It means high ordered bytes of the numerical data represents the range of the numerical data. Therefore, various kinds of keyword can be extracted. This data would help to analyze the statistics. The result of the hash function of the extracted keyword from each field in each row will be set in the bloom filter.

(3) Send the bloom filter values and the encrypted data to the server: Bloom filter value and the tuple based encrypted data will be stored in the encrypted database server.

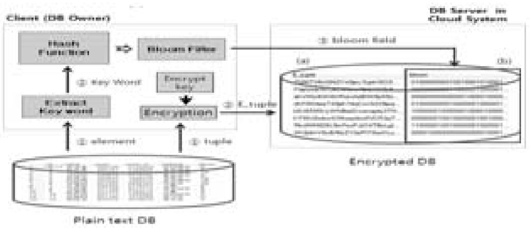


Fig.2. Data Encryption Process

#### 3.1.2 DB Lookup Process

(1) Transformation of user query statement: The key word included in the user query will be transformed to the result of the hash functions in order to work in the encrypted database with bloom filter. For example, if you want to search the tuple included ″seoul″ in an address fields, you have to calculate the hash function of the

word "seoul". The user query statement is modified to find the bloom filters that includes the resulting bits of the hash functions instead of "seoul"

(2) Search the bloom filter: Transformed query is transmitted to the database server including the bloom filter. At first, it searches the bloom filter fields that are bits of the bloom filter that are a result of the hash function set.

(3) Get the encrypted tuple (E_tuple): If the bits of the bloom filter are all set to '1", then the E_tuple will be sent to the client.

(4) Decrypt the encrypted data: The encrypted data received as a query response from the database can be decrypted by a decryption key in the client module.

The client module extracts the exact data from the encrypted result and sends the query result to the user as the last step.

### 3.1.3 DB Lookup Process for the DB analyst

Database analysts don't need to know the personal sensitive information except the statistics of the data distribution. So, this process does not include decryption processes from the encrypted database.

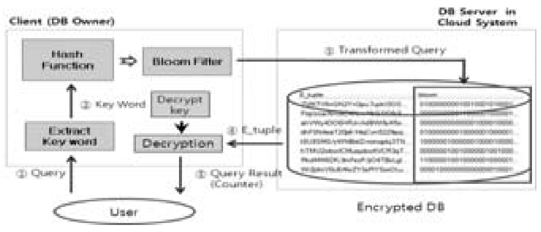


Fig.3. Data Look Up Process

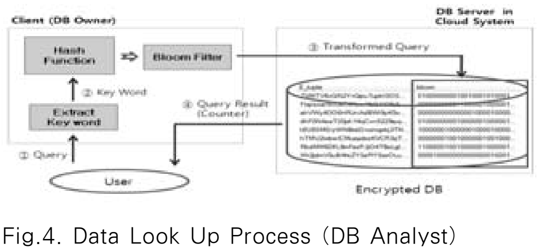


Fig.4. Data Look Up Process (DB Analyst)

## 3.2 Performance Test

### 3.2.1 The environment of the tesbed

We used two computers. One computer is used for MS SQL server and another notebook is used for client computer.

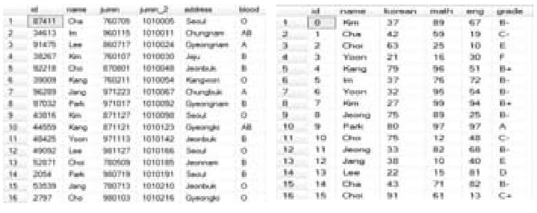
| Server : Desktop PC | Client : Notebook |
|---|---|
| Win 7 Ultimate K<br>Memory : 2.00GB, 32bit<br>Intel® Core(Tm) i5<br>CPU 650<br>MSSQL 2008 | Win 7 home premium K<br>Memory : 2.00GB, 32bit<br>Intel® Core(Tm)2 Duo<br>CPU T6600<br>MSSQL 2008,<br>Visual Studio 2010 C# |

The tables used in this performance test

We use the std_info and std_grade table shown in Table 2. to measure the performance of the proposed system

Std_info table in Table 2. stores the personal information data. Std_grade table in Table 2. stores the personal grade data. Table 3. are created encrypted databases by the bucket index method using bucket index and the proposed method using the various bucket indexes as the key word. E_tuple is the encrypted tuple data and bloom is the bloom filter value defined by

Table 2. 'Std_info'table and 'Std_grade' table.



|  | id | name | jumin | jumin_2 | address | blood |
|---|---|---|---|---|---|---|
| 1 | 87411 | Cha | 760705 | 1010005 | Seoul | O |
| 2 | 34613 | Im | 960115 | 1010011 | Chungnam | AB |
| 3 | 91475 | Lee | 860717 | 1010024 | Gyeongnam | A |
| 4 | 38267 | Kim | 760107 | 1010030 | Jeju | B |
| 5 | 92218 | Cho | 870801 | 1010048 | Jeonbuk | B |
| 6 | 39009 | Kang | 760211 | 1010054 | Kangwon | O |
| 7 | 96289 | Jang | 971223 | 1010067 | Chungbuk | A |
| 8 | 87032 | Park | 971017 | 1010092 | Gyeongnam | B |
| 9 | 43816 | Kim | 871127 | 1010098 | Seoul | O |
| 10 | 44559 | Kang | 871121 | 1010123 | Gyeongki | AB |
| 11 | 48425 | Yoon | 971113 | 1010142 | Jeonbuk | B |
| 12 | 49092 | Lee | 981127 | 1010166 | Seoul | O |
| 13 | 52871 | Choi | 780509 | 1010185 | Jeonnam | B |
| 14 | 2054 | Park | 980719 | 1010191 | Seoul | B |
| 15 | 53539 | Jang | 780713 | 1010210 | Jeonbuk | O |
| 16 | 2797 | Cho | 980103 | 1010216 | Gyeongki | O |

|  | id | name | korean | math | eng | grade |
|---|---|---|---|---|---|---|
| 1 | 0 | Kim | 37 | 89 | 67 | B- |
| 2 | 1 | Cha | 42 | 59 | 19 | C- |
| 3 | 2 | Choi | 63 | 25 | 10 | E |
| 4 | 3 | Yoon | 21 | 16 | 30 | F |
| 5 | 4 | Kang | 79 | 96 | 51 | B+ |
| 6 | 5 | Im | 37 | 76 | 72 | B- |
| 7 | 6 | Yoon | 32 | 95 | 54 | B- |
| 8 | 7 | Kim | 27 | 99 | 94 | B+ |
| 9 | 8 | Jeong | 75 | 89 | 25 | B- |
| 10 | 9 | Park | 80 | 97 | 97 | A |
| 11 | 10 | Cho | 75 | 12 | 48 | C- |
| 12 | 11 | Jeong | 33 | 82 | 68 | B- |
| 13 | 12 | Jang | 38 | 10 | 40 | E |
| 14 | 13 | Lee | 22 | 15 | 81 | D |
| 15 | 14 | Cha | 43 | 71 | 82 | B- |
| 16 | 15 | Choi | 91 | 61 | 13 | C+ |

Table 3. The encrypted table: 'Std_info_bucket' and 'Std_info_bf'

[illegible]

the result of the hash functions of the key words. We used three hash functions as SHA1, MD5, SHA256.

In the bucket index method, we used the fixed length bucket as usual [1,3]. Increasing the number of the buckets makes the bucket index method faster but insecure. We use the birth year field in std_info table by the 10 years unit.

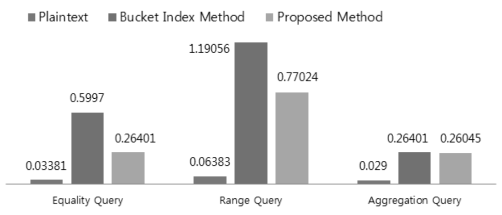
In our proposed method, we use m=256, n=8, k=3. Therefore, the false positive probability is approximately "0". In addition, we defined various buckets as keyword data instead of a bucket. For example, 10 year unit and 1 year unit as a key word of the numerical data are used to make bloom filter with the encrypted database.

### 3.2.3 Performance test on single data table

We executed the performance analysis on a single table using three kinds of the database, which are the plaintext database, the encrypted database with bucket index, and the encrypted database with bloom filter and tuple encryption. Queries used in this analysis are as follows.

(1) Equality query: Search that birth year is "1971"
(2) Range query: Search that birth year is more than 1980 and less than 1991.
(3) Aggregation query: Count the number that the birth year is 1971 and blood type is 'A'

Table 4. The result of three queries in single data table



In the bucket index method, we bucketized the birth year using the 10 years unit. So in the client module, we should decrypt the resulting data returned from the database server and choose the wanted data from the decrypted data. But in our proposed method, we are using various keyword data instead of the fixed sized bucket length. Therefore in the client module, we should decrypt the resulting data and choose the wanted data from the decrypted data. It is clear that the search performance of the proposed method is better than the bucket method by the number of query results.

In Table 4, it is evident that the proposed method has better processing time than the bucket index method.

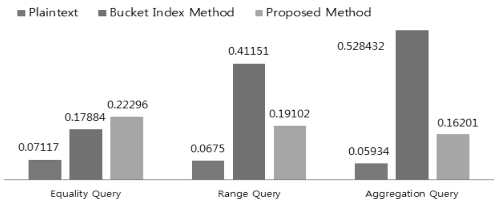
### 3.2.4 Performance test on two data tables using JIOIN operation

We executed the performance analysis on JOIN operation with two tables using three kinds of databases; the plaintext database, the encrypted database with bucket index, and the encrypted database with bloom filter. Queries used in this analysis are as follows.

(1) Equality query: Search that blood type is 'A' and grade is 'C'.
(2) Range query: Search that birth year is '1983' and Korean score is between 79 and 89.
(3) Aggregation query: Calculate the average mathematics score where the birth year is '1983'

The equality query shows that the bucket index method has better processing time than the proposed method. It is because 'A' and 'C' in query statements are used same bucket size.

Table 5. The result of three queries in two data tables



The search time actually depends on the searching range of the data included in the query statement. What is more important is that the proposed method has a little better performance and is more secure than the bucket index method.

Finally, we experiment with this proposed method on the case of the database analyst. Database analysts don't need to know the personal sensitive data. So they need not to have the decryption key. This process is very simple than the normal search process. Without the decryption process, they can achieve their job. The query used in this analysis is "Find the age distribution of the person who live in "seoul". In the plaintext database, it takes 0.025168 sec as opposed to 0.265215 sec in the proposed method. False positive probability of the bloom filter is approximately "0" because we use the length of the bloom filter as 256 bits. In all, the bucket index method can't get the exact data without the decryption process. Only our proposed method can get the exact data without a decryption process.

## IV. CONCLUSION

There are many database encryption algorithms around. Among these algorithms, tuple based encryption algorithm using bucket index is widely used. The Bucket index algorithm is generally faster to search the true numerical data from the encrypted database than other conventional encryption methods in the range query. However the fatal drawback of the Bucket based method is the probability of the data exposure. If we increase the number of the bucket, it takes a little search time but an attacker can easily infer the plaintext with the bucket value. The bucket index method is a very good method for numerical data indexing, but it is not secure and takes more time to search for the exact data because it uses only fixed length buckets.

As we demonstrated previously, it shows a good processing time to execute three queries (equality query, range query, aggregation query) in a single table and join the operation in two tables. The only drawback of the proposed method is that it takes about 0.15 sec more than the bucket index method to calculate hash functions and search the bloom filter bit by bit. If we use the database server with good quality, we may get better performance. In our opinion, it is not important because it provides a secure database. Although our examples may be simple, the result of our performance test shows the superiority of our mechanism, which can be applied to another example. Bloom data in our mechanism can be used to analyze data distribution for statisticians.

We have compared the query processing time between the plaintext and the bucket method and the proposed method using various kinds of queries. The database using the proposed method only shows encrypted data and the bloom filter. Attackers who want to access the data from the encrypted database and bloom filter can't guess and find the actual data without the decryption key. In future we hope to improve and develop our proposed mechanism and in addition, study the security of the proposed method even if the bloom filter in our proposed mechanism seems to be secure at first sight.

## References

〔1〕 S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, "Privacy of Outsourced Data," IFIP Vol.320, pp 174-187, 2007.

〔2〕 H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database service provider model," In Proc. of the ACM SIGMOD, pp 216-227, 2002,

〔3〕 H. Hacigumus, B. Iyer, and S. Mehrotra, "Efficient execution of aggregation queries over encrypted relational databases," LNCS 2973, pp 125-136, 2004.

〔4〕 R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, "Order preserving Encryption for numerical Data," Proc. of the ACM SIGMOD, pp 564-574, 2004.

〔5〕 L. Bouganim, Y. Guo, "Database Encryption," Springer (Ed.),1-9, 2009.

〔6〕 E. Damiani, S. DeCapitani di Vimercati, S. Jajordia, "Balancing confidentiality and efficiency in untrusted relational DBMS," Proc. of the 10th ACM (CCS03), 93-102, 2003.

〔7〕 A. Broder and M. Mitzemacher, "Network Applications of Bloom Filters : A Survey," Internet Mathematics Vol.1, No.4, pp 485-509, 2003.

## 〈저 자 소 개〉

김 성 용 (SungYong Kim) 학생회원
2013년 2월: 세명대학교 정보통신학부 졸업
2013년 3월~현재: 세명대학교 정보통신학 석사과정
〈관심분야〉 정보보호 응용, 데이터베이스 보안, 의료정보 보안

김 지 홍 (Ji-Hong Kim) 종신회원
1982년 2월: 한양대학교 전자공학과 졸업
1984년 2월: 한양대학교 전자통신공학과 석사 졸업
1996년 3월: 한양대학교 전자통신공학과 박사 졸업
1991년 3월~현재: 세명대학교 정보통신학부 교수
〈관심분야〉 네트워크 및 정보보호, 정보보호 응용, 데이터베이스 보안