A SECURE AND EFFICIENT ORDER PRESERVING ENCRYPTION SCHEME FOR RELATIONAL DATABASES

Hasan Kadhem
Department of Computer Science,Graduate School of Systems and Information Engineering
University of Tsukuba, Tsukuba, Japan

Toshiyuki Amagasa, Hiroyuki Kitagawa
Department of Computer Science,Graduate School of Systems and Information Engineering
Center for Computational Sciences, University of Tsukuba, Tsukuba, Japan

Keywords: Order preserving encryption, Known plaintext attack, Statistical attack.

Abstract: Encryption is a well-studied technique for protecting the confidentiality of sensitive data. However, encrypting relational databases affects the performance during query processing. Preserving the order of the encrypted values is a useful technique to perform queries over the encrypted database with a reasonable overhead. Unfortunately, the existing order preserving encryption schemes are not secure against known plaintext attacks and statistical attacks. In those attacks, it is assumed that the attacker has prior knowledge about plaintext values or statistical information on the plaintext domain.
This paper presents a novel database encryption scheme called MV-POPES (Multivalued - Partial Order Preserving Encryption Scheme), which allows privacy-preserving queries over encrypted databases with an improved security level. Our idea is to divide the plaintext domain into many partitions and randomize them in the encrypted domain. Then, one integer value is encrypted to different multiple values to prevent statistical attacks. At the same time, MV-POPES preserves the order of the integer values within the partitions to allow comparison operations to be directly applied on encrypted data. Our scheme is robust against known plaintext attacks and statistical attacks. MV-POPES experiments show that security for sensitive data can be achieved with reasonable overhead, establishing the practicability of the scheme.

1 INTRODUCTION

Encryption is a well-studied technique to protect sensitive data so that when a database is compromised by an intruder, data remains protected even when a database is successfully attacked or stolen. Recognizing the importance of encryption techniques, several database vendors offer an integrated solution that provides standard encryption functionality in their products. Even though encrypting the database provides important protection, performing queries over the encrypted databases becomes more challenging. Encrypting database using a standard block cipher such as AES, and DES causes undesirable performance degradation during query processing. The reason is that the standard encryption techniques do not preserve the order of integers, so an entire table scan would be needed to perform queries such as range query. Thus the query execution can become unacceptably slow.

Preserving the order of the encrypted values is a useful technique to perform quires over the encrypted database with a reasonable overhead. For instance, given three integers $\{a,b,c\}$, such that $(a<b<c)$, then the encrypted values are $(E^K(a)<E^K(b)<E^K(c))$. Here, $E^K(v)$ denotes ciphertext value of v with encryption key K. The order preserving encryption scheme OPES proposed firstly by (Agrawal et al., 2004). 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. The strength and novelty of OPES is that comparison operations, equality and range queries as well as aggregation queries involving MIN, MAX and COUNT can be evaluated directly on encrypted data, with-

Kadhem H., Amagasa T. and Kitagawa H..
A SECURE AND EFFICIENT ORDER PRESERVING ENCRYPTION SCHEME FOR RELATIONAL DATABASES.
DOI: 10.5220/0003095700250035
In *Proceedings of the International Conference on Knowledge Management and Information Sharing* (KMIS-2010), pages 25-35
ISBN: 978-989-8425-30-0

out decryption. Another encryption scheme proposed by (Chung and Ozsoyoglu, 2006; Ozsoyoglu et al., 2003), where 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.

Unfortunately, the previous order preserving encryption (OPE) schemes are not secure against known plaintext attacks and statistical attacks. In those attacks, it is assumed that the attacker has a prior knowledge about plaintext values or statistical information on plaintext domain. The authors of the previous OPE schemes either ignore those type of attacks or assumed that the attacker does not have any information about the plaintext domain. In reality, the attacker in many cases may have general or advanced information about plaintext domain. Here, the attacker who has access to the encrypted values and has knowledge about the plaintext can map both the plaintext and the encrypted values and make use of them to obtain the key. This is because the OPE schemes preserve the order of all integers in the domain, so the order of encrypted values is exactly the order of plaintext values (we called those schemes as full OPE).

This paper presents a new database encryption scheme called MV-POPES (Multivalued - Partial Order Preserving Encryption Scheme), which divide the plaintext domain into many partitions and randomize them in the encrypted domain. It allows one integer to be encrypted to many values using the same encryption key while preserving the order of the integer values within the partitions. Here, still we can get benefit from the partial order preserving in the encrypted data to perform queries directly at the server without decrypting data. At the same time, it prevents attackers from inferring individual information from the encrypted database even if they have statistical and special knowledge about the plaintext database. The reason is that the encrypted values are totally in different order compared with the plaintext values. The results from an implementation of MV-POPES show that security for sensitive data can be achieved with reasonable overhead, establishing the practicability of the scheme.

Our threat model is same as the one in the previous OPE schemes (Agrawal et al., 2004) but with more conservative approach. We assume that the attacker not only has access to the encrypted database files but also has special knowledge about the plaintext database. However, the attacker does not have access to the logs and the memory of the database software. Figure 1 shows the system model where the client sends a rewritten query using the metadata

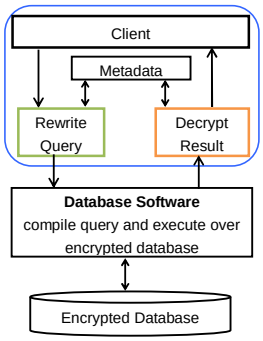


Figure 1: System model.

to the database software. Then the query is executed on the encrypted database and return the results to the client who decrypt them using same metadata. The metadata, query rewriting, and result decryption could be on the database software or on client side. In this paper we consider them to be on client side. We make the following contributions:

- We analyze the security of previous full OPE schemes based on an attack model we define, consisting of known plaintext attack and statistical attack (Section 2).
- We propose our novel database encryption scheme (MV-POPES) and the metadata used to support encryption, decryption and efficient query processing (Section 3).
- We explain how to translate a query condition over a plaintext to corresponding conditions over encrypted database (Section 4).
- We show how relational operators can be implemented efficiently over encrypted relations (Section 5).
- We analyze the security of our scheme based on definitions and theorems we define in Section 2 (Section 6).
- We complement our analytical results with experiments on different domains that measure the effect of our encryption scheme on query processing efficiency, as well as the client post processing cost (Section 7).

The related work is discussed in Section 8. We conclude with a summary and directions for future work in Section 9.

2 SECURITY ISSUES

In this section we analyze the security of the full OPE schemes based on two concepts, the mapping probability and similarity degree of statistics.

2.1 Mapping Probability

The mapping probability is related to the known plaintext attack. In this attack, it is assumed that the attacker has both the plaintext and its encrypted version (ciphertext) and makes use of them to obtain the key used in the encryption process.

Definition 1. ***(Mapping Probability (μ))*** *Let ε be an encryption scheme. Let ρ be the set of plaintext values and ς be the corresponding set of ciphertext values using ε. Let η be the number of possible mappings between ρ and ς. Then, the mapping probability μ is the probability to pickup the correct mapping among η, so $\mu = 1/\eta$.*

Theorem 1. ***(Security against Known Plaintext Attack)*** *Let ε be an encryption scheme. Let μ be the mapping probability between the set of plaintext values ρ and the corresponding set of ciphertext values ς using ε. Then, ε is secure against known plaintext attack when μ is significantly small.*

Note that the μ will be significantly small if η is exponentially large. This theorem shows that by having large η, it will be almost impossible for the attacker to infer information about encryption scheme and the key by the advantage of knowing both plaintext and cipher text values.

Using a full OPE scheme, we can clearly see that there is only one possible mappings ($\eta = 1$) between ρ and ς, then $\mu = 1$. The reason is that both the plaintext and ciphertext values have exactly the same order. So any full OPE scheme is not secure against known plaintext attack.

2.2 Similarity Degree of Statistics

The Similarity Degree of Statistics is related to the statistical attack. In this attack, the attacker tries to find a match between the ciphertext values and plaintext values based on some statistical information on plaintext and make use of them to obtain the key.

Definition 2. ***(Similarity Degree of Statistics (Δ))*** *Let ε be an encryption scheme. Let Σ^{ρ} be the statistics on plaintext values and Σ^{ς} be the statistics on the corresponding ciphertext values using ε. Then, Δ is the percentage of similarity between both Σ^{ρ} and Σ^{ς}.*

The statistics here involving frequencies of values, and aggregation functions such us MIN, MAX, SUM, and COUNT.

Theorem 2. ***(Security against Statistical Attack)*** *Let ε be an encryption scheme. Let Δ be the Similarity Degree of Statistics between the statistics on plaintext Σ^{ρ} and the statistics on the corresponding ciphertext Σ^{ς} using ε. Then, ε is secure against statistical attack when Δ is significantly small.*

Using a full OPE scheme, the aggregation function such as MIN, MAX, and COUNT are exactly the same on both plaintext and ciphertext values. In addition, many previous full OPE schemes encrypt plaintext value to another fixed value, so the frequencies of values are same before and after encryption. The result will be high similarity degree of statistics Δ which means that the full OPE scheme is not secure against statistical attack.

3 RELATIONAL ENCRYPTION

This section describes our encryption scheme in detail.

3.1 An Overview of MV-POPES

When encrypting plaintext values in a column having values in the range $[D_{min},D_{max}]$, first, we divide the domain into n partitions and assign for each partition a random number from 1 to n. This number will be the order of partitions in the encrypted domain. We change the order of partitions to hide the original order of plaintext values. Then, we generate boundaries for all integers in all partitions using an order preserving function. We preserve the order within the partition to be able to evaluate queries efficiently on encrypted database. 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, so one plaintext value is encrypted to many different values. This will change the frequencies of the plaintext values to prevent the encrypted database against statistical attack.

3.2 Partitioning and Metadata

Here, we explain the partitioning function for each attribute's domain and what is stored in the metadata for each domain. We first divide the plaintext domain of values $[D_{min},D_{max}]$ into partitions $\{p_1,...,p_n\}$, such that these partitions cover the whole domain and there is no overlap between them. Then, we assign for each partition a unique random number in the range of $[1,n]$. This number is the new order of partitions in the encrypted domain.

As an example, Figure 2 shows the partitions metadata for the domain [1,100]. The domain is divided into 5 partitions: [1,20],[21,40],[41,60],[61,80],[81,100]. (F), (L)

PID	EPID	PREV	F	L	NEXT
1	3	80	1	20	81
2	1	-	21	40	61
3	5	100	41	60	101
4	2	40	61	80	1
5	4	20	81	100	41

Figure 2: Partitioning metadata.

are the first and last number in the partition. The partition identifier (PID) represents the original order of partitions in plaintext domain. The encrypted partition identifier $(EPID)$ represents the order of partition in the encrypted domain. $(PREV)$ and $(NEXT)$ are the previous and the next number in the encrypted domain for a partition. For instance, the partition $[61,80]$ where PID=4, and $EPID$=2, the $(PREV)$ will be the last number for the partition with $EPID$=1 (which is 40), and the $(NEXT)$ will be the first number for the partition with $EPID$=3 (which is 1).

The metadata is used to generate bucket boundaries, encryption/decryption and query translation. It contains all the secret information about partitions so must be a well-kept secret, as well as the key.

3.3 Generation of Bucket Boundaries

Any order preserving function with suitable security properties can be used to generate the bucket boundaries for all integers in the plaintext domain based on the encrypted order of partitions. Here, we propose an order preserving function based on secret variables and a sequence of random numbers.

We are given a domain $[D_{min},D_{max}]$, with $(D_{max}-D_{min}+1)$ integers: $\{D_{min},D_{min+1},...,D_{max}\}$. 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 (DES, 1977), TDES, Blowfish (Schneier, 1994), AES (AES, 2001), RSA (Rivest et al., 1978), or a hashing function can be used to encrypt the value.

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:

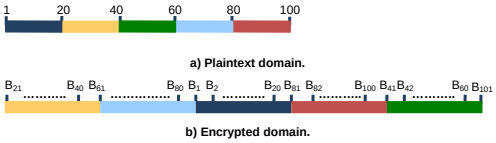


Figure 3: Plaintext domain and the boundaries in the encrypted domain.

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

where R_i is a sequence of random numbers in the range $[-1,1]$. There are many pseudorandom number generators with useful security properties (Blum and Micali, 1984; Menezes et al., 1996). The DP used in the formula to control the differences between intervals size that will be in the range $[-Enc^K(IS)*DP,Enc^K(IS)*DP]$. Note that B_{i+1} will be the $(NEXT)$ when i is the last number in the partition and B_{i-1} will be the $(PREV)$ when i is the first number in the partition. Figure 3 shows the plaintext domain and the boundaries generated in the encrypted domain for the metadata shown in Figure 2.

3.4 Encryption Function

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

$$(E(A_1),E(A_2),...,E(A_n))$$

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 $[B_i,B_{i+1})$. Note that B_{i+1} will be the $(NEXT)$ when i is the last number in the a partition. Using the example shown in Figure 3, $E(81)$ will be a random number in the interval $[B_{81},B_{82})$ while $E(80)$ will be a random number in the interval $[B_{80},B_1)$.

3.5 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-POPES 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 $p-1$. Details about choosing the initial

point for generating boundaries, the random distributions used for choosing encrypted values and decryption algorithms are given by (Kadhem et al., 2010).

4 CONDITIONS TRANSLATION

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* ∨ *Condition*)|(*Condition* ∧ *Condition*) |(→ *Condition*)

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

4.1 Partition Identification Functions

Before we discuss the translation of conditions, let us first define the necessary functions on the partitions identifiers. Those functions will be used to translate conditions that contain comparison operations.

Let A be an attribute, v be a value in the domain and i be a partition identifier. Table 1 shows the partition identification functions. Using the running example, $PID_A^<(50)=\{1,2\}$ and $PID_A^>(50)=\{4,5\}$.

Table 1: Partition Identification Functions.

PID_A	set of *PID* for attribute A
$PID_A(v)$	*PID* to which value v belongs in the domain of A
$PID_A^<(v)$	set of *PID* for attribute A that are less than the partition that contains v
$PID_A^>(v)$	set of *PID* for attribute A that are greater than the partition that contains v
$PID_A^{<i}$	set of *PID* for attribute A that are less than i
$PID_A^{>i}$	set of *PID* for attribute A that are greater than i

4.2 Translation of (*Attribute* θ *Value*) Conditions

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

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

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 $(B_{v+1}-1)$.

Attribute < Value: such condition arises in selection operations. Since the MV-POPES preserves the order of the encrypted values within the partition ($v_i<v_j \rightarrow E^K(v_i)<E^K(v_j)$, v_i and v_j belong to the same *PID*), the translation of $(A<v)$ is as follows:

$$(A^E<B_v \wedge A^E\geq B_{F(PID_A(v))}) \vee \bigvee_{i\in PID_A^<(v)} (A^E\geq B_{F(i)} \wedge A^E>B_{NEXT(i)})$$

where $F(i)$ is the first integer in the partition i, and $NEXT(i)$ is the next integer of the partition i. Simply, the result contains all encrypted values that are less than the left boundary (B_v) of the interval (I_v) within the partition that contains v. In addition, all partitions whose *PID* are less than the partition of v are included in the result. For example, the translation condition for the condition $(A<55)$ is:

$$(A^E<B_{55}\wedge A^E\geq B_{50})\vee(A^E\geq B_1\wedge A^E<B_{81})\vee(A^E\geq B_{21}\wedge A^E<B_{61})$$

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). So, instead of $(A^E<B_v)$, the condition will be $(A^E < B_{v+1})$.

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.

4.3 Translation of (*Attribute* θ *Attribute*) Conditions

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. Instead, we try to make it possible to compare ciphertext without decryption.

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.

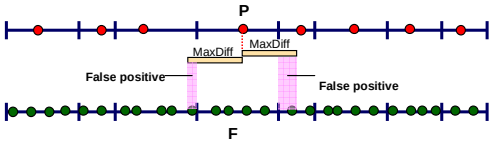


Figure 4: Overlap in connecting primary key with foreign key values based on equality condition.

Given a condition $(P\ \theta\ F)$, 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:

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

Here, $MaxDiff$ 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). Figure 4 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 as follow:

$$\bigvee_{i \in PID_P} \left((P^E \geq B_{F(i)} \wedge P^E < B_{NEXT(i)}) \wedge (P^E < F^E)\right) \vee \bigvee_{j \in PID_P^{>i}} \left((P^E \geq B_{F(i)} \wedge P^E < B_{NEXT(i)}) \wedge (F^E \geq B_{F(j)} \wedge F^E < B_{NEXT(j)})\right)$$

Simply, we check for each partition (i) in the domain, the values where $P<F$ and include all values in the partitions (j) that PID is greater than the partition (i).

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. The translation of this condition is same as the previous condition but we use $MaxDiff$ to ensure all equal values are connected together. So, instead of $P^E < F^E$ the condition will be $(P^E + MaxDiff) \leq F^E$.

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.

4.4 Translation of Composite Conditions

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 $(Attribute\ \theta\ Attribute)$. When both conditions are from $(Attribute\ \theta\ Value)$ the result will be exact. Note that the translation of comparison operators on same attribute will be either the union $(\cup)$ between sets of PIDs (in case of $C1 \vee C2$) or the intersection $(\cap)$ between sets of PIDs (in case of $C1 \wedge C2$).

Translation of $(\rightarrow Condition)$ depends on the condition type. When the condition is in the form of $(Attribute\ \theta\ Value)$, 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 $(Attribute\ \theta\ Attribute)$, 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 RELATIONAL OPERATORS

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 $(Attribute\ \theta\ Attribute)$ is attached to the operator, the returned answers might contain false positives. These answers are then filtered in client-side after decryption to generate the exact result. Beyond that, some operators cannot be performed fully 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. Table 2 shows the implementation of the operators over encrypted databases. The E on the operators emphasizes the fact that the operator is to be executed over the encrypted database. The L^E refers to the encrypted attributes.

Query Splitting. 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:

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

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

Table 2: Implementation of the operators over encrypted databases.

Operator	op	op^E
Selection (σ)	$\sigma_C(R)$	$D(\sigma^E_{CE}(R^E))$
Join ($\bowtie$)	$R^{\bowtie}_C T$	$\sigma_C\left(D(R^E \overset{\bowtie^E}{}_{CE} T^E)\right)$
Sorting (τ)	$\tau_L(R)$	$\tau_L\left(D(\tau^E_{LE}(R^E))\right)$
Projection (π)	$\pi_L(R)$	$D(\pi^E_{LE}(R^E))$
Grouping and Aggregation (γ)	$\gamma_L(R)$	$\gamma_L\left(D(\tau^E_{LG^E}(R^E))\right)$
Duplicate Elimination (δ)	$\delta(R)$	$\delta\left(D(\tau^E_{LE}(R^E))\right)$
Union ($\cup$)	$R \cup T$	$D(R^E \cup^E T^E)$ (based on bag) $\delta\left(D(\tau^E_L(R^E \cup^E T^E))\right)$ (set)
Difference ($-$)	$R-T$	$D(\tau^E_{LR^E}(R^E)) - D(\tau^E_{LT^E}(T^E))$

relations R^E on the server side.

6 SECURITY ANALYSIS

We can prove the security of our scheme against known plaintext attack and statistical attack by using the security definitions and theorems we defined in Section 2. For m distinct plaintext values and n corresponding distinct ciphertext values using our encryption scheme. The number of possible mappings (η) in an order preserving way is determined by the number of ways of partitioning the set of plaintext values into k non-empty partitions and the order of partitions in the encrypted domain. This number is:

$$\eta = \binom{n-1}{m-1} \sum_{i=1}^{m} \left(\binom{m-1}{i-1} i!\right)$$

where $\sum_{i=1}^{m} \left(\binom{m-1}{i-1} i!\right)$ is the possible ways to partitioning the plaintext domain and the possible orders for those partitions, and $\binom{n-1}{m-1}$ is the possible frequencies for the ciphertext values compared with the plaintext values in an order preserving way. We can clearly see that (η) is exponentially large even for small plaintext domain, so the mapping probability (μ) is significantly small. By ignoring the number of possible partitions and their orders which means considering just the new order for the plaintext values in the encrypted domain, the number of possible mappings is:

$$\eta = \binom{n-1}{m-1} m!$$

Note that the smallest number of η will be in the case of primary key attribute where $n = m$. The reason is that the possible frequencies in such case are 1. However, still η is large enough ($\eta = m!$) because still an attacker should figure out the new order of the plaintext in the encrypted domain. Thus, based on *theorem 1*, our scheme is secure against the known plaintext attack because the mapping probability (μ) is significantly small in all cases.

In our scheme, it is clear that the frequencies of encrypted values and plaintext values are different because one value is encrypted to many different values. Also, the statistical functions such as MIN, MAX, SUM are totally different on ciphertext and plaintext values. Thereby, the statistics on plaintext values Σ^{ρ} is diverse from the statistics on the corresponding ciphertext values Σ^{ς} using MV-POPES. Based on *theorem 2*, MV-POPES is secure against statistical attack because the similarity degree of statistics (Δ) is significantly small. Note that the aggregation functions (MIN, MAX, COUNT) are still evaluated directly on the encrypted database using the metadata. For example, MIN value for a domain will be the minimum encrypted value within the first partition and MAX value will be the maximum encrypted value within the last partition.

7 EXPERIMENTS

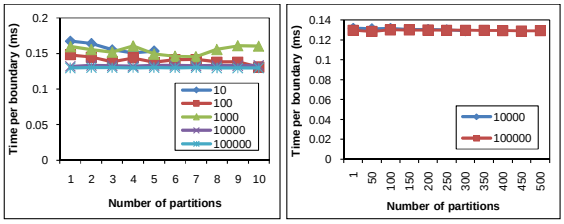
This section evaluates the performance of our encryption scheme. We have conducted many experiments to examine the validity and effectiveness of the architecture proposed in this paper. However, because of space limitations, we will discuss just four sets of experiments.

7.1 Experimental Setup

The experiments were conducted by implementing MV-POPES 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. The sets of evaluations was performed on different domains $\{10, 10^2, 10^3, 10^4, 10^5\}$ and various number of partitions (small and large number of partitions) with difference percentage (DP=0.05). The records on tables picked randomly from a uniform distribution between D_{min} and D_{max}. We used eqi-width as partitioning method.

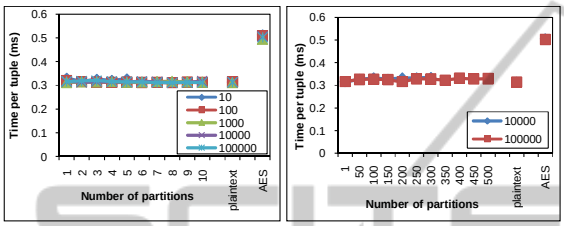
7.2 Performance for Generating Boundaries

The first set of evaluations was performed to examine the time needed to generate boundaries. The graph in Figure 5 shows the execution time for generating boundaries using the first integer in the encrypted domain as the *initial* point. The results show that we have



a) Small number of partitions. b) Large number of partitions.

Figure 5: Time per boundary (in ms) required to generate boundaries.



a) Small number of partitions. b) Large number of partitions.

Figure 6: Time per tuple (in ms) required to insert tuples.

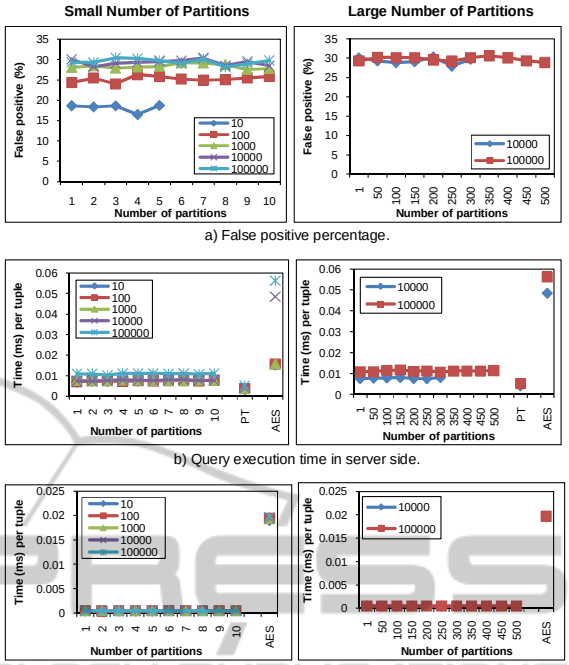
slightly better performance when the domain is large. That is because the time needed to encrypt *initial* point is divided by larger number in case of large domain. Generally, the number of partitions (both small and large) does not affect the performance of generating the boundaries.

7.3 Performance for Encryption

The second set of evaluations studied the encryption performance in our scheme using different domains and various numbers of partitions. Also, we compare the performance of our scheme with a database encrypted using AES. The table holds 100,000 records picked randomly from a uniform distribution between D_{min} and D_{max}. Figure 6 shows the times for encryption and inserting values in the tables for different domains. The results show that AES takes the longest time to insert tuple since the encryption time is much more than in MV-POPES. 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. Also, figure 6(a,b) shows that the encryption cost of our scheme does not change by increasing the number of partitions.

7.4 Performance for Equijoin

In the equijoin operation, we studied the percentage of false positives returned by performing a join operation over encrypted relations. Also, we studied the overhead on both the server and client sides. Two tables were used to perform this evaluation. The first



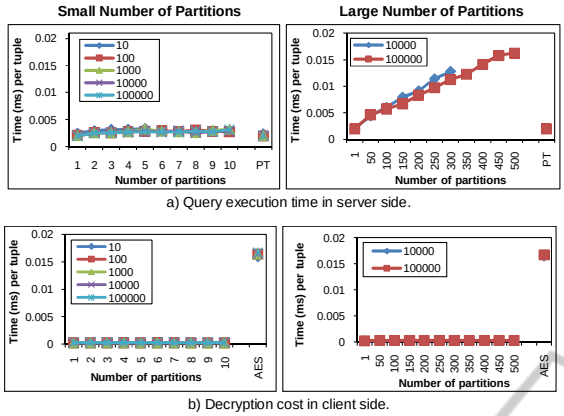
a) False positive percentage.

b) Query execution time in server side.

c) Decryption and filtering cost in client side.

Figure 7: Equijoin cost.

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. The percentage of false positives shown in Figure 7(a) increases with the domain size. That results due to the increase in the overlap between intervals in the encrypted scale when performing a join operation based on $MaxDiff$. However, the false positive percentage is same with small and large number of partitions. That because the overlap between intervals is not related to the number of partitions. From Figure 7(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 its approximately same as the join operation on the plaintext (PT) 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 (Hsueh, 2008). Figure 7(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, because of increased false positives. The results show that using small or large number of partitions does not affect the overhead on both server and client side. On the other hand, we can see the performance degradation when using AES compared with our scheme.



a) Query execution time in server side.

b) Decryption cost in client side.

Figure 8: Range query.

7.5 Performance for Range Query

The last sets of experiments studied the performance of range queries on MV-POPES. The query used in those experiments is retrieving all records that are greater than $(domain\ size/2)$. Figure 8(a) shows the range query execution times on plaintext, MV-POPES, and AES. The figure illustrates that query response time is approximately the same in plaintext and our scheme by using small number of partitions. In our scheme, using large number of partitions will cause more overhead on server side because the condition becomes more complicated. However, we think that the overhead on server side can be optimized by simplifying the translated condition. We will investigate that in details in future work. Figure 8(b) shows the decryption cost on client side using MV-POPES. We can see clearly that the decryption overhead is small and approximately fixed in our scheme using small and large number of partitions. The Figure also shows the time required to perform same range query on the database encrypted using AES. Here the time represents the decryption time and the time for performing the range query on the decrypted data. That is because the range query cannot be executed over the databases encrypted using AES. Using AES, range query takes much more time than our scheme on the client side because of decryption cost.

8 RELATED WORK

Many full OPE schemes (Agrawal et al., 2004; Chung and Ozsoyoglu, 2006; Ozsoyoglu et al., 2003; Wang and Lakshmanan, 2006) have been proposed, but no work discussed the security of the encryption schemes against known plaintext attack and statistical attack. Only (Wang and Lakshmanan, 2006) considered the frequency-based attack which is part of statistical attack in their encryption scheme. (Wang and Lakshmanan, 2006) proposed a new encryption scheme (Order preserving encryption with splitting and scaling (OPESS)) 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. However, this scheme still preserve the order for all encrypted values so it is possible to estimate at least the top and bottom parts of the plaintext domain. 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 becausehe 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.

The bucketing approach (Hore et al., 2004; Hacigümüş et al., 2002) is closely related to our scheme in sense of dividing the plaintext domain into many partitions (buckets). The encrypted database is augmented with additional information (the index of attributes), thereby allowing query processing to some extent at the server without endangering data privacy. The encrypted database in the bucketing approach contains etuples (the encrypted tuples) and corresponding bucket-ids (where many plaintext values are indexed to same bucket-id). In this scheme, executing a query over the encrypted database is based on the index of attributes. The result of this query is a superset of records containing false positive tuples. These false hits must be removed in a post filtering process after etuples returned by the query are decrypted. Because only the bucket id is used in a join operation, filtering can be complex, especially when random mapping is used to assign bucket ids rather than order preserving mapping. The number of false positive records depends on the number of buckets involved. Using a small number of buckets will hide the real values within the bucket index, but the filtering overhead can become excessive. On the other hand, a large number of buckets will reduce the filtering overhead, but the scheme will be vulnerable to estimation exposure. In bucketing, the projection operation is not implemented over the encrypted database, because a row level encryption is used. In addition, updating attributes in the bucketing approach requires that two attributes be updated, the bucket-id and the etuple. This means that all attributes in the row must be re-

encrypted, thereby increasing overhead for the update query.

Many researchers have investigated the problem of keywords searching on encrypted data using either symmetric encryption (Song et al., 2000), asymmetric encryption (Boneh et al., 2004) or a combination of symmetric, asymmetric encryption and hash functions (Dong et al., 2008). In spite of security vulnerabilities (like statistical attack (Song et al., 2000)) and significant overhead (Boneh et al., 2004; Dong et al., 2008), 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.

9 CONCLUSIONS

We propose a novel order preserving encryption scheme (MV-POPES) that is robust against known plaintext attack and statistical attack. In MV-POPES, we change the order of the plaintext values by portioning the domain into many partitions and assign a unique random number to each partition that represents the order in the encrypted domain. MV-POPES allows one integer to be encrypted to many different values using the same encryption key. It also preserves the order of the integer values within each partition to allow 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-POPES showed that security for sensitive data can be achieved with reasonable overhead, confirming the feasibility of the scheme. In the future, we will investigate the optimal algorithm for domain partitioning that minimize performance overhead in query processing and ensure high privacy. We also plan to study the encryption of non-integer data such as strings.

ACKNOWLEDGEMENTS

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

REFERENCES

AES (2001). Advanced encryption standard. *National Institute of Science and Technology*, FIPS 197.

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004). Order preserving encryption for numeric data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA. ACM.

Blum, M. and Micali, S. (1984). How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864.

Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In *EUROCRYPT 2004: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer.

Chung, S. S. and Ozsoyoglu, G. (2006). Anti-tamper databases: Processing aggregate queries over encrypted databases. In *ICDEW '06: Proceedings of International Conference on Data Engineering Workshops*, page 98, Washington, DC, USA. IEEE Computer Society.

DES (1977). Data encryption standard. *Federal Information Processing Standards Publication*, FIPS PUB 46.

Dong, C., Russello, G., and Dulay, N. (2008). Shared and searchable encrypted data for untrusted servers. In *Proceeedings of the 22nd annual IFIP working conference on Data and Applications Security*, pages 127–143, Berlin, Heidelberg. Springer-Verlag.

Hacigümüş, H., Iyer, B., Li, C., and Mehrotra, S. (2002). Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, New York, NY, USA. ACM.

Hore, B., Mehrotra, S., and Tsudik, G. (2004). A privacy-preserving index for range queries. In *VLDB '04: Proceedings of the 30th international conference on Very large databases*, pages 720–731. VLDB Endowment.

Hsueh, S. (February 2008). Database encryption in SQL server 2008 enterprise edition. *Microsoft White Papers*, SQL Server 2008.

Kadhem, H., Amagasa, T., and Kitagawa, H. (2010). MV-OPES: Multivalued - order preserving encryption scheme: A novel scheme for encrypting integer value to many different values. *IEICE Transactions*, 93-D(9):accepted.

Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. (1996). *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA.

Ozsoyoglu, G., Singer, D. A., and Chung, S. S. (2003). Anti-tamper databases: Querying encrypted databases. In *17th Annual IFIP Working Conference on Database and Applications Security, Estes Park*, pages 4–6.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

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

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *SP'00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA. IEEE Computer Society.

Wang, H. and Lakshmanan, L. V. S. (2006). Efficient secure query evaluation over encrypted XML databases. In *VLDB '06: Proceedings of the 32nd international conference on Very large databases*, pages 127–138.