Optimizing Selection Processing for Encrypted Database using Past Result Knowledge Base

Wai Kit Wong
Hang Seng Management College
wongwk@hsmc.edu.hk

Kwok Wai Wong
Hang Seng Management College
mkwai2016@gmail.com

Ho-Yin Yue
Hang Seng Management College
willyyue@hsmc.edu.hk

ABSTRACT

Data confidentiality is concerned in database-as-a-service (DBaaS) model. The cloud server should not have access to user's plain data. Data is encrypted before they are stored in cloud database. Query computation over encrypted data by the server is not straight-forward. Many research works have been done on this problem. A common goal is to let the server obtain the selection result without leaking information about plain data. In existing solutions, the selection result is simply dumped by the server after the query answer is returned. Our idea is to make use of such *past results* of selections to improve processing speed for new queries. We developed an indexing mechanism called *past result knowledge base* (PRKB) to improve processing speed of selection with comparison predicate(s) in EDBMS. All operations related to PRKB are done by the server only. In our empirical studies, PRKB can reduce processing cost by orders of magnitudes compared to the case PRKB is not used.

1 INTRODUCTION

In database-as-a-service (DBaaS) model, a data owner (DO) uploads its data to a database managed by a third party service provider (SP) who is responsible to answer DO's queries and provides administrative services, e.g., backup recovery and access control. Data confidentiality is concerned when SP is compromised, e.g., by a malicious DBA administrating the database server. For instance, a rogue DBA has stolen 2.3 millions customer records of a Fortune 500 company[1], including bank account and credit card information. Encrypted database management systems (EDBMSs), such as Cipherbase [2–4] and SDB [19, 35], are recently developed to address data confidentiality concerns in case SP is compromised. The idea in EDBMS is to use *application level encryption* where data is encrypted and decrypted by DO and the private keys are only known to DO. Even if an attacker somehow gets access to the database at SP, the attacker can only obtain encrypted information without the keys to decrypt the data.

A challenge in EDBMS is to allow SP to compute selection over encrypted data, without knowing any other information about plain data (so that minimal number of encrypted tuples are processed in the next operations, e.g., join and/or aggregation). Many solutions were proposed, e.g., [12, 25] for range query and [13] for keyword search. In this paper, we address optimization of computing selection with comparison predicate(s) in EDBMS.

Fig. 1 shows an overview of our method, *past result knowledge base* (PKRB). The results of past selections are consolidated and stored in PRKB at SP. SP can use PRKB to reduce processing cost

[1]http://www.computerworld.com/article/2542360/security0/database-admin-steals-2-3m-consumer-records-at-fidelity-national-subsidiary.html

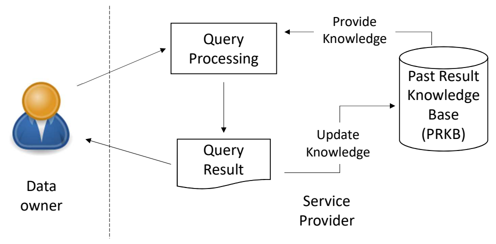


Figure 1: Overview of PRKB for query processing

Encrypted tuples	Selection		
	σ_1	σ_2	σ_3
$\overline{t_1}$	✓	X	✓
$\overline{t_2}$	X	✓	✓
$\overline{t_3}$	✓	✓	?
$\overline{t_4}$	✓	✓	?

Table 1: Example scenario. If an encrypted tuple satisfies (or does not satisfies resp.) a selection, a '✓' (or 'X' resp.) is shown. '?' denotes that the result for the encrypted tuple is not known yet.

of a new selection. We use the following example to illustrate the cost reduction idea.

Suppose there are 4 encrypted tuples $\overline{t_i}$ for $i = 1$ to 4. Each tuple has one attribute X only. There are 3 selections σ_1, σ_2, and σ_3, each with a simple comparison predicate in the form of '$X < c$' or '$X > c$' where c is a user-defined parameter unknown to SP. σ_1 and σ_2 are computed already. σ_3 is partially computed. The scenario is shown in Table 1.

SP can reason as following to determine the values of '?' in Table 1. The result of σ_1 partitions the encrypted tuples into two groups, $P_1 = \{\overline{t_1}, \overline{t_3}, \overline{t_4}\}$ and $P_2 = \{\overline{t_2}\}$. We use $P_i > P_j$ to denote that all tuples in P_i have a larger plain value than all tuples in P_j. Since σ_1 is a simple comparison predicate, there are only two possible scenarios: either (i) $P_1 > P_2$ or (ii) $P_2 > P_1$. We assume it is the case of scenario (i), i.e., $\overline{t_2}$ has the smallest plain value. Similarly, from the result of σ_2, SP obtains two partitions $P_3 = \{\overline{t_1}\}$ and $P_4 = \{\overline{t_2}, \overline{t_3}, \overline{t_4}\}$. It must be either $P_3 > P_4$ or $P_4 > P_3$. Since $\overline{t_2} \in P_4$ and $\overline{t_2}$ has the the smallest value, it must be the case $P_3 > P_4$. As a result, SP obtains the order of (plain values of) all encrypted tuples in this scenario as '$\overline{t_2} < \overline{t_3}, \overline{t_4} < \overline{t_1}$'. This order information is *partial* only as SP cannot determine which encrypted tuple is larger for $\overline{t_3}$ and $\overline{t_4}$. For σ_3, $\overline{t_1}$ and $\overline{t_2}$ are found to satisfy the selection condition. $\overline{t_3}$ and $\overline{t_4}$ are ordered between $\overline{t_1}$ and $\overline{t_2}$, so $\overline{t_3}$ and $\overline{t_4}$ must also satisfy the selection condition, i.e., both '?' must be '✓' in Table 1. SP can perform the same analysis for scenario (ii) and obtains the same conclusion.

In the above example, SP can determine whether $\overline{t_3}$ and $\overline{t_4}$ satisfy a new selection σ_3 without accessing them and thus saves the processing cost on these two encrypted tuples. Such saving is significant because the process to check whether an encrypted tuple satisfies a comparison predicate is usually expensive. For example, in Cipherbase [2], the encrypted tuple is decrypted (within a trusted hardware) before the predicate is tested. The additional decryption cost is significant compared to the cost of a simple comparison. As demonstrated in our empirical evaluation, only a small portion of encrypted tuples cannot be determined using PRKB and only this group of tuples require to be processed by SP using the usual cryptographic way. The overall processing cost is greatly reduced.

We highlight the distinguished features of PRKB as follows:

- *DO is not involved* in any part (e.g., building or using) of PRKB. No information is required to be sent from DO to SP for PRKB. All information in PRKB is solely based on what SP has observed in past query computation. It can be realized easily that PRKB does not leak more information than the underlying EDBMS.
- *PRKB is compatible to any encryption scheme* that tells SP which encrypted tuples satisfy the selection. As long as the encryption scheme provides such trapdoor, PRKB can be used on top of it. This allows PRKB to be deployed on top of many existing systems, e.g., Cipherbase and SDB.
- *Information in PRKB is in plain.* Unlike encrypted index, PRKB consists of information about past selection results. All operations related to PRKB are efficient and the size of PRKB is compact.

The rest of this paper is organized as follows. We discuss related work in Sec. 2. In Sec. 3, we define the models used in our problem. We describe what PRKB is and how SP builds PRKB in Sec. 4. In Sec. 5 and Sec. 6, we describe our algorithms for processing a single comparison predicate and multi-dimensional range query respectively. We describe how to handle database update in Sec. 7. We empirically evaluate PRKB and related algorithms in Sec 8. We conclude this paper in Sec. 9.

2 RELATED WORK

Different solutions were developed for computing individual database operations over encrypted data, e.g., range query [6, 12, 25, 28], keyword search [7, 13] and join [26]. A potential problem of these solutions is that integration of the above solutions is not trivial. Encrypted database management system (EDBMS) [2, 4, 5, 19, 29, 33, 35] offers an integrated solution that supports a wide range of SQL operations. We aim to deploy our optimization technique for selection with comparison predicate(s) in EDBMS. We first review existing EDBMSs in Sec. 2.1. Then, we review indexing options for EDBMS in Sec. 2.2. Lastly, we review techniques that hide selection results from SP in Sec. 2.3.

2.1 EDBMS

There are several approaches to implementing EDBMS.

The first approach, e.g., TrustedDB [5] and Cipherbase [2, 4], makes use of a trusted hardware. There is a trusted machine (TM) at SP. For instance, TrustedDB uses Cryptographic Coprocessor and Cipherbase uses FPGA. Such hardware devices are (physically) tamper-resistant. An attacker is assumed not to be able to see the data or process inside TM. TM is given the decryption key of DO. Any computation related to encrypted data can be handled by TM. For instance, to process a comparison predicate, the encrypted value of each tuple and the instructions (with encrypted user parameters) are passed to TM. TM decrypts the data, makes the comparison, and returns the comparison result to SP.

The second approach uses secret sharing methods, e.g., SDB [19, 35]. Secret sharing splits each data item into shares. Some shares are stored at DO[2] while some are stored at SP. Without collecting all the shares of a data item, SP cannot recover its plain value. Multi-party computation (MPC) operators are developed to execute database operations by DO and SP communicating with each other in multiple rounds. An advantage of MPC approach is that any computation can be computed [15]. However, it incurs a high communication cost in query processing.

Another approach is to use multiple encryption schemes, each to support a different set of operations. Example system is CryptDB [29] / MONOMI [33]. (MONOMI is an extension of CryptDB to further support aggregation.) Specifically, CryptDB uses order preserving encryption (OPE), e.g., [1, 28], to process comparison predicates. OPE preserves the numerical order of plain data under encryption, i.e., if $x > y$, $E(x) > E(y)$ for any x, y where E denotes the encryption function. Comparison predicates can be computed efficiently and indexing over OPE-encrypted data is just like indexing over plain data. A downside of this approach is that it leaks the total order of plain data to SP. Recent studies show that inference attack [22, 27] can recover accurately the plain data of OPE-encrypted data using the total order information.

2.2 Indexing on encrypted data

Our method, PRKB, is similar to indexing, because SP uses additional space to remember past results in order to boost the performance in query computation. PRKB has a significant difference to mainstream indexing methods for encrypted data: *PRKB is solely done by SP*, while existing indexing mechanisms for encrypted data require DO's involvement, e.g., to build the encrypted index, or requires DO to encrypt data using specific algorithms. In the following, we briefly discuss existing indexing methods for encrypted data.

In [11, 14, 31], an encrypted index tree is built by DO and stored at SP. SP simply serves as a storage without processing capabilities. DO retrieves parts of the index from SP iteratively to traverse the index. Data confidentially can be proven as SP never see any data in plain. Access pattern of the index can also be protected from SP, e.g., in [14], at the cost of increased processing and communication cost. DO has a significant amount of workload. We do not prefer this approach in DBaaS where DO may not be as powerful as SP.

[12, 23, 25] developed new encryption methods for computing comparison predicate or range query with indexing support. In [23], data is encrypted using a special vector encryption method [34]. An index can be built over these encrypted values. A problem is there is information leaked in the encryption scheme, as shown in [17], and plain data can be recovered in some scenarios. In [25], a security notion of index indistinguishable is introduced and an index is developed that achieves the proposed security notion. However, such security notion is proven to be weak [10]. In [12], it transforms the problem of range query into keyword search and uses existing searchable symmetric encryption (SSE) [7, 8] to compute the result. A series of schemes with different indexing options, each offers different security strength, is developed. In all methods above, SP see the selection results as this is the objective of the problem. Our method PRKB can also be implemented on top of these encryption methods. In our empirical studies, we will compare our method to [12].

[2]In SDB, the shares at DO can be generated using *an RSA-like share-generating function.* This reduces storage cost at DO.

Some EDBMSs we discussed in Sec. 2.1 also use indexing. Cipherbase uses an encrypted B+-tree. The index reveals the total order of plain data to SP and is thus also vulnerable to inference attack. SDB uses domain-partitioning index [18, 21]. The data domain is divided into partitions by DO. SP is informed by DO the partition each data item falls into. Due to additional information leak, we do not consider these methods suitable for our problem.

2.3 Hiding selection result from SP

Our problem assumes the selection result is known by SP. The selection result can potentially be hidden by access pattern hiding technique, e.g., oblivious RAM (ORAM) [16, 32] and/or private information retrial [9]. For instance, [4] discussed an option to integrate ORAM in Cipherbase. The trade-off is that each data access has a polylog cost. Due to high overhead, ORAM was not implemented in Cipherbase. Similar to Cipherbase, other EDBMSs we reviewed in Sec. 2.1 do not use any access pattern hiding technique. Our method can be deployed on existing EDBMSs, including TrustedDB, CipherBase and SDB.

3 MODELS

3.1 Preliminary: EDBMS

Our problem is based on an underlying encrypted database management system (EDBMS). In this section, we describe the EDBMS model, that is compatible to EDBMSs that we discussed in Sec. 2.1.

Parties. There are two parties: data owner (DO) and service provider (SP). DO has a set of relational tables in the database. Each table T is a set of tuples, i.e., $T = \{t_i\}$ where t_i denotes the i-th tuple. DO encrypts T to be $\overline{T}$ such that $\overline{T} = \{\overline{t_i} \mid \overline{t_i} = E(t_i)$ for every $t_i \in T\}$ where E denotes the encryption function. (We use $\overline{X}$ to represent the encrypted version of X in the rest of the paper.) $\overline{T}$ is sent to SP for storage and DO does not store T. The private keys are only known by DO.

Selection processing. EDBMS allows selection to be computed over encrypted data by SP. The selection contains one or more comparison predicates, e.g., '$X > 10$'. SQL supports a wide range of comparison predicates, e.g., comparison operators ($>$, $<$, $\geq$, and $\leq$), and BETWEEN operator etc. In general, for existing EDBMSs that employ attributed-based encryption, SP can tell (i) which type of operator is used because the algorithms to process them are different[3]; and (ii) which encrypted attribute is concerned so that other encrypted values of other attributes are not accessed during selection processing. In our problem, we focus on comparison predicate in the form of 'X *op c*' where *op* is a comparison operator. In Appendix A, we briefly discuss how BETWEEN operator can be handled. As we discussed in Sec. 2.1, there are different ways to implement EDBMS to support selection processing. An ideal method allows SP to observe the selection result without seeing any information about plain data. We use the following model (based on the model in [24]) to capture the selection processing mechanism of EDBMS.

QPF model. Let $p_i, \overline{p_i}$ be the plain and encrypted version of a comparison predicate. There is a query processing function (QPF) Θ such that

$$\Theta(\overline{p_i}, \overline{t_j}) = \begin{cases} 1, & \text{if } t_j \text{ satisfies } p_i \\ 0, & \text{otherwise} \end{cases}$$

[3]Comparison operators ($>$, $<$, $\geq$, and $\leq$) are handled by the same algorithm and hence SP cannot distinguish them.

$\overline{p_i}$ is generated by DO and acts as a trapdoor that allows SP to observe the selection result of p_i. SP cannot observe the selection result of any predicate without such trapdoor, i.e., SP's knowledge of selection results is limited by *number of comparison predicates* issued by DO.

Applications. The above QPF model is generic [24] such that the selection processing mechanisms of majority of related work can be captured. For instance, among the EDBMSs that we have studied in Sec. 2.1, TrustedDB [5], Cipherbase [2, 4], and SDB [19, 35] satisfy this model, i.e., our method can be implemented on top of these systems. CryptDB [29] and MONOMI [33] also satisfy our EDBMS model, but they also reveal the total order of plain data (in addition to selection results) to SP. With total order known, there are simpler options to optimize query processing and security strength is significantly lowered. CryptDB and MONOMI are not our target applications and our underlying EDBMS should not reveal the total order information.

Our method can also be integrated to many other standalone methods for selection processing on encrypted database, e.g., [12, 20, 25, 30]. As long as these methods reveal the selection results to SP, our method can be applied. Methods that do not reveal the selection results to SP (see Sec. 2.3) are not our target applications. In our empirical studies, we will compare our method to the indexing method in [12], the state-of-the-art method for range query processing on encrypted data.

3.2 Problem in this paper: optimizing selection with comparison predicate

Our objective is to reduce the processing cost of selection at SP. A baseline method for SP is to test all encrypted tuples using the QPF one by one. The bottleneck of performance is QPF evaluation. Note that a comparison can be done extremely fast, e.g., in one cycle. QPF evaluation is relatively more expensive in general. For example, in Cipherbase, the encrypted tuple is decrypted within a trusted machine before the comparison is done. The decryption cost is significant compared to the simple comparison. Our goal is to *reduce number of QPF uses.*

Our problem is similar to an indexing problem, which trades (SP's) storage for speed. However, unlike traditional indexing problem for encrypted data, our method (i) does not rely on specific encryption method; and (ii) does not require DO's involvement[4] in building and using "our index".

Our indexing mechanism, namely past result knowledge base (PRKB), composes of the following 4 algorithms.

$\mathcal{I} \leftarrow$ initPRKB($\overline{T}$): is run by SP to initialize the index $\mathcal{I}$.

$\mathcal{I} \leftarrow$ updatePRKB($\mathcal{I}, \overline{p_i}$): is run by SP to update the index $\mathcal{I}$ with an encrypted predicate $\overline{p_i}$.

$\langle \overline{T_W}, \overline{T_{NS}} \rangle \leftarrow$ QFilter($\overline{T}, \mathcal{I}, \overline{p_i}$): is run by SP to find two exclusive subsets of $\overline{T}$ using the index: (i) $\overline{T_W}$ represents the 'Winner' group. All encrypted tuples in this group must satisfy the plain predicate p_i; and (ii) $\overline{T_{NS}}$ represents the 'Not sure' group. Encrypted tuples in this group may satisfy p_i but some may be false positives. This group of encrypted tuples requires further processing by SP to confirm the exact selection result.

[4]DO may still be involved in QPF evaluation, e.g., in SDB [19, 35], and our method does invoke QPF. We do not count this as DO's involvement for the index because such involvement exists in EDBMS without our index.

$\overline{T_{\text{WNS}}} \leftarrow$ QScan($\overline{T_{\text{NS}}}, \mathcal{I}, \overline{p_i}$): is run by SP to confirm the exact selection result by examining each encrypted tuple one by one. QScan is similar to a linear scan but with optimization using information from PRKB.

SP can initiate PRKB (for an attribute) by *initPRKB* to create an 'empty' knowledge base as there is no past result observed by SP yet. As SP receives queries from DO, SP observes new selection results and can use them to extend PRKB using *updatePRKB*.

After executing *QFilter* and *QScan*, the selection result is then $\overline{T_{\text{W}}} \cup \overline{T_{\text{WNS}}}$. Fig.2b shows the procedure and messages in communication of EDBMS using PRKB. In contrast, Fig.2a shows the same procedure of EDBMS without using PRKB. As we will show in the paper, *QFilter* is cheap and $\overline{T_{\text{NS}}}$ is significantly smaller than $\overline{T}$. Only $\overline{T_{\text{NS}}}$ is processed by QScan. The overall cost can be thus reduced.

3.3 Security discussions

In the security model of EDBMS, an attacker has compromised SP and is able to observe anything SP can see. Or in simpler interpretation, SP is the attacker. An ideal situation is that SP observes no information about plain data. Unfortunately, there is some inherit information leak that we cannot avoid. The selection results and any information derived from them can be seen by SP because it is the objective of selection processing over encrypted data. The security goal of EDBMS is then to minimize leakage of any other information about plain data.

In our problem, we use the same attack model as EDBMS that the attacker has compromised SP. The security goal is to minimize additional leakage to SP caused by our method. As we will show in the paper, *our indexing method PRKB is built and used solely by SP using existing selection results.* No (encrypted or plain) information is ever sent from DO to SP for our index. For instance, readers can compare Fig.2b and Fig.2a which show the communication between DO and SP in EDBMS with or without using PRKB. The messages in the two cases are identical. Any information that can be derived by SP from PRKB can also be obtained by SP in EDBMS without PRKB. There is no additional leakage caused by PRKB.

Another important issue is that selection results are assumed to be observed by SP. We remark that this assumption is held in many existing methods. As discussed in [24] that selection results allow SP to eventually recover a total order of encrypted data on their plain values. The total order information can then be used in inference attack [22, 27] to recover accurate plain values. Data confidentiality is completely lost. The technique in [24] requires SP to observe $O(D^4)$ queries, where D is the domain size of an attribute, so as to let SP recover the total order information. Experiments in [24] showed that the total order can be recovered in a short time for a small data domain (e.g., $D \leq 365$). On the other hand, when the domain size D is large, it becomes impractical for SP to collect $O(D^4)$ queries for the attack. Yet, SP is able to observe certain number of queries. This allows SP to observe *partial* ordering information. We thus performed empirical evaluation to see how much ordering information SP can recover when SP observes limited number of queries. The details are presented in Sec. 8.1.

4 BUILDING PAST RESULT KNOWLEDGE BASE (PRKB)

In this section, we present what information SP can observe in query processing of EDBMS and how SP can use the observed information to build a past result knowledge base (PRKB). PRKB can then be used by SP to reduce query processing cost for new range queries.

Consider a comparison predicate p_C in the form of 'C *op x*' where x is a user-defined query parameter and *op* is one of the following: >, <, ≥, and ≤. $\overline{p_C}$ is a trapdoor generated by DO that allows SP to observe, using QPF Θ, whether an encrypted tuple satisfies the predicate without seeing the plain data and plain predicate. Note that SP does not know which comparison operator *op* is used in p_C. SP can divide the encrypted tuples in $\overline{T}$ into two partitions: (i) P_T: the partition of encrypted tuples where QPF outputs 1, i.e.,

$$P_T = \{\overline{t_i} \mid \overline{t_i} \in \overline{T} \text{ and } \Theta(\overline{p_C}, \overline{t_i}) = 1\}$$

and (ii) P_F: the partition of encrypted tuples P_F where QPF outputs 0, i.e.,

$$P_F = \{\overline{t_i} \mid \overline{t_i} \in \overline{T} \text{ and } \Theta(\overline{p_C}, \overline{t_i}) = 0\}$$

We can easily prove that either all encrypted tuples in P_T have a larger plain value on C than all encryped tuples in P_F or it is the reverse case. For example, consider a comparison predicate '$C < 9$'. P_T contains all encrypted tuples with plain values on C less than 9. P_F contains all encrypted tuples with plain values on C greater than or equal to 9. All encrypted tuples in P_F have a larger plain value on C than any encrypted tuple in P_T. Note that if the comparison predicate is '$C > 9$', all encrypted tuples in P_T have a larger plain value on C. SP cannot distinguish which set of encrypted tuples, P_T or P_F, is having larger plain values than the other. To capture the above special ordering relationship between two sets of encrypted tuples, we define two symbols below.

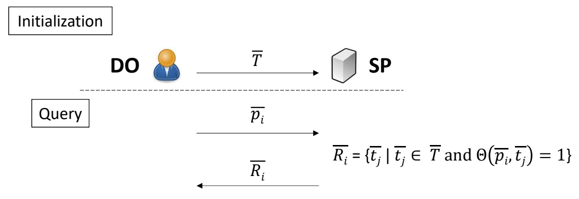
Definition 4.1. (Relationship between partitions.) Let $P_1 = \{\overline{t_i}\}$ and $P_2 = \{\overline{t_j}\}$ be two sets of encrypted tuples, $t_i[C]$ be the plain value of t_i on attribute C. We write $P_1 \overset{C}{<} P_2$ if $\forall \overline{t_i} \in P_1, \forall \overline{t_j} \in P_2, t_i[C] < t_j[C]$. We write $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \ldots \overset{C}{\mapsto} P_n$ if either (i) $P_1 \overset{C}{<} P_2 \overset{C}{<} \ldots \overset{C}{<} P_n$ or (ii) $P_1 \overset{C}{>} P_2 \overset{C}{>} \ldots \overset{C}{>} P_n$.

If the comparison predicate is '$C < 9$', $P_T \overset{C}{<} P_F$. However, SP does not observe the plain comparison predicate. SP can only conclude that $P_T \overset{C}{\mapsto} P_F$. The ordering information SP can learn from $\overline{p_C}$ is *partial* only because (i) SP does not know the ordering relationship between individual tuples in a partition; and (ii) SP cannot conclude which partition of P_T and P_F is actually larger.

Definition 4.2. (Partial order partitions of a relational table.) Let $\overline{T}$ be a set of encrypted tuples. We define partial order partitions of $\overline{T}$, denoted as POP_k^C, as a set of k partitions $P_i \subset \overline{T}$ s.t. (i) $P_i \bigcap P_j = \emptyset$ for $i \neq j$; (ii) $\bigcup_{i=1}^{k} P_i = \overline{T}$; and (iii) $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \ldots \overset{C}{\mapsto} P_k$.

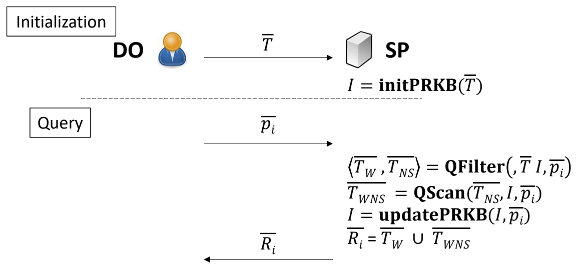
From a single encrypted predicate $\overline{p_C}$ in the above example, SP finds $POP_2^C : P_T \overset{C}{\mapsto} P_F$. With more encrypted predicates observed, SP can enhance its past result knowledge by extending POP_2^C. Before we discuss how SP extends its knowledge, we define two concepts related to a new encrypted predicate on existing partition order partitions POP_k^C for $k \geq 2$.

Definition 4.3. (Trapdoor equivalence.) Let $\overline{p_1}$ and $\overline{p_2}$ be two encrypted comparison predicates on the same attribute C of an encrypted table $\overline{T}$. Let $P_{ab} = \{\overline{t_i} \mid \overline{t_i} \in \overline{T} \text{ and } \Theta(\overline{p_a}, \overline{t_i}) = b\}$ for



Initialization
DO
$\overline{T}$
SP
Query
$\overline{p_i}$
$\overline{R_i} = \{\overline{t_j} \mid \overline{t_j} \in \overline{T} \text{ and } \Theta(\overline{p_i}, \overline{t_j}) = 1\}$
$\overline{R_i}$

(a) EDBMS without PRKB



Initialization
DO
$\overline{T}$
SP
$I = \textbf{initPRKB}(\overline{T})$
Query
$\overline{p_i}$
$\langle \overline{T_W}, \overline{T_{NS}} \rangle = \textbf{QFilter}(\overline{T}, I, \overline{p_i})$
$\overline{T_{WNS}} = \textbf{QScan}(\overline{T_{NS}}, I, \overline{p_i})$
$I = \textbf{updatePRKB}(I, \overline{p_i})$
$\overline{R_i} = \overline{T_W} \cup \overline{T_{WNS}}$
$\overline{R_i}$

(b) EDBMS with PRKB

Figure 2: Communication protocol of EDBMS between DO and SP

$a = 1$ or 2, $b = 0$ or 1. $\overline{p_1}$ is said to be *equivalent* to $\overline{p_2}$ if either (i) $P_{10} = P_{20}$ and $P_{11} = P_{21}$; or (ii) $P_{10} = P_{21}$ and $P_{11} = P_{20}$.

Definition 4.4. (Homogeneous partition & output-isomorphic partitions.) Given $POP_k^C : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$ and an encrypted predicate $\overline{p}$ on attribute C. A partition P_a is said to be *homogeneous* w.r.t. $\overline{p}$ if $\forall \overline{t_i}, \overline{t_j} \in P_a, \Theta(\overline{p}, \overline{t_i}) = \Theta(\overline{p}, \overline{t_j})$. P_a is said to be *T-homogeneous* (*F-homogeneous* resp.) w.r.t. $\overline{p}$ if $\forall \overline{t_i} \in P_a, \Theta(\overline{p}, \overline{t_i}) = 1$ (0 resp.). Two partitions P_a, P_b are said to be *output-isomorphic* w.r.t. $\overline{p}$ if either (i) both partitions are T-homogeneous or (ii) both partitions are F-homogeneous.

We explain the intuition of the above two definitions below. Two equivalent encrypted predicates divide the encrypted table $\overline{T}$ into the same two partitions. All encrypted tuples in a homogeneous partition have the same QPF output. A homogeneous partition is further labeled T-homogeneous (or F-homogeneous) if the QPF output is 1 (or 0) for all encrypted tuples in the partition. A non-homogenoeus partition contains tuples with mixed QPF outputs, i.e., some gives 1 and some gives 0. Two output-isomorphic partitions means that all encrypted tuples in the partitions have the same QPF output. Two encrypted predicates are equivalent if they divide the encrypted tuples into the same two partitions. Note that two equivalent encrypted predicates do not necessary mean their plain comparison predicates are the same. For example, if two comparison predicates are '$C < 9$' and '$C > 8$' and there is no tuple with value 8-9, the two corresponding encrypted predicates give the same two partitions, i.e., they are equivalent. Since equivalent encrypted predicates give the same partitions, only inequivalent encrypted predicates provide different partitioning information which can enhance SP's knowledge.

Now, we consider what SP observes when there is a new encrypted predicate with an existing POP_k^C. We summarize the scenario in the following lemma.

Lemma 4.5. *Given $POP_k^C : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$. Let $\mathbb{P}$ be the set of encrypted predicates for deriving POP_k^C. Let $\overline{p}$ be a new encrypted predicate on attribute C. The following two cases must hold.*

Case 1: $\overline{p}$ is equivalent to some encrypted predicate in $\mathbb{P}$ if and only if there is a separating point *s s.t. (i) all partitions P_i for $i = 1$ to s are output-isomorphic, (ii) all partitions P_j for $j = s+1$ to k are output-isomorphic; and (iii) P_i and P_j are not output-isomorphic for $i = 1$ to s and $j = s+1$ to k.*

Case 2: $\overline{p}$ is inequivalent to all encrypted predicates in $\mathbb{P}$ if and only if there is a separating point *s s.t. (i) all partitions P_i for $i = 1$ to $s-1$ are output-isomorphic, (ii) all partitions P_j for $j = s+1$ to k are output-isomorphic; (iii) P_i and P_j are not output-isomorphic for $i = 1$ to $s-1$ and $j = s+1$ to k; and (iv) P_s is non-homogeneous.*

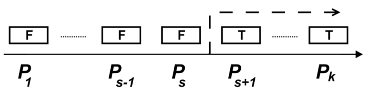


Figure 3: Example instance of Case 1 in Lemma 4.5.

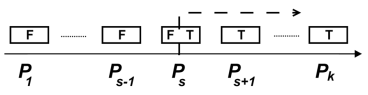


Figure 4: Example instance of Case 2 in Lemma 4.5.

Fig. 3 and Fig. 4 show the examples of Case 1 and Case 2.

Now, SP obtain $POP_k^C : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$, generated based on a set of encrypted predicates $\mathbb{P}$. Note that when SP is given a new encrypted predicate $\overline{p'}$, SP does not know whether $\overline{p'}$ is equivalent to some encrypted predicate in $\mathbb{P}$. According to Lemma 4.5, SP observes a non-homogeneous partition only in case 2. Reversely, if SP observes a non-homogeneous partition, SP can conclude that $\overline{p'}$ is inequivalent, i.e., $\overline{p'}$ allows SP to extend POP_k^C.

Assume now SP receives a new inequivalent encrypted predicate $\overline{p'}$. Let P_s be the non-homogeneous partition in POP_k^C. Since P_s is non-homogeneous, SP can divide P_s into two smaller partitions based on the outputs of Θ:

$$P_{s_T} = \{\overline{t_i} \mid \overline{t_i} \in P_s \mid \Theta(\overline{p'}, \overline{t_i}) = 1\} \text{ and}$$

$$P_{s_F} = \{\overline{t_i} \mid \overline{t_i} \in P_s \mid \Theta(\overline{p'}, \overline{t_i}) = 0\}$$

For example, in Fig. 4, the encrypted predicate divides P_s into P_{s_T} on the right and P_{s_F} on the left. P_{s_T} and P_{s_F} are now homogeneous. Either one of them must be output-isomorphic to P_{s-1} and the other partition must be output-isomorphic to P_{s+1}. Without loss of generality, assume P_{s_F} is output-isomorphic to P_{s-1} and P_{s_T} is output-isomorphic to P_{s+1} (like the scernaio in Fig. 4). SP can conclude that $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_{s-1} \overset{C}{\mapsto} P_{s_F} \overset{C}{\mapsto} P_{s_T} \overset{C}{\mapsto} P_{s+1} \overset{C}{\mapsto} P_k$. As a result, SP extends POP_k^C to POP_{k+1}^C with one more inequivalent encrypted predicate. By mathematical induction, SP can compute POP_k^C with $k-1$ inequivalent encrypted predicates.

The above discussion only shows that SP can observe POP_k^C with $k-1$ inequivalent encrypted predicates. We will describe how SP can efficiently update POP_k^C to POP_{k+1}^C with an additional encrypted predicate in Sec. 5.3 after we discuss how we make use of POP_k^C to optimize selection processing of comparison predicates. POP_k^C represents the knowledge extracted from past queries. Technically, PRKB contains only one item: POP_k^C. When SP decides to build PRKB on attribute C, the algorithm *initPRKB*($\overline{T}$) initiates PRKB as POP_1^C where all encrypted tuples in $\overline{T}$ residue in one big partition. As SP receives an inequivalent encrypted predicate, SP extends its PRKB from POP_k^C to POP_{k+1}^C.

5 SINGLE COMPARISON PREDICATE PROCESSING

In this section, we describe our method of SP processing comparison predicate using PRKB. As discussed in Sec. 4, PRKB contains one single item POP_k^C, which is a set of k partitions of encrypted tuples. Let P_i be a partition in POP_k^C for $i = 1$ to k s.t. $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$. Let $\overline{p}$ be an encrypted predicate SP receives from DO. From lemma 4.5, there is a *separating point s* such that it divides the partitions (except the non-homogeneous partition in Case 2) into two groups where all partitions in the same group are output-isomorphic to each other w.r.t. $\overline{p}$, i.e., Θ outputs the same for all encrypted tuples in all partitions within the same group. If SP knows the value of s, SP can determine the QPF outputs of all encrypted tuples in the above two groups with 2 QPF uses only. This can significantly save the computational cost at SP. However, SP does not know s in the beginning. So, the first task of SP in processing a comparison predicate is to find out the separating point s. This is done by the algorithm *QFilter*. *QFilter* can narrow the possible candidates of s down to just two candidates, i.e., only two out of k partitions could be non-homogeneous. Since there are always two candidates of partitions, we call them *Not-sure pair (NS-pair).* The QPF outputs of encrypted tuples in all the other $k-2$ partitions can be determined right away. Then, the algorithm *QScan* will scan every encrypted tuple in NS-Pair to confirm the value of s, with early stop strategy applied.

In the following, we first present how *QFilter* helps SP find out the separating point efficiently in Sec. 5.1. Then, we present *QScan* in Sec. 5.2. Finally, we discuss how SP updates PRKB from POP_k^C to POP_{k+1}^C efficiently in Sec. 5.3.

5.1 QFilter: searching for NS-Pair

Before we talk about the algorithm *QFilter*, we present the following lemma about searching for separating point s.

LEMMA 5.1. *Given partial order partitions POP_k^C of an encrypted table $\overline{t}$ and an encrypted predicate $\overline{p}$ on C. Let P_i be a partition in POP_k^C for $i = 1$ to k s.t. $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$. Let s be the* separating point *mentioned in lemma 4.5. Let $\overline{t_x}, \overline{t_y}$ be 2 encrypted tuples in P_a, P_b respectively s.t. $a < b$. We have if $\Theta(\overline{p}, \overline{t_x}) = \Theta(\overline{p}, \overline{t_y})$, then $s \le a$ or $s \ge b$.*

In the beginning, the separating point s may be any value from 1 to k. Lemma 5.1 helps to prune the candidates of s. There are two important observations from Lemma 5.1: (i) SP just needs to test on one sample encrypted tuple in P_a and P_b; and (ii) Lemma 5.1 does not prune the case of $s = a$ and $s = b$.

Following observation (i), SP adopts a sampling strategy to make use of the pruning shown in lemma 5.1. In the rest of the paper, we use '$P_i.sample$' to denote the random encrypted tuple drawn from a partition P_i. Then, in observation (ii), since the pruning always leave at least two candidates, SP cannot confirm the actual separating point using only sampled encrypted tuples. The sampling technique will always reduce the number of candidates to exactly 2. Thus, we call the two partitions corresponding to these 2 candidates *Not-sure pair (NS-pair).* After SP finds NS-Pair using the sampling technique, SP scans the two partitions and confirm the separating point using *QScan*.

Algorithm 1 shows the pseudo code of QFilter.

Algorithm 1: QFilter

```
Input  : Encrypted table $\overline{T}$
Input  : PRKB $\mathcal{I} = POP_k^C : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \dots \overset{C}{\mapsto} P_k$
Input  : Encrypted predicate $\overline{p_i}$
Output : <$\overline{T_W}, \overline{T_{NS}}$>
$label_1 = \Theta(\overline{p_i}, P_1.sample)$ ;
$label_k = \Theta(\overline{p_i}, P_k.sample)$ ;
if $label_1 = label_k$ then
    // boundary case
    if $label_1 = 1$ then
        $\overline{T_W} = \bigcup_{j=2}^{k-1} P_j$ ;
    else
        $\overline{T_W} = \phi$ ;                    // $\overline{T_W}$ is empty
    end
    $\overline{T_{NS}} = \langle P_1, P_k \rangle$ ;
else
    // use binary search to locate NS-Pair
    $a = 1$ ;            // first partition (head)
    $b = k$ ;            // last partition (tail)
    do
        $m = \lfloor \frac{a+b}{2} \rfloor$;
        $label_m = \Theta(\overline{p_i}, P_m.sample)$ ;
        if $label_a = label_m$ then
            $a = m$ ;
        else
            $b = m$ ;
        end
    while $b - a > 1$;
    $\overline{T_{NS}} = \langle P_a, P_b \rangle$ ;
    if $label_1 = 1$ then
        $\overline{T_W} = \bigcup_{j=1}^{a-1} P_j$ ;
    else
        $\overline{T_W} = \bigcup_{j=b+1}^{k} P_j$ ;
    end
end
return <$\overline{T_W}, \overline{T_{NS}}$>;
```

SP starts the search by applying Θ on $P_1.sample$ and $P_k.sample$ (line 1-2). There are two possible scenarios.

Scenario (i): if $\Theta(\overline{p_i}, P_1.sample) = \Theta(\overline{p_i}, P_k.sample)$, we call this the boundary case (line 5-10). Following Lemma 5.1, $s \le 1$ or $s \ge k$, i.e., $s = 1$ or k. In this scenario, $\langle P_1, P_k \rangle$ is the NS-Pair returned by this phase (line 10). Any two partitions P_u and P_v for $u, v = 2$ to $k - 1$ must be output-isomorphic and have the same QPF output as the samples of P_1 and P_k. The Winner group $\overline{T_W}$ can be found accordingly (line 5-8).

Scenario (ii): if $\Theta(\overline{p_i}, P_1.sample) \ne \Theta(\overline{p_i}, P_k.sample)$, we call this the recursive case (line 13-28). SP uses binary search to locate NS-Pair. SP applies Θ to the sample in the middle partition

$P_m.sample$ where $m = \lfloor \frac{1+k}{2} \rfloor$. If $\Theta(\overline{p_i}, P_1.sample) = \Theta(\overline{p_i}, P_m.sample)$, following lemma 5.1, $s \leq 1$ or $s \geq m$. If $s = 1$, all partitions P_i for $i > 1$ should be output-isomorphic to each other. Since $\Theta(\overline{p_i}, P_m.sample) \neq \sigma(P_k.sample)$, s cannot be 1. Thus, s must lie in $[m, k]$. SP recursively repeats the above procedure to find the separating point s in $[m, k]$. The search ends when there are two candidates left, x and $x + 1$. $\langle P_x, P_{x+1} \rangle$ is the NS-Pair returned. The rest of the partitions can be divided into two groups: the first group is P_1 to P_{x-1} and the second group is P_{x+1} to P_k. Any two partitions in the same group must be output-isomorphic and partitions in either one of the groups must be T-homogeneous. By looking at the QPF output of a sample encrypted tuple from the two groups, the Winner group $\overline{T_W}$ can be set (line 25-28).

5.2 QScan: finding exact selection result

Let $\langle P_a, P_b \rangle$ be the NS-Pair SP obtained from *QFilter*. Encrypted tuples in these two partitions are tested using QPF Θ to see which one is actual answer in the selection result. Note that there are two cases in lemma 4.5. Case 1: $\overline{p_i}$ is equivalent to an encrpyted predicate in $\mathbb{P}$; or Case 2: $\overline{p_i}$ is inequivalent to all encrypted predicates in $\mathbb{P}$. The difference between the two cases is that the separating partition in Case 2 is non-homogeneous while all partitions are homogeneous in Case 1. SP can make use of the above difference and adopts an early stop strategy: SP first applies Θ on every encrypted tuples in P_a. If P_a is found to be non-homogeneous, it must be Case 2 and $s = a$. SP does not need to apply Θ on any encrypted tuples in P_b. In the other case where P_a is found to be homogeneous, SP continues to apply Θ on every encrypted tuples in P_b. If P_b is non-homogeneous, it is Case 2 and $s = b$. Otherwise, it is Case 1. In either case, *QScan* finds the set of encrypted tuples $\overline{T_{\text{WNS}}} \subseteq P_a \cup P_b$ that satisfy the predicate, i.e.,

$$\overline{T_{\text{WNS}}} = \{\overline{t_j} \mid \overline{t_j} \in P_a \cup P_b \text{ and } \Theta(\overline{p_i}, \overline{t_j}) = 1\}$$

Algorithm 2 shows the pseudo code of QScan.

```
Algorithm 2: QScan
  Input  : $\overline{T_{\text{NS}}} = \langle P_a, P_b \rangle$ where $a < b$
  Input  : PRKB $\mathcal{I} = POP^C_k : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \ldots \overset{C}{\mapsto} P_k$
  Input  : Encrypted predicate $\overline{p_i}$
  Output : $\overline{T_{\text{WNS}}} = \{\overline{t_j} \mid \overline{t_j} \in P_a \cup P_b \text{ and } \Theta(\overline{p_i}, \overline{t_j}) = 1\}$
1  // First scan $P_a$
2  $P_{a_T} = \{\overline{t_j} \mid \overline{t_j} \in P_a \text{ and } \Theta(\overline{p_i}, \overline{t_j}) = 1\}$; $P_{a_F} = P_a - P_T$;
3  $\overline{T_{\text{WNS}}} = P_{a_T}$ ;
4  if $P_{a_T} = \emptyset$ or $P_{a_F} = \emptyset$ then
5      // $P_a$ is homogeneous, SP scans $P_b$ as well
6      $P_{b_T} = \{\overline{t_j} \mid \overline{t_j} \in P_b \text{ and } \Theta(\overline{p_i}, \overline{t_j}) = 1\}$; $P_{b_F} = P_b - P_T$;
7      $\overline{T_{\text{WNS}}} = \overline{T_{\text{WNS}}} \bigcup P_{b_T}$ ;
8  else
9      // $P_a$ is non-homogeneous, early stop is applied
10     if $label_b = 1$ ;     // $label_b$ is found in QFilter
11     then
12         $\overline{T_{\text{WNS}}} = \overline{T_{\text{WNS}}} \bigcup P_b$ ; // $P_b$ is T-homogeneous
13     end
14 end
15 return $\overline{T_{\text{WNS}}}$
```

The complexity of the entire selection processing is $O(\frac{n}{k} + \lg n)$ where n is number of encrypted tuples in $\overline{T}$ and k is number of partitions in PRKB.

5.3 updatePRKB: update procedure of PRKB

Recall that only inequivalent encrypted predicate can help SP to extend PRKB (see Sec. 4). During the execution of *QScan*, SP already knows whether the new encrypted predicate $\overline{p_i}$ is equivalent to some encrypted predicate in $\mathbb{P}$, which generates SP's current PRKB, POP^C_k. Only when either P_a or P_b is non-homogeneous (found in *QScan*), $\overline{p_i}$ is inequivalent (see Lemma 4.5). In such case, *QScan* (line 2 or line 6) has divided an existing partition P_s into two smaller partitions P_{s_T} and P_{s_F} where $s = a$ (line 2) or b (line 6). Without further QPF uses, SP can easily update POP^C_k to POP^C_{k+1} by replacing P_s with P_{s_T} and P_{s_F} in POP^C_k. The order of P_{s_T} and P_{s_F} in POP^C_{k+1} is determined by whether P_{s-1} is T-homogeneous or F-homogeneous. Like Fig. 4, if P_{s-1} is F-homogeneous, we have $POP^C_{k+1} : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \ldots P_{s-1} \overset{C}{\mapsto} P_{s_F} \overset{C}{\mapsto} P_{s_T} \overset{C}{\mapsto} P_{s+1} \overset{C}{\mapsto} \ldots \overset{C}{\mapsto} P_k$.

updatePRKB is efficient since it does not require additional QPF uses.

6 MULTI-DIMENSIONAL RANGE QUERY

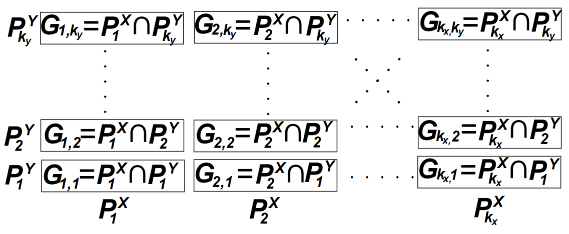
In this section, we will introduce the method to optimize processing of a d-dimensional range query for $d \geq 2$. We address the most common form of d-dimensional range: the query is to retrieve all encrypted tuples with plain values falling into a d-dimensional hyper-rectangle defined by DO. The query can be described in SQL in the following form: SELECT * FROM R WHERE $c_{1a} < C_1 < c_{1b}$ AND $c_{2a} < C_2 < c_{2b}$ AND ... AND $c_{da} < C_d < c_{db}$, where C_i is an attribute in the relational table R and $c_{ia} < c_{ib}$ are query parameters defined by DO for $i = 1$ to d.

Recall that SP is not able to see the plain query parameters and the plain values of encrypted tuples. In EDBMS, the d-dimensional range query is processed as $2d$ comparison predicates (two comparisons for each dimension: $c_{ia} < C_i$ and $C_i < c_{ib}$). DO generates and gives $2d$ encrypted predicates to SP for processing the query. In existing EDBMS, SP has to apply up to $2d$ encrypted predicates on all tuples[5], i.e., the total number of QPF uses can be $2dn$ where n is number of encrypted tuples in $\overline{T}$. A better alternative now is to use the single comparison predicate processing technique in Sec. 5. SP finds out the satisfying tuples of each comparison predicate. Then, SP intersects the set of satisfying tuples to find out the final selection result. Number of QPF uses can be greatly reduced compared to existing processing mechanism of EDBMS. This is our baseline method for processing multi-dimensional range query. In this section, we will describe our solution for multi-dimensional range query that is more efficient than the baseline method. In our discussion below, we will focus on 2D case for easier illustration.

6.1 Visualization of partitions on a grid

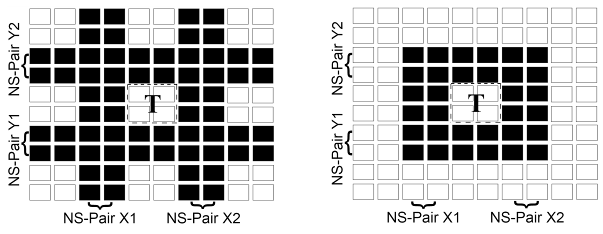
For a 2D range query, two attributes, say X and Y, are concerned. SP has maintained two partial order partitions, say $POP^X_{k_x}$ and $POP^Y_{k_y}$ of the encrypted table $\overline{T}$. $POP^X_{k_x}$ and $POP^Y_{k_y}$ are two partitioning ways of $\overline{T}$, i.e., every encrypted tuple $\overline{t}$ in $\overline{T}$ will be located in one and only one partition of $POP^X_{k_x}$ and one and

[5]EDBMS can stop processing for a tuple when one of the predicates is not satisfied. Actual number of QPF uses varies.



$P^Y_{k_y}$ $G_{1,k_y}=P^X_1 \cap P^Y_{k_y}$ $G_{2,k_y}=P^X_2 \cap P^Y_{k_y}$ $G_{k_x,k_y}=P^X_{k_x} \cap P^Y_{k_y}$
P^Y_2 $G_{1,2}=P^X_1 \cap P^Y_2$ $G_{2,2}=P^X_2 \cap P^Y_2$ $G_{k_x,2}=P^X_{k_x} \cap P^Y_2$
P^Y_1 $G_{1,1}=P^X_1 \cap P^Y_1$ $G_{2,1}=P^X_2 \cap P^Y_1$ $G_{k_x,1}=P^X_{k_x} \cap P^Y_1$
P^X_1 P^X_2 $P^X_{k_x}$

Figure 5: Visualization of the grid in 2D space



(a) Baseline **(b) Our solution**

Figure 6: Illustration of partitions scanned in processing 2D range query by different methods. Encrypted tuples in the central T-region must be part of answer set.

only one partition in $POP^Y_{k_y}$. Let P^X_i be a partition in $POP^X_{k_x}$ for $i = 1$ to k_x and P^Y_j be a partition in $POP^Y_{k_y}$ for $j = 1$ to k_y. Let $G_{i,j} = P^X_i \cap P^Y_j$. We prepare a $k_x \times k_y$ grid. $G_{i,j}$ is represented as a cell at location (i, j) in the grid. *Each encrypted tuple falls into one and only one grid cell $G_{i,j}$ for some i, j.* The grid then represents the visualization of partitions of encrypted tuples in the 2D space. Fig. 5 shows the generated 2D grid. We remark that SP does not know the plain values of boundaries of partitions or the plain values of encrypted tuples in $G_{i,j}$, and the complete grid is not actually computed in query processing.

Now, we use the grid to visualize the processing mechanism of existing methods and identify redundancy in them. A linear scan on all encrypted tuples is equivalent to scanning all the grid cells. A better baseline solution using our single comparison predicate processing method can narrow the search on each dimension to just NS-Pairs. Only the partitions of NS-Pairs require full scan on all encrypted tuples, thus saving a significant amount of QPF uses by SP. For a 2-dimensional range query, there are two comparison predicates on each dimension. Thus, we have 4 NS-Pairs, two on each dimension to be scanned. Fig. 6a shows the illustration of scanning only the NS-Pairs in the grid. Scanning for each NS-Pair is done independently and thus a full column or row in the grid is scanned. In multi-dimensional range query, an encrypted tuple has to satisfy the comparison predicates in all dimensions. In the process of finding NS-Pairs, some of the partitions are also known to be F-homogeneous. For example, in Fig. 6a, there are 2 NS-Pairs on X. Let (P^X_a, P^X_{a+1}) and (P^X_b, P^X_{b+1}) be these two NS-Pairs, where $a < b$. Partitions from P^X_1 to P^X_{a-1} must be F-homogeneous. Thus, $G_{i,j}$ for $i = 1$ to $a-1$ are not necessary to be scanned and SP can safely conclude that all encrypted tuples in these cells will not be part of the result set. Similarly, SP can apply the same pruning in other dimensions. The remaining partitions to be scanned is shown in Fig. 6b. Number of QPF uses of SP is thus reduced.

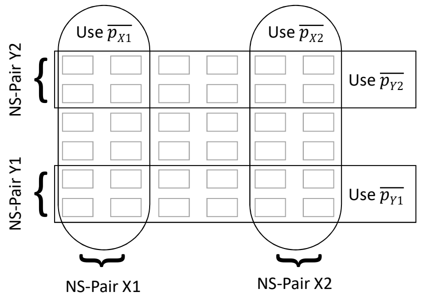


Figure 7: SP tests different encrypted predicates on encrypted tuples in different regions. NS-Pair X1, X2, Y1 and Y2 are generated according to encrypted predicates $\overline{p_{X1}}, \overline{p_{X2}}, \overline{p_{Y1}}$ and $\overline{p_{Y2}}$ respectively

6.2 Systematic scanning procedure for multi-dimensional range query

In this section, we present how SP can systematically and efficiently perform the scan. There are two major issues we need to address to achieve efficiency.

First, note that SP does not test all the encrypted predicates on the encrypted tuples in the area shown in Fig.6b. Recall that there are $2d$ encrypted predicates where d is the number of dimension of the query. Each encrypted predicate gives an NS-Pair. Let $\overline{p}$ be the encrypted predicate giving the NS-Pair (P_a, P_b). Only grid cells that are computed by intersecting P_a or P_b with other partitions require testing by $\overline{p}$. For example, Fig. 7 shows the encrypted predicates needed for different cells in Fig. 6b.

Second, as we presented in Sec. 5.2, an early stop strategy can be used to further reduce the number of QPF uses by SP in comparison predicate processing. SP can use the same strategy in processing multi-dimensional range query as well. Each dimension has two comparison predicates resulting in two NS-Pairs. Let (P_a, P_{a+1}) and (P_b, P_{b+1}) be the two NS-Pairs, where $a < b$. We call P_{a+1} and P_b the *inner NS-partition*; and P_a and P_{b+1} the *outer NS-partition*. If SP first scans the outer NS-partition and finds that it is non-homogeneous, SP can further conclude that the inner NS-partition is T-homogeneous. On the other hand, if SP scans the inner NS-partition first and finds out that it is non-homogeneous, SP can conclude that the outer NS-partition of the same NS-Pair is F-homogeneous. Besides, once an encrypted tuple is found to have QPF ouput 0 for an encrypted predicate, it can never be in the selection result. SP does not need to test other encrypted predicates on this encrypted tuple.

In summary, the procedure to process multi-dimensional range query is done by the following steps:
(1) Use *QFilter* to find the NS-Pair for each encrypted predicate.
(2) Compute the required grid cells, e.g., in Fig. 6b, by intersecting the partitions in different $POP^{C_i}_{k_{C_i}}$ for different attributes C_i.
(3) Test encrypted predicates using QPF on encrypted tuples in different regions of the grid, e.g., according to Fig. 7, and apply early stop strategy when possible. (Details are described above in this section.)
(4) Return as selection result those encrypted tuples with QPF output 1 for all encrypted predicates in step (3) and the encrypted tuples in the central T-region, e.g., in Fig. 6b.

The entire process takes $O(d(\frac{n\alpha^{d-1}}{k} + \lg k))$ where n is number of encrypted tuples in $\overline{T}$, k is number of partitions in PRKB, d is number of dimensions of the query and α is the selectivity on each dimension. Assume α remains the same, the query cost decreases as d increases. Our selection processing technique for multi-dimensional range query is scalable to number of dimensions.

7 DATABASE UPDATE HANDLING

The selection processing techniques we presented in Sec. 5 and Sec. 6 are designed for a static database where the contents of encrypted tuples do not change. In this section, we discuss how we can support update operations in a database.

There are 3 kinds of update operations in SQL: (1) INSERT statements; (2) DELETE statements; and (3) UPDATE statements. UPDATE statements can be considered as insertion of a new tuple after deletion of an existing tuple. We just need to cater for insertion and deletion.

7.1 Insertion Handling

When there is a new encrypted tuple to be inserted to EDBMS, SP needs to update PRKB, POP_k^C, to assign the new encrypted tuple to the correct partition. To facilitate the update, SP needs to remember the set of past $k-1$ encrypted predicates $\mathbb{P}$ that generates POP_k^C. SP can order the encrypted predicates in $\mathbb{P}$ according to POP_k^C: $P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} \ldots \overset{C}{\mapsto} P_k$ because the encrypted predicates are the separators that form the partitions. Let $\overline{p_x}$ be the encrypted predicate s.t. partitions P_i for $i = 1$ to x are output-isomorphic and partitions P_j for $j = x + 1$ to k are output-isomorphic but P_i and P_j are not output-isomorphic, e.g., the encrypted predicate in Fig. 3 is refereed as $\overline{p_5}$. A binary search can be used by SP to find out the partition the new encrypted tuple belongs to: SP first uses the encrypted predicate in the middle $\overline{p_m}$, where $m = \lfloor \frac{k}{2} \rfloor$, to determine whether the encrypted tuple belongs to the first half or the second half of POP_k^C; then recursively reduces the list by half until only one partition remains and this partition is where the new encrypted tuple belongs to.

It takes $O(\lg k)$ time to update PRKB for the new encrypted tuple. Let β be number of attributes with indexing. The total update cost is then $O(\beta \lg k)$.

7.2 Deletion Handling

Deletion handling is easy as SP simply removes the corresponding encrypted tuple from the corresponding partitions. When there is no tuple remained in a partition of POP_k^C, the partition is removed from POP_k^C, i.e., the knowledge of partial order partitions becomes POP_{k-1}^C.

8 EMPIRICAL STUDIES

There are two purposes in our experiments. First, as we mentioned in Sec. 3.3, SP can observe partial order information in existing EDBMS (even without implementing PRKB). We want to evaluate whether existing EDBMS model is acceptable in practice. Second, we empirically evaluate the performance of our indexing method, PRKB.

The experiment settings and its result for the first purpose is presented in Sec. 8.1. Information of experiments for the second purpose is presented in Sec. 8.2.

		Number of queries				
Victims	Size	250	1K	10K	100K	1M
Hospital	2,426,516	0.007	0.020	0.115	0.605	2.846
Labor	6,156,470	0.042	0.117	0.484	1.673	5.807
Latitude	1,122,932	0.008	0.025	0.212	1.650	11.167
Longitude	1,122,932	0.011	0.038	0.331	2.440	13.592

Table 2: Recovered portion of ordering information (RPOI) (%) on real datasets varying number of queries observed by attacker

8.1 Experiment on security of EDBMS model revealing selection result

As discussed in Sec. 3.3, we want to see how much partial order information can be derived by SP/attacker in existing EDBMS model in practice. To quantify how close the recovered partial order is to total order, we define *recovered portion of ordering information* (RPOI) as $\frac{\text{Recovered partial order length}}{\text{Total order length}}$. (Partial order length is the size of the longest chain, e.g., the length of total order is n for a dataset of n distinct numbers.)

We follow the scenario used in [24] to perform the experiment. SP is able to receive certain number of queries, randomly generated by DO. Unlike [24], SP in our case receives limited number of queries only and we pick attributes with large domain sizes as victims. Each query has has one encrypted predicate. We vary number of queries from 250 to 1M and measure RPOI in these cases. We tested on 4 victim attritubes from 3 different real datasets:

(1) **Hospital Charges**: Hospital Inpatient Discharges 2013 dataset[6]
(2) **Labor Salary**: US Labor Statistic 2017[7]
(3) **Latitude**: US Buildings dataset[8]
(4) **Longitude**: US Buildings dataset

The result is presented in Table 2.

The result shows that the partial order information observed by SP is still far from complete as the total order. RPOI increases at decreasing speed as SP observes more queries. It is because it gets harder for SP to observe a useful query to enhance the partial order knowledge. According to Quantcast[9], it could take weeks for a top-1000 website to get a million of traffic, which is still far away from recovering the total order in an attack attempt. We consider our current model of EDBMS revealing selection result as practically secure for large domain data. In contrast, if OPE is used, e.g., in CryptDB [29], RPOI is 100% even SP has not yet processed any query.

8.2 Performance evaluation of PRKB

8.2.1 Algorithm implementation. We separately evaluate single-dimensional (SD) query and multi-dimensional (MD) range query. Note that there are different processing techniques in using PRKB. To differentiate them, we use (i) 'PRKB(SD)' to denote the processing method for single-dimensional query (Sec. 5), (ii) 'PRKB(SD+)' to denote the naive extension of PRKB(SD) for multi-dimensional range query (see Sec. 6), and (iii) 'PRKB(MD)' to denote the algorithm designed for multi-dimensional range query (see Sec. 6.2).

[6]https://health.data.ny.gov/
[7]https://catalog.data.gov/
[8]http://www.geonames.org/
[9]https://www.quantcast.com/top-sites

As a competitor, we implemented the indexing method 'Logarithmic-SRC-i' in [12]. Note that Logarithmic-SRC-i may return false positives to DO. DO needs to decrypt them to confirm whether they are actual answer in the selection result. This may require a significant amount of DO's involvement to process the query. In our implementation, we deployed a trusted machine (TM), like Cipherbase [2], to perform this confirmation process on behalf of DO. In PRKB, we use the same confirmation process as QPF, i.e., SP sends the encrypted data to TM; TM decrypts and returns the QPF output of the encrypted tuple. Besides, Logarithmic-SRC-i is an encrypted index computed and maintained by DO. This is also done by TM in our implementation. In our experiment, both TM and SP equip with a machine with the same power. We also compare with the case where no indexing is used, denoted as Baseline.

PRKB replies on past queries to operate effectively. In all experiments, number of queries is limited to small values (at most 600) so as to show that PRKB is effective even without a lot of past result knowledge.

All algorithms were implemented in C/C++. All machines in the experiment equip with 2GHz CPU and 4GB RAM running Linux platform.

8.2.2 Datasets and Tests. We performed our experiments on both synthetic and real datasets. We performed most of the experiments on synthetic datasets to evaluate the performance varying different parameters, e.g., number of tuples and selectivity. In the synthetic datasets, the data domain of all attributes is set to be integers in [1, 30M]. The plain value on each attribute of each tuple is randomly generated[10].

We simulated a use case on a real dataset. The US buildings dataset contains 1,122,932 records about information of buildings in US, including location (latitude and longitude). A tourist (user) is interested to know what buildings are around the location he will visit. The user issues a range query to retrieve all buildings within a $1km \times 1km$ region in the dataset, i.e., it is a 2D range query.

We measure the average number of QPF uses[11] (# QPF use) and average execution time out of 20 runs for each experiment.

8.2.3 Experiment on Building PRKB. This experiment simulates SP building PRKB from scratch on a synthetic dataset with 10M tuples. We assume SP receives 600 distinct queries, each with one comparison predicate, and we monitor the performance of query processing cost. For reference, the performance of Baseline and Logarithmic-SRC-i is also shown. Fig. 8 shows the result of query cost and Table 3 shows the space consumption of PRKB.

We make the following observations from the result:

(1) In the beginning, PRKB has no knowledge. Query processing is as slow as Baseline. However, when SP receives queries, the cost drops fast. At 50-th query, the cost has already dropped by an order of magnitude and PRKB has almost the same performance as Logarithmic-SRC-i. At 600-th query, the query time of PRKB is one order of magnitude smaller than Logarithmic-SRC-i. It shows that PRKB is practical, reducing the query processing cost significantly with a small amount of past result knowledge.

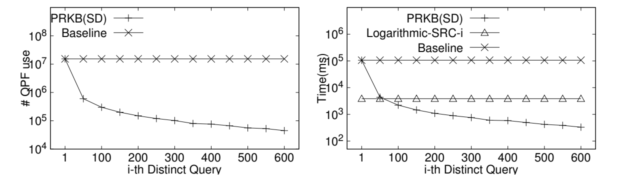


Figure 8: Performance of Query with growing PRKB on 10M tuples (1% Selectivity)

Method	Dataset size (in millions)					
	10	12	14	16	18	20
PRKB-250	38.2	45.8	53.4	61.0	68.7	76.3
PRKB-600	38.2	45.9	53.5	61.1	68.8	76.4
Logarithmic-SRC-i	3589	4050	4493	4918	6356	6758

Table 3: Storage size of the index (in MB)

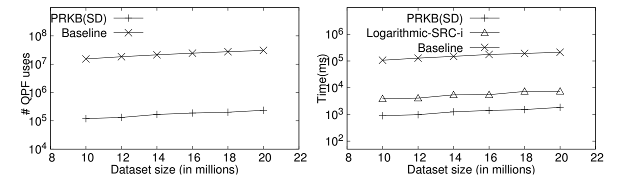


Figure 9: Performance on single-dimensional query varying dataset size (1% selectivity)

(2) PRKB occupies a small space, as PRKB is simply partition information of encrypted tuples. There is a slight increase in space consumption (from 76.3MB to 76.4MB for 20M dataset) of PRKB. It is because SP needs to keep more encrypted predicates to handle database update (see Sec. 7). The increase in space consumption is negligible. Logarithmic-SRC-i requires much more space due to its more complex index structure.

(3) The query processing cost is consistent with number of QPF uses. This shows that QPF computation is the dominant cost in EDBMS. Reducing number of QPF uses can help to reduce the overall query cost.

8.2.4 Experiment for Single-dimensional Query. We tested the performance of algorithms in handling a single-dimensional query under different settings. The query is in the form of "SELECT * FROM Dataset WHERE $lb < X < ub$". X is an attribute on synthetic dataset. lb and ub are two parameters generated randomly according to selectivity. We use a static PRKB with 250 partitions for the experiment. There are 2 parameters in the experiment: (i) dataset size, varying from 10M to 20M tuples; (ii) selectivity, varying from 1% to 10%.

Fig. 9 and Fig. 10 show the results of experiments in varying dataset size and selectivity respectively.

We make the following observations from the results.

(1) All algorithms scale well with increasing number of tuples. Cost reduction of PRKB(SD) over Baseline and Logarithmic-SRC-i is consistent, at about two orders of magnitude and a factor of 4, respectively.

(2) PRKB(SD) shows a steady performance no matter how selectivity increases. It is because PRKB simply requires SP to examine two NS-Pairs defining the boundary of the

[10]We have tested on data generated with different distributions, including uniform, normal, correlated and anti-correlated. The results are similar and so we just present the results for uniform distribution in this paper.

[11]Since majority of actions in Logarithmic-SRC-i are related to its index structure, we do not show # QPF use for Logarithmic-SRC-i.

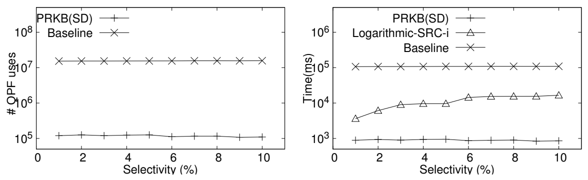


Figure 10: Performance on single-dimensional query varying selectivity (dataset of 10M tuples)

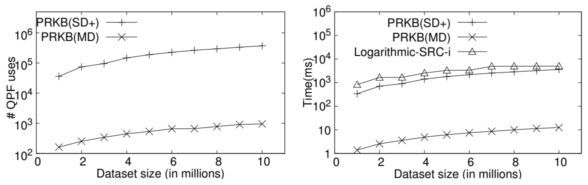


Figure 11: Performance on multi-dimensional query varying dataset size (Dimensionality of 3, 2% selectivity per dimension)

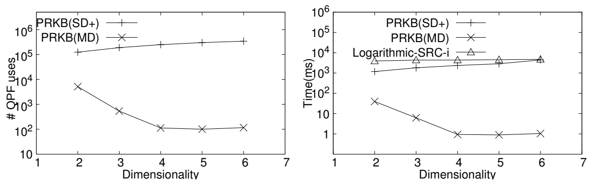


Figure 12: Performance on multi-dimensional query varying dimensionality (Dataset of 5M tuples, 2% selectivity per dimension)

answer set. All encrypted tuples in partitions between the two NS-Pairs can be returned as answer without applying QPF on them. The cost of PRKB is independent to size of answer set.

8.2.5 Experiment for Multi-dimensional Range Query. In this experiment, we study the difference in performance between PRKB(SD+), PRKB(MD) and Logarithmic-SRC-i under different settings to validate the importance of our optimization method for handling multi-dimensional range query. The range query tested is in the form of "SELECT * FROM Dataset WHERE $lb_1 < X_1 < ub_1$ AND ... AND $lb_d < X_d < ub_d$". X_i is an attribute in the synthetic dataset. lb_i and ub_i are generated randomly according to selectivity (per dimension), which is set to be 2%. Both algorithms use a static PRKB with 250 partitions. There are 2 parameters in the experiment: (i) dimensionality d, varying from 2 to 6, and (ii) dataset size, varying from 1M to 10M tuples.

Figure 11 and 12 show the results. Improvement of PRKB(MD) over PRKB(SD+) and Logarithmic-SRC-i is consistent with increasing dataset size. The cost of PRKB(SD+) increases as number of dimensions increases because PRKB(SD+) processes each dimension separately. However, number of results actually decreases with more comparison predicates. Logarithmic-SRC-i sent a set of hashed values for keyword search for each dimensions. The cost of Logarithmic-SRC-i is getting closer to PRKB(SD+) in Figure 12. PRKB(MD), on the other hand, can make use of the fact that more comparison predicates filter more candidate tuples. Thus, the cost of PRKB(MD) decreases with increasing number of dimensions. PRKB(MD) can perform well even for higher dimensional range queries.

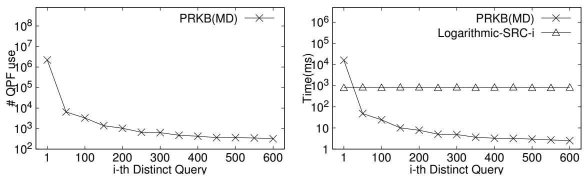


Figure 13: Performance of Query with growing PRKB on US buildings dataset (2% selectivity)

Method	Batch				
	1	2	3	4	5
PRKB	32,356	32,104	32,117	32,167	32,168
Logarithmic-SRC-i	2,936	2,967	2,967	2,935	2,937

Table 4: Average throughput (Tuples / Second) of inserting 5 batches (each with 2M tuples) of data to PRKB with 10M tuples

8.2.6 Experiment on Real Dataset. We tested PRKB and Logarithmic-SRC-i in a simulated use case (described in Sec. 8.2.2) on real dataset to validate its practicality.

In this dataset, the space consumed by PRKB is less than 1% of the size of encrypted dataset ($\frac{8.81\text{MB}}{1.04\text{GB}}$) while Logarithmic-SRC-i consumed more than 43% space ($\frac{441.346\text{MB}}{1.04\text{GB}}$).

Similar to the experiments on synthetic datasets, the query processing time is high in the beginning. Initially, the query time of Logarithmic-SRC-i is smaller than that of PRKB. After answering 50 queries, the query time of PRKB is already below 100ms and performs better than Logarithmic-SRC-i. After answering 600 queries, the query time of PRKB is further reduced to 9ms. In contrast, if EBDMS does not use any index, it takes 15.9s to process a query, which is impractical in reality. Besides, if DO wants to avoid the poor performance of EBBMS using PRKB in the beginning, DO can arbitrarily generates queries (as few as 50 queries in this case) to help SP to build an initiate PRKB.

8.2.7 Experiment for Handling Database Update. In this experiment, we evaluate the cost of SP updating PRKB in handling database update. Since deletion is simple, we only show the results for insertion here. PRKB has 250 partitions. The experiment is done on a synthetic dataset with 10M tuples. We inserted 5 batches, each with 2M new tuples, to the database, i.e., the database contains 20M tuples in the end. We measure the average throughput (number of tuples inserted per second) achieved by PRKB in each batch. For comparison, we measure the throughput of Logarithmic-SRC-i in the same setting. Table 4 shows the result.

The throughput of PRKB remains almost the same. The observation can be explained in our analysis in Sec. 7.1, as the update cost is independent to database size. SP can easily bear the update cost to maintain PRKB for optimizing query processing.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel indexing method - past result knowledge base (PRKB) for EDBMS. Unlike traditional indexing problem for encrypted data, PRKB is built solely by SP based on results of past queries. None of existing indexing methods work without DO's involvement or customized encryption method.

We showed that PRKB is effective in reducing the processing cost of new queries. Our experiments showed that PRKB consistently outperforms a state-of-the-art competitor in [12] and PRKB achieves a speed-up of at least an order of magnitude compared to EDBMS without implementing PRKB. Since SP is just making use the information that is already available to SP to build PRKB, security of PRKB is ensured. In the future, we plan to extend PRKB to incorporate different query result and to support more query types. The partial order information in PRKB can also be used in optimizing queries like Min, Max or Skyline queries.

ACKNOWLEDGEMENTS

The research is supported by FDS grant (UGC/FDS14/E05/14).

A SUPPORTING BETWEEN OPERATOR

Some methods, e.g., [12], support specifically BETWEEN operator. BETWEEN operator returns the overall result of whether the encrypted tuple falls into the range instead of two results of two comparisons, i.e., SP observes less information for BETWEEN. In fact, as we will show below, BETWEEN is equivalent to two separate comparisons, w.r.t. building PRKB, in most cases.

Say SP has $POP_5^C : P_1 \overset{C}{\mapsto} P_2 \overset{C}{\mapsto} P_3 \overset{C}{\mapsto} P_4 \overset{C}{\mapsto} P_5$. Let $\overline{p}$ be an encrypted predicate that computes 'X BETWEEN a and b'. We can derive a similar observation like lemma 4.5 that, in general, Θ returns 1 for encrypted tuples in partitions in the middle and 0 for encrypted tuples in partitions in the two ends. Say (i) P_3 is T-homogeneous, (ii) P_1, P_5 are F-homogenous, and (iii) P_2 and P_4 are non-homogeneous. Each of P_2 and P_4 is divided into two partitions and we have P_{2T}, P_{2F}, P_{4T}, and P_{4F} where P_{iT} (P_{iF} resp.) denotes the set of tuples in P_i that get 1 (0 resp.) from Θ. SP obtains $POP_7^C : P_1 \overset{C}{\mapsto} P_{2F} \overset{C}{\mapsto} P_{2T} \overset{C}{\mapsto} P_3 \overset{C}{\mapsto} P_{4T} \overset{C}{\mapsto} P_{4F} \overset{C}{\mapsto} P_5$. SP obtains the same POP_7^C as if SP obtains two encrypted predicates for '$X \geq a$' and '$X \leq b$'. The BETWEEN predicate reveals the same partial order information to SP as two separate comparison predicates in this scenario.

Only when the range in the BETWEEN operator is very small such that only some encrypted tuples in one partition get 1 from Θ, SP cannot determine the order information of other tuples in this partition.

Computing a BETWEEN operator using PRKB is similar to comparison handling. SP looks for two separating points using the samples of partitions, like *QFilter*. When a sample encrypted tuple with QPF output 1 is found, two binary searches are performed to find two NS-pairs, each containing a separating point on the two ends of the range of BETWEEN predicate. The process after that is the same as comparison handling. However, when no sample with QPF output 1 is found, SP cannot conclude whether other tuples will return 1 or 0 from QPF due to the existence of the above exceptional case. SP needs to draw more samples from partitions. The worst case is that SP finds that there is no satisfying tuple after scanning all encrypted tuples.

REFERENCES

[1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order-Preserving Encryption for Numeric Data. In *SIGMOD*.

[2] Arvind Arasu, Spyros Blanas, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, Prasang Upadhyaya, and Ramarathnam Venkatesan. 2013. Secure database-as-a-service with Cipherbase. In *SIGMOD*.

[3] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramaratnam Venkatesan. 2013. Orthogonal Security With Cipherbase. In *CIDR*.

[4] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. 2015. Transaction processing on confidential data using cipherbase. In *ICDE*.

[5] Sumeet Bajaj and Radu Sion. 2011. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *SIGMOD*.

[6] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *CRYPTO*.

[7] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *NDSS*.

[8] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO*.

[9] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *JACM* 45, 6 (1998).

[10] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*.

[11] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. 2003. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *CCS*.

[12] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. 2016. Practical Private Range Search Revisited. In *SIGMOD*.

[13] Ioannis Demertzis and Charalampos Papamanthou. 2017. Fast Searchable Encryption with Optimal Locality. In *SIGMOD*.

[14] Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. 2015. Shuffle Index: Efficient and Private Access to Outsourced Data. *TOS* 11, 4 (2015).

[15] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play any Mental Game. In *STOC*.

[16] Oded Goldreich and Rafail Ostrovsky. 1996. Software Protection and Simulation on Oblivious RAMs. *J. ACM* 43, 3 (1996).

[17] Chunsheng Gu and Jixing Gu. 2014. Known-plaintext attack on secure kNN computation on encrypted databases. *Security and Communication Networks* 7, 12 (2014).

[18] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*.

[19] Zhian He, Wai Kit Wong, Ben Kao, David Wai-Lok Cheung, Rongbin Li, Siu-Ming Yiu, and Eric Lo. 2015. SDB: A Secure Query Processing System with Data Interoperability. *PVLDB* 8, 12 (2015).

[20] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure Multidimensional Range Queries over Outsourced Data. *The VLDB Journal* (2012).

[21] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. 2004. A Privacy-Preserving Index for Range Queries. In *VLDB*.

[22] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2014. Inference attack against encrypted range queries on outsourced databases. In *CODASPY*.

[23] Panagiotis Karras, Artyom Nikitin, Muhammad Saad, Rudrika Bhatt, Denis Antyukhov, and Stratos Idreos. 2016. Adaptive Indexing over Encrypted Numeric Data. In *SIGMOD*.

[24] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *CCS*.

[25] Rui Li, Alex X. Liu, Ann L. Wang, and Bezawada Bruhadeshwar. 2014. Fast Range Query Processing with Strong Privacy Protection for Cloud Computing. *PVLDB* 7, 14 (2014).

[26] Sha Ma, Bo Yang, Kangshun Li, and Feng Xia. 2011. A Privacy-Preserving Join on Outsourced Database. In *ISC*.

[27] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *SIGSAC*.

[28] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich. 2013. An Ideal-Security Protocol for Order-Preserving Encoding. In *SP*.

[29] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *SOSP*.

[30] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. 2007. Multi-Dimensional Range Query over Encrypted Data. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP '07)*.

[31] Erez Shmueli, Ronen Waisenberg, Yuval Elovici, and Ehud Gudes. 2005. Designing Secure Indexes for Encrypted Databases. In *DBSec*.

[32] E. Stefanov and E. Shi. 2013. ObliviStore: High Performance Oblivious Cloud Storage. In *SP*.

[33] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *PVLDB* 6, 5 (2013).

[34] Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure kNN computation on encrypted databases. In *SIGMOD*.

[35] Wai Kit Wong, Ben Kao, David Wai-Lok Cheung, Rongbin Li, and Siu-Ming Yiu. 2014. Secure query processing with data interoperability in a cloud database environment. In *SIGMOD*.