Advanced Science and Technology Letters
Vol.93 (Security, Reliability and Safety 2015), pp.34-39
http://dx.doi.org/10.14257/astl.2015.93.08

Practical Database Encryption Scheme for Database-as-a-Service

Hankyu Joo[1]

[1] Dept. of Computer Engineering, Hallym University,
Chuncheon, Korea
hkjoo@hallym.ac.kr

Abstract. Database-as-a-Service (DBaaS) allows clients to use expensive database management system without purchasing it. In DBaaS environment, database tables are stored at provider's server. Since the database tables are stored at servers, the tables need to be encrypted to have data confidentiality. However, the encryption of data introduces performance degradation in executing queries over encrypted data. Especially executing range queries faces severe performance degradation. In this paper, a new database encryption scheme for DBaaS is proposed. The scheme allows range queries without severe performance degradation and without information leaks except order information.

Keywords: database as a service, database encryption, order preserving encryption.

1 Introduction

Nowadays computing as a service is gaining its ground. Clients may use the service without purchasing the system supporting the service. One of the area of computing as a service is Database-as-a-service (DBaaS). With DBaaS, clients may use expensive database management system without purchasing the system.

In DBaaS, clients store their data at servers which belong to the service providers. The servers are not under the control of clients. There is no guarantee of confidentiality of the data stored at the servers. The clients need to encrypt the data to have data confidentiality. However, the encryption of data introduces performance degradation in executing queries over encrypted data. To execute a range query, the server need to decrypt each of the items encrypted to find out whether the item is in the range. Since the index is maintained on the encrypted data, the index may not be used for performance enhancement.

Several approaches have been proposed to alleviate the performance degradation. To alleviate the performance degradation, the server should be able to perform the order comparison without the decryption of the encrypted items. To allow the server to perform order comparison without decryption, Order Preserving Encryption (OPE) schemes have been proposed. Some of the proposed OPE schemes allow servers to perform order comparison without decryption, but they leaks information besides order. Other schemes require the server to perform operations which are not database operations. The server should support such operations.

In this paper, database encryption scheme without severe performance degradation is proposed. The proposed scheme only leaks the order to the adversary and uses only database operation of the server.

2 Related Works

There has been a large amount of work on database encryption. Database encryption has been faced with severe performance degradation in range query. The work on database encryption has been mainly about relieving the performance degradation. OPE was suggested for database encryption by Agrawal et al. [1] to relieve the performance degradation. Since the introduction of OPE by [1], a lot of work has been done about OPE [2, 3, 4, 5]. OPE allows the encryption function preserves the numerical ordering of the plaintexts. Similar scheme called order preserving encoding is also proposed [6]. Order preserving encoding scheme maintains tree-structured order information at server. As indicated in [6], most of the proposed OPE schemes leak information besides order. The other scheme proposed in [6] requires operations, which are not general database operations, performed on the server.

3 Proposed Scheme

3.1 Consideration

The design of a new scheme is based on the following consideration. In DBaaS, all database tables are stored in the server. The cost of storage is not expensive. Database tables are initialized with many items. Query operations are frequent operation. Inserting an item is not frequent. Indexing is used to improve the query speed. The server is accessed by database operation interfaces.

3.2 Database Encryption Scheme

The proposed scheme uses an extended table. When a table is created and placed in a server, each sensitive column (x) of a database table is encrypted (eX) and an additional order column (oX) is added. The encryption may be performed by any existing secure encryption algorithm such as AES [7]. The order column (oX) is used to maintain the order of the sensitive column (x). Suppose that an item, m, has a value x_m in column x and oX_m in column oX. And also suppose that an item, n, has a value x_n in column x and oX_n in column oX. If $x_m < x_n$ then $oX_m < oX_n$. When a table is created, the order column has a big interval. Although x_n is next to x_m in order, $oX_n - oX_m$ is not

1 but large numbers such as 100. Indexes are used whenever necessary. For example, consider an employee table as Table 1.

Table 1. employee table in plaintext.

ID	Name	Title	Salary
650213-1234567	Hankyu Joo	Manager	14,000
690313-2345678	Sangmin Han	Programmer	12,000
700225-1234567	Jaewook Choi	Programmer	15,000
….	….	….	….

ID column and *Salary* column contain confidential information. To give security, items in *ID* column and *Salary* column are encrypted. Column names are renamed as *eID* and *eSalary* as Table 2. *oID* column and *oSalary* column are also added to have order information of *ID* and *Salary*.

Table 2. employee table in ciphertext..

eID	Name	Title	eSalary	oID	oSalary
eX12Klm	Hankyu Joo	Manager	tyK3xDs	200	600
jMcD38h	Sangmin Han	Programmer	wQ1B0ld	700	300
y08b1xK	Jaewook Choi	Programmer	d2IpRz9	2100	900
….	….	….	….	….	….

Indexes are created for *eID*, *eSalary*, *oID* and *oSalary* column.

For the original table, range query may be performed. For example the following query may be performed to get names of salary between *12,000* and *15,000*. *Salary* column may be indexed to improve the query speed.

```
Select Name
from employee
where Salary >= 12,000 and Salary <= 15,000
```

If the *Salary* column is encrypted as *eSalary* column and if *oSalary* column maintains the order of *Salary*, the query should be modified as follows.

```
Select Name
from employee
where
oSalary >= minLarge(employee, eSalary, oSalary, 12,000)
        and
oSalary <= maxSmall(employee, eSalary, oSalary, 15,000)
```

The function *minLarge(table, field, order, value)* is used to get the *order* (*oSalary*) of an item which has the minimum *field* in plaintext (*Salary*) larger than *value* (*12,000*). The function *maxSmall(table, field, order, value)* is used to get the *order* (*oSalary*) of an item which has the maximum *field* in plaintext (*Salary*) smaller than *value* (*15,000*). The function *minLarge(table, field, order, value)* is as follows.

```
int minLarge(TableName eTable, ColumnName field,
ColumnName order, int value) {
  int min = select MIN(order) from eTable;
  int max = select MAX(order) from eTable;
  return internalMinLarge(eTable, field, order, min, max,
    value);
}

int internalMinLarge(TableName eTable, ColumnName field,
ColumnName order, int min, int max, int value) {
  int dF, oX;
  if (min >= max) return min;
  int mid = (min+max)/2;
  (dF, oX) = select decrypt(field), order
                from eTable
                where order >= mid
                order by order
                limit 1;
  if (value == dF) return oX;
  else if (value < oX) return internalMinLarge(eTable,
     field, order, min, mid-1, value);
  else return internalMinLarge(eTable, field, order,
     oX+1, max, value);
}
```

The function *minLarge(table, field, order, value)* is as follows.

```
int maxSmall(TableName eTable, ColumnName field,
ColumnName order, int value) {
  int min = select MIN(order) from eTable;
  int max = select MAX(order) from eTable;
  return internalMaxSmall(eTable, field, order, min, max,
    value);
}


int internalMaxSmall(TableName eTable, ColumnName field,
ColumnName order, int min, int max, int value) {
  int dF, oX;
  if (min >= max) return max;
  int mid = (min+max)/2;
  (dF, oX) = select decrypt(field), order
                from eTable
                where order <= mid
                order by order DESC
                limit 1;
  if (value == dF) return oX;
  else if (value < oX) return internalMaxSmall(eTable,
    field, order, min, oX-1, value);
  else return internalMaxSmall(eTable, field, order,
    mid+1, max, value);
}
```

When a new item is inserted, a value for the order column should be calculated. The value of order column for the new item is the middle value of the neighboring two order values.

Since the encryption of a column uses existing secure encryption algorithm, this scheme leaks only order information. This scheme uses only standard query operations to access the server. This scheme does not require any change in server. The performance of range query is O(m), where m is the number of selected items. Without any order information, the performance of range query on encrypted column is O(n), where n is the number of total items in the table. By adding the order column to the encrypted column, the performance of the range query on the encrypted column can be improved to O(m+log_n). If many items are inserted at a specific range of value, the order column may be restructured. By inspecting order column periodically, clients may know the status of order column.

4 Conclusion

In this paper, a new scheme for database encryption is proposed. The proposed scheme may be implemented on the clients' side only without requiring any change to the server side. The proposed scheme does not leak any information except the order of the encrypted field. The proposed scheme may perform a range query on the encrypted column without severe performance degradation. The proposed scheme requires addition columns to the encrypted table. However, the additional storage for the additional column is not expensive in DBaaS environment. The proposed scheme requires periodic inspection of the order column and restructuring of the column if necessary. However, database insertion is not frequent operation, restructuring may only be necessary when the original table is under major update.

References

1. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order Preserving Encryption for Numeric Data. ACM SIGMOD, pp. 563--574. (2004)
2. Boldyreva, A., Chenette, N., Lee, Y., O'Neil, A: Order Preserving Symmetric Encryption. Eurocrypt, LNCS 5479, pp. 224--241. (2009)
3. Boldyreva, A., Chenette, N., O'Neil, A: Order Preserving Symmetric Encryption Revisited: Improved Security Analysis and Alternative Solutions. Crypto 2011, LNCS 6841, pp. 578--595. (2011)
4. Yum, D., Kim, D., Kim, J., Lee, P., Hong, S.: Order-Preserving Encryption for Non-Uniformly Distributed Plaintexts. WISA 2011, LNCS 7115. 84--97. (2012).

5. Liu, D., Wang, S.: Nonlinear Order Preserving Index for Encrypted Database Query in Service Cloud Environments. Concurrency and Computation: Practice and Experience, 25, pp. 1967--1984. (2013).
6. Popa R.A., Li, F.H., Zeldovich N.: An Ideal-Security Protocol for Order-Preserving Encoding. IEEE Symposium on Security and Privacy, pp. 463--477. (2013)
7. National Institute of Standard and Technology: Advanced Encryption Standard (AES). FIPS PUB 197. (2001)