IEICE TRANS. INF. & SYST., VOL.E93–D, NO.9 SEPTEMBER 2010

PAPER

MV-OPES: Multivalued-Order Preserving Encryption Scheme: A Novel Scheme for Encrypting Integer Value to Many Different Values

Hasan KADHEM[†a)], ***Nonmember***, **Toshiyuki AMAGASA**[†,††], ***Member***, ***and*** **Hiroyuki KITAGAWA**[†,††], ***Fellow***

SUMMARY Encryption can provide strong security for sensitive data against inside and outside attacks. This is especially true in the "Database as Service" model, where confidentiality and privacy are important issues for the client. In fact, existing encryption approaches are vulnerable to a statistical attack because each value is encrypted to another fixed value. This paper presents a novel database encryption scheme called MV-OPES (Multivalued—Order Preserving Encryption Scheme), which allows privacy-preserving queries over encrypted databases with an improved security level. Our idea is to encrypt a value to different multiple values to prevent statistical attacks. At the same time, MV-OPES preserves the order of the integer values to allow comparison operations to be directly applied on encrypted data. Using calculated distance (range), we propose a novel method that allows a join query between relations based on inequality over encrypted values. We also present techniques to offload query execution load to a database server as much as possible, thereby making a better use of server resources in a database outsourcing environment. Our scheme can easily be integrated with current database systems as it is designed to work with existing indexing structures. It is robust against statistical attack and the estimation of true values. MV-OPES experiments show that security for sensitive data can be achieved with reasonable overhead, establishing the practicability of the scheme.

key words: *encryption, order-preserving, database outsourcing, statistical attack*

1. Introduction

Critical and sensitive business data in databases is an obvious target for adversaries. Therefore, an appropriate level of protection for database content must be provided. The traditional solution for database security is to apply an access control mechanism by assigning sorts of rights, logins, roles and passwords to restrict queries and application usage. Those mechanisms, while important and necessary, cannot ensure that a database is immune to intrusion and unauthorized access.

Encryption can provide strong security for sensitive data against inside and outside attacks. The primary interest in database encryption results from the recently proposed "database as service" (DAS) architecture [1]. In DAS or database outsourcing, a database owner outsources its management to a "database service provider", which provides online access mechanisms for querying and managing the hosted database. At the same time, the service provider incurs most of the server management and query execution load.

Clients would like to take advantage of the provider's robust storage, but in many cases clients would like to protect the database even from the service provider. Specifically, the provider should be prevented from observing any of the outsourced database contents. Encryption is a common technique used to protect the confidentiality and privacy of stored data in the DAS model.

The encryption process in current known schemes that provide privacy queries over encrypted databases is performed at various levels [2]: page/block, columns, rows and attributes. Usually, encrypting the whole page/block, column or row entails the decryption of the whole page/block, column or row when a query is executed to retrieve a piece of information, even if additional indices are stored within the database such as in [3], [4]. Therefore, an implementation that encrypts the attributes individually provides flexibility to decrypt only the data of interest [5]. However, the traditional attribute level encryption approach for a database encrypts each value to another fixed value: $X_1 = X_2 \Longrightarrow E^k(X_1) = E^k(X_2)$

This approach is vulnerable to statistical attacks. A statistical attack against an encrypted database seeks to use some apparently anonymous statistical measures to infer individual data. Using such an approach, an attacker, especially an inside attacker, can infer some data by joining tables and using additional statistical information. This problem appears clearly in joining lookup tables with other tables. The lookup tables usually consist of a small and fixed values scale or domain such as gender (male or female), marital status (single, married, separated,...), and city.

Example 1: Consider the plaintext database shown in Fig. 1 (a) with the traditional encrypted database in Fig. 1 (b). In the encrypted employee table, the third column contains two distinct values (67653, 564564), which are clearly gender data. When an attacker knows that there are more male employees than female employees, then the attacker can infer the encrypted values for numbers 1 and 2. Also, it is possible to infer information by joining a

Manuscript received February 2, 2010.
Manuscript revised May 8, 2010.

[†]The authors are with Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba-shi, 305–8573 Japan.
[††]The authors are with Center for Computational Sciences, University of Tsukuba, Tsukuba-shi, 305–8577 Japan.
a) E-mail: hsalleh@kde.cs.tsukuba.ac.jp

DOI: 10.1587/transinf.E93.D.2520

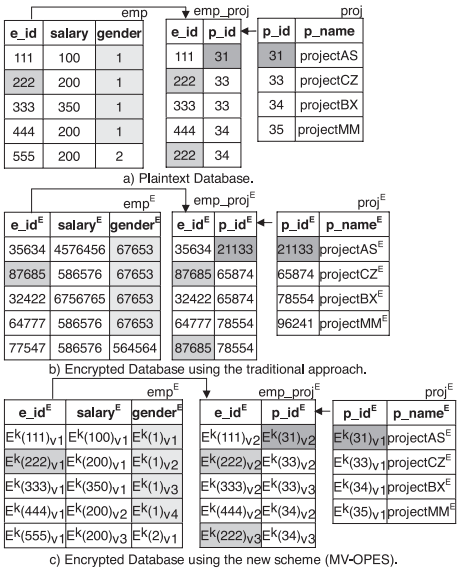

Fig. 1 Database encryption using two approaches.

project table with the (emp_proj) table. Knowing that only one employee works on project "projectAS", the attacker can recognize the encrypted value for that project from table (emp_proj). Also, the attacker can get the encrypted (emp_id) for the employee who works on that project. Using the same technique, the attacker can infer much more information from the encrypted database, and then try to determine the key used in the encryption process.

1.1 Traditional Solutions

A straightforward solution to solve the problem of statistical attack is to use different encryption keys for different fields. Actually, this solution is good to avoid direct joins between tables in the encrypted database. However, within a field, a particular plaintext is still encrypted to the same value. In addition, using different keys leads to serious performance degradation, because we need to decrypt all data to execute queries, such as join.

Another naive solution is to add to the database a randomly generated data value for each plaintext value and then both values are encrypted together. This approach is used by [6] to encrypt XML documents. Also, it is used with the Cipher-Block Chaining (CBC) [7] as initial vector IV. This approach leads to storage cost. In addition, it causes serious performance degradation because the plaintext and the additional value associated to the plaintext should be decrypted together before executing any query.

1.2 Our Contributions

This paper presents a new database encryption scheme called MV-OPES (Multivalued-Order Preserving Encryption Scheme), which allows one integer to be encrypted to many values using the same encryption key while preserving the order of the integer values. Figure 1 (c) describes the new encryption technique applied on the employee plaintext database shown in Fig. 1 (a). The equal plaintext values in Fig. 1 (a) are encrypted to different ciphertext values in Fig. 1 (c) as opposed to the ciphertext values in Fig. 1 (b). Here, $E^K(n)_{v_i}$ denotes ciphertext value of n with encryption key K, and subscript v_i means the variation of the ciphertext value. For instance, the emp_id (222) is encrypted to $E^K(222)_{v_1}$ in the emp table and $E^K(222)_{v_2}$, $E^K(222)_{v_3}$ in the emp_proj table, with high probability that $(E^K(222)_{v_1} \neq E^K(222)_{v_2} \neq E^K(222)_{v_3})$. Also, the encrypted values for the emp_id (111) are always less than the encrypted values for the emp_id (222). In this scheme, attackers cannot infer individual information from the encrypted database even if they have statistical knowledge about the plaintext database. Using calculated distance (range), we propose a novel method to join different encrypted values between relations based on inequality.

Compared with previous solutions, the advantages of our scheme (MV-OPES) are the following:

- MV-OPES is robust against statistical attack and the estimation of true values compared with the traditional order preserving encryption schemes like OPES [6] and Anti-tamper database [8], [9].
- It allows queries to be executed over an encrypted database without decrypting the operands. It supports a practical subset of relational operators. Although some operators require post-processing to eliminate false-positives, MV-OPES minimizes the amount of work done on the decrypted data. We support long list of relational operators over the encrypted database compared with the schemes in [5], [10] that support just the basic selection operation.
- Unlike the bucketing approach [3], [4], which generates a superset of answers with false positive tuples in all queries, our scheme does so only on some condition types; the results contain false positive tuples.
- MV-OPES can make a better use of resources on both server and client sides by splitting the query into server-side and client-side queries, while [5] allows queries only over the trusted server.
- MV-OPES can easily be integrated with current database systems as it is designed to work with existing indexing structures such as B-trees. In previous encryption schemes, a new metadata [10] or a new indexing structure [5] is needed to perform queries over encrypted database.
- Unlike the encryption schemes in [3]–[6], [10], MV-OPES is efficient in insertions and updates. A new value can be inserted, or an existing value can be modified without requiring changes to the encryption of other values.
- The database user in our scheme does not need to manage large amount of confidential data (like the bucketing approach [3], [4], the large keys in the OPES [6] or many different secret keys in [6], [10]). In our scheme, a user only needs to secure the secret key and a small amount of secret variables used in the encryption pro-

cess.

The results from an implementation of MV-OPES show that security for sensitive data can be achieved with reasonable overhead and false positives, establishing the practicability of the scheme.

1.3 Organization of the Paper

The rest of paper is organized as follows. We first discuss related work in Sect. 2. Section 3 introduces our new database encryption schema. Section 4 describes the condition translations. Section 5 discusses implementation of the relational operators such as selection, join, and sort over encrypted relations. Then, Sect. 6 addresses the security issues of MV-OPES. Section 7 reports the experimental results. We conclude with a summary and directions for future work in Sect. 8.

2. Related Work

Many database encryption techniques have been proposed, but most are not intended for the one-to-many encryption algorithm.

Order preserving encryption scheme (OPES): The idea of OPES is to take as input a user-provided target distribution and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution [11]. Under OPES, it is ideal from the viewpoint of query performance, because comparisons can operate directly on ciphertext, thereby saving the cost of expensive decryptions. From a security point of view, OPES is vulnerable to a tight estimation and statistical attacks if an adversary has knowledge of the distribution. Also, OPES is vulnerable to chosen plaintext attacks if the adversary can choose any number of unencrypted values to his liking and encrypt them into their corresponding encrypted values. In OPES, most probably two columns of two tables do not have the same distribution. That means they are not directly comparable. When this is true, the join query involves expensive decryptions and/or encryptions, because one side must be converted to the other.

Order preserving encryption with splitting and scaling (OPESS): The authors in [6] proposed a new encryption scheme based on the OPES to index the encrypted values in the outsourced XML databases. The idea in OPESS is to map the same plaintext values to different ciphertext values to protect the data against frequency-based or statistical attacks. OPESS consists of two stages splitting and scaling. In splitting, each plaintext value is encrypted into one or more ciphertext values by using different keys. The number of keys used to encrypt a plaintext value is based on the number of occurrence for the plaintext. In scaling, the number of occurrences of encrypted values is multiplied by a scale factor. One of the limitations of OPESS is that security achieved by scaling encrypted data causes an increase in data size. Also, this approach is not efficient in insertions and updates because the encryption method is mainly based on the number of occurrences. OPESS proposed mainly for XML database but it is not applicable for relational database. The reason behind that is the different in executing queries on the relational and the XML databases such as the join operation between different encrypted values.

Anti-tamper database: In [8], [9], a sequence of polynomial functions is used to encrypt integer values while preserving the order. The decryption is made by solving the inverses of each polynomial function in the sequence in reverse order. The degree of security in this approach depends on the number of coefficients employed by the polynomial function sequence and the set of constants in the encryption function. Unfortunately, the cost of encrypting or decrypting in an anti-tamper database can be prohibitive for large values. Also, in this scheme the shape of the distribution of encrypted values follows the shape of the input distribution [11]. From a security perspective, this could reveal information about the input distribution, which can be broken.

Structure preserving database encryption scheme: The authors in [5] proposed a new encryption scheme that breaks the correlation between ciphertext and plaintext values by encrypting each database value with its unique cell coordinates. There are two immediate advantages to this scheme. First, it eliminates substitution attacks attempting to switch encrypted values. Second, pattern matching attacks attempting to gather statistics based on the encrypted values will fail. However, this scheme can be used only on a trusted server, where a DBA can manage the new index structure in the encrypted database. The join operation is not covered by this approach. We think that the only way it can be used to join tables in this scheme is to decrypt tables first, and then perform the join over the decrypted database, which adds overhead. Moreover, if a database reorganization process changes cell coordinates, all affected cells need to be re-encrypted with their new coordinates.

Privacy-Preserving Queries on Encrypted Data: The encryption algorithm proposed in [10] appends a random string to the plaintext before encryption. Then both values are encrypted together, so multiple occurrences of a plaintext lead to different ciphertexts. This scheme is useful to protect the encrypted databases against statistical attack. While in this paper we cover many relational operations over encrypted database including various conditions, the authors in [10] analyze just the selection queries with a basic condition (*Attribute = value*). The authors proposed two solutions for basic selection queries. The experiments of the first solution show that overhead is significant because there is no index or metadata used to retrieve records. In the second solution, they replace the sequential search in the first solution with a binary search using a new encrypted metadata. The efficiency improved significantly in the second solution but there is no security proof for the two-step mapping method used in the new encrypted metadata. We think that an adversary can break the encrypted metadata with little knowledge about the plaintext values. In addition, there is no explanation about update on the encrypted database.

Keyword search on encrypted data: Many researchers have investigated the problem of keywords searching on encrypted data using either symmetric encryption [12], [13], asymmetric encryption [14] or a combination of symmetric, asymmetric encryption and hash functions [15]. In spite of security vulnerabilities (like statistical attack in [12]) and significant overhead in [14], [15], these encryption schemes are possibly useful in searching for keywords in a file, document or email. However, these solutions can not be applied to the problem of efficiently querying encrypted relational databases. Especially, we discuss in this paper the problem of encrypting integer data, executing range queries and implementing relational operations over encrypted database.

Homomorphic Encryption: A homomorphic encryption [16]–[19] function allows performing arithmetic function on two (or more) ciphertexts to produce a new ciphertext, without having any information about the plaintext or encryption/decryption keys. The homomorphic encryption is related to our work in sense of performing the arithmetic function on the ciphertext. However, in this paper we focus on performing different queries over encrypted database (Sect. 5) which are more used in real word than the arithmetic functions.

Other Relevant work: In [20], [21], the authors proposed the Chip-Secured Data Access (C-SDA) principle to insulate data encryption, query evaluation and access right management in a Secured Operating Environment (SOE). C-SDA is a client-based security component acting as an incorruptible mediator between a client and an encrypted database. This component is embedded into a smartcard to prevent tampering on the client side. However, C-SDA does not support direct execution for a range query because it requires a disjunction to be created for every possible value in the range, which is not feasible for real data values [11].

Another variation to secure databases that has recently been studied is that of "distributed architecture" [22]–[24] for enabling privacy-preserving outsourced storage of data. However, distributing the database content to many servers is not the concern of this body of work. We focus on securing the central database content stored in a trusted, untrusted, or semi-trusted [25] server. The encryption scheme proposed in this paper (MV-OPES) is a novel technique to encrypt an integer value to many different values, and in the sense of performing queries over an encrypted database based on inequality.

3. Proposed Multivalue Order Preserving Encryption Scheme

The basic idea of MV-OPES is to take a user provided domain as the input and transform the plaintext domain values in such a way that the transformation preserves the order and the equal plaintext values are encrypted to different ciphertext values. This section describes our encryption scheme in detail.

3.1 An Overview of MV-OPES

Before the discussion, we need to look at two important concepts:

- **Order preservation:** An encryption function E using a key K is called order preserving, if for all X and Y such that $X < Y$, $E^K(X) < E^K(Y)$.
- **Multivaluedness:** Every input is associated with one or more outputs. For given integers X_1 and X_2 such that $X_1 = X_2$, the encrypted values using function E and key K are $E^K(X_1)$ and $E^K(X_2)$, such that $E^K(X_1) \neq E^K(X_2)$ with high probability.

When encrypting plaintext values in a column having values in the range $[D_{min}, D_{max}]$, we generate boundaries for all integers in the domain $(B_{D_{min}}, \ldots, B_{D_{max}}, B_{D_{max+1}})$ using the increasing/decreasing function (order preservation). The generated boundaries identify the intervals. For instance, interval I_i is identified by $[B_i, B_{i+1})$. We then generate the encrypted values for integer i as random values from the interval I_i (multivaluedness). Details regarding the random distribution used to choose the encrypted values are discussed in Sect. 3.3.2.

3.2 Generation of Bucket Boundaries

Bucket boundaries are generated using two functions: initial and increasing/decreasing. Details of these two functions are discussed next.

3.2.1 Initial Function

We are given a domain $[D_{min}, D_{max}]$, with $(D_{max} - D_{min} + 1)$ integers: $\{D_{min}, D_{min+1}, \ldots, D_{max}\}$. Initially, we choose the starting (*initial*) point from the domain. We then compute the boundary for the initial point using the following function:

$$B_{initial} = Enc^K(initial)$$

where *Enc* is the function used to encrypt the (*initial*) value using key K. Any block cipher algorithm such as DES [26], TDES, Blowfish [27], AES [28], RSA [29], or a hashing function can be used to encrypt the value.

3.2.2 Increasing/Decreasing Function

To preserve the order of the integers, we use two functions to generate boundaries. First, boundaries for values greater than the initial point are generated by an increasing function. Second, a decreasing function is used to generate boundaries for values less than the initial point. The goal for the increasing/decreasing function is to create encrypted interval scales for all integers in the domain with different sizes. Differences in intervals size are ensured by predefined percentage and a sequence of random numbers.

Given the initial point (*initial*), the interval size IS, and the difference percentage on the encrypted interval size DP,

the boundaries are derived by the following function:

$$B_i = \begin{cases} B_{i+1} - (Enc^K(IS) + (Enc^K(IS) * DP) * R_i), \\ \qquad D_{min} \le i < initial \\ B_{i-1} + (Enc^K(IS) + (Enc^K(IS) * DP) * R_i), \\ \qquad initial < i \le D_{max+1} \end{cases}$$

where R_i is a sequence of random numbers in the range $[-1, 1]$. There are many pseudorandom number generators with useful security properties [30], [31]. The DP used in the formula to control the differences between intervals size that will be in the range $[-Enc^K(IS) * DP, Enc^K(IS) * DP]$. Figure 2 shows the differences between interval sizes in $[1, 100]$ domain, $DP = 0.05\%$, and $IS = 1000$.

3.3 Encryption Functions

Here we discuss how to encrypt a plaintext relation R. For each tuple $t = (A_1, A_2, \ldots, A_n)$ in R, the encrypted relation R^E stores a tuple:

$$(E(A_1), E(A_2), \ldots, E(A_n))$$

where E is the function used to encrypt an attribute value of the tuple in the relation. The encryption function $E(i)$ is applied by choosing a random number in the interval I_i, which is identified by $[B_i, B_{i+1})$. As for the distribution of random values, it should be carefully chosen by taking into account the type of attribute being encrypted. If it is a primary key attribute, the occurrence of a value is at most one, whereas there may be several occurrences for a value in foreign key attributes. We therefore classify database attributes as "related" or "unrelated". In the former, it has one-to-many relationship with another attribute in another relation, such as *emp_id*. Other attributes, such as *salary* are called unrelated.

In MV-OPES, the integer value is encrypted to many values, so in some relational operators such as grouping or in the case of a condition between two attributes, we need to have a related value for all different encrypted values. Thus, in some cases even for unrelated attributes we create an independent table as a primary key containing one value for each interval in the domain. Many unrelated attributes can have the same table as a primary key if they have the same domain.

3.3.1 Comparing Ciphertext Using MaxDiff

Here, we explain the process of comparing two different attribute values. This explanation is required before the discussion of choosing the random distribution. When comparing two different attribute values in query processing, a straightforward approach is to decrypt the ciphertext first, then compare the plaintext values, which may lead to performance degradation. Also, this approch is not applicable for DAS model. Instead, we try to make it possible to compare ciphertext without decryption.

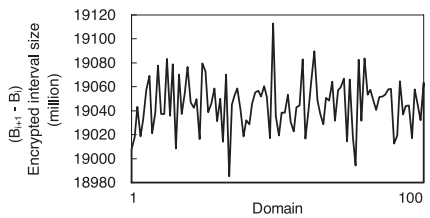

Fig. 2 Differences between intervals size.

In this work, we introduce a calculated distance *MaxDiff*, which is the maximum distance (among all intervals in the domain) between value P in the primary key table and value F in the foreign key tables. The idea is to check if a value being compared is contained within the range of $[P - MaxDiff, P + MaxDiff]$ (in case of equality), instead of comparing the value with P's bucket boundaries. Notice that it may result in a false positive, which should be eliminated in a post-processing. Figure 4 shows how to calculate *MaxDiff*.

3.3.2 Choosing the Random Distribution

The random function used to encrypt integers in the primary key table **(random_primary)** and a foreign key table **(random_foreign)** are different. Choosing random distribution for both functions is based on the following perspectives:

- **Security perspective:** Hide the interval boundaries in the primary key table because there is only one integer that represents each interval. This goal is achieved by maximizing the distance between primary key values that represent two sequential intervals. Based on that premise, the primary key should be as close as possible to the middle of the interval. This matches the normal distribution. Many records in the foreign key table represent the same interval. So to hide the interval boundaries we need to distribute the values over the whole interval so the attacker cannot differentiate between intervals. This condition matches the uniform distribution.
- **Performance perspective (reduce the false positives):** Reduce the overlap between intervals when connecting (joining) the primary key table with the foreign key table. This goal is achieved by minimizing the distance between the primary key and the foreign keys in the same interval (reducing *MaxDiff*). This is achieved by using normal distribution to choose a random number in both the primary key and foreign key tables. However, we already decided that uniform distribution is the best distribution to hide interval boundaries in the foreign key table, not normal distribution.

3.3.3 Further Optimization

Using normal distribution in the primary key table, however, presents the probability of an encrypted value appearing at the end of the interval I_v, and another encrypted value at the beginning of the interval I_{v+1}. This reduces the distance between the two encrypted values so the boundary becomes

```
Input: integer value (i)
Output: the encrypted value in primary key table E(i)

Begin
one_third = (B_{i+1} − B_i) / 3
P= random_primary(B_i + one_third , B_{i+1} - one_third)
Return E(i) = P
End
```
(a) Primary key encryption

```
Input: integer value (i)
Output: the encrypted value in foreign key table E(i)

Begin
P=Find_ primary(i)
        If P < 0 then //only in unrelated attribute
            P= encrypt_primary(i)
            insert_primary(P)
        End if
F=random_foreign(B_i, B_{i+1})
        //Update MaxDiff
        If IP-FI > MaxDiff then
                        MaxDiff = IP-FI
        End if
Return E(i) = F
End
```
(b) Foreign key encryption

Fig. 3 Encryption algorithms in MV-OPES.

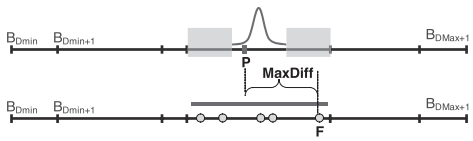

Fig. 4 Calculating the MaxDiff in MV-OPES.

easy to guess. To solve this problem, we set left and right margins in each interval. The encrypted value is then chosen to be between those margins. The margin sizes affect security and the percentage of false positives. We will investigate the optimal margin size in future work. In this paper, for simplicity, we assume equi-width partitioning. The interval is divided into three equal parts (left margin, middle part, right margin). The primary key is then chosen based on normal distribution within the middle part only. In this case, the distance between two primary keys that represent two sequential intervals will be *(right margin of the first interval + the left margin of the second interval)*. Figure 4 shows the random distributions used to choose a random number for both primary key P and foreign key F. The algorithms for encrypting integer values in a primary key table and foreign key table are represented in Fig. 3 (a, b).

3.4 Decryption Functions

Given the operator E, which encrypts a plaintext value to many ciphertext values, we define its inverse operator D, which decrypts the ciphertext value to its corresponding plaintext value. Simply, the decryption function D in MV-OPES searches for the interval where the encrypted value is located. Specifically, to decrypt an encrypted value C, the decryption function searches for the closer boundary B_p that is greater than C, then returns the plaintext value, which is the left boundary $p - 1$.

Given the boundaries $(B_{D_{min}}, \ldots, B_{D_{max+1}})$, which are stored either in memory with a small domain, or using a secondary storage-based indexing structure such as B+ tree for a large domain. The decryption function can be a sequential search (Fig. 5 (a)) or binary search (Fig. 5 (b)). The sequential search is used when decrypting ciphertext values in a sorted column; the binary search is used to decrypt ciphertext values in an unsorted column. The D operator may also be applied on query execution. The query execution may require that a super dataset be decrypted to perform advanced relational operators (e.g., grouping and aggregation); this is explained in Sect. 5.

```
Input: Encrypted value (Y),
Start point (i)
Output: Decrypted value D(Y)

Begin
while Y <= B_i
        i = i+1
    end while
Return D(Y) = i - 1
End
```
(a) Sequential search

```
Input: Encrypted value (Y),first, last
Output: Decrypted value D(Y)

Begin
while first <= last
    mid = (first + last) / 2
    if (Y >= B_mid and Y < B_{mid+1})
            Return D(Y) = mid
    else if (Y >= B_{mid-1} and Y< B_mid)
            Return D(Y) = mid − 1
    else if (Y > B_mid)
            first = mid + 1
    else
            last = mid - 1
    end if
end while
End
```
(b) Binary search

Fig. 5 Decryption algorithms in MV-OPES.

4. Condition Translations in Query Processing

This section explains how to translate a query condition C over a plaintext database in operations (such as selection and join) to corresponding conditions over encrypted database C^E. We consider query conditions characterized by the following grammar rules:

- Condition ← Attribute θ Value
- Condition ← Attribute θ Attribute
- Condition ← (Condition $\vee$ Condition) | (Condition $\wedge$ Condition) | ($\rightarrow$ Condition)

where θ is a binary operation in the set $\{=, <, \leq, >, \geq\}$.

The conditions are divided into two groups according to the type of result, that is, whether or not the result contains false positives. The first group consists of conditions that contain a binary operation between attribute and value (Attribute θ Value). The result based on those conditions contains neither false positives nor missed answer tuples. The second group consists of conditions that contain a binary operation between two attributes (Attribute θ Attribute) such as equi-join. The result based on those conditions contains false positive tuples. The discussion below uses a running example (Fig. 1) to illustrate the translation.

4.1 Conditions without False Positives

Attribute = Value: such condition arises in selection operations. The translation is defined as follows:

$$A = v \rightarrow A^E \; BETWEEN \; B_v \; and \; (B_{v+1} - 1)$$

As defined in Sect. 3.3, all encrypted values for value v are located in the interval (I_v), which is identified by boundaries B_v and B_{v+1}, $[B_v, B_{v+1})$. The BETWEEN condition allows the retrieval of values within a range of two values (inclusive). Since the right boundary B_{v+1} is not included in the interval (I_v), the second value in the BETWEEN condition will be the right boundary minus 1 $(B_{v+1} - 1)$. For instance, consider the employee table in Fig. 1. We have the condition over the plaintext $e\_id = 111$, so the translation condition is:

$$e\_id^E \; BETWEEN \; B_{111} \; and \; (B_{112} - 1)$$

Since $v = 111$, all encryption values for this value are located in the interval I_{111}.

Attribute < Value: such condition arises in selection operations. Since the MV-OPES is an order preserving encryption scheme ($v_i < v_j \rightarrow E^K(v_i) < E^K(v_j)$), the translation is as follows:

$$A < v \ \rightarrow \ A^E < B_v$$

The result contains all encrypted values that are less than the left boundary (B_v) of the interval (I_v). For example, in an employee table, the translation condition for the condition $C: (e\_id < 111)$ is: $C^E : e\_id^E < B_{111}$. Here, all encrypted values in the interval $[B_{D_{min}}, B_{111})$ will satisfy the translated condition.

Attribute ≤ Value: The difference between this condition and the previous one is that this condition includes all the encrypted values for v, in addition to those values that are less than the left boundary (B_v). The translation for the less than or equal condition is defined as follows:

$$A \leq v \ \rightarrow \ A^E < B_{v+1}$$

The result contains all encrypted values that are less than the left boundary of the next value (B_{v+1}). For instance, the translation condition for $C: (e\_id \leq 111)$ is: $C^E : e\_id^E < B_{112}$. Based on the translated condition, all returned tuples will be encrypted employee ids in the interval $[B_{D_{min}}, B_{112})$.

For conditions **Attribute > Value** and **Attribute ≥ Value**, translation is the same as the translation of $A < v$ and $A \leq v$, as described above but in the opposite direction.

4.2 Conditions with False Positives

The conditions containing a binary operation between two attributes (Attribute1 θ Attribute2) might arise in join or selection operations. With a join operation, the two attributes can be from two different tables or from two instances of the same table; in the selection operation; however, the two attributes should be from the same table. Because each integer is encrypted to many values, the translation for these conditions becomes a bit involved. Basically, we use parameter *MaxDiff* to make it possible to filter out unqualified values. However, the result may contain false positives by nature.

There are two constraints in performing these conditions. First, the two attributes should have the same domain. Second, the condition is performed between two primary keys or between a primary key and the related foreign key. Given a condition (P θ F), such that P and F have same domain, P is the primary key and F is a foreign or primary key. The translation for this condition is discussed next.

P = F: This condition is translated using the *MaxDiff* to:

$$F^E \ BETWEEN \ (P^E - MaxDiff) \ and \ (P^E + MaxDiff)$$

MaxDiff is used to ensure that all foreign key values located in same interval are connected to the related primary key. Intervals differ, so some foreign key values from neighbor intervals might connect to false primary keys (false positive).

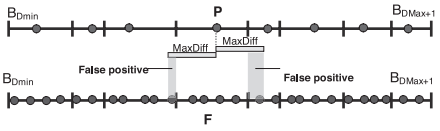

Fig. 6 Overlap in connecting primary key with foreign key values based on equality condition.

Figure 6 shows the process for connecting equal values between a primary key and foreign key. The total number of false positives (TFP) can be expressed as:

$$\sum_{i=D_{max}}^{D_{min}} |\{f \mid f \in F \wedge (P_i - MaxDiff \leq f < B_i \vee B_{i+1} \leq f < P_i + MaxDiff)\}|$$

P < F: The translation for this condition can be performed directly, just as with a plaintext database:

$$P^E \ < \ F^E$$

The translated condition, however, will contain false positives. These false positives result from foreign key values that are greater than the primary key but are located in the same interval. The total number of false positives returned can be written as:

$$\sum_{i=D_{max}}^{D_{min}} |\{f \mid f \in F \wedge (P_i \leq f < B_{i+1}\}| \ \approx \ 50\%$$

To reduce the number of false positives, we can use a distance range value from the primary key to the boundaries (left and right). Another solution is to derive benefit from the left and right margins on each interval. So, given the minimum margin (*MinMarg*) among all intervals in the domain, the translation for **P < F** is as follows:

$$(P^E + MinMarg) \ < \ F^E$$

Here the total number of false positives is reduced by *MinMarg* for each interval and can be written as:

$$\sum_{i=D_{max}}^{D_{min}} |\{f \mid f \in F \wedge (P_i + MinMarg \leq f < B_{i+1}\}|$$

P ≤ F: This condition should be translated in such a way that all foreign key values that are greater than or equal to the primary key can be connected to the related primary key. Using *MaxDiff*, the translation is as follows:

$$(P^E \ + MaxDiff) \ \leq \ F^E$$

The false positive results in this condition will be all foreign key values that are greater than the right boundary. The total number of false positives returned can be expressed as:

$$\sum_{i=D_{max}}^{D_{min}} |\{f \mid f \in F \wedge (B_{i+1} \leq f \leq P_i + MaxDiff\}|$$

The translation for **P > F** and **P ≥ F** is the same as the translation of $P < F$ and $P \leq F$ as described above, but in the

reverse direction.

(Condition1 ∨ Condition2), (Condition1 ∧ Condition2), (→ Condition): Two composite conditions are translated directly over the encrypted domain by translating each condition individually. The translation is given as follows:

$$C1 \vee C2 \rightarrow C1^E \vee C2^E,\ C1 \wedge C2 \rightarrow C1^E \wedge C2^E$$

The result based on two composite conditions might contain false positives when at least one condition is from *(Attribute θ Attribute)*. When both conditions are from *(Attribute θ Value)* the result will be exact. Translation of *(→ Condition)* depends on the condition type. When the condition is in the form of *(Attribute θ Value)*, the translation is straightforward: $\rightarrow\ C\ \rightarrow\ \rightarrow C^E$

The result based on this condition will be exact (without any false positives). However, when C is in the form of *(Attribute θ Attribute)*, this condition ($\rightarrow\ C$) cannot be translated directly because of the false positive result. This paper does not discuss this translation. Neither are conditions that involve more than one attribute and operator discussed.

5. Implementing Relational Operators over Encrypted Relations

This section describes the process of implementing relational operators (such as selection, projection, and sorting) in the proposed scheme. The relational operators are implemented, as much as possible, to be executed over the encrypted database. However, when a condition of the form *(Attribute θ Attribute)* is attached to the operator, the returned answers might contain false positives. These answers are then filtered (in client-side when client/server approach or DAS model used) after decryption to generate the exact result. Beyond that, some operators cannot be performed directly on the encrypted relations. When that happens, a post process operation is performed on the result after decryption. We attempt to minimize the amount of work done in post process operations. To explain implementation of the operators, we use the running example, which is the employee database shown in Fig. 1.

The Selection Operator (σ): consider a selection operator $\sigma_C(R)$ on a relation R, where C is a condition in the form *(Attribute θ Value)*. A straightforward implementation of such an operator in our scheme is to translate the condition C to condition C^E over encrypted relation R^E. Because the answers returned by this condition do not contain false positives, no extra operation is needed after decryption operator D. Specifically, the operator can be rewritten as follows:

$$\sigma_C(R) = D(\sigma^E_{C^E}(R^E))$$

In the above notation, we change the σ operator that is executed as a plaintext relation to σ^E to highlight the fact that the select operator is executed on an encrypted database. All operators that do not contain E are assumed to be executed over a plaintext relation. Let us take an example from the employee database. This example $\sigma_{gender=1 \wedge salary>200}(emp)$ selects all male employees with salary greater than 200. The above selection operator is translated into $D(\sigma^E_{C^E}(emp^E))$ where the condition C^E is:

$$C^E = (gender^E\ BETWEEN\ B_1\ and\ B_2 - 1)\ and\ (salary^E \geq B_{201})$$

The Join Operator (⋈): consider a join operation $R \overset{\bowtie}{C} T$. The join condition C is always in the form *(Attribute θ Attribute)*. Where θ can be either equality (in this case the join corresponds to a natural join or equijoin), or can be inequality (resulting in a theta join). The above join operation can be implemented as follows:

$$R \overset{\bowtie}{C} T = \sigma_C\left(D\left(R^E \overset{\bowtie^E}{C^E} T^E\right)\right)$$

As before, the E on the join operator emphasizes the fact that the join is to be executed over the encrypted database. The result from executing the join operator over an encrypted database will contain false positives. The result after decryption operator D, therefore, needs to be filtered using the σ operator with the same condition C. For instance, the join operation between *emp* table and *emp_proj* table ($emp \overset{\bowtie}{emp.e\_id=emp\_proj.e\_id} emp\_proj$) is translated to:

$$\sigma_{emp.e\_id=emp\_proj.e\_id}\left(D\left(emp^E \overset{\bowtie^E}{C^E} emp\_proj^E\right)\right)$$

Given *MaxDiff* for the *e_id* domain, the condition C^E on the encrypted database is:

$$emp\_proj.e\_id^E\ BETWEEN\ (emp.e\_id^E - MaxDiff)\ and\ (emp.e\_id^E + MaxDiff)$$

The Sorting Operator (τ): The sorting operator can be implemented directly on the encrypted database because MV-OPES preserves the order of the integer values. In MV-OPES, given encrypted values ($E_1 < E_2 < \cdots < E_{n-1} < E_n$), the plaintext values ($P_1 \leq P_2 \leq \cdots \leq P_{n-1} \leq P_n$).

Consider a sorting operation $\tau_A(R)$, on a relation R, where A refers to an attribute on which the sorting is performed. The implementation of the above sorting operator can be achieved directly as follows:

$$\tau_A(R) = D(\tau^E_{A^E}(R^E))$$

For instance, the sorting operation $\tau_{e\_id}(emp)$ is translated to $D(\tau^E_{e\_id^E}(emp^E))$. There are several contexts in which we want to sort the tuples of a relation by more than one of its attributes. This sorting operation is denoted by $\tau_L(R)$, where L is a list of some of R's attributes $(A_1, A_2, \cdots, A_n)$ on which the sorting is performed. Here, the sort operation on the encrypted database might yield an incorrect order when particular attributes with higher priority have tuples with the same value. For example, given a set of plaintext tuples $(1, 1)$ and $(1, 2)$, we would like to sort them according to

the two attributes. The encrypted value may be $(110, 100)$ and $(101, 201)$, when we apply an ordinary sort operator on these tuples; the resulting tuples order is incorrect. So when many attributes are used in the sorting operation, another implementation should be applied after the decryption operator. The implementation of such a sorting operation is as follows:

$$\tau_L(R) = \tau\prime_L\Big(D(\tau^E_{L^E}(R^E))\Big)$$

That is, we first sort on the encrypted attributes L^E. Then, after data decryption, another sorting operation ($\tau\prime$) applied on the partly sorted data of attributes L. Note that the amount of work done after decryption is less than the process of sorting on the encrypted data. This is so because sorting the results will be a simple operation as it will be performed only over records that have equal values. For example, the sorting operation $\tau_{e\_id,salary}(emp)$ can be implemented as follows:

$$\tau\prime_{e\_id,salary}\Big(D(\tau^E_{e\_id^E,salary^E}(emp^E))\Big)$$

The Projection Operator (π): Because each attribute A in a relation R is encrypted individually into a ciphertext attribute A^E of encrypted relation R^E, a projection operation π is implemented directly on R^E. The projection operation can be (with duplicate elimination) or (without duplicate elimination). Since the integer value is encrypted to many values in our scheme, the implementation for the two types of projection differ. Consider a projection operation without duplicate elimination $\pi_L(R)$, where L is a set of attributes of relation R. The operator can be implemented on the encrypted relation R^E as follows:

$$\pi_L(R) = D(\pi^E_{L^E}(R^E))$$

where L^E is a set of encrypted attributes. This operation is used also for a projection with duplicate elimination on a unique set of records such as primary key attributes. For instance, the projection operation $\pi_{e\_id}(emp)$ is translated to $D(\pi^E_{e\_id^E}(emp^E))$.

The projection operation with duplicate elimination can be implemented as follows:

$$\pi_L(R) = \delta\Big(D(\pi^E_{L^E}(R^E))\Big)$$

That is, we first perform a projection operation $\pi^E_{L^E}$ on the encrypted relation R^E. We then apply duplicate elimination operation δ on the decrypted data.

The Grouping and Aggregation Operator (γ): Consider a grouping operation $\gamma_{F,L}(T)$ on a relation T, where F is a foreign key attribute. As described above, MV-OPES encrypts integer values to many different values. To perform a grouping operation on a foreign key attribute, therefore, we need to first apply a join operation between the primary key and the foreign key. This will ensure that all records for a particular plaintext value have the same encrypted value. We can then apply a grouping operation on the encrypted relation. Given P as primary key for F in relation R, the above grouping operation can be implemented as follows:

$$\gamma_{F,L}(T) = \pi_{P,L}\left(\sigma_{JC}\left(D\left(\gamma^E\left(R^E \underset{JC^E}{\bowtie^E} T^E\right)\right)\right)\right)$$

where JC^E is the condition for the joint operation $\bowtie^E$ on the encrypted relations. Because this join operation will generate false positives, a selection operation σ_{JC} on the decrypted data is applied to remove these false positives. The projection operation with duplicate elimination $\pi_{P,L}$ on the decrypted data is performed after the selection operation to project the primary key attribute instead of the foreign key.

When applying the grouping operation on many foreign key attributes in a relation, the translation will be more complicated and expensive because many join operations need to be performed. Here the grouping might be performed after data decryption to avoid a join operation on the encrypted database. Also, aggregation operators are applied on decrypted data because our scheme does not support the aggregation operation over an encrypted relation. To minimize the work after data decryption, we can apply a sorting operation τ on the encrypted relation, so the grouping operation will be faster. Efficient implementation of the grouping operation is as follows:

$$\gamma_L(R) = \gamma_L\Big(D(\tau^E_{LG^E}(R^E))\Big)$$

where L refers to a list of attributes on which the grouping and aggregation operations are performed, and LG^E is a list of encrypted attributes on which the grouping operation is applied on the encrypted relation. For instance, if we want to find the number of male and female employees in the *emp* table, the translation is as follows:

$$\gamma_{gender,COUNT(e\_id)\to M}(R) = \gamma_{gender,COUNT(e\_id)\to M}\Big(D(\tau^E_{gender^E}(emp^E))\Big)$$

That is, we first sort on the $gender^E$ attribute on the encrypted relation emp^E. After data decryption, we perform the grouping operator. This step can be done efficiently, since all records have already been sorted on the encrypted relation based on the *gender* value. Finally, we perform the aggregation $COUNT(e\_id) \to M$ to count the number of employees for each *gender* value.

The Duplicate Elimination Operator (δ): The implementation of a duplicate elimination operator δ on the encrypted relation is similar to the grouping operator γ. That is, we first need to perform a join operation and grouping operation on the encrypted relations. After the results are decrypted, we apply selection to remove false positives generated by the join operation. Finally, we apply a projection operation with duplicate elimination also on the decrypted data. As discussed above, this implementation might be expensive, especially with many attributes. Thus, the efficient way to perform a duplicate elimination operation is to sort on the encrypted relation. Then after the results are decrypted, we apply the duplicate elimination operation:

$$\delta(R) = \delta\Big(D(\tau^E_{L^E}(R^E))\Big)$$

where L^E is a list of attributes in the encrypted relation R^E.

The amount of work to perform the duplicate elimination operation δ after decryption operator D is small because the data is already sorted. For example, the operation $\delta(emp)$ is translated to:

$$\delta\Big(D(\tau^E_{e\_id^E, salary^E, gender^E}(emp^E))\Big)$$

The Union Operator ($\cup$): consider two relations R and T that have schemas with identical sets of attributes, and the domains for each attribute are the same in R and T. The union operation can be based on bag (does not eliminate duplicates) or set (with duplicate elimination). The union operator based on bag can be implemented directly on the encrypted relation as follows:

$$R \cup T = D(R^E \cup^E T^E)$$

The implementation of the union operator based on set is performed as follows:

$$R \cup T = \delta\Big(D(\tau^E_L(R^E \cup^E T^E))\Big)$$

where L^E refers to the encrypted attributes in relation R, and $\cup^E$ is the union operation on the encrypted database. The sorting operation τ^E_L on the encrypted result will help to speed the process of eliminating the duplication δ on the decrypted data. The decryption operator D is performed with no trouble because we assume that the domains for each attribute are the same in both relations. When a union operator is performed between different domains, special care should be taken to apply the decryption operator on the result. Implementing a union operator on different attribute domains is not discussed in this paper.

The Set Difference Operator ($-$): Performing the set difference operation $R-T$ on the encrypted relations is complicated and expensive. This is especially true when the difference operation is applied on a relation that holds many foreign key attributes. That is because many equi-join operations and outer join operations need to be performed on the encrypted relations. In addition, for each join operation, we need to apply a selection operation on the decrypted data to remove false positives generated by the join operator. The strategy to perform set difference, therefore, is to decrypt data for both relations and then apply the set difference operator on the decrypted data. We can get benefits from the order preserving concept in our scheme to reduce the work needed to perform set difference after data decryption. That is, we first sort on both encrypted relations. Then, after data decryption, we perform the set difference. The implementation for set difference can be expressed as follows:

$$R - T = D(\tau^E_{LR^E}(R^E)) - D(\tau^E_{LT^E}(T^E))$$

where LR^E refers to a list of attributes in the encrypted relation R^E, and LT^E is a list of attributes in T^E. For example, to find all *eid* for employees who do not work on any project, the set difference operator is:

$$\pi_{e\_id}(emp) - \pi_{e\_id}(emp\_proj) = D\Big(\tau^E_{e\_id^E}\Big(\pi^E_{e\_id^E}(emp^E)\Big)\Big) - D\Big(\tau^E_{e\_id^E}\Big(\pi^E_{e\_id^E}(emp\_proj^E)\Big)\Big)$$

Query Splitting: In the client/server or DAS model, we split the computation of a query Q across the server and the client. The client will use the implementation of the relational operators to send part of the query Q_s to the server to be executed on the encrypted database. The second part, which is client query part Q_c, is performed on the decrypted data. Query splitting is as follows:

$$\underbrace{op\ (R)}_{Q} = \underbrace{op^c\ D}_{Q_c}\ \underbrace{(op^E\ (R^E))}_{Q_s}$$

where op^c refers to operations performed on the client side, and op^E is operations performed on encrypted relations R^E on the server side.

6. Security Analysis

The security of our scheme derives from the security of pseudorandom permutation function and block cipher used to generate boundaries. First, we assume that the key used in the encryption process is secure, and that the pseudorandom number generator R_i used to generate boundaries is also secure. In our scheme, we have different interval sizes based on R_i, an encrypted interval size $Enc^K(IS)$ and the difference percentage DP. An adversary with no information about the plaintext domain, therefore, cannot deduce any information about the encrypted domain. The success probability of an adversary A in decryption ciphertext values for a particular domain can be expressed as follows:

$$Adv_A = Pr[R] \cdot Pr[EIS] \cdot Pr[DP] \cdot Pr[IP]$$

where $Pr[R]$ is the probability breaking the pseudorandom function, $Pr[EIS]$, $Pr[DP]$, $Pr[IP]$ are the probability to get the encrypted interval size, the difference percentage and the initial point. The value Adv_A is sufficiently small. Thus, our encryption scheme is secure enough in many practical situations even if an adversary has access to all ciphertext values. Notice that the security analysis discussion here is slightly different from that in the cryptography field, that is, the probability of information leakage is not 0 in all cases. In some cases, the adversary can infer a very small amount of information about the ciphertext values when she has deep knowledge about the plaintext domain. However, we can avoid even those small vulnerabilities with some small improvements. The following is the security analysis of our scheme for different attack scenarios.

In our scheme, it is clear that the adversary cannot infer information by applying a traditional join operation on the encrypted relation. Here the number of possible mappings between m distinct ciphertext values in the primary key table and n distinct ciphertext values in the foreign key table (in an order-preserving way) is $\binom{n-1}{m-1}^{\dagger}$, where $n > m$. Out

[†]The possible combinations between ciphertext values and plaintext values in the one to many encryption schemes in an order-preserving way is discussed in [6].

of those possible mappings, only one mapping combination correctly determines the joining between the ciphertext values. The adversary can join the primary key with the closest encrypted values in the foreign key, but not all values. Using a shifting distance (right or left) in either the primary or foreign key, the adversary will not even be able to make a join based on the closest distance. That is not possible because the primary and foreign key values will be in completely different ranges. Using shifting distance, the number of mappings is $(s\binom{n-1}{m-1})$, where s is the number of tries till get the shifting distance. A change might occur in the implementation of MV-OPES using shifting distance, so we merely consider this distance in the condition translation process.

Now, we assume an adversary A knows the plaintext domain and has access to the ciphertext values. The adversary tries to map p distinct plaintext values and c distinct ciphertext values in an order-preserving way. The number of possible mappings in this case is $\binom{c-1}{p-1}$, where $c > p$. The adversary can estimate at most the first and last intervals in the encrypted scale. That is possible because the intervals are ordered the same as the plaintext values. We can improve our scheme to avoid this attack by generating boundaries in a circle mode. That means we generate boundaries normally until reaching a particular point, then we continue generating the rest of the boundaries starting from a point that is less than the boundary for the first integer in the domain $B_{D_{min}}$. Here we will have order preserved in two super intervals. The number of mappings in this case is $(c\binom{c-1}{p-1})$. The only change needed to apply this technique is translation for the inequality operator, which will be in two directions instead of one.

We now formally analyze the security of our scheme in the chosen plaintext attack. The idea is to give the adversary advantage and freedom to encrypt a set of plaintext to discover the starting and ending points for the integer i in the encrypted scale. To achieve this, the adversary needs to get at least four encrypted integers (X, Y, Z, U), such that X is a ciphertext for $i-1$, Y *and* Z are ciphertext for i, U is a ciphertext for $i+1$, where $Y = X+1$, and $U = Z+1$. Given the interval sizes for $i-1, i, i+1$ as S_{i-1}, S_i, S_{i+1} the probability to get the four integers is: $\frac{1}{S_{i-1}\ 2S_i\ S_{i+1}}$. This means that security here depends on the interval size. In a small domain, the scheme will be more secure than a large domain where we can use larger encrypted intervals.

7. Experiments

This section evaluates the performance of our encryption scheme. We have conducted many experiments to examine the validity and effectiveness of the architecture proposed in this paper.

7.1 Experimental Setup

The experiments were conducted by implementing MV-OPES on MS SQL Server 2008. The algorithms were implemented in VB.NET as a client side application. The experiments were run using version 3.0 of the Microsoft.Net framework and on a Microsoft XP workstation with a 2.6 GHz Intel Core 2 processor and 3 GB of memory. The results sketched in this section are the average for at least 10 executions.

7.2 Performance for Generating Boundaries

The first set of evaluations was performed on different domains to examine the time needed to generate boundaries. The graph in Fig. 7 shows the execution time for generating boundaries using different *initial* points. The results show that we have better performance when the initial point is chosen in the middle of the domain. That is because we have two directions to compute the boundaries compared with one increasing function when we use D_{min} as initial point.

7.3 Performance for Encryption and Equijoin

The second set of evaluations studied the encryption performance and equijoin operation in our scheme using different domains and various difference percentages (DP). Also, we compare the performance of our scheme with a database encrypted using AES. Two tables were used to perform this evaluation. The first table is the primary key table, which contains all integers in the domain. The second table is the foreign key table, which holds 100,000 records picked randomly from a uniform distribution between D_{min} and D_{max}. Figure 8 shows the times for encryption and inserting values in the primary key and foreign key tables for different domains. The results show that AES takes the longest time to insert tuple in both tables since the encryption time is much more than in MV-OPES. The small difference in time shown in the figure between plaintext and our scheme is the cost of encryption. The figure shows that this overhead is negligible.

In the equijoin operation, we studied the percentage of false positives returned by performing a join operation over

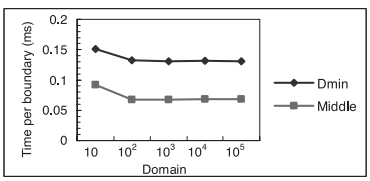

Fig. 7 Time per boundary (in ms) required to generate boundaries.

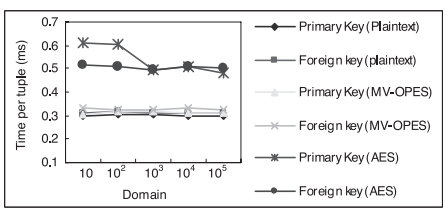

Fig. 8 Time per tuple (in ms) required to insert tuples.

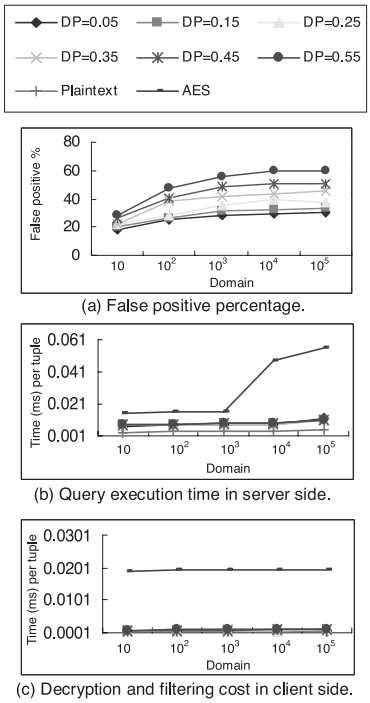

Fig. 9 Equijoin cost.

encrypted relations. Also, we studied the overhead on both the server and client sides. The percentage of false positives shown in Fig. 9 (a) increases with the domain size and *DP*. That results due to the increase in the overlap between intervals in the encrypted scale when performing a join operation based on *MaxDiff*. From Fig. 9 (b), we can easily see that the time required to perform a join operation on the server side in our scheme increases according to the size of domain and takes approximately the same shape as the join operation on the plaintext database. While the cost of join operation using AES is much more than our scheme. This is especially when using large domains ($> 10^3$) since the index is essentially unusable for many operations (including join) which turn into full table scans [32].

Figure 9 (c) shows the client side performance to decrypt and filter the result returned by performing a join operation on the server side. The figure shows that our scheme has only small overhead on the client side. We also observe that the time slightly increases as the domain and *DP* increase, because of increased false positives. On the other hand, we can see the performance degradation when using AES compared with our scheme.

7.4 Selection and Grouping Operations

In the third set of experiments, we studied queries that include selection, grouping and aggregation operations. Experiments are based on the *emp* table (100,000 records) shown in Fig. 1 and queries shown in Fig. 10. Figure 11 shows the plaintext, client-side, server-side, and total query execution times for both queries. The figure illustrates that

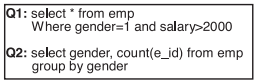

Fig. 10 Queries used for the third set of experiments.

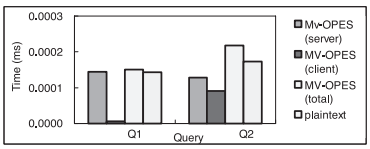

Fig. 11 Selection, grouping and aggregation cost.

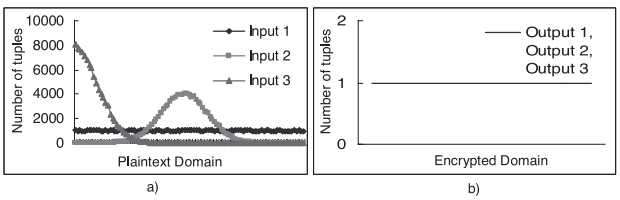

Fig. 12 Input (a) and output (b) distributions in MV-OPES.

$Q1$ response time is approximately the same in both approaches. The reason is due to the small cost of decryption performed at the client site. Beyond that, this selection query can be fully performed on the encrypted relation and the result is exact. On the other hand, total query execution time for $Q2$ in MV-OPES is greater than the execution time in the plaintext. That is because the grouping and aggregation are performed at the client site after the decryption process. However, query execution time on the client side is less than the query execution time on the server side because the data is already sorted on the encrypted relation.

7.5 Distribution of Encrypted Domains

In this experiment, we studied the distribution of plaintext and encrypted values. Here, we generated 100,000 records using $[1, 100]$ as an input domain, $DP = 0.05\%$, and $IS = 1000$. We tested whether it is possible to distinguish or match between the output distributions of MV-OPES and different input distributions. Figure 12 shows the result of running MV-OPES with different input distributions (Uniform, Normal, and Zipf). We can see that the encrypted values always followed a uniform distribution regardless the distribution of the plaintext domain. Notice that the frequency for all encrypted values is 1. These results reflect the robustness of our scheme against attacks that try to match between input and output distributions.

8. Conclusion and Future Work

Encryption can be used to provide confidentiality and privacy for sensitive databases, which are important issues, especially in the DAS model. Unfortunately, traditional attribute level encryption is vulnerable to statistical attacks because each value is encrypted to another fixed value. We

propose a novel encryption scheme (MV-OPES) that is robust against statistical attack and estimation of the true value because it allows one integer to be encrypted to many different values using the same encryption key. It also preserves the order of the integer values to allow any comparison operation to be directly applied to the encrypted data. We have developed techniques so that most processes in executing SQL queries can be done on encrypted databases. In some cases, a small amount of work to filter false positives or perform relational operations is needed on the decrypted data. Experiments on MV-OPES showed that security for sensitive data can be achieved with reasonable overhead, confirming the feasibility of the scheme. In the future, we will study the encryption of non-integer data such as strings. We also plan to investigate security improvement issues, such as circle encrypted domain and shifting distance.

Acknowledgements

This research has been supported in part by the Grant-in-Aid for Scientific Research from MEXT (#21013004) and Grant-in-Aid for Young Scientists (B) (#21700093) by JSPS.

References

[1] H. Hacigumus, S. Mehrotra, and B. Iyer, "Providing database as a service," Proc. 18th International Conference on Data Engineering (ICDE '02), pp.29–38, IEEE Computer Society, Washington, DC, USA, 2002.

[2] B.R. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu, "A framework for efficient storage security in RDBMS," Proc. International Conference on Extending Database Technology (EDBT '04), pp.147–164, 2004.

[3] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," Proc. 30th International Conference on Very large databases (VLDB '04), pp.720–731, VLDB Endowment, 2004.

[4] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," Proc. 2002 ACM SIGMOD International Conference on Management of data (SIGMOD '02), pp.216–227, ACM, New York, NY, USA, 2002.

[5] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes, "A structure preserving database encryption scheme," VLDB 2004 Workshop, Proc. Secure Data Management SDM 2004, pp.28–40, Springer, 2004.

[6] H. Wang and L.V.S. Lakshmanan, "Efficient secure query evaluation over encrypted XML databases," Proc. 32nd International Conference on Very Large Databases (VLDB '06), pp.127–138, VLDB Endowment, 2006.

[7] E. William, M. Carl, S. John, and T. Walter, "Message verification and transmission error detection by block chaining," 4074066, Feb. 1978.

[8] S.S. Chung and G. Ozsoyoglu, "Anti-tamper databases: Processing aggregate queries over encrypted databases," Proc. 22nd International Conference on Data Engineering Workshops (ICDEW '06), p.98, IEEE Computer Society, Washington, DC, USA, 2006.

[9] G. Ozsoyoglu, D.A. Singer, and S.S. Chung, "Anti-tamper databases: Querying encrypted databases," 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, pp.4–6, Estes Park, 2003.

[10] Z. Yang, S. Zhong, and R.N. Wright, "Privacy-preserving queries on encrypted data," Proc. 11th European Symposium on Research in Computer Security (ESORICS), pp.479–495, Springer, 2006.

[11] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," Proc. 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04), pp.563–574, ACM, New York, NY, USA, 2004.

[12] D.X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," Proc. 2000 IEEE Symposium on Security and Privacy (SP'00), p.44, IEEE Computer Society, Washington, DC, USA, 2000.

[13] Y.C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," Proc. Third International Conference Applied Cryptography and Network Security (ACNS 2005), Lect. Notes Comput. Sci., vol.3531, pp.442–455, 2005.

[14] D. Boneh, G.D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," Proc. International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2004), pp.506–522, Springer, 2004.

[15] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," Proc. 22nd annual IFIP WG 11.3 working conference on Data and Applications Security, pp.127–143, Springer-Verlag, Berlin, Heidelberg, 2008.

[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," Proc. 41st Annual ACM Symposium on Theory of Computing (STOC '09), pp.169–178, ACM, New York, NY, USA, 2009.

[17] T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," Proc. 33rd International Conference on Very Large Databases (VLDB '07), pp.519–530, VLDB Endowment, 2007.

[18] E. Mykletun and G. Tsudik, "Aggregation queries in the database-as-a-service model," Proc. 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec), pp.89–103, Springer, 2006.

[19] H. Hacigümüs, B.R. Iyer, and S. Mehrotra, "Efficient execution of aggregation queries over encrypted relational databases," Proc. 9th International Conference on Database Systems for Advances Applications (DASFAA), pp.125–136, Springer, 2004.

[20] L. Bouganim and P. Pucheral, "Chip-secured data access: Confidential data on untrusted servers," Proc. 28th International Conference on Very Large Databases (VLDB '02), pp.131–142, 2002.

[21] L. Bouganim, F.D. Ngoc, P. Pucheral, and L. Wu, "Chip-secured data access: Reconciling access rights with data encryption," Proc. 29th International Conference on Very Large Databases (VLDB '03), pp.1133–1136, 2003.

[22] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu, "Two can keep a secret: A distributed architecture for secure database services," CIDR, pp.186–199, 2005.

[23] F. Emekci, D. Agrawal, A.E. Abbadi, and A. Gulbeden, "Privacy preserving query processing using third parties," Proc. 22nd International Conference on Data Engineering (ICDE '06), p.27, IEEE Computer Society, Washington, DC, USA, 2006.

[24] D. Agrawal, A.E. Abbadi, F. Emekci, and A. Metwally, "Database management as a service: Challenges and opportunities," Proc. 2009 IEEE International Conference on Data Engineering (ICDE '09), pp.1709–1716, IEEE Computer Society, Washington, DC, USA, 2009.

[25] H. Kadhem, T. Amagasa, and H. Kitagawa, "A novel framework for database security based on mixed cryptography," International Conference on Internet and Web Applications and Services, pp.163–170, 2009.

[26] DES, "Data encryption standard," Federal Information Processing Standards Publication, vol.FIPS PUB 46, 1977.

[27] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," Fast Software Encryption, Cambridge Security Workshop, pp.191–204, Springer-Verlag, London, UK, 1994.

[28] AES, "Advanced encryption standard," National Institute of Science and Technology, vol.FIPS 197, 2001.

[29] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," ACM Commun.,

vol.21, no.2, pp.120–126, 1978.
[30] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," SIAM J. Comput., vol.13, no.4, pp.850–864, 1984.
[31] A.J. Menezes, S.A. Vanstone, and P.C.V. Oorschot, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, USA, 1996.
[32] S. Hsueh, "Database encryption in SQL server 2008 enterprise edition," Microsoft White Papers, vol.SQL Server 2008, Feb. 2008.

Hasan Kadhem received the B.Sc. degree in computer science from University of Qatar in 2003. He received the M.Sc. degrees in computer security from the New York Institute of Technology (NYIT), in 2007. Since April 2008, he has been a Ph.D. student at Department of Computer Science, and at Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests include database security, database encryption, web application security, and privacy preserving.

Toshiyuki Amagasa received B.E., M.E., and Ph.D. degrees from Department of Computer Science, Gunma University in 1994, 1996, and 1999, respectively. He had been an assistant professor at Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) from April 1999 to March 2005. Since April 2005, he has been an assistant professor at Center for Computational Sciences, and Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests include database systems, XML databases, and scientific databases. He is a member of IPSJ, DBSJ, ACM, and IEEE CS.

Hiroyuki Kitagawa received the B.Sc. degree in physics and the M.Sc. and Dr.Sc. degrees in computer science, all from the University of Tokyo, in 1978, 1980, and 1987, respectively. He is currently a full professor at Graduate School of Systems and Information Engineering and at Center for Computational Sciences, University of Tsukuba. His research interests include integration of information sources, data mining, stream-based ubiquitous data management, web data management, XML, and scientific databases. He has published more than 150 papers in refereed journals and international conference proceedings. He is a fellow of the IPSJ (Information Processing Society of Japan), a Trustee of the DBSJ (The Database Society of Japan), and a member of the ACM, the IEEE Computer Society, and the JSSST. He served as Chairperson of the IEICE Special Interest Group on Data Engineering from 1999 to 2001, Chairperson of ACM SIGMOD Japan Chapter from 2003 to 2007, and Vice Chairperson of DBSJ from 2005 to 2007.